МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Пермский государственный национальный исследовательский университет»

Колледж профессионального образования

ПРИКЛАДНОЕ ПРОГРАММИРОВАНИЕ

Методические рекомендации

для лабораторных работ по изучению дисциплины

для студентов Колледжа профессионального образования

специальности

09.02.03 Программирование в компьютерных системах

Утверждено на заседании ПЦК

Информационных технологий

Протокол № 9 от 23.05.2018

председатель _____Н.А. Серебрякова

Пермь 2018

Составитель:

Бочкарев Алексей Михайлович, преподаватель первой квалификационной категории, преподаватель ПГНИУ

ПРИКЛАДНОЕ ПРОГРАММИРОВАНИЕ: методические указания по лабораторной работе для студентов Колледжа профессионального образования по специальностям 09.02.03 Программирование в компьютерных системах/ сост. А.М. Бочкарев; Колледж проф. образ. ПГНИУ. – Пермь, 2018. – 16 с.

Методические указания «ПРИКЛАДНОЕ ПРОГРАММИРОВАНИЕ» разработаны на основе требований Федерального государственного образовательного стандарта среднего профессионального образования по специальностям 09.02.03 Программирование в компьютерных системах и 09.02.04 Информационные системы (по отраслям) для оказания помощи студентам специальностей 09.02.03 Программирование в компьютерных системах по дисциплине «ПРИКЛАДНОЕ ПРОГРАММИРОВАНИЕ». Содержат типичные лабораторные задания по всем разделам дисциплины.

Предназначены для студентов Колледжа профессионального образования ПГНИУ специальностей 09.02.03 Программирование в компьютерных системах (СПО) всех форм обучения.

Печатается по решению педагогического совета Колледжа профессионального образования Пермского государственного национального исследовательского университета

СОДЕРЖАНИЕ

Технологии разработки прикладного программирования

При изучении любого языка, первая программа, которую обычно пишут – «Hello, World». Рассмотрим, как она выглядит в C#. using System; class Hello { public static void Main() { Console.WriteLine(“Hello, World”); } } При выполнении приведенного кода на экране появится надпись «Hello, World». В C# приложение – это коллекция одного или нескольких классов, структур и других типов. Класс определяется как набор данных и методов, работающих с ними. При рассмотрении кода приложения «Hello, World», видим, что есть единственный класс названный Hello. После имени класса открываем фигурные скобки ({). Всё, что находится до соответствующей закрывающейся скобки (}), является частью класса. Можно распределить один класс приложения C# на несколько файлов. Также, с другой стороны, можно поместить несколько классов в один файл. Каждое приложение должно начинаться с какого-то оператора. В C# при запуске приложения начинает выполняться метод Main.

Несмотря на то, что в приложении может быть много классов, точка входа всегда одна. Может быть несколько методов Main, но запускаться будет только один, он выбирается при компиляции. Синтаксис Main важен, если проект создан в Visual Studio, то он 12 сгенерируется автоматически. При завершении метода Main приложение заканчивает работу. Как часть MS.NET Framework C# содержит много различных классов, которые упорядочены в пространствах имен. Пространство имен (namespace) – это набор связанных классов. Пространство имен также может содержать вложенные пространства имен. Основное пространство имен – это System. К объектам пространств имен можно обращаться при помощи полного имени, используя префиксы.

