

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Саратовский государственный технический университет
имени Гагарина Ю.А.»

Филиал федерального государственного бюджетного образовательного
учреждения высшего образования
«Саратовский государственный технический университет
имени Гагарина Ю.А.» в г. Петровске



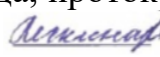
МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ

по дисциплине

МДК. 01.01 «Разработка программных модулей»

направление подготовки

09.02.07 «Информационные системы и программирование»

Методические указания рассмотрены
на заседании предметной (цикловой)
комиссии общепрофессиональных
дисциплин, профессиональных модулей
специальностей технического профиля
«14» июня 2023 года, протокол №12
Председатель ПЦК /Лескина Т.А./

Петровск 2023

Пояснительная записка

Методические указания по выполнению практических работ подготовлены на основе рабочей программы учебной дисциплины МДК. 01.01 «Разработка программных модулей», разработанной на основе ФГОС СПО по специальности 09.02.07 «Информационные системы и программирование» и соответствующих общих (ОК) и профессиональных (ПК) компетенций:

ОК 01. Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02. Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях.

ОК 04. Эффективно взаимодействовать и работать в коллективе и команде.

ОК 05. Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 06. Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных общечеловеческих ценностей, в том числе с учетом гармонизации межнациональных и межрелигиозных отношений, применять стандарты антикоррупционного поведения

ОК 07. Содействовать сохранению окружающей среды, ресурсосбережению, применять знания об изменении климата, принципы бережливого производства, эффективно действовать в чрезвычайных ситуациях

ОК 08. Использовать средства физической культуры для сохранения и укрепления здоровья в процессе профессиональной деятельности и поддержания необходимого уровня физической подготовленности

ОК 09. Пользоваться профессиональной документацией на государственном и иностранном языках.

ОК 10. Использовать знания по финансовой грамотности, планировать предпринимательскую деятельность в профессиональной сфере

ПК 1.1. Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием

ПК 1.2. Разрабатывать программные модули в соответствии с техническим заданием.

При выполнении практических работ студент должен *знать*:

- основные этапы разработки программного обеспечения;
- основные принципы технологии структурного и объектно-ориентированного программирования;
- способы оптимизации и приемы рефакторинга;
- основные принципы отладки и тестирования программных продуктов

При выполнении практических работ студент должен *уметь*:

- осуществлять разработку кода программного модуля на языках низкого и высокого уровней;
- создавать программу по разработанному алгоритму как отдельный модуль;
- выполнять отладку и тестирование программы на уровне модуля; осуществлять разработку кода программного модуля на современных языках программирования;
- уметь выполнять оптимизацию и рефакторинг программного кода;
- оформлять документацию на программные средства

Содержание практических занятий определено рабочей программой и тематическим планированием, соответствует теоретическому материалу изучаемых разделов учебной дисциплины.

Объем практических занятий по дисциплине определяется учебным планом по данной специальности.

Продолжительность практического занятия – 2 академических часа. Перед проведением практического занятия преподавателем организуется инструктаж, а по его окончании – обсуждение итогов.

Комплект методических указаний по выполнению практических работ по дисциплине МДК.01.01 «Разработка программных модулей» содержит 48 практических занятий.

**Перечень практических работ
по дисциплине МДК.01.01 «Разработка программных модулей»**

ПРАКТИЧЕСКАЯ РАБОТА № 1

Тема: Изучение методов оценки алгоритмов

ПРАКТИЧЕСКАЯ РАБОТА № 2

Тема: Оценка сложности алгоритмов сортировки

ПРАКТИЧЕСКАЯ РАБОТА № 3

Тема: Оценка сложности алгоритмов поиска

ПРАКТИЧЕСКАЯ РАБОТА № 4

Тема: Оценка сложности рекурсивных алгоритмов

ПРАКТИЧЕСКАЯ РАБОТА № 5

Тема: Оценка сложности эвристических алгоритмов

ПРАКТИЧЕСКАЯ РАБОТА № 6

Тема: Работа с классами

ПРАКТИЧЕСКАЯ РАБОТА № 7

Тема: Работа с классами

ПРАКТИЧЕСКАЯ РАБОТА № 8

Тема: Работа с объектами через интерфейсы

ПРАКТИЧЕСКАЯ РАБОТА № 9

Тема: Использование стандартных интерфейсов

ПРАКТИЧЕСКАЯ РАБОТА № 10

Тема: Работа с типом данных структура

ПРАКТИЧЕСКАЯ РАБОТА № 11

Тема: Работа с типом данных структура

ПРАКТИЧЕСКАЯ РАБОТА № 12

Тема: Коллекции. Параметризованные классы

ПРАКТИЧЕСКАЯ РАБОТА № 13

Тема: Использование регулярных выражений

ПРАКТИЧЕСКАЯ РАБОТА № 14

Тема: Операции со списками

ПРАКТИЧЕСКАЯ РАБОТА № 15

Тема: Операции со списками

ПРАКТИЧЕСКАЯ РАБОТА № 16

Тема: Использование основных шаблонов

ПРАКТИЧЕСКАЯ РАБОТА № 17

Тема: Использование порождающих шаблонов

ПРАКТИЧЕСКАЯ РАБОТА № 18

Тема: Использование структурных шаблонов

ПРАКТИЧЕСКАЯ РАБОТА № 19

Тема: Использование поведенческих шаблонов

ПРАКТИЧЕСКАЯ РАБОТА № 20

Тема: Разработка приложения с использованием текстовых компонентов

ПРАКТИЧЕСКАЯ РАБОТА № 21

Тема: Разработка приложения с использованием текстовых компонентов

ПРАКТИЧЕСКАЯ РАБОТА № 22

Тема: Разработка приложения с использованием текстовых компонентов

ПРАКТИЧЕСКАЯ РАБОТА № 23

Тема: Разработка приложения с несколькими формами

ПРАКТИЧЕСКАЯ РАБОТА № 24

Тема: Разработка приложения с несколькими формами

ПРАКТИЧЕСКАЯ РАБОТА № 25

Тема: Разработка приложения с несколькими формами

ПРАКТИЧЕСКАЯ РАБОТА № 26

Тема: Разработка приложения с не визуальными компонентами

ПРАКТИЧЕСКАЯ РАБОТА № 27

Тема: Разработка приложения с не визуальными компонентами

ПРАКТИЧЕСКАЯ РАБОТА № 28

Тема: Разработка приложения с не визуальными компонентами

ПРАКТИЧЕСКАЯ РАБОТА № 29

Тема: Разработка приложения с графикой и анимацией

ПРАКТИЧЕСКАЯ РАБОТА № 30

Тема: Разработка приложения с графикой и анимацией

ПРАКТИЧЕСКАЯ РАБОТА № 31

Тема: Разработка приложения с графикой и анимацией

ПРАКТИЧЕСКАЯ РАБОТА № 32

Тема: Разработка игрового приложения

ПРАКТИЧЕСКАЯ РАБОТА № 33

Тема: Разработка игрового приложения

ПРАКТИЧЕСКАЯ РАБОТА № 34

Тема: Разработка игрового приложения

ПРАКТИЧЕСКАЯ РАБОТА № 35

Тема: Оптимизация и рефакторинг кода

ПРАКТИЧЕСКАЯ РАБОТА № 36

Тема: Оптимизация и рефакторинг кода

ПРАКТИЧЕСКАЯ РАБОТА № 37

Тема: Оптимизация и рефакторинг кода

ПРАКТИЧЕСКАЯ РАБОТА № 38

Тема: Разработка интерфейса пользователя

ПРАКТИЧЕСКАЯ РАБОТА № 39

Тема: Разработка интерфейса пользователя

ПРАКТИЧЕСКАЯ РАБОТА № 40

Тема: Разработка интерфейса пользователя

ПРАКТИЧЕСКАЯ РАБОТА № 41

Тема: Разработка интерфейса пользователя

ПРАКТИЧЕСКАЯ РАБОТА № 42

Тема: Разработка интерфейса пользователя

ПРАКТИЧЕСКАЯ РАБОТА № 43

Тема: Создание приложения с БД

ПРАКТИЧЕСКАЯ РАБОТА № 44

Тема: Создание приложения с БД
ПРАКТИЧЕСКАЯ РАБОТА № 45

Тема: Создание приложения с БД
ПРАКТИЧЕСКАЯ РАБОТА № 46

Тема: Создание приложения с БД
ПРАКТИЧЕСКАЯ РАБОТА № 47

Тема: Создание запросов к БД
ПРАКТИЧЕСКАЯ РАБОТА № 48

Тема: Создание запросов к БД

ИНСТРУКЦИИ ДЛЯ ОБУЧАЮЩИХСЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ

Прежде чем приступить к выполнению заданий, внимательно прочитайте данные рекомендации.

В ходе выполнения практических работ студент должен:

- выполнять требования по охране труда
- соблюдать инструкцию по правилам и мерам безопасности в кабинете информационных технологий
- строго выполнять весь объем работы, указанный в задании
- соблюдать требования эксплуатации компьютерной техники (правила включения и выключения)
- предоставить отчет о проделанной работе по окончании выполненной работы, который должен содержать:

1. Название работы.
2. Цель работы.
3. Задание и его решение.
4. Вывод о проделанной работе.

Текст отчета по практической работе должен быть набран на компьютере шрифтом Times New Roman размером 14 пт. (при оформлении текста используется текстовый редактор MS Word). Шрифт, используемый в иллюстративном материале (таблицы и рисунки), рекомендуется уменьшить до 12 пт. Межстрочный интервал в основном тексте - полуторный. В иллюстративном материале межстрочный интервал рекомендуется сделать одинарным. Поля страницы должны быть: левое поле - 30 мм; правое поле – 15 мм; верхнее и нижнее поле - 20 мм.

Каждый абзац должен начинаться с красной строки. Отступ абзаца – 1,25 см от левой границы текста.

Студент должен выполнить практическую работу самостоятельно (или в группе, если это предусмотрено заданием). Практическая работа выполняется согласно заданию и методическим рекомендациям. Результат работы представляется преподавателю в виде файла (файлов) в личном каталоге, защищается обучающимися.

По ходу выполнения работы при возникновении вопросов обучающийся может получить консультацию у преподавателя или самостоятельно воспользоваться лекционным материалом, рекомендуемой литературой.

1. Составление программы на языке программирования

Правила оформления кода:

1. Используйте разумные имена для переменных и функций

Программа должна быть хорошо понятна человеку при чтении. Если при чтении программы приходится понимать назначение переменных и функций по тому, как они используются, то читать код становится гораздо сложнее. Неудачно выбранные имена могут привести к тому, что смысл программы может быть неправильно понят. Выбирайте такие имена, которые бы объясняли смысл переменных и функций, тогда код станет гораздо понятнее, и не потребуется писать множество комментариев.

При написании составных слов, например в именах переменных, пишите их слитно без пробелов, при этом каждое новое слово пишется с большой буквы.

2. Не дублируйте код. Если в программе есть одинаковые выражения или фрагмента кода, вынесите этот код в отдельную функцию. Верным признаком необходимости создания новой функции является желание скопировать фрагмент кода из одного места программы в другое. В таком случае сразу перенесите этот фрагмент в отдельную функцию.

Если фрагменты кода похожи, но не идентичны, подумайте, не получится ли и их вынести в одну функцию, возможно добавив параметры или условия.

3. Не используйте «магические константы». Использование неименованных «магических» констант в коде нежелательно:

- при чтении кода может быть не понятно, что это за число, и почему оно именно такое;
- чаще всего одно и то же число потребуется написать в нескольких местах кода. Если его придётся изменять, можно пропустить одно из использований, что приведёт к ошибке.

Если в коде нужно использовать константу, дайте ей имя, используя `const`.

4. Расставляйте пробелы вокруг бинарных операторов. Это улучшает читаемость формул.

5. Всегда выделяйте блоки условных операторов и циклов скобками. В любой блок условия или цикла может захотеться добавить новое выражение. При этом можно забыть добавить скобки. Лучше сразу добавить скобки, чтобы потом не было с этим проблем.

6. Расставляйте скобки одинаково. Выберите и используйте для себя один из стилей расстановки скобок. Это улучшает читаемость структуры программы.

7. Не делайте строки слишком длинными. Строка программы должна помещаться на экране. Обычно рекомендуют ограничить максимальную ширину строки в 80 символов. Если определение или вызов функции получается слишком широким поместите по одному параметру на каждой строке. Длинные математические выражения разбивайте на несколько строк, разбивая по границам логических блоков выражения.

8. Объявляйте переменные непосредственно перед использованием. Обязательно указывайте начальное значение для переменных. Значение неинициализированной переменной может быть любым. Использование (чтение) такого значения приведёт к недетерминированной работе программы, а в некоторых случаях является неопределённым поведением. Чтобы избежать проблем всегда инициализируйте переменные прямо в момент их создания.

9. Единый стиль оформления кода во всем проекте;

10. Визуальное выделение наиболее значимых частей — используя *вертикальное форматирование*, мы выделяем объявление переменных, цикл заполнения массива случайными числами и цикл обработки по формуле. Если ваша функция выполняет несколько действий — то разумно разделить соответствующие блоки кода пустыми строками.

ПРАКТИЧЕСКАЯ РАБОТА № 1

Тема: Изучение методов оценки алгоритмов

Цель работы: Изучение методов оценки алгоритмов и программ и определение временной и емкостной сложности типовых алгоритмов и программ.

Оборудование: ПК, интернет, программное обеспечение – MS Word, среда программирования Visual Studio, инструкции по выполнению работы

Справочный материал:

Алгоритм – это точное предписание о выполнении в определенном порядке некоторых операций, приводящих к решению всех задач данного класса. Важнейшей характеристикой алгоритма и соответствующей ему программы является их сложность, которая может оцениваться: а) временем решения задачи (трудоемкостью алгоритма); б) требуемой емкостью памяти. В общем случае сложность алгоритма можно оценить по порядку величины. Этот метод применим как к временной, так и к емкостной сложности.

О-сложность алгоритмов.

Понятие О-сложности алгоритмов введено для того, чтобы измерять скорость роста функции в зависимости от входных данных.

Пусть даны две функции из натуральных положительных чисел f , g , которые показывают время работы двух разных алгоритмов на различных длинах входов.

Таблица 1. О-сложности алгоритма

Обозначение	Определение
$O(1)$	Константная сложность. Большинство операций выполняется только раз или несколько раз
$O(N)$	Линейная сложность алгоритма. Время работы программы линейно. Обычно, когда элемент входных данных требуется обработать лишь линейное число раз
$O(N^2)$, $O(N^3)$, $O(N^a)$	Полиномиальная сложность алгоритмов
$O(\log(N))$	Логарифмическая сложность алгоритма. Когда время работы программы логарифмировано, программа начинает работать намного медленнее с увеличением N . Такое время работы встречается обычно в программах, которые делят большую проблему на малые и решают их по отдельности
$O(N \cdot \log(N))$	Такое время работы имеют программы, которые делят большую проблему на маленькие, а затем, решив их отдельно, соединяют вместе
$O(2^N)$	Экспоненциальная сложность. Такие алгоритмы возникают чаще всего в результате подхода «метод грубой силы»

Программист должен уметь проводить анализ алгоритмов и определять их сложность. Временная сложность алгоритма может быть посчитана исходя из анализа его управляющих структур.

Алгоритмы без циклов и рекурсивных вызовов имеют константную сложность. Если нет рекурсии и циклов, все управляющие структуры могут быть сведены к структурам константной сложности. Следовательно, и весь алгоритм также характеризуется константной сложностью.

Определение сложности алгоритма в основном сводится к анализу циклов и рекурсивных вызовов.

Как правило, около 90% времени работы программы требует выполнение повторений и только 10% составляют непосредственно вычисления.

Анализ сложности программ показывает, на какие фрагменты выпадают эти 90% – это циклы наибольшей глубины вложенности. Повторения могут быть организованы в виде вложенных циклов или вложенной рекурсии.

Правила для определения сложности:

1. Мультипликативные константы можно опускать. Например, $5n^3 = O(n^3)$
2. n^a растёт быстрее, чем n^b , при $a > b$. Например, $n^2 = O(n^{2.1})$
3. Любая экспонента растёт быстрее любого полинома. Например, $n^5 = O(2^n)$ или $n^{1000} = O(1.01^n)$
4. Любой полином растёт быстрее полилогарифма. Например, $(\log_2 n)^{20} = O(\sqrt{n})$.

Содержание работы:

Работа предполагает выполнение следующих этапов.

1. Знакомство со всеми разделами руководства.
2. Получение задания на асимптотическую и верхнюю оценку сложности алгоритма и выполнение этой оценки:

Типы алгоритмов

1. Составить программу, которая формирует одномерный массив из n случайных чисел. Определить среднее арифметическое этих чисел. Значение n меняется в пределах от 10 до 50 миллионов.

2. Составить программу, которая формирует одномерный массив из n случайных чисел. Получить из него два новых массива: один из четных чисел, а другой из нечетных. Значение n меняется в пределах от 10 до 50 миллионов.

3. Составить программу, которая формирует одномерный массив из n случайных чисел. Определить сумму отрицательных чисел и отдельно сумму остальных. Значение n меняется в пределах от 10 до 50 миллионов.

4. Составить программу, которая формирует одномерный массив из n случайных чисел. Определить количество четных чисел и количество нечетных чисел. Значение n меняется в пределах от 10 до 50 миллионов.

5. Составить программу, которая формирует одномерный массив из n случайных чисел. Отдельно определить произведение четных чисел, и

произведение нечетных чисел. Значение n меняется в пределах от 10 до 50 миллионов.

6. Составить программу, которая формирует матрицу из $n \times n$ случайных чисел. Определить произведение чисел, лежащих на главной диагонали матрицы. Значение n меняется в пределах от 5 до 10 тысяч.

7. Составить программу, которая формирует матрицу из $n \times n$ случайных чисел. Определить произведение чисел, лежащих на побочной диагонали матрицы. Значение n меняется в пределах от 5 до 10 тысяч.

8. Составить программу, которая формирует матрицу из $n \times n$ случайных чисел. Определить сумму чисел, лежащих выше главной диагонали матрицы. Значение n меняется в пределах от 5 до 10 тысяч.

9. Составить программу, которая формирует матрицу из $n \times n$ случайных чисел. Определить произведение всех чисел в матрице. Значение n меняется в пределах от 5 до 10 тысяч.

10. Составить программу, которая формирует матрицу из $n \times n$ случайных чисел. Определить сумму отрицательных чисел и отдельно сумму остальных. Значение n меняется в пределах от 5 до 10 тысяч.

11. Составить программу, которая формирует матрицу из $n \times n$ случайных чисел. Определить количество четных чисел и количество нечетных. Значение n меняется в пределах от 5 до 10 тысяч.

3. Оценка экспериментальным способом времени выполнения того же алгоритма. Значения исходных данных необходимо задавать в начале работы программы с помощью генератора случайных чисел, причем делать это до начала измерения времени работы алгоритма. Сам алгоритм в ходе измерений должен выполняться в цикле несколько миллионов раз, чтобы он не заканчивал работу слишком быстро, а выполнялся хотя бы несколько секунд.

4. Измерения необходимо повторить пять раз для различного объема исходных данных. Количество повторений алгоритма в каждом измерении должно быть одинаковым.

5. Построить график зависимости времени выполнения от объема входных данных.

Контрольные вопросы

1. Чем характеризуется сложность алгоритма?
2. Как оценивается асимптотическая сложность алгоритма?
3. Как получается верхняя оценка сложности алгоритма?
4. Отличаются ли и на сколько асимптотическая и верхняя оценка сложности алгоритма?
5. Какие функции используются для представления верхней оценки сложности алгоритма?
6. У каких известных вам алгоритмов сложность является константной, а у каких - линейной?
7. Как оценивается сложность экспериментальным методом?
8. Совпадают ли результаты экспериментальной и верхней оценок и, если нет, то на сколько они отличаются?
9. Как влияет размер массива на временную сложность алгоритма?

ПРАКТИЧЕСКАЯ РАБОТА № 2

Тема: Оценка сложности алгоритмов сортировки

Цель работы: изучить несколько алгоритмов сортировки и сравнить их свойства; разработка программ, реализующих различные алгоритмы сортировки, и оценка их временной и пространственной сложности.

Оборудование: ПК, интернет, программное обеспечение – MS Word, среда программирования Visual Studio, инструкции по выполнению работы

Справочный материал:

Сортировка — это упорядочение элементов в списке. В случае, когда элемент имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки. На практике в качестве ключа часто выступает число, а в остальных полях хранятся любые данные, не влияющие на работу алгоритма.

К алгоритмам устойчивой сортировки относят следующие.

1. Сортировка пузырьком (англ. Bubble sort) — сложность алгоритма: $O(n^2)$; для каждой пары индексов производится обмен, если элементы расположены не по порядку.
2. Сортировка перемешиванием (Шейкерная, Cocktail sort, bidirectional bubble sort) — сложность алгоритма: $O(n^2)$.
3. Сортировка вставками (Insertion sort) — сложность алгоритма: $O(n^2)$; определяют место элемента в упорядоченном списке и вставляют его туда.
4. Гномья сортировка — сложность алгоритма: $O(n^2)$; сочетает методы пузырьковой сортировки и вставками.
5. Блочная сортировка (Корзинная, Bucket sort) — сложность алгоритма: $O(n)$; требуется $O(k)$ дополнительной памяти и знание о природе сортируемых данных.
6. Сортировка подсчётом (Counting sort) — сложность алгоритма: $O(n+k)$; требуется $O(n+k)$ дополнительной памяти (существует три варианта).
7. Сортировка слиянием (Merge sort) — сложность алгоритма: $O(n \log n)$; требуется $O(n)$ дополнительной памяти; упорядочивают две половины списка отдельно (первую и вторую), а затем — сливают их воедино.
8. Сортировка с помощью двоичного дерева (англ. Tree sort) — сложность алгоритма: $O(n \log n)$; требуется $O(n)$ дополнительной памяти.

Алгоритмами неустойчивой сортировки являются следующие методы.

1. Сортировка выбором (Selection sort) — сложность алгоритма: $O(n^2)$; выполняется поиск наименьшего или наибольшего элемента и помещение его в начало или конец упорядоченного списка.
2. Сортировка Шелла (Shell sort) — сложность алгоритма: $O(n \log^2 n)$; попытка улучшить сортировку вставками.
3. Сортировка расчёской (Comb sort) — сложность алгоритма: $O(n \log n)$
4. Пирамидальная сортировка (Сортировка кучи, Heapsort) — сложность алгоритма: $O(n \log n)$; список превращается в кучу, берется наибольший элемент и добавляется в конец списка.
5. Плавная сортировка (Smoothsort) — сложность алгоритма: $O(n \log n)$.

6. Быстрая сортировка (Quicksort), в варианте с минимальными затратами памяти сложность алгоритма: $O(n \log n)$ — среднее время, $O(n^2)$ — худший случай; широко известен как быстрейший из известных для упорядочения больших случайных списков; исходный набор данных разбивается на две половины так, что любой элемент первой половины упорядочен относительно любого элемента второй; затем алгоритм применяется рекурсивно к каждой половине. При использовании $O(n)$ дополнительной памяти сортировка становится устойчивой.

7. Поразрядная сортировка — сложность алгоритма: $O(n \cdot k)$; требуется $O(k)$ дополнительной памяти.

8. Сортировка перестановкой — $O(n \cdot n!)$ — худшее время. Для каждой пары осуществляется проверка верного порядка и генерируются всевозможные перестановки исходного массива.

Содержание работы:

Работа предполагает выполнение следующих этапов.

1. Знакомство со всеми разделами руководства.

2. Получение у преподавателя задания на разработку программы для алгоритмов сортировки:

1. Составить две программы, которые реализуют алгоритмы простой сортировки «пузырьком» и вставками. Исходные данные задавать с помощью датчика случайных чисел.

2. Составить две программы, которые реализуют алгоритмы простой сортировки «пузырьком» и выбором. Исходные данные задавать с помощью датчика случайных чисел.

3. Составить две программы, которые реализуют алгоритмы простой сортировки «пузырьком» и шейкером. Исходные данные задавать с помощью датчика случайных чисел.

4. Составить две программы, которые реализуют алгоритмы простой сортировки «пузырьком» и слиянием. Исходные данные задавать с помощью датчика случайных чисел.

5. Составить две программы, которые реализуют алгоритмы простой сортировки «пузырьком» и быстрой сортировки. Исходные данные задавать с помощью датчика случайных чисел.

6. Составить две программы, которые реализуют алгоритмы простой сортировки «пузырьком» и сортировки Шелла. Исходные данные задавать с помощью датчика случайных чисел.

7. Составить две программы, которые реализуют алгоритмы простой сортировки «пузырьком» и методом Боуза- Нельсона. Исходные данные задавать с помощью датчика случайных чисел.

8. Составить две программы, которые реализуют алгоритмы ускоренной сортировки «пузырьком» и вставками. Исходные данные задавать с помощью датчика случайных чисел.

9. Составить две программы, которые реализуют алгоритмы ускоренной сортировки «пузырьком» и выбором. Исходные данные задавать с помощью датчика случайных чисел.

10. Составить две программы, которые реализуют алгоритмы ускоренной сортировки «пузырьком» и шейкером. Исходные данные задавать с помощью датчика случайных чисел.

11. Составить две программы, которые реализуют алгоритмы ускоренной сортировки «пузырьком» и слиянием. Исходные данные задавать с помощью датчика случайных чисел.

12. Составить две программы, которые реализуют алгоритмы ускоренной сортировки «пузырьком» и быстрой сортировки. Исходные данные задавать с помощью датчика случайных чисел.

13. Составить две программы, которые реализуют алгоритмы ускоренной сортировки «пузырьком» и сортировки Шелла. Исходные данные задавать с помощью датчика случайных чисел.

14. Составить две программы, которые реализуют алгоритмы ускоренной сортировки «пузырьком» и методом Боуза-Нельсона. Исходные данные задавать с помощью датчика случайных чисел.

3. Разработка и отладка заданных программ.

4. Получение верхней и экспериментальной оценки времени выполнения заданных алгоритмов и программ.

5. Нахождение предельной оценки емкости памяти, необходимой для выполнения разработанных программ.

Контрольные вопросы:

1. Что такое сортировка и для чего она нужна?

2. По каким признакам выполняется классификация алгоритмов сортировки?

3. Как оценивается временная сложность алгоритмов упорядочения?

4. Как оценивается емкостная сложность алгоритмов сортировки?

5. Какой метод упорядочения самый простой?

6. Какой алгоритм сортировки самый быстрый?

7. Какие алгоритмы пригодны для упорядочения файлов?

8. Чем отличается сортировка чисел от строк?

9. Как Вы определили время выполнения Ваших алгоритмов?

10. Как Вы определили объем памяти, необходимой для выполнения Ваших алгоритмов?

11. У каких известных Вам методов сортировки временная сложность зависит от объема используемой памяти?

ПРАКТИЧЕСКАЯ РАБОТА № 3

Тема: Оценка сложности алгоритмов поиска

Цель работы: научиться разрабатывать программы, реализующие различные алгоритмы поиска, и оценка их временной и пространственной сложности.

Оборудование: ПК, интернет, программное обеспечение – MS Word, среда программирования Visual Studio, инструкции по выполнению работы

Справочный материал:

Процедуры поиска широко используются в информационно-справочных системах (базах и банках данных), при разработке синтаксических анализаторов и компиляторов (поиск служебных слов, команд и т.п. в таблицах) и др. В простейшем случае считается, что исходными данными для этой процедуры являются массив (чисел, слов и т.д.) и некоторое значение, которое принято называть аргументом поиска.

Эталоны в таблице (массиве) могут иметь сложную структуру, но характеризуются неповторяющимися значениями некоторых ключей. Искомый аргумент сравнивается с каждым из этих ключей. Если они совпадают, то аргумент найден. Результат поиска может быть булевский (аргумент есть в таблице или нет) или числовой (номер ключа в таблице).

Наиболее сложным является случай, когда аргумента поиска нет в таблице. Суждение об этом можно сделать только по окончании просмотра всего массива.

Поскольку в процессе поиска участвуют только ключи, а информационная часть элементов не важна, в дальнейшем будем рассматривать только алгоритмы поиска ключей, значения которых обычно являются целыми числами.

Наиболее распространенными являются три алгоритма поиска: 1) Линейный; 2) Дихотомический; 3) Интерполирующий.

Содержание работы:

Работа предполагает выполнение следующих этапов.

1. Знакомство со всеми разделами руководства.
2. Получение у преподавателя задания на разработку программы для алгоритмов поиска:
 1. Разработать алгоритм и программу простого линейного поиска с циклом For. В качестве исходных данных использовать строку текста, из которой необходимо выделить слова. Аргумент поиска – слово.
 2. Разработать алгоритм и программу ускоренного линейного поиска. В качестве исходных данных использовать строку текста, из которой необходимо выделить слова. Аргумент поиска – слово.
 3. Разработать алгоритм и программу дихотомического поиска. В качестве исходных данных использовать массив целых чисел, который вводится с клавиатуры. Аргумент поиска – число.
 4. Разработать алгоритм и программу дихотомического поиска. В качестве исходных данных использовать массив целых чисел, который

формируется с помощью датчика случайных чисел с диапазоном от 0 до 100. Аргумент поиска – число.

5. Разработать алгоритм и программу интерполирующего поиска. В качестве исходных данных использовать массив целых чисел, который вводится с клавиатуры. Аргумент поиска – число.

6. Разработать алгоритм и программу интерполирующего поиска. В качестве исходных данных использовать массив целых чисел, который формируется с помощью датчика случайных чисел с диапазоном от 0 до 100. Аргумент поиска – число.

7. Разработать алгоритм и программу простого линейного поиска с циклом For. В качестве исходных данных использовать строку текста, из которой необходимо выделить слова. Затем слова упорядочить по алфавиту. Аргумент поиска – слово.

8. Разработать алгоритм и программу ускоренного линейного поиска. В качестве исходных данных использовать строку текста, из которой необходимо выделить слова. Затем слова упорядочить по алфавиту. Аргумент поиска – слово.

3. Разработка и отладка заданной программы.

4. Получение верхней и экспериментальной оценки времени выполнения заданного алгоритма и программы.

5. Нахождение предельной оценки емкости памяти, необходимой для выполнения разработанной программы.

Контрольные вопросы:

1. Что такое поиск и для чего он нужен?
2. Что является исходными данными для поиска?
3. Какие алгоритмы поиска Вы знаете?
4. Приведите словесное описание простейшего алгоритма линейного поиска (с циклами For).
5. Приведите словесное описание ускоренного алгоритма линейного поиска.
6. Приведите словесное описание алгоритма дихотомического поиска.
7. Приведите словесное описание алгоритма интерполирующего поиска.
8. Какова верхняя оценка трудоемкости алгоритма линейного поиска?
9. Какова верхняя оценка трудоемкости алгоритма дихотомического поиска?
10. Какова верхняя оценка трудоемкости алгоритма интерполирующего поиска?
11. Какова верхняя оценка емкостной сложности алгоритма линейного поиска, дихотомического поиска, интерполирующего поиска?

ПРАКТИЧЕСКАЯ РАБОТА № 4

Тема: Оценка сложности рекурсивных алгоритмов

Цель работы: научиться разрабатывать программы, реализующие различные рекурсивные алгоритмы, и оценка их временной и пространственной сложности.

Оборудование: ПК, интернет, программное обеспечение – MS Word, среда программирования Visual Studio, инструкции по выполнению работы

Справочный материал:

Алгоритм называется рекурсивным, если он прямо или косвенно обращается к самому себе. Часто в основе такого алгоритма лежит рекурсивное определение какого-то понятия.

Примером сложной рекурсии является случай, когда метод А вызывает метод В, а метод В – метод А.

Количество вложенных вызовов методов называется глубиной рекурсии. Реализация рекурсивных вызовов опирается на механизм стека

вызовов. Адрес возврата и локальные переменные метода записываются в стек, благодаря чему каждый следующий рекурсивный вызов этого метода пользуется своим набором локальных переменных. Обычно рекурсия реализуется в два прохода:

- 1) формирование в стеке последовательности аргументов;
- 2) собственно вычисления (реализация метода), выполняемые над данными, помещенными в стек.

На каждый рекурсивный вызов требуется некоторое количество оперативной памяти компьютера, т.е. для рекурсии необходимо оценивать емкостную сложность. При чрезмерно большой глубине рекурсии может наступить переполнение стека, и возникнуть исключительная ситуация – переполнение стека.

Содержание работы:

Работа предполагает выполнение следующих этапов.

1. Знакомство со всеми разделами руководства.
2. Получение у преподавателя задания на разработку программы для рекурсивных алгоритмов: Разработать следующие алгоритмы и программы с использованием рекурсии:
 1. Линейного поиска целочисленного значения ключа в заданном массиве и вывода этого массива.
 2. Линейного поиска слова в заданном массиве и вывода этого массива.
 3. Дихотомического поиска целочисленного значения ключа в заданном массиве и вывода этого массива.
 4. Интерполирующего поиска целочисленного значения ключа в заданном массиве и вывода этого массива.
 5. Вычисления целой степени целого числа.
 6. Вычисления целой степени вещественного числа.
 7. Перевода целого числа, введенного с клавиатуры, в двоичную систему счисления.

8. Перевод целого числа, введенного с клавиатуры, в систему счисления с основанием q .

9. Ввода одномерного массива и линейного поиска целочисленного значения ключа в нем.

10. Ввода одномерного массива слов и линейного поиска заданного слова в нем.

11. Ввода одномерного массива и дихотомического поиска целочисленного значения ключа в нем.

12. Ввода одномерного массива и интерполирующего поиска целочисленного значения ключа в нем.

3. Разработка и отладка заданной программы.

4. Получение верхней и экспериментальной оценки времени выполнения заданного алгоритма и программы.

5. Нахождение предельной оценки емкости памяти, необходимой для выполнения разработанной программы.

Контрольные вопросы:

1. Что такое рекурсия и для чего она нужна?

2. Чем отличается простая рекурсия от сложной?

3. Какие классические рекурсивные алгоритмы Вы знаете?

4. Чем можно заменить рекурсию?

5. Какими характеристиками сложности описывается рекурсия?

6. Что такое глубина рекурсии?

7. Что представляет собой дерево рекурсии?

8. Как можно вычислить факториал без использования рекурсивного алгоритма?

9. Как можно вычислить числа Фибоначчи без использования рекурсивного алгоритма?

10. Как можно оценить емкостную сложность рекурсивного алгоритма?

ПРАКТИЧЕСКАЯ РАБОТА № 5

Тема: Оценка сложности эвристических алгоритмов

Цель работы: научиться разрабатывать программы, реализующие различные эвристические алгоритмы, и оценка их временной и пространственной сложности; получение практических навыков в работе с задачами на определение сложности алгоритма.

Оборудование: ПК, интернет, программное обеспечение – MS Word, среда программирования Visual Studio, инструкции по выполнению работы

Содержание работы:

Задание: Составить алгоритм и определить его сложность для следующих задач:

1. Задан массив $A(6,5)$. Найти номер строки и номер столбца с минимальным значением.
2. Найти количество отрицательных элементов в массиве $C1, C2, \dots, C20$, используя оператор цикла *do until_loop*.
3. Найти максимальное значение из значений элементов последовательности $x1, x2, \dots, x20$ (одномерного массива), используя оператор *do while...loop*
4. Определите произведение трех переменных.

ПРАКТИЧЕСКАЯ РАБОТА № 6

Тема: Работа с классами

Цель работы: научиться разрабатывать классы и их элементы на языке C#

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Справочный материал:

Все свойства и методы классов имеют права доступа. По умолчанию, все содержимое класса является доступным для чтения и записи только для него самого. Для того, чтобы разрешить доступ к данным класса извне, используют модификатор доступа `public`. Все функции и переменные, которые находятся после модификатора `public`, становятся доступными из всех частей программы. Закрытые данные класса размещаются после модификатора доступа `private`. Если отсутствует модификатор `public`, то все функции и переменные, по умолчанию являются закрытыми (как в первом примере).

Обычно, приватными делают все свойства класса, а публичными — его методы. Все действия с закрытыми свойствами класса реализуются через его методы.

Содержание работы:

Задание 1. Написать программу для определения класса, описывающего некоторого студента вуза.

1. Запустить Visual Studio
2. Создать консольное приложение.
3. Код будет выглядеть следующим образом:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication12
{
    class Student
    {
        //описываем характеристики объекта класса
        //характеристики доступны только внутри данного класса
        private string name;
        private string sename;
        private int kurs;
        private string adress;
        private int tel;
        //определяем конструктор класса для создания объектов данного класса
        //в скобках указаны параметры, которые передаются методу для
        //создания//объекта с //указанными свойствами
        public Student (string name, string sename, int kurs, string adress, int tel)
        {
            this.name = name;
            this.sename = sename;
            this.kurs = kurs;
            this.adress = adress;
```

```

this.tel = tel;      }
//определяем метод для объекта класса (его поведение)
//метод будет выводить на экран информацию о студенте
public void Print ()      {
Console.WriteLine ("Имя\t"+name+"\nФамилия\t"+sename+"\nКурс\t"+kurs+"\nАдрес\t"+adress+"\nТелефон\t"+tel+"\n");      }      }
class Program      {
static void Main (string [] args) {
//создаем первый объект нашего класса
Student St1 = new Student («Иван»,»Иванов»,1,»Ленина, 50–25»,3251565);
//создаем второй объект нашего класса
Student St2 = new Student («Петр», «Петров», 2, «Мира, 15–3», 2256580);
//используем метод вывода на экран информации о студентах,
//определенный ранее в классе
St1.Print ();
St2.Print ();
Console.ReadKey ();      }      }      }

```

```

C:\Windows\system32\cmd.exe
Имя      Иван
Фамилия  Иванов
Курс     1
Адрес    Ленина, 50–25
Телефон  3251565

Имя      Петр
Фамилия  Петров
Курс     2
Адрес    Мира, 15–3
Телефон  2256580

```

Задание 2. Описать класс «Домашняя библиотека».

Задание 3. Описать класс «Записная книжка».

Задание 4. Описать класс «Студенческая группа».

ПРАКТИЧЕСКАЯ РАБОТА № 7

Тема: Работа с классами

Цель работы: научиться разрабатывать классы и их элементы на языке C#

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Содержание работы:

Задание 1. Создать программу, которая будет заниматься учетом успеваемости студентов в группе. Создайте заголовочный файл students.h, в котором будет находиться класс Students.

1. Запустить Visual Studio

2. Создать консольное приложение.

3. Введите следующий код:

```
/* students.h */
#include
class Students {
public:
    // Установка имени студента
    void set_name(std::string student_name)    {
        name = student_name;    }
    // Получение имени студента
    std::string get_name()    {
        return name;    }
    // Установка фамилии студента
    void set_last_name(std::string student_last_name) {
        last_name = student_last_name;    }
    // Получение фамилии студента
    std::string get_last_name()    {
        return last_name;    }
    // Установка промежуточных оценок
    void set_scores(int student_scores[])    {
        for (int i = 0; i < 5; ++i) {
            scores[i] = student_scores[i];    }    }
    // Установка среднего балла
    void set_average_ball(float ball)    {
        average_ball = ball;    }
    // Получение среднего балла
    float get_average_ball() {
        return average_ball;    }
private:
    // Промежуточные оценки
    int scores[5];
    // Средний балл
    float average_ball;
    // Имя
```

```
std::string name;
```

```
// Фамилия
```

```
std::string last_name;    };
```

4. Функция `get_average_ball()` используется для получения средней оценки студента, и `set_average_ball()` для выставления этой оценки.

5. Функция `set_average_ball()` принимает средний балл в качестве параметра и присваивает его значение закрытой переменной `average_ball`.

6. Функция `get_average_ball()` просто возвращает значение этой переменной.

7. Функция `set_name()` сохраняет имя студента в переменной `name`, а `get_name()` возвращает значение этой переменной. Принцип работы функций `set_last_name()` и `get_last_name()` аналогичен.

8. Функция `set_scores()` принимает массив с промежуточными оценками и сохраняет их в приватную переменную `int scores[5]`.

9. Теперь создайте файл `main.cpp` со следующим содержимым.

```
/* main.cpp */
```

```
#include
```

```
#include "students.h"
```

```
int main() {
```

```
// Создание объекта класса Student
```

```
Students student;
```

```
std::string name;
```

```
std::string last_name;
```

```
// Ввод имени с клавиатуры
```

```
std::cout << "Name: ";
```

```
getline(std::cin, name);
```

```
// Ввод фамилии
```

```
std::cout << "Last name: ";
```

```
getline(std::cin, last_name);
```

```
// Сохранение имени и фамилии в объект класса Students
```

```
student.set_name(name);
```

```
student.set_last_name(last_name);
```

```
// Оценки
```

```
int scores[5];
```

```
// Сумма всех оценок
```

```
int sum = 0;
```

```
// Ввод промежуточных оценок
```

```
for (int i = 0; i < 5; ++i) {
```

```
std::cout << "Score " << i+1 << ": ";
```

```
std::cin >> scores[i];
```

```
// суммирование
```

```
sum += scores[i];    }
```

```
// Сохраняем промежуточные оценки в объект класса Student
```

```
student.set_scores(scores);
```

```
// Считаем средний балл
```

```
float average_ball = sum / 5.0;
```

```
// Сохраняем средний балл в объект класса Students
student.set_average_ball(average_ball);
// Выводим данные по студенту
std::cout << "Average ball for " << student.get_name() << " "
<< student.get_last_name() << " is "
<< student.get_average_ball() << std::endl;
return 0; }
```

В самом начале программы создается объект класса Students. Дело в том, что сам класс является только описанием его объекта. Класс Students является описанием любого из студентов, у которого есть имя, фамилия и возможность получения оценок.

Объект класса Students характеризует конкретного студента. Если мы захотим выставить оценки всем ученикам в группе, то будем создавать новый объект для каждого из них. Использование классов очень хорошо подходит для описания объектов реального мира.

После создания объекта student, мы вводим с клавиатуры фамилию, имя и промежуточные оценки для конкретного ученика. Пускай это будет Вася Пупкин, у которого есть пять оценок за семестр — две тройки, две четверки и одна пятерка.

Введенные данные мы передаем set-функциям, которые присваивают их закрытым переменным класса. После того, как были введены промежуточные оценки, мы высчитываем средний балл на основе этих оценок, а затем сохраняем это значение в закрытом свойстве average_ball, с помощью функции set_average_ball().

10.Скомпилируйте и запустите программу.

Задание 2. Разработать класс, инкапсулирующий двумерный массив. Класс должен содержать поля и методы, необходимые для реализации приведенного ниже задания:

Варианты заданий

Вариант 1. Дана вещественная матрица размером 4 строки, 5 столбцов. Переставляя ее строки и столбцы, добейтесь того, чтобы наибольший элемент (один из них) оказался в верхнем левом углу.

Вариант 2. Определите, является ли заданная целочисленная квадратная матрица порядка 5 симметричной относительно главной диагонали.

Вариант 3. Дана вещественная матрица размером 4 строки, 5 столбцов. Поменяйте местами максимальный и минимальный элементы матрицы.

Вариант 4. Дана целочисленная квадратная матрица порядка 5. Найдите максимальный элемент среди элементов, лежащих ниже главной диагонали, и максимальный элемент среди элементов, лежащих выше главной диагонали, поменяйте их местами.

Вариант 5. Дана целочисленная квадратная матрица порядка 5. Найдите максимальный элемент среди элементов, лежащих левее вспомогательной диагонали, и максимальный элемент среди элементов, лежащих правее вспомогательной диагонали, поменяйте их местами.

ПРАКТИЧЕСКАЯ РАБОТА № 8

Тема: Работа с объектами через интерфейсы

Цель работы: формирование целостного представления о назначении интерфейса овладение практическими навыками работы с объектами через интерфейсы.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Справочный материал:

Интерфейс представляет некое описание типа, набор компонентов, который должен иметь тип данных. И, собственно, мы не можем создавать объекты интерфейса напрямую с помощью конструктора, как например, в классах:

```
IMovable m = new IMovable(); // ! Ошибка, так сделать нельзя
```

В конечном счете, интерфейс предназначен для реализации в классах и структурах. Например, возьмем следующий интерфейс IMovable:

```
interface IMovable{  
    void Move();    }
```

Затем какой-нибудь класс или структура могут применить данный интерфейс:

```
// применение интерфейса в классе
```

```
class Person : IMovable {  
    public void Move()    {  
        Console.WriteLine("Человек идет");    }    }
```

```
// применение интерфейса в структуре
```

```
struct Car : IMovable    {  
    public void Move()    {  
        Console.WriteLine("Машина едет");    }    }
```

При применении интерфейса, как и при наследовании после имени класса или структуры указывается двоеточие и затем идут названия применяемых интерфейсов. При этом класс должен реализовать все методы и свойства применяемых интерфейсов, если эти методы и свойства не имеют реализации по умолчанию.

Если методы и свойства интерфейса не имеют модификатора доступа, то по умолчанию они являются публичными, при реализации этих методов и свойств в классе и структуре к ним можно применять только модификатор public.

Содержание работы:

Задание 1. Применение интерфейса в программе:

```
using System;  
namespace HelloApp{  
    interface IMovable    {  
        void Move();    }  
    class Person : IMovable    {  
        public void Move()    {  
            Console.WriteLine("Человек идет");    }    }  
    struct Car : IMovable    {
```

```

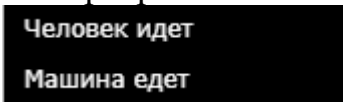
public void Move()    {
    Console.WriteLine("Машина едет");    } }
class Program    {
    static void Action(IMovable movable)    {
        movable.Move();    }
    static void Main(string[] args)    {
        Person person = new Person();
        Car car = new Car();
        Action(person);
        Action(car);
        Console.Read();    } } }

```

В данной программе определен метод Action(), который в качестве параметра принимает объект интерфейса IMovable. На момент написания кода мы можем не знать, что это будет за объект - какой-то класс или структура. Единственное, в чем мы можем быть уверены, что этот объект обязательно реализует метод Move и мы можем вызвать этот метод.

Иными словами, интерфейс - это контракт, что какой-то определенный тип обязательно реализует некоторый функционал.

Консольный вывод данной программы:



```

Человек идет
Машина едет

```

Задание 2. Интерфейс для класса «Матрица»

1. Интерфейс работы с математическими объектами, в качестве которых могут выступать матрицы, полиномы, векторы, дроби, комплексные числа должен определять общие операции: сложение, вычитание, умножение на объект и умножение на число. Также требуется включить в интерфейс метод получения строкового представления объекта, чтобы использовать полученную строку для вывода объекта.

