УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«МОГИЛЕВСКИЙ ГОСУДАРСТВЕННЫЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ»

УТВЕРЖДАЮ
Директор колледжа
_____С.Н.Козлов
29.08.2025

КОНСТРУИРОВАНИЕ ПРОГРАММ И ЯЗЫКИ ПРОГРАММИРОВАНИЯ

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ
ПО ИЗУЧЕНИЮ УЧЕБНОГО ПРЕДМЕТА
ЗАДАНИЯ НА ДОМАШНЮЮ КОНТРОЛЬНУЮ РАБОТУ № 1
ДЛЯ УЧАЩИХСЯ ЗАОЧНОЙ ФОРМЫ ПОЛУЧЕНИЯ ОБРАЗОВАНИЯ
ПО СПЕЦИАЛЬНОСТИ 5-04-0612-02
«РАЗРАБОТКА И СОПРОВОЖДЕНИЕ ПРОГРАММНОГО
ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ»

2025

Автор: Карманов А.В., преподаватель первой квалификационной категории учреждения образования «Могилевский государственный политехнический колледж»

Рецензент: Денисовец Д.А., преподаватель высшей квалификационной категории учреждения образования «Могилевский государственный политехнический колледж»

Разработано на основе учебной программы учреждения образования по учебному предмету профессионального компонента учебного плана учреждения образования по специальности 5-04-0612-02 «Разработка и сопровождение программного обеспечения информационных систем» для реализации образовательной программы среднего специального образования, обеспечивающей получение квалификации специалиста со средним специальным образованием, утвержденной директором колледжа, 2025

Обсуждено и одобрено
на заседании цикловой комиссии
специальностей в области программного
обеспечения информационных систем
Протокол № 13 от 29.08.2025___
Председатель цикловой комиссии
_____Д,А.Денисовец

Пояснительная записка

Для учащихся специальности «Разработка и сопровождение программного обеспечения информационных систем» необходимо получить навыки работы с объектно-ориентированными технологиями программирования с использованием современного языка программирования и интегрированной среды разработки. Для этого необходимо изучить основные приемы объектно-ориентированной технологии программирования, язык C#, среду программирования VisualStudio.NET, работу с XML-данными, развертывания Windows-приложений.

Изучение учебного предмета «Конструирование программ и языки программирования» основано на знаниях, полученных учащимися при изучении учебных предметов, таких как «Информатика», «Системное программное обеспечение». Программный учебный материал учебного предмета «Конструирование программ и языки программирования» тесно связан с программным учебным материалом учебных предметов «Основы алгоритмизации и программирования», «Структуры и алгоритмы обработки данных».

Учебный предмет «Конструирование программ и языки программирования» ставит своей целью изучение основ языка C#, приемы объектно-ориентированной технологии программирования и применения их на практике.

В качестве базового языка для изучения основ программирования выбран C# по следующим причинам:

- относительно небольшое количество базовых конструкций;
- широкие возможности для реализации объектно-ориентированных технологий;
- широкие возможности для создания классов, объектов и работы с ними;
- гибкие возможности для работы с пользовательскими функциями, типами данных, классами, объектами.

В результате изучения учебного предмета учащиеся должны знать на уровне представления:

- перспективы развития технологий создания программных средств;
- современные среды разработки программных средств для различных платформ;

знать на уровне понимания:

- методику создания программ;
- современную интегрированную среду разработки программного обеспечения;
- принципы объектно-ориентированного программирования;
- объектно-ориентированный язык программирования;
- процесс разработки программного обеспечения;

уметь:

- разрабатывать классы и приложения с использованием объектно-ориентированного языка программирования;
- создавать Windows-приложения, используя современную интегрированную среду разработки;
- организовывать доступ к базам данных из приложения;
- работать с XML-данными;
- создавать инсталляторы приложений;
- оформлять разработанные программные средства для дальнейшего внедрения и сопровождения.

В целях контроля усвоения программного учебного материала для учащихся предусмотрено выполнение 2 домашних контрольных работ, курсовой проект и сдача экзамена.

Общие методические рекомендации по выполнению домашней контрольной работы 1

Домашняя контрольная работа содержит 90 вариантов. Вариант работы выбирается в соответствии с двумя последними цифрами шифра учащегося по таблице вариантов. Каждый вариант содержит 4 задания: два теоретических вопроса и два практических задания.

Для выполнения домашней контрольной работы вначале изучается теория и приведенные примеры решения задач.

При оформлении домашней контрольной работы следует придерживаться следующих требований:

- работа выполняется в отдельной тетради или на листах А4 (шрифт 12-14, межстрочный интервал - одинарный). Следует пронумеровать страницы и оставить на них поля: справа – не менее 3 см для замечаний преподавателя, остальные поля – 2,5 см;
- на титульном листе указываются учебный предмет и номер работы, номер учебной группы, фамилия, имя, отчество учащегося, шифр;
- ответ следует начинать с номера и полного названия вопроса, он должен содержать схему алгоритма решения задачи, текст программы на языке C# с краткими, но достаточно обоснованными, пояснениям;
- чертежи и схемы следует выполнять аккуратно, соблюдая масштаб иГОСТ19.701-90 (ИСО 5807-85);
- в конце работы следует указать список используемых источников, которым вы пользовались, проставить дату выполнения работы и подпись;
- к домашней контрольной работе прикладывается диск с файлами, содержащими программы решения задачи;
- если в работе допущены недочеты или ошибки, то учащийся должен выполнить все указания преподавателя, сделанные в рецензии.
- домашняя контрольная работа должна быть выполнена в срок (в соответствии с учебным графиком);
- учащиеся, не имеющие зачета по домашней контрольной работе, к экзамену не допускаются;
- перед экзаменом зачтенная домашняя контрольная работа предоставляется преподавателю.

Критерии оценки домашней контрольной работы

Домашняя контрольная работа, признанная преподавателем удовлетворительной и содержащая 75% положенного объема, оценивается отметкой «зачтено».

Домашняя контрольная работа оценивается отметкой «не зачтено», если:

- выполнена не по варианту;
- не содержит схем алгоритмов;
- нет диска с программами;
- есть существенные недочеты в заданиях (в сумме более 25%);
- не выполнено одно задание (17%) и есть незначительные недочеты в остальных заданиях (в сумме более 10%).

Программа учебного предмета и методические рекомендации по ее изучению

Ведение

Цели и задачи учебного предмета «Конструирование программ и языки программирования», связь с другими учебными предметами

Классификационная характеристика современных языков программирования

Перспективы развития технологии конструирования программ

Литература: [1], с.3-15

Раздел 1 Объектно-ориентированное программирование и основы работы в интегрированной среде разработки

Тема 1.1 Принципы объектно-ориентированного программирования

Принципы объектно-ориентированного программирования (ООП).

Литература: [1], с.33-41; [4], с.25-32; [5], с.114-118; [3], с.11-21

Объектно-ориентированное программирование, или ООП — это одна из парадигм разработки. Парадигмой называют набор правил и критериев, которые соблюдают разработчики при написании кода. Если представить, что код — это рецепт блюда, то парадигма — то, как рецепт оформлен в кулинарной книге. Парадигма помогает стандартизировать написание кода. Это снижает риск ошибок, ускоряет разработку и делает код более читабельным для других программистов.

Суть понятия объектно-ориентированного программирования в том, что все программы, написанные с применением этой парадигмы, состоят из объектов. Каждый объект — это определённая сущность со своими данными и набором доступных действий.

Например, нужно написать для интернет-магазина каталог товаров. Руководствуясь принципами ООП, в первую очередь нужно создать объекты: карточки товаров. Потом заполнить эти карточки данными: названием товара, свойствами, ценой. И потом прописать доступные действия для объектов: обновление, изменение, взаимодействие.

Кроме ООП, существуют и другие парадигмы. Из них наиболее распространена функциональная, в которой работают не с объектами, а с функциями. Если использовать функциональную парадигму, чтобы

сделать каталог товаров, то начинать нужно не с карточек, а с функций, заполняющих эти карточки. То есть объект будет не отправной точкой, а результатом работы функции.

Обычно написать функцию быстрее, чем создавать объекты и прописывать взаимодействие между ними. Но если объём кода большой, работать с разрозненными функциями сложно.

В коде, написанном по парадигме ООП, выделяют четыре основных элемента:

1. Объект.

Часть кода, которая описывает элемент с конкретными характеристиками и функциями. Карточка товара в каталоге интернет-магазина — это объект. Кнопка «заказать» — тоже.

2. Класс.

Шаблон, на базе которого можно построить объект в программировании. Например, у интернет-магазина может быть класс «Карточка товара», который описывает общую структуру всех карточек. И уже из него создаются конкретные карточки — объекты.

Классы могут наследоваться друг от друга. Например, есть общий класс «Карточка товара» и вложенные классы, или подклассы: «Карточка бытовой техники», «Карточка ноутбука», «Карточка смартфона». Подкласс берёт свойства из родительского класса, например, цену товара, количество штук на складе или производителя. При этом имеет свои свойства, например, диагональ дисплея для «Карточки ноутбука» или количество сим-карт для «Карточки смартфона».

3. Метод.

Функция внутри объекта или класса, которая позволяет взаимодействовать с ним или другой частью кода. В примере с карточками товара метод может:

- заполнить карточку конкретного объекта нужной информацией.
- обновлять количество товара в наличии, сверяясь с бд.
- сравнивать два товара между собой.
- предлагать купить похожие товары.

4. Атрибут.

Характеристики объекта в программировании — например, цена, производитель или объём оперативной памяти. В классе прописывают, что такие атрибуты есть, а в объектах с помощью методов заполняют эти атрибуты данными.

Объектно-ориентированное программирование базируется на трёх

основных принципах, которые обеспечивают удобство использования этой парадигмы.

Инкапсуляция

Вся информация, которая нужна для работы конкретного объекта, должна храниться внутри этого объекта. Если нужно вносить изменения, методы для этого тоже должны лежать в самом объекте — посторонние объекты и классы этого делать не могут. Для внешних объектов доступны только публичные атрибуты и методы.

Например, метод для внесения данных в карточку товара должен обязательно быть прописан в классе «Карточка товара». А не в классе «Корзина» или «Каталог товаров».

Такой принцип обеспечивает безопасность и не даёт повредить данные внутри какого-то класса со стороны. Ещё он помогает избежать случайных зависимостей, когда из-за изменения одного объекта что-то ломается в другом.

Наследование

В этом принципе — вся суть объектно-ориентированного программирования. Разработчик создаёт:

- класс с определёнными свойствами;
- подкласс на его основе, который берёт свойства класса и добавляет свои;
- объект подкласса, который также копирует его свойства и добавляет свои.

Каждый дочерний элемент наследует методы и атрибуты, прописанные в родительском. Он может использовать их все, отбросить часть или добавить новые. При этом заново прописывать эти атрибуты и методы не нужно.

Например, в каталоге товаров:

1. У класса «Карточка товара» есть атрибуты тип товара, название, цена, производитель, а также методы «Вывести карточку» и «Обновить цену».
2. Подкласс «Смартфон» берёт все атрибуты и методы, записывает в атрибут «тип товара» слово «смартфон плюс добавляет свои атрибуты — «Количество сим-карт» и «Ёмкость аккумулятора».
3. Объект «Смартфон Xiaomi 11» заполняет все атрибуты своими значениями и может использовать методы класса «Карточка товара».

Наследование хорошо видно в примере кода выше, когда сначала создавали класс, потом подкласс, а затем объект с общими свойствами.

Полиморфизм

Один и тот же метод может работать по-разному в зависимости от объекта, где он вызван, и данных, которые ему передали. Например, метод «Удалить» при вызове в корзине удалит товар только из корзины, а при вызове в карточке товара — удалит саму карточку из каталога.

То же самое с объектами. Можно использовать их публичные методы и атрибуты в других функциях и быть уверенным, что всё сработает нормально.

Этот принцип ООП, как и другие, обеспечивает отсутствие ошибок при использовании объектов.

Перечислим преимущества и недостатки ООП

В парадигме объектов легче писать код. Удобно один раз создать класс или метод, а потом его использовать. Не нужно повторно переписывать десятки строк кода. Можно пользоваться специальными рекомендациями по написанию ООП-кода — SOLID.