Например, пространство System, содержит класс Console, который выполняет несколько методов, включая WriteLine System.Console.WriteLine(“Hello, World”); Директива using позволяет обращаться к членам пространства имен напрямую без использования полного имени. using System; Console.WriteLine(“Hello, World”); Основные операции ввода/вывода В этом разделе рассмотрим класс Console и его методы ввода и вывода. Класс Console обеспечивает приложению C# доступ к стандартным потокам ввода, вывода и ошибок. Стандартный поток ввода – клавиатура. Стандартный поток вывода и ошибок – экран. Эти потоки могут быть перенаправлены из/в файлы. Write и WriteLine методы вывода информации на консоль. Они перегружаемы, то есть могут выводить как строки, так и числа. Console.WriteLine(99); Console.WriteLine("Hello, World"); Console.WriteLine("The sum of {0} and {1} is {2}", 100, 130, 100+130); Можно использовать левое и правое выравнивание, задавать ширину вывода. Console.WriteLine("\"Левое выравн. в поле ширины 10: {0, -10}\"", 99); Console.WriteLine("\"Правое выравн. в поле ширины 10: {0,10}\"", 99); Будет выведено “Левое выравн. в поле ширины 10: 99 ” “Правое выравн. в поле ширины 10: 99” 13 Есть настройки для вывода числовых форматов: Console.WriteLine("Валюта - {0:C} {1:C4}", 88.8, -888.8); Console.WriteLine("Целое число - {0:D5}", 88); Console.WriteLine("Экспонента - {0:E}", 888.8); Console.WriteLine("Фиксированная точка - {0:F3}", 888.8888); Console.WriteLine("Общий формат- {0:G}", 888.8888); Console.WriteLine("Числовой формат - {0:N}", 8888888.8); Console.WriteLine("Шестнадцатиричный - {0:X4}", 88); Соответственно отобразят: Валюта - $88.80 ($888.8000) Целое число - 00088 Экспонента - 8.888000E+002 Фиксированная точка - 888.889 Общий формат - 888.8888 Числовой формат - 8,888,888.80 Шестнадцатеричный формат – 0058 Рекомендации по оформлению кода Основная рекомендация – это комментирование кода. Комментирование программы позволяет разработчикам, не участвовавшим при написании, разобраться, как работает приложение. Нужно использовать полные и значимые имена для именования компонент. Хорошие комментарии объясняют не что написано в программе, а отвечают на вопрос, почему именно так. Если в вашей организации есть стандарты комментирования, придерживайтесь их. C# поддерживает несколько вариантов комментирования кода: однострочные(//), многострочные комментарии(/* */) и XMLдокументация(///). В XML-документации можно использовать различные предопределенные теги. Для генерации XML файла с комментариями используется дополнительный параметр компиляции. Для компиляции кода из командной строки: csc myprogram.cs /doc:mycomments.xml

Надежная программа на C# обязана обрабатывать непредвиденные ситуации. Независимо от того, сколько различных ошибок предусмотрено, 14 всегда существует вероятность того, что что-то может пойти не так. Когда происходит ошибка при выполнении приложения, операционная система генерирует исключение. Конструкцией try-catch можно перехватывать эти исключения. Если при выполнении программы в блоке try произошло исключение, управление передаётся блоку catch. Если в программе не обрабатывается исключение, то оно вызовет окно операционной системы с предложением отладить программу при помощи Just-in-Time Debugging. Перед запуском приложения C# необходимо его откомпилировать. Компилятор преобразует исходный код в машинные коды. Компилятор C# можно вызывать как из командной строки, так и из Visual Studio. Пример вызова компилятора из командной строки: csc Hello.cs /debug+ /out:Greet.exe (Заметим, что csc – это файл, и путь к нему нужно прописывать полностью.) Если компилятор находит ошибки, он сообщает строку ошибки и номер символа. Если ошибок нет, и приложение откомпилировалось, то его можно запускать как при помощи Visual Studio, так и в командной строке по имени файла с расширением exe.

Вопросы к разделу 1. Откуда начинается выполнение приложения C#? 2. Когда приложение заканчивает работу? 3. Сколько классов может содержать приложение C#? 4. Сколько методов Main может содержать приложение? 5. Как прочитать данные, введенные пользователем с клавиатуры? 6. В каком пространстве имен находится класс Console? 7. Что произойдёт при необработанном в приложении исключении?

Лабораторная работа

Задания на создание C# программ, компилирование и отладку, использование отладчика Visual Studio, обработку исключительных ситуаций. Время, необходимое на выполнение задания 60 мин.

Упражнение 2.1 Написать программу, которая спрашивает имя пользователя, и затем приветствует пользователя по имени. (Создать консольное приложение.) 15 · Откомпилировать и запустить программу с помощью Visual Studio. · Откомпилировать и запустить программу с командной строки. · В среде Visual Studio использовать пошаговую отладку. Посмотреть, как изменяется текущее значение переменной.

Упражнение 2.2 Написать программу, которой на вход подается два целых числа, на выходе – результат деления одного числа на другое. Предусмотреть обработку исключительной ситуации, возникающей при делении числа на ноль.

