Учреждение образования
БЕЛОРУССКАЯ ГОСУДАРСТВЕННАЯ АКАДЕМИЯ СВЯЗИ
Кафедра ПОСТ

ЛАБОРАТОРНЫЙ ПРАКТИКУМ
ПО ДИСЦИПЛИНЕ
«КОНСТРУИРОВАНИЕ ПРОГРАММ И ЯЗЫКИ ПРОГРАММИРОВАНИЯ»

для учащихся дневной формы получения образования
по специальности
2-40 01 31 – Тестирование программного обеспечения

Составители: Лущик Н. И.
Труханович Т.Л.

Утвержден на заседании кафедры ПОСТ
протокол № 1 от 30.08.2018 г.

Зав. кафедрой ПОСТ _____ РыбакВ.А.

Минск,
2018 г.

СОДЕРЖАНИЕ

Лабораторная работа №1
Этапы работы с программой на java

Цель: сформировать умения установки JDK, а также компиляции и выполнения программ на языке Java; сформировать умения программирования линейных алгоритмов на языке Java с использованием вывода данных.

Аппаратное, программное обеспечение: персональный компьютер, JDK, текстовый редактор (TextPad или NotePad++).

Краткие теоретические сведения

Структура программы на языке Java:

```
/* Комментарий, отражающий назначение программы */
public class Classname {      // Choose a meaningful Classname.
                              //Save as "Classname.java"
  public static void main(String[] args) {  // точка входа в программу
    // операторы
  }
}
```

Порядок установки JDK и запуска программ

Шаг 0: Деинсталлировать старые версии JDK/JRE.

Открыть *«Control Panel»* ⇒ *«Program and Features»* ⇒ Деинсталлировать все программы, связанные с Java, такие как «Java SE Development Kit ...», «Java SE Runtime ...», и т.д..

Рекомендуется установить последнюю версию JDK.

Шаг 1: Загрузить JDK. Дистрибутивы можно найти по ссылке *http://www.oracle.com/technetwork/java/javase/downloads/index.html.*

Далее «Java Platform, Standard Edition» ⇒ «Java SE 8u{xx}», где {xx} номер последней версии ⇒ Нажать кнопку «JDK Download».

Выбрать «Accept License Agreement».

Выбрать платформу «Windows x64» (для 64-х битной Windows OS) или «Windows x86» (для 32-х битной Windows OS). Выбрать 32-х битную или 64-х битную Windows OS можно через «Control Panel» ⇒ «System» ⇒ Under «System Type».

Шаг 2: Установить JDK и JRE

Запустить загруженный инсталлятор (jdk-8u{xx}-windows-x64.exe), который установит как JDK, так и JRE. По умолчанию JDK будет установлен в директорию «C:\Program Files\Java\jdk1.8.0_xx», где xx обозначает номер последнего обновления версии; а JRE а директорию «C:\Program Files\Java\jre1.8.0_xx». Как и при установке других программ, директорию можно указать. Установим в директорию <JAVA_HOME>.

Шаг 3: Прописать путь к директории «bin» JDK в переменной окружения PATH.

ОС Windows ищет текущую директорию для исполняемых программ в списке значений переменной окружения PATH. Программы JDK (такие как Java-компилятор *javac.exe* и Java-исполнитель java.exe) физически находятся в каталоге «<JAVA_HOME>\bin» (где <JAVA_HOME> обозначает путь к директории, где установлен JDK installed directory). Необходимо включить «<JAVA_HOME>\bin» в список значений переменной окружения PATH, чтобы

запустить JDK программы.

Чтобы отредактировать список значений переменной окружения PATH в Windows XP/Vista/7 необходимо:

Вызвать *«Control Panel» ⇒ «System» ⇒ «Advanced system settings»*. Переключить на вкладку *«Advanced» ⇒ «Environment Variables»*.

В Windows 8/10:

Вызвать *«Панель управления» ⇒ «Система и безопасность» ⇒ «Система» ⇒ «Дополнительные параметры системы»*. Вкладка *«Дополнительно» ⇒ «Переменные среды…»*

В *«System Variables»*, прокрутить вниз и выбрать *«Path» ⇒ «Edit...»*.

Необходимо быть очень внимательным, та как операции отменить в данном случае нет.

Для Windows 10: будет таблица, содержащая существующие значения путей. Нажать *«New»* ⇒ ввести путь к директории bin JDK *«c:\Program Files\Java\jdk1.8.0_xx\bin»* (Заменить *Replace xx* соответствующим номером последнего обновления версии) ⇒ выбрать *«Move Up»* чтобы переместить их до самого верха.

В версиях до Windows 10: В поле *«Variable value»*, ввести *«c:\ Program Files\Java\jdk1.8.0_xx\bin»* впереди списка существующий директорий и поставить (;), которая является разделителем в списке значений. Не удалять существующие значения путей, иначе установленное прикладные программы могут не работать!

Variable name : PATH

Variable value : c:\Program Files\Java\jdk1.8.0_xx\bin;[exiting entries...]

Шаг 4: Проверить, установились ли JDK

Вызвать командную строку (CMD shell) (Нажать *«Start» ⇒ run...* ⇒ запустить *«cmd»*; или *«Start» ⇒ All Programs ⇒ Accessories ⇒ Command Prompt*).

Ввести команду *«path»*, чтобы отразить список существующих значений переменной окружения PATH. Убедиться, что <JAVA_HOME>\bin отображен в списке.

```
prompt> path
    PATH=c:\Program Files\Java\jdk1.8.0_xx\bin;[other entries...]
```

Ввести следующую команду, чтобы проверить, что JDK/JRE правильно установлены и отображают номер их версий:

```
// Отображение номера версии JRE
prompt> java -version
    java version "1.8.0_xx"
    Java(TM) SE Runtime Environment (build 1.8.0_xx-b13)
    Java HotSpot(TM) 64-Bit Server VM (build 25.5-b02, mixed mode)
// Отображение номера версии JDK
prompt> javac -version
    javac 1.8.0_xx
```

Шаг 5: Написать код простейшей Java-программы, которая выводит строку-приветствие.

Создать рабочую директорию, в которой будут хранится разработанные программы, например, «d:\myProject», или «c:\myProject». Свои работы не рекомендуется сохранять в «Desktop» или «Documents» поскольку путь к ним трудно найти и прописать. Имя директории не должно содержать пробелов и

специальных символов.

Запустить текстовый редактор (можно TextPad или NotePad++). Набрать код программы:

```
/*
 * First Java program to say Hello
 */
public class Hello {   // Save as "Hello.java" under "d:\myProject"
  public static void main(String[] args) {
    System.out.println("Hello, world!");
  }
}
```

Сохранить в файле с именем «Hello.java», в рабочей директории (d:\myProject). Код программы на Java необходимо сохранять в файле с именем соответствующий имени класса, программный код которой хранится в данном файле.

Шаг 6: Скомпилировать и запустить программу

Чтобы скомпилировать исходный код «Hello.java»:

1. Запустить командную строку (CMD Shell).

2. Установить в качестве текущей директории диск, где сохранен файл с исходным кодом. Например, если файл сохранен на диске d, то чтобы перейти на него необходимо ввести «d:» и нажать «Enter»:

prompt> d:
D:\xxx>

3. С помощью команды *cd* перейти в рабочую директорию:

D:\xxx> cd \myProject
D:\myProject>

4. Ввести команду *dir*, с помощью которой можно просмотреть список файлов в директории и убедиться, что файл с исходным кодом в нем присутствует:

D:\myProject> dir
.....
xx-xxx-xx 06:25 PM 277 Hello.java
.....

5. Вызвать JDK-компилятор «javac» для компиляции исходного кода «Hello.java»:

D:\myProject> javac Hello.java

Если компиляция прошла успешно, то возвращается командная строка. Иначе будут отображены сообщения об ошибках, найденных в процессе компиляции.

6. Результатом компиляции станет файл с расширением *.class* (в данном случае «Hello.class»). Ввести команду *dir*, чтобы проверить его наличие:

D:\myProject> dir
.....
xx-xxx-xx 01:53 PM 416 Hello.class
xx-xxx-xx 06:25 PM 277 Hello.java
.....

7. Чтобы запустить программу необходимо вызвать Java Runtime «java»:

D:\myProject> java Hello

Hello, world!

Исходный код для JDK предоставляется и хранится в «<JAVA_HOME> \ src.zip». Рекомендуется ознакомиться с некоторыми исходными файлами, такими как «String.java», «Math.java» и «Integer.java», в разделе «java \ lang».

Средства вывода данных в языке Java

Для вывода сообщения в консоль можно использовать:

- ***System.out.println(aString)*** печатает строку *aString* и переводит курсор в начало новой строки;
- ***System.out.print(aString)*** печатает строку *aString*, но курсор оставляет сразу за ней.

Пример. Найдём сумму двух целых чисел

```
public class TwoNumberSum {   // Сохранить как "TwoNumberSum.java"
  public static void main(String[] args) {
    /*Объявляем две целочисленные переменные с присваиваем им значения*/
    int number1 = 11;
    int number2 = 22;
    int sum;  // объявление переменной для накопления суммы
    sum = number1 + number2;  // Compute sum
    System.out.print("The sum is ");  // Вывод поясняющего сообщения
    System.out.println(sum);/*Вывод значения, хранящегося в переменной sum*/
  }
}
```

`System.out.print()` и `println()` не обеспечивают форматирование вывода, например, управление количеством позиций для печати *int* и количеством десятичных знаков для *double*.

Начиная с Java SE 5 представлен новый метод `printf()` для форматного вывода (схож с функцией `printf()` в языке C), который имеет вид:

printf(*formatting-string, arg1, arg2, arg3, ...*);

Formatting-string содержит *обычный текст* и *спецификаторы формата* – ***"%[flags][width]conversion-code"***, которые будут заменены на значения аргументов с учетом их форматирования в порядке их следования. Спецификатор формата начинается с «%» и заканчивается кодом преобразования: `%d` для целых чисел, `%f` для дробных значений, `%c` для символов и `%s` для строк.

Необязательный параметр `[width]` может быть вставлен между ними, чтобы указать ширину поля. Аналогичным образом, дополнительные опции *[flags]* могут использоваться для управления выравниванием, дополнением и другими. Например:

- `%αd`: целое число, напечатанное в α позиций.
- `%αs`: строка, напечатанная в α позиций. Если не указано, то количество позиций соответствует длине строки.
- `%α.βf`: дробное значение (`float` and `double`) напечатанное α позиций до , и β позиций после.
- `%n`: символ перехода на новую строку (в Windows – `"\r\n"`, в Unix и Mac – `"\n"`).

Примеры:

Метод	Результат
System.out.printf("Hello%2d and %6s", 8, "HI!!!%n");	*Hello*8 and ****HI!!!*
System.out.printf("Hi,%s%4d%n", "Hello", 88);	*Hi,Hello**88*
System.out.printf("Hi, %d %4.2f%n", 8, 5.556);	*Hi, 8 5.56*
System.out.printf("Hi,%-4s&%6.2f%n", "Hi", 5.5);	*Hi,Hi**&**5.50*
// "%-ns" для выравнивания по левому краю	
System.out.printf("Hi, Hi, %.4f%n", 5.56);	*Hi, Hi, 5.5600*

Следует обратить внимание, что `printf()` не переводит курсор на новую строку. Для того чтобы перевести курсор на новую строку необходимо использовать спецификатор "`%n`" (в языке C "`\n`"). Также, обращает на себя внимание, что метод `printf()` принимает переменной количество аргументов.

Работа с пакетами

Пакет, подобно библиотеке – это набор классов, а также других связанных с ним объектов, таких как интерфейсы, ошибки, исключения, аннотации и перечисления.

На UML-диаграммах пакеты обозначаются в виде папок, как показано на рисунке 2.1.1. В верхней части пишется имя пакета, в нижней перечисляются названия составляющих его объектов.

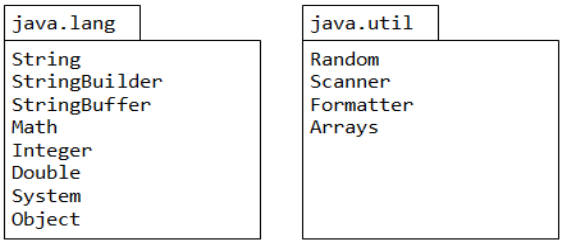


Рисунок 2.1.1 – Изображение пакета на UML-диаграммах

Пакеты используются для:

1. Организации классов и относящихся к ним объектов.
2. Управления пространством имен – каждый пакет – свое пространство имен.
3. Разрешения конфликтов имен.
4. Контроля доступа. Кроме public и private можно присвоить доступ видимый внутри пакета.
5. Распространение классов и пакетов. Все объекты в пакете могут быть скомбинированы и помещены в файл .jar, который можно распространять.

Имя пакета и имя класса вместе составляют полное имя (*fully-qualified name)*, которое однозначно идентифицирует класс. Имя пакета состоит из перевернутого в обратном порядке имени домена плюс собственное имя проекта, разделенные через точку. Имена пакетов пишутся символами нижнего регистра. Пример: «com.zzz.project1.subproject2».

Точка в имени пакета передает структуру директорий для хранения файлов классов. Например, пакет «com.yyy.project1.subproject2.Orange» хранится в директории «...\com\yyy\project1\subproject2\Orange.class», где «...» обозначает базовую директорию пакета.

JVM может разместить файлы классов, только если указаны базовая

директория пакета и полное имя. Базовая директория пакета обеспечивает путь к классу. Точка не обозначает подпакет (в Java нет такого понятия). Например, java.awt и java.awt.event – два различных пакета. Пакет java.awt содержится в директории «...\java\awt»; тогда как пакет java.awt.event хранится в «...\java\awt\event».