```

interface IMathObject    {
    // метод получения суммы объектов
    IMathObject Summa (IMathObject ob);
    // метод получения разности объектов
    IMathObject Substract (IMathObject ob);
    // метод умножения объектов
    IMathObject Multiply (IMathObject ob);
    // метод умножения объекта на число
    IMathObject Multiply(double chislo);
    // метод получения строкового представления объекта
    string ToString(); }

```

2. Для раскрытия интерфейса требуется определить в классе Matrix все методы, которые указаны в интерфейсе. Приведем код класса Matrix с указанием переопределенных операций.

```

class Matrix: IMathObject    {
    // количество строк и столбцов матрицы

```

```

protected int m, n;
// массив элементов матрицы
protected double[,] a;
// конструктор - осуществляет выделение памяти под хранение матрицы
public Matrix(int m1, int n1) {
    n = n1;
    m = m1;
    a = new double[m, n]; }
// определение операции сложения двух матриц,
// раскрывающей метод интерфейса IMathObject
public IMathObject Summa(IMathObject ob) {
    // приведение типа аргумента к классу Matrix
    Matrix ob1 = ob as Matrix;
    if (m != ob1.m || n != ob1.n)
        throw new Exception("Сложение таких матриц невозможно");
    Matrix res = new Matrix(m, n);
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            res[i, j] = a[i, j] + ob1[i, j];
    return res; }
// определение операции вычитания двух матриц,
// раскрывающей метод интерфейса IMathObject
public IMathObject Subtract(IMathObject ob) {
    Matrix ob1 = ob as Matrix;
    if (m != ob1.m || n != ob1.n)
        throw new Exception("Вычитание таких
матриц невозможно");
    Matrix res = new Matrix(m, n);
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            res[i, j] = a[i, j] - ob1[i, j];
    return res; }
// определение операции умножения двух матриц,
// раскрывающей метод интерфейса IMathObject
public IMathObject Multiply(IMathObject ob) {
    Matrix ob1 = ob as Matrix;
    if (n != ob1.m)
        throw new Exception("Такие матрицы перемножить нельзя");
    Matrix res = new Matrix(m, ob1.n);
    for (int i = 0; i < m; i++)
        for (int j = 0; j < ob1.n; j++) {
            res[i, j] = 0;
            for (int k = 0; k < n; k++)
                res[i, j] = res[i, j] + a[i, k] * ob1[k, j];
        }
    return res; }

```

```

// определение операции умножения матрицы на число,
// раскрывающей метод интерфейса IMathObject
public IMathObject Multiply(double chislo)      {
    Matrix res = new Matrix(m, n);
    for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++)
    res[i, j] = a[i, j] * chislo;
    return res;  }

// индексатор для получения элементов матрицы по индексам
double this[int i, int j]  {
    get { return a[i, j]; }
    set { a[i, j] = value; }  }

// метод ввода матрицы
public void Input()      {
    Console.WriteLine("Введите элементы матрицы");
    for (int i = 0; i < m; i++)      {
        string str = Console.ReadLine();
        string [] s = str.Split(' ');
        for (int j = 0; j < n; j++)
        a[i, j] = double.Parse(s[j]);    }    }

// метод получения строкового представления матрицы.
// Элементы в матрице располагаются через символ
// табуляции, при переходе на новую строку матрицы
// добавляется символ '\n'
public override string ToString()      {
    string str = "";
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n - 1; j++)
        str = str + a[i, j] + "\t";
        str = str + a[i, n - 1] + "\n";    }
    return str;  }    }

```

Задание 3. Раскрыть интерфейс для класса «Полином».

ПРАКТИЧЕСКАЯ РАБОТА № 9

Тема: Использование стандартных интерфейсов

Цель работы: формирование целостного представления о назначении интерфейса, овладение практическими навыками работы со стандартными интерфейсами.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Справочный материал:

В библиотеке классов .NET определено множество стандартных интерфейсов, задающих желаемое поведение объектов. Например, интерфейс `Comparable` задает метод сравнения объектов на «больше-меньше», что позволяет выполнять их сортировку. Реализация интерфейсов `Enumerable` и `Enumerator` дает возможность просматривать содержимое объекта с помощью `foreach`, а реализация интерфейса `Cloneable` — клонировать объекты.

Стандартные интерфейсы поддерживаются многими стандартными классами библиотеки. Например, работа с массивами с помощью `foreach` возможна потому, что тип `Array` реализует интерфейсы `Enumerable` и `Enumerator`.

Можно создавать и собственные классы, поддерживающие стандартные интерфейсы, что позволит использовать объекты этих классов стандартными способами.

Содержание работы:

Задание 1. Написать программу, реализующую интерфейс

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    // Создаем два интерфейса, описывающих абстрактные методы
    // арифметических операций и операций Sqrt и Sqr
    public interface IArOperation
    {
        // Определяем набор абстрактных методов
        int Sum();
        int Otr();
        int Prz();
        int Del();
    }
    public interface ISqrSqrt
    {
        int Sqr(int x);
        int Sqrt(int x);
    }
    // Данный класс реализует интерфейс IArOperation
    class A : IArOperation
    {
        int My_x, My_y;
        public int x
        {
            set { My_x = value; }
        }
    }
}
```

```

        get { return My_x; }      }
public int y      {
    set { My_y = value; }
    get { return My_y; }      }
public A() { }
public A(int x, int y)      {
    this.x = x;
    this.y = y;      }
// Реализуем методы интерфейса
public virtual int Sum()      {
    return x + y;      }
public int Otr()      {
    return x - y;      }
public int Prz()      {
    return x * y;      }
public int Del()      {
    return x / y;      }
// В данном классе так же можно реализовать собственные методы
public virtual void rewrite()      {
    Console.WriteLine("Переменная x: {0}\nПеременная y: {1}",x,y);      } }
// Данный класс унаследован от класса A, но при этом в нем не нужно
// заново реализовывать интерфейс, но при этом можно переопределить
// некоторые его методы
class Aa: A      {
    public int z;
    public Aa(int z, int x, int y)
        : base(x, y)      {
        this.z = z;      }
    // Переопределим метод Sum
    public override int Sum()      {
        return base.x + base.y + z;      }
    public override void rewrite()      {
        base.rewrite();
        Console.WriteLine("Переменная z: " + z);      } }
// Данный класс унаследован от класса A, и при этом
// реализует интерфейс ISqrSqrt
class Ab: A, ISqrSqrt      {
    public int Sqr(int x)      {
        return x * x;      }
    public int Sqrt(int x)      {
        return (int)Math.Sqrt((double)(x));      } }
class Program      {
    static void Main()      {
        A obj1 = new A(x: 10, y: 12);
        Console.WriteLine("obj1: ");
    }
}

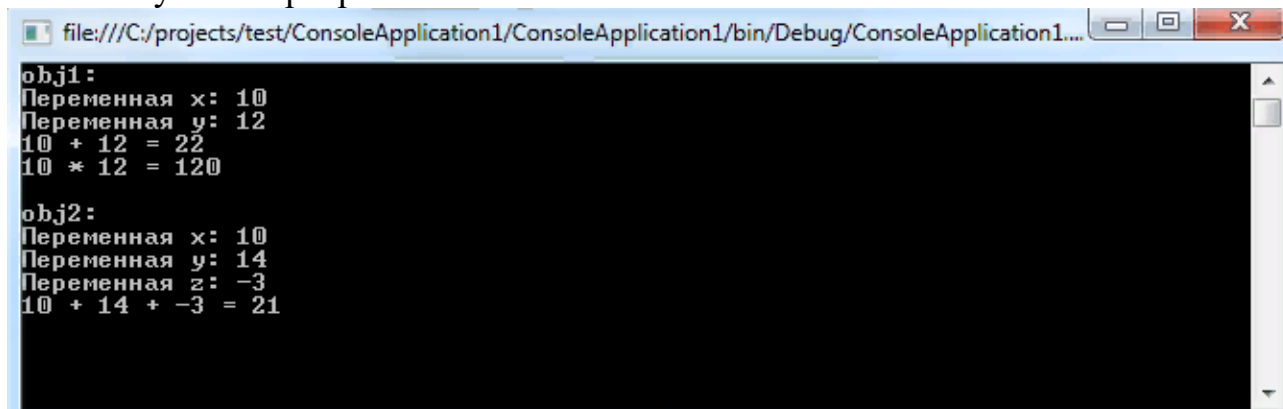
```

```

obj1.rewrite();
Console.WriteLine("{0} + {1} = {2}",obj1.x,obj1.y,obj1.Sum());
Console.WriteLine("{0} * {1} = {2}", obj1.x, obj1.y, obj1.Prz());
Aa obj2 = new Aa(z: -3, x: 10, y: 14);
Console.WriteLine("\nobj2: ");
obj2.rewrite();
Console.WriteLine("{0} + {1} + {3} = {2}", obj2.x, obj2.y, obj2.Sum(),
obj2.z);
Console.ReadLine();    }    }

```

Результат программы:



```

obj1:
Переменная x: 10
Переменная y: 12
10 + 12 = 22
10 * 12 = 120

obj2:
Переменная x: 10
Переменная y: 14
Переменная z: -3
10 + 14 + -3 = 21

```

ПРАКТИЧЕСКАЯ РАБОТА № 10

Тема: Работа с типом данных структура

Цель работы: формирование целостного представления о назначении типа данных структура, овладение практическими навыками работы с типом данных структура.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Справочный материал:

Структуры отличаются от классов тем, как они сохраняются в памяти и как к ним осуществляется доступ, а также некоторыми свойствами (например, структуры не поддерживают наследование). Из соображений производительности вы будете использовать структуры для небольших типов данных. Однако в отношении синтаксиса структуры очень похожи на классы. Главное отличие состоит в том, что при их объявлении используется ключевое слово `struct` вместо `class`. Ниже приведена общая форма объявления структуры:

```
struct имя : интерфейсы {  
    // объявления членов  
}
```

где имя обозначает конкретное имя структуры.

Содержание работы:

Задание 1. Программа применения структуры

```
using System;  
namespace ConsoleApplication1    {  
    // Создадим структуру  
    struct UserInfo    {  
        public string Name;  
        public byte Age;  
        public UserInfo(string Name, byte Age)    {  
            this.Name = Name;  
            this.Age = Age;    }  
        public void WriteUserInfo()    {  
            Console.WriteLine("Имя: {0}, возраст: {1}",Name,Age);    }    }  
    class Program    {  
        static void Main()    {  
            UserInfo user1 = new UserInfo("Alexandr", 26);  
            Console.Write("user1: ");  
            user1.WriteUserInfo();  
            UserInfo user2 = new UserInfo("Elena",22);  
            Console.Write("user2: ");  
            user2.WriteUserInfo();  
            // Показать главное отличие структур от классов  
            user1 = user2;  
            user2.Name = "Natalya";  
            user2.Age = 25;  
            Console.Write("\nuser1: ");
```



```

user1.WriteUserInfo();
Console.WriteLine("user2: ");
user2.WriteUserInfo();
Console.ReadLine();    }    }    }

```

```

file:///C:/projects/test/ConsoleApplication1/ConsoleApplication1/bin/Debug/ConsoleApplication1...
user1: Имя: Alexandr, возраст: 26
user2: Имя: Elena, возраст: 22

user1: Имя: Elena, возраст: 22
user2: Имя: Natalya, возраст: 25

```

Задание 2. Известно, что точка на экране задается парой целых чисел (X, Y) в пикселях. Объединим эти два числа в структуру *точка*. Установим максимальную защиту (*private* – по умолчанию) для этих полей. Добавим конструктор с параметрами *точка*(x, y) для инициализации объектов структуры *точка*, а также метод *Get()* для извлечения полей в строковом формате.

Размер прямоугольника на экране также задается парой чисел: шириной и высотой. Соответственно объявим структуру *размер* с полями (W, H) . Добавим аналогичный конструктор с параметрами *размер*(w, h) для инициализации объектов структуры *размер*, а также метод *Get()* для извлечения полей в строковом формате.

Прямоугольник на экране определяется координатами верхнего левого угла (*точка*) и размером изображения (*размер*). Поэтому третью структуру построим на основе первых двух, назовем ее *Rect* (сокращенно от «прямоугольник»), полями которой будут объекты первых двух структур: *точка* *P* и *размер* *S*. Эти поля также будут защищенными (*private* – по умолчанию). Добавим в эту структуру конструктор с параметрами *public Rect*($int\ x, int\ y, int\ w, int\ h$) и три метода: *Get()* для извлечения информации о прямоугольнике, *Get_P()* и *Get_S* для извлечения координат и размеров по отдельности.

В методе *Main()* последовательно инициализируем объект *Rect* *r* и покажем работу методов всех этих трех структур.

```

using System;
namespace прямоугольник {
    struct точка {
        int X;
        int Y;
        public точка (int x, int y)    {
            X = x;
            Y = y;    }
        public string Get()    {
            return X.ToString()+","+Y.ToString();    }    }
    struct размер {

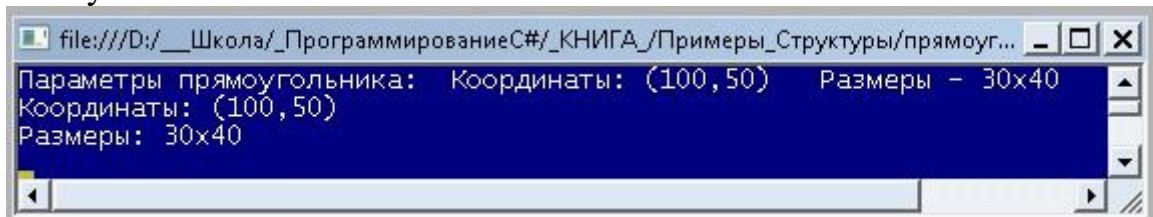
```

```

int W;
int H;
public размер(int w, int h)    {
    W = w;
    H = h;    }
public string Get()    {
    return W.ToString() + "x" + H.ToString();    } }
struct Rect {
    точка P;
    размер S;
    public Rect(int x, int y, int w, int h)    {
        P = new точка(x,y);
        S = new размер(w,h);    }
    public void Get()    {
        Console.WriteLine("Параметры прямоугольника: ");
        Console.WriteLine("Координаты: ({0})", P.Get());
        Console.WriteLine("  Размеры - {0}", S.Get());
        Console.WriteLine();    }
    public точка Get_P()    {
        return P;    }
    public размер Get_S()    {
        return S;    } }
class Program {
    static void Main(string[] args)    {
        Rect r = new Rect(100,50,30,40);
        r.Get();
        точка q = r.Get_P();    // т.к. r.P; - ошибка
        string z = "Координаты: (" + q.Get()+")";
        Console.WriteLine(z);
        размер d = r.Get_S();    // т.к. r.S; - ошибка
        z = "Размеры: " + d.Get();
        Console.WriteLine(z);
        Console.ReadKey();    } }}

```

Результат:



```

file:///D:/__Школа/_ПрограммированиеС#/_КНИГА/_Примеры_Структуры/прямоуг...
Параметры прямоугольника: Координаты: (100,50)  Размеры - 30x40
Координаты: (100,50)
Размеры: 30x40

```

ПРАКТИЧЕСКАЯ РАБОТА № 11

Тема: Работа с типом данных структура

Цель работы: формирование целостного представления о назначении типа данных структура, овладение практическими навыками работы с типом данных структура.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Содержание работы:

Задание 1. Создать структуру студент

```
namespace ConsoleApplication9    {
//объявление структуры, обратите внимание на место в консольном
приложении
public struct student    {
//поля структуры
public string name;
public int kurs;
public string gruppa;
public int stipendia;
// метод структуры (перегруженный)
public override string ToString()    {
return (string.Format( "Имя студента {0}; Курс{1}; Группа № {2};Размер
стипендии: {3}" , name, kurs,gruppa,stipendia ) );    }//конец метода
}; //конец описания структуры student
class Program    {
static void Main(string[] args)    {
student s; //объявление экземпляра (переменной) структуры
Console.WriteLine("Введите данные о студенте:");
Console.WriteLine("Имя:");
//ДОСТУП к элементам структуры – через операцию “точка”
s.name = Console.ReadLine();
Console.WriteLine("Курс:");
s.kurs = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("Группа");
s.gruppa = Console.ReadLine();
Console.WriteLine("Размер стипендии:");
s.stipendia = Convert.ToInt32(Console.ReadLine());
// вызов перегруженного метода ToString()
Console.WriteLine( "структура s: " +s); //вывод всех полей структуры на экран
Console.ReadKey();} } }
```

Задание 2. Создать структуру данных, которая хранит информацию о банковском счете – его номер, тип и баланс. Создать переменную такого типа, заполнить структуру значениями и напечатать результат.

Задание 3. Создать структуру работник с двумя полями: имя, ВУЗ. Заполнить структуру данными

ПРАКТИЧЕСКАЯ РАБОТА № 12

Тема: Коллекции. Параметризованные классы

Цель работы: формирование целостного представления о назначении коллекции и параметризованных классов, овладение практическими навыками работы с коллекциями, параметризованными классами.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Справочный материал:

Коллекции предоставляют гибкий способ работы с группами объектов. В отличие от массивов, коллекция, с которой вы работаете, может расти или уменьшаться динамически при необходимости. Некоторые коллекции допускают назначение ключа любому объекту, который добавляется в коллекцию, чтобы в дальнейшем можно было быстро извлечь связанный с ключом объект из коллекции.

Коллекция является классом, поэтому необходимо объявить экземпляр класса перед добавлением в коллекцию элементов. Пространство имен System.Collections содержит классы и интерфейсы, которые определяют различные коллекции объектов.

Параметризованные классы – классы, позволяющие определить тип своих аргументов при непосредственном создании объектов.

Пример параметризованного класса:

```
public class AClass1<T>      {
private T[] myArray = new T[20];      }
public class M      {
static void Main(string[] args)      {
AClass1<string> K = new AClass1<string>();
AClass1<int> K2 = new AClass1<int>(); } }
```

Основное ограничение, налагаемое на параметризованные классы при их создании: необходимо следить, чтобы операции, используемые для типа-параметра, были определены для всех типов или же использовать механизмы преобразования типов.

Содержание работы:

Задание 1. Создание и применение двух коллекций

```
using System;
using System.Collections;
using System.Collections.Generic;
namespace Collections {
class Program {
static void Main(string[] args) {
// неособобщенная коллекция ArrayList
ArrayList objectList = new ArrayList() { 1, 2, "string", 'c', 2.0f };
object obj = 45.8;
objectList.Add(obj);
objectList.Add("string2");
objectList.RemoveAt(0); // удаление первого элемента
```

```

        foreach (object o in objectList)
        {
            Console.WriteLine(o);
        }
        Console.WriteLine("Общее число элементов коллекции: {0}",
objectList.Count);
        // обобщенная коллекция List
        List<string> countries = new List<string>() {"Россия", "США",
"Великобритания", "Китай" };
        countries.Add("Франция");
        countries.RemoveAt(1); // удаление второго элемента
        foreach (string s in countries)
        {
            Console.WriteLine(s);
        }
        Console.ReadLine();
    } }

```

Здесь используются две коллекции: необобщенная - `ArrayList` и обобщенная - `List`. Большинство коллекций поддерживают добавление элементов. Например, в данном случае добавление производится методом `Add`, но для других коллекций название метода может отличаться. Также большинство коллекций реализуют удаление (в данном примере производится с помощью метода `RemoveAt`).

С помощью свойства `Count` у коллекций можно посмотреть количество элементов.

И так как коллекции реализуют интерфейс `IEnumerable/IEnumerable<T>`, то все они поддерживают перебор в цикле `foreach`.

Задание 2. Для заданной предметной области создать программу, состоящую из трех-пяти классов. Каждый из создаваемых классов должен иметь не менее трёх методов, свойств, конструкторов. Предусмотреть использование типа данных – перечисление, коллекций `List<T>` (варианты заданий с нечётными номерами), `Dictionary<TKey, TValue>` (варианты заданий с чётными номерами). Ввод/вывод данных должен быть реализован вне классов.

Вариант 1. Предметная область: АТС. На АТС хранится информация о всех клиентах станции. АТС имеет список тарифов на междугородние разговоры. Клиент АТС может совершать множество звонков в различные города. Система должна:

- позволять вводить информацию о тарифах;
- вводить информацию о клиентах и регистрировать звонки;
- по введенной фамилии о клиенте определять стоимость всех сделанных им звонков в соответствии с действующими тарифами;
- вычислять общую стоимость всех выполненных на АТС звонков.

Вариант 2. Предметная область: Вокзал. Касса вокзала имеет список тарифов на различные направления. При покупке билета регистрируются паспортные данные пассажира. Пассажир покупает билеты на различные направления. Система должна:

- позволять вводить данные о тарифах;

- позволять вводить паспортные данные пассажира и регистрировать покупку билета;
- рассчитывать стоимость купленных пассажиром билетов;
- после ввода наименования направления выводить список всех пассажиров, купивших на него билет.

Вариант 3. Предметная область: ЖЭК. В ЖЭК хранятся тарифы на коммунальные услуги. ЖЭК имеет информацию о всех жильцах. При потреблении жильцами коммунальных услуг информация регистрируется в системе. Система должна позволять выполнять следующие задачи:

- ввод тарифов;
- ввод информации о жильцах и потребленных услугах;
- после ввода фамилии выводить сумму всех потребленных услуг;
- выводить стоимость всех оказанных услуг.

Вариант 4. Предметная область: Аэропорт. Касса аэропорта имеет список тарифов на различные направления. При покупке билета регистрируются паспортные данные. Система должна:

- позволять вводить данные о тарифах;
- позволять вводить паспортные данные пассажира и регистрировать покупку билета;
- рассчитывать стоимость купленных пассажиром билетов;
- рассчитывать стоимость всех проданных билетов.

Вариант 5. Предметная область: Банк. Информационная система банка хранит описание процентов по различным вкладам. Система хранит информацию о вкладчиках и сделанных ими вкладах. Каждый клиент может поместить в банк только один вклад. Система должна позволять выполнять следующие задачи:

- хранить информацию о процентах по вкладам;
- хранить информацию о клиентах;
- пополнять клиенту величину вклада;
- вычислять общую сумму выплат по процентам для всех вкладов.

Вариант 6. Предметная область: Отдел расчета зарплаты. Информационная система отдела расчета зарплаты на предприятии хранит данные о величине оплаты за различные виды работ. Система хранит информацию о работниках предприятия. Система должна позволять выполнять следующие задачи:

- вводить информацию о различных видах работ;
- вводить информацию о работниках и выполненных ими работах;
- после ввода фамилии выводить для работника зарплату;
- выводить сумму выплат всем работникам.

Вариант 7. Предметная область: Фирма грузоперевозок. Фирма имеет список тарифов по перевозке грузов. Клиент регистрируется в системе, после чего может заказать перевозку определенного объема груза. Система должна позволять выполнять следующие задачи:

- ввод тарифов;
- регистрация клиента и заказ на перевозку грузов;
- вывод суммы заказа для определенного клиента;
- подсчет суммарной стоимости всех заказов.

Вариант 8. Предметная область: Гостиница. Информационная система гостиницы хранит информацию о всех номерах и их стоимости. Система регистрирует клиентов. Каждый клиент может заказать один номер. При попытке заказа номера, который занят, выводится предупреждение. Система должна позволять выполнять следующие задачи:

- ввод информации о номерах и их стоимости;
- регистрация клиента и заказ номера;
- вывод списка незанятых номеров;
- после ввода фамилии клиента вывод стоимости проживания.

Вариант 9. Предметная область: Интернет-оператор. Провайдер имеет различные тарифы доступа в Интернет за 1 Мбайт в зависимости от величины абонентской платы. Информационная система провайдера хранит данные о клиентах. Система должна позволять выполнять следующие задачи:

- ввод тарифов;
- регистрация пользователя;
- ввод данных о потребленном трафике для конкретного пользователя;
- подсчет общей стоимости реализованного трафика;
- поиск клиента, заплатившего наибольшую стоимость за услуги.

Вариант 10. Предметная область: Интернет-магазин. В информационной системе хранятся данные о товарах. Клиент звонит в магазин и оставляет заказ на товар. Система должна позволять выполнять следующие задачи:

- ввод информации о товарах;
- регистрация заказа клиента на покупку определенного товара;
- после ввода фамилии покупателя вывод списка заказанных им товаров;
- после ввода фамилии покупателя вывод суммы заказа.

ПРАКТИЧЕСКАЯ РАБОТА № 13

Тема: Использование регулярных выражений

Цель работы: освоить принципы поиска и сопоставления строковых данных с использованием регулярных выражений, написать программу с их использованием.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Справочный материал:

Регулярные выражения – это один из способов поиска подстрок (соответствий) в строках. Осуществляется с помощью просмотра строки в поисках некоторого шаблона. С помощью регулярных выражений выполняются 3 действия:

- проверка наличия соответствующей шаблону подстроки;
- поиск и выдача пользователю соответствующих шаблону подстрок;
- замена соответствующих шаблону подстрок.

Регулярное выражение на C# задается строковой константой. В C# работа с регулярными выражениями выглядит следующим образом:

```
Regex re = new Regex(«образец», «опции»);  
MatchCollection me = re.Matches(—строка для поиска);  
iCountMatchs = me.Count,
```

где re – это объект типа Regex. В конструкторе ему передается образец поиска и опции.

Основные методы класса Regex:

- метод Match запускает поиск первого соответствия. Параметром передается строка поиска. Метод возвращает объект класса Match, описывающий результат поиска.

- метод Matches позволяет разыскать все непересекающиеся вхождения подстрок, удовлетворяющие образцу. В качестве результата возвращается объект MatchCollection, представляющий коллекцию объектов Match.

- метод NextMatch запускает новый поиск.

- метод Split является обобщением метода Split класса String. Он позволяет, используя образец, разделить искомую строку на элементы.

- метод Replace – позволяет делать замену найденного образца.

Содержание работы:

Задание 1. Поиск первого соответствия шаблону

```
string FindMatch(string str, string strpat){  
    Regex pat = new Regex(strpat);  
    Match match = pat.Match(str);  
    string found = "";  
    if (match.Success) {  
        found = match.Value;  
        Console.WriteLine("Строка ={0}\tОбразец={1}\t Найдено={2}",  
            str, strpat, found);    }  
    return(found);    }
```



```

public void TestSinglePat(){
string str, strpat, found;
Console.WriteLine("Поиск по образцу");
//образец задает подстроку, начинающуюся с символа a, далее идут буквы или
цифры.
str="start"; strpat=@"a\w+";
found = FindMatch(str,strpat); //art
str="fab77cd efg";
found = FindMatch(str,strpat); //ab77cd
//образец задает подстроку, начинающуюся с символа a,
//заканчивающуюся f с возможными символами b и d в середине
strpat = "a(b|d)*f"; str = "fabadddbdf";
found = FindMatch(str,strpat); } //adddbdf

```

Задание 2. Поиск всех соответствий шаблону

```

void FindMatches(string str, string strpat) {
Regex pat = new Regex(strpat);
MatchCollection match =pat.Matches(str);
Console.WriteLine("Строка ={0}\tОбразец={1}\t Найдено={2}",
str,strpat,match.Count);      }
Console.WriteLine("око и рококо");
strpat="око"; str = "рококо";
FindMatches(str, strpat); //найдено одно соответствие

```

Индивидуальные задания:

1. В файле с телефонными переговорами клиента определить по какому номеру выполнено максимальное количество звонков.
2. Выполнить анализ кода программы на наличие в нем всех циклов for языка C#. Напечатать результаты анализа на экране монитора.
3. Считать текст из файла и вывести на экран монитора строку, содержащую максимальное количество знаков пунктуации.
4. В файле с телефонными переговорами клиента выполнить сортировку переговоров по их стоимости.
5. Считать текст из файла и вывести его на экран монитора, заменив словами, например, «0» на слово «ноль»; «1» на слово «один» и т. д. на экране монитора.
6. Считать текст из файла и вывести его на экран монитора только цитаты текста, т. е. предложения, заключенные в кавычки.
7. В файле с телефонными переговорами клиента определить, по какому номеру выполнены подряд (2 и более соединений) звонки.

ПРАКТИЧЕСКАЯ РАБОТА № 14

Тема: Операции со списками

Цель работы: формирование целостного представления о назначении списков, овладение практическими навыками работы со списками.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Справочный материал:

Класс `List<T>` из пространства имен `System.Collections.Generic` представляет простейший список однотипных объектов.

Содержание работы:

Задание 1. Реализовать список

```
using System;
using System.Collections.Generic;
namespace Collections {
class Program {
static void Main(string[] args) {
List<int> numbers = new List<int>() { 1, 2, 3, 45 };
numbers.Add(6); // добавление элемента
numbers.AddRange(new int[] { 7, 8, 9 });
numbers.Insert(0, 666); // вставляем на первое место в списке число 666
numbers.RemoveAt(1); // удаляем второй элемент
foreach (int i in numbers) {
Console.WriteLine(i); }
List<Person> people = new List<Person>(3);
people.Add(new Person() { Name = "Том" });
people.Add(new Person() { Name = "Билл" });
foreach (Person p in people) {
Console.WriteLine(p.Name); }
Console.ReadLine(); } }
class Person {
public string Name { get; set; } }
```

Здесь у нас создаются два списка: один для объектов типа `int`, а другой – для объектов `Person`. В первом случае выполняется начальная инициализация списка: `List<int> numbers = new List<int>() { 1, 2, 3, 45 };` Во втором случае используется другой конструктор, в который передается начальная емкость списка: `List<Person> people = new List<Person>(3);`. Указание начальной емкости списка (`capacity`) позволяет в будущем увеличить производительность и уменьшить издержки на выделение памяти при добавлении элементов.

Задание 2. Демонстрационная программа, реализующая операции создания, обработки, просмотра содержимого списка произвольного вида

```
Using System;
public class Node { // описание узла списка
private int info; // информационное поле узла// поле связи узла
private Node link;
public int Info {...}}
```

```

// свойства
public Node Link {...}
public Node (int info) {
// конструкторы
Info = info; }
public Node (int info, Node link)    {
Info = info; Link = link; }      }
public class SingleLinkedList {      // класс «односвязные списки»
private Node first;                  // ссылка на первый узел списка
public Node First {
// свойства
get{ return first; }
set{ first = value; }
public SingleLinkedList()           {      // конструктор по умолчанию
first = null; }                     // создание пустого списка
public SingleLinkedList(int[ ] dates) {
// конструктор с параметрами
// first – ссылка на первый узел списка,
// dates – массив значений информационных полей
first = null;
// создание пустого списка
for (int i = 0; i < dates.Length; i++) {
Node p = new Node( dates[i], first );    // вставка узла в начало списка
first = p; }
public void Print() {                  // просмотр информационных полей узлов списка
Node p = first;
while (p != null) {
Console.WriteLine (p.Info);
p = p.Link; } }
public int Work() {                    // суммирование значений информ. полей узлов списка
Node p = first; int s = 0;
while (p != null) {
s = s + p.Info; p = p.Link; } }
public void Destroy() {                // разрушение списка
first = null; } }
public class Program {
static void Main() {
SingleLinkedList list;
list = new SingleLinkedList(new Int32[] { 1, 2, 3, 4, 5, 6 }); // конструктор
Console.ReadLine();
list.Print(); // просмотр списка
Console.Write ( “Сумма значений инф. полей = “ );
Console.WriteLine ( list.Work() ); // обработка списка
Console.ReadLine();
list.Destroy(); } } // разрушение списка

```

ПРАКТИЧЕСКАЯ РАБОТА № 15

Тема: Операции со списками

Цель работы: формирование целостного представления о списках, овладение практическими навыками работы со списками.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Справочный материал:

Содержание работы:

Задание 1. Использование списковой структуры стека для решения задачи проверки баланса скобок различного вида в тексте.

Будем рассматривать последовательность открывающихся и закрывающихся круглых и квадратных скобок () []. Среди всех таких последовательностей выделим правильные, т.е. те, которые могут быть получены по следующим правилам:

- пустая последовательность правильна;
- если A и B правильны, то и AB правильна;
- если A правильна, то [A] и (A) правильны.

Например, последовательности (), [], [()()] правильны, а последовательности],), ([, ([)] – нет. Закодируем члены последовательности числами: (– 1, [– 2,) – -1,] – -2

Вначале стек пуст. Члены последовательности просматриваем слева направо. Встретив открывающуюся скобку (круглую или квадратную), заносим ее в стек. Встретив закрывающуюся скобку, забираем открывающуюся скобку из вершины стека и проверяем, что в вершине стека находится скобка парная к закрывающейся. Если это не так, можно утверждать, что последовательность неправильна. Последовательность правильна, если в конце стек оказывается пуст.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace BracketControl_Stack {
    public class Node {           // Класс "Элемент стека"
        private int info;
        private Node next;
        public int Info {
            get {
                return info; }
            set {
                info = value; } }
        public Node Next {
            get {
                return next; }
            set {
                next = value; } }
    }
```

```

public Node(int info, Node next) {           // Конструктор
Next = next;
Info = info; }
public class Stack {           // Класс "Стек"
private Node top; // Ссылка на верхушку стека
public Node Top {
get {
return top; }
set {
top = value; } } // Конструктор: создание пустого стека
public Stack() {
top = null; }
// Занесение элемента в стек info - значение, заносимое в верхушку стека
public void Push(int info) {
Node p = new Node(info, top);
top = p; }
// Выталкивание элемента из стека x - значение, извлекаемое из верхушки стека
public int Pop()
{ int x = 0;
if (top != null) {
x = top.Info;
top = top.Next; }
return x; } // Вывод содержимого стека
public void Print() {
Node p = top;
while (p != null) {
Console.Write(p.Info + " ");
p = p.Next; }
Console.WriteLine(); } }
public class Bracket { // Класс "Контроль баланса скобок"
private String input; // Исходная последовательность символов
private int[] coded; // Закодированная последовательность символов
public String Input{
get {
return input; }
set {
input = value; } }
public int[] Coded {
get {
return coded; }
set {
coded = value; } }
public Bracket(String s) { // Конструктор
Input = s; coded = new int[Input.Length]; }
// Кодирование последовательности открывающихся и закрывающихся скобок

```

```

public void Coder()
{
    Console.WriteLine(Input); Console.ReadLine();
    for (int j = 0; j < Input.Length; j++)
    {
        switch (Input[j])
        {
            case '(': { Coded[j] = 1; break; }
            case ')': { Coded[j] = -1; break; }
            case '[': { Coded[j] = 2; break; }
            case ']': { Coded[j] = -2; break; }
            default: break; }; } // Контроль баланса скобок
    public bool Bracket_Control()
    {
        Stack St = new Stack(); // Стек для хранения закодированной послед-ти
        int i = 0; int t;
        Boolean er = false; // Признак соблюдения баланса скобок
        Coder(); // Кодирование исходной последовательности символов
        // Пока последовательность не окончена и ошибка не обнаружена
        while ( (i < Input.Length) && (er == false) )
        {
            // Если обнаружена открывающаяся скобка, занести ее в стек
            if ((Coded[i] == 1) || (Coded[i] == 2)) St.Push(Coded[i]);
            // Если обнаружена закрывающаяся скобка, проверить состояние стека
            Else // Если стек пуст, зафиксировать ошибку
            if (St.Top == null) er = true;
            else { // Иначе - вытолкнуть из стека открывающуюся скобку
                t = St.Pop();
                // Если открывающаяся скобка не соответствует закрывающейся скобке,
                // зафиксировать ошибку
                er = ( t != (-Coded[i]) );
                i++;
            }
            // Ошибка не обнаружена, если все закрывающиеся скобки соответствуют
            // открывающимся скобкам и стек пуст
            er = ( (er == false) && (St.Top == null) );
            return er; } }
    public class Program
    {
        static void Main(string[] args)
        {
            String str = "[()[]()[]"; // Тестовая последовательность символов
            Console.WriteLine(str);
            Bracket B = new Bracket(str); // Объект для контроля последовательности
            символов
            if (B.Bracket_Control()) Console.WriteLine("Баланс скобок соблюдается");
            else Console.WriteLine("Баланс скобок не соблюдается");
            Console.ReadLine(); } } }

```

ПРАКТИЧЕСКАЯ РАБОТА № 16

Тема: Использование основных шаблонов

Цель работы: изучение основных шаблонов и умение их использовать при написании программ

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Справочный материал:

Можно выделить несколько основных отношений: наследование, реализация, ассоциация, композиция и агрегация.

Содержание работы:

Задание 1. Отношение наследования

```
class User{  
public int Id { get; set; }  
public string Name { get; set; } }  
class Manager : User{  
public string Company{ get; set; } }
```

В данном случае используется наследование, а объекты класса Manager также являются и объектами класса User.

Задание 2. Отношение реализации

Реализация предполагает определение интерфейса и его реализация в классах. Например, имеется интерфейс IMovable с методом Move, который реализуется в классе Car:

```
public interface IMovable {  
void Move(); }  
public class Car : IMovable {  
public void Move() {  
Console.WriteLine("Машина едет"); } }
```

Задание 3. Отношение ассоциации

```
class Team { }  
class Player {  
public Team Team { get; set; }}
```

Задание 4. Отношение композиции

```
public class ElectricEngine { }  
public class Car {  
ElectricEngine engine;  
public Car() {  
engine = new ElectricEngine(); } }
```

Задание 5. Отношение агрегации

```
public abstract class Engine { }  
public class Car {  
Engine engine;  
public Car(Engine eng) {  
engine = eng; } }
```

ПРАКТИЧЕСКАЯ РАБОТА № 17

Тема: Использование порождающих шаблонов

Цель работы: изучение порождающих шаблонов и умение их использовать при написании программ

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

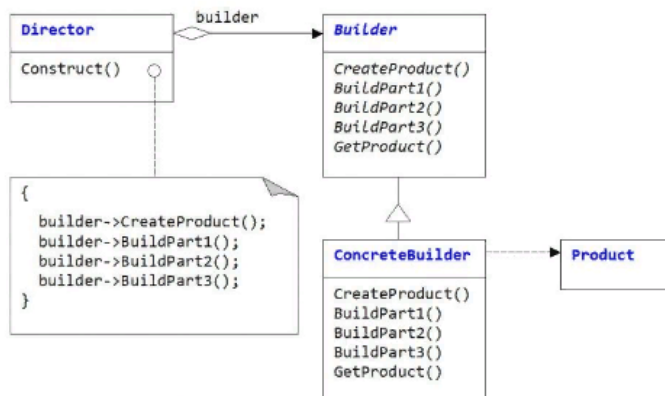
Справочный материал:

Паттерн Builder принадлежит к порождающим паттернам и используется для порождения объектов. Необходимость использования паттерна Builder в программе возникает в случаях, когда нужно добавлять новые возможности без существенного изменения кода. Под возможностями понимаются дополнительные преобразования, которые генерируют конечный продукт (объект). В паттерне Builder представление объекта отделяется от его конструирования (построения). При этом, для конкретной конструкции получаются разные представления.

Паттерн Singleton (одиночка, одиночка) относится к порождающим паттернам. Паттерн Singleton предназначен для создания заданного количества экземпляров (объектов) класса. Чаще всего паттерн Singleton используется для создания гарантированно одного экземпляра класса. Паттерн Singleton важен в случаях, когда для некоторых классов нужно чтобы существовало определенное количество экземпляров, например один экземпляр.

Содержание работы:

Задание 1. Реализация паттерна Builder на языке C#, структура которого изображена на рисунке



Объявляются следующие классы:

- Product — класс, который является продуктом (результатом работы паттерна). Это произвольный класс. В демонстрационных целях в этом классе объявляется три целочисленных переменных, каждая из которых условно считается частью продукта;

- Builder — абстрактный класс, который служит интерфейсом между распорядителем (Director) и классом, создающим конкретный объект (ConcreteBuilder);

- ConcreteBuilder — класс, который строит конкретный объект;

– Director — класс, являющийся распорядителем. Этот класс строит (конфигурирует) объект класса ConcreteBuilder для использования его клиентом.

– Program — класс, который выступает в роли клиента. В этом классе реализуется функция main(), в которой продемонстрирована работа клиента. Клиент обращается к распорядителю (Director) для того, чтобы тот построил объект класса ConcreteBuilder. Затем построенный объект возвращается клиенту в методе GetResult().

Текст программы:

```
using System;
namespace ConsoleApp8
{
    // Паттерн Builder. Пример реализации на C#
    // 1. Класс, служащий продуктом. Содержит три части
    class Product
    {
        // Внутренние переменные
        private int part1, part2, part3;
        // Конструктор
        public Product()
        {
            part1 = 0;    part2 = 0;    part3 = 0;
        }
        // Методы доступа
        public int GetPart1() { return part1; }
        public int GetPart2() { return part2; }
        public int GetPart3() { return part3; }
        public void SetPart1(int part1)
        {
            this.part1 = part1;
        }
        public void SetPart2(int part2)
        {
            this.part2 = part2;
        }
        public void SetPart3(int part3)
        {
            this.part3 = part3;
        }
    }
    // 2. Класс, который служит интерфейсом между распорядителем и
    конкретным строителем
    abstract class Builder
    {
        // Метод, создающий продукт
        public abstract void CreateProduct();
        // Методы, которые строят части продукта
        public abstract void BuildPart1(int part);
        public abstract void BuildPart2(int part);
        public abstract void BuildPart3(int part);
        // Метод, возвращающий продукт клиенту
        public abstract Product GetProduct();
    }
    // Класс - конкретный строитель, наследует абстрактный класс Builder
    class ConcreteBuilder: Builder
    {
        // Внутренняя переменная - ссылка на продукт
        private Product currentBuilder;
```

// Методы, которые объявляются в абстрактном классе Builder, их нужно реализовывать

```
public override void CreateProduct() {  
    currentBuilder = new Product(); }  
public override void BuildPart1(int part) {  
    currentBuilder.SetPart1(part); }  
public override void BuildPart2(int part) {  
    currentBuilder.SetPart2(part); }  
public override void BuildPart3(int part) {  
    currentBuilder.SetPart3(part); }  
public override Product GetProduct() {  
    return currentBuilder; } }
```

// Класс-распорядитель, содержит методы построения объекта Product из частей

```
class Director {  
    // Ссылка на Builder  
    private Builder builder;  
    // Конструктор - инициализируется экземпляром Builder  
    public Director(Builder _builder) {  
        builder = _builder; }  
    // Метод, который строит объект класса Product из частей  
    public void Construct() {  
        // Построить экземпляр класса Product  
        builder.CreateProduct();  
        builder.BuildPart1(10);  
        builder.BuildPart2(20);  
        builder.BuildPart3(30); } }
```

// Класс, содержащий функцию клиента