Читать код гораздо проще. Даже в чужом коде обычно сразу видны конкретные объекты и методы, их удобно искать, чтобы посмотреть, что именно они делают.

Код легче обновлять. Класс или метод достаточно изменить в одном месте, чтобы он изменился во всех наследуемых классах и объектах. Не нужно переписывать каждый объект отдельно, выискивая, где именно в коде он расположен.

Программистам удобнее работать в команде. Разные люди могут отвечать за разные объекты и при этом пользоваться плодами трудов коллег.

Код можно переиспользовать. Один раз написанный класс или объект можно затем переносить в другие проекты. Достаточно однажды написать объект «Кнопка заказа» и потом можно вставлять его в почти неизменном виде в разные каталоги товаров и мобильные приложения.

Шаблоны проектирования. Именно на базе ООП построены готовые решения для взаимодействия классов друг с другом, которые позволяют не писать этот код с нуля, а взять шаблон.

Недостатки

Сложность в освоении. ООП сложнее, чем функциональное программирование. Для написания кода в этой парадигме нужно знать гораздо больше. Поэтому перед созданием первой рабочей программы придётся освоить много информации: разобраться в классах и наследовании, научиться писать публичные и внутренние функции, изучить способы взаимодействия объектов между собой.

Громоздкость. Там, где в функциональном программировании

хватит одной функции, в ООП нужно создать класс, объект, методы и атрибуты. Для больших программ это плюс, так как структура будет понятной, а для маленьких может оказаться лишней тратой времени.

Низкая производительность. Объекты потребляют больше памяти, чем простые функции и переменные. Скорость компиляции от этого тоже страдает.

Тема 1.2 Принципы объектно-ориентированного программирования. Язык программирования C# и платформа .NET

Становление и создание языка C#. Обзор архитектуры .NET Framework, основные идеи, принципы и возможности

Общеязыковая исполняющая среда CLR, общая система типов CTS, промежуточный язык CIL, общеязыковая инфраструктура. Интерфейс Командной строки (CLI)

Интегрированная среда разработки приложений. История развития. Основные возможности. Создание, компилирование, отладка и выполнение проектов в интегрированной среде разработки

Литература: [1], с.33-41; [4], с.25-32; [5], с.114-118; [3], с.11-21

Методические рекомендации

C# - это изящный объектно-ориентированный язык со строгой типизацией, позволяющий разработчикам создавать различные безопасные и надежные приложения, работающие на платформе .NET Framework. C# можно использовать для создания клиентских приложений Windows, XML-веб-служб, распределенных компонентов, приложений клиент-сервер, приложений баз данных и т. д. Visual C# предоставляет развитый редактор кода, удобные конструкторы пользовательского интерфейса, интегрированный отладчик и многие другие средства, которые упрощают разработку приложений на языке C# для платформы .NET Framework.

Синтаксис C# очень богат, но при этом прост и удобен в изучении. Характерные фигурные скобки C# мгновенно узнаются всеми, кто знаком с C, C++ или Java. Разработчики, знающие любой из этих языков, обычно очень быстро начинают эффективно работать в C#. Синтаксис C# упрощает многие сложности C++, но при этом

предоставляет отсутствующие в Java мощные функции, например обнуляемые типы значений, перечисления, делегаты, лямбда-выражения и прямой доступ к памяти. C# поддерживает универсальные методы и типы, которые обеспечивают более высокий уровень безопасности и производительности, а также итераторы, позволяющие определять в классах коллекций собственное поведение итерации, которое может легко применить в клиентском коде. Выражения LINQ создают очень удобную языковую конструкцию для строго типизированных запросов.

C# является объектно-ориентированным языком, а значит поддерживает инкапсуляцию, наследование и полиморфизм. Все переменные и методы, включая метод Main, представляющий собой точку входа в приложение, инкапсулируются в определения классов. Класс наследуется непосредственно из одного родительского класса, но может реализовывать любое число интерфейсов. Методы, которые переопределяют виртуальные методы родительского класса, должны содержать ключевое слово override, чтобы исключить случайное переопределение. В языке C# структура похожа на облегченный класс: это тип, распределяемый в стеке, реализующий интерфейсы, но не поддерживающий наследование.

Помимо этих основных принципов объектно-ориентированного программирования, C# предлагает ряд инновационных языковых конструкций, упрощающих разработку программных компонентов:

- инкапсулированные сигнатуры методов, именуемые делегатами, которые позволяют реализовать типобезопасные уведомления о событиях;
- свойства, выполняющие функцию акцессоров для закрытых переменных-членов;
- атрибуты, предоставляющие декларативные метаданные о типах во время выполнения;
- внутристрочные комментарии для XML-документации;
- LINQ для создания запросов к различным источникам данных.

Для взаимодействия с другим программным обеспечением Windows, например с объектом COM или собственными библиотеками DLL Win32, вы можете применить процесс C#, известный как "Взаимодействие". Взаимодействие позволяет программам на C# делать практически все, что возможно в приложении машинного кода C++. C# поддерживает даже указатели и понятие "небезопасного" кода для тех случаев, в которых критически важен прямой доступ к памяти.

Процесс построения в C# проще по сравнению с C или C++, но

более гибок, чем в Java. Отдельные файлы заголовка не используются, и нет необходимости объявлять методы и типы в определенном порядке.Исходный файл C# может определить любое число классов, структур, интерфейсов и событий.

Архитектура платформы .NET Framework.

Программы C# выполняются на платформе .NET Framework, которая интегрирована в Windows и содержит виртуальную общеязыковую среду выполнения (среду CLR) и унифицированный набор библиотек классов.Среда CLR корпорации Майкрософт представляет собой коммерческую реализацию международного стандарта Common Language Infrastructure (CLI), который служит основой для создания сред выполнения и разработки, позволяющих совместно использовать разные языки и библиотеки.

Исходный код, написанный на языке C# компилируется в промежуточный язык (IL), который соответствует спецификациям CLI. Код на языке IL и ресурсы, в том числе точечные рисунки и строки, сохраняются на диск в виде исполняемого файла (обычно с расширением .exe или .dll). Такой файл называется сборкой.Сборка содержит манифест с информацией о типах, версии, требований безопасности, языке и региональных параметрах для этой сборки.

При выполнении программы C# среда CLR загружает сборку и выполняет различные действия в зависимости от сведений, сохраненных в манифесте. Если выполняются все требования безопасности, среда CLR выполняет JIT-компиляцию из кода на языке IL в инструкции машинного языка. Также среда CLR выполняет другие операции, например автоматическую сборку мусора, обработку исключений и управление ресурсами. Код, выполняемый средой CLR, иногда называют "управляемым кодом", чтобы подчеркнуть отличия этого подхода от "неуправляемого кода", который сразу компилируется в машинный язык для определенной системы. На рисунке 1 показаны связи между файлами исходного кода C#, библиотеками классов .NET Framework, сборками и средой CLR, существующие во время компиляции и во время выполнения.

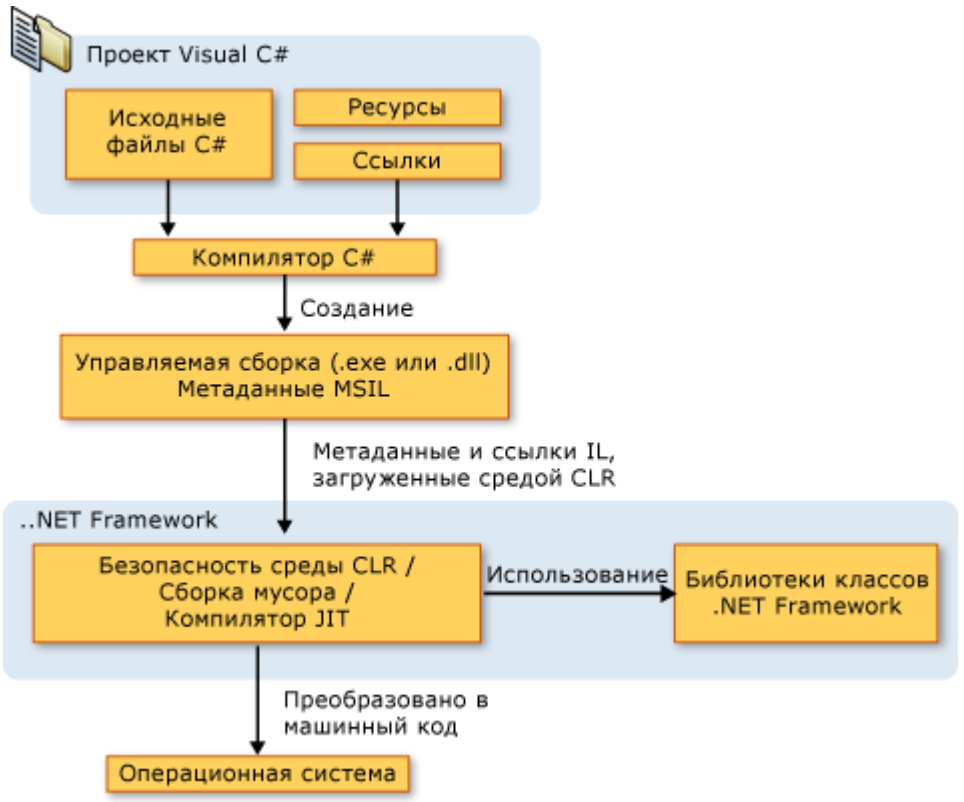

Рисунок 1 – связи между файлами исходного кода C#, библиотеками классов .NET Framework, сборками и средой CLR, существующие во время компиляции и во время выполнения

Взаимодействие между языками является важнейшей возможностью .NET Framework. Создаваемый компилятором C# код IL соответствует спецификации общих типов (CTS). Это означает, что этот код IL может успешно взаимодействовать с кодом, созданным из Visual Basic и Visual C++ для платформы .NET или из любого другого CTS-совместимого языка, которых существует уже более 20. Одна сборка может содержать несколько модулей, написанных на разных языках .NET, и все типы могут ссылаться друг на друга, как если бы они были написаны на одном языке.

Помимо служб времени выполнения, платформа .NET Framework содержит обширную библиотеку, в которую входит более 4000 классов. Эти классы распределены по пространствам имен, соответствующим разным полезных функциям: от операций файлового ввода и вывода до управления строками, от синтаксического анализа XML до элементов управления Windows Forms. Обычно приложения C# активно используют библиотеку классов .NET Framework для решения типовых задач взаимодействия.

Интегрированная среда (integrated development environment - IDE) - набор инструментов для разработки и отладки программ, имеющий общую интерактивную графическую оболочку, поддерживающую выполнение всех основных функций жизненного цикларазработки программы - набор и редактирование исходного текста (кода), компиляцию (сборку), исполнение, отладку, профилирование и др.

Использование интегрированной среды - один из возможных подходов к разработке программ. Альтернативой ему является более ранний, традиционный подход системы UNIX, основанный на использовании набора инструментов (toolkit, toolbox), родственных по тематике и функциональности, но не объединенных в одну интегрированную интерактивную среду и подчас (в ранних версиях системы UNIX) вызываемых в режиме командной строки (command line interface).

Разумеется, использовать интегрированную среду гораздо удобнее для разработчика, чем и объясняется бурное развитие и разнообразие интегрированных сред, начиная с 1980-х годов.

Одной из первых интегрированных сред стала среда Turbo Pascal фирмы Borland, руководителем разработки которой в середине 1980-х гг. стал Филипп Кан, ученик Никлауса Вирта.

Корпорация Microsoft внесла особо выдающийся вклад в развитие интегрированных сред, благодаря созданию и развитию среды Visual Studio, которая является одним из лучших образцов современной интегрированной среды. Ее новую версию, Visual Studio 2013, мы и рассмотрим в данном курсе.