Есть два способа сослаться на класс в исходных кодах:

1. Использовать полное имя в форме *packagename.classname* (java.util.Scanner).

Пример. Ссылка на класс

```
public class ScannerNoImport {
  public static void main(String[] args) {
    // Используется полное имя пакета во всех ссылках
    java.util.Scanner in = new java.util.Scanner(System.in);
    System.out.print("Enter a integer: ");
    int number = in.nextInt();
    System.out.println("You have entered: " + number);
  }
}
```

2. Использовать оператор *import*. Добавить строку вида

import полное_имя_пакета

в самое начало программного кода. Далее в коде можно будет использовать только имя класса.

Пример. Использование import

```
import java.util.Scanner;
public class ScannerWithImport {

  public static void main(String[] args) {
    // Имя пакета может быть опущено для импортированного класса
    Scanner in = new Scanner(System.in);  // только имя класса
    System.out.print("Enter a integer: ");
    int number = in.nextInt();
    System.out.println("You have entered: " + number);
  }
}
```

Компилятор, когда столкнется с неразрешенным именем класса, будет искать оператор *import*, чтобы получить полное имя класса. Оператор *import* обеспечивает возможность обращаться к классу без указания его полного имени. Данный оператор не выполняет загрузку класса. Загрузку класса выполняет загрузчик во время выполнения программы. Оператор *import* просто ставит в соответствие имени класса его полное имя и вносит имя класса в пространство имен. Он работает только во время компиляции. Java-компилятор заменяет имя класса на полное имя и удаляет все операторы *import* из байт-кода. Это приносит небольшие затраты во время компиляции, но исключает затраты во время выполнения.

Операторы *import* в программном коде должны быть размещены после оператора *package*, но перед объявлением классов. Синтаксис:

import packagename.classname;
import packagename.*

Если нужно импортировать только один класс, то пишется его полное имя. Если нужно импортировать все классы пакета, то пишется *.

Примеры:

Оператор	Назначение
import java.util.Scanner;	*импорт класса Scanner из пакета java.util*
import java.awt.Graphics;	*импорт класса Graphics in package ja-va.awt*
import java.util.;*	*импорт всех классов из пакета java.util*
import java.awt.;*	*импорт всех классов из пакета java.awt*
import java.awt.event.;*	*импорт всех классов из пакета java.awt.event*

Использование * сократит несколько строк в исходном коде, но никак не повлияет на байт код. Однако это может привести к путанице, если в разных пакетах существуют классы с одинаковыми именами.

Пакет *java.lang*, составляющий ядро языка Java, неявно импортируется в каждую программу. Таким образом, не нужно импортировать его классы (такие как *System*, *String*, *Math*, *Integer* и *Object*) в программу с помощью оператора *import*. Также нет необходимости использовать оператор *import* для классов в пределах того же пакета.

Начиная с версии JDK 1.5 статические переменные и методы класса могут быть импортированы через оператор *import static*. Таким образом, можно опустить имя класса для импортируемой статической переменной или метода. Синтаксис оператора *import static*:

import static packagename.classname.staticVariableName;
import static packagename.classname.staticMethodName;
import static packagename.classname.*;

* означает импорт всех переменных и методов класса.

Пример. Импорт статических переменных

```
import static java.lang.System.out;
    /*импорт статической переменной out из класса System*/
import static java.lang.Math.*;
    /*импорт всех статических переменных и методов из класса Math*/
public class TestImportStatic {
  public static void main(String[] args) {
    out.println("Hello, PI is " + PI);
    out.println("Square root of PI is " + sqrt(PI));
  }
}
```

Статический импорт удобен в такой ситуации. Предположим, необходимо определить множество констант в программе. С одной стороны это можно сделать, объявив *interface*. Вместо этого можно объявить в одном классе константы и подключить их с помощью *import static* к другим классам.

Пример. Импорт статических констант

```
public class GameMain {
  public static final ROWS = 3;
  public static final COLS = 3;
}
import static GameMain.ROWS;// импорт константы ROWS
import static GameMain.COLS; ;// импорт константы COLS
public class GamePanel {
  int[][] score = int[ROWS][COLS];
  for (int row = 0; row < ROWS; ++row) {
    for (int col = 0; col < COLS; ++col) {
    }
  }
}
```

Преимущества такого подхода:

1. Данные константы можно будет использовать без имени класса.
2. Если необходимо перенести константу в другой класс, то нужно просто изменить оператор *import static* и не перерабатывать весь программный код.

Чтобы поместить класс в пакет, необходимо добавить оператор *package* перед определением класса (первым оператором программы).

Пример. Пакет

```
package com.zzz.test;
public class HelloPackage {
  public static void main(String[] args) {
    System.out.println("Hello from a package...");
  }
}
```

Можно создавать и использовать пакеты в IDE, что будет проще, так как IDE берет на себя детали. Там имеется возможность визуальными средствами создать новый пакет и создать класс в пакете.

Чтобы скомпилировать классы в пакете используя JDK, необходимо использовать утилиту *javac* с опцией *-d*, чтобы указать базовую директорию.

>javac ***-d e:\myproject*** *HelloPackage.java*

Опция *-d* указывает компилятору поместить файлы с классами в данную базовую директорию, а также создать необходимую структуру для каталогов класса. «.» в имени пакета передает структуру поддиректорий. Скомпилированный байт-код для *com.zzz.test.HelloPackage* будет помещен в «*e:\myproject\com\zzz\test\HelloPackage.class*»

Чтобы запустить программу, необходимо перейти в базовую директорию пакета и запустить утилиту *java*, указав полное имя программы:

e:\myproject> java com.zzz.test.HelloPackage

Важно помнить, что всегда нужно работать в базовой директории пакета и указывать полное имя класса.

Если класс не помещен в определенный пакет, то считается что он принадлежит *default unnamed package*. Такой подход не рекомендуется использовать, т.к. программу нельзя импортировать в другие приложения.

Java приложение обычно состоит из множества классов. Для упрощения ее распространения можно связать все файлы и относящиеся к ним ресурсы в один

единый файл, который имеет расширение *JAR*. (Java Archive). Он использует известный алгоритм сжатия «zip». Он создан по образцу Unix-"tar" (лента Архивный) утилиты. Можно также включить цифровую подпись (или сертификат) в файле JAR для проверки подлинности получателей.

В JDK имеется утилита «jar» с помощью которой можно создавать и управлять JAR-файлами, введя следующую команду:

>jar cvf myjarfile.jar c1.class ... cn.class

Пример:

Чтобы поместить ранее созданный класс в JAR-файл, нужно использовать утилиту *jar*, указав в качестве параметров имя пакета и имя помещаемого файла:

e:\myproject> ***jar cvf hellopackage.jar com\zzz\test\HelloPackage.class***

added manifest

adding:com/zzz/test/HelloPackage.class(in=454)(out=310)(deflated 31%)

Eclipse хранит файлы с исходным кодом в директории *src*, классы – в *bin*, jar-файлы и встроенные библиотеки – в *lib*. *NetBeans* хранит файлы с исходным кодом также в директории *src*, классы – в *build\classes*, jar-файлы и встроенные библиотеки – в *build\lib*.

Если два класса имеют одинаковое имя, то необходимо использовать полные имена для обоих из них, или импортировать один класс, а второй использовать с полным его именем. Попытка импортировать два класса вызовет ошибку компиляции.

Для того, чтобы определить местонахождение класса, нужно найти базовую директорию или JAR-файл.

JVM ищет классы в следующем порядке:

1. Классы Java Bootstrap: такие как «rt.jar» (runtime class), «i18n.jar» (internationalization class), charsets.jar, jre/classes, и др.

2. Стандартные классы Java Standard Extension classes: файлы JAR, located в директории «$JDK_HOME\jre\lib\ext» (для Windows и Ubuntu); «/Library/Java/Extensions» и «/System/Library/Java/Extensions» (для Mac). Местоположение каталогов расширения Java хранится в «java.ext.dirs».

3. Классы пользователя ищутся в следующем порядке:

1. По умолчанию в текущей директории.

2. В директориях согласно списка значений переменной окружения CLASSPATH, которая имеет приоритет по умолчанию.

3. По пути, указанному с помощью опции -cp (или -classpath), который перекрывает переменную CLASSPATH среды и по умолчанию.

4. По пути, указанному с помощью опции -jar, которые замещают все выше.

Практические задания и методические указания

1. Изучить порядок установки среды программирования на Java. теоретические сведения по теме: «Изучение этапов работы с программой на java в системе программирования».

2. Установить среду программирования и выполнить запуск тестовой программы согласно указаниям в теоретических сведениях.

3. Выполнить индивидуальные задания согласно варианту.

4. Создать пакеты и JAR-файлы согласно следующего описания (рисунок 2.1.1)

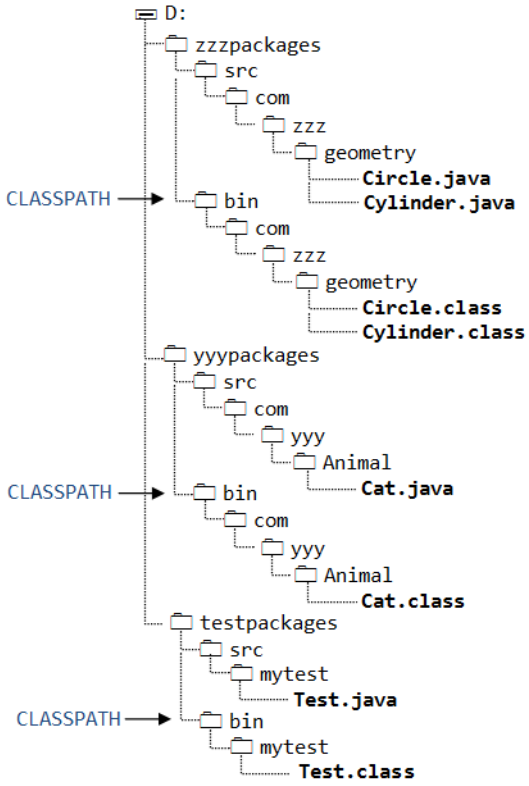


Рисунок 2.1.1 – Схема расположения файлов в пакетах

com.zzz.geometry.Circle

Создать класс *Circle* в пакете *com.zzz.geometry*. Файл с исходным кодом сохранить как *d:\zzzpackages**src**\com\zzz\geometry\Circle.java*, файл с классом – в базовую директорию пакета *d:\zzzpackages**bin***.

Исходный код:

```
package com.zzz.geometry;
public class Circle {
  public String toString() {
    return "This is a Circle";
  }
}
```

Чтобы скомпилировать класс *Circle*, необходимо использовать *javac* с опцией *–d*, чтобы указать расположение базовой директории пакета.

```
> javac -d d:\zzzpackages\bin Circle.java
```

com.zzz.geometry.Cylinder

Далее создать класс *Cylinder* в том же пакете (*com.zzz.geometry*), который наследует класс *Circle*.

Сохранить как *d:\zzzpackages\src\com\zzz\geometry\Cylinder.java*.

Исходный код:

```
package com.zzz.geometry;
public class Cylinder extends Circle {  // save as
  public String toString() {
    return "This is a Cylinder";
  }
}
```

Импортировать операторы класса *Circle* в класс *Cylinder* не требуется, т.к.

файлы с исходным кодом находятся в одной директории. Чтобы скомпилировать класс *Cylinder*, необходимо с помощью опции *-cp* (*or -classpath*) указать путь к исходному коду класса *Circle*, т.к. класс *Cylinder* ссылается на класс *Circle*.

> javac -d d:\zzzpackages\bin -cp d:\zzzpackages\bin Cylinder.java

com.yyy.animal.Cat

Создать другой класс *Cat* в другом пакете (*com.yyy.animal*). Файл с исходным кодом сохранить как *d:\yyypackages\src\com\yyy\animal\Cat.java* файл с классом – в базовую директорию пакета *d:\yyypackages\bin*.

Исходный код:

```
package com.yyy.animal;
public class Cat {   // save as d:\yyypackages\src\com\yyy\animal\Cat.java
  public String toString() {
    return "This is a Cat!";
  }
}
```

Чтобы скомпилировать класс Cat, использовать *javac* с опцией *–d*.

> javac -d d:\yyypackages\bin Cat.java

myTest.test

Создать класс Test в пакете *myTest*, который будет использовать созданные ранее классы. Файл с исходным кодом сохранить как *d:\testpackages\src\mytest\Test.java* файл с классом – в базовую директорию пакета *d:\testpackages\bin*.

Исходный код:

```
package mytest;
import com.zzz.geometry.Circle;
import com.zzz.geometry.Cylinder;
import com.yyy.animal.Cat;

public class Test {   // save as d:\testpackages\src\mytest\Test.java
  public static void main(String[] args) {
    Circle circle = new Circle();
    System.out.println(circle);
    Cylinder cylinder = new Cylinder();
    System.out.println(cylinder);
    Cat cat = new Cat();
    System.out.println(cat);
  }
}
```

Чтобы скомпилировать класс *Test*, необходимо использовать *javac* с опциями *–d*, чтобы указать размещение, и *–cp*, чтобы указать базовые директории пакетов классов *Circle* и *Cylinder* (*d:\zzzpackages\bin*) и *Cat* (*d:\yyypackages\bin*).

> javac -d d:\testpackages\bin -cp d:\zzzpackages\bin;d:\yyypackages\bin Test.java

Чтобы запустить класс *myTest.Test*, установить в командной строке в качестве текущей директории базовую директорию пакета *mytest.Test* (*d:\testpackages\bin*) и использовать *java*, указав с помощью опции *–cp* путь к классам *Circle* и *Cylinder* (d:\zzzpackages\bin), *Cat* (*d:\yyypackages\bin*).