```
class Program {  
    static void Main(string[] args) {  
        // Это клиент  
        // 1. Объявить ссылку на продукт  
        Product product;  
        // 2. Создать конкретного строителя  
        Builder B = new ConcreteBuilder();  
        // 3. Создать распорядителя и сконфигурировать его строителем  
        Director D = new Director(B);  
        // 4. Вызвать методы построения продукта - доверить это распорядителю  
        D.Construct();  
        // 5. Вернуть построенный продукт клиенту  
        product = B.GetProduct();  
        // 6. Проверить, как построен продукт  
        Console.WriteLine("product.part1 = {0}", product.GetPart1());  
        Console.WriteLine("product.part2 = {0}", product.GetPart2());  
        Console.WriteLine("product.part3 = {0}", product.GetPart3());  
    }  
}
```

```
} } }
```

Результат работы программы:

```
product.part1 = 10  
product.part2 = 20  
product.part3 = 30
```

Задание 2. Используя средства языка C# и возможности паттерна Builder разработать программу для генерирования объектов, являющихся массивами случайных чисел. Массивы случайных чисел представлены следующими классами:

- ArrayRandomInt — массив из целых чисел. Для данного массива указывается диапазон;
- ArrayRandomChar — массив случайных символов от 'A' до 'Z'.

Задание 3. Написать программу на C#, реализующую паттерн Singleton

```
using System;  
using static System.Console;  
namespace ConsoleApp9{  
    // Реализация паттерна Singleton - Одиночка  
    class Singleton {  
        // Статический метод, который возвращает экземпляр класса Singleton.  
        // Данный метод может быть заменен соответствующим свойством.  
        public static Singleton Instance() {  
            if (_instance == null) {  
                _instance = new Singleton();  
                return _instance; }  
            else {  
                return null; } }  
        // Конструктор класса, объявленный как protected, для того чтобы:  
        // - запретить создание экземпляра класса оператором new;  
        // - можно было наследовать данный класс.  
        protected Singleton() { }  
        // Статическая внутренняя переменная, которая хранит экземпляр класса.  
        // К этой переменной есть доступ из методов данного класса.  
        // Из методов других классов доступа к переменной нет.  
        private static Singleton _instance = null;  
        // -----  
        // Другие внутренние поля класса  
        private int d;  
        // Свойство для доступа к полю d  
        public int D {  
            get { return d; }  
            set { d = value; } }  
        // Метод, который выводит значение поля d  
        public void Print(string text) {
```

```

    WriteLine("-----");
    WriteLine("{0}. d = {1}", text, d);  } }
// Класс, выступающий клиентом
class Program {
    static void Main(string[] args) {
        // Это есть код клиента.
        // Создать единственный экземпляр класса Singleton
        Singleton obj1 = Singleton.Instance();
        // Проверить obj1 на равенство null
        if (obj1 != null) {
            obj1.D = 25;
            obj1.Print("obj1");  }
        else
            WriteLine("obj1 == null");
        // Попытка создать другой экземпляр класса
        Singleton obj2 = Singleton.Instance();
        if (obj2 != null) {
            obj2.D = 77;
            obj2.Print("obj2");  }
        else
            WriteLine("obj2 == null");  } } }
    Результат выполнения программы:

```

```

-----
obj1. d = 25
obj2 == null

```

ПРАКТИЧЕСКАЯ РАБОТА № 18

Тема: Использование структурных шаблонов

Цель работы: изучение структурных шаблонов и умение их использовать при написании программ

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

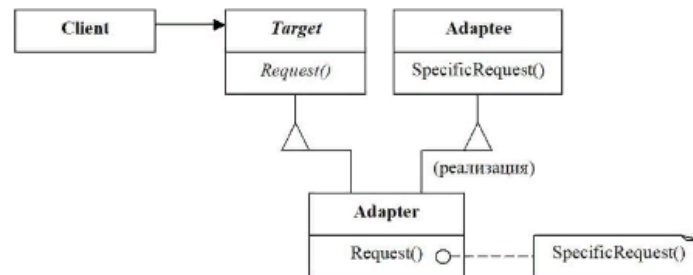
Справочный материал:

Паттерн Adapter принадлежит к структурным паттернам и используется для структурирования классов и объектов. Необходимость использования паттерна Adapter возникает в случаях, когда нужно привести (адаптировать) одну систему к требованиям другой системы. Паттерн Adapter имеет две разновидности, которые отличаются реализацией и полученными результатами:

- адаптер для класса. В этом случае используется механизм наследования;
- адаптер для объекта. В этом случае используется композиция объектов.

Содержание работы:

Задание 1. Реализация паттерна Адаптер для класса, которая содержит такие же имена как изображено на рисунке



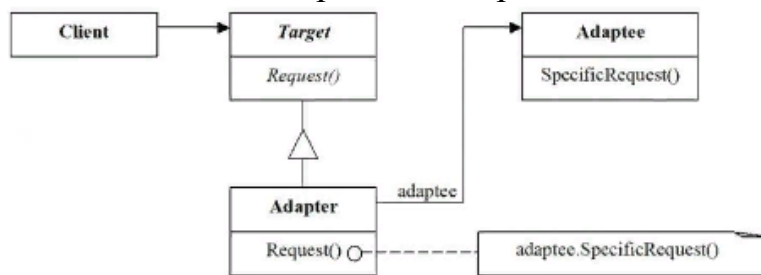
```
using System;
using static System.Console;
namespace ConsoleApp10    {
    // Адаптирует класс Adaptee к интерфейсу ITarget через промежуточный класс
    Adapter
    // Интерфейс ITarget
    interface ITarget    {
        public void Request();    }
    // Класс, метод которого нужно адаптировать к другой системе.
    // В данном случае адаптируется имя метода SpecificRequest() в метод
    Request()
    class Adaptee        {
        // Некоторый специфический метод
        public void SpecificRequest()    {
            WriteLine("Adaptee.SpecificRequest()");    }    }
    // Класс Adapter - реализует интерфейс ITarget и наследует класс Adaptee
    class Adapter : Adaptee, ITarget        {
        // Реализация метода Request() интерфейса ITarget
        public void Request()        {
            // Всередине метода вызывается метод SpecificRequest() класса Adaptee()
```

```

        SpecificRequest();    }    }
// Класс Client - получает ссылку на интерфейс ITarget
class Client {
    // некоторый метод класса Client
    public void ClientMethod(ITarget target) {
        target.Request();    }    }
class Program {
    static void Main(string[] args) {
        // Демонстрация паттерна Adapter для класса.
        // Нужно адаптировать экземпляр класса Adaptee к потребностям
экземпляра класса Client.
        // 1. Создать экземпляр класса Client
        Client client = new Client();
        // 2. Создать экземпляр класса Adapter, который ссылается на интерфейс
ITarget
        ITarget target = new Adapter();
        // 3. Передать клиенту target
        client.ClientMethod(target);    }    }    }

```

Задание 2. Реализацию паттерна Адаптер для объекта из рисунка



```

using System;
using static System.Console;
namespace ConsoleApp10 {
    // Адаптирует класс Adaptee к интерфейсу Target через промежуточный класс
Adapter
    // Интерфейс ITarget
    interface ITarget {
        public void Request();    }
    // Класс, метод которого нужно адаптировать к другой системе.
    // В данном случае адаптируется имя метода SpecificRequest() в метод Request()
    class Adaptee {
        // Некоторый специфический метод
        public void SpecificRequest() {
            WriteLine("Adaptee.SpecificRequest()");    }    }
    // Класс Adapter - реализует интерфейс ITarget
    class Adapter : ITarget {
        private Adaptee adaptee; // ссылка на Adaptee
        // Конструктор
        public Adapter(Adaptee adaptee) {

```

```

    // инициализировать внутреннюю ссылку adaptee параметром
    this.adaptee = adaptee;  }
// Реализация метода Request() интерфейса ITarget
public void Request() {
    // Всередине метода вызывается метод SpecificRequest() класса Adaptee()
    adaptee.SpecificRequest();  } }
// Класс Client - получает ссылку на интерфейс ITarget
class Client {
    // некоторый метод класса Client
    public void ClientMethod(ITarget target) {
        target.Request();  } }
class Program {
    static void Main(string[] args) {
        // Демонстрация паттерна Adapter для объекта.
        // Нужно адаптировать экземпляр класса Adaptee к потребностям
экземпляра класса Client.
        // 1. Пусть заданы некоторые экземпляры классов Client и Adaptee
        Client client = new Client();
        Adaptee adaptee = new Adaptee();
        // 2. Создать экземпляр класса Adapter, передать экземпляр класса Adaptee
        // в конструктор класса Adapter
        ITarget target = new Adapter(adaptee);
        // 3. Передать клиенту target
        client.ClientMethod(target);  } } }

```

ПРАКТИЧЕСКАЯ РАБОТА № 19

Тема: Использование поведенческих шаблонов

Цель работы: изучение поведенческих шаблонов и умение их использовать при написании программ

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Справочный материал:

Паттерн Iterator относится к *паттернам поведения объектов*. Второе название паттерна — Cursor. Паттерн Iterator предназначен для обеспечения доступа к элементам объекта агрегированной структуры таким образом, что не раскрывается внутреннее представление этого объекта.

Суть паттерна Iterator заключается в том, что для каждого объекта агрегированной структуры (массив, список, очередь и т.д.) разрабатывается соответствующий механизм (функционал) обхода элементов этого объекта. Механизм обхода обеспечивается разработкой одного или нескольких классов.

Содержание работы:

Задание 1. На рисунке изображена структура паттерна Iterator с фрагментами кодов на языке C#. Данная структура может быть использована для любого количества контейнеров (полиморфный контейнер) и любого количества итераторов (полиморфный итератор). Для того, чтобы добавить свой специфический контейнер, нужно в классе реализовать интерфейс IAggregate. В этом интерфейсе объявляется единственный метод CreateIterator(). Этот метод возвращает итератор для контейнера.



1. Программирование составляющих, определяющих структуру паттерна Iterator

Для реализации паттерна Iterator нужно разработать следующие интерфейсы и классы:

- интерфейс IAggregate, содержащий объявления метода CreateIterator();
- интерфейс Iterator, содержащий базовые методы получения итератора и движения по контейнеру;
- один или несколько конкретных итераторов. Здесь могут быть объявлены итераторы, осуществляющих различные стратегии обхода. Все итераторы реализуют интерфейс Iterator;

- один или несколько классов, реализующих интерфейс `IAggregate`.

2. Интерфейс контейнера `IAggregate<T>`

Прежде всего объявляется интерфейс контейнера `IAggregate<T>`, содержащий объявления следующих методов:

- `CreateIterator()` — возвращает итератор для контейнера;
- `Count()` — возвращает количество элементов контейнера;
- `GetItem()` — возвращает элемент контейнера по его индексу.

Согласно синтаксису C# все эти методы должны быть обязательно реализованы в классах, реализующих интерфейс `IAggregate<T>`. После ввода интерфейса листинг программы следующий:

```
using System;
namespace ConsoleApp17{
    // Реализация паттерна Iterator на C#.
    // Пример для консольного приложения
    // 1. Объявить интерфейс контейнера с поддержкой обобщенного типа T
    interface IAggregate<T> {
        // Вернуть итератор на контейнер
        Iterator<T> CreateIterator();
        // Количество элементов контейнера
        long Count();
        // Вернуть элемент по его индексу
        T GetItem(long index); }
    class Program {
        static void Main(string[] args) { } }}
```

Интерфейс служит связующим звеном между подклассами, что его реализуют и интерфейсом `Iterator<T>`.

3. Интерфейс итератора `Iterator<T>`

Для обеспечения возможности реализации различных видов итераторов, нужно объявить интерфейс итератора `Iterator<T>`. Этот интерфейс будет содержать общие методы для всех классов итераторов, которые будут его реализовывать. В нашем случае в интерфейс `Iterator<T>` вводятся следующие объявления методов:

- `First()` — перейти на первый элемент контейнера;
- `Next()` — переместить курсор на следующий элемент контейнера;
- `IsDone()` — проверка, конец ли списка;
- `CurrentItem()` — получить текущий элемент по курсору.

По желанию можно расширить список методов для унаследованных итераторов. Например, добавить метод `Previous()` для перехода к предыдущему элементу контейнера.

Сокращенный код программы после ввода интерфейса следующий.

```
using System;
namespace ConsoleApp17{
    // Реализация паттерна Iterator на C#.
    // Пример для консольного приложения
```

```
// 1. Объявить интерфейс контейнера с поддержкой обобщенного типа T
interface IAggregate<T> { ... }
// 2. Объявить интерфейс итератора для типа T
interface IIterator<T> {
    // Переход на первый элемент контейнера
    void First();
    // Переход на следующий элемент контейнера
    void Next();
    // Получить текущий элемент
    T CurrentItem();
    // Проверка, указывает ли курсор на конец контейнера
    bool IsDone(); } ...}
```

4. Конкретный итератор ConcreteIterator<T>

На этом этапе создается один или несколько классов, реализующих интерфейс IIterator<T>. В нашем случае создается один класс с именем ConcreteIterator1<T>. Этот класс реализует прямой итератор.

В конкретном итераторе реализованы следующие программные элементы:

- внутреннее поле aggregate — это ссылка на контейнер, для которого создается данный итератор;
- внутреннее поле current — значение (позиция), которое указывает на один из элементов контейнера. Это курсор;
- конструктор, который получает агрегатный объект в качестве параметра;
- методы First(), Next(), IsDone() и CurrentItem(), которые переопределяют методы интерфейса IIterator<T>.

```
using System;
namespace ConsoleApp17{
    // Реализация паттерна Iterator на C#.
    // Пример для консольного приложения
    // 1. Объявить интерфейс контейнера с поддержкой обобщенного типа T
    interface IAggregate<T> { ... }
    // 2. Объявить интерфейс итератора для типа T
    interface IIterator<T> { ... }
    // 3. Конкретный итератор
    class ConcreteIterator1<T> : IIterator<T> {
        // внутренние данные
        private IAggregate<T> aggregate; // Ссылка на агрегат
        private long current; // текущая позиция
        // Конструктор, получающий агрегированный объект
        public ConcreteIterator1(IAggregate<T> obj) {
            aggregate = obj;
            current = 0; }
        // Перевод курсора в начало списка
        virtual public void First() {
            current = 0; }
```

```
// Перевод курсора на следующий элемент списка
virtual public void Next() {
    current++; }
// Проверка, конец ли списка
virtual public bool IsDone() {
    return current >= aggregate.Count(); }
// Вернуть текущий элемент списка
virtual public T CurrentItem() {
    if (!IsDone())
        return aggregate.GetItem(current);
    else {
        throw new NotImplementedException("Error"); } } ...}
```

5. Класс ConcreteAggregate<T>

Объявленный в п. 2 интерфейс IAggregate<T> является общим для всех классов, реализующих конкретный агрегат (контейнер). Благодаря полиморфизму в программе может быть реализовано любое количество конкретных агрегатов. Как известно, в каждом классе контейнера данные представлены в виде набора элементов (список, массив, дерево и т.д.).

Если создается класс конкретного контейнера (агрегата), то к нему предъявляется ряд требований:

- класс обязательно должен реализовывать (наследовать) интерфейс IAggregate<T>;
- класс должен содержать минимальный функционал для оперирования набором элементов;
- класс должен реализовывать методы интерфейса IAggregate<T>. Обязательным методом является метод CreateIterator().

В нашем случае объявляется класс ConcreteAggregate1<T>, который реализует интерфейс IAggregate<T>. Класс содержит следующие основные составляющие:

- внутреннее поле-массив array обобщенного типа **T**. Вместо массива может использоваться любая другая структура данных: список, дерево, множество и тому подобное;

- два конструктора;
- метод Append() — добавляет новый элемент в конец массива;
- метод Remove() — удаляет элемент из массива по указанной позиции;
- метод Print() — выводит массив.

– методы, объявленные в интерфейсе IAggregate<T>: CreateIterator(), Count(), GetItem().

Сокращенный фрагмент программы с кодом класса ConcreteAggregate1<T> имеет вид.

```
using System;
namespace ConsoleApp17{
    // Реализация паттерна Iterator на C#.
    // Пример для консольного приложения
```

```

// 1. Объявить интерфейс контейнера с поддержкой обобщенного типа T
interface IAggregate<T> { ... }
// 2. Объявить интерфейс итератора для типа T
interface IIterator<T> { ... }
// 3. Конкретный итератор
class ConcreteIterator1<T> : IIterator<T> { ... }
// 4. Конкретный контейнер для элементов типа T
class ConcreteAggregate1<T>:IAggregate<T> {
    // Внутренние поля
    private T[] array; // динамический массив элементов типа T
    // Конструкторы
    // Конструктор, получающий внешний массив
    public ConcreteAggregate1(T[] _array) {
        array = _array; }
    // Конструктор, создающий пустой массив
    public ConcreteAggregate1() {
        array = null; }
    // Метод, добавляющий элемент в конец контейнера
    public void Append(T item) {
        T[] array2 = array;
        array = new T[array2.Length + 1];
        array2.CopyTo(array, 0);
        array[array.Length - 1] = item; }
    // Удаляет элемент из контейнера в позиции index
    public void Remove(long index) {
        T[] array2 = array;
        array = new T[array2.Length - 1];
        for (long i = 0; i < index; i++)
            array[i] = array2[i];
        for (long i = index + 1; i < array2.Length; i++)
            array[i - 1] = array2[i]; }
    // Вывести содержимое контейнера
    public void Print(string text) {
        Console.WriteLine(text + ":");
        for (int i = 0; i < array.Length; i++)
            Console.Write(array[i] + " ");
        Console.WriteLine(); }
    // Реализация методов интерфейса IAggregate<T>
    // Вернуть итератор
    public IIterator<T> CreateIterator() {
        return new ConcreteIterator1<T>(this); }
    // Количество элементов
    public long Count() {
        return array.Length; }
    // Вернуть отдельный элемент

```

```

public T GetItem(long index) {
    if ((index >= 0) && (index < array.Length))
        return array[index];
    else
        throw new NotImplementedException("Error. Bad index"); } }

```

6. Код клиента. Демонстрация паттерна

Ниже приведены клиентский код, демонстрирующий получение итератора и его использование.

```

using System;
namespace ConsoleApp17{
    // Реализация паттерна Iterator на C#.
    // Пример для консольного приложения
    // 1. Объявить интерфейс контейнера с поддержкой обобщенного типа T
    interface IAggregate<T> { ... }
    // 2. Объявить интерфейс итератора для типа T
    interface Iterator<T> { ... }
    // 3. Конкретный итератор
    class ConcreteIterator1<T> : Iterator<T> { ... }
    // 4. Конкретный контейнер для элементов типа T
    class ConcreteAggregate1<T>:IAggregate<T> { ... }
    class Program {
        static void Main(string[] args) {
            // Тестирование паттерна Iterator - код клиента
            // 1. Создать массив чисел
            double[] A = { 2.3, 3.2, 4.4, 5.1, 8.2 };
            // 2. Создать экземпляр контейнера на основе массива A
            ConcreteAggregate1<double> ag1 = new ConcreteAggregate1<double>(A);
            ag1.Print("ag1"); // вывод контейнера
            // 3. Создать итератор для экземпляра ag1
            ConcreteIterator1<double> it1 =
            (ConcreteIterator1<double>)ag1.CreateIterator();
            // 4. Демонстрация работы итератора
            Console.WriteLine("-----");
            Console.WriteLine("Access using iterator:");
            it1.First();
            while (!it1.IsDone()) {
                Console.Write("{0} ", (double)it1.CurrentItem());
                it1.Next(); }
            Console.WriteLine();
            Console.ReadKey(); } }

```

7. После запуска, программа выдает следующий результат ag1:

2.3 3.2 4.4 5.1 8.2

Access using iterator:

2.3 3.2 4.4 5.1 8.2

ПРАКТИЧЕСКАЯ РАБОТА № 20

Тема: Разработка приложения с использованием текстовых компонентов

Цель работы: сформировать умения по использованию компонентов для работы с текстом в среде программирования Visual Studio, сформировать умения по созданию приложений с компонентами для работы с текстом в среде программирования Visual Studio.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Справочный материал:

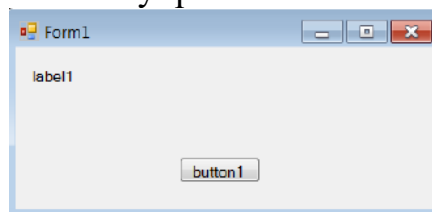
Компоненты ввода — вывода данных можно условно разделить на несколько различных блоков: компоненты вывода текстовой информации на экран; однострочные поля ввода текстовой и числовой информации; многострочные поля ввода.

Для вывода определенной информации на экран, кроме уже ранее используемого компонента label, есть и другие компоненты. Текст, который будет отображен, можно задавать как на этапе разработки формы, так и в процессе выполнения программы, присвоив значение свойству Text.

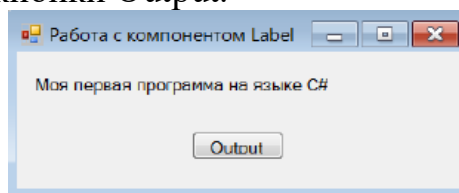
Содержание работы:

Задание 1. Разработать программу, которая при нажатии на кнопку «Output» выводит сообщение «Моя первая программа на языке C#», а затем при повторном нажатии на эту же кнопку сообщение исчезает. При повторном выводе цвет надписи должен быть красным.

1. Запустите среду программирования Visual Studio. Создайте новое Приложение Windows Forms. Имя проекта и приложения Label. Папка для размещения проекта Текст.
2. Разместите на форме компонент label и кнопку button вкладки панели элементов Стандартные элементы управления.



3. Задайте для формы заголовок «Работа с компонентом Label» на панели Свойств свойство Text.
4. Выделите надпись label1, найдите на панели Свойства свойство Text и вставьте новое название надписи Моя первая программа на языке C#.
5. Выделите кнопку button1, найдите на панели Свойства свойство Text и вставьте новое название кнопки Output.



6. Перейдите на панели Свойства на страницу События, найдите событие

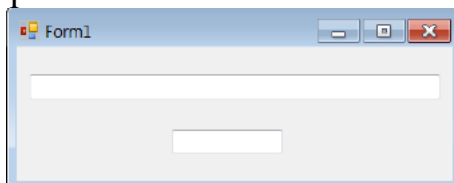
Click и справа от него дважды щелкните мышкой. Оказавшись в коде программы, но теперь в процедуре кнопки Button1, напишите следующий программный код:

```
private void button1_Click(object sender, EventArgs e)
{
    label1.Visible = !label1.Visible;
    if (label1.Visible)
    {
        label1.ForeColor = System.Drawing.Color.Red;
    }
}
```

7. Сохраните изменения и запустите проект. Протестируйте его работу

Задание 2. Разработать программу, которая при вводе текста в первый компонент textBox1, во втором компоненте textBox 2 отображает реальную длину вводимой строки. Кроме этого, при выходе из компонента textBox 1 его содержимое копируется в буфер обмена и удаляется, а при возвращении в программу появляется снова.

1. Создайте новое Приложение Windows Forms. Имя проекта и приложения textBox1. Папка для размещения проекта Текст.
2. Разместите на форме два компонент textBox вкладки панели элементов Стандартные элементы управления.



3. Задайте для формы заголовок «Работа с компонентом textBox».
4. Выделите текстовое поле textBox1, найдите на панели Свойства свойство Text и оставьте его пустым. Аналогичные действия выполните со вторым текстовым полем.
5. Выделите компонент textBox1, на панели Свойства перейдите на вкладку События и найдите событие TextChanged и справа от него дважды щелкните мышкой. Оказавшись в коде программы, но теперь в процедуре текстового поля textBox1, напишите следующий программный код:

```
private void textBox1_TextChanged(object sender, EventArgs e)
{
    textBox2.Text = Convert.ToString(textBox1.Text.Length);
}
```

6. Выделите компонент textBox1, на панели Свойства перейдите на вкладку События и найдите событие Enter, справа от него дважды щелкните мышкой. Оказавшись в коде программы, но теперь в процедуре текстового поля textBox1, напишите следующий программный код:

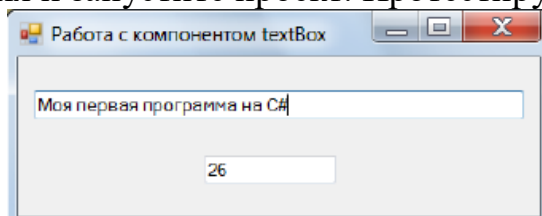
```
private void textBox1_Enter(object sender, EventArgs e)
{
    textBox1.Paste();
}
```

7. Выделите компонент Edit1, на панели Свойства перейдите на вкладку События и найдите событие Leave и справа от него дважды щелкните мышкой. Оказавшись в коде программы, но теперь в процедуре текстового поля

Edit1, напишите следующий программный код:

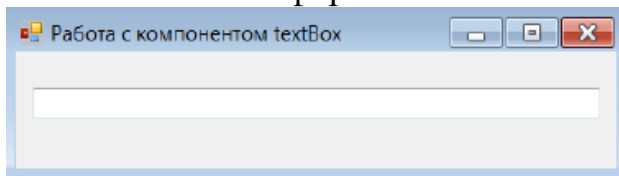
```
private void textBox1_Leave(object sender, EventArgs e)
{
    textBox1.SelectAll();
    textBox1.Copy();
    textBox1.Clear();
}
```

8. Сохраните изменения и запустите проект. Протестируйте его работу



Задание 3. Разработать программу, которая запрещает ввод в компонент textBox1 подряд двух одинаковых символов.

1. Создайте новое Приложение Windows Forms. Имя проекта и приложения textBox2. Папка для размещения проекта Текст.
2. Разместите на форме компонент textBox1.



3. Перейдите в код программы. Объявите глобальную переменную ch типа char, в которой будет храниться последний нажатый символ.

```
protected char ch;
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
```

4. Задайте для формы заголовок «Работа с компонентом textBox».

5. Выделите текстовое поле textBox1, найдите на панели Свойства свойство Text и оставьте его пустым.

6. Создайте процедуру обработки события KeyPress текстового поля textBox1, параметр Key данной процедуры содержит символ нажатой клавиши. Если вновь введенный символ совпадает с только что нажатым символом, то он игнорируется. В противном случае, новый символ запоминается в переменной ch.

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    if (ch==e.KeyChar)
    {
        e.KeyChar = Convert.ToChar(0);
    }
    else
    {
        ch = e.KeyChar;
    }
}
```

7. Сохраните изменения и запустите проект. Протестируйте его работу.

Задание 4. Разработать программу, которая при нажатии на кнопку заполняет текстовое поле фразой «Специальность Информационные системы и программирование»

ПРАКТИЧЕСКАЯ РАБОТА № 21

Тема: Разработка приложения с использованием текстовых компонентов

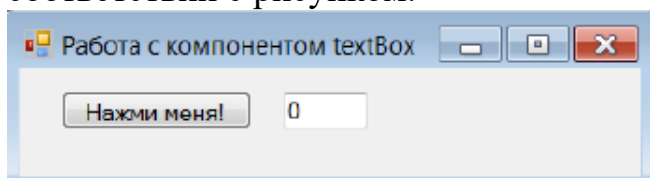
Цель работы: сформировать умения по использованию компонентов для работы с текстом в среде программирования Visual Studio, сформировать умения по созданию приложений с компонентами для работы с текстом в среде программирования Visual Studio.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Содержание работы:

Задание 1. Разработать программу, которая считает количество нажатий на кнопку и выдает это значение в компоненте textBox.

1. Создайте новое Приложение Windows Forms. Имя проекта и приложения textBox3. Папка для размещения проекта Текст.
2. Разместите на форме компонент textBox и кнопку button.
3. Используя панель Свойства, задайте значения свойств компонентов формы в соответствии с рисунком.



4. Если в целочисленной переменной *i* будем считать количество нажатий, то процедура обработки события Click кнопки может быть записана в виде:

```
private void button1_Click(object sender, EventArgs e)
{
    i = i + 1;
    textBox1.Text = Convert.ToString(i);
}
```

5. Однако остается вопрос, где описывать данную переменную *i*. Если сделать это внутри данной процедуры, то также необходимо осуществлять обнуление переменной, а это приведет к получению одного и того же результата, равного единице. Следовательно, переменная *i* должна быть глобальной переменной. Перейдите в код программы опишите глобальную переменную *i* типа `int`, в которой будет храниться последний нажатый символ.

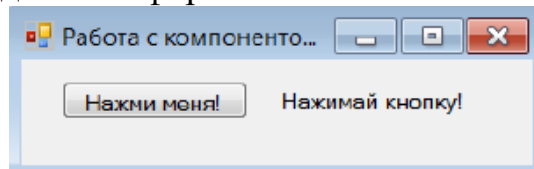
```
protected int i;
private void button1_Click(object sender, EventArgs e)
```

8. Сохраните изменения и запустите проект. Протестируйте его работу.
9. Данную программу можно легко модифицировать так, чтобы после определенного количества нажатий появлялось некоторое сообщение или кнопка блокировалась, или приложение автоматически закрывалось. Результат можно отображать не только посредством компонента textBox, но и через не редактируемый текст, т. е. компонент label, что в данном случае является более естественным.

Свойству Visible компонента label присваиваем False, т. е. при открытии формы надпись отражаться не будет. Затем, как и ранее, при нажатии на кнопку переменная *i* увеличивается на 1. Когда значение переменной *i* будет равно 10, 20, 30 или 40 компонент label становится

видимым, а свойству Text надписи присваиваем значение «Вы нажали i раз». При следующем нажатии надпись становится невидима. Когда i станет равной 50, кнопку необходимо сделать неактивной, для чего необходимо изменить значение свойства Enabled с True — включено на False — выключено.

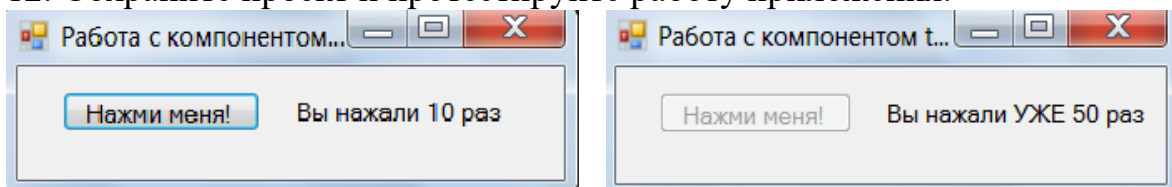
10. Разместите на форме компонент label. Задайте свойство Text равным «Нажимай кнопку!». Удалите с формы компонент textBox1.



11. Перейдите в окно редактирования кода и внесите изменения в код процедуры обработки события Click кнопки.

```
private void button1_Click(object sender, EventArgs e)
{
    label1.Visible = false;
    i = i + 1;
    if ((i==10)||(i==20)||(i==30)||(i==40))
    {
        label1.Visible = true;
        label1.Text = "Вы нажали " + Convert.ToString(i) + " раз";
    }
    if (i==50)
    {
        label1.Visible = true;
        label1.Text = "Вы нажали УЖЕ " + Convert.ToString(i) + " раз";
        button1.Enabled = false;
    }
}
```

12. Сохраните проект и протестируйте работу приложения.

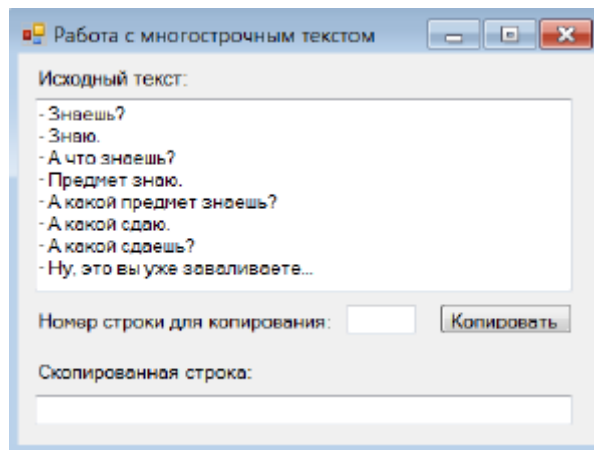


Задание 2. Разработать программу, которая считывает строку под определенным номером и помещает её в текстовое поле.

1. Создайте новое Приложение Windows Forms. Имя проекта и приложения textBox4. Папка для размещения проекта Текст.

2. Для ввода или вывода многострочного текста, воспользуемся компонентом textBox. строк. Для возможности доступа к строкам вместо свойства Text у него имеется свойство Lines, при выборе которого во время проектирования задается начальное значение строк. Для доступа к строкам во время выполнения программы также используется свойство Lines. Разместите на форме компоненты textBox, компоненты label, кнопку button.

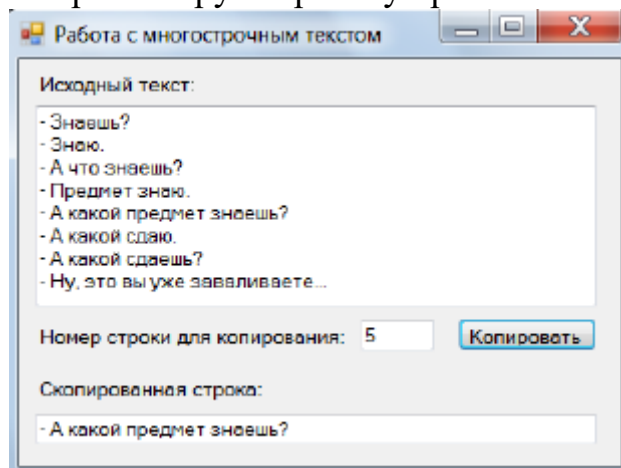
3. Используя панель Свойства, задайте значения свойств компонентов формы в соответствии с рисунком.



4. Для реализации решения задачи процедура обработки события Click кнопки может быть записана в виде:

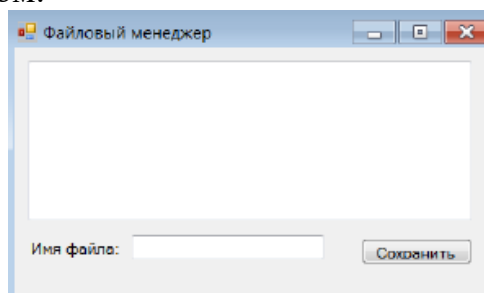
```
private void button1_Click(object sender, EventArgs e)
{
    textBox3.Text = textBox1.Lines[Convert.ToInt32(textBox2.Text) - 1];
}
```

5. Сохраните проект и протестируйте работу приложения.



Задание 3. Разработать программу, которая сохраняет текст, набранный в поле `textBox1` в файл, имя которого задано в текстовом поле `textBox2`.

1. Создайте новое Приложение WindowsForms. Имя проекта и приложения `textBox5`. Папка для размещения проекта Текст.
2. Разместите на форме компоненты `textBox`, компонент `ILabel`, кнопку `button`.
3. Используя панель Свойства, задайте значения свойств компонентов формы в соответствии с рисунком.



4. Поскольку мы будем работать с файлами, то необходимо выполнить подключение пространства имен для работы с файлами:

```
using System.Windows.Forms;
using System.IO;
```

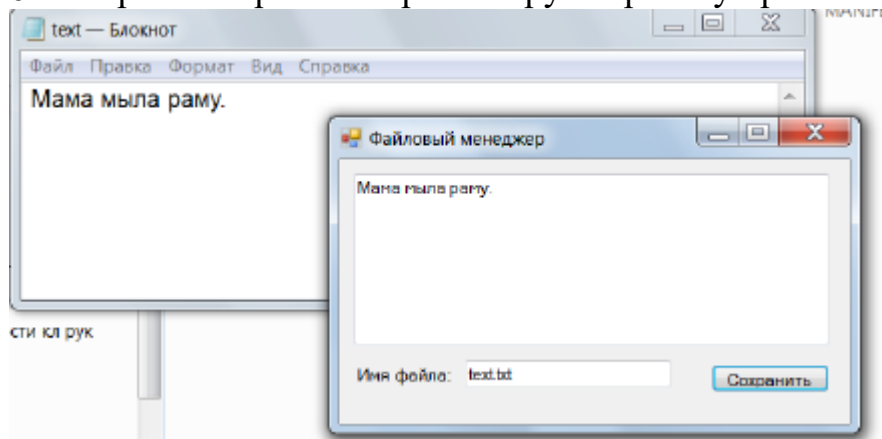
```
namespace textBox4
{
```

5. Процедура обработки события Click кнопки button будет иметь следующий вид:

```
private void button1_Click(object sender, EventArgs e)
{
    StreamWriter write_text;
    FileInfo file = new FileInfo(textBox3.Text);
    write_text = file.AppendText();
    write_text.WriteLine(textBox1.Text);
    write_text.Close();
}
```

Первой строкой в процедуре определяется класс для записи в файл. Далее запись информации в файл с именем заданным в textBox2. Если такого файла не существует, то он будет создан на диске.

6. Сохраните проект и протестируйте работу приложения.



ПРАКТИЧЕСКАЯ РАБОТА № 22

Тема: Разработка приложения с использованием текстовых компонентов

Цель работы: сформировать умения по использованию компонентов для работы с текстом в среде программирования Visual Studio, сформировать умения по созданию приложений с компонентами для работы с текстом в среде программирования Visual Studio.

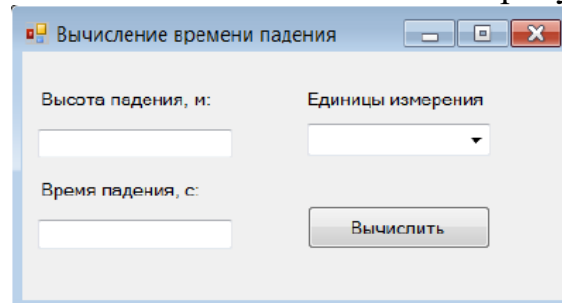
Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.


Содержание работы:

Задание 1. Разработать приложение, с помощью которого можно вычислить время падения тела с некоторой высоты при условии, что высота может задаваться в метрах, сантиметрах и дюймах.

1.Создайте новое Приложение Windows Forms. Имя проекта и приложения Zad1. Папка для размещения проекта Контейнер Элементы.

2.Разместите на форме компоненты в соответствии с рисунком

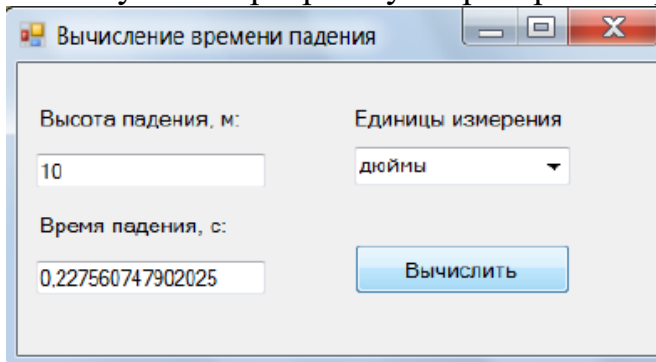


9. Выделите элемент управления comboBox1 на панели Свойства найдите свойство Items, щелкните по кнопке  и в диалоговом окне Редактор коллекции строк введите строки с единицами измерения: метры, сантиметры, дюймы.

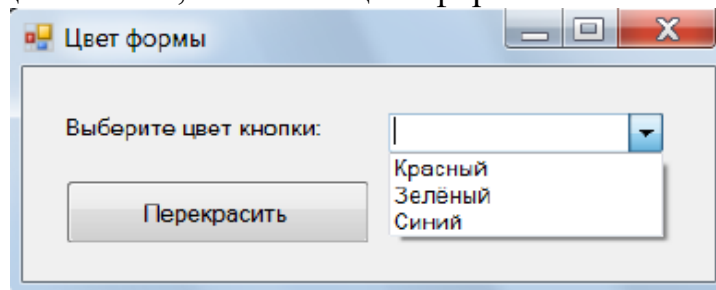
10. Создайте обработчик события Click кнопки Вычислить и вставьте следующий код:

```
private void button1_Click(object sender, EventArgs e)
{
    double g = 9.81;
    double h;
    double t;
    h = Convert.ToDouble(textBox1.Text);
    if (comboBox1.Text=="метры")
    {
        t = Math.Sqrt(2 * h / g);
        textBox2.Text = Convert.ToString(t);
    }
    if (comboBox1.Text == "сантиметры")
    {
        t = Math.Sqrt(2 * h /100/ g);
        textBox2.Text = Convert.ToString(t);
    }
    if (comboBox1.Text == "дюймы")
    {
        t = Math.Sqrt(2 * h *2.54/100/ g);
        textBox2.Text = Convert.ToString(t);
    }
}
```

11. Запустите программу и проверьте её работоспособность.



Задание 2. Разработайте приложение, которое при выборе определенного цвета в выпадающем списке, изменяет цвет формы.



Задание 3. Разработать приложение «Конвертер валют» - перевод из одной валюты в другую.

ПРАКТИЧЕСКАЯ РАБОТА № 23

Тема: Разработка приложения с несколькими формами

Цель работы: сформировать умения использования контейнерных элементов управления при создании проекта Visual Studio, изучить их основные свойства; сформировать умения по созданию процедур на основе событий компонентов.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

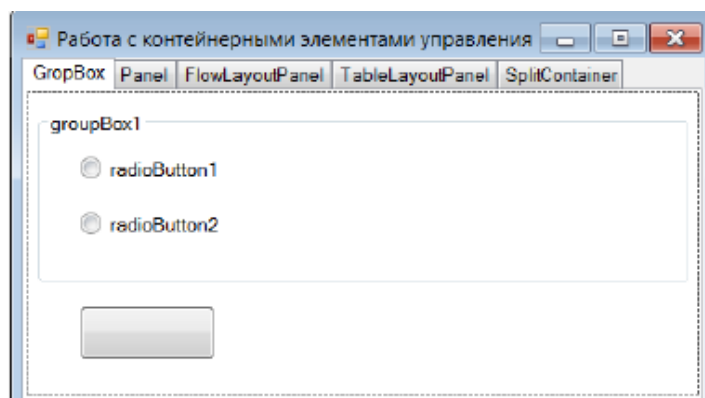
Справочный материал:

Контейнерные элементы управления – это специализированные элементы управления, выступающие в роли настраиваемого вместилища для других элементов управления. К контейнерным элементам управления относятся Panel и GroupBox. Они представляют форме логические и физические подразделы, которые могут группировать другие элементы управления в единообразные подгруппы пользовательского интерфейса.

Содержание работы:

Задание 1. Создать проект с возможностью группировки элементов на вкладках.

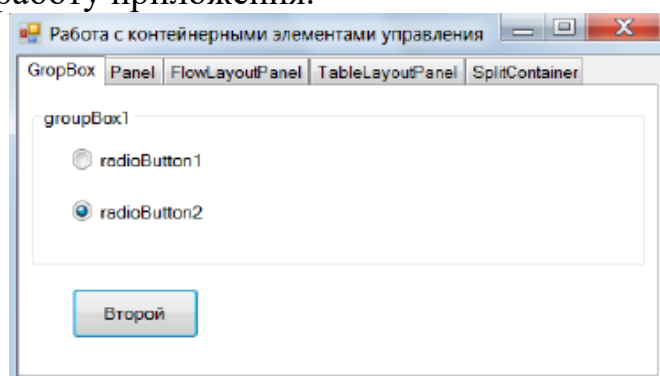
1. Запустите среду программирования Visual Studio. Создайте новое Приложение Windows Forms. Имя проекта и приложения Zad1. Папка для размещения проекта Контейнер Элементы.
2. Перетащите на форму элемент управления TabControl с вкладки Контейнеры панели элементов. На панели Свойства задайте свойству Dock значение Fill.
3. На панели Свойства выберите свойство TabPages, чтобы открыть Редактор Коллекции TabPage. Добавьте вкладки так, чтобы их стало всего пять. Задайте свойствам Text этих пяти элементов управления TabPage значения GropBox, Panel, FlowLayoutPanel, TableLayoutPanel и SplitContainer. Щелкните ОК.
4. В форме выберите вкладку GroupBox. Перетащите элемент управления groupBox с вкладки Контейнеры панели элементов в элемент управления TabPage.
5. Перетащите в GroupBox два элемента управления radioButton с вкладки Стандартные элементы управления панели элементов.
6. Добавьте на вкладку GroupBox вне элемента управления groupBox кнопку button. Свойство Text кнопки задайте пустым, а свойству Name задайте значение but.