Идея интегрированных сред достигла еще большего развития к середине 1980-х гг., когда появились две группы популярных интегрированных сред:

- Турбо-среды (Turbo Pascal, Turbo C, Turbo C++, Delphi и др.) фирмы Borland для поддержки программирования на этих языках, реализованные сначала для операционной системы MS DOS, затем - для ОС Windows;
- GNU Emacs [2] - многоязыковая и многоплатформная интегрированная среда разработки, реализованная для MS DOS, затем для Windows, OpenVMS и для Linux. Среди сотрудников моей группы разработчиков, работавших с фирмой Sun Microsystems в 1990-х гг., было немало пользователей и энтузиастов среды GNU Emacs, благодаря ее реализации для платформы Solaris.

Следует также упомянуть интегрированную среду тех лет для

разработки программ на объектно-ориентированном языке Smalltalk фирмы Xeror PARC - одну из первых интегрированных сред ООП, в которой впервые появилось понятие байт-кода как бинарной постфиксной формы промежуточного представления программы и понятие just-in-time (JIT, динамического) компилятора, выполняющего при первом вызове метода его компиляцию в платформно-зависимый (native) код целевого компьютера.

Суммируем теперь основные возможности интегрированных сред разработки программ. Для каждой из них характерно наличие следующих компонент:

- единая интерактивная оболочка, обеспечивающая вызов всех других компонент, не выходя из среды, с широким использованием функциональных клавиш;
- текстовый редактор для набора и редактирования исходных текстов программ. В недавнем прошлом в отечественной традиции использовался именно термин исходный текст, впоследствии стал использоваться термин исходный код (source code);
- система поддержки сборки (build), то есть компиляции проектов из исходных кодов, включающая компилятор с исходного реализуемого языка и компоновщик (linker) объектных бинарных кодов в единый исполняемый код (загрузочный модуль); компоновщик используется либо штатный, входящий в состав операционной системы, либо специфичный для данной среды;

отладчик (debugger) для отладки программ в среде с помощью типичного набора команд: установить контрольную точку остановки; остановиться в заданной процедуры (методе); визуализировать значения переменных (или, на более низком уровне, регистров и областей памяти).

Вопросы для самоконтроля

1 Опишите основные принципы объектно-ориентированного программирования.

2 Опишите основные принципы и возможности архитектуры .NET Framework.

3 Дайте характеристику понятию общеязыковой исполняющей среды, общей системы типов, промежуточного языка, общеязыковой инфраструктуры.

4 Опишите сущность понятия «интегрированная среда разработки» приложений.

5 Опишите создание, компилирование, отладку и выполнение

проектов в интегрированной среде разработки.

Раздел 2 Основы программирования на языке C#

Тема 2.1 Система типов. Структура программы. Переменные, операции и выражения. Операторы

Типы данных (обозначение, диапазон представления, занимаемый размер в байтах). Классификация типов. Встроенные типы. Совместимость типов. Преобразование типов. Преобразования строкового типа. Класс Convert и его методы

Структура программы. Объявление переменных и их инициализация. Константы. Время жизни и область видимости переменных. Выражения. Правила построения выражений. Операции и их приоритеты.

Операторы. Простые и составные операторы. Пустой оператор. Условный и безусловный переход. Операторы выбора. Операторы цикла. Операторы передачи управления

Литература: [2], с.52; [3], с.31-36; [3], с.38-59; [5], с.37-40; [1], с.81-89

Методические рекомендации

Изучая материалы данной темы, следует обратить внимание, на сведения представленные ниже.

C# является языком, в котором учитывается регистр символов.

Примитивные типы, доступные в языке C#, показаны в таблице 1.

Таблица 1 – Примитивные типы

Примитивный тип	Псевдоним	Описание
Boolean	bool	Может содержать значение true или false. Языковые конструкции if, while и do-while требуют использования этого типа
Byte	byte	Целочисленный тип, содержащий беззнаковое 8-битное значение

Продолжение таблицы 1

Примитивный тип	Псевдоним	Описание
Char	char	Символьный тип, содержащий 16-битный Unicode-символ
Decimal	decimal	Высокоточный тип для финансовых и научных вычислений
Double	double	Значение с плавающей точкой двойной точности
Single	float	Значение с плавающей точкой одинарной точности
Int32	int	32-битное значение со знаком
Int64	long	64-битное значение со знаком
SByte	sbyte	8-битное значение со знаком
Int16	short	16-битное значение со знаком
UInt32	uint	32-битное беззнаковое значение
UInt64	ulong	64-битное беззнаковое значение
String	string	Строка, содержащая набор символов
Object	object	Базовый тип для всех типов в управляемом коде

В таблице 2 показаны операторы, которые можно использовать в выражениях на языке C#.

Таблица 2 - Операторы

Категория оператора	Операторы
Арифметический	+ - * / %
Логический (булев или битовый)	& | ^ ! ~ && || true false
Объединение строк	+
Инкремент и декремент	++ --
Сдвиг	<< >>
Отношения	== != < > <= >=
Присваивания	= += -= *= /= %= &= |= ^= <<= >>=
Доступ к членам	.
Индексация	[]
Преобразование типов	()
Операторы условия	?:

Продолжение таблицы 2

Категория оператора	Операторы
Задание и удаление делегатов	+ -
Создание объектов	new
Информация о типах	is sizeof typeof
Управление исключениями	checked unchecked
Адресация	* -> [] &

Методы

Можно выделить два основных типа методов - статические методы и методы экземпляров. Статические методы объявляются с использованием ключевого слова static и могут вызываться без создания экземпляра типа, в котором они реализованы.

Переменная — это именованная область памяти, предназначенная для хранения данных определенного типа. Во время выполнения программы значение переменной можно изменять. Все переменные, используемые в программе, должны быть описаны явным образом. При описании для каждой переменной задаются ее имя и тип.

Пример описания целой переменной с именем a и вещественной переменной x:

int a; float x;

Имя переменной служит для обращения к области памяти, в которой хранится значение переменной. Имя дает программист. Тип переменной выбирается, исходя из диапазона и требуемой точности представления данных. При объявлении можно присвоить переменной некоторое начальное значение, то есть инициализировать ее, например:

int a, b = 1;

float x = 0.1, y = 0.1f;

Здесь описаны:

переменная a типа int, начальное значение которой не присваивается;

переменная b типа int, ее начальное значение равно 1;

переменные x и y типа float, которым присвоены одинаковые начальные значения 0.1. Разница между ними состоит в том, что для

инициализации переменной x сначала формируется константа типа double, а затем она преобразуется к типуfloat ; переменной y значение 0.1 присваивается без промежуточного преобразования.

Рекомендуется всегда инициализировать переменные при описании. При инициализации можно использовать не только константу, но и выражение - главное, чтобы на момент описания оно было вычисляемым, например:

int b = 1, a = 100;

int x = b * a + 25;

Операции и выражения

Выражение – это правило вычисления значения. В выражении участвуют операнды, объединенные знаками операций. Операндами простейшего выражения могут быть константы, переменные и вызовы функций.

Например, a + 2 - это выражение, в котором + является знаком операции, а a и 2 - операндами. Пробелы внутри знакаоперации, состоящей из нескольких символов, не допускаются.

Операции в выражении выполняются в определенном порядке в соответствии с приоритетами, как и в математике. Результат вычисления выражения характеризуется значением и типом.

Изучая материалы данной темы, следует обратить внимание, на то что:

1 Инструкция if используется для выбора одного из двух направлений дальнейшего хода программы.

2Выбор последовательности инструкций осуществляется в зависимости or значения условия - заключенного в скобки выражения, записанного после if.

3 Инструкция, записанная после else, выполняется в том случае, если значение выражения условие равно нулю, во всех остальных случаях выполняется инструкция, следующая за условием.

4 Если при соблюдении или несоблюдении условия надо выполнить несколько инструкций программы, то эти инструкции следует объединить в группу - заключить в фигурные скобки.

5 При помощи вложенных одна в другую нескольких инструкций if можно реализовать множественный выбор.

Вопросы для самоконтроля

1 Перечислите типы данных C#.
2 Перечислите базовые операции и выражения C#.
3 Опишите структуру линейной программы C#.
4 Опишите операторы цикла.
5 Опишите базовые конструкции структурного программирования.
6 Опишите правила построения выражений на языке C#.
7 Перечислите и описать операции отношений.
8 Опишите ввод и вывод.
9 Опишите именованные константы.
10 Опишите условный оператор if.
11 Опишите оператор выбора switch.
12 Опишите оператор цикла for.
13 Опишите операторы цикла while, do/while.

Тема 2.2 Сборки. Атрибуты. Директивы. Пространства имен

Сущность сборки, манифеста сборки, атрибута, пространства имен, рефлексии, директивы препроцессора

Однофайловые и многофайловые сборки. Приватные и разделяемые сборки. Рефлексия. Класс System.Attribute. Директивы препроцессора

Литература: [3], с.272-290

Методические рекомендации

Атрибут – средство добавления декларативной информации к элементам программного кода. Назначение атрибутов – внесение всевозможных не предусмотренных обычным ходом выполнения приложения изменений:

- описание взаимодействия между модулями;
- дополнительная информация, используемая при работе с данными (управление сериализацией);
- отладка.

Эта декларативная информация составляет часть метаданных кода. Она может быть использована при помощи механизмов отражения.

Структура атрибута регламентирована. Атрибут – это класс. Общий предок всех атрибутов – класс System.Attribute.

Информация, закодированная с использованием атрибутов, становится доступной в процессе ОТРАЖЕНИЯ (рефлексии типов).

Атрибуты типизированы.

.NET способна прочитать информацию в атрибутах и использовать ее в соответствии с предопределенными правилами или замыслами разработчика. Различаются:

- предопределенные атрибуты. В .NET реализовано множество атрибутов с предопределенными значениями:

1 DllImport – для загрузки .dll-файлов;

2 Serializable – означает возможность сериализации свойств объекта представителя класса;

3 NonSerialized – обозначает данные-члены класса как несериализуемые. Карандаши (средство графического представления информации, элемент GDI+) не сериализуются;

- производные (пользовательские) атрибуты могут определяться и использоваться в соответствии с замыслами разработчика. Возможно создание собственных (пользовательских) атрибутов. Главные условия:

1 соблюдение синтаксиса;

2 соблюдение принципа наследования.

В основе пользовательских атрибутов – все та же система типов с наследованием от базового класса System.Attribute.

Информация, содержащаяся в атрибутах, предназначается для CLR, и она в случае необходимости должна суметь разобраться в этой информации. Пользователи или другие инструментальные средства должны уметь кодировать и декодировать эту информацию.

Добавлять атрибуты можно к:

- сборкам;
- классам;
- элементам класса;
- структурам;
- элементам структур;
- параметрам;
- возвращаемым значениям.

Ниже приводится описание членов класса Attribute.

Открытые свойства.

TypeId – при реализации в производном классе получает уникальный идентификатор для этого атрибута Attribute.

Открытые методы.

Equals – переопределен.

GetCustomAttribute – перегружен. Извлекает пользовательский атрибут указанного типа, который применен к заданному члену класса.

GetCustomAttributes – перегружен. Извлекает массив пользовательских атрибутов указанного типа, которые применены к заданному члену класса.

GetHashCode – переопределен. Возвращает хэш-код для этого экземпляра.

GetType (унаследовано от Object) – возвращает Type текущего экземпляра.

IsDefaultAttribute – при переопределении в производном классе возвращает значение, показывающее, является ли значение этого производного экземпляра значением по умолчанию для производного класса.

IsDefined – перегружен. Определяет, применены ли какие-либо пользовательские атрибуты заданного типа к указанному члену класса.

Match – при переопределении в производном классе возвращает значение, указывающее, является ли этот экземпляр эквивалентным заданному объекту.

ToString (унаследовано от Object) – возвращает String, который представляет текущий Object.

Защищенные конструкторы.

Attribute –конструктор – инициализирует новый экземпляр класса Attribute.

Защищенные методы

Finalize (унаследовано от Object) – переопределен. Позволяет объекту Object попытаться освободить ресурсы и выполнить другие завершающие операции, перед тем как объект Object будет уничтожен в процессе сборки мусора. В C# для функций финализации используется синтаксис деструктора.

MemberwiseClone (унаследовано от Object) – создает неполную копию текущего Object.

Вопросы для самоконтроля

1 Опишите сущность сборки, манифеста сборки, атрибута, пространства имен, рефлексии, директивы препроцессора.