Основные возможности языка C# и их применение

Все приложения работают с теми или иными данными. Разработчику C# необходимо понимать, как хранятся и обрабатываются данные в приложении. Обычно данные хранятся в переменных, которые необходимо объявить до их использования. При объявлении переменной под неё резервируется некоторое количество памяти в зависимости от типа переменной и объявляется имя. После объявления переменной можно присваивать ей значения. В этой главе рассмотрим как использовать структурные переменные в C#. Как именовать переменные в соответствии со стандартами, а также как присваивать значения и переводить существующие переменные из одного типа в другой. Общая система типов (Common Type System) Каждая переменная имеет тип данных, который определяет какие значения хранятся в ней. C# – язык безопасных типов, т.е. компилятор гарантирует, что значение хранящееся в переменной будет всегда соответствующего типа. CTS (Common Type System) – интегрированная часть общеязыковой среды выполнения. Эта модель определяет правила, которыми руководствуется среда выполнения при объявлении, использовании и управления типами. CTS обеспечивает структуру, необходимую для многоязыковой интеграции, безопасности типов и высокой эффективности выполнения кода. Рассмотрим два типа переменных: структурные и ссылочные. Структурные типы данных напрямую содержат данные. Каждая переменная содержит свою копию данных, т.е. при операции над одной переменной невозможно изменить другую. Структурные типы данных включают в себя встроенные и пользовательские типы. Разница между ними в C# минимальна, так как они используются одина-

ково. Все структурные типы напрямую содержат данные и не могут быть null. Ссылочные типы данных содержат ссылки на данные. Две переменные могут указывать на один и тот же объект, т.е. при изменении одной ссылочной переменной, можно изменить другую. 17 Все базовые типы данных содержатся в пространстве имен System. Все типы наследуются от System.Object. Все структурные типы наследуются от System.ValueType. Встроенные типы объявляются при помощи зарезервированных слов, кроме того, можно объявить при помощи типа структуры из пространства имен System: sbyte System.SByte byte System.Byte short System.Int16 ushort System.UInt16 int System.Int32 uint System.UInt32 long System.Int64 ulong System.UInt64 char System.Char float System.Single double System.Double bool System.Boolean decimal System.Decimal Правила именования переменных При именовании переменных необходимо соблюдать следующие правила (если не соблюдать, то получим ошибку при компиляции): · можно использовать только буквы, нижнее подчеркивание и цифры, · имя переменной начинается только с буквы или подчеркивания, · после первого символа можно использовать и цифры, · нельзя использовать зарезервированные слова. Рекомендации по именованию: · избегать имена только из заглавных букв, · избегать начинать имя с подчеркивания, · избегать аббревиатур, · именование PascalCasing – каждое слово начинается с большой буквы, используется для классов, методов, свойств, перечислений, интерфейсов, констант, пространств имен, · именование camelCasing – каждое слово, кроме первого, начинается с большой буквы, используется для переменных, полей и параметров. 18 Использование встроенных типов данных Для использования переменной необходимо выбрать ей имя, назначить тип и присвоить начальное значение. Переменные, которые объявлены в методах, называются локальными. В C# нельзя использовать неинициализированные переменные, в этом случае выдаётся ошибка при компиляции. Переменной можно присвоить значение соответствующего типа. Добавление значения переменной осуществляется очень просто: int itemCount; itemCount = 2; itemCount = itemCount + 40; Приведем сокращенные способы записи арифметический действий: var += expression; // var = var + expression var -= expression; // var = var - expression var *= expression; // var = var * expression var /= expression; // var = var / expression var %= expression; // var = var % expression Выражения (expression) состоят из операций и операндов. Существуют следующие общие операции: Операции присваивания – присваивают значение правого операнда левому: = *= /= %= += -= <<= >>= &= ^= |= . Операции сравнения – сравнивают два значения: == != . Логические – производит побитовые операции над операндами: << >> & ^ | . Условные – выполняют одно выражение из двух в зависимости от булевого условия: && || ?: . Инкремент и декремент – увеличивает/уменьшает значение на единицу: ++ -- . Арифметические – выполняют стандартные арифметические операции: + - * / % . Операции инкремент и декремент имеют два варианта префиксный (++var) и постфиксный (var++). Соответственно, сначала выполняется увеличение значения переменной var, а затем выполняется выражение и 19 наоборот, сначала выражение, и только после него изменяется значение операнда. Пользовательские типы данных В приложениях есть возможность создавать свои типы данных: структуры и перечисления. Перечисления полезны, когда переменные могут принимать значения только из определенного набора. Каждому значению в перечислении соответствует свой номер. По умолчанию нумерация элементов начинается с нуля. enum Color { Red, Green, Blue } Цвет Red будет иметь значение 0, Green – 1, а Blue будет 2. Пример использования переменной перечислимого типа: Color colorPalette; // объявление переменной colorPalette = Color.Red; // установка значения или colorPalette = (Color)0; //явное преобразование из int Структуры можно использовать для создания объектов, ведущих себя как встроенные типы. Они группируют данные различного типа. public struct Employee { public string firstName; public int age; } Для доступа к элементам структур используется следующая конструкция: Employee companyEmployee; // объявление переменной companyEmployee.firstName = “Joe”; // присваивание companyEmployee.age = 23; // значений Преобразование типов Различают два способа преобразования типов: явное и неявное. При неявном преобразовании возможно потеря точности. Для явного преобразования используется операция приведения типов. 20 Пример неявного преобразования: using System; class Test; { static void Main() { int intValue = 123; long longValue = intValue; Console.WriteLine(“(long) {0} = {1}”, intValue, longValue) } } Пример явного преобразова-