7. Дважды щелкните мышью по кнопке и добавьте код обработчика события Click установки надписи на кнопке в зависимости от выбранного переключателя:

```
private void button1_Click(object sender, EventArgs e)
{
    if (radioButton1.Checked==true)
    {
        this.but.Text = "Первый";
    }
    else
    {
        if (radioButton2.Checked==true)
        {
            this.but.Text = "Второй";
        }
    }
}
```

8. Протестируйте работу приложения.



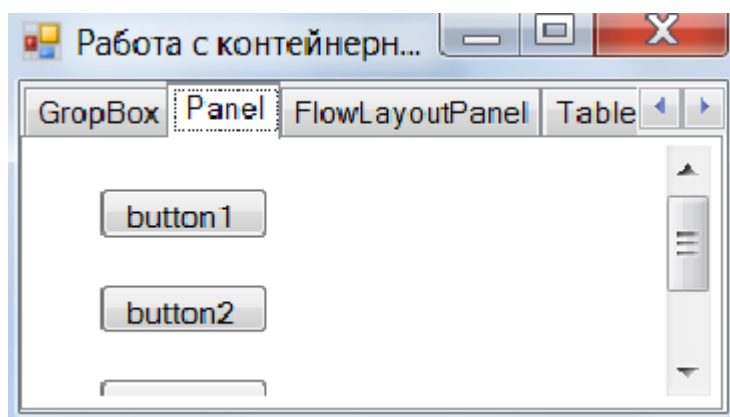
9. Выберите на форме вкладку Panel. Перетащите элемент управления Panel с вкладки Контейнеры панели элементов в элемент управления TabPage. Для элемента Panel задайте свойству Dock значение Fill.

10. Перетащите в элемент управления Panel четыре элемента управления button с вкладки Стандартные элементы управления панели элементов.

11. Выделите элемент Panel на панели Свойства найдите свойство AutoScroll и установите значение True. в этом случае элемент управления Panel будет отображать полосы прокрутки, если элементы находятся за пределами видимых границ.

12. Протестируйте работу приложения.

13. Выберите на форме вкладку FlowLayoutPanel. Перетащите элемент управления FlowLayoutPanel с вкладки Контейнеры панели элементов в элемент управления TabPage. Для элемента FlowLayoutPanel задайте свойству Dock значение Fill.



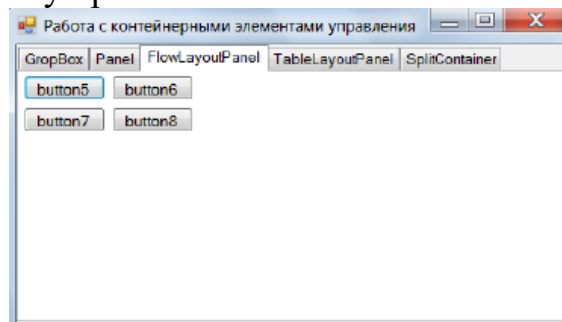
14. Перетащите в элемент управления FlowLayoutPanel четыре элемента управления button с вкладки Стандартные элементы управления панели эле

ментов. Обратите внимание на размещение добавляемых элементов: по умолчанию порядок следования элементов управления в `FlowLayoutPanel` – слева направо. Это значит, что элементы управления, расположенные в `FlowLayoutPanel`, будут находиться в левом верхнем углу и размещаться вправо до тех пор, пока не достигнут края панели. Такое поведение контролируется свойством `FlowDirection`, которому может быть задано четыре значения заполнения в `FlowLayoutPanel`: `LeftToRight` – по умолчанию, `RightToLeft` – справа налево, `TopDown` – сверху вниз, `BottomUp` – снизу вверх.

15. Дважды щелкните кнопку `button5` и добавьте в обработчик события `Click` следующий код:

```
private void button5_Click(object sender, EventArgs e)
{
    flowLayoutPanel1.SetFlowBreak(button6, true);
}
```

16. Протестируйте работу приложения



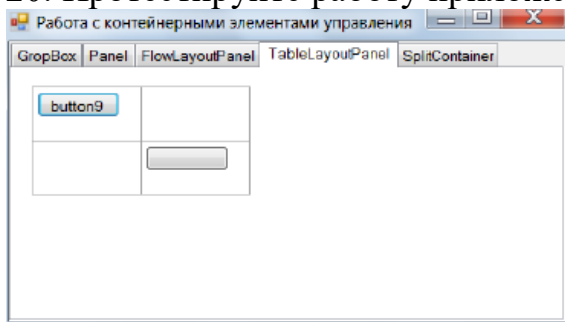
17. Выберите на форме вкладку `TableLayoutPanel`. Перетащите элемент управления `TableLayoutPanel`с вкладки Контейнеры панели элементов в элемент управления `TabPage`. Задайте свойству `CellBorderStyle` значение `Inset`, а свойству `AutoScroll` значение `True`.

18. Перетащите в левую верхнюю ячейку элемента управления `TableLayoutPanel` элемента управления `button` с вкладки Стандартные элементы управления панели элементов.

19. Дважды щелкните `button9` и добавьте в обработчик события `Click` следующий код:

```
private void button9_Click(object sender, EventArgs e)
{
    Button aButton = new Button();
    tableLayoutPanel1.Controls.Add(aButton, 1, 1);
}
```

20. Протестируйте работу приложения.



21. Выберите на форме вкладку SplitContainer. Перетащите элемент управления SplitContainerc вкладки Контейнеры панели элементов в элемент управления TabPage. Задайте свойству BorderStyle значение Fixed3D.

22. Перетащите два элемента управления button с вкладки Стандартные элементы управления панели элементов в Panel1. Задайте свойствам Text этих кнопок значения Fix/UnfixPanel1 и Fix/UnfixSplitter. Измените размеры кнопок так, чтобы отображался текст.

23. Добавьте кнопку в Panel2 и задайте свойству Text значение Collapse/UncollapsePanel1. Измените размеры кнопки так, чтобы отображался текст.

24. Дважды щелкните кнопку Fix/UnfixPanel1 и добавьте в обработчик события Click следующий код:

```
private void button10_Click(object sender, EventArgs e)
{
    if (splitContainer1.FixedPanel==FixedPanel.Panel1)
    {
        splitContainer1.FixedPanel = FixedPanel.None;
    }
    else
    {
        splitContainer1.FixedPanel = FixedPanel.Panel1;
    }
}
```

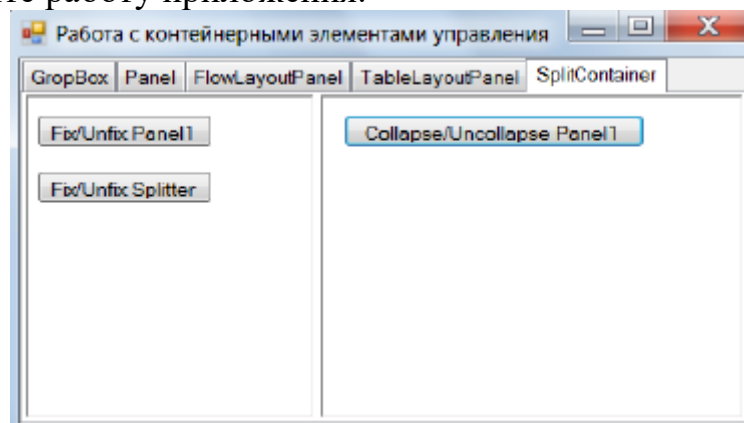
25. Дважды щелкните кнопку Fix/UnfixSplitter и добавьте в обработчик события Click следующий код:

```
private void button11_Click(object sender, EventArgs e)
{
    splitContainer1.IsSplitterFixed = !(splitContainer1.IsSplitterFixed);
}
```

26. Дважды щелкните кнопку Collapse/UncollapsePanel1 и добавьте в обработчик события Click следующий код:

```
private void button12_Click(object sender, EventArgs e)
{
    splitContainer1.Panel1Collapsed = !(splitContainer1.Panel1Collapsed);
}
```

27. Протестируйте работу приложения.



ПРАКТИЧЕСКАЯ РАБОТА № 24

Тема: Разработка приложения с несколькими формами

Цель работы: сформировать практические умения создания многооконного приложения средствами среды программирования Visual Studio

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Содержание работы:

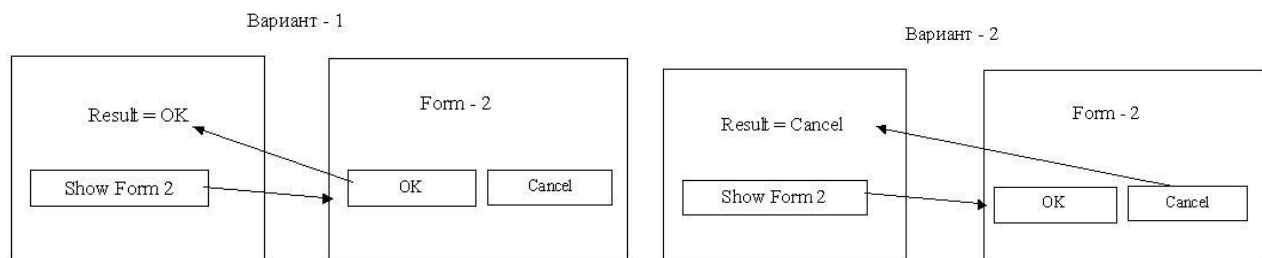
Задание 1. Разработать демонстрационное приложение, осуществляющее вызов из главной формы второстепенной формы по схеме, изображенной на рис. 1. Приложение реализует взаимодействие между различными формами, которыми могут быть диалоговые окна любой сложности.

В главной форме Form1 разместить:

- элемент управления типа Label для вывода результата возврата из второстепенной формы;
- элемент управления типа Button для вызова второстепенной формы.

Во второстепенной форме Form2 разместить:

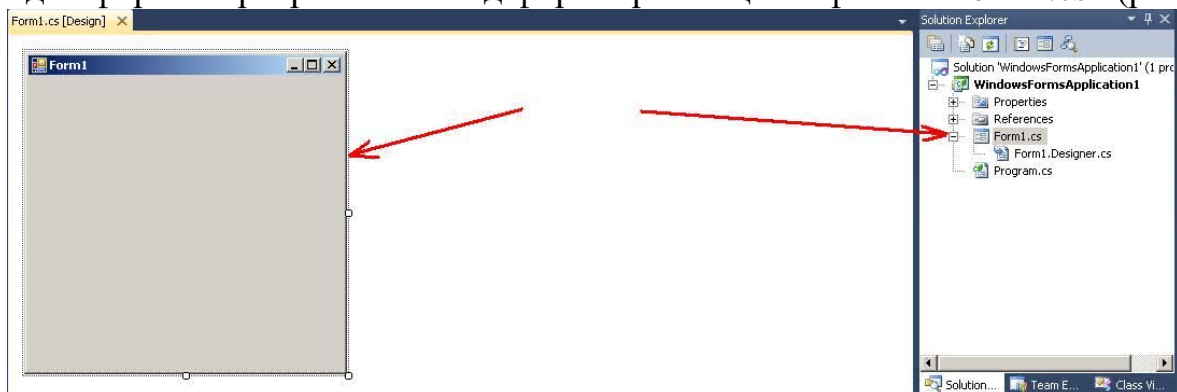
- элемент управления типа Label для вывода заголовка формы;
- два элемента управления типа Button для обеспечения подтверждения или неподтверждения выбора (действия) во второстепенной форме.



1. Создать приложение типа Windows Forms Application

1.1. Запустить Visual Studio.

1.2. Сохранить проект в своей папке. После создания приложения у нас есть одна форма. Программный код формы размещен в файле «Form1.cs» (рис. 2).



2. Разработка главной формы приложения

Из палитры элементов управления Toolbox выносим на форму:

- элемент управления типа Button;
- элемент управления типа Label.

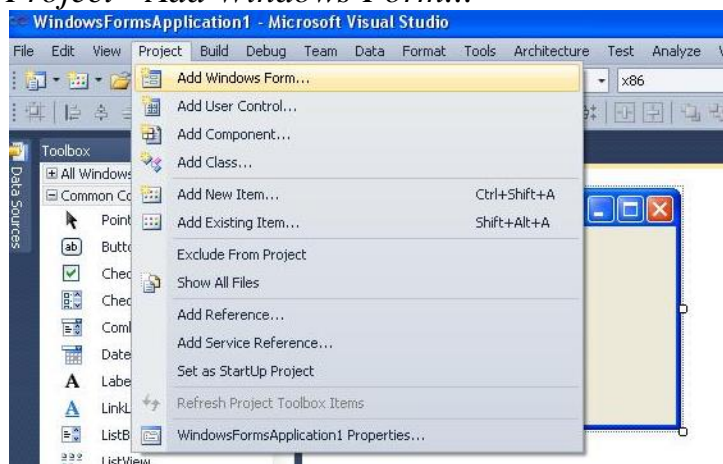
Автоматически создаются два объекта-переменные с именами `button1` и `label1`. В элементе управления типа `Button` свойство `Text` установить в значение «Show Form 2». В элементе управления типа `Label` свойство `Text` установить в значение «Result =». После внесенных изменений главная форма `Form1` приложения будет иметь вид



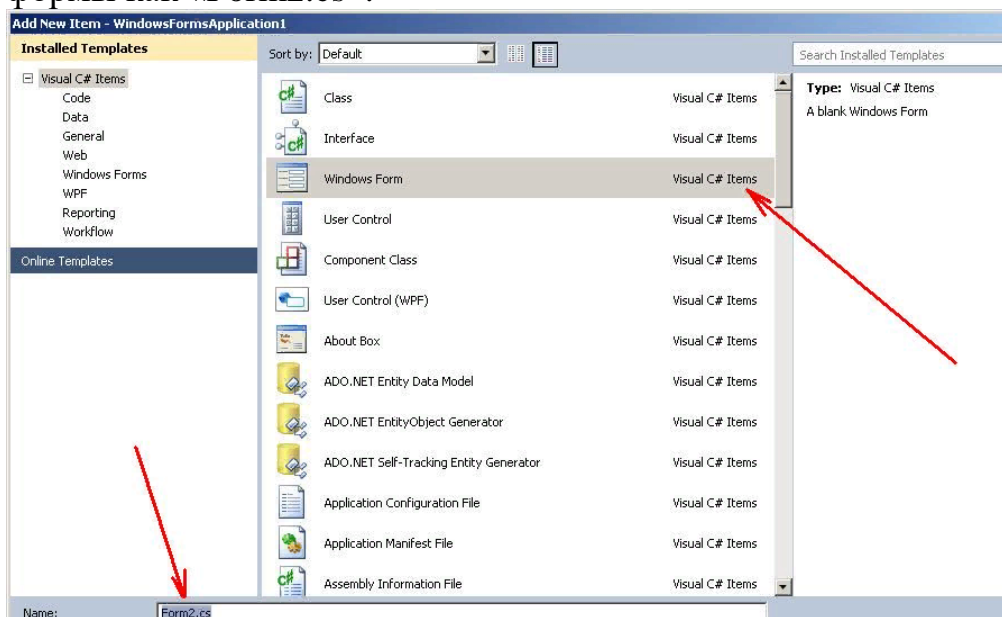
3. Создание второстепенной формы

Для создания второстепенной формы в `C#` можно воспользоваться несколькими способами.

Способ 1. Для добавления формы №2 в проект этим способом нужно вызвать команду *Project - Add Windows Form...*



В результате откроется окно «Add New Item — Windows Forms Application1». В этом окне выбираем элемент «Windows Form». Оставляем имя формы как «Form2.cs».

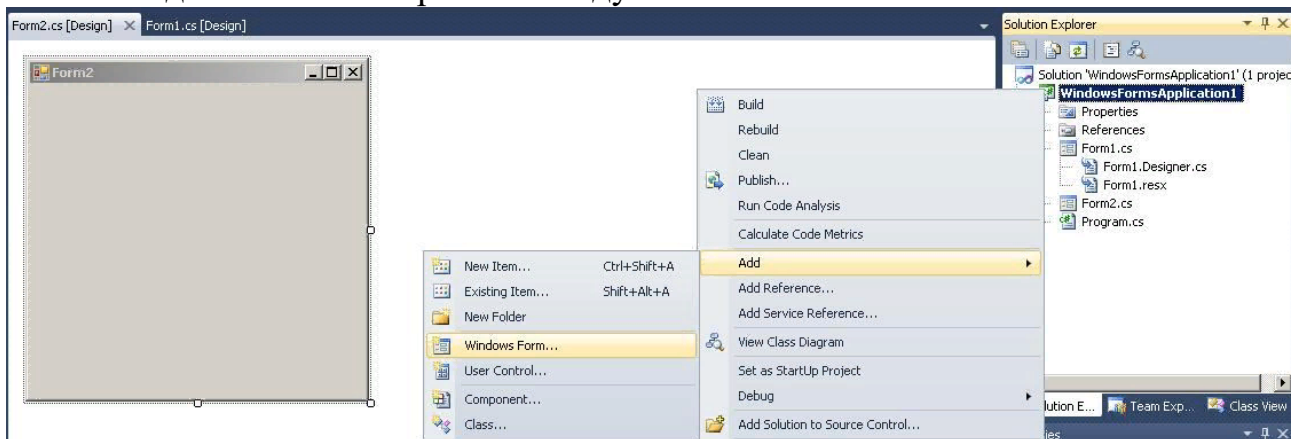


После нажатия на кнопке «Add» новая форма будет добавлена к проекту



Способ 2. Также новую форму можно добавить к проекту с помощью соответствующей команды из контекстного меню. Последовательность действий следующая:

- в Solution Explorer сделать клик правой кнопкой «мышки» на названии приложения WindowsFormsApplication1;
- выбрать подменю Add;
- в подменю Add выбрать команду «Windows Form...».



4. Разработка второстепенной формы

Следующим шагом есть разработка второстепенной формы. Используя средства панели инструментов Toolbox создаем второстепенную форму Form2. Такое построение формы соответствует условию задачи. Таким же образом, на Form2 имеем элементы управления label1, button1, button2.



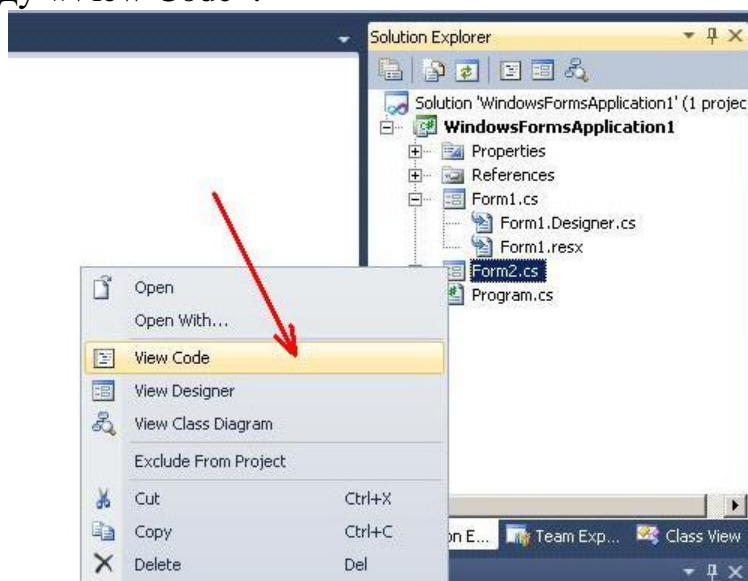
5. Программирование событий клика на кнопках OK и Cancel формы Form2

Программируем событие клика на кнопке OK. В программный код обработчика события button1_Click() (кнопка «OK») вписываем следующую строку: `this.DialogResult = DialogResult.OK;` это значит, что результат возврата из формы Form2 есть «OK». Точно так же в обработчике события button2_Click вписываем: `this.DialogResult = DialogResult.Cancel;` это значит выбор кнопки «Cancel» (button2).

После внесенных изменений листинг программного кода файла «Form2.cs» будет иметь следующий вид:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication1 {
    public partial class Form2: Form    {
        public Form2()    {
            InitializeComponent();    }
        private void button1_Click(object sender, EventArgs e)    {
            this.DialogResult = DialogResult.OK;    }
        private void button2_Click(object sender, EventArgs e)    {
            this.DialogResult = DialogResult.Cancel;    }    }
    }
```

Переход к программному коду формы Form2 (файл «Form2.cs») можно осуществить с помощью Solution Explorer. Для этого в Solution Explorer вызываем контекстное меню для формы Form2 и из этого меню выбираем команду «View Code».



6. Вызов формы Form2 из главной формы приложения

Согласно с условием задачи, для вызова Form2 из Form1 нужно запрограммировать событие клика на кнопке «Show Form 2». Программный код обработчика события будет иметь следующий вид:

```
...
private void button1_Click(object sender, EventArgs e)    {
    Form2 f = new Form2(); // создаем объект типа Form2
    if (f.ShowDialog() == DialogResult.OK) // вызов диалогового окна формы
        Form2    {
```



```

        label1.Text = "Result = OK!";    }
    else {
        label1.Text = "Result = Cancel!";    }    }    ...

```

В листинге, приведенном выше, сначала создается экземпляр класса типа Form2. В операторе условного перехода if осуществляется вызов диалогового окна формы Form2 с помощью строки *f.ShowDialog();*

Функция ShowDialog() выводит окно формы и держит его открытым до тех пор, пока пользователь не сделает какой-либо выбор. После выбора пользователем той или иной команды, окно закрывается с кодом возврата. Происходит проверка кода возврата с известными константами класса DialogResult. После проверки выводится сообщение о действии, выбранном пользователем в Form2 (элемент управления label2).

Листинг всего программного кода формы Form1 следующий:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication1 {
    public partial class Form1: Form {
        public Form1() {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e) {
            Form2 f = new Form2();
            if (f.ShowDialog() == DialogResult.OK) {
                label1.Text = "Result = OK!";    }
            else {
                label1.Text = "Result = Cancel!";    }    }    }

```

ПРАКТИЧЕСКАЯ РАБОТА № 25

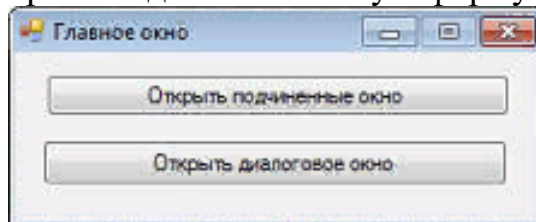
Тема: Разработка приложения с несколькими формами

Цель работы: сформировать практические умения создания многооконного приложения средствами среды программирования Visual Studio

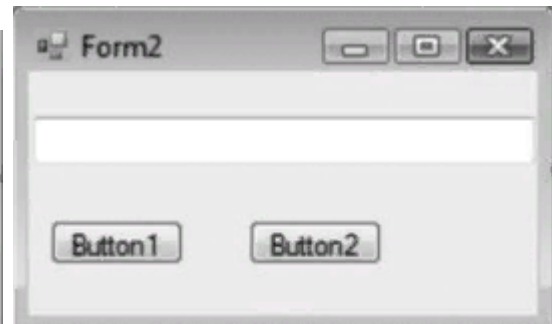
Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Содержание работы:

Задание 1. Разработать приложение в соответствии с приведенной формой, каждая кнопка открывает дополнительную форму.



Задание 2. Разработать приложение в соответствии с приведенными формами



ПРАКТИЧЕСКАЯ РАБОТА № 26

Тема: Разработка приложения с невидимыми компонентами

Цель работы: научиться разрабатывать оконные приложения с использованием дополнительных компонентов

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Справочный материал:

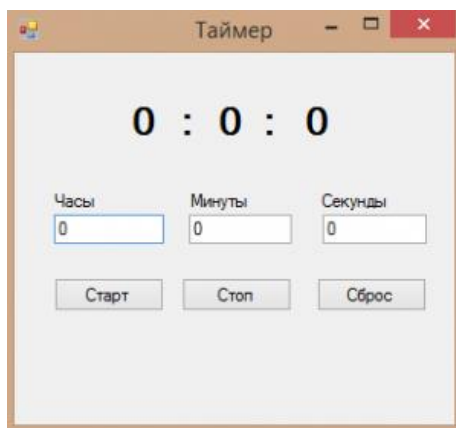
Невидимые компоненты представляют собой, как правило, компоненты, с помощью которых осуществляется доступ к системным ресурсам. Они отображаются только во время конструирования интерфейса, но не видны во время работы приложения. Невидимые компоненты (такие, как таймер) появляются на специальной панели и видимы только во время конструирования формы в виде имен со значками. Во время выполнения программы невидимые компоненты не отображаются. Доступ к компонентам .NET можно получить из панели Toolbox. Если ее не видно, то нужно выполнить в меню команду View | Toolbox.

Примером таких компонентов служит компонент Timer. Кроме него к невидимым компонентам относят диалоговые компоненты и компоненты-меню.


Содержание работы:


Задание 1. Создать приложение «Таймер»

1. Для начала в Windows Forms создаём внешнюю оболочку программы. У нас она выглядит вот так:



2. Здесь у нас 8 Label'ов, 3 TextBox'а, 3 Button'а и сам Timer.

Примечание: при переносе элемента Timer в форму, на неё ничего не появляется. Лишь в нижней части окна программы под формой появляется значок  timer1, не пугайтесь.


3. Щёлкнем на значок таймера  timer1 и в окне “Свойства” в группе “Поведение” устанавливаем значение параметра Interval равным 1000. Данный параметр определяет длину тика таймера в миллисекундах, указав 1000, мы сделали один тик равным одной секунде.

4. После оформления и настройки приступаем к коду. Вводим целочисленные переменные h – часы, m- минуты, s – секунды.

5. Затем дважды щёлкаем мышью на кнопке “Старт” и переходим на участок кода, отвечающий за клик на эту кнопку. Туда мы пишем следующий код:

```
h = Convert.ToInt32(textBox1.Text);  
m = Convert.ToInt32(textBox2.Text);  
s = Convert.ToInt32(textBox3.Text);  
timer1.Start();
```

То есть мы считываем с TextBox'ов данные, которые ввёл туда пользователь, и после этого включаем таймер. Время пошло.

6. Также нам надо настроить счёт времени самого таймера. Для этого дважды кликаем на элементе  timer1 и внутри тела кода, в который нас отправило, пишем:

```
private void timer1_Tick(object sender, EventArgs e)    {  
    s = s - 1;  
    if (s == -1)    {  
        m = m - 1;  
        s = 59;    }  
}
```

Здесь мы настраиваем таймер таким образом, чтобы каждую секунду переменная s уменьшалась на единицу. Если s становится меньше нуля, значит прошла минута, следовательно, m должна уменьшаться на единицу, а отсчёт с секундами s снова начнётся с 59.

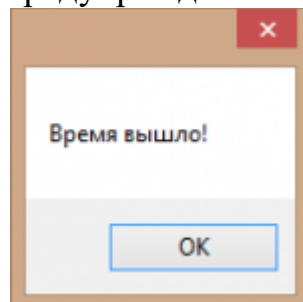
7. То же самое мы делаем с часами и минутами:

```
if (m == -1)    {  
    h = h - 1;  
    m = 59;    }
```

8. Теперь позаботимся о том, что случится, когда время, указанное пользователем, выйдет:

```
if (h == 0 && m == 0 && s == 0)    {  
    timer1.Stop();  
    MessageBox.Show("Время вышло!");    }
```

Как только часы, минуты и секунды будут вместе равняться нулю, мы выведем пользователю окно с предупреждением об этом.



9. А чтобы пользователь мог видеть, как идёт время, и как отсчитываются часы, минуты и секунды, мы вынесем всё вышепроясходящее на экран при помощи label'ов:

```
label1.Text = Convert.ToString(h);  
label3.Text = Convert.ToString(m);  
label5.Text = Convert.ToString(s);
```

10. Теперь надо разобраться с кнопками “Стоп” и “Сброс”. В первом случае при нажатии на кнопку пользователем, таймер просто останавливается и может быть возобновлён после нажатия на кнопку “Старт”. При нажатии на вторую

кнопку счётчики сбрасываются и при нажатии на “Старт”, отчёт начнётся заново.

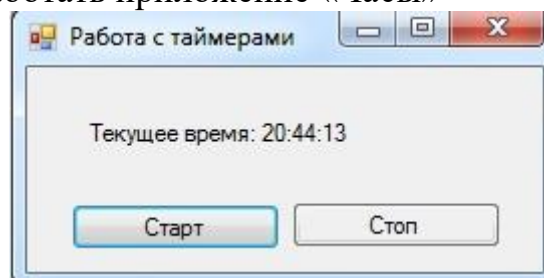
Код кнопки “Стоп”

```
private void button2_Click(object sender, EventArgs e)    {  
    timer1.Stop();    }
```

В кнопке “Сброс” нам надо помимо остановки сбросить значения переменных до нулей

```
private void button3_Click(object sender, EventArgs e)    {  
    timer1.Stop();  
    label1.Text = "0";  
    label3.Text = "0";  
    label5.Text = "0";    }
```

Задание 2. Разработать приложение «Часы»



ПРАКТИЧЕСКАЯ РАБОТА № 27

Тема: Разработка приложения с невидимыми компонентами

Цель работы: сформировать умения использования полос прокрутки для ввода информации, изучить основные свойства; сформировать умения по созданию процедур на основе событий компонента.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Справочный материал:

Элементы управления HScrollBar и VScrollBar представляют собой обычные полосы прокрутки, отображаемые на границах окон редактирования и просмотра приложений Microsoft Windows. Полоса прокрутки – классический элемент оконного интерфейса, использующийся для скроллинга информации, которая не помещается целиком в область вывода. В объектах просмотра и редактирования Windows этот элемент появляется при необходимости автоматически.

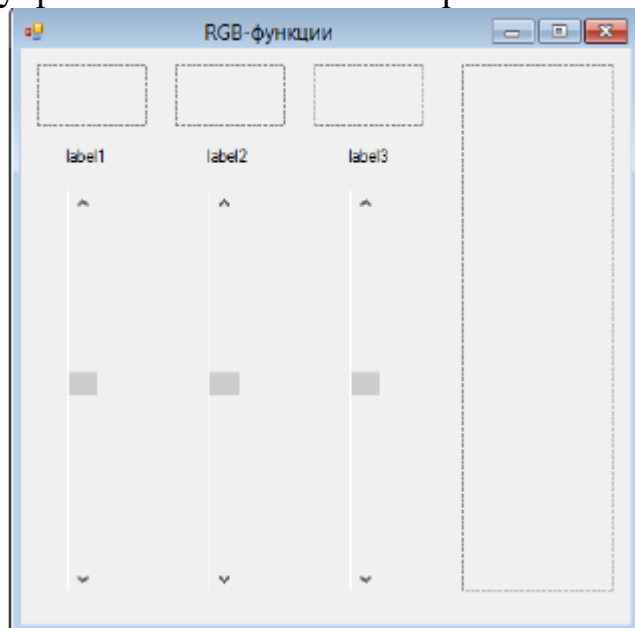
Содержание работы:

Задание 1. Разработать проект демонстрации работы RGB – функций (установок цвета по трем составляющим) с помощью полос прокрутки. Каждый бегунок полос прокрутки должен будет менять вклад RGB – компонента, отображающийся на панели как цвет, а на метке как число. Результирующий цвет должен отображаться на панели.

1. Запустите среду программирования Visual Studio. Создайте новое Приложение Windows Forms. Имя проекта и приложения ScrollBox1. Папка для размещения проекта ScrollBox.

2. Разместите на форме компоненты в соответствии с рисунком.

Окно элемента управления HScrollBar располагается горизонтально, а элемента управления VScrollBar— вертикально.



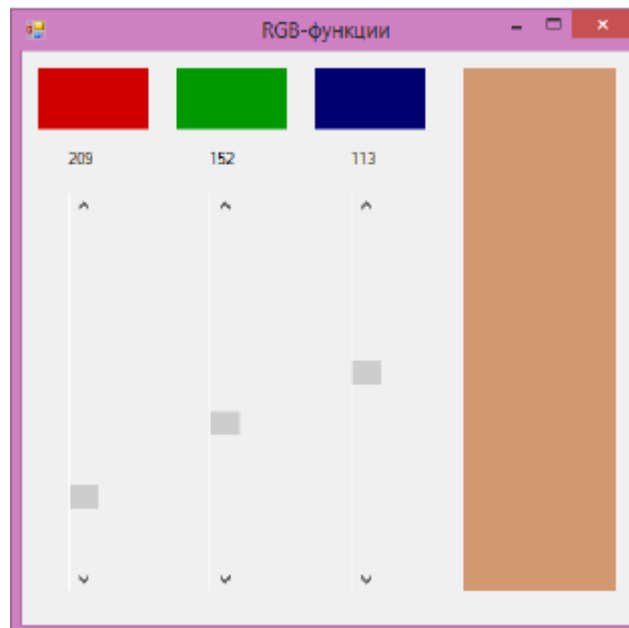
3. Задайте для элемента управления VScrollBar1 значение свойства Name=RedBar и установить значение свойств Max=255, Value=122, LargeChange=1.

4. Задайте для элемента управления VScrollbar2 значение свойства Name=GreenBar и установите значение свойств Max=255, Value=122, LargeChange=1.
5. Задайте для элемента управления VScrollbar3 значение свойства Name=BlueBar и установите значение свойств Max=255, Value=122, LargeChange=1.
6. Выделите элемент RedBar, на панели Свойства и перейдите на вкладку События. Найдите событие Scroll, справа от него в поле сделайте двойной щелчок левой кнопкой мыши. Оказавшись в коде программы, введите следующий код:

```
private void RedBar_Scroll(object sender, ScrollEventArgs e)
{
    panel1.BackColor = Color.FromArgb(Convert.ToInt32(RedBar.Value), 0, 0);
    label1.Text = Convert.ToString(RedBar.Value);
    panel4.BackColor = Color.FromArgb(Convert.ToInt32(RedBar.Value), Convert.ToInt32(GreenBar.Value), Convert.ToInt32(BlueBar.Value));
}
```

Метод FromArgb создает структуру Color из указанных 8-разрядных значений цветов (красный, зеленый, синий).

9. Аналогично запишите процедуры обработки события Scroll для элементов GreenBar и BlueBar.
10. Задайте имя формы проекта RGB-функции.
11. Свойство Text компонентов Label сделайте пустым.
12. Сохраните изменения и запустите проект, убедитесь в его работоспособности.



Задание 2. Разработать проект, который позволяет пользователю вычислить факториал числа. Число, для которого рассчитывается факториал, выбирается с помощью элемента управления TrackBar. При щелчке по кнопке «Расчет», меняется надпись «Число n» на «N!» и в строке ввода выводится значение факториала числа.

ПРАКТИЧЕСКАЯ РАБОТА № 28

Тема: Разработка приложения с невидимыми компонентами

Цель работы: научиться разрабатывать оконные приложения с использованием дополнительных компонентов

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Содержание работы:

Задание 1. Создание невидимого компонента Planets

1.Создадим невидимый компонент Planets, инкапсулирующий в себе названия планет Солнечной системы, и добавим его в нашу библиотеку компонентов сборки MyComponents. Названия планет будут храниться во внутреннем массиве компонента, а доступ к ним будет осуществляться через индексы по имени планеты или ее индексу.

2.В панели Solution Explorer выделите проект MyComponents и выполните команду меню Project/Add Component, чтобы добавить файл Planets.cs нового компонента

3.Дополните содержимое файла, автоматически сгенерированное мастером, следующим кодом

```
using System;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;
using System.Collections;
namespace MyCompany.MyComponents
{
    public partial class Planets : Component
    {
        public Planets()
        {
            InitializeComponent();
        }
        public Planets(IContainer container)
        {
            container.Add(this);
            InitializeComponent();
        }
    }
}
namespace MyCompany.MyComponents{
partial class Planets { // Планеты
    private string[] PlanetNames ={"Меркурий","Венера", "Земля", "Марс",
"Юпитер", "Сатурн","Уран", "Нептун", "Плутон" };
    // Вернуть имя
    private string GetPlanetName(int index)
    {
        // Нижняя и верхняя границы массива
        int lowP = 0, highP = PlanetNames.Length - 1;
        // Контролирует диапазон индекса планеты и возвращает
        // ее название или генерирует исключение
        if (index < lowP || index > highP)
        {
            MessageBox.Show(String.Format("Индекс должен находиться в диапазоне {0}-{1}",
                lowP, highP));
            index = 0;
        }
        return PlanetNames[index];
    }
    // Вернуть индекс
```

```

private int GetPlanetPosition(string planetName)    {
    // Сравниваем переданное имя планеты с массивом
    // PlanetNames. При несовпадении возвращаем -1
    int result=-1;
    for(int i=0; i<PlanetNames.Length;i++)
        if (String.Compare(planetName, PlanetNames[i], true) == 0) {
            result = i;
            break;        }
    return result;    }

    // Свойство - индекатор: возвращает имя планеты по индексу
public string this[int index]    {
    get {return GetPlanetName(index); }    }

    // Свойство - индекатор: возвращает индекс планеты по имени
public int this[string planetName]    {
    get { return GetPlanetPosition(planetName); }    }

    // Свойство максимального размера массива
public int MaxIndex    {
    get { return PlanetNames.Length - 1; }    } }

```

4. Создали два внутренних контролирующих метода, возвращающих название планеты по ее индексу и наоборот. Добавили два общедоступных свойства - индекатора, позволяющих работать с экземпляром компонента как с массивом, а также добавили общедоступное свойство максимального размера поля - массива с планетами.

5. Теперь осталось перекомпилировать сборку MyComponents.dll, в которой в одном пространстве имен MyCompany.MyComponents будут находиться уже два наших компонента, и испытать новый невидуальный компонент.

6. В панели Solution Explorer вызовите контекстное меню для узла проекта MyComponets и выполните команду Rebuild, чтобы перекомпилировать проект с компонентами

7. Убедитесь, что в панели Toolbox появился новый компонент Planets, который теперь можно перетаскивать на форму также, как и обычный библиотечный компонент

8. Испытание созданных компонентов. В проекте ComponentTest настройте пользовательский интерфейс, как показано на рисунке и в таблице (в скобках приведены имена экземпляров компонентов)

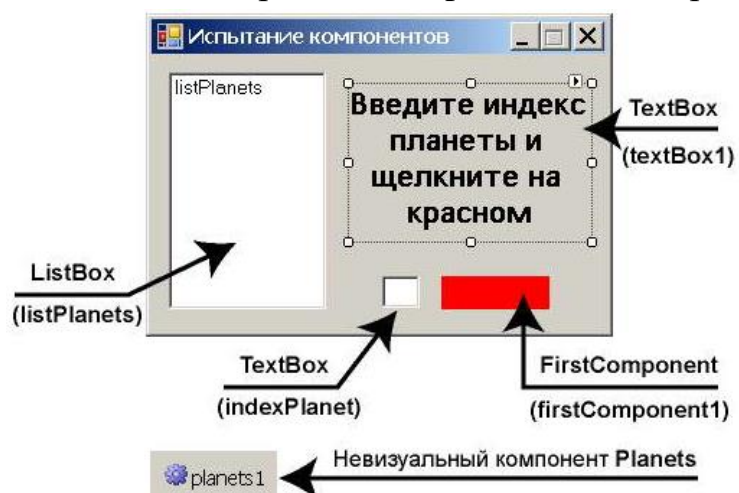


Таблица свойств элементов интерфейса		
Элемент	Свойство	Значение
Form1	MaximizeBox	false
	Size	324; 224
	Text	ИСПЫТАНИЕ КОМПОНЕНТОВ
listPlanets	FormattingEnabled	true
	ItemHeight	16
	Location	12; 12
	Size	109; 164
indexPlanet	Location	160; 153
	Size	26; 22
textBox1	BorderStyle	None
	Font	Microsoft Sans Serif; 12pt; style=Bold
	Location	138; 21
	Multiline	true
	ReadOnly	true
	Size	166; 107
	Text	Введите индекс планеты и щелкните на красном
	TextAlign	Center
firstComponent1	Location	201; 153
	Size	75; 23

9.В панели Properties перейдите на вкладку Events и создайте обработчики для элементов согласно таблице

Таблица событий элементов интерфейса		
Элемент	Событие	Имя обработчика
listPlanets	SelectedIndexChanged	listPlanets_SelectedValueChanged
indexPlanet	KeyPress	indexPlanet_KeyPress
firstComponent1	Click	firstComponent1_Click

10.Заполните файл Form1.cs следующим кодом

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;
namespace ComponentTest
{
    public partial class Form1: Form
    {
        // Конструктор формы
        public Form1()
        {
            InitializeComponent();
            // Заполнение списка планетами
            for (int i = 0; i <= planets1.MaxIndex; i++)
            {
```



```

listPlanets.Items.Add(String.Format(
    "{0}) {1}", i, planets1[i]));    }
    listPlanets.SelectedIndex = 0;    }
    // Обработчики событий
private void firstComponent1_Click(object sender, EventArgs e)    {
    // Контролируем пустой ввод
    if (indexPlanet.Text == String.Empty) return;
    int index = Convert.ToInt32(indexPlanet.Text);
    // Контролируем максимальный индекс ввода
    index = Math.Min(index, planets1.MaxIndex);
    if (listPlanets.SelectedIndex != index)
        listPlanets.SelectedIndex = index;
    else
        MessageBox.Show(String.Format("Вы выбрали
            планету {0}", planets1[index]));    }
    private void indexPlanet_KeyPress(object sender,
        KeyPressEventArgs e)    {
    // Фильтруем цифры, Backspace, Enter (Delete и стрелки по умолчанию)
    if ((e.KeyChar < Convert.ToChar(Keys.D0) ||
        e.KeyChar > Convert.ToChar(Keys.D9))
        && e.KeyChar != Convert.ToChar(Keys.Back)
        && e.KeyChar != Convert.ToChar(Keys.Enter))
        e.Handled = true;
    // Реакция на клавишу Enter
    if (e.KeyChar == Convert.ToChar(Keys.Enter))
        firstComponent1_Click(null, EventArgs.Empty);    }
    bool loadFlag = true; // Локальное поле-флаг
private void listPlanets_SelectedValueChanged
    (object sender, EventArgs e)    {
    int index = listPlanets.SelectedIndex;
    indexPlanet.Text = index.ToString();
    if (loadFlag)    {
    // При первом запуске не показывать
    loadFlag = false;
    return;    }
    else
        MessageBox.Show(String.Format("Вы выбрали
            планету {0}", planets1[index]));    }    }}

```

11.Откомпилируйте приложение текущего уровня готовности и испытайте его работу

Задание 2. Создайте приложение, содержащее визуальные и невидимые компоненты.

ПРАКТИЧЕСКАЯ РАБОТА № 29

Тема: Разработка приложения с графикой и анимацией

Цель работы: изучить теоретические принципы использования графических объектов GDI+ и получить практические навыки разработки программ, имитирующих движение графических объектов.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Справочный материал:

Все методы класса Graphics, предназначенные для рисования фигур или текста, получают через один из параметров перо класса Pen или кисть класса Brush, с помощью которых и выполняется рисование.

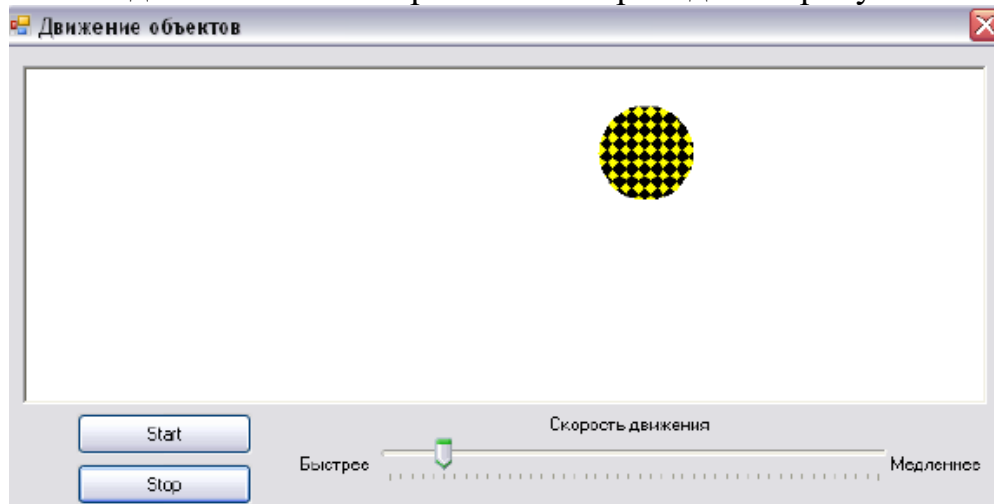
Перья используются для рисования линий и простейших геометрических фигур и создаются как объекты класса Pen.

Для добавления графики используется панель PictureBox.

Содержание работы:

Задание 1. Реализовать эффект движения простейшего геометрического объекта (круга) внутри области рисования с «отскакиванием от стенок». Предусмотреть возможность изменения его цвета и скорости движения. Изменение цвета круга происходит при щелчке в нем левой кнопкой мыши. Скорость регулируется с помощью компонента TrackBar.

1. Внешний вид главного окна приложения приведен на рисунке:



Для графической панели pictureBox1 устанавливаются свойства:

Свойство	Значение	Описание
BorderStyle	Fixed3D	Стиль границы
BackColor	White	Цвет фона

2. Проектирование программного кода

2.1. Начальное рисование фигуры. Т.к. в данном приложении будет создаваться кисть со штриховой заливкой, прежде всего, необходимо в файл главной формы подключить пространство имен System.Drawing.Drawing2D.

Для рисования круга необходимо иметь координаты его центра и радиус. Для этого нужно объявить соответствующие переменные в классе формы:
`int x0, y0; int radius;`

Начальные значения этим переменным можно установить в конструкторе формы или в обработчике события Load. Здесь начальное положение круга – левый нижний угол области рисования, радиус – 30 пикселей.