2 Дайте характеристику классу System.Attribute.

Тема 2.3 Массивы. Класс Array

Массив

Статические и динамические массивы. Одномерные, многомерные и ступенчатые массивы. Способы описания и создания. Родительский класс Array, его методы и свойства

Литература: [1], с.65-68, 302-309

Методические рекомендации

Массив – это конечная группа элементов одного типа, имеющая общее имя.

Массивы относятся к ссылочным типам данных. Массивы построены на основе класса System.Array, поэтому любой массив получает методы и свойства класса Array, что значительно упрощает работу с массивами. Работа с массивами более безопасна, поскольку контролируется выход за границы массива. По умолчанию элементам массива присваиваются начальные значения:

- для арифметических типов – 0;
- для ссылочных типов – null;
- для символов – пробел;
- для логических - false.

Имеются одномерные, многомерные и ступенчатые массивы.

Одномерный массив

тип[] имя_массива = new тип [размер];

Например: int[] a=new int [10];

Возможно объявление массива с инициализацией:

тип [] имя _массива={список инициализации};

Например: int[] a= {0, 1, 2, 3};

Массивы и исключения

Выход за границы массива в C# расценивается как ошибка, в ответ на которую генерируется исключение - IndexOutOfRangeException.

Массив как объект

Базовым классом для массивов является класс Array, определенный в пространстве имен System. Данный класс содержит различные свойства и методы:

- Length - количество элементов массива (по всем размерностям);
- Rank - размерность массива;
- SetValue(value, index) - установка значения элемента массива;
- GetValue (index) - получение значения элемента массива;
- Clear - присваивание элементам массива значений по

умолчанию;

- Copy - копирование заданного диапазона элементов одного массива в другой;
- CopyTo - копирование всех элементов текущего одномерного массива в другой массив;
- IndexOf - поиск первого вхождения элемента в одномерный массив;
- LastIndexOf - поиск последнего вхождения элемента в одномерный массив;
- Reverse - изменение порядка следования элементов на обратный;
- Sort - упорядочивание элементов одномерного массива;
- Resize - заменяет размер массива указанным новым размером.

Многомерные массивы. Объявить двумерный массив можно одним из предложенных способов:

тип [,] a = new тип [размер 1, размер 2];

тип [,] a ={{эл - ты 1-ой строки}, … , {эл-ты n-ой строки}};

тип [,] a = new тип [,]{{эл-ты 1-ой строки}, … ,{эл-ты n-ой строки}};

Ступенчатые массивы. У ступенчатых массивов задается столько пар квадратных скобок, сколько размерностей у массива.

Определение ступенчатых массивов:

```
int[][] arr = new int [3][]; //Объявляем ступенчатый массив
arr[0]=new int[3]; //Определяем нулевой элемент
arr[1]=new int[2]; //Определяем первый элемент
arr[2] =new int[5]; //Определяем второй элемент
```

Каждый элемент представляет собой одномерный массив целых чисел. Первый элемент массива состоит из трех целях чисел, второй - из двух и третий - из пяти.

Для заполнения элементов массива значениями можно также использовать инициализаторы, при этом размер массива знать не требуется.

Вопросы для самоконтроля

1 Дайте определение понятию «массив».
2 Опишите статические и динамические массивы.
3 Опишите одномерные, многомерные и ступенчатые массивы.
4 Опишите способы описания и создания массивов.

5 Опишите родительский класс Array, его методы и свойства.

Тема 2.4 Методы C#

Понятие метода. Виды методов. Описание методов. Атрибуты доступа. Параметры методов и их статус. Тело метода. Семантика вызова. Перезагрузка методов. Встроенные методы

Литература: [1], с.95-108; [2], с.156-169

Методические рекомендации

Изучая материалы данной темы, следует обратить внимание, на то что:

1 Процедуры и функции связываются теперь с классом, они обеспечивают требуемую функциональность класса и называются методами класса. Поскольку класс в объектно-ориентированном программировании рассматривается как некоторый тип данных, то главную роль в классе начинают играть его данные – поля класса, задающие свойства объектов класса. Методы класса «служат» данным, занимаясь их обработкой. Помните, в C# процедуры и функции существуют только как методы некоторого класса, они не существуют вне класса.

2 В языке C# нет специальных ключевых слов – method, procedure, function, но сами понятия конечно же присутствуют. Синтаксис объявления метода позволяет однозначно определить, чем является метод – процедурой или функцией.

3 Прежнюю роль библиотек процедур и функций теперь играют библиотеки классов. Библиотека классов FCL, доступная в языке C#, существенно расширяет возможности языка. Знание классов этой библиотеки, методов этих классов совершенно необходимо для практического программирования на C#, использование всей его мощи.

Описание методов (процедур и функций).

Синтаксис

Синтаксически в описании метода различают две части – описание заголовка и описание теламетода:

заголовок_метода

тело_метода

Рассмотрим синтаксис заголовка метода:

[атрибуты][модификаторы]{void| тип_результата_функции}

имя_метода([список_формальных_аргументов])

Имя метода и список формальных аргументов составляют сигнатуру метода. Заметьте, в сигнатуру не входят имена формальных аргументов, здесь важны типы аргументов. В сигнатуру не входит и тип возвращаемого результата.

Квадратные скобки (метасимволы синтаксической формулы) показывают, что атрибуты и модификаторы могут быть опущены при описании метода. Подробное их рассмотрение будет дано в лекциях, посвященных описанию классов. Сейчас же упомяну только об одном из модификаторов – модификаторе доступа. У него четыре возможных значения, из которых пока рассмотрим только два – public и private. Модификатор public показывает, что метод открыт и доступен для вызова клиентами и потомками класса. Модификатор private говорит, что метод предназначен для внутреннего использования в классе и доступен для вызова только в теле методов самого класса. Заметьте, если модификатор доступа опущен, то по умолчанию предполагается, что он имеет значение private и метод является закрытым для клиентов и потомков класса.

Обязательным при описании заголовка является указание типа результата, имени метода и круглых скобок, наличие которых необходимо и в том случае, если сам список формальных аргументов отсутствует. Формально тип результата метода указывается всегда, но значение void однозначно определяет, что метод реализуется процедурой. Тип результата, отличный от void, указывает на функцию. Вот несколько простейших примеров описания методов:

```
void A() {…};
int B(){…);
public void C(){…};
```

Методы A и B являются закрытыми, а метод C – открыт. Методы A и C реализованы процедурами, а метод B – функцией, возвращающей целое значение.

Тело метода

Синтаксически тело метода является блоком, представляющим последовательность операторов и описаний переменных, заключенную в фигурные скобки. Если речь идет о теле функции, то в блоке должен быть хотя бы один оператор, возвращающий значение функции в форме return <выражение>.

Переменные, описанные в блоке, считаются локализованными в этом блоке. В записи операторов блока участвуют имена локальных

переменных блока, имена полей класса и имена аргументов метода.

Знание семантики описаний и операторов достаточно для понимания семантики блока. Необходимые уточнения будут даны чуть позже.

Вызов метода. Синтаксис

Как уже отмечалось, метод может вызываться в выражениях или быть вызван как оператор тела блока. В качестве оператора может использоваться любой метод – как процедура, так и функция. Конечно, функцию разумно вызывать как оператор только, если она обладает побочным эффектом. В последнем случае она вызывается ради своего побочного эффекта, а возвращаемое значение никак не используется. Любое выражение с побочным эффектом может выступать в роли оператора, классическим примером является оператор x++;.

Если же попытаться вызвать процедуру в выражении, то это приведет к ошибке еще на этапе компиляции. Возвращаемое процедурой значение void не совместимо с выражениями. Так что в выражениях могут быть вызваны только функции.

Сам вызов метода, независимо от того, процедура это или функция, имеет один и тот же синтаксис:

имя_метода([список_фактических_аргументов])

Если это оператор, то вызов завершается точкой с запятой.

Вопросы для самоконтроля

1 Опишите методы класса.
2 Опишите параметры методов.
3 Опишите встроенные функции.
4 Опишите способы передачи параметров в методы.

Тема 2.5 Символы и строки постоянной длины. Классы String и StringBuilder

Символы и строки C#

Строки постоянной и переменной длины. Классы Char, Char[], String для работы со строками. Изменяемые и неизменяемые строковые классы. Классы .Net Framework, расширяющие строковый тип. Класс StringBuilder

Литература: [1], с.223-231

Методические рекомендации

В C# есть символьный класс Char, основанный на классе System.Char и использующий двухбайтную кодировку Unicode представления символов. Для этого типа в языке определены символьные константы - символьные литералы. Константу можно задавать:

- символом, заключенным в одинарные кавычки;
- escape-последовательностью, задающей код символа;
- Unicode-последовательностью, задающей Unicode-код символа.

Вот несколько примеров объявления символьных переменных и работы с ними:

```
public void TestChar()
{
char ch1='A', ch2 ='\x5A', ch3='\u0058';
char ch = new Char();
int code; string s;
ch = ch1;
//преобразование символьного типа в тип int
code = ch; ch1=(char) (code +1);
//преобразование символьного типа в строку
//s = ch;
s = ch1.ToString()+ch2.ToString()+ch3.ToString();
Console.WriteLine("s= {0}, ch= {1}, code = {2}",
s, ch, code);
}//TestChar
```

Три символьные переменные инициализированы константами, значения которых заданы тремя разными способами. Переменная ch объявляется в объектном стиле, используя new и вызов конструктора класса. Тип char, как и все типы C#, является классом. Этот класс наследует свойства и методы класса Object и имеет большое число собственных методов.

Явные или неявные преобразования между классами char и string отсутствуют, но, благодаря методу ToString, переменные типа char стандартным образом преобразуются в тип string.

В результате работы процедуры TestChar строка s, полученная сцеплением трех символов, преобразованных в строки, имеет значение BZX, переменная ch равна A, а ее код - переменная code - 65.

Не раз отмечалось, что семантика присваивания справедлива при

вызове методов и замене формальных аргументов на фактические. Приведу две процедуры, выполняющие взаимно-обратные операции - получение по коду символа и получение символа по его коду:

```
public int SayCode(char sym)
{
return (sym);
}//SayCode
public char SaySym(object code)
{
return ((char)((int)code));
}// SaySym
```

В первой процедуре преобразование к целому типу выполняется неявно. Во второй - преобразование явное. Ради универсальности она слегка усложнена. Формальный параметр имеет тип Object, что позволяет передавать ей в качестве аргумента код, заданный любым целочисленным типом. Платой за это является необходимость выполнять два явных преобразования.

Класс string не разрешает изменять существующие объекты. Строковый класс StringBuilder позволяет компенсировать этот недостаток. Этот класс принадлежит к изменяемым классам и его можно найти в пространстве имен System.Text. Рассмотримкласс StringBuilder подробнее.

Объявление строк. Конструкторы класса StringBuilder

Объекты этого класса объявляются с явным вызовом конструктора класса. Поскольку специальных констант этого типа не существует, то вызов конструктора для инициализации объекта просто необходим. Конструктор класса перегружен, и наряду с конструктором без параметров, создающим пустую строку, имеется набор конструкторов, которым можно передать две группы параметров. Первая группа позволяет задать строку или подстроку, значением которой будет инициализироваться создаваемый объект класса StringBuilder. Вторая группа параметров позволяет задать емкость объекта - объем памяти, отводимой данному экземпляру класса StringBuilder. Каждая из этих групп не является обязательной и может быть опущена. Примером может служить конструктор без параметров, который создает объект, инициализированный пустой строкой, и с некоторой емкостью , заданной по умолчанию, значение которой зависит от реализации. Приведу в качестве примера синтаксис трех конструкторов:

public StringBuilder (string str, int cap). Параметр str задает строку

инициализации, cap - емкость объекта ;

public StringBuilder (int curcap, int maxcap). Параметры curcap и maxcap задают начальную и максимальнуюемкость объекта ;

public StringBuilder (string str, int start, int len, int cap). Параметры str, start, len задают строку инициализации, cap - емкость объекта.

Операции над строками

Над строками этого класса определены практически те же операции с той же семантикой, что и над строками класса String:

присваивание (=);

две операции проверки эквивалентности (==) и (!=);

взятие индекса ([]).