> java -cp .;d:\zzzpackages\bin;d:\yyypackages\bin mytest.Test

Jarring-up com.zzz.geometry package

Упаковать пакет *com.zzz.geometry* в файл *geometry.jar* (сохранить в *d:\jars*). Для этого установить в командной строке в качестве текущей директории базовую директорию пакета (*d:\zzzpackages\bin*) и использовать *jar* с опциями *c* (создать), v, f:

prompt>jar cvf d:\jars\geometry.jar com\zzz\geometry.class*
added manifest
adding:com/zzz/geometry/Circle.class(in=300) (out=227)(deflated 24%)
adding:com/zzz/geometry/Cylinder.class(in=313)(out=228)(deflated 27%)

Чтобы упаковать все файлы из текущей директории, необходимо при вызове утилиты *jar* использовать «.» вместо указания пути к упаковываемым файлам:

>jar cvf d:\jars\geometry.jar .
added manifest
adding: com/(in = 0) (out= 0)(stored 0%)
adding: com/zzz/(in = 0) (out= 0)(stored 0%)
adding: com/zzz/geometry/(in = 0) (out= 0)(stored 0%)
adding:com/zzz/geometry/Circle.class(in=300) (out=227)(deflated 24%)
adding:com/zzz/geometry/Cylinder.class(in=313)(out=228)(deflated 27%)

Чтобы запустить *mytest.Test* из JAR-файла, необходимо использовать команду *java*, указав с помощью опции *-cp* пути (путь включает директорию и путь к *JAR*-файлу).

prompt> java -cp .;d:\jars\geometry.jar;d:\yyypackages\bin mytest.Test

5. Создать отчет, включив в него описание работы по пунктам 3 и 4, ответы на контрольные вопросы.

Индивидуальные задания

Написать на языке Java программы для решения следующих задач.

1. Дан рост одного человека x аршин. Выразить этот рост в вершках и сантиметрах.
2. Дан вес человека *x* пудов. Выразить этот вес в фунтах и килограммах.
3. Дано расстояние между двумя населенными пунктами *x* км. Перевести эту величину в версты, сажени.
4. Имеются три человека: у одного вес *x* пудов, у второго – *y* фунтов, у третьего – *z* кг. Найти средний вес.
5. Дано *x* градусов Цельсия. Перевести эту величину в градусы по Фарингейту.
6. Дано *x* м. Перевести эту величину в вершки и футы.
7. Роман Ж.Верна называется «Двадцать тысяч лье под водой». Если бы расстояние измерялось в километрах, то как бы звучало название романа?
8. Угол *A* задан в градусах, минутах и секундах. Найти его величину в радианах.
9. Дано *x* м. Перевести эту величину в дюймы и см.
10. Дана цена товара в рублях. Выразить ее в долларах США и евро.

Контрольные вопросы

1. Приведите структуру программы на языке Java.

2. Под каким именем необходимо сохранять файл с исходным кодом на языке Java?

3. Назовите компоненты среды программирования на Java?

4. Изобразите в виде блок-схемы алгоритм разработки и запуска программы на языке Java.

5. Назовите средства вывода данных в языке Java.

6. Что такое пакеты в Java? Какие опции и для чего используются при компиляции файлов в пакете?

7. Каково назначение операторов *import* и *package*?

8. Что представляют собой, для чего и как используются JAR-файлы?

9. Что делать, если имена классов совпадают?

10. Где JVM ищет классы?

11. Каковы структуры хранения файлов в средах *Eclipse* и *NetBeans*?

12. Приведите отличия в разработке программ на Java и C++.

Лабораторная работа №2
Основные конструкции языка Java

Цель: сформировать умения использовать условные и циклические операторы в языке Java.

Аппаратное, программное обеспечение: персональный компьютер, JDK, текстовый редактор (TextPad или NotePad++).

Краткие теоретические сведения

Типы данных и объявления переменных

Чтобы использовать переменные в программе, их нужно объявить, указав имя и тип данных. При объявлении переменная размещается в памяти, объемом, достаточным для хранения значения заданного типа.

Синтаксис объявления переменных в языке Java:

type identifier;

//объявление нескольких переменных типа *type*

type identifier1, identifier2, ..., identifierN;

type identifier = initialValue;

//объявление с инициализацией нескольких переменных типа *type*

type identifier1 = initValue1, ..., identifierN = initValueN;

Типы данных в языке Java делятся на две категориис: *primitive types* (примитивные типы) и *reference types* (объекты и массивы*)*. В таблице приведены примитивные типы языка Java.

Таблица 2.1.4 – Примитивные типы данных

Тип	Описание	
byte	Целое число	8-битное беззнаковое целое в диапазоне `[-2^7, 2^7-1] или [-128, 127]`
short		16-битное целое в диапазоне `[-2^15, 2^15-1] = [-32768, 32767]`
int		32-битное целое в диапазоне `[-2^31, 2^31-1] = [-2147483648, 2147483647]` (≈9 знаков)
long		64-битное целое в диапазоне `[-2^63, 2^63-1] = [-9223372036854775808, +9223372036854775807]` (≈19 знаков)
float	Число с плавающей точкой	32-битное число с плавающей запятой (≈6-7 значимых десятичных цифр, диапазон `±[≈10^-45, ≈10^38]`)
double		64-битное число с плавающей запятой (≈14-15 значимых десятичных цифр, диапазон `±[≈10^-324, ≈10^308]`)

Продолжение таблицы 2.1.4

Тип	Описание	
char	символ	Представляет 16-битную последовательность

ar		Unicode `'\u0000'` до `'\uFFFF'`. В арифметических операциях может быть интерпретирован как 16-битное беззнаковой целое в диапазоне `[0, 65535]`.
boolean	бинарный	Принимает значения либо *`true`* либо *`false`*. Размер логического значения не определен в спецификации Java, но требует как минимум одного бита.

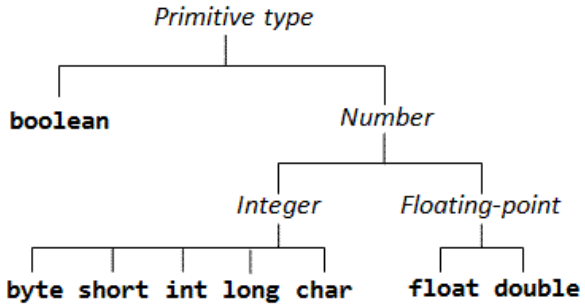


Рисунок 2.1.2 – Примитивные типы в языке Java

Java является «строго типизированным» языком программирования. Это означает, что переменная объявляется с определенным типом данных и может содержать значения только этого типа. Тип переменной не может быть изменен внутри программы после ее объявления. Переменные могут быть один раз объявлены в любом месте программы, но до того как они будут использованы.

Константы – это не модифицируемые переменные, объявленные с помощью ключевого слова `final`. Их значение не может быть изменено в ходе выполнения программы. Константы должны быть проинициализированы при объявлении.

final double PI = 3.1415926;

Соглашения по именованию констант: Использовать слова, написанные символами верхнего регистра, разделенные «_». Например, `MIN_VALUE`, `MAX_SIZE`.

Операции и выражения

Выражение – это комбинация знаков операций и операндов (переменных и литералов), которая в результате вычисления дает значение определенного типа.

Пример:

```
double principal, interestRate;
principal * (1 + interestRate)  // вычисляется значение типа double
```

В языке Java существуют:

– *арифметические операции:* +,-,*,/,%.

– *составные операции присваивания:* +=,-=,*=,/=,%=.

– *унарные арифметические операции:* инкремент (++) и декремент (--) для всех primitive types (*`byte`*, *`short`*, *`char`*, *`int`*, *`long`*, *`float`*, *`double`*) кроме *`boolean`*).

– *операции сравнения (или операции отношения):*==(равно), !=(не равно),>,>=,<,<=.

– *логические операции*, которые оперируют только с операндами типа

`boolean` в порядке приоритета: !,^,&&,||.

– *операция ?*

booleanExpr ? trueExpr : falseExpr

Если *booleanExpr* истинно, то результат операции будет *trueExpr*, если ложно – *falseExpr*.

Пример. Вычислить значение выражения 1/(x-5)

```
double x=5;
System.out.print("%n y="+(x==5?"Функция не определена":1/(x-5)));
```

Условные операторы

```
if ( booleanExpression ) {
  true-block ;
} else {
  false-block ;
}
```

Пример. Условный оператор if

```
if (mark >= 50) {
  System.out.println("Поздравляем!");
  System.out.println("Так держать!");
 } else {
  System.out.println("Старайся!");
 }
```

Если блок состоит из одного оператора, то { } можно опускать. Однако, рекомендуется использовать для повышения читабельности кода.

```
if ( booleanExpr-1 ) {
  block-1 ;
} else if ( booleanExpr-2 ) {
  block-2 ;
} else if ( booleanExpr-3 ) {
  block-3 ;
} else if ( booleanExpr-4 ) {
  ......
} else {
  elseBlock ;
}
```

Пример:

```
if (mark >= 80) {
  System.out.println("A");
 }
 else if (mark >= 70) {
          System.out.println("B");
        } else if (mark >= 60) {
                  System.out.println("C");
                } else if (mark >= 50) {
                          System.out.println("D");
                        } else {
                                  System.out.println("F");
 }
```

Оператор выбора имеет синтаксис:

```
switch ( selector ) {
  case value_1:
    block_1; break;
  case value_2:
    block_2; break;
  case value_3:
    block_3; break;
  ......
  case value_n:
    block_n; break;
  default:
    default-block;
}
```

В качестве *selector* может быть использовано значение типов `int`, `byte`, `short` или `char`, а начиная с версии JDK 1.7 и `string`, использование значений типов `long`, `float`, `double` и `boolean` не допускается. Если оператор *break;* будет отсутствовать, то будут выполняться операторы последующих блоков пока не встретиться оператор *break;*, либо до конца оператора *switch*.

Пример. Оператор выбора switch

```
char oper; int num1, num2, result;
.....
switch (oper) {
  case '+': result = num1 + num2; break;
  case '-':  result = num1 - num2; break;
  case '*': result = num1 * num2; break;
  case '/': result = num1 / num2; break;
  default: System.err.println("Unknown operator);
}
```

Операторы циклов

Оператор *цикла с известным числом повторений* имеет синтаксис:

```
for (initialization; test; post-processing) {
  body;
}
```

Пример. Оператор for

```
int sum;
for (int number = 1; number <= 1000; ++number) {
  sum += number;
}
```

Оператор *цикла с предусловием* имеет синтаксис:

```
while ( test ) {
  body;
}
```

Пример. Цикл с предусловием

```
int sum, number = 1;
while (number <= 1000) {
  sum += number;
  ++number;
}
```

Оператор *цикла с постусловием* имеет синтаксис:

```
do {
  body;
} while ( test ) ;
```

Пример. Цикл с постусловием

```
int sum, number = 1;
do {
  sum += number;
  ++number;
} while (number <= 1000);
```

Если известно, что должна выполниться как минимум одна итерация цикла, то используется цикл с постусловием.

continue; – прекращение текущей итерации цикла

break; – выход из цикла

Практические задания и методические указания

1. Изучить теоретические сведения по теме «Условные операторы в языке Java».

2. Выполнить индивидуальное задание 1 согласно варианту.

3. Изучить теоретические сведения по теме «Условные операторы в языке Java».

4. Выполнить индивидуальное задание 2 согласно варианту.

5. Создать отчет, включив в него описание работы по пунктам 2 и 4, ответы на контрольные вопросы.

Индивидуальные задания

Задание 1. Написать программы на языке Java для решения следующих задач. Для решения задачи 1 использовать операцию ?, задачи 2 – оператор if, задачи 3 – оператор case.

Вариант	Задание
1	1.Вычислить значение выражения $y = \sqrt{x} - 5$. 2.На трех бензоколонках имеется *a, в, с* литров бензина. На какую бензоколонку вести новую партию бензина?
	3.Написать программу, которая по номеру дня недели (целому числу от 1 до 7) выдает в качестве результата количество уроков в вашей группе в этот день.
2	1.Вычислить значение выражения $y = \sqrt{7 - x}$. 2.В трех магазинах один и тот же товар имеет разные цены: в первом магазине *a руб.,* во втором – *b руб.,* в третьем – *с руб.* Определить в каком магазине товар самый дешевый.
	3.Написать программу, позволяющую по последней цифре числа определить последнюю цифру его квадрата.