```
x0 = 30;    y0 = pictureBox1.Height - 35; radius = 30;
```

Рисование круга будет производиться в обработчике события Paint области рисования pictureBox1. Процесс рисования состоит из создания объекта Graphics, очистки области рисования, создания кисти и непосредственного рисования круга. Кисть создается как объект класса HatchBrush (штриховая кисть). При этом первым параметром задается стиль заливки (здесь – HatchStyle.SolidDiamond – «шахматка»), вторым параметром – основной цвет заливки (желтый). Можно указать и третий параметр – «неосновной» цвет или цвет фона (по умолчанию – черный). При рисовании круга (метод FillEllipse()) необходимо задать кисть для заливки, координаты левого верхнего угла прямоугольника, ограничивающего окружность, а также его стороны. Код обработчика приведен ниже:

```
private void pictureBox1_Paint(object sender, PaintEventArgs e) {  
    Graphics g = e.Graphics;    //создаем графический объект  
    g.Clear(Color.White); //очищаем область рисования  
    //создаем штриховую кисть желтого цвета  
    HatchBrush brush = new HatchBrush(HatchStyle.SolidDiamond,  
Color.Yellow);  
    //рисуем круг  
    g.FillEllipse(brush, x0 - radius, y0 - radius, 2 * radius, 2* radius); }
```

2.2. Реализация движения фигуры

Реализация движения основана на использовании компонента-таймера (Timer). После помещения на форму его свойству Interval (интервал в миллисекундах, через который поступает сигнал от таймера) устанавливается значение 10. По нажатию кнопки «Start» необходимо активизировать («включить») таймер, а кнопки «Stop» – остановить его.

```
private void button1_Click(object sender, EventArgs e) { //кнопка "Start"  
    timer1.Start();}  
    private void button2_Click(object sender, EventArgs e) { //кнопка "Stop"  
        timer1.Stop(); }
```

Чтобы заставить фигуру «двигаться», достаточно изменять ее координаты по сигналу таймера, после чего перерисовать фигуру:

```
private void timer1_Tick(object sender, EventArgs e) {  
    //изменяем координаты круга  
    x0++;    y0--;  
    //обновляем область рисования (перерисовываем круг)  
    pictureBox1.Invalidate(); }
```

В данном случае после нажатия кнопки «Start» круг будет двигаться по диагонали вправо вверх (координата x увеличивается, y – уменьшается), пока не исчезнет за пределами области рисования или пока не будет нажата кнопка

«Stop».

2.3. Программная реализация «отскакивания от стенок»

Для создания подобного эффекта нужно отслеживать координаты круга с учетом его радиуса. В этом случае удобно создать в классе формы две дополнительные переменные, задающие направление движения по каждой из осей координат: `int xDir, yDir`;

Переменная `xDir` принимает значение 1, если круг движется вправо, и -1, если влево. Переменная `yDir` равна 1, если идет движение вниз, и -1, если вверх. Поскольку изначально направление движения задается вправо вверх, этим переменным нужно задать соответствующие исходные значения (в конструкторе или в обработчике события `Load`): `xDir = 1; yDir = -1`;

«Отскакивание от стенок» реализуется за счет изменения значений этих переменных. Если круг «подлетел» к левой стенке, то после этого он должен начать двигаться вправо, т.е. переменной `xDir` нужно присвоить значение 1, и т.д. Приближение к стенкам отслеживается по значениям переменных `x0` и `y0` с учетом радиуса круга. При этом при «касании» кругом правой и нижней стенок необходимо вносить корректировку в 5 пикселей из-за наличия трехмерной рамки у области рисования.

Таким образом, обработчик события `Tick` для таймера модифицируется следующим образом:

```
private void timer1_Tick(object sender, EventArgs e) {  
    if (x0 - radius < 0) //если круг "подлетел" к левой стенке,  
        xDir = 1; //то начинаем двигаться вправо  
    if (x0 + radius + 5 > pictureBox1.Width) //если к правой стенке,  
        xDir = -1; //то влево  
    if (y0 - radius < 0) //если к верхней стенке,  
        yDir = 1; //то вниз  
    if (y0 + radius + 5 > pictureBox1.Height) //если к нижней стенке,  
        yDir = -1; //то вверх  
    x0 += xDir; y0 += yDir; //изменяем координаты круга  
    //обновляем область рисования (перерисовываем круг)  
    pictureBox1.Invalidate(); }  

```

2.4. Изменение скорости движения круга

Для изменения скорости движения нужно изменять интервал поступления сигнала от таймера, что, в свою очередь, будет регулироваться компонентом `TrackBar`. Интервал сигнала от таймера в данном приложении задается диапазоном от 5 до 50 миллисекунд. Соответствующие настройки задаются и для компонента `TrackBar`:

Свойство	Значение	Описание
Minimum	5	Минимальное значение
Maximum	50	Максимальное значение
Value	10	Текущее значение

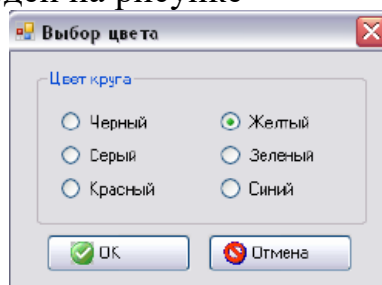
При прокрутке движка компонента `TrackBar` программе поступает сообщение `Scroll`, в обработчике которого и нужно изменять значение интервала таймера:

```
private void trackBar1_Scroll(object sender, EventArgs e) {
    //изменяем значение интервала таймера на выбранное
    timer1.Interval = trackBar1.Value; }

```

2.5. Создание диалоговой панели выбора цвета заливки

Для выбора цвета заливки круга в данном приложении будет использоваться дополнительное диалоговое окно с группой радиокнопок. Внешний вид этого окна приведен на рисунке



В этом окне в компоненте GroupBox расположены 6 радиокнопок (Radio-Button) с названиями цветов, а также 2 кнопки: «ОК» и «Отмена». На кнопки помимо надписей добавлены соответствующие рисунки. Рисунки можно добавить на кнопку либо через ее свойство Image, либо, как и сделано в данной программе, через компонент ImageList. Этот компонент представляет собой список изображений, которые потом можно назначать обычным кнопкам, пунктам меню, кнопкам панели инструментов и другим компонентам. После помещения этого компонента на форму нужно настроить его свойство Images, где с помощью отдельного диалогового окна в проект добавляются файлы с изображениями. Каждое добавленное изображение имеет свой индекс (порядковый номер), начиная с 0. В данном приложении рисунок для кнопки «ОК» имеет индекс 0, а для кнопки «Отмена» – 1.

Настройки кнопок «ОК» и «Отмена» приведены в таблице:

Свойство	Значение	Описание
ImageList	imageList1	Источник изображений
ImageIndex	0 для кнопки «ОК», 1 для кнопки «Отмена»	Индекс изображения в источнике
ImageAlign	MiddleCenter	Выравнивание изображения
TextAlign	MiddleCenter	Выравнивание текста
TextImageRelation	ImageBeforeText	Взаимное расположение текста и изображения
DialogResult	OK для кнопки «ОК», Cancel для кнопки «Отмена»	Возвращаемое значение для нажатия кнопки

Установка значения OK свойству DialogResult для кнопки «ОК» означает, что при нажатии на эту кнопку форма закроется и вернет в главное окно значение DialogResult.OK (для кнопки «Cancel» – DialogResult.Cancel).


2.6. Создание свойства для получения цвета










После отображения на экране диалогового окна значение выбранного цвета необходимо передать в главную форму программы. Непосредственно

значение выбранной радиокнопки передать не удастся, т.к. переменные, описывающие компоненты формы, закрыты (private). Поэтому для решения этой проблемы необходимо создать в классе Form2 общедоступное свойство для получения выбранного цвета, а также для установки текущего цвета круга (соответствующая радиокнопка становится выбранной). Тип свойства – Color. Код свойства приведен далее:

```
public Color CircleColor {
    get {
        if (radioButton1.Checked)    //если выбрана первая радиокнопка,
            return Color.Black; //то вернуть черный цвет
        if (radioButton2.Checked) //и т.д.
            return Color.Gray;
        if (radioButton3.Checked)
            return Color.Red;
        if (radioButton4.Checked)
            return Color.Yellow;
        if (radioButton5.Checked)
            return Color.Green;
        return Color.Blue;    }
    set {
        if (value == Color.Black)    //если текущий цвет круга - черный
            radioButton1.Checked = true; //то сделать выбранной первую радиокнопку
        if (value == Color.Gray)    //и т.д.
            radioButton2.Checked = true;
        if (value == Color.Red)
            radioButton3.Checked = true;
        if (value == Color.Yellow)
            radioButton4.Checked = true;
        if (value == Color.Green)
            radioButton5.Checked = true;
        if (value == Color.Blue)
            radioButton6.Checked = true;    } }
```

Задание 2. В каждом варианте необходимо реализовать движение требуемого изображения по заданной траектории, не допуская при этом выход фигуры за пределы области рисования. Каждое изображение состоит из нескольких (двух-трех) элементарных фигур. Предусмотреть возможность изменения скорости движения с помощью компонента TrackBar;

№ ва- рианта	Вид изображения	Траектория движения
1.		Вдоль границ области рисования по часовой стрелке

2.		Вдоль границ области рисования против часовой стрелки
3.		От центра области рисования: вверх, вниз, влево, вправо
4.		От центра области рисования: влево, вверх, вправо, вниз. От каждой стенки – к центру
5.		От центра области рисования поочередно к углам. От угла – назад в центр
6.		Вдоль границ области рисования по часовой стрелке
7.		Вдоль границ области рисования против часовой стрелки
8.		От центра области рисования: вверх, вниз, влево, вправо
9.		От центра области рисования: влево, вверх, вправо, вниз. От каждой стенки – к центру
10.		От центра области рисования поочередно к углам. От угла – назад в центр

ПРАКТИЧЕСКАЯ РАБОТА № 30

Тема: Разработка приложения с графикой и анимацией

Цель работы: получить практические навыки разработки программ, имитирующих движение графических объектов.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Содержание работы:

Задание 1. С помощью графических операторов в с# сделать часы

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace AnalogClock{
    public partial class Form1: Form    {
        public Form1()    {
            InitializeComponent();
            Timer timerTime = new Timer();
            timerTime.Tick += new EventHandler(timerTime_Tick);
            timerTime.Interval = 1000;
            timerTime.Enabled = true;
            g = this.CreateGraphics();
            g.SmoothingMode    =    System.Drawing.Drawing2D.SmoothingMode.
HighQuality;    }
    //величина, изменение которой масштабирует размер самих часов
    double Lenght = 120;
    Graphics g;
    //цвет фона часов
    Color bkColor = Control.DefaultBackColor;
    //обновление-рисовка фона за часам
    private void PaintBackGround()    {
        g.FillRectangle(new SolidBrush(bkColor),new Rectangle(new
Point((int)(this.ClientRectangle.Width / 2 - Lenght),(int)(this.ClientRectangle.Height
/ 2 - Lenght)),
        new Size((int)Lenght * 2, (int)Lenght * 2)));    }
    private void PaintCircle(){    //просовка циферблата
        //прорисовка внешней окружности
        g.DrawEllipse(new Pen(new SolidBrush(Color.CornflowerBlue),2),
            new Rectangle(new Point((int)(this.ClientRectangle.Width / 2 -
Lenght),(int)(this.ClientRectangle.Height / 2 - Lenght)),new Size((int)Lenght * 2,
(int)Lenght * 2)));
        //прорисовка линий, который указывают на деления часов
        for (int i = 0; i < 12; i++)    {
            g.DrawLine(new Pen(new SolidBrush(Color.CornflowerBlue), 2),
                new Point((int)(ClientRectangle.Width / 2),(int)(ClientRectangle.Height/ 2)),
                new Point((int)(ClientRectangle.Width / 2 + Lenght * Math.Cos(Math.PI /
6 * i)), (int)(ClientRectangle.Height / 2 + Lenght * Math.Sin(Math.PI / 6 * i))));
```



```

        if (i == 3)
        {
            float x = (float) (ClientRectangle.Width/2 + Lenght*
Math.Cos(Math.PI/6*i));
            float y = (float) (ClientRectangle.Height/2 + Lenght*
Math.Sin(Math.PI/6*i));
            string s = "16";
            g.DrawString(s, new Font("Mono", 20), Brushes.Green, x+110, y-65); } }
        //прорисовка круга, который закрывает внутреннюю часть линий чтобы
остались только черточки
        //этот круг меньше диаметром внешней окружности
        g.FillEllipse(new SolidBrush(bkColor),new Rectangle(new
Point((int)(this.ClientRectangle.Width / 2 - Lenght + 10),
(int)(this.ClientRectangle.Height / 2 - Lenght + 10)),new
Size((int)(Lenght - 10) * 2, (int)(Lenght - 10) * 2))); }
        //прорисовка стрелок часов
        private void PaintArrows(DateTime dt)
        {
            //прорисовка минутной стрелки
            g.DrawLine(new Pen(new SolidBrush(Color.Black), 2),new
Point((int)(ClientRectangle.Width / 2), (int)(ClientRectangle.Height / 2)),
new Point((int)(ClientRectangle.Width / 2 + (Lenght - 4) * Math.Sin(2 *
Math.PI / 60 * dt.Minute)), (int)(ClientRectangle.Height / 2 - (Lenght - 4) *
Math.Cos(2 * Math.PI / 60 * dt.Minute))));
            //определения количества часов, прошедших после полудня или после
полуночи
            //фактически перевод 23=>11 и так далее
            int hour;
            if (dt.Hour <= 12)
            {
                hour = dt.Hour;
            }
            else
            {
                hour = dt.Hour - 12;
            }
            //прорисовка часовой стрелки
            g.DrawLine(new Pen(new SolidBrush(Color.Black), 2),new
Point((int)(ClientRectangle.Width / 2), (int)(ClientRectangle.Height / 2)),
new Point((int)(ClientRectangle.Width / 2 + (Lenght - 10) * Math.Sin(2 *
Math.PI / 12 * hour + 2 * Math.PI / (12 * 60) * dt.Minute)),
(int)(ClientRectangle.Height / 2 - (Lenght - 10) * Math.Cos(2 * Math.PI / 12 * hour +
2 * Math.PI / (12 * 60) * dt.Minute)))); }
            //общая прорисовка часов
            private void PaintClock(DateTime dtArg)
            {
                //PaintBackGround(); //фон
                PaintCircle(); //циферблат
                PaintArrows(dtArg); } //стрелки
            //"тик" таймера
            private void timerTime_Tick(object sender, EventArgs e)
            {
                PaintClock(DateTime.Now);
            }

```

```

//первая загрузка формы
private void Form1_Shown(object sender, EventArgs e)    {
    PaintClock(DateTime.Now);    }
private void Form1_Load(object sender, EventArgs e)    {
    g.FillRectangle(new SolidBrush(bkColor),new Rectangle(new
Point((int)(this.ClientRectangle.Width / 2 - Lenght),(int)(this.ClientRectangle.Height
/ 2 - Lenght)),
    new Size((int)Lenght * 2, (int)Lenght * 2)));    } }

```

Задание 2. Разработайте приложение по индивидуальным заданиям

- 1.Создайте программу, показывающую пульсирующее сердце.
2. Создайте приложение, отображающее вращающийся винт самолета.
3. Разработайте программу анимациидвигающегося человечка.
4. Разработайте программу анимации падения снежинки.
5. Создайте программу, показывающую скачущий мячик.
6. Разработайте программу анимации летающего бумеранга.
7. Разработайте программу анимации взлета ракеты. Старт осуществляется по нажатию специальной «красной» кнопки.
8. Разработайте программу анимации движения планет в Солнечной системе. .
9. Разработайте программу анимации падения нескольких листков с дерева. Движение не должно быть линейным.
10. Создайте приложение, отображающее движение автомобиля с вращающимися колесами

ПРАКТИЧЕСКАЯ РАБОТА № 31

Тема: Разработка приложения с графикой и анимацией

Цель работы: изучить возможности Visual Studio по открытию и сохранению файлов, написать и отладить программу для обработки изображений.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Справочный материал:

Обычно для отображения точечных рисунков, рисунков из метафайлов, значков, рисунков из файлов в формате BMP, JPEG, GIF или PNG используется объект PictureBox, т. е. элемент управления PictureBox действует как контейнер для картинок. Можно выбрать изображение для вывода, присвоив значение свойству Image. Свойство Image может быть установлено в окне свойств или в коде программы, указывая на рисунок, который следует отображать.

Элемент управления PictureBox содержит и другие полезные свойства, в том числе свойство AutoSize, определяющее, будет ли изображение растянуто в элементе PictureBox, и SizeMode, которое может использоваться для растягивания, центрирования или увеличения изображения в элементе управления PictureBox.

Элемент управления OpenFileDialog является стандартным диалоговым окном. Он аналогичен диалоговому окну «Открыть файл» операционной системы Windows. Элемент управления OpenFileDialog позволяет пользователям просматривать папки личного компьютера или любого компьютера в сети, а также выбирать файлы, которые требуется открыть.

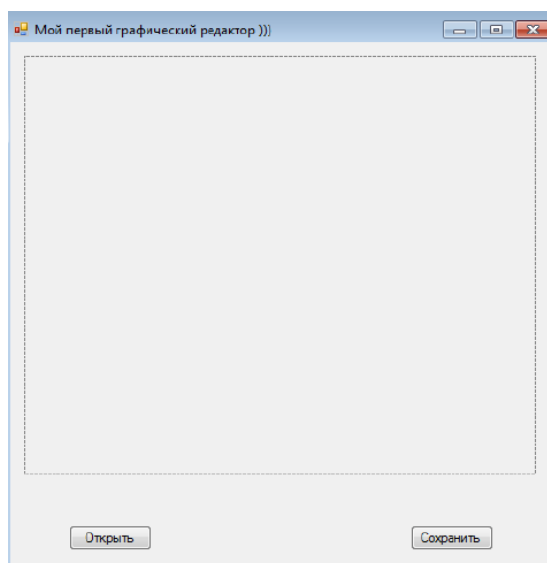
Для вызова диалогового окна для выбора файла можно использовать метод ShowDialog(), который возвращает значение DialogResult.OK при корректном выборе. Диалоговое окно возвращает путь и имя файла, который был выбран пользователем в специальном свойстве FileName.

Содержание работы:

Задание 1. Создать приложение, реализующее простой графический редактор. Функциями этого редактора должны быть: открытие рисунка, рисование поверх него простой кистью, сохранение рисунка в другой файл. Для этого создать форму и разместить на ней элементы управления Button и PictureBox.

В этом случае не понадобится из панели элементов размещать на форме элементы диалоговых окон OpenFileDialog и SaveFileDialog.

Эти элементы будут порождены динамически в ходе выполнения



программы с помощью конструктора. Например, так:

```
OpenFileDialog dialog = new OpenFileDialog();
```

Далее они будут вызываться с помощью метода ShowDialog(). Для кнопок «Открыть» и «Сохранить» создайте свои обработчики события. Также создайте обработчик события Load для формы. Для события

MouseDown, MouseMove.

Код приложения будет выглядеть следующим образом:

```
// Глобальные переменные
private Point PreviousPoint, point;
private Bitmap bmp;
private Pen blackPen;
private Graphics g;
// Действия при загрузке формы
private void Form1_Load(object sender, EventArgs e) {
    // Подготавливаем перо для рисования
    blackPen = new Pen(Color.Black, 4); }
// Действия при нажатии кнопки загрузки изображения
private void button1_Click(object sender, EventArgs e) {
    // Описываем объект класса OpenFileDialog
    OpenFileDialog dialog = new OpenFileDialog();
    // Задаем расширения файлов
    dialog.Filter = "Image files (*.BMP, *.JPG, " + "*.GIF, *.PNG)|*.bmp;*.jpg;*.gif;*.png";
    // Вызываем диалог и проверяем выбран ли файл
    if (dialog.ShowDialog() == DialogResult.OK) {
        // Загружаем изображение из выбранного файла
        Image image = Image.FromFile(dialog.FileName);
        int width = image.Width;
        int height = image.Height;
        pictureBox1.Width = width;
        pictureBox1.Height = height;
        // Создаем и загружаем изображение в формате bmp
        bmp = new Bitmap(image, width, height);
        pictureBox1.Image = bmp; // Записываем изображение в pictureBox1
        // Подготавливаем объект Graphics для рисования
        g = Graphics.FromImage(pictureBox1.Image); } }
// Действия при нажатии мышки в pictureBox1
private void pictureBox1_MouseDown(object sender,
    MouseEventArgs e) {
    // Записываем в предыдущую точку текущие координаты
    PreviousPoint.X = e.X;
    PreviousPoint.Y = e.Y; }
// Действия при перемещении мышки
private void pictureBox1_MouseMove(object sender,
    MouseEventArgs e) {
```

```

// Проверяем нажата ли левая кнопка мыши
if (e.Button == MouseButton.Left) {
    // Запоминаем текущее положение курсора мыши
    point.X = e.X;
    point.Y = e.Y;
    // Соединяем линией предыдущую точку с текущей
    g.DrawLine(blackPen, PreviousPoint, point);
    // Текущее положение курсора - в PreviousPoint
    PreviousPoint.X = point.X;
    PreviousPoint.Y = point.Y;
    // Принудительно вызываем перерисовку
    pictureBox1.Invalidate(); } }
// Действия при нажатии кнопки сохранения файла
private void button2_Click(object sender, EventArgs e) {
    // Описываем и порождаем объект savedialog
    SaveFileDialog savedialog = new SaveFileDialog();
    // Задаем свойства для savedialog
    savedialog.Title = "Сохранить картинку как ...";
    savedialog.OverwritePrompt = true;
    savedialog.CheckPathExists = true;
    savedialog.Filter =
        "Bitmap File(*.bmp)|*.bmp" + "GIF File(*.gif)|*.gif" + "JPEG File(*.jpg)|
*.jpg" + "PNG File(*.png)|*.png";
    // Показываем диалог и проверяем задано ли имя файла
    if (savedialog.ShowDialog() == DialogResult.OK) {
        string fileName = savedialog.FileName;
        // Убираем из имени расширение файла
        string strFilExtn = fileName.Remove(0,
            fileName.Length - 3);
        switch (strFilExtn) { // Сохраняем файл в нужном формате
            case "bmp":
                bmp.Save(fileName,
                    System.Drawing.Imaging.ImageFormat.Bmp);
                break;
            case "jpg":
                bmp.Save(fileName,
                    System.Drawing.Imaging.ImageFormat.Jpeg);
                break;
            case "gif":
                bmp.Save(fileName,
                    System.Drawing.Imaging.ImageFormat.Gif);
                break;
            case "tif":
                bmp.Save(fileName,
                    System.Drawing.Imaging.ImageFormat.Tiff);

```

```

        break;
    case "png":
        bmp.Save(fileName,
            System.Drawing.Imaging.ImageFormat.Png);
        break;
    default:
        break; } }

```

Далее добавим в проект кнопку для перевода изображения в градации серого цвета:

```

// Действия при нажатии кнопки перевода в градации серого
private void button3_Click(object sender, EventArgs e) {
    // Циклы для перебора всех пикселей на изображении
    for (int i = 0; i < bmp.Width; i++)
        for (int j = 0; j < bmp.Height; j++) {
            int R = bmp.GetPixel(i, j).R; // Извлекаем в R значение красного цвета
            int G = bmp.GetPixel(i, j).G; // Извлекаем в G значение зеленого цвета
            int B = bmp.GetPixel(i, j).B; // Извлекаем в B значение синего цвета
            int Gray = (R + G + B) / 3; // Вычисляем среднее арифметическое
            // Переводим число в значение цвета.
            // 255 – показывает степень прозрачности.
            // Остальные значения одинаковы
            Color p = Color.FromArgb(255, Gray, Gray, Gray);
            // Записываем цвет в текущую точку
            bmp.SetPixel(i, j, p); }
    // Вызываем функцию перерисовки окна
    Refresh(); }

```

Данный код демонстрирует возможность обращения к отдельным пикселям. Цвет каждого пикселя хранится в модели RGB и состоит из трех составляющих: красного, зеленого и синего цвета, называемых каналами. Значение каждого канала может варьироваться в диапазоне от 0 до 255.

Задание 2. Добавьте в приведенный графический редактор свои функции в соответствии с вариантом.

1. Расширьте приложение путем добавления возможности выбора пользователем цвета и величины кисти.
2. Разработайте функцию, добавляющую на изображение 1000 точек с координатами, заданными случайным образом. Цвет также задается случайным образом.
3. Создайте функцию, переводящую изображение в черно-белый формат. Пороговое значение задавать с помощью элемента управления TrackBar.
4. Разработайте функцию, оставляющую на изображении только один из каналов (R, G, B). Канал выбирается пользователем.

ПРАКТИЧЕСКАЯ РАБОТА № 32

Тема: Разработка игрового приложения

Цель работы: получение навыков разработки игровых приложений, используя язык C#.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Содержание работы:

Задание 1. Создать простой шутер «Вторжение НЛО» («UFO Invasion»), в котором придется отражать нашествие на Землю армады НЛО.

Действующие персонажи:

1) противник (враги — Enemies); 2) НЛО (один из видов противника — Bugs-«жучки»), ограничимся пока одним видом врагов; 3) защитник Земли (игрок — Player).

Место действия. Сражение происходит в космическом пространстве в окрестностях Земли.

Противник. НЛО имеют традиционную форму тарелок, каждая из них отличается размерами и окраской. Над Землей они появляются сериями и планируют сверху вниз на Землю с различной скоростью, при этом возможно и их горизонтальное смещение. Интервал времени между появлениями серий может уменьшаться. Количество НЛО в небе одновременно не превышает некоторого максимального числа — количество НЛО в космическом флоте противника.

Игрок. Игрок, находясь в своем корабле, использует свое оружие с лазерным прицелом (blaster) и перемещается в околоземном пространстве, старается сбить НЛО противника. Для управления перемещением игрока используется мышь, стрельба — нажатием левой кнопки мыши.

Взаимодействие. При попадании в НЛО следует обозначить его подрыв и удалить его останки из околоземного пространства. В первом приближении будем считать, что физическое касание НЛО и корабля не приводит к потере корабля игрока (это будет следующая задача).

Цель игры — уничтожить максимальное количество НЛО противника (максимальный результат — 100%).

Проектирование классов

Обозначив в сценарии сущности, мы определим три класса:

- 1) Enemies — противник;
- 2) Bugs — НЛО;
- 3) Player – игрок.
- 4) Form1 – форма. Автоматически создается класс
- 5) BrushColor – кисти/цвета. Вспомогательный класс

Класс Player

Поля класса:

public Point point;	// положение игрока в 2D-пространстве
public Size size;	// размеры игрока
public Region reg;	// занимаемая им область в пространстве
public Pen laser_pen;	// свойство оружия

Методы класса:

```
public void New_player()      // задать свойства (параметры) игрока
public void Show_player()     // показать его на поле битвы
```

Содержание методов этого и последующих классов рассмотрим позднее.

Класс Bugs

Поля:

```
public Point point;           // положение НЛО в 2D-пространстве
public Size size;             // размеры НЛО
int veloX;                    // скорость смещения по X
int veloY;                    // скорость_падения по Y
public HatchBrush br;         // кисть для покраски НЛО
public Region reg = new Region(); // занимаемая им область в пространстве
public Boolean life = true;    // НЛО жив (true) или мертв (false)
```

Методы:

```
public void New_bug()         // задать свойства (параметры) НЛО
public void Form_bug()        // задать форму НЛО, например, тарелку
public void Move_bug()        // задать новое местоположение НЛО
```

Класс Enemies

Поля:

а) для генерации серий

```
public int Delta_N;           // количество НЛО в серии
public int N_generation;      // число генераций — серий
public int k_generation;      // номер серии
public int N;                  // актуальное количество НЛО на экране
```

б) массив НЛО-объектов

```
public Bugs[] bugs = new Bugs[Form1.N_max];
```

Примечание. Поле N_max задается константой в классе Form1 (см. ниже), а последнее поле bugs задает массив ссылок на объекты-НЛО.

Методы:

```
public void New_Enemies()     // инициализация объектов НЛО
public void Show_bugs()       // сдвинуть и показать «живые» НЛО
public void Enemy()           // генерация одной серии НЛО
public void Killed_bugs()     // определение сбитых НЛО
public int Select_bugs()      // удаление сбитых НЛО
```

Вспомогательный класс BrushColor

Поля класса:

```
public Color FonColor;        // цвет фона
public Color LaserColor;      // цвет лазера
public Color DashBug;         // цвет штриховки НЛО
public Color KilledBug;       // цвет сбитого НЛО
```

Методы класса:

```
public BrushColor()            // конструктор (настройка цветов)
public HatchBrush New_br(int rch) // кисть для задания цвета НЛО
public Color RandomColor(int rch) // генератор случайного цвета
```

Класс Form1

Поля:

```
public const int N_max = 200;    // Максимальное количество НЛО на экране
public Player player = new Player();    // Игрок, который сбивает НЛО (объект)
public Boolean laser = false;        // Его оружие — бластер
public Bitmap imageP;                // Изображения игрока
public int Result = 0;                // Количество сбитых НЛО (счет игры)
public Graphics g;                    // холст для битвы
public BrushColor bc= new BrushColor();    // набор кистей и цветов
public Enemies nlo = new Enemies();        // Все НЛО
```

Как мы видим, задана всего одна константа (необходима для определения размерности массива ссылок, см. класс Enemies); три объекта классов Player, Enemies и BrushColor; поле laser (включен/выключен); поле imageP для хранения изображения игрока; поле Result для ведения счета подбитых НЛО; поле g — холст (графический контекст) для рисования.

Управление игрой. Выберем на панели элементов 4 визуальных объекта: три кнопки — button1 («Старт»), button2 («Лазер: включить/выключить»), button3 («Стоп») и textBox1 для отображения счета игры (Result) и перенесем их на форму. Также добавим 3 невидимых объекта: timer1, timer2 и imageList1. Объект imageList1 будем использовать для хранения изображений игрока (player). Таймер timer1 будет задавать частоту изменений (минимальный временной такт). Таймер timer2 будем использовать для генерации серий НЛО. Активность игрока свяжем, как указано в сценарии, с нажатием левой кнопки мыши и перемещением ее по экрану — событие MouseClick.

Методы класса Form1 (реакции на события):

```
а) конструктор формы      public Form1()
б) при загрузке формы      private void Form1_Load(object sender, EventArgs e)
в) старт игры              private void button1_Click(object sender, EventArgs e)
г) включение/отключение лазера
private void button2_Click(object sender, EventArgs e)
д) стоп игры, результат
private void button3_Click(object sender, EventArgs e)
е) один временной такт игры
private void timer1_Tick(object sender, EventArgs e)
ж) генерация серий
private void timer2_Tick(object sender, EventArgs e)
3) попадание НЛО под вертикальный обстрел лазером
private void Form1_MouseClick(object sender, MouseEventArgs e)
```

Настройка объектов

Для использования полного экрана зададим свойство Form1.WindowState=Maximized.

Подготовим в графическом редакторе изображения игрока размером 100x100 пикселей с именами player.bmp и player1.bmp, например так:



и сохраним их в папке Resources проекта. Зададим свойство `imageList1.ImageSize = 100;100` и внесем в коллекцию `imageList1.Images` эти два файла (члены 0 и 1)

Зададим `button3.Enabled=false`.

Изображения НЛО будет генерировать метод `public void Form_bug()`.

Настройки таймеров:

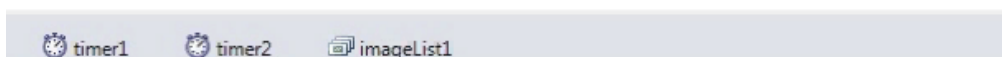
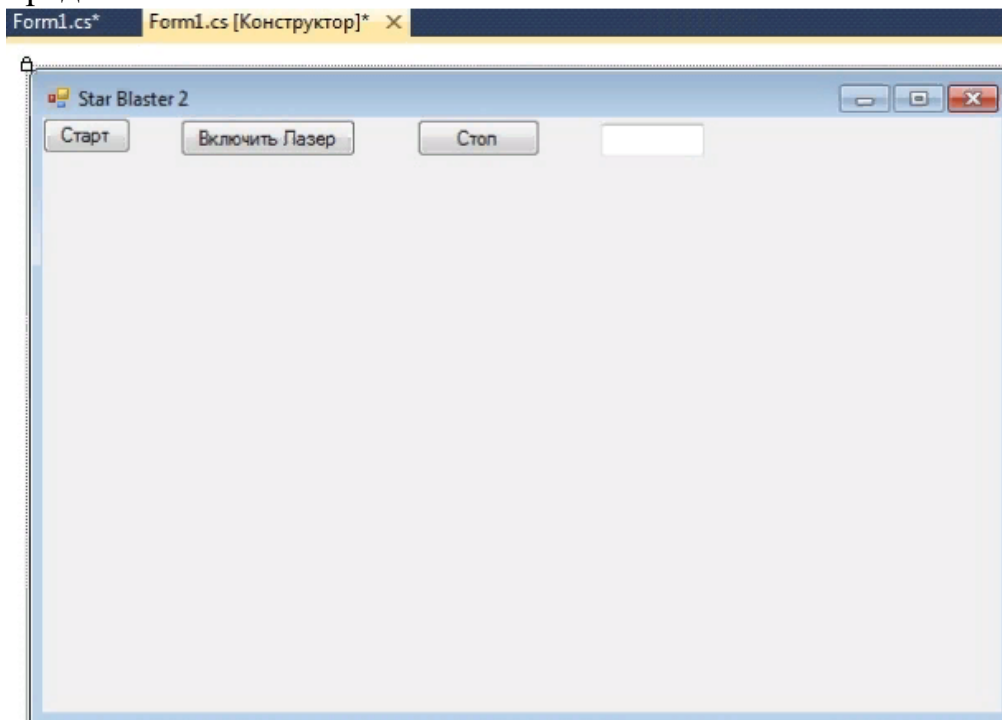
`timer1.Enable=false` (выключен);

`timer2.Enable=false` (выключен);

`timer1.Interval=400` (0,4с);

`timer2.interval=5000` (5с).

Исходный вид формы (в том числе с не визуальными компонентами) представлен ниже:



Вместо кнопок можно задать меню из трех позиций: Старт, Лазер, Стоп. Соответственно сменяются и названия методов, связанных с выбором пунктов меню.

1. Реализация методов класса Player:

а) Новый игрок

```
public void New_player(Form1 F) {  
    size = F.imageP.Size;  
    point.X = 0;
```

```

point.Y = 0;
Rectangle rec = new Rectangle(point, size);
reg = new Region(rec);
laser_pen = new Pen(new HatchBrush(HatchStyle.DashedUpwardDiagonal,
F.bc.LaserColor, F.bc.LaserColor), 3);}

```

Комментарии: Размер изображения определяется через размер рисунка (см. далее метод `Form1_Load()`). Левый верхний угол объекта имеет координаты (0,0). Область, занимаемая игроком, прямоугольная. Цвет луча лазера определяется свойством `F.bc.LaserColor`. Для доступа к объекту `bc`, расположенному на форме, параметр метода задаем как `Form1 F`.

б) показать игрока

```

public void Show_player(Form1 F, int x, int y)    {
    F.g.ResetClip();
    F.g.FillRegion(new SolidBrush(F.BackColor), reg);
    point.X = x - size.Width / 2;
    point.Y = y;
    Rectangle rec = new Rectangle(point, size);
    reg = new Region(rec);
    F.g.DrawImage(F.imageP, point);
    F.g.ExcludeClip(reg); }

```

Комментарии: Метод имеет параметры: `Form1 F` (для доступа к объекту `g` и свойству `BackColor` – цвету фона), `int x`, `int y` — новые координаты игрока (`x` – ось симметрии). Первый оператор снимает защиту с предыдущей области расположения игрока. Также определяется новая область `reg`. Метод `DrawImage(F.imageP, point)` обеспечивает рисование игрока, метод `ExcludeClip(reg)` обеспечивает защиту заданной области до следующего вызова метода. Совет: Уберите первый и последний операторы и посмотрите на изменения в отображении игрока.

2. Реализация методов класса Bugs

а) генерация одного НЛО

```

public void New_bug(Form1 F, int rch)    {
    Random rv = new Random(rch);
    point.X = rv.Next(10, Form1.ActiveForm.Width - 40);
    point.Y = rv.Next(10, Form1.ActiveForm.Height / 5);
    size.Width = rv.Next(20,50);
    size.Height = size.Width * 2 / 3;
    veloX = rv.Next(7) - 3;
    veloY = rv.Next(3, 10);
    br = F.bc.New_br(rch);
    reg = Form_bug();    }

```

Комментарии: Метод формирует начальные координаты НЛО, его размеры, скорости смещения за 1 такт срабатывания таймера `timer1`, выбирает цвет кисти для его покраски. Везде используется генератор случайных чисел класса `Random`. Для задания области `reg` вызывается следующий метод.

б) задание формы НЛО

```

public Region Form_bug()    {
    Point pt = new Point();
    Size st = new Size();
    pt.X = point.X;
    pt.Y = point.Y+size.Height/4;
    st.Width = size.Width;
    st.Height=size.Height/2;
    Rectangle rec = new Rectangle(pt, st);
    GraphicsPath path1 = new GraphicsPath();
    path1.AddEllipse(rec);
    Region reg= new Region(path1);
    rec.X = point.X + size.Width / 4;
    rec.Y = point.Y;
    rec.Width = size.Width / 2;
    rec.Height = size.Height;
    path1.AddEllipse(rec);
    reg.Union(path1);
    return reg; }

```

Комментарий: Отметим использование объекта класса GraphicsPath и метода reg.Union(path1) — объединение двух эллипсов.

в) вертикальное падение НЛО с горизонтальными смещениями

```

public void Move_bug() {
    point.X += veloX;
    point.Y += veloY;
    reg = Form_bug();    }

```

Комментарий: новая область размещения НЛО заполняется на каждом такте срабатывания таймера 1 после изменения координат через метод Form_bug().

3.Реализация методов класса Enemies

а) настройка серий и создание ссылок на объекты НЛО

```

public void New_Enemies(Form1 F){
    N_generation = 10;
    Delta_N = Form1.N_max / N_generation;
    k_generation = 0;
    N = 0;
    for (int j = 0; j < Form1.N_max; j++)
        bugs[j] = new Bugs();}

```

Комментарий: В методе задается жестко число серий — 10. Вычисляется число НЛО в каждой серии. Обнуляется счетчик числа генераций (серий) и числа активных НЛО. Создаются ссылки на максимально возможное число объектов.

б) удаление сбитых НЛО

```

public int Select_bugs(){
    int k = 0;
    for (int j = 0; j < N; j++) {

```

```

    if (!bugs[j].life)
        k++; }
for (int i = 0; i < k; i++) {
    for (int j = 0; j < N; j++) {
        if (!bugs[j].life) {
            for (int j1 = j; j1 < (N - 1); j1++)
                bugs[j1] = bugs[j1 + 1];
            break; } }
    N--; } return k; } // счетчик подбитых НЛО

```

Комментарии: Если в результате попадания НЛО под луч лазера его свойство life=false, то такой объект удаляется из массива bugs. Сначала определяется общее количество подбитых НЛО, затем в цикле они по очереди удаляются путем сдвига. Количество активных НЛО N также уменьшается. Метод возвращает k — число сбитых НЛО на данном такте.

в) смещение и отображение НЛО

```

public void Show_bugs(Form1 F){
    for (int j = 0; j < N; j++) {
        bugs[j].Move_bug();
        F.g.FillRegion(bugs[j].br, bugs[j].reg); } }

```

Комментарий: Метод в цикле для активных НЛО обеспечивает их смещение и отображение на экране.

г) одна серия НЛО

```

public void Enemy(Form1 F){
    int N0 = N;
    N = N + Delta_N;
    int rch;
    Random rnd = new Random();
    for (int j = N0; j < N; j++) {
        bugs[j] = new Bugs();
        rch = rnd.Next();
        bugs[j].New_bug(F,rch);
        F.g.FillRegion(bugs[j].br, bugs[j].reg); } }

```

Комментарий: Метод добавляет новую серию НЛО, каждый из которых имеет свой цвет, размеры и начальное местоположение.

д) подбитые НЛО, выделены F.bc.KilledBug цветом

```

public void Killed_bugs(Form1 F, int x, int y){
    for (int j = 0; j < N; j++) {
        Rectangle r = new Rectangle(x - bugs[j].size.Width / 2, 0, bugs[j].size.Width, y);
        if (bugs[j].reg.IsVisible(r, F.g) & F.laser) {
            bugs[j].br = new HatchBrush(HatchStyle.DarkHorizontal, F.bc.KilledBug,
            F.bc.KilledBug);
            F.g.FillRegion(bugs[j].br, bugs[j].reg);
            bugs[j].life = false; } } }

```

Комментарий: проверяет попадание объектов под лазерный прицел, отмечает их в свойстве life как false, обеспечивает вспышку НЛО при подрыве.

ПРАКТИЧЕСКАЯ РАБОТА № 33

Тема: Разработка игрового приложения

Цель работы: получение навыков разработки игровых приложений, используя язык C#.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Содержание работы:

Задание 1. Создать простой шутер «Вторжение НЛО» («UFO Invasion»), в котором придется отражать нашествие на Землю армады НЛО.

Открыть приложение, созданное в Практической работе № 32 и продолжить разработку.

1. Реализация методов класса BrushColor

а) кисть для задания цвета НЛО

```
public HatchBrush New_br(int rch){  
    return new HatchBrush (HatchStyle.DashedUpwardDiagonal, DashBug,  
RandomColor(rch));}
```

Комментарий: Метод возвращает кисть класса HatchBrush (шаблон штриховки) из библиотеки System.Drawing.Drawing2D, состоит из одного оператора.

б) случайный цвет

```
public Color RandomColor(int rch) {  
    // rch - случайное число  
    int r, g, b;  
    byte[] bytes1 = new byte[3];    // массив 3 цветов  
    Random rnd1 = new Random(rch);  
    rnd1.NextBytes(bytes1);    // генерация в массив  
    r = Convert.ToInt16(bytes1[0]);  
    g = Convert.ToInt16(bytes1[1]);  
    b = Convert.ToInt16(bytes1[2]);  
    return Color.FromArgb(r, g, b); } // возврат цвета
```

Комментарий: этот метод мы уже использовали ранее в других примерах.

2. Реализация методов класса Form1

а) конструктор формы

```
public Form1(){  
    InitializeComponent();}
```

Комментарий: без изменений.

б) при загрузке формы

```
private void Form1_Load(object sender, EventArgs e){  
    g = this.CreateGraphics();    // инициализация холста  
    BackColor = bc.FonColor;    // цвет фона  
    imageP = new Bitmap(imageList1.Images[0], 100, 100);  
    player.New_player(this);    // инициализация игрока  
    nlo = new Enemies();    // инициализация противника  
    nlo.New_Enemies(this); }    // инициализация НЛО как объектов
```

Комментарии: При загрузке формы задается холст, цвет фона, изображение игрока (прямоугольная область — первое изображение из коллекции `imageList1.Images[]`), инициализируются объекты `imageP`, `player`, `nlo` (все НЛО).