ния: using System; class Test; { static void Main() { long longValue = Int64.MaxValue; int intValue = (int) longValue; Console.WriteLine(“(int) {0} = {1}”, longValue, intValue) } } Вторая программа на экране напечатает (int) 9223372036854775807 = -1, т.к. произойдет переполнение переменной типа int. Есть возможность отслеживать такие ошибки, путем размещения данного кода внутри блока checked. Checked используется для явной проверки переполнения при выполнении арифметических операций и преобразований с данными целого типа. Пример: using System; class Test { static void Main() { long longValue = Int64.MaxValue; checked{ try{ 21 int intValue = (int) longValue; Console.WriteLine("(int) {0} = {1}", longValue, intValue); } catch (Exception e) { Console.WriteLine("Ошибка приведения типов"); } } } } Более подробно о блоке try-catch написано в четвертой главе.

Вопросы к разделу 1. Что такое общеязыковая система типов? 2. Может ли структурная переменная иметь значение null? 3. Можно ли не инициализировать переменные в C#? Почему? 4. Можно ли потерять данные в результате неявного преобразования?

Лабораторная работа Задания на создание пользовательских типов данных, объявление и использование переменных. Время, необходимое на выполнение задания 35 мин.

Упражнение 3.1 Создать перечислимый тип данных отображающий виды банковского счета (текущий и сберегательный). Создать переменную типа перечисления, присвоить ей значение и вывести это значение на печать.

Упражнение 3.2 Создать структуру данных, которая хранит информацию о банковском счете – его номер, тип и баланс. Создать переменную такого типа, заполнить структуру значениями и напечатать результат.

Интегрированная среда разработки Visual Studio. Проекты и решения

C# – объектно-ориентированный язык. В этой главе изучим терминологию и концепцию ООП. Классы и объекты Ключевым словом в ООП является класс. Все языки программирования могут обращаться с общими данными и методами. Эта возможность помогает избежать дублирования. Главная концепция программирования – не писать один и тот же код дважды. Программы без дублирования лучше и понятнее, так как содержат меньше кода. ООП переводит эту концепцию на новый уровень, он позволяет описывать классы (множества объектов), которые делают общими и структуру, и поведение. Классы не ограничиваются описанием конкретных объектов, они также могут описывать и абстрактные вещи. Объект – это конкретный представитель класса. Его определяет три характеристики: уникальность, поведение и состояние. Уникальность – это характеристика, определяющая отличие одного объекта от другого. Поведение определяет то, чем объект может быть полезен, что он может делать. Поведение объекта классифицирует его. Объекты разных классов отличаются своим поведением. Состояние описывает внутреннюю работу объекта то, что обеспечивает его поведение. Хорошо спроектированный объект оставляет своё состояние недоступным. Нас не интересует, как он это делает, нам важно то, что он умеет это делать. Сравним структуры и классы. В C# структуры могут иметь методы, но желательно избегать этого. Однако в некоторых структурах необходимы операторы. Операторы – стилизованные методы, но они не добавляют поведение, а обеспечивают более краткий синтаксис. Структурные типы – нижний уровень программы, это элементы, из которых строятся более сложные элементы. Переменные структурных типов свободно копируются и используются как поля и атрибуты объектов. Ссылочные типы – верхний уровень программы, они состоят из мелких элементов. Ссылочные типы в основном не могут быть скопированы. 49 Абстракция – это тактика очистки объекта от всех несущественных деталей, оставляя только существенную минимальную форму. Абстракция – важный принцип программирования. Хорошо спроектированный класс содержит минимальный набор методов, полностью описывающий его поведение. Инкапсуляция данных Традиционное процедурное программирование содержит много данных и много процедур. Любая функция имеет доступ к любым данным. Когда программы становятся большими, это создаёт много проблем – при небольших изменениях в коде необходимо следить за всей программой. Другая проблема – это хранение данных отдельно от функций. В ООП эта проблема решается за счет объединения данных и методов, которые работают с этими данными как одно целое. Данные и функции объединены в одну сущность, эта сущность ограничивает капсулу. Получаем две области: снаружи этой капсулы и внутри неё. Элементы, которые доступны извне, называются public, те, которые доступны только внутри класса – private. C# не ограничивает области видимости, любой элемент может быть как public, так и private. Две причины использования инкапсуляции – это контроль использования и минимизация воздействий при изменениях. Можно использовать инкапсуляцию данных и определить поведение для того, чтобы с объектом работали по заданным правилам. Вторая причина вытекает из первой, если данные закрыты от внешнего использования, то их изменение не влияет на использование объекта извне. Большинство данных внутри объектов описывают информацию об индивидуальности объекта. Данные внутри объекта обычно private и доступны только из методов класса. Иногда необязательно хранить информацию внутри каждого объекта. Т.е. может быть информация, одинаковая для всех объектов данного класса. Для этого используются статичные поля, которые принадлежат не конкрет-