Вариант	Задание
3	1.Вычислить значение выражения $y = \frac{x^2+4}{x-6}$. 2.В трех магазинах один и тот же товар имеет разные цены: в первом магазине *a руб.,* во втором – *b руб.,* в третьем – *c руб.* Определить в каком магазине товар самый дорогой.
	3.Вводится номер месяца. Дать этому месяцу наименование.
4	1.Вычислить значение выражения $y = \sqrt{x+9}$. 2.Дан рост трех человек: первого – *x см,* второго – *y см,* третьего – *z см*. Определить самого высокого.
	3.Вводится номер месяца 2014 года. Определить сколько дней в этом месяце.
5	1.Вычислить значение выражения $\frac{x+y}{y+1} - \frac{xy-12}{34+x}$. 2.Дан рост трех человек: первого – *x см,* второго – *y см,* третьего – *z см*. Определить самого маленького.
	3.Вводится номер месяца 2014 года. Определить время года, которому соответствует этот месяц. (Использовать оператор Case).
6	1.Вычислить значение выражения. $y = \sqrt{x-5}$. 2.Дан вес трех человек: первого – *x кг,* второго – *y кг*, третьего – *z кг*. Определить самого легкого человека. 3.Для каждой введенной цифры (0-9) вывести соответствующее ей название на английском языке (0 – *zero,* 1 – *one,2 – two,...*).
7	1.Вычислить значение выражения $y = \sqrt{7-2x}$. 2.Даны оценки по одному экзамену трех студентов: первый получил *a баллов*, второй – *в баллов*, третий – *c баллов.* Определить, кто сдал экзамен лучше всех. (Использовать сложное логическое выражение и неполную форму оператора *IF*). 3.Имеется пронумерованный список деталей: 1) шуруп, 2) гайка, 3) винт, 4) гвоздь, 5) болт. Составить программу, которая по номеру детали выводит на экран ее название.
8	1.Вычислить значение выражения $y = \frac{x^3+1}{x+6}$. 2.Студент сдал три экзамена и получил соответственно *a, b, c* баллов по каждому из экзаменов. Какой предмет он сдал лучше других. 3.Написать программу, которая по номеру месяца выдает название следующего за ним месяца (при *m* = 1 получаем февраль, 4 – май и т.д.).
9	1.Вычислить значение выражения $y = \sqrt{x+9}$. 2.Дан вес трех человек: первого – *x кг,* второго – *y кг*, третьего – *z кг*. Определить самого тяжелого человека. 3.Написать программу, которая бы по введенному номеру единицы измерения (1 – дециметр, 2 – километр, 3 – метр, 4 –миллиметр, 5 – сантиметр) и длине отрезка *L* выдавала бы соответствующее значение длины отрезка в метрах.

Вариант	Задание
10	1.Вычислить значение выражения $\frac{x+y}{y+1} - \frac{xy-12}{34+x}$. 2.Студент сдал три экзамена и получил соответственно *a, b, c* баллов по каждому из экзаменов. Какой предмет он сдал хуже других. 3.Составить программу, позволяющую по последней цифре данного числа определить последнюю цифру куба этого числа.

Задание 2.

Для решения задачи 1 использовать оператор цикла с постусловием *do{}while()*, для задачи 2 – оператор цикла с предусловием *while(){}*, для задачи 3 – оператор цикла с известным числом повторений *for(){}*.

Вариант	Задание
1	1.Составить таблицу значений функции $y= x^2- 2x +5$ на отрезке *[a; b]* с шагом *h*. 2.Сумма в *A руб.* лежит на расчетном счету. На сумму начисляется *в%* каждый год прибыли на ту сумму, которая находится на счету. Прибыль со счета не снимается. Через сколько лет данная сумма удвоится.
	3.Дан натуральный отрезок *[a; b]*. Вывести из этого отрезка все четные числа.
2	1.Составить таблицу значений функции $y=(x-2)/(x^2+4)$ на отрезке *[a; b]* с шагом *h*. 2.Спортсмен марафонец, готовясь к соревнованиям в первый день тренировки пробежал *A км.* Каждый следующий день он наращивал норму тренировки на *B%* от предыдущего дня. В какой день тренировок норма его пробега превысит *C км*?
	3.Дан целочисленный отрезок *[a; b]*. Вывести из этого отрезка все отрицательные числа.
3	1.Составить таблицу значений функции $y=x^3+2x^2-4x+7$ на отрезке *[a; b]* с шагом *h*. 2.Сумма в *A руб.* лежит на расчетном счету. На сумму начисляется *в%* каждый год прибыли на ту сумму, которая находится на счету. Прибыль со счета не снимается. Через сколько лет данная сумма удвоится.
	3.Дан натуральный отрезок *[a; b]*. Вывести из этого отрезка все числа, которые делятся на данное число *X*.
4	1.Составить таблицу значений функции $y=sin(x)-cos(x)$ а отрезке *[a; b]* с шагом *h*. 2.Спортсмен марафонец, готовясь к соревнованиям в первый день тренировки пробежал *A км.* Каждый следующий день он наращивал норму тренировки на *B%* от предыдущего дня. В какой день тренировок общий пробег за все дни превысит *C км*?
	3.Дан натуральный отрезок *[a; b]*. Вывести из этого отрезка все нечетные числа.
5	1.Составить таблицу значений функции $y=(x+2)^3$ на отрезке *[a; b]* с

Вариант	Задание
	шагом *h*. 2.Сумма в *A руб.* лежит на расчетном счету. На сумму начисляется *в%* каждый год прибыли. Прибыль со счета не снимается. Через сколько лет данная сумма превысит *C руб.*? 3.Найти сумму всех нечетных двухзначных чисел.
6	1.Составить таблицу значений функции $y=x^4-2x+8$ на отрезке *[a; b]* с шагом *h*. 2.Количество граждан некоторого города увеличивается ежегодно на *B%*. Через сколько лет население города возрастет в 2 раза, если вначале было *A* человек. 3.Найти сумму всех четных двузначных чисел.
7	1.Составить таблицу значений функции $y=(x^5+7x-1)/4$ на отрезке *[a; b]* с шагом *h*. 2.Мячик упал с высоты *P*. Ударился о землю и поднялся на 2/3 предыдущей высоты. Через сколько ударов мячик поднимется на высоту *H*? 3.Дан натуральный отрезок *[a; b]*. Вывести из этого отрезка все нечетные числа.
8	1.Составить таблицу значений функции $y=e^x+6x-3$ на отрезке *[a; b]* с шагом *h*. 2.Гриб за сутки увеличивает свою массу на 40%. Через сколько суток масса гриба увеличится в 2,5 раза, если первоначально масса гриба составляла *A*? 3.Найти сумму всех двузначных чисел кратных 6.
9	1.Составить таблицу значений функции $y=cos(x)+5$ на отрезке *[a; b]* с шагом *h*. 2.Сумма в *A руб.* лежит на расчетном счету. На сумму начисляется *в%* каждый год прибыли. Прибыль со счета не снимается. Через сколько лет данная сумма превысит *C руб.*? 3.Дан натуральный отрезок *[a; b]*. Вывести из этого отрезка все числа кратные 7.
10	1.Составить таблицу значений функции $y = cos(x+2)$ на отрезке *[a; b]* с шагом *h*. 2.Спортсмен марафонец, готовясь к соревнованиям в первый день тренировки пробежал *A км.* Каждый следующий день он наращивал норму тренировки на *B%* от предыдущего дня. В какой день тренировок норма его пробега превысит *C км*? 3.Дан натуральный отрезок *[a; b]*. Вывести из этого отрезка все числа, меньшие данного числа *X*.

Контрольные вопросы

1. Назовите средства языка Java для реализации условных алгоритмов.
2. Назовите средства языка Java для реализации циклических алгоритмов.

Лабораторная работа №3
Средства для ввода данных

Цель: научить использовать средства ввода данных языка Java Scaner и DialogBox, а также организовывать файловый ввод/вывод данны.

Аппаратное, программное обеспечение: персональный компьютер, JDK, текстовый редактор (TextPad или NotePad++).

Краткие теоретические сведения

Ввод данных с клавиатуры с использованием класса Scanner

В языке Java, как и в других языка программирования, поддерживается три стандартных потока ввода/вывода: `System.in` *(standard input device),* `System.out` *(standard output device*), и `System.err` *(standard error device).* `System.in` по умолчанию связан с клавиатурой; в то время как `System.out` и `System.err` – с консолью. Они могут быть перенаправлены на другие устройства, например, часто `System.err` перенаправляется на файл на диске, чтобы сохранять сообщения об ошибках.

В Java SE 5 для форматного ввода в пакете `java.util` представлен класс `Scanner`. Можно создать экземпляр класса `Scanner` для ввода из потока `System.in` (с клавиатуры), и использовать методы `nextInt()`, `nextDouble()`, `next()` для считывания следующего значения типа `int`, `double` или `String` (разделенных пробелами, знаками табуляции или переводом строки) соответственно.

Пример. Использование класса Scanner

```
import java.util.Scanner;  // импортируем метода класса Scanner
public class ScannerTest {
   public static void main(String[] args) {
    // Объявляем переменные разных типов
    int num1;
    double num2;
    String str;
    // Создаем экземпляр класса Scanner с именем in для чтения из потока
    // System.in (клавиатуры)
    Scanner in = new Scanner(System.in);
    // Чтение входных данных с клавиатуры
    System.out.print("Enter an integer: ");  // Вывод подсказки
    num1 = in.nextInt(); // Чтение целого значения
    System.out.print("Enter a floating point: ");
    num2 = in.nextDouble();  // Чтение значения типа double
    System.out.print("Enter a string: ");
    str = in.next();  // Чтение строки
    // Форматированный вывод
    System.out.printf("%s, Sum of %d & %.2f is %.2f%n", str, num1, num2,
    num1+num2);
    // закрыть поток ввода
    in.close();
   }
```

```
  }
```

Можно использовать метод `nextLine()` для считывания строки целиком, включая пробелы, но без символа перевода строки.

Пример. Использование NextLine

```
import java.util.Scanner;
public class ScannerNextLineTest {
  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.print("Enter a string (with space): ");
    String str  = in.nextLine(); //чтение строки с пробелами
    System.out.printf("%s%n", str);
    in.close();
  }
}
```

Не следует смешивать методы `nextLine()` и `nextInt() | nextDouble() | next()` в программе.

Ввод данных из текстового файла через класс Scanner

`Scanner` можно также соединять с другими источниками ввода, такими как файл на диске или сетевой сокет и использовать теже методы `nextInt()`, `nextDouble()`, `next()`, `nextLine()` для чтения следующего значения `int`, `double`, `String` и строки соответственно.

Пример:

```
// Construct a Scanner to scan a text file
Scanner in = new Scanner(new File("in.txt"));
// Use the same set of methods
int anInt = in.nextInt();        // next String
double aDouble = in.nextDouble(); // next double
String str = in.next();          // next int
String line = in.nextLine();     // entire line
```

Открывая файл через `new File (filename)`, нужно обработать исключение `FileNotFoundException`, (файл, который пытаемся открыть не найден), в противно случае программа не будет скомпилирована. Для обработки исключение можно использовать: *throws* либо *try-catch*.

Работа с исключениями в языке Java

Исключением является ненормальное (аномальное) событие, которое возникает в процессе выполнения программы и нарушает нормальный ход выполнения программы. Аномалии происходят во время выполнения программы. Например, ожидается, что пользователь введет целое число, а введена строка, не найден файл, который необходимо открыть. Если исключение не обработать должным образом, то программа резко прекратит выполняться, что может привести к нежелательным последствиям. Например, сетевые соединения, соединения с базами данных, файлы останутся открытыми, записи в базах данных и файлах не будут сохранены должным образом.

В языке С программист может не создавать программный код для

обработки исключения. Например, не обрабатывать ситуацию, если файл не существует. Или же реализовать обработку исключения используя оператор *if-else* в функции *main()*. В Java обработка исключений встроена в язык программирования. Программа не скомпилируется без кода обработки исключения. Разработчик должен быть проинформирован об исключениях, которые могут возникнуть при вызове метода, поэтому исключение объявляется при описании метода. Исключение должно быть обработано при написании главной логики работы программы. Код обработки исключения отделяется от логики работы главного кода конструкцией *try-catch-finally*.

Рассмотрим пример. Пусть необходимо использовать *java.util.Scanner*, чтобы выполнить форматированный ввод с файла на диске. Сигнатура метода-конструктора с параметром имя_файла выглядит следующим образом:

public Scanner(File source) throws FileNotFoundException;

Как видно из примера, сигнатура метода информирует программиста, что во время вызова метода может возникнуть исключительное условие «file not found».

Если метод объявлен с исключением в сигнатуре, то его нельзя использовать без обработки исключения – программа не будет компилироваться.

Пример. Ошибка компиляции

```
import java.util.Scanner;
import java.io.File;
public class ScannerFromFile {
  public static void main(String[] args) {
    Scanner in = new Scanner(new File("test.in"));  }
}
```

Результат: выдана ошибка компиляции

```
ScannerFromFile.java:5: unreported exception java.io.FileNotFoundException;
must be caught or declared to be thrown
      Scanner in = new Scanner(new File("test.in"));
```

Чтобы использовать такой метод в программе, необходимо реализовать один из следующих вариантов:

1. Заключить программный код в конструкции *try-catch* или *try-catch-finally*
2. Не обрабатывать исключительную ситуацию в данном фрагменте программного кода, но объявить, что исключение будет перенаправлено вверх в стеке вызовов следующему по уровню методу для обработки.

Пример. Перехват исключения с использованием конструкции *try-catch* (или *try-catch-finally*)

```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
public class ScannerFromFileWithCatch {
  public static void main(String[] args) {
    try {
      Scanner in = new Scanner(new File("test.in"));}
    catch (FileNotFoundException ex) {// если возникло исключение
      ex.printStackTrace();} //распечатать трассировку стека
  }
}
```

Если файл не найден, то сгенерированное исключение будет перехвачено блоком *catch*. В данном примере обработчик ошибок просто напечатает трассировки стека, в котором содержится полезная информация для отладки. Обратите внимание, что логика главного алгоритма в блоке *try* отделена от обработчика ошибок в блоке *catch*.

Пример. Перенаправление исключения в стеке вызовов вверх следующему по уровню методу для обработки

```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
public class ScannerFromFileWithThrow {
  public static void main(String[] args) throws FileNotFoundException {
    Scanner in = new Scanner(new File("test.in"));   }
}
```

В данном примере исключение *FileNotFoundException* сгенерированное в методе *Scanner(File)* не обрабатывается. Вместо этого метод *main()* объявлен с сигнатурой «*throws FileNotFoundException*», которая означает, что исключение будет передано в стек вызова на следующий уровень обработки. В данном случае следующий более высокий уровень метода *main()* – это JVM, которая просто завершит работу программы и печатает трассировку стека.