в) Старт игры

```
private void button1_Click(object sender, EventArgs e){
    nlo.k_generation = 0;
    nlo.Enemy(this);
    timer1.Start();
    timer2.Start();
    button3.Enabled = true;
    button1.Enabled = false; }
```

Комментарии: Номер первой серии = 0. Генерация первой серии НЛО — `nlo.Enemy(this)`. Запуск таймеров. Открыть доступ к кнопке «Стоп», закрыть доступ к кнопке «Старт».

г) Включение/отключение лазера игроком

```
private void button2_Click(object sender, EventArgs e){
    if (laser) {
        laser = false;
        button2.Text = "Включить Лазер"; }
    else {
        laser = true;
        button2.Text = "Отключить Лазер"; } }
```

Комментарий: включение/отключение лазера игроком

д) Стоп. Результат

```
private void button3_Click(object sender, EventArgs e){
    timer1.Stop();
    timer2.Stop();
    imageP = new Bitmap(imageList1.Images[1], 100, 100);
    int procent = Result * 100 / (nlo.Delta_N * nlo.N_generation);
    string msg = "Подбито " + Result.ToString() + " НЛО, " + procent.ToString() +
"% результат";
    MessageBox.Show(msg, "Ваш результат", MessageBoxButtons.OK);
    player.Show_player(this, 50, 50);
    nlo.N = 0;
    button1.Enabled = true;
    Result = 0;
    textBox1.Text = Result.ToString(); }
```

Комментарии: Остановка таймеров (игры). Помещение игрока в «гараж» (на красном фоне, вывод второго изображения из коллекции `imageList1.Images[]`), расчет результата игры в процентах, его вывод в окне `MessageBox`, обнуление числа активных объектов НЛО, открытие доступа к кнопке «Старт», обнуление результата. Игра снова может быть запущена без перезапуска приложения.

е) один временной такт

```
private void timer1_Tick(object sender, EventArgs e){
    g.Clear(BackColor);
    Result = Result + nlo.Select_bugs();
    nlo.Show_bugs(this);
    textBox1.Text = Result.ToString();}
```

Комментарии: При срабатывании 1-го таймера производится очистка фона кроме защищенной области под игроком (см. выше метод Show_player(Form1 F, int x, int y)), что позволяет избежать исчезновения игрока. К результату добавляется число сбитых на предыдущем такте НЛО. Отображаются активные (ещё «живые») НЛО и текущий счет игры.

ж) генерация серий

```
private void timer2_Tick(object sender, EventArgs e){
    nlo.k_generation++;
    timer2.Interval -= 100;
    if (nlo.k_generation < nlo.N_generation)
        nlo.Enemy(this);
    else
        timer2.Stop(); }
```

Комментарии: Увеличение номера серии на 1, сокращение на 100 мс интервала выпуска следующей серии, если реализованы все nlo.N_generation, то остановка генерации (timer2.Stop();), иначе генерация очередной серии.

з) попадание НЛО под вертикальный обстрел лазером

```
private void Form1_MouseClick(object sender, MouseEventArgs e){
    player.Show_player(this, e.X, e.Y);
    if (laser)
        g.DrawLine(player.laser_pen, player.point.X + player.size.Width / 2,
        player.point.Y, player.point.X + player.size.Width / 2, 0);
    nlo.Killed_bugs(this,e.X, e.Y);}
```

Комментарии: И, наконец, последний самый главный метод, вызываемый при клике на форме, обеспечивает показ игрока в заданном мышью положении, выстрел из оружия, отметка подбитых НЛО, вспышки при их подрыве.

Интересно, что событие MouseClick демонстрирует взаимодействие объектов трех классов: Form1, Player и Enemies, а опосредовано и всех пяти (+ Bugs и BrushColor).

Запустить игровое приложение на выполнение.

Задание 2. Развить продолжение игры, созданной в Практических работах № 32 и № 33, используя любое направление:

1. Чтобы столкновение игрока с НЛО приводило к уменьшению счета игры вплоть до ее остановки при ранениях, несовместимых с жизнью игрока.
2. Изменение траекторий движения, как игрока, так и противников.
3. Добавление нового вида (класса) противников, отличающихся от безобидных пока НЛО, после чего создайте для них общий класс, от которого они будут наследоваться (принцип наследования).

ПРАКТИЧЕСКАЯ РАБОТА № 34

Тема: Разработка игрового приложения

Цель работы: получение навыков разработки игровых приложений, используя язык C#.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Содержание работы:

Задание 1. Разработать игру «Мозаика». Игра Мозаика похожа на пятнашки, только вместо цифр картинка. Загружается желаемое изображение. Изображение делится на сегменты, сегменты перемешиваются, один сегмент исчезает. Необходимо передвигая сегменты на пустое место собрать картинку полностью.

Исходный код игры Мозаика состоит из двух логических блоков: управление игрой и методы непосредственно с логикой игры. Управление игрой построено на событиях, которые вызывают соответствующие логические методы.

Загрузка картинки. Загрузка изображения осуществляется событием нажатия кнопки в панели инструментов. При этом загружаемая картинка разделяется на установленное количество сегментов, которые первично расположены по порядку.

```
/// <summary>    /// Загрузка картинки.    /// </summary>
```

```
private void ToolStripButtonLoadPicture_Click(object sender, EventArgs e){  
    LoadPicture();}
```

```
/// <summary>/// Загрузка картинки и создание сегментов./// </summary>
```

```
private void LoadPicture(){
```

```
    var ofDlg = new OpenFileDialog();
```

```
    // Фильтр показа изображений с определенным расширением.
```

```
    ofDlg.Filter = "файлы картинок (*.bmp;*.jpg;*.jpeg;*.gif)|";
```

```
    ofDlg.Filter += "/*.bmp;*.jpg;*.jpeg;*.gif|All files (*.*)|*.*";
```

```
    ofDlg.FilterIndex = 1;
```

```
    ofDlg.RestoreDirectory = true;
```

```
    if (ofDlg.ShowDialog() == DialogResult.OK) {
```

```
        Picture = new Bitmap(ofDlg.FileName);    // Загружаем выбранную картинку
```

```
        CreatePictureSegments(); } }    // Создание сегментов
```

Создание сегментов изображения. Сегменты для хранения частей разделенной картинки представляют собой массив элементов класса class PictureBox. В каждом сегменте хранится своя порция изображения. Когда сегменты расположены по порядку изображение визуально выглядит как единое целое. Размер каждого сегмента вычисляется исходя из размеров клиентской части окна и количества сегментов в одном ряду. Каждый сегмент хранит информацию о начальной позиции, чтобы можно было восстановить

картинку. Начальная позиция используется также для определения успешной сборки изображения.

/// <summary>/// Создание сегментов картинки/// </summary>

```
private void CreatePictureSegments(){
    // Удалим предыдущий массив, чтобы создать новый.
    if (pbSegments != null) {
        for (int i = 0; i < pbSegments.Length; i++) {
            pbSegments[i].Dispose();
        }
        pbSegments = null;
    }
    // Создаем массив прямоугольников установленного размера.
    pbSegments = new PictureBox[numRect * numRect];
    // Вычислим габаритные размеры прямоугольников.
    int w = ClientSize.Width / numRect;
    int h = ClientSize.Height / numRect;
    // Счетчики порядкового номера по координатам X и Y.
    int countX = 0; int countY = 0;
    for (int i = 0; i < pbSegments.Length; i++) {
        // Размеры и координаты размещения созданного прямоугольника.
        pbSegments[i] = new PictureBox {
            Width = w, Height = h, Left = countX * w, Top = countY * h
        };
        // Запомним начальные координаты прямоугольника для восстановления
        // перемешанной картинки, и определения полной сборки картинки.
        Point pt = new Point();
        pt.X = pbSegments[i].Left;
        pt.Y = pbSegments[i].Top;
        // сохраним координаты в свойстве Tag каждого прямоугольника
        pbSegments[i].Tag = pt;
        // Считаем прямоугольники по рядам и столбцам.
        countX++;
        if (countX == numRect) {
            countX = 0;
            countY++;
        }
        pbSegments[i].Parent = this;
        pbSegments[i].BorderStyle = BorderStyle.None;
        pbSegments[i].SizeMode = PictureBoxSizeMode.StretchImage;
        // Новые сегменты должны быть все видимы.
        pbSegments[i].Show();
    }
}
```

```
// Для всех прямоугольников массива событие клика мыши
// будет обрабатываться в одной и том же методе.
pbSegments[i].Click += new EventHandler(PB_Click);
} // for (int i = 0; i < pbSegments.Length; i++)
DrawPicture();}
```

Разделение изображения на части. В сформированные сегменты копируется изображение. В соответствии со своей позицией каждый сегмент получает свою долю картинка. Размер части картинка, также как и сегмента, автоматически высчитывается получая данные о количестве сегментов и размере клиентской части окна.

/// <summary>/// Копируем части картинка в соответствующие боксы и боксы рисуют свою порцию изображения./// </summary>

```
private void DrawPicture(){
    if (Picture == null) return;
    int countX = 0;
    int countY = 0;
    for (int i = 0; i < pbSegments.Length; i++)    {
        int w = Picture.Width / numRect;
        int h = Picture.Height / numRect;
        pbSegments[i].Image =
            Picture.Clone(new RectangleF(countX * w, countY * h, w, h),
                Picture.PixelFormat);
        countX++;
        if (countX == numRect)    {
            countX = 0;
            countY++;    }    }    }
```

Обработка события изменения размеров окна. При изменениях размеров окна изменяется и размер клиентской части, в которую вписана картинка. Чтобы изображение корректно отображалось при любом размере окна, габариты сегментов пересчитываются при генерировании события формы FormMain.SizeChanged. Благодаря установленному свойству сегментов SizeMode = StretchImage изображение сохраняет целостный вид при любых изменениях размеров окна приложения.

/// <summary>/// Корректировка размеров сегментов при изменении размеров окна./// </summary>

```
private void CorrectSizeSegments(){
    if (pbSegments == null) return;
    int oldwidth = pbSegments[0].Width;    // Предыдущие размеры сегментов
    int oldheight = pbSegments[0].Height;
```

```

// Новые размеры прямоугольников.
int w = ClientSize.Width / numRect;
int h = ClientSize.Height / numRect;
//int countX = 0; // счетчик прямоугольников по координате X в одном ряду
//int countY = 0; // счетчик прямоугольников по координате Y в одном
столбце
for (int i = 0; i < pbSegments.Length; i++) {
    pbSegments[i].Width = w;
    pbSegments[i].Height = h;
    // Получим порядковый номер сегмента по координате X
    int countX = pbSegments[i].Left /= oldwidth;
    // Получим порядковый номер сегмента по координате Y
    int countY = pbSegments[i].Top /= oldheight;
    pbSegments[i].Left = countX * w;
    pbSegments[i].Top = countY * h; } }

```

Рандомное перемешивание сегментов. Чтобы игра Мозаика началась необходимо перемешать сегменты картинки. Для данного типа игры достаточно использовать генератор псевдослучайных чисел class Random . Перемешивание заключается в рандомном обмене позиций расположения сегментов. Но благодаря тому, что в свойстве Tag сегмента хранится начальная позиция прямоугольника очень легко восстановить изображение вновь.

/// <summary>/// Перемешивание сегментов/// </summary>

```

private void MixedSegments(){
    if (Picture == null) return;
    // Создаем объект генерирования псевдослучайных чисел, для различного
набора случайных чисел инициализацию объекта Random производим от
счетчика количества миллисекунд прошедших со времени запуска
операционной системы.

```

```

Random rand = new Random(Environment.TickCount);

```

```

for (int i = 0; i < pbSegments.Length; i++) {
    pbSegments[i].Visible = true;
    int temp = rand.Next(0, pbSegments.Length);
    Point ptR = pbSegments[temp].Location;
    Point ptI = pbSegments[i].Location;
    pbSegments[i].Location = ptR;
    pbSegments[temp].Location = ptI;
    // Бордюр чтобы видно было прямоугольники
    pbSegments[i].BorderStyle = BorderStyle.Fixed3D; }

```

// Случайным образом выбираем пустой прямоугольник, делаем его невидимым.

```
int r = rand.Next(0, pbSegments.Length);  
pbSegments[r].Visible = false;}
```

Проверка полной сборки картинки. После каждого хода происходит проверка на полную сборку изображения. Как уже было описано выше, каждый сегмент хранит информацию о своей начальной позиции. При перемещении прямоугольника на новую позицию сравниваются текущая позиция и начальная. Если эти позиции совпадают значит картинка полностью собрана. В этом случае на сегментах исчезает бордюр, пустое место для перемещения заполняется недостающей частью изображения, игра заканчивается.

// После каждого хода проверка на полную сборку картинки.

//***** блок проверки *****

// Если хоть у одного прямоугольника не совпадают

// реальные координаты и первичные заканчиваем

// проверку и выходим из метода.

```
for (int j = 0; j < pbSegments.Length; j++){
```

```
    Point point = (Point)pbSegments[j].Tag;
```

```
    if (pbSegments[j].Location != point) {
```

```
        return;    }}
```

// Если у всех прямоугольников совпали реальные и первичные

// координаты - картинка собрана!

```
for (int m = 0; m < pbSegments.Length; m++){
```

// Делаем видимыми все сегменты картинки.

```
pbSegments[m].Visible = true;
```

// Убираем обрамление прямоугольников.

```
pbSegments[m].BorderStyle = BorderStyle.None; }
```

//***** окончание блока проверки *****

ПРАКТИЧЕСКАЯ РАБОТА № 35

Тема: Оптимизация и рефакторинг кода

Цель работы: получение практических навыков по оптимизации и рефакторингу кода программ.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Справочный материал:

При создании приложений многие разработчики сначала заботятся об их функциональности, а когда она обеспечена, переделывают приложения таким образом, чтобы они были более управляемыми и удобочитаемыми. Это называется рефакторингом (refactoring). Под рефакторингом понимается процесс переделки кода для повышения удобочитаемости и производительности приложений, а также обеспечения безопасности типов и приведения кода к такому виду, в котором он лучше соответствует рекомендуемым приемам объектно-ориентированного программирования. За счет использования меню Refactor (Рефакторинг), которое становится доступным при открытом файле кода, а также соответствующих клавиатурных комбинаций быстрого вызова, смарт-тегов (smart tags) и/или вызывающих контекстные меню щелчков, можно существенно видоизменять код с минимальным объемом усилий.

Содержание работы:

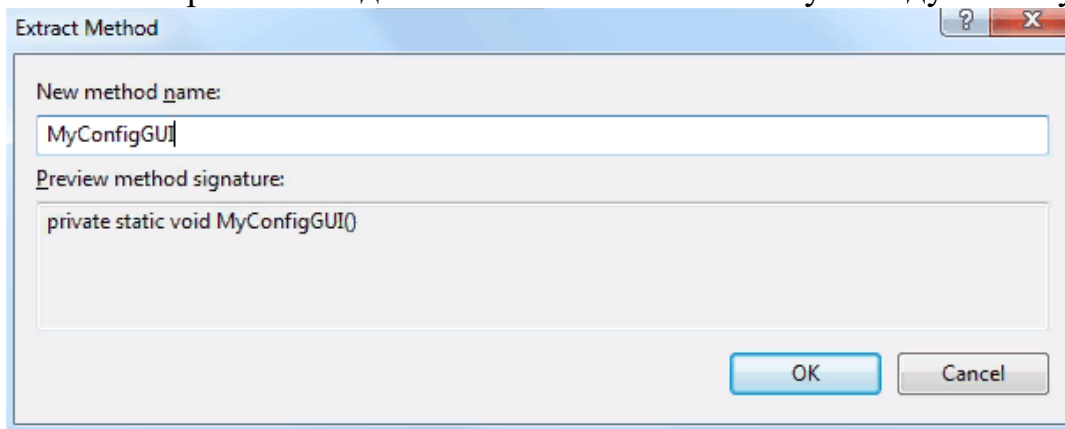
Задание 1. Модифицировать метод Main(), добавив в него следующий код:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1 {
    class Program {
        static void Main(string[] args) {
// Настраиваем консольный интерфейс (CUI)
            Console.Title = "Мое приложение";
            Console.ForegroundColor = ConsoleColor.Yellow;
            Console.BackgroundColor = ConsoleColor.Blue;
            Console.WriteLine("Привет, это мой проект!");
            Console.BackgroundColor = ConsoleColor.Black;
            Console.ReadLine();    } } }
```

В таком, как он есть виде, в этом коде нет ничего неправильного, но давайте представим, что возникло желание сделать так, чтобы данное приветственное сообщение отображалось в различных местах по всей программе. В идеале вместо того, чтобы заново вводить ту же самую отвечающую за настройку консольного интерфейса логику, было бы неплохо иметь вспомогательную функцию, которую можно было бы вызывать для решения этой задачи. С учетом этого, попробуем применить к существующему коду прием рефакторинга Extract Method (Извлечение метода).

Для этого выделите в окне редактора все содержащиеся внутри Main() операторы, кроме последнего вызова Console.ReadLine(), и щелкните на выделенном коде правой кнопкой мыши. Выберите в контекстном меню пункт Refactor --- Extract Method (Рефакторинг --- Извлечь метод).

В открывшемся далее окне назначьте новому методу имя MyConfigCUI():



После этого метод Main() станет вызывать новый только что сгенерированный метод MyConfigCUI(), внутри которого будет содержаться выделенный ранее код:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1 {
    class Program {
        static void Main(string[] args) {
            //Настраиваем консольный интерфейс (CUI)
            MyConfigCUI();
            Console.ReadLine();
        }
        private static void MyConfigCUI() {
            Console.Title = "Мое приложение";
            Console.ForegroundColor = ConsoleColor.Yellow;
            Console.BackgroundColor = ConsoleColor.Blue;
            Console.WriteLine("Привет, это мой проект!");
            Console.BackgroundColor = ConsoleColor.Black;
        }
    }
}
```

Задание 2. Выберите любую созданную ранее программу и оптимизируйте.

ПРАКТИЧЕСКАЯ РАБОТА № 36

Тема: Оптимизация и рефакторинг кода

Цель работы: получение практических навыков по оптимизации и рефакторингу кода программ.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Содержание работы:

Задание 1. Изучить код программы, исправить при необходимости ошибки и оптимизировать данный код.

Условие задачи: Программа на C#, которая создаёт два объекта, после нажатия на кнопку Запуск 1 объекта, начинает двигаться 1 объект по вытянутой параболе. Затем нажимаем на Запуск 2 объекта, он должен догнать первый.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace laba_36{
public partial class Form1 : Form {
bool l = false;
MyLabel1 D;
MyLabel T;
static public arch d = new arch();
public class MyLabel1{
public System.Windows.Forms.Label m3;
public System.Windows.Forms.Timer MyTimer1;
double y0;
double x0;
double M;
public MyLabel1(Control Sender){
m3 = new Label();
m3.Parent = Sender;
m3.Size = new System.Drawing.Size(100, 100);
m3.Top = 190;
m3.BackColor = System.Drawing.SystemColors.Desktop;
MyTimer1 = new Timer();
MyTimer1.Interval = 1;
MyTimer1.Tick += new System.EventHandler(movett);}
public double x1{
set{
x0 = value;
double Mhor = m3.Parent.Width / 3000.0;
```



```

m3.Left = System.Convert.ToInt32(x0 * Mhor);}
get{
return x0;}}
public double y1{
set{
y0 = value;
double Mhor = m3.Parent.Width / 3000.0;
m3.Top = -System.Convert.ToInt32(y0 * Mhor) + m3.Parent.Height / 2 - m3.Height;}
get{
return y0;}}
public void movett(object sender, EventArgs e){
if (d.archx >= x1) x1 = x1 + 2;
else x1 = x1 - 2;
if (d.archy >= y1) y1 = y1 + 2;
else y1 = y1 - 2;}}
public class arch{ //класс архив нужен чтобы второй объект получал координаты
от первого
public double archx;
public double archy;}
    public class MyLabel{
double y0;
double x0;
double a = 10;
double k = 0.0003;
public System.Windows.Forms.Label m;
public System.Windows.Forms.Label m1;
public System.Windows.Forms.Label m2;
public System.Windows.Forms.Timer MyTimer;
    public MyLabel(Control Sender) {//конструктор класса
m = new Label();
m.BackColor = System.Drawing.SystemColors.GradientActiveCaption;
m.Parent = Sender;
m.Text = " 1 ";
m.Click += new System.EventHandler(this.move);
m.Left = 150;
m.Top = 190;
m.Width = 100;
m.Height = 30;
    m1 = new Label();
m1.BackColor = System.Drawing.SystemColors.GradientActiveCaption;
m1.Parent = Sender;
m1.Text = " 2 ";
m1.Click += new System.EventHandler(this.move);
m1.Top = 190;
m1.Left = m.Width / 2 - (m1.Width / 2);

```

```

m1.Left = 4;
m1.Width = 50;
m1.Height = 30;
m2 = new Label();
m2.BackColor = System.Drawing.SystemColors.GradientActiveCaption;
m2.Parent = Sender;
m2.Text = " 3 ";
m2.Click += new System.EventHandler(this.move);
m2.Width = 25;
m2.Left = m.Width / 2 - (m2.Width / 2);
m2.Left = 10;
m2.Top = 180;
m2.Width = 25;
m2.Height = 30;
MyTimer = new Timer();
MyTimer.Interval = 2;
MyTimer.Tick += new System.EventHandler(this.move1);}
public void move(object sender, EventArgs e){
x = x + 1; y = y + 1;}
public void move1(object sender, EventArgs e){
xcor = x + a;
d.archx = xcor;
k = 0.0003;
ycor = k * Math.Pow(x, 2);
d.archy = ycor;}
public double xcor{
set{
x0 = value;
double Mhor = m.Parent.Width / 3000.0;
m.Left = System.Convert.ToInt32(x0 * Mhor);
m1.Left = m.Left + m.Width / 2 - (m.Width / 4);
m2.Left = m.Left + m.Width / 2 - (m.Width / 10);}
get{
return x0;}}
public double ycor{
set{
y0 = value;
double Mhor = m.Parent.Height / 3000.0;
m.Top = (m.Parent.Height - (m.Height + 30)) - System.Convert.ToInt32(y0 * Mhor);
m1.Top = m.Top - m.Height;
m2.Top = m.Top - m.Height - m1.Height;}
get{
return y0;}}
public double x{ //класс координация меток на форме по X
set{

```

```

x0 = value;
double Mhor = m.Parent.Width / 3000.0;
m.Left = System.Convert.ToInt32(x0 * Mhor);
m1.Left = m.Left + m.Width / 2 - (m.Width / 4);
m2.Left = m.Left + m.Width / 2 - (m.Width / 10);}
get{
return x0;}}
public double y{           //класс координация меток на форме по У
set{
y0 = value;
double Mhor = m.Parent.Height / 3000.0;
m.Top = (m.Parent.Height - (m.Height + 30)) - System.Convert.ToInt32(y0 * Mhor);
m1.Top = m.Top - m.Height;
m2.Top = m.Top - m.Height - m1.Height;}
get{
return y0;} }
// public void movet(object sender, EventArgs e)// {
// xcor = x + 1;    // ycor = y + 1;    // }}
public Form1(){
InitializeComponent();}
private void Form1_Load(object sender, EventArgs e){           }
private void button1_Click(object sender, EventArgs e)
{
    // d = new arch(this);
    T = new MyLabel(this);
    T.x = 0;
    T.y = 0;
    D = new MyLabel1(this);}
private void button2_Click(object sender, EventArgs e){
if (l == false) l = true;
else l = false;
T.MyTimer.Enabled = 1;}
private void button3_Click(object sender, EventArgs e){
if (l == false) l = true;
else l = false;
D.MyTimer1.Enabled = 1;}}}

```

ПРАКТИЧЕСКАЯ РАБОТА № 37

Тема: Оптимизация и рефакторинг кода

Цель работы: получение практических навыков по оптимизации и рефакторингу кода программ.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Содержание работы:

Задание 1. Оптимизируйте код игрового приложения «Вторжение НЛО»

Задание 2. Оптимизируйте код программы, написанной в Практической работе № 6 по созданию класса, чтобы он был создан в Windows Form.

ПРАКТИЧЕСКАЯ РАБОТА № 38

Тема: Разработка интерфейса пользователя

Цель работы: сформировать умения использования компонентов стандартных диалогов и системы меню, изучить основные свойства; сформировать умения по созданию процедур на основе событий компонентов.

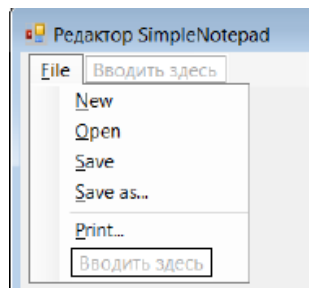
Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Справочный материал:

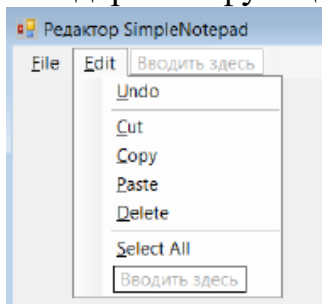
Содержание работы:

Задание 1. Разработать приложение SimpleNotepad, представляющее собой простейший текстовый редактор. Использовать в приложении компонент для работы с меню и стандартные окна диалога.

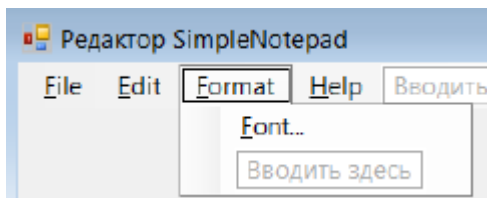
1. Запустите среду программирования VisualStudio. Создайте новое Приложение WindowsForms. Имя проекта и приложения SimpleNotepad.
2. Измените свойство Name формы на SimpleNotepadForm.
3. Перейдите в окно Обозреватель решений и переименуйте файл Form1.cs на SimpleNotepadForm.cs.
4. Измените размеры окно формы. Измените заголовок окна на РедакторSimpleNotepad.
5. Добавьте в окно приложения элемент управления MenuStrip.
6. В поле Вводить здесь меню введите строку &File. В результате в окне приложения появится меню File. Обратите внимание, что первая буква в названии подчеркнута. Это получилось потому, что перед ней стоит префикс &. Этим префиксом отмечается буква, предназначенная для ускоренного выбора меню при помощи клавиатуры.
7. Создайте меню File в соответствии с рисунком. Для вставки разделительной черты необходимо вызвать контекстное меню и выбрать команду Separator.



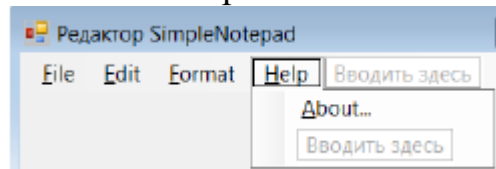
8. Создайте меню Edit в соответствии с рисунком. В меню Edit реализованы стандартные функции приложения Блокнот.



9. Меню Format состоит только из одной строки Font, с помощью которой пользователь может изменить шрифт текста



10. Меню Help состоит только из одной строки About.



11. Переименуйте идентификаторы меню и строки меню таким образом, чтобы с ними было удобнее работать в программе. Для этого в окне дизайнера формы щелкните правой кнопкой мыши главное меню приложения и затем выберите из контекстного меню строку Правка DropDownItems. В окне Редактор коллекций элементов отредактируйте имена меню и строки меню, изменив значение свойства Name соответствующего элемента. При этом меню верхнего уровня должны называться menuFile, menuEdit, menuFormat, menuHelp. Имена строк формируются путем добавления к имени меню текста, отображаемого в строке меню. Например, строка New называется menuFileNew.

12. Запустите приложение. Убедитесь, что меню корректно отображается, и вы можете выбирать его строки.

13. Редактирование текста в приложении будет выполнять компонент RichTextBox. Перетащите компонент RichTextBox панели элементов в окно приложения. Задайте значение свойства Dock данного компонента равным Fill, для того чтобы он занимал все окно приложения.

14. Создайте обработчики событий, необходимые для выполнения таких функций как создание нового документа, открытие и сохранение документа.

15. Добавьте на форму компонент OpenFileDialog с вкладки Диалоговые окна панели элементов. Выделите пункт меню Open... и дважды щелкните по нему левой кнопкой мыши, для того чтобы добавить обработчик события. В обработчик события добавьте следующий код для отображения диалогового окна открытия файла.

```
private void menuFileOpen_Click(object sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK && openFileDialog1.FileName.Length > 0)
    {
        try
        {
            richTextBox1.LoadFile(openFileDialog1.FileName, RichTextBoxStreamType.RichText);
        }
        catch (System.ArgumentException ex)
        {
            richTextBox1.LoadFile(openFileDialog1.FileName, RichTextBoxStreamType.PlainText);
        }
        this.Text = "Файл[" + openFileDialog1.FileName + "];
    }
}
```

16. Выделите компонент openFileDialog1, на панели Свойства найдите свойство Filter и введите следующую строку RTF files|*.rtf|Text files|*.txt|All files|*.*

17. Добавьте на форму компонент SaveFileDialog с вкладки Диалоговые окна

панели элементов. Выделите пункт меню SaveAs... и дважды щелкните по нему левой кнопкой мыши, для того чтобы добавить обработчик события. В обработчик события добавьте следующий код для отображения диалогового окна Сохранение файла. Этот код сохраняет текст введенный в элемент управления richTextBox, в текстовый файл в указанной папке. Метод ShowDialog отображает диалоговое окно, а затем с помощью поля DialogResult.OK осуществляется проверка нажата ли пользователем кнопка ОК.

```
private void menuFileSave_Click(object sender, EventArgs e)
{
    if (saveFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK && saveFileDialog1.FileName.Length > 0)
    {
        richTextBox1.SaveFile(saveFileDialog1.FileName);
        this.Text = "Файл [" + saveFileDialog1.FileName + "]";
    }
}
```

18. Выделите компонент saveFileDialog1, на панели Свойства найдите свойство Filter и введите следующую строку RTFfiles|*.rtf, найдите свойство FileName и задайте значение doc1.rtf

19. Добавьте в окно дизайнера форм компоненты PrintDocumet, PrintDialog вкладки Печать панели элементов.



20. Компонент PrintDocumet предназначен для вывода данных документа на принтер. Свойства компонента PrintDocumet описывают, как именно нужно распечатывать документ. В свойство DocumentName данного компонента запишите строку SimpleNotepadDocument. Эта строка будет идентифицировать документ при отображении состояния очереди печати. Значения остальных свойств оставьте без изменения.

21. С помощью компонента PrintDialog приложение выведет на экран стандартное диалоговое окно печати документа. Задайте для данного компонента значение свойства Document равным printDocument1. Этим обеспечивается связь компонента printDialog с компонентом PrintDocumet.

22. Для работы с классами, предназначенными для выполнения операций с потоками и печати, добавьте в начало программы следующие строки:

```
using System.IO;
using System.Drawing.Printing;
```

23. Добавьте в класса SimpleNotepadForm поля m_myReader (для печати содержимого редактора текста) и m_PrintPageNumber (номер текущей распечатываемой страницы документа).

```
public partial class SimpleNotepadForm : Form
{
    private StringReader m_myReader;
    private uint m_PrintPageNumber;

    public SimpleNotepadForm()
    {

```

24. Создайте обработчик события печати документа.

```
private void menuFilePrint_Click(object sender, EventArgs e)
{
    m_PrintPageNumber = 1;
    string strText = this.richTextBox1.Text;
    m_myReader = new StringReader(strText);
    Margins margins = new Margins(100, 50, 50, 50);
    printDocument1.DefaultPageSettings.Margins = margins;
    if (printDialog1.ShowDialog() == DialogResult.OK)
    {
        this.printDocument1.Print();
    }
    m_myReader.Close();
}
```

Печать документа будет начинаться с первой страницы, поэтому в поле `m_PrintPageNumber` записывается значение 1. Далее выполняется чтение текущего содержимого окна редактирования текста в поток `m_myReader` класса `StringReader`. Далее задаются границы отступов на распечатываемой странице и отображается диалоговое окно печати документа. Если пользователь щелкает в этом окне кнопку ОК, документ `printDocument1` отправляется на печать методом `Print`. Далее ненужный более поток `m_myReader` закрывается методом `Close`.

25. На данном этапе приложение еще не в состоянии распечатать документ. Причина этого в том, что приложение пока еще не знает, каким именно образом нужно печатать документ.

26. Чтобы в нашем приложении заработала функция печати, необходимо создать обработчик события `PrintPage`. Для этого нужно дважды щелкнуть левой клавишей мыши значок компонента `printDocument1`. В тело обработчика событий вставьте программный код из текстового файла `Print.txt`. Комментарии в тексте обработчика событий `PrintPage` поясняют назначение отдельных программных строк. Заметим, что для полного понимания действий, выполняемых нашим обработчиком событий, требуется предварительное знакомство с графической подсистемой `Graphics Device Interface Plus (GDI+)`, реализованной компанией `Microsoft` в рамках библиотеки классов `.NET Framework`. Пока же нужно отметить, что приложение распечатывает текст построчно в цикле. После завершения печати всех строк текущей страницы обработчик событий `PrintPage` печатает верхний и нижний колонтитулы, а также рисует горизонтальные линии, отделяющие текст колонтитулов от текста документа.

ПРАКТИЧЕСКАЯ РАБОТА № 39

Тема: Разработка интерфейса пользователя

Цель работы: сформировать умения использования компонентов стандартных диалогов и системы меню, изучить основные свойства; сформировать умения по созданию процедур на основе событий компонентов

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Содержание работы:

Задание 1. Разработать приложение SimpleNotepad, представляющее собой простейший текстовый редактор. Использовать в приложении компонент для работы с меню и стандартные окна диалога.

Откройте проект, начатый в практической работе № 38. Продолжите дорабатывать его

1. Измените меню File, добавив пункт Exit, при выборе которого окно редактора текста должно быть закрыто. Добавьте обработчик события для данного пункта меню:

```
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.Close();
}
```

2. Однако при закрытии окно приложения возникает проблема: окно редактора текста будет закрыто и в том случае, если пользователь не сохранил сделанные им изменения. Чтобы решить эту проблему, нам нужно каким-то образом отслеживать наличие изменений в окне редактирования текста. Определите в классе SimpleNotepadForm поле m_DocumentChanged, в котором будет храниться флаг, отмечающий изменения, сделанные пользователем в документе. В новом или только что загруженном документе изменений нет, поэтому начальное значение этого флага равно false.

```
private bool m_DocumentChanged = false;
```

3. Создайте обработчик события richTextBox1_TextChanged, выполнив двойной щелчок по компоненту richTextBox. Этот обработчик получит управление, как только пользователь внесет любые изменения в содержимое редактируемого документа.

```
private void richTextBox1_TextChanged(object sender, EventArgs e)
{
    m_DocumentChanged = true;
}
```

4. Если пользователь редактировал документ, а потом решил создать новый, выбрав из меню File строку New, изменения, внесенные в старый документ, могут быть потеряны.

5. Чтобы избежать этого, проверьте флаг m_DocumentChanged перед тем как очищать содержимое редактора текста. Если в редакторе есть не сохраненные изменения, необходимо выполнить сохранение документа. Отредактируйте код обработчика события пункта меню New следующим образом:

```
private void menuFileNew_Click(object sender, EventArgs e)
{
    if (m_DocumentChanged)
    {
        if (saveFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK && saveFileDialog1.FileName.Length > 0)
        {
            richTextBox1.SaveFile(saveFileDialog1.FileName);
            this.Text = "Файл [" + saveFileDialog1 + "]";
            m_DocumentChanged = false;
        }
    }
    richTextBox1.Clear();
}
```

6. Отредактируйте код обработчика события пунктов меню Save и Save As, добавив строку кода m_DocumentChanged=false

7. Измените обработчик события пункта меню Exit следующим образом:

```
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (saveFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK && saveFileDialog1.FileName.Length > 0)
    {
        richTextBox1.SaveFile(saveFileDialog1.FileName);
        this.Text = "Файл [" + saveFileDialog1 + "]";
        m_DocumentChanged = false;
    }
    this.Close();
}
```

8. Надо выполнить еще одну проверку флага m_DocumentChanged в методе Dispose, который вызывается при закрытии окна приложения. Для этого на панели Обозреватель решений выделите SimpleNotepadForm.Designer.cs, найдите метод Dispose и внесите необходимые изменения.

```
protected override void Dispose(bool disposing)
{
    if (saveFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK && saveFileDialog1.FileName.Length > 0)
    {
        richTextBox1.SaveFile(saveFileDialog1.FileName);
        this.Text = "Файл [" + saveFileDialog1 + "]";
        m_DocumentChanged = false;
    }
    if (disposing && (components != null))
    {
        components.Dispose();
    }
    base.Dispose(disposing);
}
```

Теперь, когда пользователь попытается закрыть программу с помощью соответствующей кнопки заголовка окна, не сохранив сделанные изменения, на экране появится стандартное диалоговое окно, предлагающее ему сохранить документ.

9. Создайте обработчики событий пунктов меню Edit. Реализация данных функций является простой, поскольку элемент управления RichTextBox имеет все необходимые методы. Добавьте в пункт меню Edit строку Redo.

10. Подготовьте обработчики событий для всех строк меню Edit.

```
private void menuEditUndo_Click(object sender, EventArgs e)
{
    richTextBox1.Undo();
}

private void menuEditRedo_Click(object sender, EventArgs e)
{
    richTextBox1.Redo();
}

private void menuEditCut_Click(object sender, EventArgs e)
{
    richTextBox1.Cut();
}

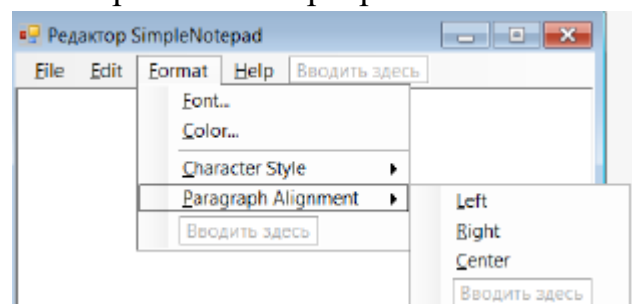
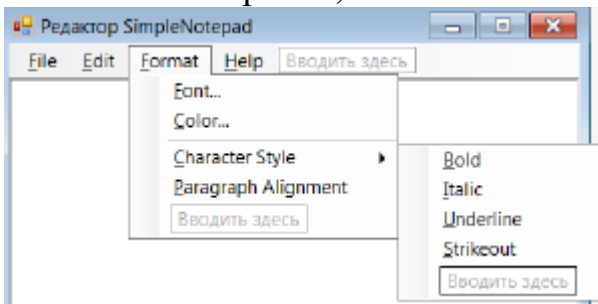
private void menuEditCopy_Click(object sender, EventArgs e)
{
    richTextBox1.Copy();
}

private void menuEditPaste_Click(object sender, EventArgs e)
{
    richTextBox1.Paste();
}

private void menuEditDelete_Click(object sender, EventArgs e)
{
    richTextBox1.Cut();
}

private void menuEditSelectAll_Click(object sender, EventArgs e)
{
    richTextBox1.SelectAll();
}
```

11. Измените меню Format, добавив в него строку Color, а также два меню второго уровня – CharacterStyle и ParagraphAlignment. Измените имена строк меню таким образом, чтобы с ними было легче работать в программе.



12. Добавьте на форму компонент FontDialog с вкладки Диалоговые окна панели элементов. Этот компонент отображает на экране стандартное диалоговое окно выбора шрифта. Создайте обработчик события пункта меню Font и вставьте следующий код:

```
private void menuFormatFont_Click(object sender, EventArgs e)
{
    if (FontDialog1.ShowDialog() == DialogResult.OK)
    {
        richTextBox1.SelectionFont = fontDialog1.Font;
    }
}
```

После того как пользователь выбрал нужный ему шрифт, обработчик события переписывает этот шрифт из свойства fontDialog1.Font в свойство richTextBox1.SelectionFont. Свойство SelectionFont позволяет изменить шрифт фрагмента текста, выделенного пользователем (или программой) в

окне редактирования.

13. Добавьте на форму компонент `ColorDialog` с вкладки Диалоговые окна панели элементов. Этот компонент отображает на экране стандартное диалоговое окно выбора шрифта. Создайте обработчик события пункта меню `Color` и вставьте следующий код:

```
private void colorToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (colorDialog1.ShowDialog() == DialogResult.OK)
    {
        richTextBox1.SelectionColor = colorDialog1.Color;
    }
}
```

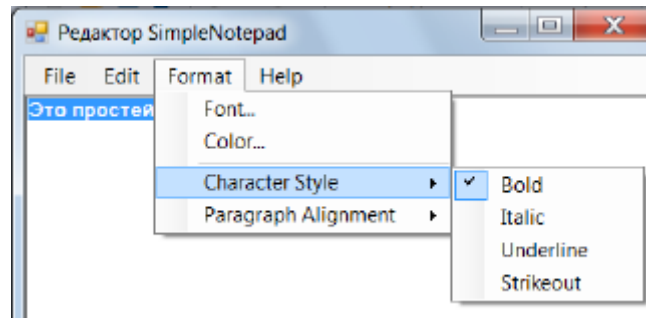
14. Создайте обработчик события для пункта меню `Bold` и вставьте следующий код:

```
private void menuFormatFontCharacterStyleBold_Click(object sender, EventArgs e)
{
    if (richTextBox1.SelectionFont != null)
    {
        System.Drawing.Font currentFont = richTextBox1.SelectionFont;
        System.Drawing.FontStyle newFontStyle;
        if (richTextBox1.SelectionFont.Bold == true)
        {
            newFontStyle = FontStyle.Regular;
        }
        else
        {
            newFontStyle = FontStyle.Bold;
        }
        richTextBox1.SelectionFont = new Font(currentFont.FontFamily, currentFont.Size, newFontStyle);
        CheckMenuFontCharacterStyle();
    }
}
```

Получив управление данный метод прежде всего, определяет шрифт фрагмента текста, выделенного пользователем, анализируя свойство `richTextBox1.SelectionFont`. Если шрифт определить не удалось, и это свойство содержит значение `null`, программа не делает никаких изменений. В противном случае программа сохраняет текущий шрифт в переменной `currentFont` класса `System.Drawing.Font`. Далее метод проверяет, был ли выделен фрагмент текста жирным шрифтом, анализируя свойство `richTextBox1.SelectionFont.Bold`. Если это свойство содержит значение `true`, то метод снимает выделение, если нет, то устанавливает его. Для снятия выделения программа записывает в переменную `newFontStyle` значение `FontStyle.Regular`, а для установки— значение `FontStyle.Bold`.

15. При выделении фрагмента текста жирным шрифтом в меню одновременно отмечается «галочкой» строка `Bold`. Для этого необходимо создать `CheckMenuFontCharacterStyle`. Для этого перейдите в окно кода и вставьте программный код процедуры из текстового документа `CheckMenu.txt`. Метод `CheckMenuFontCharacterStyle` по очереди проверяет стилевое оформление выделенных фрагментов, отмечая «галочками» соответствующие строки меню `CharacterStyle` или снимая со строк этого меню отметки.

16. Запустите программу и протестируйте её работу.



17. Аналогично создайте обработчики для пунктов меню Italic, Underline, Strikeout.

18. Подготовьте обработчики событий для строк меню Paragraph Alignment следующим образом:

```
private void menuFormatParagraphAlignmentLeft_Click(object sender, EventArgs e)
{
    richTextBox1.SelectionAlignment = HorizontalAlignment.Left;
}

private void menuFormatParagraphAlignmenRight_Click(object sender, EventArgs e)
{
    richTextBox1.SelectionAlignment = HorizontalAlignment.Right;
}

private void menuFormatParagraphAlignmentCenter_Click(object sender, EventArgs e)
{
    richTextBox1.SelectionAlignment = HorizontalAlignment.Center;
}
```

Все эти обработчики событий строк меню ParagraphAlignment изменяют значение свойства richTextBox1.SelectionAlignment, задающего выравнивание параграфа текста, выделенного пользователем (для выделения параграфа с целью изменения выравнивания достаточно установить в него текстовый курсор).

19. Сохраните изменения и протестируйте работу приложения.

ПРАКТИЧЕСКАЯ РАБОТА № 40

Тема: Разработка интерфейса пользователя

Цель работы: сформировать умения использования компонентов стандартных диалогов и системы меню, изучить основные свойства; сформировать умения по созданию процедур на основе событий компонентов

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.


Содержание работы:

Задание 1. Создание панели инструментов приложения из Практической работе № 39.


1. Чтобы добавить инструментальную панель в окно приложения, перетащите мышью элемент управления ToolStrip вкладки Все формы Windows Forms панели элементов в окно проектирования формы нашего приложения. По умолчанию окно инструментальной панели появится в верхней части формы. В только что добавленной панели нет ни одной кнопки.

2. Сразу после добавления окно инструментальной панели будет расположено над окном редактора текста. Чтобы исправить это положение, щелкните правой кнопкой мыши окно редактора текста, а затем выберите из контекстного меню строку На передний план. В результате окна примут правильное взаимное расположение.

3. Создайте в папке проекта отдельную папку для хранения изображений инструментальной панели, назвав ее, например, ImageList. Далее скопируйте в созданную папку изображения, подготовленные для кнопок панели.

4. Выделите компонент ToolStrip и на панели свойств найдите свойство Items, щелкните по кнопке . Откроется окно редактора коллекций.

5. В диалоговом окне Редактор коллекции элементов в списке элементов выберите Button и нажмите кнопку Добавить. На панели свойств измените значение свойства Name данной кнопки на создатьToolStripButton, а размер кнопки установите равным 24.

6. В окне свойств найдите свойство Image. Нажмите на кнопку , в диалоговом окне Выбор ресурса нажмите на кнопку Импорт и выберите соответствующее изображение из сохранённых ранее в папке ImageList.


7. Задайте комментарий-подсказку для кнопки создатьToolStripButton, установив свойство ToolTipText равным Создать.

8. Аналогично создайте кнопки открытьToolStripButton, сохранитьToolStripButton, печатьToolStripButton.

9. Далее необходимо создать разделитель между группами кнопок, ответственных за выполнение различных групп операций. Для этого в списке элементов выберите Separator и нажмите кнопку Добавить.

10. Далее аналогично создайте кнопки вырезатьToolStripButton, копироватьToolStripButton, вставитьToolStripButton.

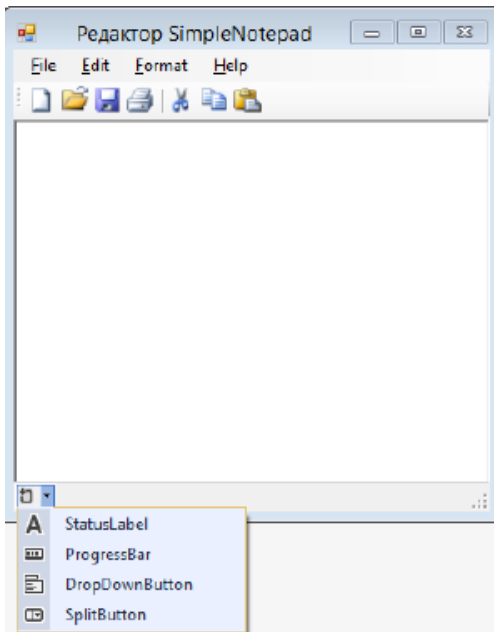
11. Теперь необходимо связать созданные ранее функции-обработчики события команд пункт меню с соответствующими кнопками панели инструментов. Выделите кнопку создатьToolStripButton, на панели Свойства перейдите

на вкладку События , найдите событие Click и в списке выберите функцию menuFileNew_Click. Аналогично добавьте функции для остальных кнопок панели инструментов.

12. Сохраните приложение. Запустите его и протестируйте работу кнопок панели инструментов.

Задание 2. Создание строки состояния

1. Разместите на форме компонент StatusStrip со вкладки Меню и панели инструментов панели элементов.



2. В меню компонента StatusStrip выберите опцию StatusLabel.

3. Далее на панели Свойства найдите свойство Text и сделайте его пустым.

4. Для того чтобы отслеживать выбор строк меню необходимо предусмотреть специальный обработчик события DropDownItemClicked. Это событие создается строками меню, когда пользователь выбирает их при помощи мыши

или клавиатуры. Создайте обработчик событий. Для этого выделите мышью меню File в окне дизайнера формы, а затем перейдите на вкладку События панели Свойства. В списке событий найдите событие

DropDownItemClicked и дважды щелкните по

строке рядом с ним. После этого будет создано пустое тело обработчика события. Вставьте следующий код в обработчик события:

```
private void menuFile_DropDownItemClicked(object sender, ToolStripItemClickedEventArgs e)
{
    this.UpdateStatus(e.ClickedItem);
}
```

5. Создайте пустую строку перед обработчиком данного события и опишите метод UpdateStatus следующим образом:

```
private void UpdateStatus(ToolStripItem item)
{
    if (item != null)
    {
        string msg = String.Format("{0} selected", item.Text);
        this.statusStrip1.Items[0].Text = msg;
    }
}
```

6. Выделите мышью меню Edit в окне дизайнера формы, а затем перейдите на вкладку События панели Свойства. В списке событий найдите событие DropDownItemClicked и в списке рядом с ним выберите событие menuFile_DropDownItemClicked. Аналогично выполните для всех остальных пунктов меню.

7. Сохраните приложение. Запустите его и протестируйте работу строки состояния

ПРАКТИЧЕСКАЯ РАБОТА № 41

Тема: Разработка интерфейса пользователя

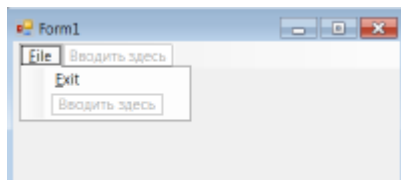
Цель работы: сформировать умения использования компонентов стандартных диалогов и системы меню, изучить основные свойства; сформировать умения по созданию процедур на основе событий компонентов

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Содержание работы:

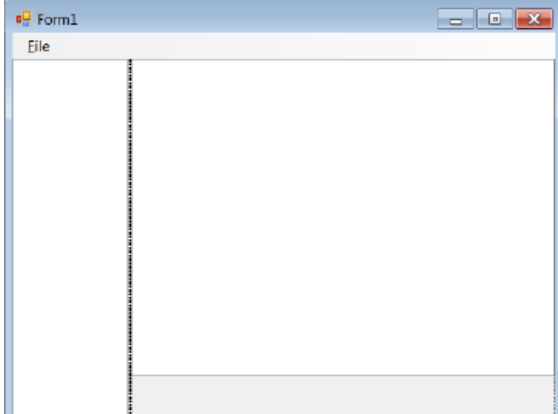
Задание 1. Создания многооконного приложения с фреймами.

1. Запустите среду программирования VisualStudio. Создайте новое Приложение WindowsForms. Имя проекта и приложения FramesApp.
2. Добавьте в форму главного окна приложения меню (MenuStrip) и строку состояния (StatusStrip). В меню File создайте строку Exit, предназначенную для завершения работы приложения. Напишите обработчик данного события.



3. Добавьте в окно нашего приложения элемент управления TreeView, перетащив мышью его значок из панели элементов вкладки Все формы WindowsForms в окно формы. Установите значение свойства Dock данного элемента управления равным Left. В результате элемент управления TreeView будет выровнен по левой границе главного окна приложения. В нашем приложении окно элемента управления TreeView будет использовано для отображения списка дисков и каталогов.
4. Для элемента управления TreeView создайте разделитель. Для этого перетащите значок элемента управления Splitter из панели элементов вкладки Все формы WindowsForms в окно формы. Разделитель будет автоматически расположен справа от окна элемента управления TreeView.
5. Элемент управления Splitter имеет свойства, которые можно менять в процессе разработки приложения, а также динамически во время его работы. Самое важное свойство разделителя Splitter — это свойство Dock, задающее его расположение. По умолчанию значение этого свойства равно Left, благодаря чему разделитель разместился слева от окна дерева TreeView. При необходимости можно изменять расположение разделителя с помощью окна редактирования, аналогичного окну.
6. Перетащите значок элемента управления Panel из панели элементов вкладки Все формы WindowsForms в окно формы, а затем установите значение свойства Dock этой панели равным Fill. В результате наша панель будет расположена справа от разделителя и займет всю правую часть окна приложения.
7. В верхнем окне будет находиться элемент управления ListView, показывающий список файлов и каталогов. Добавьте в окно приложения элемент управления ListView, перетащив его значок из панели элементов

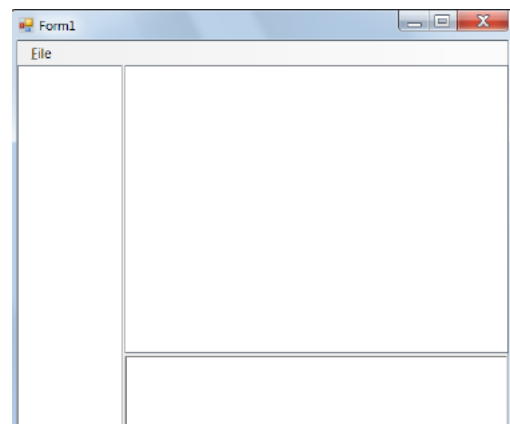
вкладки Все формы WindowsForms в окно формы. После установки значения свойства Dock, равным Top, и увеличения высоты окна этого элемента управления, главное окно нашего приложения примет следующий вид.



8. Добавьте в правую часть главного окна горизонтальный разделитель Splitter. Перетащите его пиктограмму из панели элементов вкладки Все формы WindowsForms в окно формы, а затем установите значение свойства Dock добавленного разделителя, равным Top. В результате разделитель будет расположен непосредственно под окном элемента управления ListView.

9. В нижней части окна приложения будет находиться элемент управления RichTextBox, предназначенный для отображения дополнительной информации. Перетащите значок элемента управления RichTextBox из панели элементов вкладки Все формы WindowsForms в окно формы и установите значение свойства Dock равным Fill.

10. Протестируйте работу приложения. Сохраните полученный проект. Не написав ни единой строчки кода, вы создали простейшее приложение с действующим многооконным интерфейсом пользователя.



Задание 2. Создание методов для элемента управления TreeView

1. Инициализация дерева TreeView осуществляется в конструкторе класса Form1. Для этого необходимо подготовить метод DriveTreeInit.

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();

        DriveTreeInit();
    }
}
```

2. Для обращения к этому методу, а также к другим методам, работающим с дисками, каталогами и файлами, подключите пространство имен System.IO.

```
using System.IO;
```

3. Перейдите в окно редактирования кода и создайте метод DriveTreeInit:

```

public void DriveTreeInit()
{
    string[] drivesArray = Directory.GetLogicalDrives();

    treeView1.BeginUpdate();
    treeView1.Nodes.Clear();

    foreach(string s in drivesArray)
    {
        TreeNode drive = new TreeNode(s, 0, 0);
        treeView1.Nodes.Add(drive);

        GetDirs(drive);
    }
    treeView1.EndUpdate();
}

```

В самом начале своей работы этот метод получает список логических дисковых устройств, установленных в системе, и сохраняет его в массиве drivesArray. Далее метод DriveTreeInit вызывает метод treeView1.BeginUpdate. Этот метод временно блокирует перерисовку окна дерева до тех пор, пока не будет вызван метод treeView1.EndUpdate. Пара этих методов используется в том случае, когда нужно добавить, удалить или изменить большое количество элементов дерева. Если не заблокировать перерисовку окна, на обновление дерева уйдет слишком много времени.

В классе TreeView определено свойство Nodes, хранящее все узлы дерева. Перед тем как приступить к заполнению дерева, метод DriveTreeInit удаляет все узлы, вызывая для этого метод treeView1.Nodes.Clear. Заполнение дерева происходит в цикле. Массив drivesArray содержит массив текстовых строк с обозначениями логических устройств, установленных в системе. Для каждого такого устройства в цикле создается объект класса TreeNode — узел дерева. В качестве первого параметра конструктору передается текстовая строка названия устройства. Созданный узел добавляется в набор узлов дерева с помощью метода treeView1.Nodes.Add. В качестве параметра этому методу передается ссылка на объект TreeNode, т.е. на добавляемый узел. Далее вызывается метод GetDirs, добавляющий в дерево список содержимого корневого каталога текущего дискового устройства.

4. Метод GetDirs получает в качестве параметра ссылку на узел дерева, который требуется наполнить списком имен каталогов и файлов. Создайте метод GetDirs.

```

public void GetDirs(TreeNode node)
{
    DirectoryInfo[] diArray;
    node.Nodes.Clear();

    string fullPath = node.FullPath;
    DirectoryInfo di = new DirectoryInfo(fullPath);
}

```

```

try
{
    diArray = di.GetDirectories();
}
catch
{
    return;
}

foreach(DirectoryInfo dirinfo in diArray)
{
    TreeNode dir = new TreeNode(dirinfo.Name, 0, 0);
    node.Nodes.Add(dir);
}
}

```

Первым делом метод GetDirs удаляет все элементы из текущего узла, вызывая для этого метод node.Nodes.Clear.

Далее метод переписывает в переменную fullPath типа string полный путь node.FullPath к узлу дерева. Эта строка получается объединением текстовых строк всех родительских узлов. Если корневой узел хранит текстовую строку обозначения логического диска, то после выполнения этой процедуры в переменной fullPath будет храниться полный путь к файлу или каталогу.

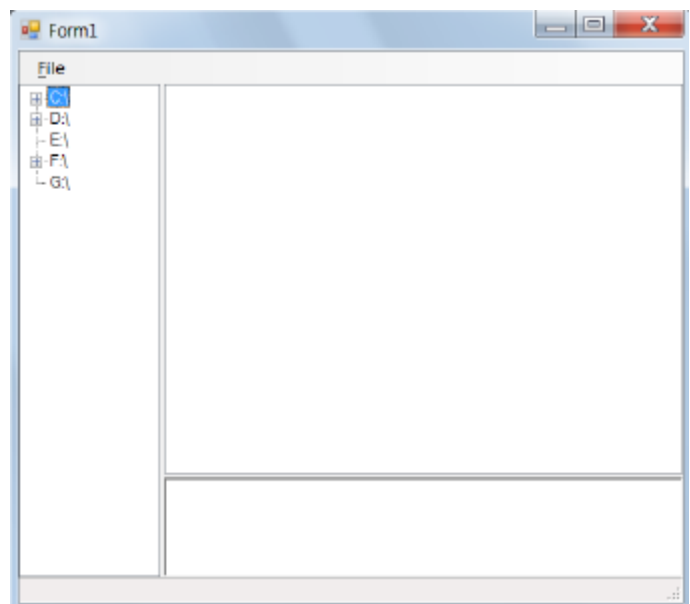
Далее для получения содержимого каталога, на который ссылается переменная fullPath, используется метод GetDirectories. При возникновении исключения программа просто возвращает управление, не выполняя над узлом дерева никаких функций.

В том случае если содержимое каталога было успешно получено, оно сохраняется в массиве diArray. Далее содержимое массива diArray используется для заполнения узла дерева содержимым каталога.

5. Запустите программу на исполнение. В окне дерева появится список дисковых устройств.

6. Чтобы узлы дерева раскрывались, когда пользователь щелкает их мышью или пытается раскрыть при помощи клавиатуры, нужно обрабатывать событие BeforeExpand. Для создания обработчика этого события выделите дерево в окне дизайнера формы, а затем откройте вкладку событий. Далее в поле BeforeExpand ведите имя обработчика событий treeView1_OnBeforeExpand.

После этого добавьте в этот обработчик событий следующий код:



```
private void treeView1_OnBeforeExpand(object sender, TreeViewCancelEventArgs e)
{
    treeView1.BeginUpdate();

    foreach (TreeNode node in e.Node.Nodes)
    {
        GetDirs(node);
    }

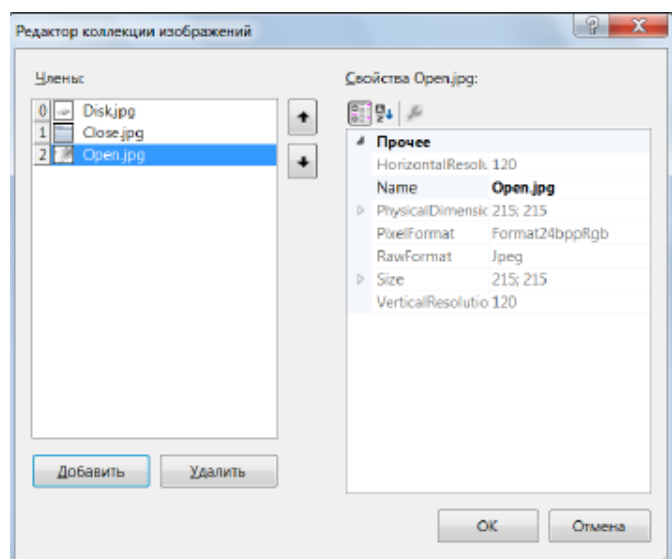
    treeView1.EndUpdate();
}
```

Событие BeforeExpand возникает при попытке пользователя раскрыть узел дерева. В этом случае наш обработчик заполняет открываемый узел при помощи рассмотренного ранее метода GetDirs. Ссылка на узел извлекается из поля e.Node.Nodes, передаваемого обработчику событий в качестве параметра.

7. Запустите приложение.

8. Вы можете улучшить внешний вид дерева, если добавите к его узлам значки дисковых устройств и каталогов (папок). В папке проекта создайте папку ImageList. Подготовьте эти значки, используя любой графический редактор. Скопируйте эти значки в папку ImageList. Чтобы подключить флажки к проекту, выберите из меню проекта системы Microsoft Visual Studio строку Добавить существующий элемент, а затем добавьте файлы значков при помощи появившегося на экране диалогового окна.

9. Чтобы использовать изображения в дереве, их необходимо объединить в список класса ImageList. Добавьте этот список в приложение, перетащив значок компонента ImageList в окно проектирования формы из панели элементов вкладки Все формы Windows Forms. Выделив элемент imageList1 левой клавишей мыши, отредактируйте его свойство Images. Для редактирования будет открыто окно Редактор коллекций изображений. Воспользуйтесь кнопкой Добавить для добавления в список файлов изображений, скопированных ранее в каталог нашего приложения. Изображения следует разместить в следующем порядке: первым (с индексом 0) должно идти изображение для диска, вторым (с индексом 1) — изображение закрытой папки, и третьим (с индексом 2) — изображение открытой папки.



10. Создав и заполнив список изображений, подключите его к дереву просмотра дисков и каталогов. Для этого отредактируйте свойство ImageList элемента управления treeView1, присвоив ему ссылку на список изображений imageList1.

11. Для отображения значков в узлах дерева необходимо изменить исходный текст методов `DriveTreeInit` и `GetDirs`. Первый из этих методов инициализирует дерево, а второй— добавляет к узлу дерева список каталогов. Обратите внимание на конструктор класса `TreeNode`, создающий узлы дерева внутри тела цикла `foreach`. Первый параметр задает текст надписи для узла дерева. Если к элементу управления `TreeView` подключен список изображений, то второй и третий параметры конструктора класса `TreeNode` задают индексы изображений для узла дерева. При этом второй параметр определяет изображение невыделенного узла дерева, а третий — выделенного. Что касается метода `DriveTreeInit`, то расположенный в нем конструктор создает узлы, отображающий только дисковые устройства. В любом состоянии (как выделенном, так и невыделенном) нам необходимо отображать один и тот же значок дискового устройства, имеющий в нашем случае индекс 0. Поэтому второй и третий параметры конструктора передают нулевые значения.

```
public void DriveTreeInit()
{
    string[] drivesArray = Directory.GetLogicalDrives();

    treeView1.BeginUpdate();
    treeView1.Nodes.Clear();

    foreach(string s in drivesArray)
    {
        TreeNode drive = new TreeNode(s, 0, 0);
        treeView1.Nodes.Add(drive);

        GetDirs(drive);
    }
    treeView1.EndUpdate();
}
```

12. В методе `GetDirs` конструктору класса `TreeNode`, размещенному внутри оператора цикла `foreach`, через второй и третий параметры передаются индексы значков закрытой и открытой папки, соответственно.

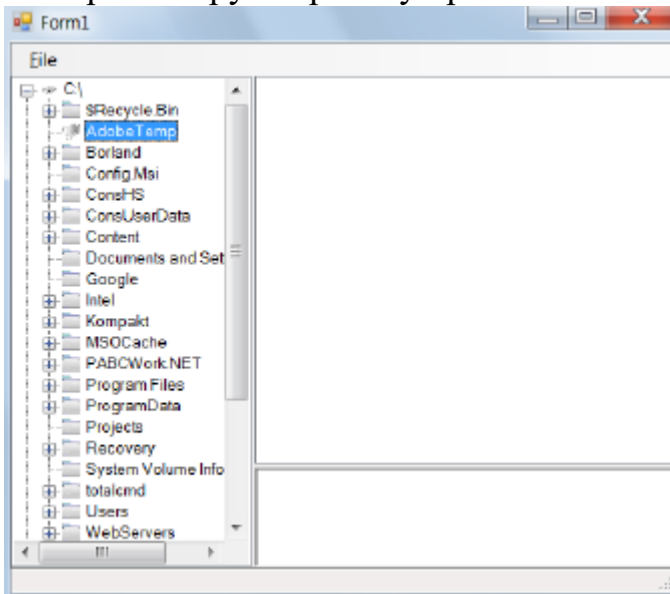
```
public void GetDirs(TreeNode node)
{
    DirectoryInfo[] diArray;
    node.Nodes.Clear();

    string fullPath = node.FullPath;
    DirectoryInfo di = new DirectoryInfo(fullPath);

    try
    {
        diArray = di.GetDirectories();
    }
    catch
    {
        return;
    }

    foreach(DirectoryInfo dirinfo in diArray)
    {
        TreeNode dir = new TreeNode(dirinfo.Name, 1, 2);
        node.Nodes.Add(dir);
    }
}
```

13. Протестируйте работу приложения.



14. Выделите элемент управления TreeView на панели Свойства найдите свойство CheckBoxes и установите его значение True. Теперь узлы дерева будут иметь индивидуальные флажки. Отметив все или некоторые узлы дерева флажками, можно выполнять над соответствующими объектами групповые операции.

15. Выделите элемент управления TreeView на панели Свойства найдите свойство LabelEdit и установите его значение True. Теперь пользователь может редактировать текстовые надписи, расположенные около узлов дерева.

ПРАКТИЧЕСКАЯ РАБОТА № 42

Тема: Разработка интерфейса пользователя

Цель работы: сформировать практические умения создания многооконного приложения MDI средствами среды программирования VisualStudio.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Справочный материал:

Многооконный интерфейс MDI (Multiple Document Interface) позволяет в одном приложении работать одновременно с несколькими документами или с разными представлениями одного и того же документа. Окна, отображающие содержимое документов внутри главного окна MDI-приложения, называются MDI-окнами. Любое MDI-приложение содержит меню Windows, предназначенное для управления окнами, отображающими различные документы или различные представления одного и того же документа. Как правило, в меню Windows есть строки Cascade и Tile, с помощью которых пользователь может упорядочить MDI-окна, расположив их с перекрытием (друг за другом) или рядом друг с другом. В этом меню могут быть и другие строки, управляющие расположением MDI-окон.

Ниже разделительной черты в меню Windows обычно находятся строки, обозначающие отдельные MDI-окна. Если выбрать одну из этих строк, указанное окно будет активизировано и показано на первом плане. Меню Windows может быть разным в различных MDI-приложениях. Однако в любом случае это меню позволяет пользователю автоматически упорядочить расположение MDI-окон и выбрать нужное окно из списка для активизации.

Двигая MDI-окна при помощи заголовка, Вы не сможете переместить их за пределы главного окна приложения. В то же время MDI-окна можно делать активными, при этом заголовок активного MDI-окна выделяется цветом. Каждое MDI-окно имеет, как правило, системное меню и кнопки изменения размера. С помощью системного меню пользователь может изменять размеры или перемещать MDI-окно (строки Restore, Move, Size, Maximize, Minimize), закрывать окно (строка Close), передавать фокус ввода от одного MDI-окна другому (строка Next Window).

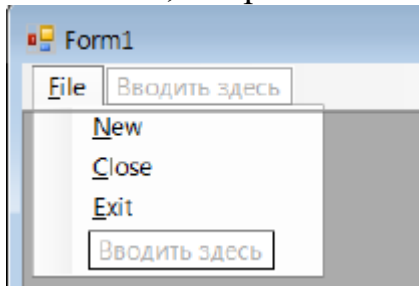
Если MDI-окно сворачивается в значок (минимизируется), то этот значок располагается в нижней части главного окна приложения.

Содержание работы:

Задание 1. Создайте приложение, демонстрирующее особенности работы с несколькими окнами.

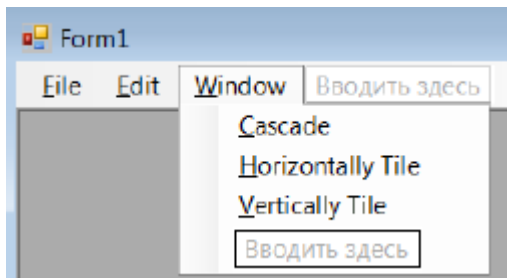
1. Запустите среду программирования VisualStudio. Создайте новое Приложение WindowsForms. Имя проекта и приложения MDIApp.
2. Выделите форму, на панели Свойства найдите свойство IsMDIContainer и установите значение этого свойства равным True. В результате главное окно приложения превратится в контейнер для дочерних MDI-окон.
3. Добавьте в приложение главное меню. Для этого перетащите из панели элементов системы Visual Studio значок главного меню MenuStrip.

4. Создайте меню File со строками New, Close и Exit. С помощью строки New этого меню мы будем создавать новые MDI-окна с редактором текста на базе элемента управления RichTextBox. Строка Close предназначена для закрытия MDI-окон, а строка Exit — для завершения работы приложения



5. Создайте меню Edit со строками Copy и Paste.

6. Создайте меню Window, предназначенное для управления MDI-окнами. Строка Cascade предназначена для каскадного расположения открытых MDI-окон, а строки HorizontallyTile и VerticallyTile — для размещения окон рядом друг с другом в горизонтальном и вертикальном направлении, соответственно.



7. После создания меню Window выделите его, на панели Свойства установите значение свойства MdiWindowListItem равным windowToolStripMenuItem. При этом во время работы приложения в меню Window будут автоматически добавлены строки списка открытых MDI-окон. В этом меню строки активных окон будут отмечены.

8. MDI-окна обычно используются для отображения различных документов или различных представлений одного и того же документа. В приложении MDIAppMDI-окна будут содержать внутри себя элемент управления RichTextBox, т.е. редактор текста.

9. Добавьте в проект приложения новую форму, которая будет играть роль шаблона для создания дочерних MDI-окон. Для добавления формы щелкните правой клавишей мыши в окне Обозреватель решений по имени проекта MDIApp и выберите в контекстном меню Добавить строку Форму Windows. После этого на экране появится диалоговое окно Добавление нового элемента, в котором нужно выбрать значок ФормаWindows Form и затем нажать кнопку Открыть. В результате в окне дизайнера форм появится новая форма Form2.

10. Так как это окно будет предназначено для редактирования текста, перетащите в него из панели элементов значок элемента управления RichTextBox. Чтобы окно редактора текста RichTextBox заполнило собой клиентскую часть MDI-окна, установите значение свойства Dock равным Fill. Кроме того, проследите, чтобы свойство Anchor было равно Top, Left. Окно

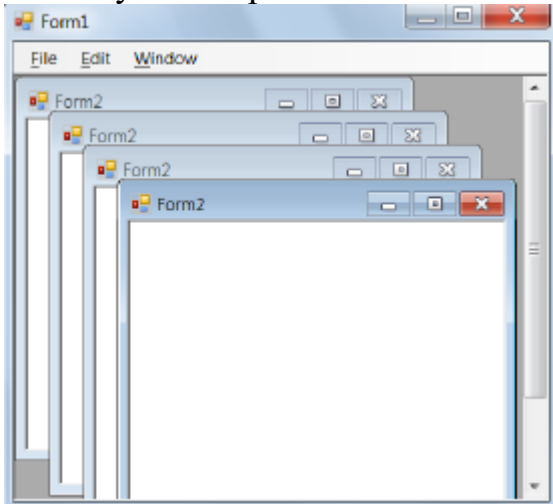
элемента управления RichTextBox будет заполнять клиентскую область MDI-окна при любом изменении размеров последнего.

11. Напишите программный код, создающий MDI-окна. Этот код должен получать управление, когда пользователь выбирает строку New из меню File. Добавьте следующий обработчик события Click для строки New меню File:

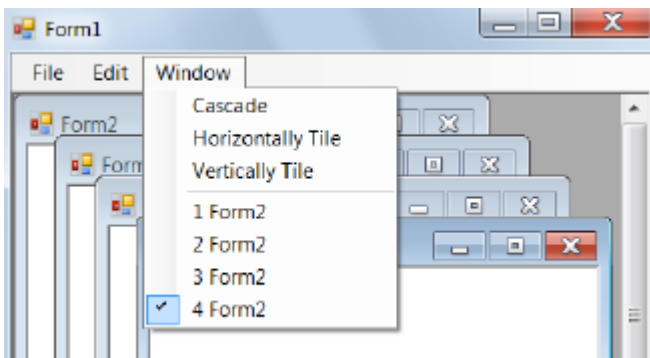
```
private void newToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form2 mdiChild = new Form2();
    mdiChild.MdiParent = this;
    mdiChild.Show();
}
```

Сначала создается новая форма как объект класса Form2, а затем выполняется сохранение ссылки на эту форму в переменной mdiChild. Свойство MdiParent этого окна должно содержать ссылку на родительское окно приложения MDI, поэтому в него записывается ссылка на объект класса Form1, используя ключевое свойство this. Для того чтобы MDI-окно появилось на экране, его необходимо отобразить явным образом при помощи метода Show.

12. Запустите приложение на исполнение. Протестируйте его работу.



13. Создав с помощью меню File несколько дочерних MDI-окон, раскройте меню Window и убедитесь, что в нем появилось несколько новых строк. С помощью этих строк можно выдвинуть на передний план любое нужное MDI-окно.



14. Чтобы пользователь мог упорядочивать дочерние MDI-окна, в приложении предусмотрены строки Cascade, Horizontally Tile и Vertically Tile в меню Window. Добавить в исходный текст приложения обработчики событий

данных пунктов меню:

```
private void cascadeToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.LayoutMdi(MdiLayout.Cascade);
}

private void horizontallyTileToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.LayoutMdi(MdiLayout.TileHorizontal);
}

private void verticallyTileToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.LayoutMdi(MdiLayout.TileVertical);
}
```

Все они вызывают метод `LayoutMdi` родительской формы, передавая ему в качестве параметра одну из констант, определяющих нужный тип упорядочивания. Это константы `MdiLayout.Cascade`, `MdiLayout.TileHorizontal` и `MdiLayout.TileVertical`.

15. Реализуйте операцию копирования выделенного фрагмента текста из активного MDI-окна в Clipboard (буфер обмена). Для этого добавьте следующий обработчик событий к строке `Copy` меню `Edit`:

```
private void copyToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form activeChild = this.ActiveMdiChild;

    if (activeChild != null)
    {
        RichTextBox editBox = (RichTextBox)activeChild.ActiveControl;

        if (editBox != null)
        {
            Clipboard.SetDataObject(editBox.SelectedText);
        }
    }
}
```

Здесь вначале определяется идентификатор активного MDI-окна, извлекая его из свойства `ActiveMdiChild` родительского окна приложения MDI. Этот идентификатор сохраняется в переменной `activeChild`. Если в приложении нет активного окна, то приведенный выше обработчик событий завершает свою работу, не выполняя никаких других действий.

Определив идентификатор активного MDI-окна, обработчик события получает идентификатор активного элемента управления, расположенного внутри этого окна. Для этого используется свойство `activeChild.ActiveControl`, хранящее ссылку на активный элемент управления.

MDI-окно содержит только один элемент управления — редактор `RichTextBox`. Поэтому полученная ссылка преобразуется явным образом к типу `RichTextBox` и сохраняется в переменной `editBox` для дальнейшего использования. Теперь идентификатор редактора текста получен и можно переписать выделенный в его окне фрагмент текста в универсальный буфер обмена `Clipboard`. Запись в `Clipboard` осуществляется методом

Clipboard.SetDataObject. В качестве параметра методу Clipboard.SetDataObject передается выделенный текст, извлеченный из редактора текста с помощью свойства editBox.SelectedText.

16. Дополните приложение кодом, необходимым для вставки данных из универсального буфера обмена Clipboard в активное MDI-окно. Добавьте следующий обработчик события для строки Paste меню Edit:

```
private void pasteToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form activeChild = this.ActiveMdiChild;

    if (activeChild != null)
    {
        RichTextBox editBox = (RichTextBox)activeChild.ActiveControl;

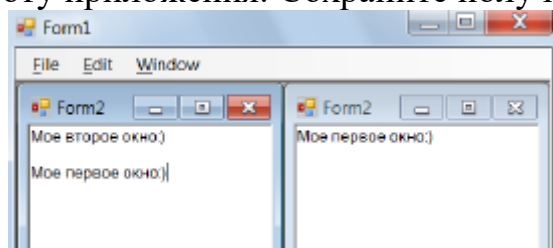
        if (editBox != null)
        {
            IDataObject data = Clipboard.GetDataObject();

            if (data.GetDataPresent(DataFormats.Text))
            {
                editBox.SelectedText =
                    data.GetData(DataFormats.Text).ToString();
            }
        }
    }
}
```

Этот обработчик вначале определяет идентификатор активного MDI-окна, а затем, пользуясь этим идентификатором и свойством ActiveControl, получает идентификатор редактора текста RichTextBox. Далее для вставки данных из буфера Clipboard метод получает ссылку на интерфейс IDataObject, вызывая для этого метод Clipboard.GetDataObject. Эта ссылка сохраняется в переменной data.

Пользуясь полученной ссылкой на интерфейс IDataObject, обработчик события определяет, имеется ли в буфере Clipboard текст, который можно было бы вставить в редактор текста. Для этого используется метод GetDataPresent. В качестве параметра этому методу передается идентификатор формата текстовых данных DataFormats.Text. Если в буфере Clipboard имеются текстовые данные, программа извлекает их при помощи метода GetData, а затем преобразует в текстовую строку при помощи метода ToString. Далее эта текстовая строка записывается в свойство SelectedText нашего редактора текста, благодаря чему и происходит вставка данных из буфера Clipboard.

18. Протестируйте работу приложения. Сохраните полученное приложение.



ПРАКТИЧЕСКАЯ РАБОТА № 43

Тема: Создание приложения с БД

Цель работы: научиться создавать клиентское приложение для работы с базой данных с применением встроенных инструментов в Visual Studio.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

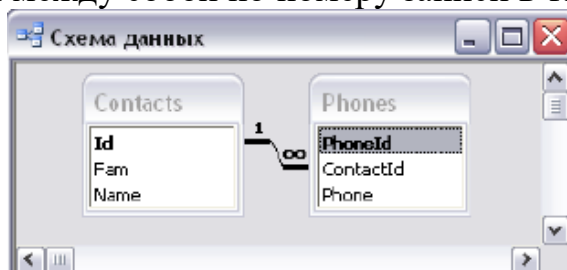
Содержание работы:

Задание 1. База данных из двух связанных таблиц «Телефонная книжка» создана в Microsoft Access. Реализовать с помощью управляемого провайдера OLE DB доступ к этой базе данных.

1. Создание базы данных в Microsoft Access. База данных Organizer состоит из двух связанных таблиц: Contacts и Phones. Структура таблиц приведена ниже:

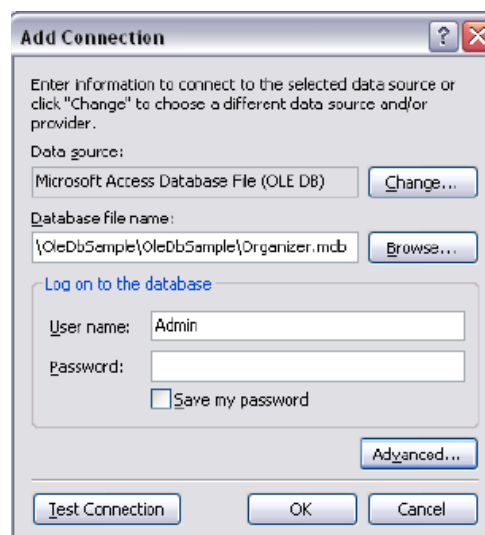
Название поля	Тип поля	Размер поля	Примечание	Назначение поля
<i>Таблица Contacts</i>				
Id	Счетчик	Длинное целое	Ключевое поле	Номер записи
Fam	Текстовый	20		Фамилия
Name	Текстовый	20		Имя
<i>Таблица Phones</i>				
PhoneId	Счетчик	Длинное целое	Ключевое поле	Номер записи
ContactId	Числовой	Длинное целое		Идентификатор контакта
Phone	Текстовый	20		Телефон

Таблицы связываются между собой по номеру записи в таблице контактов.

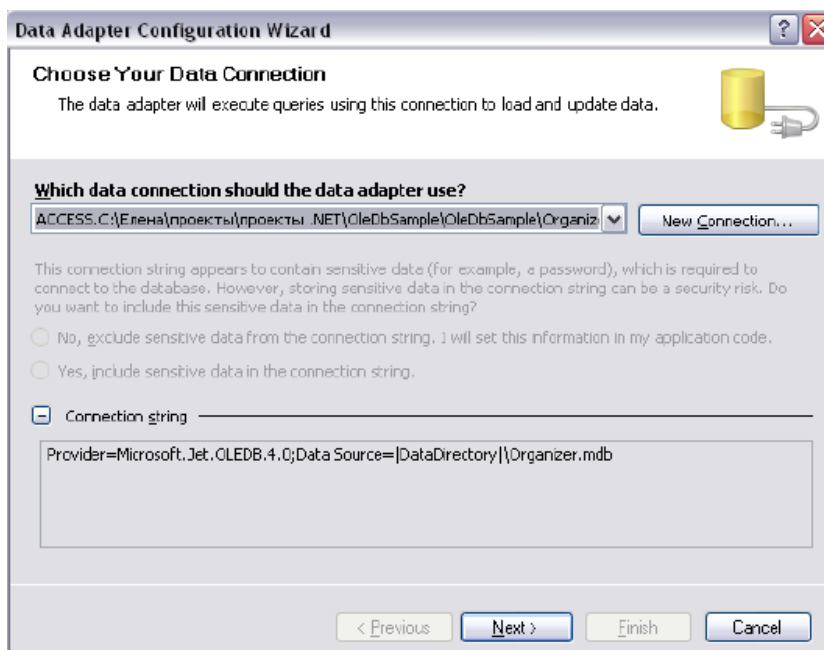


2. Доступ к данным с помощью управляемого провайдера OLE DB. Прежде всего, следует поместить на форму компонент `oleDbConnection` и настроить его свойство `ConnectionString`, выбрав пункт `New connection....` Открывается диалоговое окно.

С помощью кнопки `Change...` в правом верхнем углу окна надо установить вид источника данных (Data source) как `Microsoft Access Database file`. Далее с помощью кнопки `Browse...` в строке `Database file name` нужно указать путь и имя файла с базой данных. Кнопка `Test Connection` позволяет проверить правильность установления соединения.



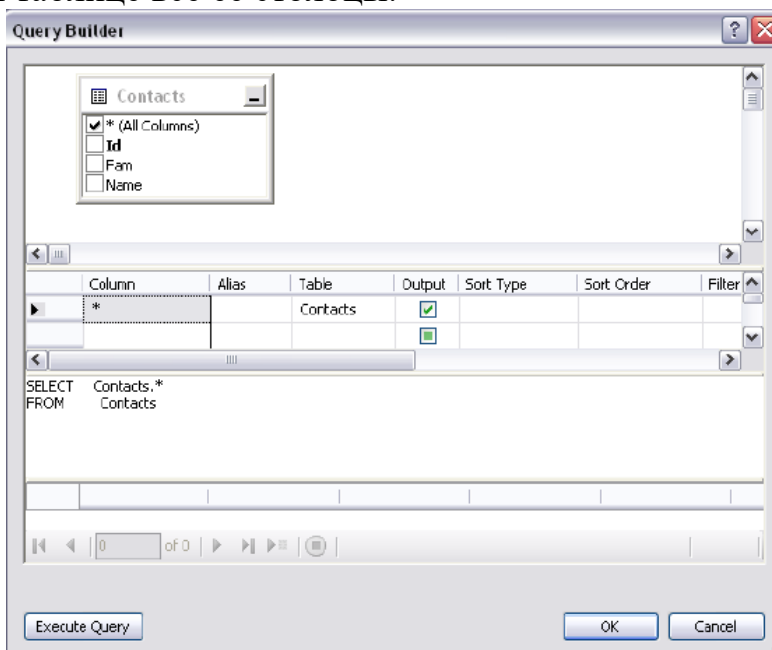
Следующим шагом является создание адаптера – компонента, выполняющего непосредственную передачу данным между приложением и базой данных. Для начала необходимо поместить на форму компонент OleDbDataAdapter. При этом на экране появляется окно мастера настройки адаптера.



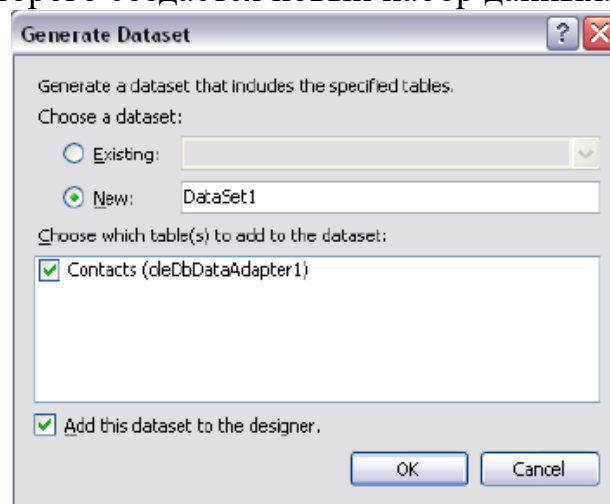
Здесь следует проверить правильность указания пути к файлу БД.

Примечание. После нажатия кнопки Next мастером настройки будет предложен вариант работы с базой данных, при котором при каждом запуске программы будет создаваться копия базы данных в папке приложения. На данном шаге следует нажать кнопку «NO»!

Далее мастер предлагает задать команды SQL для загрузки данных в приложение. Кнопка Query Builder... данного окна предлагает визуальный метод построения данной команды. В открывшемся окне Add Table следует добавить в Query Builder главную таблицу БД – таблицу Contacts, а затем отметить в этой таблице все ее столбцы.

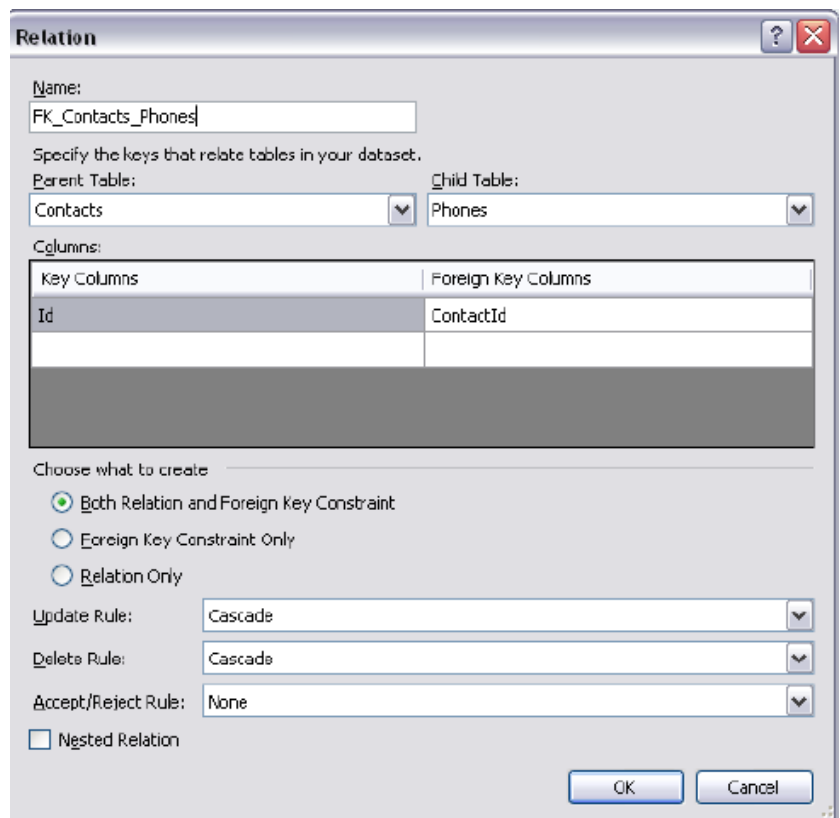


На этом создание адаптера таблицы Contacts завершается. Следующим шагом является формирование набора данных – объекта DataSet. Для этого нужно в режиме дизайнера формы вызвать контекстное меню и выбрать в нем пункт Generate DataSet... Открывается соответствующее диалоговое окно, с помощью которого создается новый набор данных.