ному объекту, а всему классу. 50 Статичные методы инкапсулируют статичные данные. Статичные методы существуют на уровне класса, в них нельзя использовать оператор this, но можно обращаться к полям класса, если получить объект класса как параметр. class Time { public static void Reset(Time t) { t.hour = 0; // ОК t.minute = 0; // ОК hour = 0; // ошибка при компиляции minute = 0; // ошибка при компиляции } private int hour, minute; } Теперь вернемся к программе Hello world, рассмотрим её со стороны объектно-ориентированного программирования. Ответим на два вопроса: как при выполнении вызывается класс и почему метод Main статичный? Если в файле два класса с методом Main, то точка входа определяется при компиляции. // TwoEntries.cs using System; class EntranceOne { public static void Main( ) { Console.Write("EntranceOne.Main( )"); } } class EntranceTwo { public static void Main( ) { Console.Write("EntranceTwo.Main( )"); } } // Конец файла 51 c:\> csc /main:EntranceOne TwoEntries.cs c:\> twoentries.exe EntranceOne.Main( ) c:\> csc /main:EntranceTwo TwoEntries.cs c:\> twoentries.exe EntranceTwo.Main( ) c:\> Если в файле нет классов с методом Main, то из него нельзя скомпилировать запускаемый файл, только библиотеку dll. // NoEntrance.cs using System; class NoEntrance { public static void NotMain( ) { Console.Write("NoEntrance.NotMain( )"); } } // Конец файла c:\> csc /target:library NoEntrance.cs c:\> dir ... NoEntrance.dll ... Почему метод Main должен быть статичным? Так как для вызова нестатичного метода необходимо создать объект, а при вызове Main программа только начинает работу, и никаких объектов ещё нет. Для определения простых классов необходимо выполнить следующую последовательность действий: обозначить ключевым словом class начало класса, определить поля как в структурах, определить методы внутри класса, установить модификаторы доступа для всех полей и методов класса. Модификатор public означает, что “доступ неограничен”, private – “доступ ограничен типом, которому принадлежит”. Если пропустить модификатор, то по умолчанию поле или метод будут private. 52 class BankAccount { public void Deposit(decimal amount) { balance += amount; } private decimal balance; } При объявлении объекта класса, объект не создаётся, необходимо использовать оператор new, при этом все поля инициализируются нулями. При использовании объекта без создания будет ошибка при компиляции. Time now = new Time(); // пример объявления объекта класса Ключевое слово this неявно указывает на объект вызвавший метод. Например, в следующем примере полю name нельзя присвоить значение, компилятор будет считать его параметром class BankAccount { public void SetName(string name) { name = name; } private string name; } Здесь необходимо было использовать this.name = name; В C# можно выделить пять различных видов типов: class struct interface enum delegate Любой из них можно использовать в классе. Т.е. в классе могут содержаться другие классы. Вложенные классы должны помечаться модификатором доступа, использование вложенных классов переводит имена из глобальной области видимости и пространства имен. Public вложенные классы не имеют ограничений по использованию, полное имя класса можно использовать в любом месте программы. Private вложенный класс виден только из класса, содержащего его. Класс без модификатора по умолчанию является private. 53 Наследование и полиморфизм Наследование – это связи на уровне классов. Новый класс может наследовать существующий класс. Наследование – это мощная связь, так как наследуемый класс наследует все (не private) элементы базового класса. От базового класса может наследоваться любое количество классов. Изменение базового класса, автоматически изменяет все