Организовать обработку исключений можно с помощью конструкции *try-catch-finally*, которая имеет следующий синтаксис:

```
try {
// main logic, uses methods that may throw Exceptions
  ......
} catch (Exception1 ex) {
// error handler for Exception1
  ......
} catch (Exception2 ex) {
// error handler for Exception1
  ......
} finally {// finally is optional
// clean up codes, always executed regardless of exceptions
......
}
```

Если во время выполнения программы не возникнет исключительной ситуации, то все блоки будут пропущены, но блок *finally* будет все равно выполнен после блока *try*. Если какой-то из операторов вызовет исключение, JVM проигнорирует оставшиеся операторы блока *try*, и начнет поиск соответствующего обработчика возникшего исключения. Поиск подходящего обработчика будет осуществляться исключения по всем блокам *catch* последовательно. Если блок *catch* перехватил исключение, то начинается выполнение операторов в этом блоке. Затем будут выполнены операторы блока *finally*. Затем программа продолжит выполнение со следующего оператора, после конструкции *try-catch-finally*, если она не будет досрочно прекращена или не произойдет выход из метода.

Пример.

```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
public class TryCatchFinally {
  public static void main(String[] args) {
    try {  // основной алгоритм
       System.out.println("Start of the main logic");
       System.out.println("Try opening a file ...");
       Scanner in = new Scanner(new File("test.in"));
       System.out.println("File Found, processing the file ...");
       System.out.println("End of the main logic");
    }
    catch (FileNotFoundException ex){// обработка ошибок
       System.out.println("File Not Found caught ...");
       }
    finally {// всегда выполняется независимо от статуса исключения
       System.out.println("finally-block runs regardless of the state of exception");
    }
    // после try-catch-finally
    System.out.println("After try-catch-finally, life goes on...");
  }
}
```

Результат, если в процессе выполнения возникло исключение FileNotFoundException

```
Start of the main logic
Try opening a file ...
File Not Found caught ...
finally-block runs regardless of the state of exception
After try-catch-finally, life goes on...
```

Результат, если исключений не возникало

```
Start of the main logic
Try opening a file ...
File Found, processing the file ...
End of the main logic
finally-block runs regardless of the state of exception
After try-catch-finally, life goes on...
```

За блоком *try* должен следовать как минимум один блок *catch* или блок *finally*. Может быть несколько блоков *catch*, каждый из которых перехватывает только один вид исключения.

Блоку *catch* требуется один аргумент, который является объектом *Throwable* (подкласс) *java.lang.Throwable:*

```
catch (A Throwable SubClass a ThrowableObject) {
   // код обработки исключения
}
```

Можно использовать следующие методы, чтобы получить тип исключения и состояние программы от объекта *Throwable*:

– *printStackTrace()*: печатает данные объекта *Throwable* и трассировки его стека вызовов в стандартный поток *System.err*. Первая строка вывода содержит

результат выполнения метода *toString()*, а остальные – трассировки стека вызовов. Это наиболее общий обработчик событий.

Пример:

```
try {
      Scanner in = new Scanner(new File("test.in"));
   // processthefilehere
      .....
} catch (FileNotFoundExceptionex) {
   ex.printStackTrace();
}
```

Можно также использовать *printStackTrace(PrintStream s)* или *printStackTrace(PrintWriter s)*.

– *getMessage()*: возвращает заданное сообщение, если объект строится с помощью конструктора *Throwable(String message)*.

– *toString()*: Возвращает краткое описание текущего объекта *Throwable*, состоящее из имени класса, далее идет «:», и сообщение метода *getMessage()*.

Блок *catch*, отлавливающий исключения определенного класса, может отлавливать исключения его подкласса. Из этого следует, что *catch(Exception ex) {...}* ловит все виды исключений. Тем не менее, это не является хорошей практикой, так как слишком общий обработчик исключения может непреднамеренно улавливать некоторые подклассы, которые он не должен улавливать.

Порядок расположения блоков *catch* важен. Подкласс должен быть пойман (и находится впереди), его суперкласса. В противном случае будет выдана ошибка компиляции «*exception XxxException has already been caught*».

Блок *finally* предназначен для очистки кода, как закрытие файла, соединение с базой данных независимо от того, выполнился ли блок *try* успешно.

Учитывая выше сказанное, программа на языке Java по обработке файлов может быть написана следующим образом:

Пример. Обработка файлов

```
import java.util.Scanner;
import java.io.File;
```

```
import java.io.FileNotFoundException;
import java.util.NoSuchElementException;
public class TextFileScannerWithCatch {
  public static void main(String[] args) {
    int num1;
    double num2;
    String name;
    try {//выполнение алгоритма считывания и обработки полученных данных
       Scanner in = new Scanner(new File("in.txt"));
       while(in.hasNext()){//пока есть символы в потоке
          num1=in.nextInt();
          num2=in.nextDouble();
          name=in.next();
          System.out.println(num1+" "+num2+" "+name+" ");
       }
       in.close();
    } //обработка возможных ошибок
    catch (FileNotFoundException ex) { // если файл не найден
       ex.printStackTrace(); //печать трассировки стека
    }
    catch (NoSuchElementException ex) {
       /*нет значения для считывания или значение не соответствуют ожидаемому
       формату (InputMismatchException является подклассом
       NoSuchElementException) */
       System.out.println("Input File is incorrect ...");
    }
  }
}
```

В данном программном коде использован метод *hasNext()*, который возвращает *true*, если в потоке еще имеются символы для считывания. Т.е. цикл, организующий считывание и обработку полученных данных продолжает работать, пока есть символы в потоке.

Форматированный вывод данных в текстовый файл

В Java SE 5.0 для форматированного вывода присутствует класс `Formatter`, содержащий метод `format()`. Этот метод имеет тот же синтаксис, что и метод `printf()`.

Пример. Использование метода format()

```
import java.io.File;
import java.util.Formatter;
import java.io.FileNotFoundException;
public class TextFileFormatterWithCatch {
  public static void main(String[] args) {
    try {
    /*используем класс Formatter для записи форматированного вывода в текстовый
    файл*/
```

```
        Formatter out = new Formatter(new File("out.txt"));
        //записываем в файл метода format() (похож на printf())
        int n1 = 1234;
        double n2 = 55.66;
        String name = "Tanya";
        out.format("Hi %s,%n", name);
        out.format("The sum of %d and %.2f is %.2f%n", n1, n2, n1 + n2);
        out.close();//закрыть файл
        System.out.println("Done"); //вывод в консоль
    }
    catch (FileNotFoundException ex) { //ловим исключение
        ex.printStackTrace(); //печать трассировки стека
    }
  }
}
```

Ввод данных через диалоговое окно

В Java в пакете *swing* имеется класс `JOptionPane`, с помощью которого можно организовать ввод данных через диалоговое окно, вид которого представлен на рисунке 2.2.1.

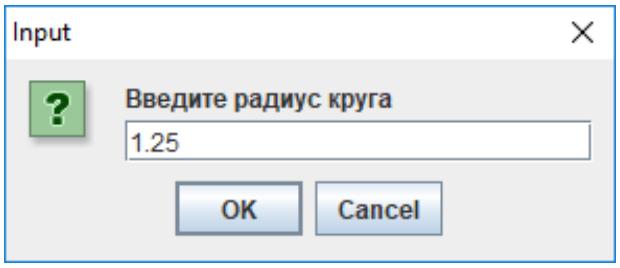


Рисунок 2.2.1 – Вид диалогового окна в Java

Пример. Программа просит пользователя ввести радиус круга, на основании чего вычисляет его площадь

```
import javax.swing.JOptionPane; // импорт класса JOptionPane
public class JOptionPaneTest {
  public static void main(String[] args) {
    String radiusStr;
    double radius, area;
    //чтение данных из диалогового окна ввода в переменную radiusStr
    radiusStr = JOptionPane.showInputDialog("Введите радиус круга");
    radius = Double.parseDouble(radiusStr);//преобразование String в double
    area = radius*radius*Math.PI;
    System.out.println("Площадь = " + area);
  }
}
```

Вывод диалогового окна осуществляет метод *method* `JOptionPane.showInputDialog(promptMessage)`, который получает в качестве параметра текст сообщения-подсказки для пользователя, выводимый в окне. Данные, введенные через данное окно поступают в программу в виде строки типа

String. Для выполнения арифметических операций с ними их необходимо конвертировать в тип *double*, что и выполняет метод *parseDouble()*.

Ввод данных через аргументы метода main()

Как и в языке С, в языке Java метод *main()* может принимать параметры, которые можно передать через командную строку. Принятые параметры хранятся в массиве *args[]* в виде строковых литералов.

Пример:

```
public class MainParamTest{
    public static void main (String[] args){
        System.out.println("Hello, " + args[0]+ "! It`s work! ");
    }
}
```

Результат выполнения программы приведен на рисунке 2.2.2.

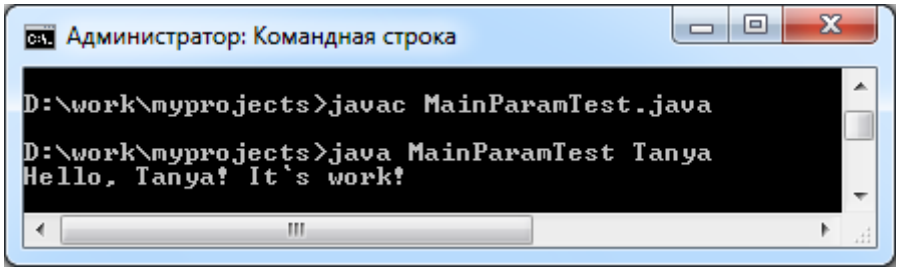


Рисунок 2.2.2 – Результат программы ввода данных через аргументы *main()*

Практические задания и методические указания

1. Изучить теоретические сведения по теме «Ввод данных с клавиатуры с использованием класса Scanner», выполнить приведенные примеры.

2. Модернизировать код индивидуального задания 1 из лабораторной работы №1, добавив в них форматированный ввод данных с клавиатуры.

3. Изучить теоретические сведения по теме «Ввод данных из текстового файла через класс Scanner» и «Форматированный вывод данных в текстовый файл», выполнить приведенные примеры.

4. Выполнить индивидуальные задания согласно варианту.

5. Модернизировать код индивидуального задания 1 из лабораторной работы №1, добавив в них ввод данных через диалоговое окно.

6. Изучить теоретические сведения по теме «Ввод данных через аргументы метода main()», выполнить приведенные примеры.

7. Модернизировать программный код, разработанный по индивидуальному заданию: имя файла передавать в качестве аргумента метода *main()*.

8. Создать отчет, включив в него описание работы по пунктам 1-7, ответы на контрольные вопросы.

Индивидуальные задания

1 Создать файл данных, компонентами которого являются вещественные числа. Найти сумму квадратов компонентов файла.

2 Создать файл данных, компонентами которого являются целые числа. Вывести на экран все четные числа данного файла.

3 Создать файл данных, компонентами которого являются вещественные

числа. Вывести на экран все числа этого файла, которые больше данного числа X и меньше данного числа У.

4 Создать файл данных, компонентами которого являются вещественные числа. Найти разность между первой и последней компонентой файла.

5 Создать файл данных, компонентами которого являются вещественные числа. Поменять местами первую и последнюю компоненту данного файла.

6 Создать файл данных, компонентами которого являются целые числа. Заменить в этом файле все четные числа на 0.

7 Создать файл данных, компонентами которого являются целые числа. Найти сумму нечетных чисел этого файла.

8 Создать файл данных, компонентами которого символы латинского алфавита (не в алфавитном порядке и не обязательно все). Поменять местами первую и 5-ую компоненты местами.

9 Создать файл данных, компонентами которого являются символы латинского алфавита (не в алфавитном порядке и не обязательно все). Поменять местами последнюю и 5-ую компоненты местами.

10 Создать файл данных, компонентами которого натуральные числа. Определить количество компонент, которые делятся на 3 без остатка.

Контрольные вопросы

1. Какие стандартные потоки поддерживаются в языке Java?
2. С помощью каких средств осуществляется ввод данных с клавиатуры в программах на Java.
3. Изобразите в виде блок-схемы алгоритм ввода данных.
4. С помощью каких средств осуществляется ввод/вывод данных из текстовых файлов в программах на Java.
5. Что такое исключения в Java? Как реализуется их обработка?
6. Является ли обработка исключений в программном коде обязательной?
7. Изобразите в виде блок-схемы алгоритмы файлового ввода и вывода данных.
8. Объясните назначение класса **JOptionPane.**
9. Изобразите в виде блок-схемы алгоритм ввода данных через диалоговое окно в программах на языке Java.
10. Где хранятся в программе данные, переданные в качестве аргументов метода *main()*? Каким образом передать параметры в программу?

Лабораторная работа №4
Работа с массивами в Java

Цель: сформировать знания об описании и инициализации, порядке размещения в памяти одно- и двумерных массивов, о доступе к элементам массива, и их обработке в языке Java; сформировать навыки решения задач с использованием массивов на языке Java.

Аппаратное, программное обеспечение: персональный компьютер, JDK, текстовый редактор (TextPad или NotePad++).