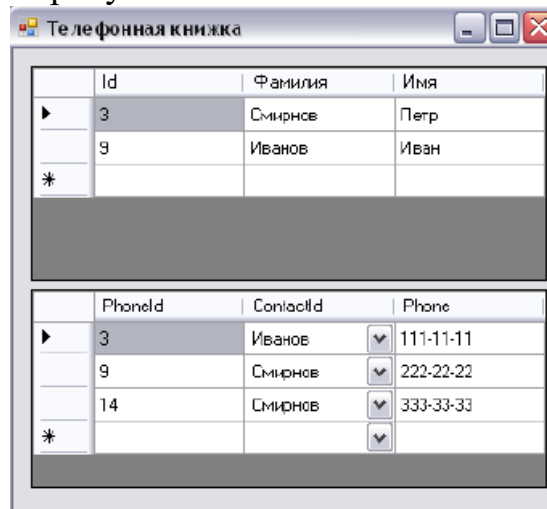
Аналогичным образом добавляется адаптер и для второй таблицы БД – таблицы Phones. Процесс создания данного адаптера аналогичен адаптеру таблицы Contacts (в окне Query Builder, естественно, нужно добавлять таблицу Phones). Чтобы добавить таблицу Phones в DataSet, следует щелкнуть правой кнопкой мыши на адаптере данной таблицы, в контекстном меню выбрать Generate DataSet... и в открывшемся диалоговом окне выбрать существующий объект DataSet (т.е. оставить настройки по умолчанию).

После этого в сгенерированный набор данных можно добавлять другие необходимые элементы: остальные таблицы БД, связи между ними, запросы и т.п. Для этого нужно в окне Solution Explorer выбрать (дважды щелкнуть мышью) элемент DataSet1.xsd. Так, для нашего приложения в набор данных следует добавить отношение – связь между таблицами БД: в окне DataSet1.xsd в контекстном меню выбирается пункт Add-Relation... Открывается диалоговое окно редактирования отношения, где можно задать имя создаваемого

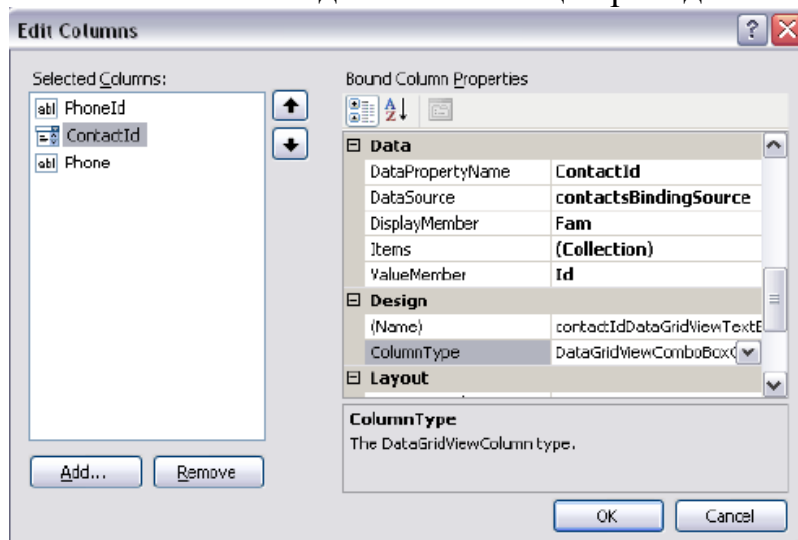


отношения, родительскую и подчиненную таблицы, ключевые поля, а также режимы автоматических изменений дочерней таблицы при изменении родительской таблицы. Для разрабатываемого приложения создается отношение с именем FK_Contacts_Phones. Его настройки приведены на рисунке.

3. Визуальное проектирование диалогового окна. Внешний вид работающего приложения приведен на рисунке



Для отображения данных, полученных из базы, в программе будут использоваться компоненты DataGridView, по одному для каждой таблицы базы данных. Имена их по умолчанию устанавливаются как dataGridView1 и dataGridView2. Компонентам-таблицам dataGridView1 и dataGridView2 нужно установить свойство DataSource, задающее источник данных для отображения в таблице, в «contactsBindingSource» и «phonesBindingSource» соответственно. Кроме того, при необходимости можно отредактировать заголовки и другие настройки столбцов таблиц (свойство Columns). В данном приложении в столбце ContactId компонента dataGridView2 для наглядности отображается не числовой идентификатор человека, которому принадлежит телефон, а его фамилия, причем для отображения используется тип столбца «DataGridViewComboBoxColumn» (свойство ColumnType). Настройка остальных свойств данного столбца приведена на рисунке



4. Проектирование программного кода. Для отображения в компонентах DataGridView данных из таблиц необходимо прежде всего открыть настроенное соединение с базой данных, а затем заполнить таблицы, созданные в объекте DataSet данными из таблиц в базе. В нашем случае делается это при загрузке главного окна программы, в обработчике события Load:

```
private void Form1_Load(object sender, EventArgs e) {  
    OleDbConnection1.Open(); //открыть соединение  
    //заполнить таблицы в объекте DataSet  
    OleDbDataAdapter1.Fill(dataSet11.Contacts);  
    phonesTableAdapter.Fill(dataSet11.Phones); }  
    
```

При закрытии формы необходимо отключить соединение с базой данных:

```
private void Form1_FormClosing(object sender,  
FormClosingEventArgs e) {  
    OleDbConnection1.Close();} //закрыть соединение  
    
```


ПРАКТИЧЕСКАЯ РАБОТА № 44

Тема: Создание приложения с БД

Цель работы: научиться создавать клиентское приложение для работы с базой данных с применением встроенных инструментов в Visual Studio.

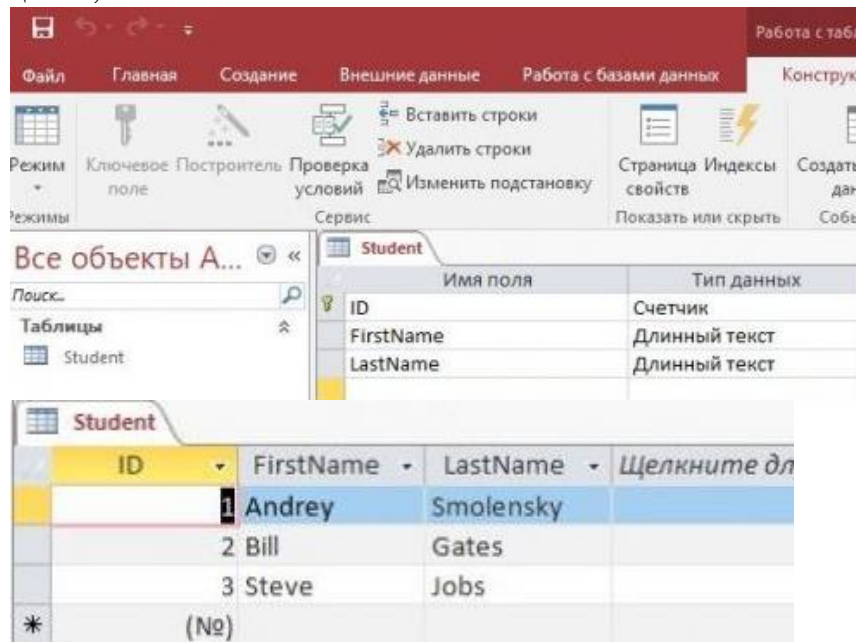
Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Содержание работы:

Задание 1. Написать приложение для работы с БД MS Access, которое сможет обновлять, удалять и вставлять данные.

1. Откройте MS Access, нажмите на пустую базу данных рабочего стола. Дайте базе данных имя «dbSchool.accdb», а затем нажмите кнопку Создать.

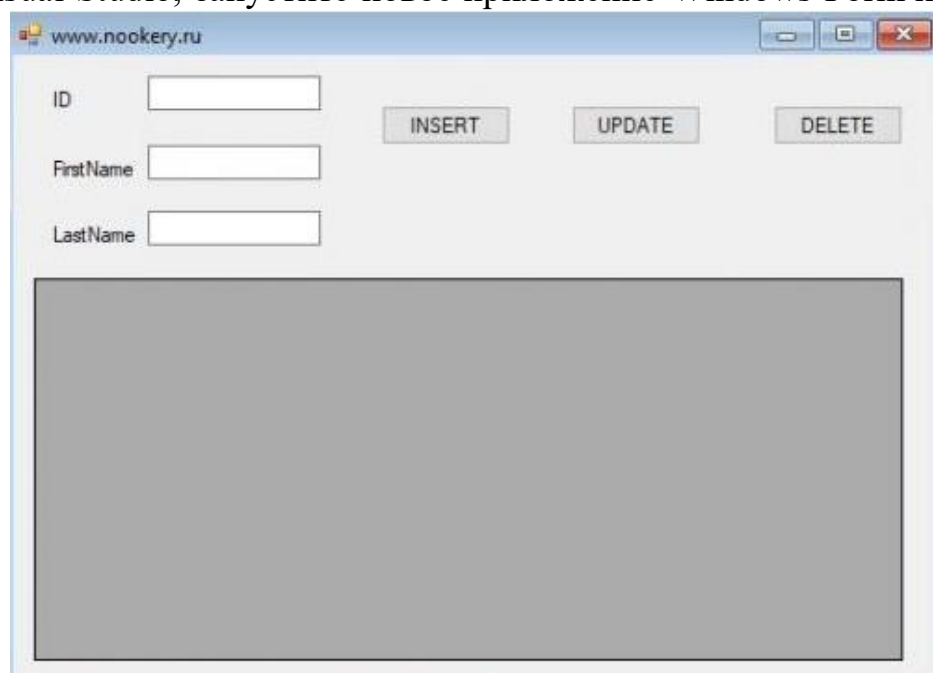
2. Теперь создайте таблицу в базе данных, вы можете назвать таблицу как хотите, здесь я назвал ее “Student”. Существует три столбца в таблице ID, FirstName и LastName



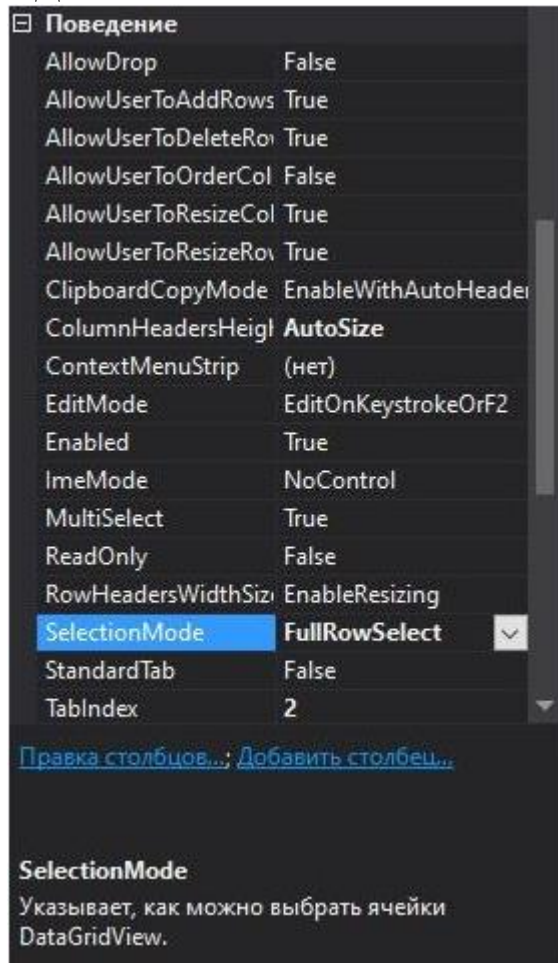
3. Теперь откройте Visual Studio, запустите новое приложение Windows Form и дайте любое имя, которое вы хотите.

4. Теперь перетащите файл базы данных из документов в папку каталога проекта. Так что бы она находилась рядом с нашей будущей программой, для удобства работы.

5. Дизайн Формы:



6. Добавим элемент DataGridView и установим для него свойство



7. Напишите пространство имен для подключения: using System.Data.OleDb;

8. Определите глобальные переменные.

```
OleDbConnection con;  
OleDbDataAdapter da;  
OleDbCommand cmd;  
DataSet ds;
```

9. Создайте метод для получения списка учащихся.

```
void GetStudent() {  
    con = new OleDbConnection("Provider=Microsoft.ACE.Oledb.12.0;Data  
Source=dbSchool.accdb");  
    da = new OleDbDataAdapter("SELECT *FROM Student", con);  
    ds = new DataSet();  
    con.Open();  
    da.Fill(ds, "Student");  
    dataGridView1.DataSource = ds.Tables["Student"];  
    con.Close(); } }
```

10. Создайте исходный код для кнопки вставки данных

```
private void btnInsert_Click(object sender, EventArgs e) { //INSERT BUTTON  
    string query = "Insert into Student (FirstName,LastName) values  
(@fName,@lName)";  
    cmd = new OleDbCommand(query, con);
```

```

cmd.Parameters.AddWithValue("@fName", txtFirstName.Text);
cmd.Parameters.AddWithValue("@lName", txtLastName.Text);
con.Open();
cmd.ExecuteNonQuery();
con.Close();
GetStudent();    }

```

11. Создайте исходный код для кнопки Удалить

```

private void btnDelete_Click(object sender, EventArgs e) { //DELETE BUTTON
    string query = "Delete From Student Where Id=@id";
    cmd = new OleDbCommand(query, con);
    cmd.Parameters.AddWithValue("@id", dataGridView1.CurrentRow.Cells[0].Value);
    con.Open();
    cmd.ExecuteNonQuery();
    con.Close();
    GetStudent();    }

```

12. Создайте исходный код для кнопки Обновить

```

private void btnUpdate_Click(object sender, EventArgs e) { //UPDATE BUTTON
    string query = "Update Student Set
FirstName=@fName,LastName=@lName Where Id=@id";
    cmd = new OleDbCommand(query, con);
    cmd.Parameters.AddWithValue("@ad", txtFirstName.Text);
    cmd.Parameters.AddWithValue("@soyad", txtLastName.Text);
    cmd.Parameters.AddWithValue("@id", Convert.ToInt32(txtId.Text));
    con.Open();
    cmd.ExecuteNonQuery();
    con.Close();
    GetStudent();    }

```

13. Создайте исходный код для события dataGridView1_cellEnter

```

private void dataGridView1_CellEnter(object sender, DataGridViewCellEventArgs
e)    {
    txtId.Text = dataGridView1.CurrentRow.Cells[0].Value.ToString();
    txtFirstName.Text = dataGridView1.CurrentRow.Cells[1].Value.ToString();
    txtLastName.Text = dataGridView1.CurrentRow.Cells[2].Value.ToString();
}

```

14. Создайте исходный код для события Form_Load

```

private void Form1_Load(object sender, EventArgs e)    {
    GetStudent();    }

```

15. Полный текст исходного кода

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;

```

```

using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.OleDb;
namespace Пример_приложения_для_работы_с_Access_базой{
    public partial class Form1 : Form //nookery.ru    {
        OleDbConnection con;
        OleDbDataAdapter da;
        OleDbCommand cmd;
        DataSet ds;
        public Form1()    {
            InitializeComponent();    }
        void GetStudent()    {
            //либо укажите полный путь до БД
            //con = new
OleDbConnection(@"Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=D:\Projects\Пример приложения для работы с Access базой\Пример
приложения для работы с Access базой\bin\Debug\dbSchool.accdb");
            con = new OleDbConnection("Provider=Microsoft.ACE.Oledb.12.0;Data
Source=dbSchool.accdb");
            da = new OleDbDataAdapter("SELECT *FROM Student", con);
            ds = new DataSet();
            con.Open();
            da.Fill(ds, "Student");
            dataGridView1.DataSource = ds.Tables["Student"];
            con.Close();    }
        private void Form1_Load(object sender, EventArgs e)    {
            GetStudent();    }
        private void btnInsert_Click(object sender, EventArgs e)//INSERT BUTTON
        {
            string query = "Insert into Student (FirstName,LastName) values
(@fName,@lName)";
            cmd = new OleDbCommand(query, con);
            cmd.Parameters.AddWithValue("@fName", txtFirstName.Text);
            cmd.Parameters.AddWithValue("@lName", txtLastName.Text);
            con.Open();
            cmd.ExecuteNonQuery();
            con.Close();
            GetStudent();    }
        private void btnDelete_Click(object sender, EventArgs e)//DELETE BUTTON
        {
            string query = "Delete From Student Where Id=@id";
            cmd = new OleDbCommand(query, con);
            cmd.Parameters.AddWithValue("@id",
dataGridView1.CurrentRow.Cells[0].Value);
            con.Open();

```

```

cmd.ExecuteNonQuery();
con.Close();
GetStudent();    }
private void btnUpdate_Click(object sender, EventArgs e)//UPDATE BUTTON
{
    string query = "Update Student Set FirstName=@fName,LastName=@lName
Where Id=@id";
    cmd = new OleDbCommand(query, con);
    cmd.Parameters.AddWithValue("@ad", txtFirstName.Text);
    cmd.Parameters.AddWithValue("@soyad", txtLastName.Text);
    cmd.Parameters.AddWithValue("@id", Convert.ToInt32(txtId.Text));
    con.Open();
    cmd.ExecuteNonQuery();
    con.Close();
    GetStudent();    }
private void dataGridView1_CellEnter(object sender,
DataGridViewCellEventArgs e)    {
    txtId.Text = dataGridView1.CurrentRow.Cells[0].Value.ToString();
    txtFirstName.Text = dataGridView1.CurrentRow.Cells[1].Value.ToString();
    txtLastName.Text = dataGridView1.CurrentRow.Cells[2].Value.ToString();
} } }

```

www.nookery.ru

ID: INSERT UPDATE DELETE

First Name: Last Name:

	ID	First Name	Last Name
▶	1	Andrey	Smolensky
	2	Bill	Gates
	3	Steav	Jobs
•			

Внимание если у вас возникает ошибка: System.InvalidOperationException: «Поставщик «Microsoft.ACE.OLEDB.12.0» не зарегистрирован на локальном компьютере.»

Вам надо изменить целевую сборку проекта на x64

ПРАКТИЧЕСКАЯ РАБОТА № 45

Тема: Создание приложения с БД

Цель работы: научиться создавать клиентское приложение для работы с базой данных с применением встроенных инструментов в Visual Studio.

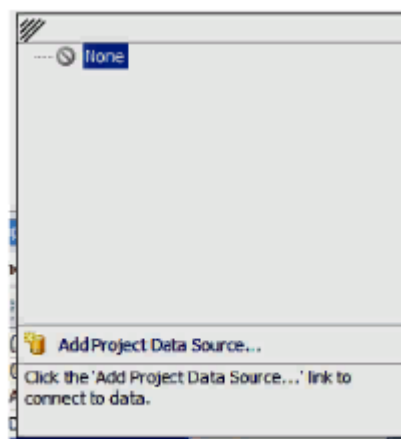
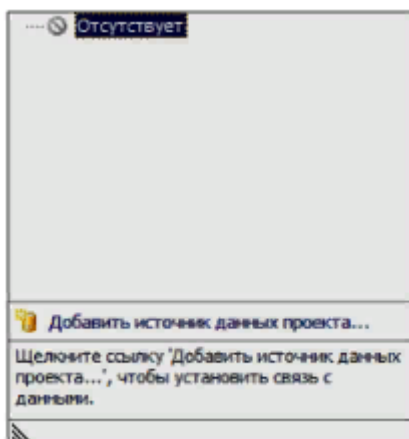
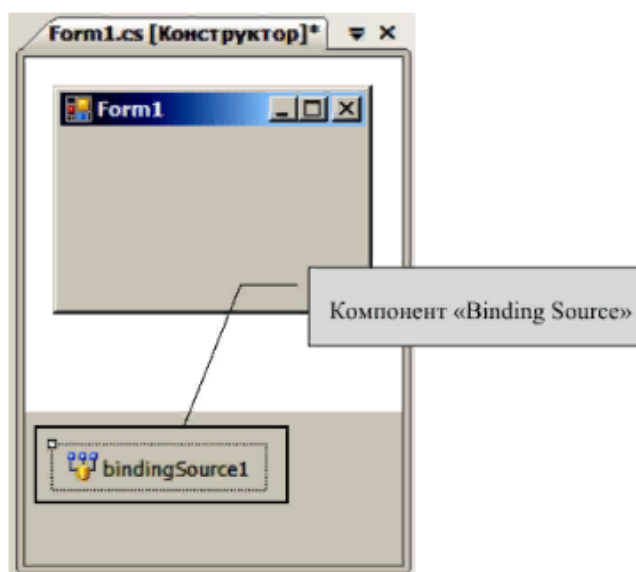
Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Содержание работы:

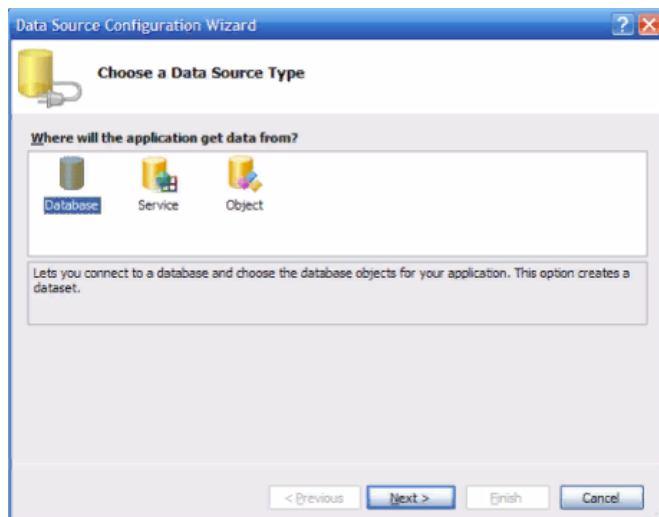
Задание 1. Создадим простое приложение баз данных, которое выводит на экранную форму информацию из таблицы «Туристы» и связанную с текущей записью таблицы «Туристы» запись таблицы «Информация о туристах» из базы данных Microsoft Access.

1.Привязку данных БД к форме осуществляет компонент «Binding Source». Перенесем его на форму. После размещения его на форме среда разработки принимает следующий вид.

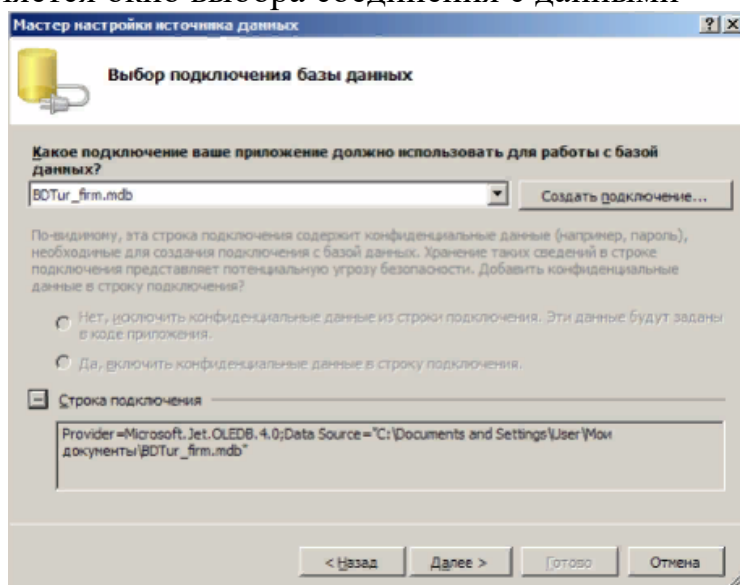
Компонент является не визуальным, поэтому он отображается на дополнительной панели. Основным свойством компонента является свойство DataSource, указывающее на источник данных. По умолчанию свойство является пустым, поэтому необходимо сформировать его значение. При выборе данного свойства в окне свойств появляется следующее окно



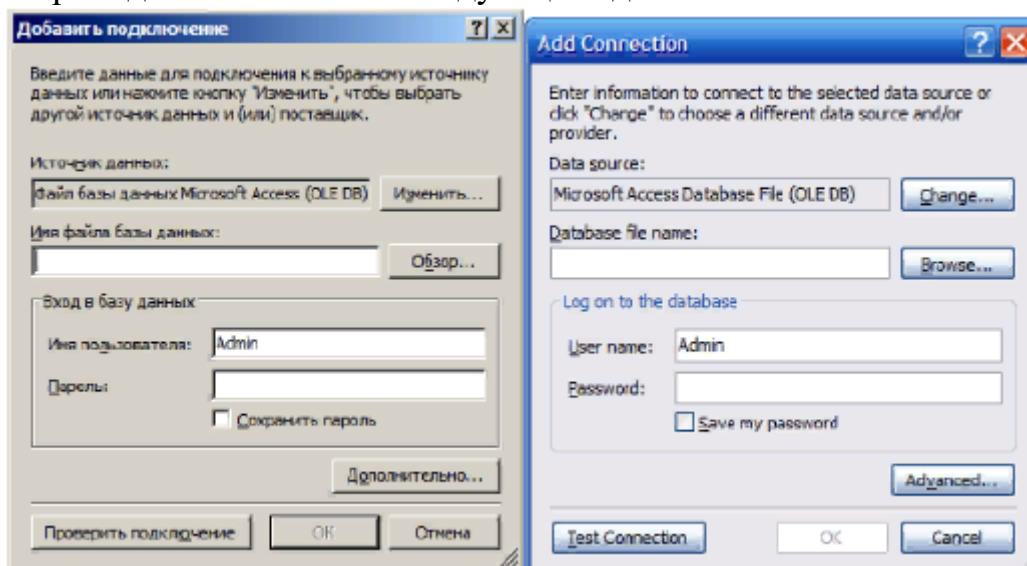
В настоящий момент список пуст, поэтому необходимо создать новый источник данных, выбрав команду «Add Project Data Source» для создания нового источника данных и соединения с ним. Появляется следующее окно диалога



В нашем случае необходимо выбрать пункт «База данных» («Database»). Появляется окно выбора соединения с данными

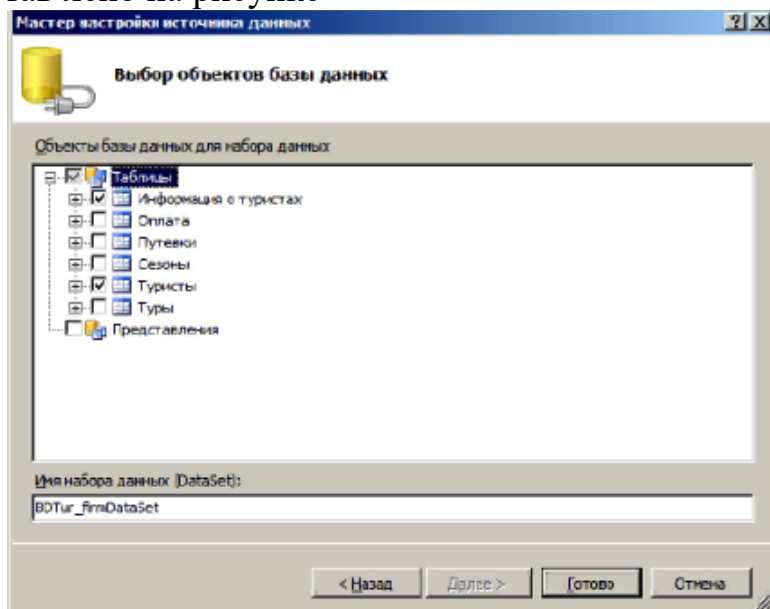


В выпадающем списке диалога находятся все создаваемые ранее соединения. Если необходимого соединения в списке нет, то следует использовать кнопку «Создать подключение» («New connection»). Нажатие кнопки приводит к появлению следующего диалога

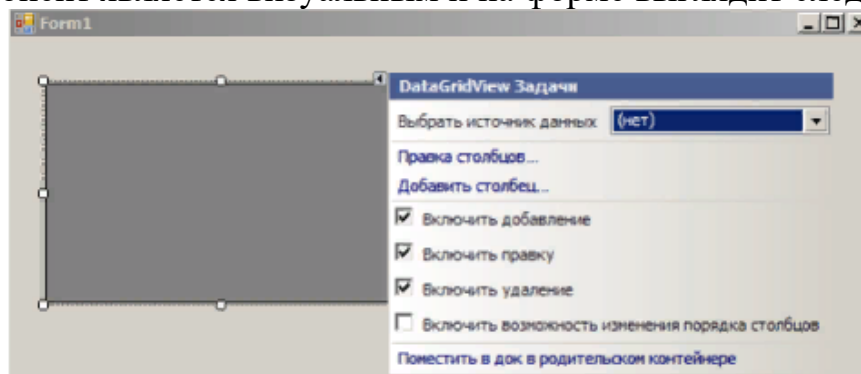


Следующий шаг диалога предлагает сохранить полученную строку соединения в файле настроек приложения. Рекомендуется принять данный выбор для упрощения последующего размещения и поддержки программного продукта.

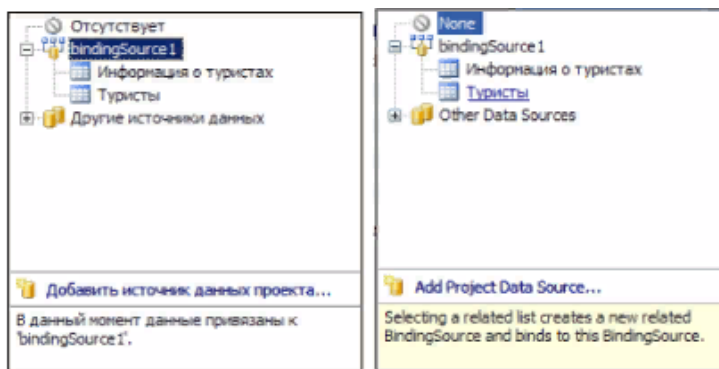
Последний шаг диалога – выбор тех таблиц или иных объектов базы данных, которые необходимы в данном источнике данных. Окно выбора представлено на рисунке



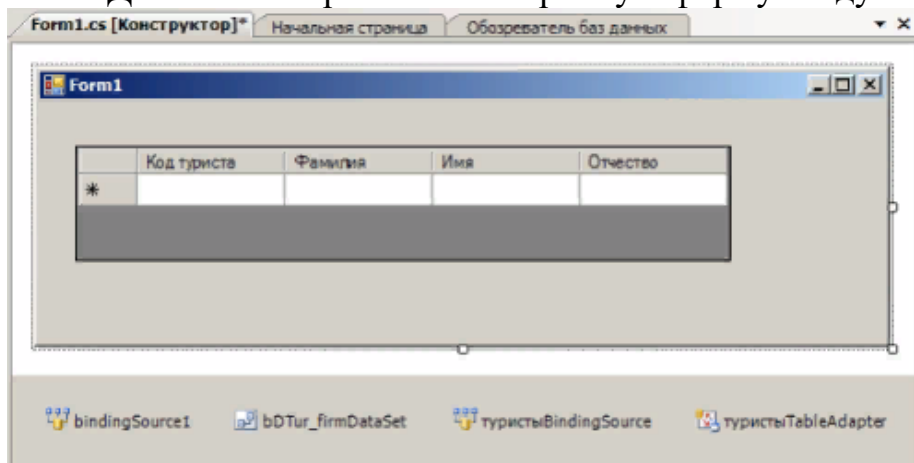
На этом создание источника данных завершено. После нажатия кнопки «Готово» («Finish») рядом с компонентом BindingSource на форме появляется компонент DataSet. Теперь данные, подключенные выше, необходимо отобразить на форме. Простейшим способом отображения данных является использование компонента DataGridView из группы компонентов Data. Компонент является визуальным и на форме выглядит следующим образом.



Для того чтобы компонент мог отображать данные, необходимо выбрать источник данных в выпадающем списке. Выбор выпадающего списка приводит к появлению следующего диалога.

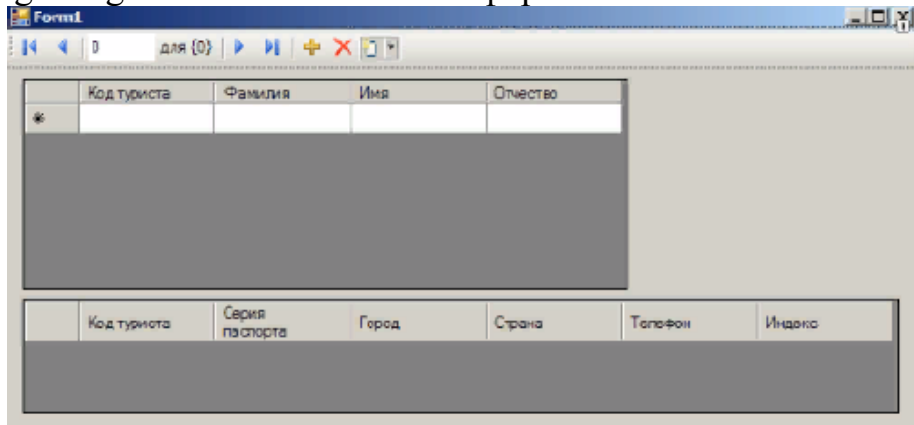


В данном случае мы выбрали в качестве источника данных таблицу «Туристы». Данный выбор изменяет экранную форму следующим образом



Разместить на форме еще один компонент DataGridView для подключения таблицы «Информация о туристах».

Для упрощения навигации по данным существует компонент BindingNavigator. Поместим его на форму

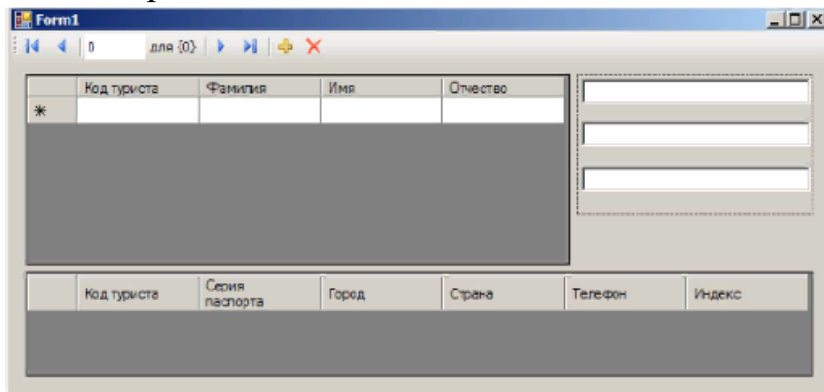


Данный компонент позволяет осуществлять навигацию между записями таблицы, добавлять и удалять строки таблицы. Возможности и внешний вид компонента можно настраивать, так как он представляет собой полосу меню ToolStripContainer.

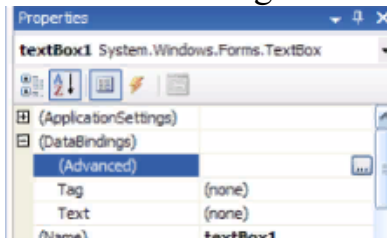
Свойством, определяющим таблицу, по которой производится навигация, является свойство BindingSource. Установим значение этого свойства равным «туристыBindingSource». В работе компонент выглядит следующим образом



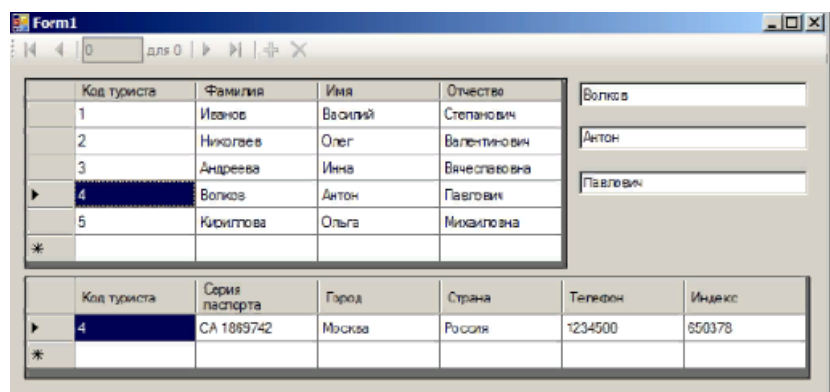
Для таблицы «Туристы» сделаем экранную форму, позволяющую отображать данные в компонентах TextBox и редактировать их. Для этого разместим на форме контейнер типа Panel, а на нем три компонента TextBox следующим образом.



Теперь необходимо осуществить привязку компонентов TextBox к соответствующим полям таблицы «Туристы». Для этого используем свойство из группы DataBindings – Advanced.



Для верхнего компонента TextBox в выпадающем списке Binding выберем источником данных «туристыBindingSource» и поле источника – «Фамилия». Для среднего и нижнего компонентов TextBox выберем тот же источник данных и поля «Имя» и «Отчество» соответственно.



Однако при внесении изменений все новые данные остаются только на форме. Это происходит потому, что данные были загружены в объект DataSet, который представляет собой копию таблицы в памяти. Все действия выполняются с этой копией. Для того чтобы изменения отобразились в базе данных, необходимо выполнить метод Update класса TableAdapter. Таким образом, в разрабатываемом приложении необходимо разместить кнопку «Обновить» и записать в обработчик события Click следующий программный код:

```
туристыTableAdapter.Update(bDTur_firmDataSet);
информация_o_туристахTableAdapter.Update(bDTur_firmDataSet);
```

Данный код обновляет информацию в таблицах «Туристы» и «Информация о туристах», предоставляемых источником данных

ПРАКТИЧЕСКАЯ РАБОТА № 46

Тема: Создание приложения с БД

Цель работы: научиться создавать клиентское приложение для работы с базой данных с применением встроенных инструментов в Visual Studio

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Содержание работы:

Задание 1. В каждом варианте необходимо разработать базу данных минимум из двух связанных между собой таблиц. В каждой таблице – не менее трех полей. Реализовать доступ к созданной базе с помощью управляемого провайдера OLE DB.

№ варианта	БД	Таблицы
1	Склад магазина	Товары, поставщики, категории товаров и т.п.
2	Владельцы автомобилей	Автомобили, автовладельцы и т.д.
3	Персонал предприятия	Сотрудники, отделы, документы отдела кадров и т.п.
4	Аптека	Лекарства, категории лекарств, виды болезней и т.п.
5	Библиотека	Книги, читатели, книги на руках у читателей и т.п.
6	Банковские кредиты	Заемщики, виды кредитов, поручители и т.п.
7	Гостиница	Список номеров, категории номеров, постояльцы и т.п.
8	Таксопарк	Транспортные средства, водители, рейсы и т.п.
9	Туристическая фирма	Виды туров, клиенты, заказы и т.п.
10	Кафе	Продукты, рецепты, поставщики и т.д.
11	ЖЭК	Дома, жильцы, виды обслуживания, заявки и т.п.
12	Кинопрокат	Кинотеатры, фильмы в прокате, жанры и т.п.
13	Учебные курсы	Области знаний курсов, преподаватели, курсы и т.д.
14	Банкомат	Карточки, виды операций, совершенные операции и т.п.
15	Семейный бюджет	Виды поступлений, виды затрат, покупки и т.д.

ПРАКТИЧЕСКАЯ РАБОТА № 47

Тема: Создание запросов к БД

Цель работы: научиться создавать запросы для работы с базой данных в Visual Studio

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Справочный материал:

Для извлечения таблиц и содержащихся в них данных используются SQL-запросы. Переменная CommandText содержит в себе SQL-запрос, синтаксис которого адаптирован для данного поставщика данных. Мы можем управлять извлечением данных, изменяя строку CommandText.

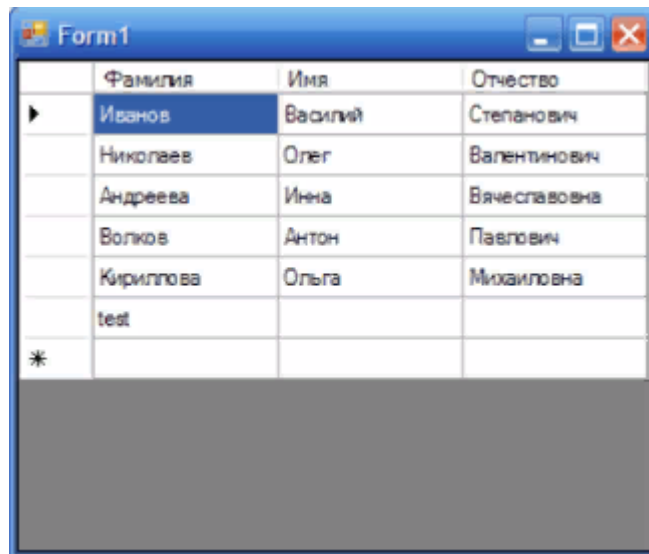
Содержание работы:

Задание 1. Создать запросы к БД «Туристы» из Практической работы № 45.

Если на экранной форме приложения столбец «Код туриста» отображать не нужно, то SQL-запрос будет выглядеть следующим образом:

```
string CommandText = "SELECT Фамилия, Имя, Отчество FROM Туристы";
```

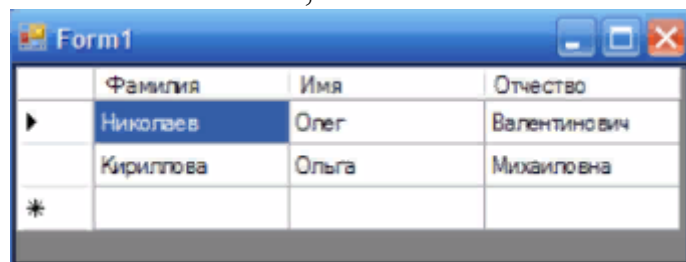
В окне работающего приложения теперь будут выводиться только соответствующие три поля.



	Фамилия	Имя	Отчество
▶	Иванов	Василий	Степанович
	Николаев	Олег	Валентинович
	Андреева	Инна	Вячеславовна
	Волков	Антон	Павлович
	Кириллова	Ольга	Михайловна
	test		
*			

Выведем теперь все записи клиентов, имена которых начинаются на "О":

```
string CommandText = "SELECT Фамилия, Имя, Отчество  
FROM Туристы where Имя like 'О%'";
```



	Фамилия	Имя	Отчество
▶	Николаев	Олег	Валентинович
	Кириллова	Ольга	Михайловна
*			

ПРАКТИЧЕСКАЯ РАБОТА № 48

Тема: Создание запросов к БД

Цель работы: научиться создавать запросы для работы с базой данных в Visual Studio

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Содержание работы:

Задание 1. Создать запросы к БД по индивидуальным заданиям из Практической работы № 46.

Информационное обеспечение обучения

Печатные издания:

Основные учебные издания:

1. Зыков, С. В. Введение в теорию программирования. Объектно-ориентированный подход: учебное пособие для СПО / С. В. Зыков. — Саратов: Профобразование, 2021. — 187 с. — ISBN 978-5-4488-0995-8. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование: [сайт]. — URL: <https://profspo.ru/books/102188>
2. Кариев, Ч. А. Разработка Windows-приложений на основе Visual C#: учебное пособие / Ч. А. Кариев. — 3-е изд. — Москва: Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2021. — 978 с. — ISBN 978-5-4497-0909-7. — Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. — URL: <https://www.iprbookshop.ru/102057.html>
3. Лебедева, Т. Н. Технология программирования: учебное пособие для СПО / Т. Н. Лебедева, С. С. Юнусова. — Саратов: Профобразование, 2019. — 140 с. — ISBN 978-5-4488-0351-2. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование: [сайт]. — URL: <https://profspo.ru/books/86081>

Дополнительные учебные издания:

4. Биллиг, В. А. Основы программирования на C#: учебное пособие / В. А. Биллиг. — 3-е изд. — Москва: Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2021. — 573 с. — ISBN 978-5-4497-0893-9. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование: [сайт]. — URL: <https://profspo.ru/books/102033>
5. Зубкова, Т. М. Технология разработки программного обеспечения: учебное пособие для СПО / Т. М. Зубкова. — Саратов: Профобразование, 2019. — 468 с. — ISBN 978-5-4488-0354-3. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование: [сайт]. — URL: <https://profspo.ru/books/86208>

Электронные издания (электронные ресурсы)

6. Учебники по программированию <http://programm.ws/index.php>

Электронно-библиотечная система:

7. ЭБС «IPRbooks», ООО «Ай Пи Ар Медиа»
8. ЭБС «Znanium»
9. ЭБС «PROФобразование»
10. ЭБС «Book.ru»