Операция конкатенации (+) не определена над строками класса StringBuilder, ее роль играет метод Append, дописывающий новую строку в хвост уже существующей.

Со строкой этого класса можно работать как с массивом, но, в отличие от класса String, здесь уже все делается как надо: допускается не только чтение отдельного символа, но и его изменение. Рассмотрим с небольшими модификациями наш старый пример:

```
public void TestStringBuilder()
{
//Строки класса StringBuilder
//операции над строками
StringBuilder s1 =new StringBuilder("ABC"),
s2 =new StringBuilder("CDE");
StringBuilder s3 = new StringBuilder();
//s3= s1+s2;
s3= s1.Append(s2);
bool b1 = (s1==s3);
char ch1 = s1[0], ch2=s2[0];
Console.WriteLine("s1={0}, s2={1}, b1={2}," +
"ch1={3}, ch2={4}", s1,s2,b1,ch1,ch2);
s2 = s1;
b1 = (s1!=s2);
ch2 = s2[0];
Console.WriteLine("s1={0}, s2={1}, b1={2}," +
"ch1={3}, ch2={4}", s1,s2,b1,ch1,ch2);
StringBuilder s = new StringBuilder("Zenon");
```

```
s[0]='L';
Console.WriteLine(s);
}//TestStringBuilder
```

Этот пример демонстрирует возможность выполнения над строками класса StringBuilder тех же операций, что и над строками класса String. В результате присваивания создается дополнительная ссылка на объект, операции проверки на эквивалентность работают со значениями строк, а не со ссылками на них. Конкатенацию можно заменить вызовом метода Append. Появляется новая возможность - изменять отдельные символы строки. (Для того чтобы имя класса StringBuilder стало доступным, в проект добавлено предложение using System.Text, ссылающееся на соответствующее пространство имен.)

Основные методы

У класса StringBuilder методов значительно меньше, чем у класса String. Это и понятно - класс создавался с целью дать возможность изменять значение строки. По этой причине у класса есть основные методы, позволяющие выполнять такие операции над строкой как вставка, удаление и замена подстрок, но нет методов, подобных поиску вхождения, которые можно выполнять над обычными строками. Технология работы обычно такова: конструируется строка класса StringBuilder ; выполняются операции, требующие изменение значения; полученная строка преобразуется в строку класса String ; над этой строкой выполняются операции, не требующие изменения значения строки. Давайте чуть более подробно рассмотрим основные методы класса StringBuilder:

public StringBuilder Append (<объект>). К строке, вызвавшей метод, присоединяется строка, полученная из объекта, который передан методу в качестве параметра. Метод перегружен и может принимать на входе объекты всех простых типов, начиная от char и bool до string и long. Поскольку объекты всех этих типов имеют метод ToString, всегда есть возможность преобразовать объект в строку, которая и присоединяется к исходной строке. В качестве результата возвращается ссылка на объект, вызвавший метод. Поскольку возвращаемую ссылку ничему присваивать не нужно, то правильнее считать, что метод изменяет значение строки;

public StringBuilder Insert (int location,<объект>). Метод вставляет строку, полученную из объекта, в позицию, указанную параметром location. Метод Append является частным случаем

метода Insert ;

public StringBuilder Remove (int start, int len). Метод удаляет подстроку длины len, начинающуюся с позицииstart ;

public StringBuilder Replace (string str1,string str2). Все вхождения подстроки str1 заменяются на строку str2 ;

public StringBuilder AppendFormat (<строка форматов>, <объекты>). Метод является комбинацией метода Format класса String и метода Append. Строка форматов, переданная методу, содержит только спецификации форматов. В соответствии с этими спецификациями находятся и форматируются объекты. Полученные в результате форматирования строки присоединяются в конец исходной строки.

За исключением метода Remove, все рассмотренные методы являются перегруженными. В их описании дана схема вызова метода, а не точный синтаксис перегруженных реализаций. Приведу примеры, чтобы продемонстрировать, как вызываются и как работают эти методы:

```
//Методы Insert, Append, AppendFormat
StringBuilder strbuild = new StringBuilder();
string str = "это  это не ";
strbuild.Append(str); strbuild.Append(true);
strbuild.Insert(4,false); strbuild.Insert(0,"2*2=5 - ");
Console.WriteLine(strbuild);
string txt = "А это пшеница, которая в темном чулане
хранится," +" в доме, который построил Джек!";
StringBuilder txtbuild = new StringBuilder();
int num =1;
foreach(string sub in txt.Split(','))
{
txtbuild.AppendFormat(" {0}: {1} ", num++,sub);
}
str = txtbuild.ToString();
Console.WriteLine(str);
```

В этом фрагменте кода конструируются две строки. Первая из них создается из строк и булевых значений true и false. Для конструирования используются методы Insert и Append. Вторая строка конструируется в цикле с применением методаAppendFormat. Результатом этого конструирования является строка, в которой простые предложения исходного текста пронумерованы.

Вопросы для самоконтроля

1 Опишите строки в C#.
2 Опишите массив символов.
3 Опишите символьный тип данных.
4 Опишите строки типа string.

Тема 2.6 Регулярные выражения

Регулярные выражения. Пространство RegularExpressions и его классы

Свойства и методы класса Regex и других классов, связанных с регулярными выражениями. Примеры создания регулярных выражений

Литература: [1], с.981-999

Методические рекомендации

Стандартный класс String позволяет выполнять над строками различные операции, в том числе поиск, замену, вставку и удаление подстрок. Существуют специальные операции, такие как Join, Split, которые облегчают разбор строки на элементы. Тем не менее, есть классы задач по обработке символьной информации, где стандартных возможностей явно не хватает. Чтобы облегчить решение подобных задач, в Net Framework встроен более мощный аппарат работы со строками, основанный на регулярных выражениях. Специальное пространство имен RegularExpression, содержит набор классов, обеспечивающих работу с регулярными выражениями. Все классы этого пространства доступны для C# и всех языков, использующих каркас Net Framework.

Синтаксис регулярных выражений

Регулярное выражение на C# задается строковой константой. Это может быть обычная или @ -константа. Чаще всего, следует использовать именно @ -константу. Дело в том, что символ " \ " широко применяется в регулярных выражениях как для записи escape-последовательностей, так и в других ситуациях. Обычные константы в таких случаях будут выдавать синтаксическую ошибку, а @ -константы не выдают ошибок и корректно интерпретируют запись регулярного выражения.

Синтаксис регулярного выражения простой формулой не описать, здесь используются набор разнообразных средств:

- символы и escape-последовательности;
- символы операций и символы, обозначающие специальные классы множеств;
- имена групп и обратные ссылки;
- символы утверждений и другие средства.

Конечно, регулярное выражение может быть совсем простым, например, строка " abc " задает образец поиска, так что при вызове соответствующего метода будут разыскиваться одно или все вхождения подстроки " abc " в искомую строку. Но могут существовать и очень сложно устроенные регулярные выражения.

Вопросы для самоконтроля

1 Опишите регулярные выражения.
2 Опишите пространство RegularExpressions и его классы.
3 Опишите свойства и методы класса Regex.

Тема 2.7 Классы в .NET. Специальные типы классов

Сущность понятия «класса» и «объекта». Стратегии доступа к членам класса

Синтаксис описания класса. Члены класса. Конструкторы. Свойства. Индексаторы. Статические поля и методы. Создание объектов. Сборка мусора и деструкторы

Литература: [3], с.100-124

Методические рекомендации

Класс — это производный структурированный тип, введенный программистом на основе уже существующих типов. Другими словами, механизм классов задает некоторую структурированную совокупность типизированных данных и позволяет определить набор операций над этими данными.

Общий синтаксис класса можно определить с помощью конструкции:

class имя_класса { список_компонентов };

- имя_класса - произвольно выбираемый идентификатор;
- список_компонентов - определения и описания типизированных данных и принадлежащих классу функций.

Компонентами класса могут быть данные, функции, классы, перечисления, битовые поля и имена типов. Вначале для простоты будем считать, что компоненты класса - это типизированные данные (базовые и производные) и функции;

- заключенный в фигурные скобки список компонентов называют телом класса;
- телу класса предшествует заголовок. В простейшем случае заголовок класса включает слово class и имя;
- определение класса всегда заканчивается точкой с запятой.

Принадлежащие классу функции мы будем называть методами класса или компонентными функциями. Данные класса - компонентными данными или элементами данных класса.

Вернемся к определению: класс - это тип, введенный программистом. А, так как, каждый тип служит для определения объектов, определим синтаксис создания объекта класса.

имя_класса имя_объекта;

Определение объекта класса предусматривает выделение участка памяти и деление этого участка на фрагменты, соответствующие отдельным элементам объекта**.**

Существует несколько уровней доступа к компонентам класса. Рассмотрим основные:

- public - члены класса открыты для доступа извне;
- private - члены класса закрыты для доступа извне.

По умолчанию все переменные и функции, принадлежащие классу, определены как закрытые (private). Это означает, что они могут использоваться только внутри функций-членов самого класса. Для других частей программы, таких как функция main(), доступ к закрытым членам запрещен. Это, кстати, единственное отличие класса от структуры - в структуре все члены по умолчанию - public.

С использованием спецификатора доступа public можно создать открытый член класса, доступный для использования всеми функциями программы (как внутри класса, так и за его пределами):

```
class имя_класса
{
    закрытые переменный и функции;
        защищенные члены данных; защищенные конструкторы; защищенные методы;
public:
    открытые переменные и функции;
```

общедоступные свойства; общедоступные члены данных; общедоступные конструкторы; общедоступный деструктор; общедоступные методы;

} список имен объектов;

Синтаксис для доступа к данным конкретного объекта заданного класса (как и в случае структур), таков:

имя_объекта.имя_члена класса;

Вопросы для самоконтроля

1 Дайте определение понятию «класс».
2 Дайте определение понятию «объект».
3 Дайте определение понятию «член класса».
4 Опишите синтаксис описания класса.

Тема 2.8 Отношения между классами

Отношения между классами: наследование, вложение. Синтаксис наследования. Конструкторы и наследование. Абстрактные классы. Виртуальные методы. Сокрытие имен. Исключение наследования

Статический и динамический полиморфизм

Литература: [1], с.108-116

Методические рекомендации

Изучая материалы данной темы, следует обратить внимание, на сведения, представленные ниже.

Наследование - механизм создания нового класса из старого. Т.е., к существующему классу можно что-либо добавить, или изменять его каким-либо образом для создания нового (порожденного) класса. Это мощный механизм для повторного использования кода. Наследование позволяет создавать иерархию связанных типов, совместно использующих код и интерфейс.

В определении и описании производного класса приводится список базовых классов, из которых он непосредственно наследует данные и методы. Между именем вводимого (нового) класса и списком базовых классов помещается двоеточие. Например, при таком

определении

class S: X, Y, Z { ... } ;

класс S порожден классами X, Y, Z, откуда он наследует компоненты. Наследование компонента не выполняется, если его имя будет использовано в качестве имени компонента в определении производного класса S. Как уже говорилось, по умолчанию из базовых классов наследуются методы и данные со спецификаторами доступа - public (общедоступные) и protected (защищенные).

В порожденном классе эти унаследованные компоненты получают статус доступа private, если новый класс определен с помощью ключевого слова class, и статус доступа public, если новый класс определен как структура, т.е. с помощью ключевого слова struct.

Таким образом, при определении класса

struct J: X, Z { ... };

любые наследуемые компоненты классов X, Z будут иметь в классе J статус общедоступных (public).

Пример:

```
class B   {
protected:
int  t;
public:
char u;
};
class E: B { ... };  // t, u наследуются как private.
struct S: B { ... }; // t, u наследуются как public.
```

Явно изменить умалчиваемый статус доступа при наследовании можно с помощью спецификаторов доступа:

- private,
- protected
- public.