Краткие теоретические сведения

Одномерные массивы в Java

Массив – это упорядоченная коллекция элементов одного типа, идентифицируемых `[ ]`. Чтобы использовать массив в программе его необходимо *объявить, указав имя и тип,* и *определить* – в*ыделить память под его элементы, используя оператор* `new`*. В качестве имен массивов рекомендуется использовать существительные во множественном числе.*

Объявить массив *marks*

int[] marks;

Определить массив *marks* – выделить память для 5 элементов:

marks = new int[5];

Объявить целочисленный массив *factors* и выделить память для хранения 20 его элементов:

int[] factors = new int[20];

Объявить целочисленный массив *numbers*, выделить память для хранения 6-ти его элементов (размер определяется согласно количеству элементов, указанных в списке инициализации), проинициализировав их начальными значениями, указанными в списке:

int[] numbers = {11, 22, 33, 44, 55, 66};

Чтобы создать массив необходимо знать его длину (или размер) заранее и выделять память в соответствии с размером. Длина массива фиксируется при создании и не может быть изменена в ходе выполнения программы. Когда массив объявлен, но память под элементы не выделена, он содержит значение `null`. При выделении памяти под элементы массива через оператор `new` по умолчанию все они инициализируются значениями по умолчанию: `0` для `int`, `0.0` для `double`, `false` для `boolean`, `null` для объектов. (В отличии от C/C++, где не инициализируется содержимое массива)

Сослаться на элемент массива можно через индекс (или подстроку), заключенный в `[ ]`. Индексация массивов в Java начинается с `0`.

Присвоить значения элементам массива *marks* можно:

marks[0] = 95;

marks[1] = 85;

Вывод значений элементов массива *marks*:

System.out.println(marks[0]);

System.out.println(marks[3] + marks[4]);

В Java, длина массива содержится в ассоциативной переменной *length* и может быть получена, с использованием команды *arrayName*`.length`. Вывести

на экран длину массива *marks* можно:

System.out.println("Длина массива - "+marks.length+" элементов");

В отличии от языков C/C++, в Java во время исполнения программы выполняется проверка выхода за границы массива. Другими словами, для каждой ссылки на элемент массива, индекс проверяется на соответствие длине массива.

Если индекс находится за пределами диапозона `[0, arrayName.legnth-1]`, JVM генерирует исключение `ArrayIndexOutOfBoundException`. Данного рода проверка несколько замедляет скорость выполнения программы, однако весьма полезна с точки зрения программной инженерии, чтобы избежать ошибки.

Для обработки всех элементов массива используют циклы.

Пример. Найти среднее арифметическое элементов массива *marks*

```
public class MeanStdArray {
   public static void main(String[] args) {
     int[] marks = {74, 43, 58, 60, 90, 64, 70};
     int sum = 0;
     int count = marks.length;
     for (int i=0; i<count; ++i) {
        sum += marks[i];
     }
     mean = (double)sum/count;
     System.out.printf("Mean is %.2f%n", mean);
   }
}
```

Начиная с версии JDK 1.5 появился оператор цикла для обработки элементов массива *for-each, который имеет следующий синтаксис:*

for (*type item* : *anArray*) {
  ***body* ;**
}

где *type* – тип элементов массива, *anArray* – название массива.

Пример. Найти сумму элементов массива с использованием *for-each*

```
int[] marks = {8, 2, 6, 4, 3};
int sum = 0;
for (int mark : marks) { // for each int number in int[] numbers
   sum += mark;
}
System.out.println("The sum is " + sum);
```

С помощью данного цикла можно только прочитать значение элемента массива, но нельзя его изменить.

Многомерные массивы в Java

Как и в языке C, многомерные массивы в Java рассматриваются как массив, элементами которого являются массивы, причем каждый из элементов-массивов может иметь разную длину.

Объявить и выделить память под элементы двумерного массива можно:

int grid[][] = new int[3][4];

Объявление с инициализацией:

```
int[][] grid = {
    {1, 2},
    {3, 4, 5},
    {6, 7, 8, 9}
};
```

Доступ к элементам двумерного массива осуществляется через два индекса:

```
grid[0][0] = 8;
grid[1][1] = 5;
```

Вывод длины:

```
System.out.println(grid.length);  // 3
System.out.println(grid[0].length);// 4
```

Пример. Программа по обработке двумерного массива

```
public class Array2DWithDifferentLength {
    public static void main(String[] args) {
        int[][] grid = {{1, 2},{3, 4, 5},{6, 7, 8, 9}};
        // Вывод массива grid
        for (int y = 0; y < grid.length; ++y) {
            for (int x = 0; x < grid[y].length; ++x) {
                System.out.printf("%2d", grid[y][x]);
            }
            System.out.println();
        }
        int[][] grid1 = new int[3][];
        grid1[0] = new int[2];
        grid1[1] = new int[3];
        grid1[2] = new int[4];
        // Вывод массива grid1
        for (int y = 0; y < grid1.length; ++y) {
            for (int x = 0; x < grid1[y].length; ++x) {
                System.out.printf("%2d", grid1[y][x]);
            }
            System.out.println();
        }
    }
}
```

Практические задания и методические указания

1. Изучить теоретические сведения по теме «Одномерные массивы», выполнить приведенные там примеры.

2. Выполнить индивидуальное задание № 1.

3. *Модифицировать программный код п.2 индивидуального задания: добавить возможность ввода данных из текстового файла

4. Изучить теоретические сведения по теме «Многомерные массивы», выполнить приведенные там примеры.

5. Создать отчет, включив в него описание работы по пункту 2 – 4 ответы на контрольные вопросы.

Индивидуальные задания

Вариант	Задание
1	1.Дан одномерный массив, который содержит не более 50 вещественных чисел. Найти сумму элементов, которые расположены на местах с нечетными номерами
	2.Даны оценки, полученные на экзамене по информатике студентами одной группы, по 10 бальной системе. Определить сколько студентов получили 10 баллов, сколько – 9 баллов, сколько – 8 баллов, сколько – 7 баллов, сколько – 6 баллов, сколько – 5 баллов, сколько – 4 балла, сколько – 3 балла. При выводе ответа слово «студент» должно стоять в соответствующей форме.
2	1.Дан одномерный массив, который содержит не более 50 целых чисел. Найти сумму элементов, которые делятся на данное число X.
	2.Имеется 5 магазинов с разными названиями: ЦУМ, ГУМ, Немига, Восточный, Первомайский. В магазинах имеется один и тот же товар, который пользуется большим спросом. Количество товара в каждом магазине задать с помощью датчика случайных чисел. Необходимо определить, в какой магазин в первую очередь необходимо завести новую партию этого товара.
3	1.Дан одномерный массив, который содержит не более 50 целых чисел. Найти сумму отрицательных и сумму положительных элементов и сравнить их по модулю.
	2.Имеется 5 бензоколонок по разным адресам: ул.Володарского, ул.Партизанская, ул.Комсомольская, ул.Ленина, ул.Рокоссовского. На этих бензоколонках имеется бензин марки 95 в разных количествах. Количество бензина на бензоколонках задать с помощью датчика случайных чисел. Необходимо заправить колонну машин, которая потребит 500 литров этого бензина. По какому адресу расположена бензоколонка, на которую можно отправлять данную колонну машин?
4	1.Дан одномерный массив, который содержит не более 50 целых чисел. Найти и вывести те элементы, которые больше предыдущего.
	2.Есть группа спортсменов из n<=20 человек. Для каждого спортсмена приводится его рост и вес. Вес спортсмена считается нормальным, если от роста отнять 100 и полученное число отличается от веса не более чем на 3. Вывести номера тех спортсменов, чей вес превышает норму.
5	1.Дан одномерный массив, который содержит не более 50 целых чисел. Заменить в массиве каждый элемент с четным номером цифрой 2, а с нечетным номером цифрой 5.
	2.Приводятся показатели производства работы n<=30 рабочих. Определить номер рабочего, у которого показатель наибольший, и номер рабочего, у которого показатель второй по величине.
6	1.Дан одномерный массив, который не более 50 целых чисел. Заменить каждый элемент массива произведением индексов соседних элементов, если нет соседних элементов оставить число без изменения. 2.Приводится рост учеников одного класса, где количество учеников n<=30. Определить номер ученика самого высокого и номер ученика

Вариант	Задание
	второго по росту в классе.
7	1.Дан одномерный массив, который содержит не более 50 целых чисел. Заменить каждый элемент массива суммой соседних элементов. Если соседних элементов нет, то число оставить без изменения. 2.Приводится среднесуточная температура воздуха за месяц. Вывести номера тех дней, когда среднемесячная температура была ниже среднесуточной температуры. Количество дней в месяце определить по названию месяца.
8	1.Дан одномерный массив, который содержит не более 50 целых чисел. Заменить каждый элемент массива суммой соседних индексов. Если соседних элементов нет, то число оставить без изменения. 2.В университете n<=10 факультетов. Известен план приема студентов на каждый факультет и число поданных заявлений. Определить конкурс на каждый факультет.
9	1.Дан одномерный массив, который содержит не более 50 целых чисел. Вывести только те элементы, которые меньше предыдущего. 2.Дано длинное целое число, содержащее не более 100 цифр. Рассматривать цифры числа, как массив символов. Определить, есть ли в данном числе цифра 0, и если есть, то сколько раз она попадается в данном числе.
10	1.Дан одномерный массив, который содержит не более 50 целых чисел. Найти среднее арифметическое элементов, которые стоят на нечетных местах. 2.Дано длинное целое число, содержащее не более 100 цифр. Рассматривать цифры числа, как массив символов. Определить, есть ли в данном числе четные цифры и сколько их.

Контрольные вопросы

1. Раскройте суть понятия массив в языке Java.

2. С какого значения начинается индексация массива в языке Java.

3. Имеется ли в языке Java возможность контроля выхода за границы массива.

4. Приведите синтаксис объявления одномерного массива в языке Java.

5. Какие разновидности операторов циклов используются для перебора элементов массива в языке Java?

6. Как интерпретируется двумерный массив в языке Java.

7. Приведите синтаксис объявления и доступа к элементам много мерного массива в языке Java.

8. Назовите особенности работы с массивами в языке Java по сравнения с другими языками программирования

Лабораторная работа №5
Работа со строками в Java

Цель: сформировать знания об особенностях обработки строковых данных в языке Java, а также о методах классах java.lang.String по обработке строковых данных; сформировать навыки решения задач по обработке символьных строк на языке Java.

Аппаратное, программное обеспечение: персональный компьютер, JDK, текстовый редактор (TextPad или NotePad++).

Краткие теоретические сведения

Работа сос строками в Java

Строка в Java – это объект класса *String*, входящего в состав пакета *java.lang*, представляет собой неизменную последовательность Unicode-символов. *String* имеет свои особенности по сравнению с другими классами. Строковое значение не может быть изменено в ходе выполнения программы. Методы данного объекта возвращают новое строковое значение. Например, метод *toUpperCase()* создаст и вернет новую строку, содержащую преобразованное к верхнему регистру значение, но исходную строку не изменит. В Java есть специальные классы *StringBuffer* и *StringBuilder*, которые допускают изменения в строке. Оператор + для данного класса перегружен и работает по следующему принципу, если два операнда типа `String`s, то выполняется контактенация строк, если один из операндов имеет числовое значение, то он преобразуется с типу `String`s и также выполняется контактенация. Строковые значения можно заключаются в двойные кавычки, создавая так называемый строковый литерал, и непосредственно присваиваются строковой переменной без вызова конструктора и создания экземпляра класса.

Создать строку можно:

String str1 = "Hello";

String str2 = new String("Hello");

Таблица 2.3.1 – Методы класса *String* по обработке строк

Метод	Назначение
int length()	возвращает длину строки
boolean isEmpty()	определяет, является ли строка пустой
boolean equals(String S)	выполняет сравнение текущей строки со строкой S, переданной в качестве аргумента. Операции «==» и «!=» к строкам в Java не применяются.
int indexOf(String S)	ищет подстроку *S* в строке
int indexOf(String S, int i)	ищет подстроку *S* в строке, начиная с индекса *i*
char charAt(int i)	возвращает символ, находящийся под индексом *i*
String substring(int i1, int i2)	возвращает подстроку с индекса *i1* по *i2*
String toLowerCase()	возвращает строку, содержащую значение

	текущей строки, приведенное к нижнему регистру
String toUpperCase()	возвращает строку, содержащую значение текущей строки, приведенное к верхнему регистру
String replace(char A, char B)	возвращает строку, в которой символы *A* заменены на *B*
String concat(String S)	возвращает строку, в которой к текущей строке добавлена строка *S*
static String ValueOf(type a)	преобразует значения примитивных типов в строку

Методы, использующие регулярные выражения:
boolean matches(String regexe)
String replaceAll(String regexe, String replacement)
String replaceAll(String regexe, String replacement)
String[] split(String regexe)
String[] split(String regexe, int count)

Пример. Программа по обработке строк

```
public class TestString {   // Save as "Hello.java" under "d:\myProject"
  public static void main(String[] args) {
    //объявление строк
    String firstStr="Тестовый текст";
    String secondStr= new String("Тестовый текст");
    //Вычисление длины строки
    System.out.println("Длина строки firstStr "+firstStr.length());
    //поиск подстроки в строке
    String findText="текст";
    System.out.println("В строке firstStr"+
        ((firstStr.indexOf(findText)>0)?" есть ":" отсутствует ")+
        " слово "+findText);
    //замена символов в строке
    String newStr=firstStr.replace(" ","+");
    System.out.println("Новая строка:"+newStr);
    //сравнение строк
    System.out.print("Строки firstStr и secondStr ");
    if(firstStr.equals(secondStr))System.out.println("совпадают");
       else System.out.println("совпадают");
    //сколько раз встречается заданный символ в строке
    int count=0;
    for(int i=0;i<firstStr.length();i++){
       if(firstStr.charAt(i)=='e') count++;
    }
    System.out.println("Символ е встречается в firstStr "+count+" раз.");
  }
}
```

Работа с датами в Java

Для работы с датами в Java можно использовать класс *Calendar*. *Calendar*

является абстрактным классом, т.е. нельзя создать его экземпляр с помощью конструктора. Вместо этого используется статический метод *Calendar.getInstance()*, который возвращает экземпляр класса Calendar, на основе текущего времени.