классы-потомки. Классы потомки могут быть одновременно и базовыми классами для других классов. Группа классов, связанных наследованием, формирует структуру, называемую иерархией классов. При движении по иерархии вверх переходим к более общим классам. При движении вниз – к более специализированным классам. Простое наследование – это случай, когда у класса есть только один прямой базовый класс. Множественное наследование, когда у класса есть несколько прямых базовых классов. Множественное наследование создаёт предпосылки к ошибочному использованию наследования. Поэтому C#, как и большинство современных языков программирования, запрещает множественное наследование. Напомним, что наследование, особенно множественное, позволяет рассматривать один объект с разных точек зрения. Полиморфизм с литературной точки зрения означает много форм или много обликов. Это концепция, по которой один и тот же метод, определенный в базовом классе, может быть по-разному определен в разных классах потомках. Появляется новая проблема, как работать методу у объекта базового класса? Есть возможность не определять метод в базовом классе, т.е. оставить тело метода пустым. Такие методы называют операциями. В типичной иерархии классов операции объявляются в базовом классе, а определяются различными путями в различных классах потомках. Базовый класс представляет имя метода в иерархии. В случае, когда метод не определен в базовом классе, то нельзя создавать объекты этого класса, так для этого объекта будет не определен этот метод. Такие классы называются абстрактными. Абстрактные классы и интерфейсы похожи, так как не могут иметь объектов. Отличие между ними в том, что абстрактный класс может содержать 54 определения методов, интерфейсы содержат только операции (имена методов). То есть интерфейсы абстрактнее абстрактных классов. Когда вызываем метод напрямую из объекта, а не из операции базового класса, то метод связывается с вызовом при компиляции – раннее связывание. При вызове метода не напрямую через объект, а через операцию базового типа, он вызывается при выполнении программы – позднее связывание. Гибкость позднего связывания обеспечивается физической и логической ценой. Позднее связывание выполняется дольше, чем раннее. При позднем связывании классыпотомки могут заменять базовые классы. Операции могут быть вызваны из интерфейса, а классы-потомки обеспечат правильное выполнение.

Вопросы к разделу 1. Объясните концепцию абстракции, и почему она важна для программной инженерии? 2. Назовите два принципа инкапсуляции. 3. Опишите наследование в контексте ООП. 4. Что такое полиморфизм? Как он связан с ранним и поздним связыванием? 5. Опишите разницу между интерфейсами, абстрактными классами и конкретными классами.

Лабораторная работа Задания на классы. Время, необходимое на выполнение задания 45 мин. Упражнение 7.1 Создать класс счет в банке с закрытыми полями: номер счета, баланс, тип банковского счета (использовать перечислимый тип из упр. 3.1). Предусмотреть методы для доступа к данным – заполнения и чтения. Создать объект класса, заполнить его поля и вывести информацию об объекте класса на печать. Упражнение 7.2 Изменить класс счет в банке из упражнения 7.1 таким образом, чтобы номер счета генерировался сам и был уникальным. Для этого надо создать в классе статическую переменную и метод, который увеличивает значение этого переменной. Упражнение 7.3 Добавить в класс счет в банке два метода: снять со счета и положить на счет. Метод снять со счета проверяет, возможно ли снять запрашиваемую сумму, и в случае положительного результата изменяет баланс. Домашнее задание 7.1 Реализовать класс для описания здания (уникальный номер здания, высота, этажность, количество квартир, подъездов). Поля сделать закрытыми, предусмотреть методы для заполнения полей и получения значений полей для печати. Добавить методы вычисления высоты этажа, количества квартир в подъезде, количества квартир на этаже и т.д. Предусмотреть возможность, чтобы уникальный номер здания генерировался программно. Для этого в классе предусмотреть статическое поле, которое бы хранило последний использованный номер здания, и предусмотреть метод, который увеличивал бы значение этого поля.

Методическое издание

«ПРИКЛАДНОЕ ПРОГРАММИРОВАНИЕ»:

методические указания по лабораторной работе

для студентов Колледжа профессионального образования специальностям 09.02.03 Программирование в компьютерных системах

Составитель:

Бочкарев Алексей Михайлович

Редактор _____

Корректор _____

Подписано в печать _____

Формат 60x84/16. Усл.печ.л. _____. Уч.-изд.л. _____.

Тираж 100 экз. Заказ

Редакционно-издательский отдел

Пермского государственного университета

614990. Пермь, ул. Букирева, 15

Типография Пермского государственного университета

614990. Пермь, ул. Букирева, 15