Эти спецификаторы доступа указываются в описании производного класса непосредственно перед нужными именами базовых классов. Учитывая определение класса B, можно ввести следующие производные классы:

```
class M: protected B { ... }; // t, u наследуются как protected.
class P: public B { ... }; //t - protected, u - public.
class D: private B { ... }; //t, u наследуются как private.
struct F: private B {...}; //t, u наследуются как private.
struct G: public B { ... };//t - protected, u - public.
```

Вопросы для самоконтроля

1 Опишите отношения между классами.
2 Дайте определение понятию «отношение между классами»?
3 Опишите единичное наследование.
4 Дайте определение понятиям «родители и наследники»?

Тема 2.9 Структуры. Перечисления

Развернутый и ссылочный типы. Структуры

Синтаксис структур. Создание и использование структур и их элементов. Перечисления. Синтаксис перечислений. Назначение, создание и использование перечислений

Литература: [3], с.212-220

Методические рекомендации

В ООП главная роль, которую играют классы, состоит в задании типа данных. В этой лекции, как и во многих других, говоря о классах, будем иметь в виду именно эту роль. Хотя следует помнить, что некоторые классы могут играть только одну роль - роль модуля, и для них невозможно создавать объекты. Такие сервисные классы подробно рассматривались в предыдущей лекции, а примеры таких классов появлялись многократно.

Рассматрим классы, задающие тип данных. Такой класс T представляет описание множества объектов - экземпляров класса, задавая их свойства и поведение. Если класс представляет собой текст - статическое описание, то объекты класса создаются динамически в процессе работы программы. Как правило, объекты класса T создаются в других классах, являющихся клиентами класса T. Рассмотрим в клиентском классе объявление объекта класса T, выполненное с инициализацией:

T x = new T();

Объектам нужна память, чтобы с ними можно было работать. Рассмотрим две классические стратегии выделения памяти и связывания объекта, создаваемого в памяти, и сущности, объявленной в тексте. Есть два типа памяти - стек (stack) и куча (heap). Сущности x в стеке всегда отводится память, но какая память - зависит от того, к развернутому или

ссылочному типу относится класс T.

Определение 1. Если класс T относится к развернутому типу, то в стеке сущности x отводится память, необходимая объекту класса T. Говорят, что объект разворачивается на памяти, жестко связанной с сущностью x. Эту память сущность x ни с кем не разделяет.

Определение 2. Если класс T относится к ссылочному типу, то память объекту отводится в куче, а в стеке сущности x отводится дополнительная память, содержащая ссылку на объект.

Для развернутого типа характерно то, что каждая сущность ни с кем не разделяет свою память; сущность жестко связывается со своим объектом. В этом случае сущность и объект можно и не различать, они становятся неделимым понятием. Для ссылочных типов ситуация иная - несколько сущностей могут ссылаться на один и тот же объект. Такие сущности разделяют память и являются разными именами одного объекта. Полезно понимать разницу между сущностью, заданной ссылкой, и объектом, на который в текущий момент указывает ссылка. Заметим, что тип объекта в куче может не совпадать с базовым типом сущности, заданным классом T. Более того, типы объектов, с которыми динамически связывается сущность, могут быть различными, хотя и требуется определенное согласование типов. Подробнее эти вопросы будут обсуждаться при рассмотрении наследования.

Развернутые и ссылочные типы порождают две различные семантики. Рассмотрим присваивание:

y = x;

Когда сущность y принадлежит развернутому типу, значения полей объекта, связанного с сущностью y, при присваивании изменяются, получая значения полей объекта, связанного с x. Напомним, что поля y хранятся в стеке.

Когда сущность y принадлежит ссылочному типу, происходит присваивание ссылок. Ссылка y получает значение ссылки x, и обе они после присваивания указывают на один и тот же объект. Объект в куче, на который до присваивания указывала ссылка y, теряет одну из своих ссылок и, возможно, становится "висячим" объектом без ссылок, становясь добычей для сборщика мусора.

Рассмотрим операцию проверки объектов на эквивалентность:

if (x == y)

Для развернутых типов объекты x и y эквивалентны, если эквивалентны значения всех их полей. Для ссылочных типов

объекты хи у эквивалентны, если эквивалентны значения ссылок.

Язык программирования должен позволять программисту в момент определения класса указать, к развернутому или ссылочному типу относится класс. В языке C# это делается следующим образом. Если объявление класса задается с использованием служебного слова class, то такой класс относится к ссылочным типам.

public class A { ... }

Если объявление класса задается с использованием служебного слова struct, то такой класс относится к развернутым типам, которые также называют значимыми типами или value типами.

public struct B { ... }

К значимым типам относятся все встроенные арифметические типы, булевский тип, перечисления, структуры. К ссылочным типам относятся массивы, строки, классы. Так можно ли в C# спроектировать свой собственный класс так, чтобы он относился к значимым типам? Ответ на этот вопрос - положительный, хотя и с рядом оговорок. Для того чтобы класс отнести к значимым типам, его достаточно объявить как структуру.

Структура - это частный случай класса. Исторически структуры используются в языках программирования раньше классов. В языках PL/1, C, Pascal они представляли собой только совокупность данных (полей класса), но не включали ни методов, ни событий. В языке C++ возможности структур были существенно расширены, и они стали настоящими классами, хотя и с некоторыми ограничениями. В языке C# сохранен именно такой подход к структурам.

Чем следует руководствоваться, делая выбор между структурой и классом? Полагаю, можно пользоваться следующими правилами:

- если необходимо отнести класс к развернутому типу, делайте его структурой;

- если у класса число полей относительно невелико, а число возможных объектов относительно велико, делайте его структурой. В этом случае память объектам будет отводиться в стеке, не будут создаваться лишние ссылки, что позволит повысить эффективность использования памяти;

- в остальных случаях проектируйте настоящие классы.

Поскольку на структуры накладываются дополнительные ограничения, может возникнуть необходимость в компромиссе - согласиться с ограничениями и использовать структуру либо пожертвовать развернутостью и эффективностью и работать с

настоящим классом. Стоит отметить: когда говорится, что встроенные типы - int и другие - представляют собой классы, то, на самом деле, речь идет о классах, реализованных в виде структур.

Синтаксис объявления структуры аналогичен синтаксису объявления класса:

[атрибуты][модификаторы]structимя_структуры[:список_интерфейсов]

{тело_структуры}

Вопросы для самоконтроля

1 Дайте определение понятиям «развернутый и ссылочный типы».

2 Дайте определение понятиям «структуры», «перечисления».

3 Опишите синтаксис структур и перечислений.

Тема 2.10 Интерфейсы

Интерфейсы как частный случай класса и их назначение. Множественное наследование

Синтаксис и структура интерфейса. Стандартные интерфейсы среды .NET (IComparable, ICloneable и др.)

Литература: [1], с.122-128

Методические рекомендации

Изучая материалы данной темы, следует обратить внимание, на сведения представленные ниже.

Интерфейсы – это еще один инструмент реализации полиморфизма в Си-шарп. Интерфейс представляет собой набор методов (свойств, событий, индексаторов), реализацию которых должен обеспечить класс, который реализует интерфейс.

Интерфейс может содержать только сигнатуры (имя и типы параметров) своих членов. Интерфейс не может содержать конструкторы, поля, константы, статические члены. Создавать объекты интерфейса невозможно.

Объявление интерфейса

Интерфейс – это единица уровня класса, он объявляется за пределами класса, при помощи ключевого слова interface:

```
interface ISomeInterface
{
// тело интерфейса
}
```

* Имена интерфейсам принято давать, начиная с префикса «I», чтобы сразу отличать где класс, а где интерфейс.

Внутри интерфейса объявляются сигнатуры его членов, модификаторы доступа указывать не нужно:

```
interface ISomeInterface
{
string SomeProperty { get; set; } // свойство
void SomeMethod(int a); // метод
}
```

Реализация интерфейса

Чтобы указать, что класс реализует интерфейс, необходимо, так же, как и при наследовании, после имени класса и двоеточия указать имя интерфейса:

```
class SomeClass : ISomeInterface // реализация интерфейса ISomeInterface
{
// тело класса
}
```

Класс, который реализует интерфейс, должен предоставить реализацию всех членов интерфейса:

```
class SomeClass : ISomeInterface
{
public string SomeProperty
{
Get
{
// тело get аксессора
}
Set
{
// тело set аксессора
}
}
public void SomeMethod(int a)
{
```

```
// тело метода
}
}
```

Вопросы для самоконтроля

1 Опишите синтаксис и реализацию интерфейс.
2 Опишите основы работы с объектами через интерфейс.
3 Опишите стандартные интерфейсы.NET.

Тема 2.11 Обработка исключительных ситуаций

Виды ошибок в программах. Исключительные ситуации. Обработка исключительных ситуаций. Составляющие процесса обработки исключений. Создание пользовательских классов для обработки исключений

Литература: [1], с.163-171

Методические рекомендации

В .Net есть событие AppDomain CurrentDomain UnhandledException которое вызывается для всех не перехваченных исключений. Оно позволяет приложению записать в журнал информацию об исключении до того, как системный обработчик по умолчанию сообщит пользователю об исключении и прекратит выполнение приложения. Если имеется достаточно информации о состоянии приложения, можно выполнить другие действия, такие как сохранение данных программы для последующего восстановления. Рекомендуется действовать осторожно, поскольку данные программы могут быть повреждены, если исключение не обработано.

Событие Application.ThreadException позволяет приложению Windows Forms обрабатывать исключения, возникающие в потоках Windows Forms, которые в противном случае не были бы обработаны. Присоедините обработчики событий к событию ThreadException, чтобы обработать эти исключения, которые оставят приложение в неопределенном состоянии. Если это возможно, исключения следует обработать с помощью блоков структурной обработки исключений.

Чтобы добавить свой глобальный обработчик исключений, нужно подписаться на данные события:

```
AppDomain.CurrentDomain.UnhandledException += delegate( object sender, UnhandledExceptionEventArgs args )
{
Trace.WriteLine( "Global exception: " + args.ExceptionObject.ToString( ) );
};
Application.ThreadException += delegate( Object sender, ThreadExceptionEventArgs args )
{
Trace.WriteLine( "Global exception: " + args.Exception.ToString( ) );
Environment.Exit( 0 );
};
```

Вопросы для самоконтроля

1 Опишите средства для обработки исключительных ситуаций в языке C#.

2 Опишите обработку исключительных ситуаций.

3 Опишите события для исключительных ситуаций.

Тема 2.12 Делегаты, лямбда-выражения и события

Делегаты и их назначение. Функциональный тип

Синтаксис делегата. Классы Delegate и MulticastDelegate. Операции над делегатами. Комбинирование делегатов. Список вызовов.

Литература: [1], с.145-153;

Методические рекомендации

Делегаты представляют такие объекты, которые указывают на методы. То есть делегаты — это указатели на методы и с помощью делегатов мы можем вызвать данные методы.

Определение делегатов. Методы, на которые ссылаются делегаты, должны иметь те же параметры и тот же тип возвращаемого значения. Создадим два делегата:

```
delegate int Operation(int x, int y);
delegate void GetMessage();
```

Для объявления делегата используется ключевое слово delegate, после которого идет возвращаемый тип, название и параметры. Первый

делегат ссылается на функцию, которая в качестве параметров принимает два значения типа int и возвращает некоторое число. Второй делегат ссылается на метод без параметров, который ничего не возвращает.

Чтобы использовать делегат, нам надо создать его объект с помощью конструктора, в который мы передаем адрес метода, вызываемого делегатом. Чтобы вызвать метод, на который указывает делегат, надо использовать его метод Invoke. Кроме того, делегаты могут выполняться в асинхронном режиме, при этом нам не надо создавать второй поток, нам надо лишь вместо метода Invoke использовать пару методов BeginInvoke/EndInvoke:

```
{
    GetMessage del; // 2. Создаем переменную делегата
    if (DateTime.Now.Hour < 12)
    {
        del = GoodMorning; // 3. Присваиваем этой переменной адрес метода
    }
    else
    {
       del = GoodEvening;
    }
    del.Invoke(); // 4. Вызываем метод
    Console.ReadLine();
}
private static void GoodMorning()
{
    Console.WriteLine("Good Morning");
}
private static void GoodEvening()
{
    Console.WriteLine("Good Evening");
}
}
```

С помощью свойства DateTime.Now.Hour получаем текущий час. И в зависимости от времени в делегат передается адрес определенного метода. Обратите внимание, что методы эти имеют то же возвращаемое значение и тот же набор параметров (в данном случае отсутствие параметров), что и делегат.