Пример. Вывод текущго времени

```
import java.util.Calendar;
public class GetYMDHMS {
   public static void main(String[] args) {
      Calendar cal = Calendar.getInstance();
      // You cannot use Date class to extract individual Date fields
      int year = cal.get(Calendar.YEAR);
      int month = cal.get(Calendar.MONTH); // 0 to 11
      int day = cal.get(Calendar.DAY_OF_MONTH);
      int hour = cal.get(Calendar.HOUR_OF_DAY);
      int minute = cal.get(Calendar.MINUTE);
      int second = cal.get(Calendar.SECOND);
      System.out.printf("Сейчас %4d/%02d/%02d %02d:%02d:%02d\n",
          year, month+1, day, hour, minute, second);}
}
```

Практические задания и методические указания

1. Изучить теоретические сведения. Разобрать и выполнить пример.
2. Выполнить индивидуальные задания согласно варианту.
3. Добавить к программному коду, разработанному в ходе выполнения п.2 вывод информации об авторе и дате запуска программы в следующему виде:

Автор:XXXX

Дата: YY-MM-DD HH:MM:SS

4. Создать отчет, включив в него описание работы по пункту 2, ответы на контрольные вопросы.

Индивидуальные задания

1 Дана строка, которая содержит не более 60 английских букв, среди которых есть одинаковые. Посчитать сколько раз встречается заданный символ.

2 Дана строка, которая содержит не более 60 английских букв, цифр, знаков препинания. Получить новый массив, в котором данная буква заменена на символ «_».

3 Дана строка, которая содержит не более 60 английских букв, цифр, знаков препинания. Посчитать сколько раз встречается символ «_»

4 Дана строка, которая содержит не более 60 английских букв, цифр, знаков препинания. Посчитать сколько символов в первом слове.

5 Дана строка, которая содержит не более 60 английских букв, цифр, знаков препинания. Эти символы образуют некоторые слова, разделенные одним пробелом. Вывести каждое слово отдельно (т.е. вывести слова в столбик).

6 Дана строка, которая содержит не более 60 английских букв, цифр, знаков препинания. Символы образуют слова, которые друг от друга отделяются пробелом. Заменить в словах букву «С» на букву «А».

7 Дана строка, которая содержит не более 60 английских букв, цифр, знаков препинания., которые образуют слова, разделенные одним пробелом. Ответить на вопрос, сколько слов.

8 Дана строка, которая содержит не более 60 английских букв, цифр, знаков препинания. Ответить на вопрос, есть ли среди данных символов сочетание «…».

9 Дана строка, которая содержит не более 60 английских букв, цифр, знаков препинания. Сколько раз в данном массиве встречается сочетание символов «xx».

10 Дана строка, которая содержит не более 60 английских букв, цифр, знаков препинания. Ответить на вопрос, есть ли среди этих символов подряд идущие одинаковые символы.

Контрольные вопросы

1. Как интерпретируется строка в языке Java?
2. Назовите возможные варианты создания строки в программе на языке Java.
3. Как работает операция +, если она применяется к строковым операндам.
4. Назовите методы для работы со строками в языке Java.

Лабораторная работа №6
Объявление и использование классов

Цель: сформировать умения разработки программ с применением пользовательских классов.

Аппаратное, программное обеспечение: персональный компьютер, JDK, текстовый редактор (TextPad или NotePad++).

Краткие теоретические сведения

Синтаксис определения класса в языке Java:

[*AccessControlModifier*] class *ClassName* {

// компоненты класса: переменные и методы

}

Соглашения по именованию. Имя переменной – существительное, обозначает атрибут, в то время как имя метода – глагол – обозначает действие. Имя класса – существительное, начинающееся с символа в верхнем регистре.

Пример. Описание класса «Круг»

```
public class Circle {// описывает объект «круг»
    double radius;// радиус
    String color;// цвет
    double getRadius() {//возвращает значение переменной радиус
    }
    double getArea(){//вычисляет площадь круга
    ...}
}
```

Класс имеет название *Circle*. Он содержит две *private* переменные-члена: *radius* типа *double* и *color* типа *String*; и два *public* метода-члена: *getRadius(), getColor()*.

Чтобы создать экземпляр класса необходимо **объявить *идентификатор (имя) экземпляра* и определить его – выделить память с помощью оператора *new*.**

имя_экземпляра = new имя_класса;

Пример. Создадим экземпляры класса `Circle`:

```
//Объявить 3 переменные типа Circle, c1, c2, и c3
c1 = new Circle();
c2 = new Circle(2.0);
c3 = new Circle(3.0, "red");
//Объявить и определить экземпляр класса Circle с именем c4:
Circle c4 = new Circle();
```

Если экземпляр объявлен, но не определен, то он содержит значение `null`.

Для доступа к переменным и методам класса используется оператор «.»:

имя_экземплара_класса.имя_переменной;

имя_экземплара_класса.имя_метода();

Пример:

```
//вызвать метод getArea() у экземпляра c1
System.out.println(c1.getArea());
```

Синтаксис объявления переменных класса в Java:

[AccessControlModifier] type variableName [= initialValue];
[AccessControlModifier] type variableName-1 [= initialValue-1]
[, type variableName-2 [= initialValue-2]] ... ;

Пример:

private double radius;
public int length = 1, width = 1;

В отличии от языка C++ модификатор доступа необходимо писать перед каждым компонентам класса. Хорошим стилем ООП считается, когда метод получает параметры (аргументы), выполняет операции, определенные в теле метода, возвращает результат (или пусто).

Синтаксис определения метода в Java:

[AccessControlModifier] returnType methodName ([parameterList]) {
// тело метода
}

В качестве примера создадим метод, который вычисляет площадь круга:

```
public double getArea() {
   return radius * radius * Math.PI;
}
```

В отличии от C++ в Java методы описываются внутри класса вместе с объявлением и не считаются встроенными.

Java-приложение, как правило, представляет собой композицию классов. Как правило только один класс является запускаемый (содержит метод main()). Остальные классы предназначены для использования другими классами. Программный код каждого класса должен храниться в отдельном файле с именем, совпадающим с именем класса.

Пример. Разработаем Java-приложение для работы с объектом «круг»

Исходный код для класса `Circle`:

```
public class Circle { // описывает объект «круг»
   private double radius;//рацус
   private String color;//цвет
   // Конструкторы
   public Circle() {  // по умолчанию
      radius = 1.0;
      color = "red";
   }
   public Circle(double r) {
      radius = r;
      color = "red";
   }
   public Circle(double r, String c) {
      radius = r;
      color = c;
   }
   public double getRadius() {
      return radius;
   }
```

```
    public String getColor() {
        return color;
    }
    public double getArea() {//возвращает площадь круга
        return radius * radius * Math.PI;
    }
}
```

Данный код необходимо сохранить в файл с расширением `Circle.java` и скомпилировать в `Circle.class`. Обратите внимание, что класс `Circle` не имеет метода `main()`. Следовательно, он не является автономной программой, которую можно запустить саму по себе, а предназначен, чтобы его использовать встроенным блоком в другие программы.

Напишем другой класс `TestCircle`, который будет использовать класс `Circle`, а точнее тестировать его работоспособность. `TestCircle` класс имеет метод `main()` и может быть запущен на выполнение.

Пример. Java-приложение для работы с объектом «круг»

Класс `TestCircle`, который будет использовать класс `Circle`

```
public class TestCircle { // тестирует работу класса Circle
    public static void main(String[] args) {
        Circle c1 = new Circle(2.0, "blue");
        System.out.println("Радиус: " + c1.getRadius());
        System.out.println("Цвет: " + c1.getColor());
        System.out.printf("Площадь: %.2f%n", c1.getArea());
        Circle c2 = new Circle(2.0);  // используем 2-ой конструктор
        System.out.println("Радиус:: " + c2.getRadius());
        System.out.println("Цвет:: " + c2.getColor());
        System.out.printf("Площадь: %.2f%n", c2.getArea());
        Circle c3 = new Circle();  // используем 1-ый конструктор
        System.out.println("Радиус: " + c3.getRadius());
        System.out.println("Цвет:: " + c3.getColor());
        System.out.printf("Площадь: %.2f%n", c3.getArea());
    }
}
```

Компилируем файл `TestCircle.java` в `TestCircle.class` и запускаем на выполнение. На экран будет выведено:

```
The radius is: 2.0
The color is: blue
The area is: 12.57
The radius is: 2.0
The color is: red
```

The area is: 12.57
The radius is: 1.0
The color is: red
The area is: 3.14

Конструкторы

Конструктор – это специальный метод, который используется для того чтобы определить экземпляр класса и проинициализировать переменные члены. Конструктор имеет тоже имя, что и класс. Конструктор, как и другие методы, может быть перегружен. (Перегрузка методов означает, что один и тот же метод может иметь различные варианты реализации (версии) для разного набора параметров.) В классе `Circle` определено три перегруженных версии конструктора. Чтобы создать экземпляр класса, необходимо использовать оператор *new*, за которым следует вызов одного из конструкторов в зависимости от указанных аргументов. Например:

Circle c1 = new Circle();
Circle c2 = new Circle(2.0);
Circle c3 = new Circle(3.0, "red");

Если список аргументов не соответствует ни одному из объявленных конструкторов, выдается ошибка компиляции. Конструктор не возвращает значение. В нем не допускается использование оператора return. Конструктор может быть вызван только через оператор *new* один раз, когда создается экземпляр класса. Конструкторы не наследуются. Конструктор без параметров называется *конструктор по умолчанию*. Он инициализирует члены переменные их значениями по умолчанию.

Модификатор доступа

Модификатор доступа (*access control modifier)* используется для управления видимостью как самого класса так и его переменных и методов. Для начала рассмотрим два модификатора:

1. `public`: Класс/переменная/метод доступна для всех других объектов системы.

2. `private`: Класс/переменная/метод доступна *только внутри класса.*

Например, переменная `radius` в классе `Circle` объявлена с модификатором `private`. В результате, переменная `radius` доступна только внутри `Circle`, но не доступна в классе `TestCircle`.

Метод `getRadius()` объявлена с модификатором `public`. Следовательно, он может быть вызван в классе `TestCircle`.

В Java в отличие от C++ по умолчанию используется модификатор *public*.

Getter и Setter методы

Принцип инкапсуляции в ООП требует, что переменные-члены класса должны быть скрыты от внешнего мира. Доступ к ним в языке Java принято обеспечивать через *getter*- и *setter*-методы. Переменная-член класса объявляется как *public*, только в случае наличия существенных на то причин.

Getter-методы объявляются как *public* и возвращают значения переменных, при этом они не должны их модифицировать. *Setter*-методы также объявляются как *public* и предназначены для установки значений переменным, при этом они

должны обеспечить валидацию данных, приведение их к требуемому внутреннему формату.

В классе `Circle` два getter-метода – *`getRadius()`* и *`getColor()`*. В классе *`TestCircle`* можно вызвать эти методы, чтобы извлечь значения переменных *`radius`* и *`color`* конкретного экземпляра класса *`Circle`*, например, *`c1.getRadius()`* и *`c1.getColor()`*.

Пример. Чтобы изменить значения переменных конкретного экземпляра класса `Circle`, необходимо добавить в него методы:

```
public void setColor(String newColor) {//присваивает значение цвета
  color = newColor;
}
public void setRadius(double newRadius) {//присваивает значение радиуса
  radius = newRadius;
}
```

Ключевое слово this

Иногда требуется, чтобы метод ссылался на вызвавший его объект. Для ссылки на текущий объект внутри любого его метода можно использовать ключевое слово *this*. Одним из направлений использования данного ключевого слова так же является устранение неоднозначности.

Пример:

```
public class Circle {
   double radius; // переменная-член radius
   public Circle(double radius) { // параметр метода radius
      this.radius = radius;
         // "this.radius" ссылается на переменную-член класса
         // "radius" имя аргумента метода.
   }...
}
```

В приведенном выше коде два идентификатора имеют имя *`radius`* – переменная-член и параметр метода. Это приводит к конфликту имен. Чтобы избежать конфликта имен можно использовать другое имя для переменной члена, например *r*. Однако, *`radius`* более близкое и значимое в данном контексте. Ключевое слово *`this`* обеспечит разрешение конфликта имен: *`this.radius`* ссылается на переменную-член, в том время как *`radius`* – имя параметра метода.

Пример. Использование *`this`* в конструкторах, *getter* и *setter* методах для *`private`*-переменных

```
public class ClassName {
   // private-переменная с именем varName типа T
   private T varName;
   // Конструктор
   public ClassName (T varName) {
      this. varName = varName;
   }
   //getter-метод, возвращает значение переменной varName
   public T getVarName () {
```

```
        return varName;  // или return this.varName для читабельности
    }
    /*setter-метод для установки значения переменной varName, принимает
    параметр типа T возвращает void*/
    public void setVarName (T varName) {
        this.xxx = varName;
    }
}
```

Для переменной *varName* типа *boolean* getter-метод должен быть назван `isVarName()` или `hasVarName()`, что более значимо, чем идентификатор `getVarName()`.

```
private boolean varName;
public boolean isVarName () {
  return varNam;  // или this.varNam для читабельности
}
```

Таким образом, `this.varName` ссылается на переменную `varName` текущего экземпляра класса; `this.methodName(...)` вызывает `methodName(...)` текущего экземпляра класса. В конструкторе можно использовать *this(...)*, чтобы вызвать другой конструктор этого класса. Внутри метода можно использовать оператор `return this` чтобы вернуть текущий экземпляр класса в точку вызова.

Метод toString()

Правилом хорошего тона считается, когда каждый класс в Java (аналогично в PHP) имеет метод `toString()`, который возвращает в виде строки описание экземпляра класса. Вызвать его можно либо с помощью команды *anInstanceName*`.toString()`, либо через `println()` или как операнд операции `+` для строк.