Посмотрим на примере другого делегата:

```
class Program
{
   delegate int Operation(int x, int y);

   static void Main(string[] args)
   {
      // присваивание адреса метода через контруктор
         Operation del = new Operation(Add); // делегат указывает на метод Add
      int result = del.Invoke(4,5);
      Console.WriteLine(result);

      del = Multiply; // теперь делегат указывает на метод Multiply
      result = del.Invoke(4, 5);
      Console.WriteLine(result);

      Console.Read();
   }
   private static int Add(int x, int y)
   {
      return x+y;
   }
   private static int Multiply (int x, int y)
   {
      return x * y;
   }
}
```

Здесь описан способ присваивания делегату адреса метода через конструктор. И поскольку связанный метод, как и делегат, имеет два параметра, то при вызове делегата в метод Invoke мы передаем два параметра. Кроме того, так как метод возвращает значение типа int, то мы можем присвоить результат работы метода Invoke какой-нибудь переменной.

Метод Invoke() при вызове делегата можно опустить и использовать сокращенную форму:

```
del = Multiply; // теперь делегат указывает на метод Multiply
result = del(4, 5);
```

То есть делегат можно вызывать как обычный метод, передавая

ему аргументы.

Вопросы для самоконтроля

1 Дайте определение понятию «делегат».
2 Дайте определение понятию «функциональный тип».
3 Опишите синтаксис делегата, классы Delegate и MulticastDelegate.

Тема 2.13 Лямбда-выражения

Лямбда-выражения. Анонимные методы
Литература: [1], с.145-153;

Методические рекомендации

Лямбда-выражения представляют упрощенную запись анонимных методов. Лямбда-выражения позволяют создать емкие лаконичные методы, которые могут возвращать некоторое значение и которые можно передать в качестве параметров в другие методы.

Ламбда-выражения имеют следующий синтаксис: слева от лямбда-оператора => определяется список параметров, а справа блок выражений, использующий эти параметры: (список_параметров) => выражение. Например:

```
class Program
{
   delegate int Operation(int x, int y);
   static void Main(string[] args)
   {
      Operation operation = (x, y) => x + y;
      Console.WriteLine(operation(10, 20));      // 30
      Console.WriteLine(operation(40, 20));      // 60
      Console.Read();
   }
}
```

Вопросы для самоконтроля

1 Дайте определение понятию «лямбда-выражение».

2 Опишите синтаксис лямбда-выражения

Тема 2.14 События

События

Синтаксис события. Класс Sender и классы Receivers. Делегаты и события. Связывание обработчика с событием. Связывание обработчика с событием. Отключение обработчика. Классы с событиями, допускаемые .Net Framework

Литература: [1], с.145-153; [5], 294-308

Методические рекомендации

В C# каждое событие определяется делегатом, описывающим сигнатуру сообщения. Объявление события — это двухэтапный процесс.

Вначале объявляется делегат - функциональный класс, задающий сигнатуру. Как отмечалось при рассмотрении делегатов, объявление делегата может быть помещено в некоторый класс, например, класс sender. Но чаще всего это объявление находится вне класса в пространстве имен. Поскольку одна и та же сигнатура, может быть, у разных событий, для них достаточно иметь одного делегата. Для некоторых событий можно использовать стандартные делегаты, встроенные в каркас. Тогда достаточно знать только их имена.

Если делегат определен, то в классе sender, создающем события, достаточно объявить событие как экземпляр соответствующего делегата. Это делается точно так же, как и при объявлении функциональных экземпляров делегата. Исключением является добавление служебного слова event. Формальный синтаксис объявления таков:

[атрибуты] [модификаторы]event [тип, заданный делегатом] [имя события]

Чаще всего атрибуты не задаются, а работа происходит с модификатором доступа - public. Приведем пример объявления делегата и события, представляющего экземпляр этого делегата:

```
namespace Events
{
  public delegate void FireEvent(int day, int building);
   public class TownWithEvents
```

```
  {
    public event FireEvent fireEvent;
    …
  }//TownWithEvents
    …
}//namespace Events
```

В этом примере в классе TownWithEvents введено событие. Делегат FireEvent описывает класс событий, сигнатура которых содержит два аргумента, характеризующих событие. Объект fireEvent в классе TownWithEvents является экземпляром класса, заданного делегатом. Поскольку он снабжен модификатором event, он является событием со всеми вытекающими отсюда последствиями. Модификатор доступа public делает событие доступным для клиентов класса, обрабатывающего это событие.

Вопросы для самоконтроля

1 Какие существуют операции над делегатами?
2 Дайте определение понятию «событие».
3 Опишите синтаксис события, класс Sender и классы Receivers.

Список используемых источников

1 Албахари, Д. C# 5.0. Справочник. Полное описание языка / Д.Албахари. – М.: Вильямс, 2014.

2 Лабор, В.В. Си Шарп: Создание приложений для Windows / В.В.Лабор. - Мн.: Харвест, 2003.

3 Павловская, Т.А C#. Программирование на языке высокого уровня / Т.А.Павловская. - СПб: Питер, 2014.

4 Фролов, А.В. Визуальное проектирование приложений C# / А.В.Фролов. - М: КУДИЦ - ОБРАЗ, 2003.

5 Фленов, М. Библия C# / М.Фленов. - СПб.: Питер, 2011.

Задания на домашнюю контрольную работу по учебному предмету «Конструирование программ и языки программирования»

1-60 Теоретические вопросы

1 Дайте определение понятию объектно-ориентированного программирования.

2 Перечислите принципы объектно-ориентированного программирования: полиморфизм, наследование, инкапсуляция.

3 Дайте определение понятию среды разработки.

4 Дайте определение понятию состав среды разработки.

5 Дайте определение понятию платформы программирования.

6 Опишите схему выполнения программы в .NET.

7 Опишите среду разработки Microsoft Visual Studio .NET.

8 Опишите среду компиляции программы среду разработки Microsoft Visual Studio .NET.

9 Опишите среду отладки программы среду разработки Microsoft Visual Studio .NET.

10 Опишите среду запуска программы среду разработки Microsoft Visual Studio .NET.

11 Опишите классификацию типов данных.

12 Перечислите этапы создания проекта консольной программы.

13 Опишите различия между типом – значением и ссылочным типом.

14 Опишите назначение упаковки и распаковки величин различного типа.

15 Опишите понятие метода класса и его синтаксис.

16 Опишите правила описания и инициализации переменных в языке C#.

17 Опишите правила описания именованных констант.

18 Опишите правила построения выражений в языке C#.

19 Опишите операции инкремента, декремента.

20 Опишите операцию new.

21 Опишите арифметические операции.

22 Опишите операции отношения и проверки на равенство.

23 Перечислите приоритет операций.

24 Опишите операции присваивания.

25 Опишите синтаксис операций ввода/вывода информации на

консоль.

26 Опишите условный оператор if.

27 Опишите логические операции.

28 Опишите оператор выбора switch.

29 Опишите операторы цикла for.

30 Опишите операторы цикла while, do/while.

31 Дайте определение понятию метода класса.

32 Перечислите параметры метода класса.

33 Дайте определение понятию массив.

34 Опишите синтаксис и обработку одномерных массивов.

35 Опишите синтаксис и обработку многомерных массивов.

36 Опишите синтаксис оператора foreach.

37 Опишите назначение методов класса System.Array.

38 Дайте определение понятию регулярного выражения.

39 Опишите символьный тип данных.

40 Опишите синтаксис объявления массива символов.

41 Опишите синтаксис объявления строк.

42 Дайте определение понятиям развернутого и ссылочного типов.

43 Опишите особенности наследования классов в языке C#.

44 Опишите отношения между классами.

45 Опишите операции is и as.

46 Дайте определение понятию делегаты.

47 Опишите назначение делегатов.

48 Опишите описание и использование делегатов.

49 Опишите механизм событий.

50 Опишите механизм обработки исключительных ситуаций.

51 Опишите средства для обработки исключительных ситуаций в языке C#.

52 Опишите отношение клиенты – поставщики.

53 Опишите отношения наследования.

54 Опишите единичное наследование.

55 Опишите отношения классов «Родитель - наследник».

56 Дайте определение понятию события.

57 Дайте определение понятию анонимного метода.

58 Перечислите особенности перечислений.

59 Перечислите математические функции языка C#.

60 Опишите стратегии доступа к полям класса.

61-75 Практическое задание 1

Решение задач при помощи циклов While и Do While

В отчете необходимо представить текст программы с комментариями, блок-схему алгоритма решения составленную в соответствии с ГОСТ 1.9.003-80 и 19.701-90 (ИСО 5807-85).

61 Напишите программу, вычисляющую сумм и среднее арифметическое последовательности положительных чисел, которые вводятся с клавиатуры. Ниже приведен рекомендуемый вид экрана во время выполнения программы (данные, введенные пользователем, выделены полужирным шрифтом).

Вычисление среднего арифметического последовательности положительных чисел.

Вводите после стрелки числа. Для завершения ввода введите ноль.

- > 45
- > 23
- > 75
- >0

Введено чисел: 3

Сумма чисел: 83

Среднее арифметическое: 27.67

62 Напишите программу, которая определяет максимальное число из введенной с клавиатуры последовательности положительных чисел (длина последовательности неограниченна). Ниже приведен рекомендуемый вид экрана во время выполнения программы (данные, введенные пользователем, выделены полужирным шрифтом).

Определение максимального числа последовательности положительных чисел.

Вводите после стрелки числа. Для завершения ввода введите ноль,

- > 56
- > 75
- >43
- >0

Максимальное число: 75

63 Напишите программу, которая определяет минимальное число во введенной с клавиатуры последовательности положительных чисел (длина последовательности неограниченна). Ниже приведен рекомендуемый вид экрана во время выполнения программы (данные, введенные пользователем, выделены полужирным шрифтом).

Определение минимального числа в последовательности положительных чисел. Вводите после стрелки числа. Для завершения ввода введите ноль.

- > 12
- > 75
- > 10
- > 9
- >23
- >0

Минимальное число: 9

64 Напишите программу, которая определяет число положительных чисел во введенной с клавиатуры последовательности (длина последовательности неограниченна). Ниже приведен рекомендуемый вид экрана во время выполнения программы (данные, введенные пользователем, выделены полужирным шрифтом).

Определение числа положительных членов в последовательности чисел

Вводите после стрелки числа. Для завершения ввода введите ноль.

- > 28
- > -72
- > 10
- > 15
- >-23
- >0

Число положительных: 3

65 Напишите программу, которая определяет число отрицательных чисел во введенной с клавиатуры последовательности (длина последовательности неограниченна). Ниже приведен рекомендуемый вид экрана во время выполнения программы (данные, введенные пользователем, выделены полужирным шрифтом).

Определение числа отрицательных членов в последовательности чисел

Вводите после стрелки числа. Для завершения ввода введите ноль.

- >28
- > - 72
- >10
- >15
- > - 23
- > 0

Число отрицательных: 2

66 Напишите программу, которая "задумывает" число в диапазоне от 1 до 10 и предлагает пользователю угадать число за 5 попыток. Ниже приведен рекомендуемый вид экрана во время выполнения программы (данные, введенные пользователем, выделены полужирным шрифтом).

Игра 'Угадай число".

Компьютер "задумал" число от 1 до 10.

Угадайте его за 5 попыток.

Введите число и нажмите <Enter>

- >5

Нет

- > 3

Вы выиграли! Поздравляю!

67 Вычислите приближенно значение бесконечной суммы с точностью до E:

$$S = \frac{1}{2 * 3} + \frac{1}{3 * 4} + ...$$

Точность достигается тогда, когда очередное полученное значение суммы S будет отличаться от предыдущего менее чем на e. Ниже приведен рекомендуемый вид экрана во время выполнения программы (данные, введенные пользователем, выделены полужирным шрифтом).

Определение суммы бесконечной последовательности с заданной точностью.

Задайте точность вычисления суммы-> 0.001 Значение суммы с точностью 0.001 равно 0,46969697 Просуммировано 32 члена ряда.

68 Вычислите приближенно значение бесконечной суммы с точностью до Е:

$$S = \frac{1}{2*4} + \frac{1}{4*6} + \frac{1}{6*8} \ldots$$

Точность достигается тогда, когда очередное полученное значение суммы S б5[г]дет отличаться от предыдущего менее чем на Е. Ниже приведен рекомендуемый вид экрана, во время выполнения программы (данные, введенные пользователем, выделены полужирным шрифтом).

Определение суммы бесконечной последовательности с заданной точностью.

Задайте точность вычисления суммы-> 0.0001

Значение суммы с точностью 0.0001 равно 0,245

Просуммировано 5 J члена ряда.

69 Вычислите приближенно значение бесконечной суммы с точность» до е:

$$S = \frac{1}{1*3} + \frac{1}{3*5} + \frac{1}{5*7} \ldots$$

Точность достигается тогда, когда очередное полученное значение суммы S будет отличаться от предыдущего менее чем на е. Ниже приведен рекомендуемый вид экрана во время выполнения программы (данные, введенные пользователем, выделены полужирным шрифтом).

Определение суммы бесконечной последовательности с заданной точностью.

Задайте точность вычисления суммы-> 0.0001

Значение суммы с точностью 0.0001 равно 0,295

Просуммировано 51 члена ряда.

70 Напишите программу, которая выводит на экран таблицу значений функции $y = 2x^2-5x-8$ в диапазоне от -3 до 3. Шаг изменения аргумента 0,5.

71 Напишите программу, которая выводит на экран таблицу

значений функции $y = \sqrt{x^2 + 1}$ в диапазоне от -2 до 2. Шаг изменения аргумента ОД.

72 Напишите программу, которая выводит на экран таблицу значений функции $y = x^3-2x^2+8$ в диапазоне от -4 до 4. Шаг изменения аргумента 0,5.

73 Напишите программу, которая выводит на экран таблицу значений функции $y = \sqrt{|2 * x - 5|}$ в диапазоне от -2 до 2. Шаг изменения аргумента 0,1.

74 Напишите программу, которая выводит на экран таблицу значений $y = \frac{1}{x^2 + 1}$ в диапазоне от-3 до 3. Шаг изменения аргумента 0,5.

75 Напишите программу, которая выводит на экран таблицу значений $y = \frac{x}{x^2 + 2}$ в диапазоне от -4 до 4. Шаг изменения аргумента 0,5.

76- 90 Практическое задание 2

Решение задач обработки массивов

В отчете необходимо представить текст программы с комментариями, блок-схему алгоритма решения, составленную в соответствии с ГОСТ 29.003-80 и 19.701-90 (ИСО 5807-85).

76 Решите задачу. Дан двумерный массив. Заполните его по строкам с клавиатуры и определите:

- количество строк, не содержащих ни одного нулевого элемента;
- максимальное из чисел, в заданной строке массива.

77 Решите задачу. Дан двумерный массив. Заполните его по строкам с клавиатуры и определите:

- количество и сумму положительных элементов;
- минимальное из чисел, в заданной строке массива.

78 Решите задачу. Дан двумерный массив. Заполните его по строкам с клавиатуры и определите:

- количество столбцов, не содержащих ни одного нулевого элемента;
- минимальное из чисел, в заданном столбце массива.

79 Решите задачу. Дан двумерный массив. Заполните его по строкам с клавиатуры и определите:

- количество и сумму отрицательных элементов;
- максимальное из чисел, в заданном столбце массива

80 Решите задачу. Дан двумерный массив. Заполните его по строкам с клавиатуры и определите:

- количество строк, содержащих хотя бы один нулевой элемент;
- максимальное по модулю число, в заданной строке массива

81 Решите задачу. Дан двумерный массив. Заполните его по строкам с клавиатуры. Определите номер строки, сумма элементов которого максимальна

82 Решите задачу. Дан двумерный массив. Заполните его по строкам с клавиатуры. Отсортируйте каждый столбец по убыванию.

83 Решите задачу. Дан двумерный массив. Заполните его по строкам с клавиатуры. Отсортируйте каждую строку по возрастанию.

84 Решите задачу. Дан двумерный массив. Заполните его по строкам с клавиатуры. Отсортируйте каждый столбец по возрастанию.

85 Решите задачу. Дан двумерный массив n x n. Заполните его по строкам с клавиатуры и определите:

- минимальный элемент в главной диагонали;
- произведение элементов в столбцах, не содержащих нулей.

86 Решите задачу. Дан двумерный массив n x n. Заполните его но строкам с клавиатуры и определите:

- минимальный элемент в побочной диагонали;
- произведение элементов в строках, не содержащих нулей.

87 Решите задачу. Дан двумерный массив n x n. Заполните

его по строкам с клавиатуры. Определите минимальный элемент в побочной диагонали и поменяйте местами столбец содержащего его с последним столбцом.

88 Решите задачу. Дан двумерный массив n x n. Заполните его по строкам с клавиатуры. Определите минимальный элемент в главной диагонали и поменяйте местами столбец содержащего его с первым столбцом.

89 Решите задачу. Дан двумерный массив n x n. Заполните его по строкам с клавиатуры. Определите максимальный элемент в побочной диагонали и поменяйте местами столбец содержащего его с первым столбцом.

90 Решите задачу. Дан двумерный массив n x n. Заполните его по строкам с клавиатуры. Определите максимальный элемент в побочной диагонали и поменяйте местами столбец содержащего его с первым столбцом.

Методические рекомендации по выполнению задач домашней контрольной работы № 1

Решение задач при помощи циклов While и Do While (задачи 61-75)

Перед написанием программы откроем интегрированную среду VisualC#:

Пуск/Программы/Microsoft Visual Studio 2012/ Microsoft Visual 2012

Далее создадим проект. Для этого:

1) New Project..
2) В открывшемся окне:

- выберитеVisual C# -> Console Application;
- введите имя проекта в текстовом поле Name;
- введите (выберете с помощью кнопки …) имя каталога размещения файлов проекта в текстовом поле Location, например G:/ASOIZ/;
- щелкните левой кнопкой мыши на кнопке ОК.

Пишем программный код.

Далее компилируем программу. Для этого нажимаем кнопку на панели инструментовBUILD->BUILDSOLUCHION либо комбинацию клавиш F6. В окне вывода (внизу экрана) должно вывестись сообщение 0 error(s), 0 warning(s) (0 ошибок, 0 предупреждений). Если есть ошибки - сверьте с оригиналом.

Для запуска программы нажимаем кнопку Starts на панели инструментов либо комбинацию клавиш F5.

Напишите программу, вычисляющую сумму и среднее арифметическое последовательности положительных чисел, которые вводятся с клавиатуры. Ниже приведен рекомендуемый вид экрана во время выполнения программы (данные» введенные пользователем, выделены полужирным шрифтом).

Вычисление среднего арифметического последовательности положительных чисел.

Вводите после стрелки числа. Для завершения ввода введите ноль.

- >45
- > 23
- >15

```
- >0
Введено чисел: 3
Сумма чисел: 83
Среднее арифметическое: 27.67
Вычисление среднего арифметического
// последовательности положительных чисел
using System;
usingSystem.Collections.Generic;
usingSystem.Linq;
usingSystem.Text;
namespace
{
class Program
{
static void Main(string[] args)
{
inta;   // число, введенноесклавиатуры
intp;   // количество чисел
ints;   //сумма чисел
floatt; //среднее арифметическое
s = 0;
p = 0;
Console.WriteLine("Вычисление среднего арифметического");
Console.WriteLine("Последовательности положительных чисел");
Console.WriteLine("Вводите числа. Для завершения введите 0");
do  {
Console.Write("->");
a= Convert.ToInt32(Console.ReadLine());
if (a>0)
{
s+=a;
p++;
}
}
while (a>0);
Console.WriteLine("Введеночисел: "+ p);
Console.WriteLine("Суммачисел: "+ s);
t=(float) s/p;
```

```
Console.WriteLine("Среднеарифметическое: {0:F2}", t);
Console.WriteLine("Для завершения нажмите <Enter>: ");
Console.ReadLine();
}
}
}
```

Решение задач обработки массивов (задачи 76 -90)

Дан двумерный массив. Заполните его по строкам с клавиатуры. Определите номер строки, сумма элементов которого максимальна.

```
// Строка с максимальной суммой элементов
using System;
usingSystem.Collections.Generic;
usingSystem.Linq;
usingSystem.Text;
namespace Zd_2_mir
{
class Program
{
static void Main(string[] args)
{
constint n=3; //размер квадратной матрицы
int[,] A = newint[n, n];//объявление массива
intsummax,sumst=0,max = 0,length=0;// строка с максимальной суммой элементов
inti,j;    //индексы
Console.WriteLine("Размер матрицы: ");
for (int i = 0; i < n; i++)
{
summax = 0;
for (int j = 0; j < n; j++)
{
Console.WriteLine("Заполните элемент матрицы A[{0},{1}]", i, j);
A[i, j] = Int32.Parse(Console.ReadLine());
summax+=A[i,j];
}
for (int j = n; j < n; j++)
```

```
sumst += A[i,j];

if (max <= summax)
{
max = summax;
length = i;
}
}
Console.WriteLine("Строка с макс. суммой эл-ов {0}", length+1);
Console.ReadLine();
}
}
}
```

Таблица 3 – Варианты заданий на домашнюю контрольную работу № 1 по учебному предмету «Конструирование программ и языки программирования»

Предпоследняя цифра шифра	Последняя цифра шифра									
	0	1	2	3	4	5	6	7	8	9
0	1, 60 67, 77	13, 57 74, 81	14, 38 61, 89	16, 58 63, 90	15, 60 70, 78	21, 55 71, 90	9, 54 74, 82	9, 43 64, 90	29, 32 68, 90	22, 56 70, 89
1	17, 52 61, 84	3, 42 73, 82	11, 35 75, 86	15, 39 67, 79	25, 51 67, 78	11, 33 61, 78	10, 44 62, 89	22, 35 69, 88	28, 48 64, 84	23, 57 73, 90
2	20, 48 73, 86	27, 49 62, 84	25, 54 65, 89	6, 50 74, 82	23, 38 66, 81	6, 58, 68, 88	18, 52 63, 87	16, 53 73, 76	12, 59 71, 82	12, 52 65, 76
3	16, 40 62, 77	19, 41 61, 85	30, 40 68, 79	20, 33 70, 85	2, 43 66, 76	3, 42 68, 78	25, 41 75, 81	13, 58 62, 88	29, 33 66, 80	13, 36 69, 83
4	21, 39 72, 82	5, 55 75, 87	22, 56 65, 87	8, 46 72, 77	5, 45 69, 76	16, 40 62, 88	14, 52 67, 78	14, 36 72, 82	4, 51 63, 83	4, 45 61, 85
5	30, 52 71, 83	24, 57 69, 86	28, 43 65, 83	28, 41 75, 80	3, 48 70, 84	20, 58 74, 77	6, 37 64, 77	26, 37 70, 81	21, 34 67, 83	30, 38 61, 84
6	23, 35 68, 87	5, 37 66, 86	18, 44 61, 77	21, 47 70, 87	7, 49 71, 79	8, 33 60, 78	24, 35 64, 84	21, 57 69, 89	4, 57 68, 87	24, 37 71, 89
7	15, 38 68, 90	7, 43 71, 79	17, 41 64, 88	27, 53 73, 85	10, 41 66, 77	16, 31 73, 76	7, 46 72, 88	19, 39 68, 77	9, 45 67, 80	27, 35 71, 80
8	26, 46 61, 80	8, 55 63, 79	8, 59 72, 80	1, 46 69, 88	1, 44, 75, 80	14, 57 75, 89	19, 39 66, 85	7, 42, 68, 83	22, 31 66, 81	26, 49 69, 86
9	17, 52 70, 81	29, 34 72, 87	14, 50 63, 89	25, 60 61, 83	18, 35 63, 79	7, 47 72, 85	2, 50 72, 81	2, 47 68, 78	3, 57 63, 85	8, 32 75, 82