Синтаксис объявления метода *toString*:

```
public String toString() { ..... }
```

Пример:

Включим в класс `Circle` метод `toString()`

```
public String toString() {//возвращает описание экземпляра класса
  return "Circle[radius=" + radius + ",color=" + color + "]";
}
```

В классе *TestCircle* получим описание экземпляра *c1* класса *Circle*

```
Circle c1 = new Circle();
System.out.println(c1.toString());  // явный вызов toString()
System.out.println(c1);  // неявный вызов c1.toString()
System.out.println("c1 is: " + c1);  /* + вызывает toString(), чтобы преобразовать c1
      в строку для выполнения контактенации*/
```

Ключевое слово final

В процессе выполнения программы:

– переменной примитивного типа, объявленной с ключевым словом *final*, не может быть присвоено новое значение;

– экземпляру класса, объявленному с ключевым словом *final*, не может

быть присвоен новый объект;

– класс, определенный с ключевым словом *final*, не может быть унаследован;

– метод, определенный с ключевым словом *final*, не может быть переопределен.

Пример. Программный код класса *Circle* с использованием *final*

```
public class Circle { // реализует объект круг
    // public-константы
    public static final double DEFAULT_RADIUS = 8.8;
    public static final String DEFAULT_COLOR  = "red";
    // private переменные
    private double radius;
    private String color;
    // Конструкторы (overloaded)
    public Circle() {  // конструктор по умолчанию
        this.radius = DEFAULT_RADIUS;
        this.color  = DEFAULT_COLOR;
    }
    public Circle(double radius) {
        this.radius = radius;
        this.color = DEFAULT_COLOR;
    }
    public Circle(double radius, String color) {
        this.radius = radius;
        this.color = color;
    }
```

```
    // public getter и setter методы
    public double getRadius() {
       return this.radius;
    }
    public void setRadius(double radius) {
       this.radius = radius;
    }
    public String getColor() {
       return this.color;
    }
    public void setColor(String color) {
       this.color = color;
    }
    // Метод toString()
    public String toString() {///возвращает строку-описание экземпляра класса
       return "Circle[radius=" + radius + ", color=" + color + "]";
    }
    public double getArea() {/// возвращает площадь круга
       return radius * radius * Math.PI;
    }
    public double getCircumference() {//возвращает длину окружности
       return 2.0 * radius * Math.PI;
    }
 }
```

Статические свойства и методы класса

Каждый экземпляр класса поддерживает свое собственное хранилище данных. В результате каждые свойство или метод имеют свою собственную копию для конкретного экземпляра класса, созданного в программе.

Свойства и методы могут иметь идентификатор "static". Такие свойства и методы принадлежат классу, но являются общими для всех экземпляров данного класса. Они размещаются в общей для всех экземпляров класса области памяти и совместно используются всеми экземплярами класса. JVM размещает в памяти static-переменные во время загрузки класса. Статические переменные существуют, даже если экземпляр класса не создан, а также независимо от количества экземпляров классов. Синтаксис обращения к ним такой же как и к нестатическим, их можно вызывать из любых экземпляров класса, хотя такой подход не рекомендуется. На UML-диаграммах статические переменные подчеркиваются.

Т.к. нестатические переменные принадлежат конкретному экземпляру класса, то для того чтобы их использовать необходимо построить с помощью конструктора экземпляр класса. Статические переменные являются глобальными по своей природе. Чтобы их использовать, необязательно создавать экземпляр класса с помощью конструктора.

Одной из областей применения статических переменных является использование глобальных переменных, которые доступны всем экземплярам класса.

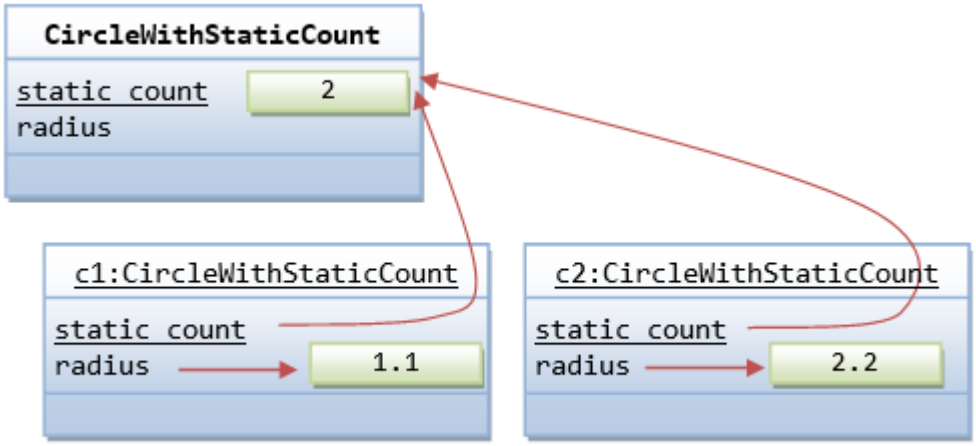
Пример. Создадим класс, в котором будет статическое свойство – счетчик экземпляров данного класса в программе

```
public class CircleWithStaticCount {
public static int count = 0; /*Статическая переменная для подсчета количества
    созданных экземпляров. Устанавливаем модификатор доступа public */
  private double radius;
  publicCircleWithStaticCount(double radius) {
    this.radius = radius;
    ++count; // one more instance created
  }
}
```

Протестируем класс *CircleWithStaticCount*

Пример:

```
public class TestCircleWithStaticCount {
  public static void main(String[] args) {
    CircleWithStaticCount c1 = new CircleWithStaticCount(1.1);
    System.out.println(c1.count);  // 1
System.out.println(CircleWithStaticCount.count); /*Можно также получить доступ
    к статической переменной через имя класса*/
    CircleWithStaticCount c2 = new CircleWithStaticCount(2.2);
    System.out.println(CircleWithStaticCount.count); // 2
    System.out.println(c1.count);  // 2
    System.out.println(c2.count);  // 2
  }
}
```



Другая область применения модификатора *static* – обеспечить глобальные переменные и методы, которые доступны другим классам без создания экземпляра статического класса. Например, класс *java.lang.Math* содержит чисто статические переменные и методы. Поэтому, чтобы использовать математические методы и константы не нужно создавать экземпляр класса, а можно вызывать их напрямую через имя класса: Math.PI, Math.E, Math.random(), Math.sqrt().

Статические методы могут иметь доступ только к статическим переменным и методам. В то время как нестатические методы могут иметь доступ как к статическим, так и нестатическим переменным и методам.

Пример:

```
public class Hello {
  private static String msgStatic = "Hello from static";
  private String msgInstance = "Hello from non-static";
  public static void main(String[] args) {
```

```
        System.out.println(msgStatic);  // компиляция проходит успешно
        //System.out.println(msgInstance);
        /*Ошибка компиляции: нестатическая переменная не может ссылаться
    на статический контекст*/
      }
    }
```

Наследование

В ООП часто выстраиваются иерархии классов, для того чтобы избежать дублирования и уменьшить избыточность кода. Подкласс наследует все переменные и методы (кроме конструктора) своего суперкласса, включая непосредственных родителей и всех предков.

В Java для определения подкласса используется ключевое слово `extends`. Синтаксис:

class Cylinder ***extends*** *Circle {.....}*

Рассмотрим пример. В этом примере будет создан производный подкласс `Cylinder` от суперкласса `Circle`, созданного ранее. Важно заметить, что класс `Circle` используется повторно. Класс `Cylinder` наследует все переменные-члены (`radius` и `color`), а также методы (`getRadius()`, `getArea()` и др.) от своего суперкласса `Circle`. Кроме того, в нем определены переменная `height`, его собственные конструкторы и два public-метода.

Пример. Программный код класса *Cylinder*

```
public class Cylinder extends Circle {//описывает объект цилиндр
  private double height;//высота
  // конструкторы
  public Cylinder() {
    super(); // вызов конструктора родительского класса Circle()
    this.height = 1.0;
  }
  public Cylinder(double height) {
    super();
    this.height = height;
  }
  public Cylinder(double height, double radius) {
    super(radius); //вызов конструктора родительского класса Circle(radius)
    this.height = height;
```

```
  }
  public Cylinder(double height, double radius, String color) {
    super(radius, color); /*вызов конструктора родительского класса Circle(radius,
          color)*/
    this.height = height;
  }
  // Getter и Setter методы
  public double getHeight() {
    return this.height;
  }
  public void setHeight(double height) {
    this.height = height;
  }
  public double getVolume() {//вычисляет объем цилиндра
    return getArea()*height; //использует метод getArea() суперкласса Circle
  }
  public String toString(){
    return "This is a Cylinder";
  }
}
```

Данный программный код необходимо сохранить в файле «Cylinder.java» и скомпилировать.

Пример. Программный код для тестирования класса *Cylinder*

```
public class TestCylinder { // тестирует работу класса Cylinder
   public static void main(String[] args) {
      Cylinder cy1 = new Cylinder();
      System.out.println("Radius is " + cy1.getRadius()
         + " Height is " + cy1.getHeight()
         + " Color is " + cy1.getColor()
         + " Base area is " + cy1.getArea()
         + " Volume is " + cy1.getVolume());
      Cylinder cy2 = new Cylinder(5.0, 2.0);
      System.out.println("Radius is " + cy2.getRadius()
         + " Height is " + cy2.getHeight()
         + " Color is " + cy2.getColor()
         + " Base area is " + cy2.getArea()
         + " Volume is " + cy2.getVolume());
   }
      }
```

Файлы «Cylinder.java» и «TestCylinder.java» должны быть размещены в той же директории, что и файл «Circle.class», т.к. они используют класс *Circle*. Ожидаемый вывод на экран *TestCylinder*:

Radius is 1.0 Height is 1.0 Color is red Base area is 3.141592653589793 Volume is 3.141592653589793

Radius is 5.0 Height is 2.0 Color is red Base area is 78.53981633974483 Volume is 157.07963267948966

Подкласс может использовать унаследованные переменные и методы как свои собственные. В нем могут быть переопределены унаследованные методы, а

также скрыты унаследованные переменные путем определения переменных с таким же именем. Например, унаследованный метод *getArea()* у объекта *Cylinder* вычисляет площадь основания. Предположим, что решено переопределить метод *getArea()* в подклассе *Cylinder*, чтобы он вычислял площадь поверхности цилиндра.

Пример:

```
public class Cylinder extends Circle {
    // Переопределение унаследованного метода getArea()
    @Override
    public double getArea() {
        return 2*Math.PI*getRadius()*height + 2*super.getArea();
    }
    // Необходимо изменить getVolume()
    public double getVolume() {
        return super.getArea()*height; // используем getArea() из суперкласса
    }
    // Переопределяем унаследованный метод toString()
    @Override
    public String toString() {
        return "Cylinder[" + super.toString() + ",height=" + height + "]";
    }
}
```

@Override – аннотация, представленная JDK 1.5, которая указывает компилятору проверить будет ли данный метод переопределять метод в суперклассе. Это помогает избежать ошибок при переопределении методов. Данная директива не является обязательной, но рекомендуется ее использовать.

Если *getArea()* вызывается объектом класса *Circle*, то она вычисляет площадь круга. Если *getArea()* вызывается объектом класса *Cylinder*, то он вычисляет площадь поверхности цилиндра, используя переопределенную реализацию.

Несмотря на то что подкласс включает в себя все члены суперкласса, он не может иметь доступ к членам супер класса, объявленным как *private*. Для получения значения переменной *radius* необходимо использовать *public* метод *getRadius()*, т.к. переменная *radius* объявлена как *private* и поэтому недоступна в других классах, включая подкласс *Cylinder*.

Если будет переопределен метод *getArea()* в классе *Cylinder*, то метод *getVolume()* *(=getArea()*height)* перестанет работать, т.к. переопределенный метод, действующий в классе *Cylinder* не вычисляет площадь. Проблему можно решить, вызвав через super версию *getArea()* из родительского класса. Следует обратить внимание, что *super.getArea()* может быть вызвано только при определении подкласса, но не когда экземпляр уже создан, т.е. *c1.super.getArea();* - недопустимый оператор.

Ключевое слово *super* позволяет подклассу получить доступ к методам суперкласса и переменным в определении подкласса. Если подкласс

переопределяет метод, унаследованный от его суперкласса, например *getArea()*, то в определении подкласса можно использовать *super.getArea()* для вызова версии данного метода, реализованной в суперкласса. Аналогично, если подкласс перекрывает одну из переменных суперкласса, в определении подкласса можно использовать *super.variableName* для ссылки на скрытую переменную.

В теле конструктора можно использовать оператор `super(args);` чтобы вызвать конструктор своего непосредственного суперкласса, при чем данный оператор должна быть первым оператором тела конструктора подкласса. Если конструктор отсутствует, то компилятор Java автоматически подставляет оператор *super();* чтобы вызвать конструктор без параметров его непосредственного суперкласса. Если ни один из конструкторов не определен, то компилятор Java автоматически создает конструктор без параметров. Конструктор без параметров по умолчанию автоматически не сгенерируется, если в классе есть хоть один другой конструктор. В этом случае его необходимо явно определять. Если непосредственный суперкласс не имеет конструктора по умолчанию, то будет выдана ошибка компиляции при попытке выполнить команду `super()`;.

Java в отличии от C++ не поддерживает множественного наследования. В Java, каждый подкласс может иметь один и только один суперкласс (*single inheritance*).

Java использует так называемый подход с общим корнем (*common-root* approach). Все классы в Java производные от общего корневого класса `java.lang.Object`, который определяет и реализует общие требования к поведению для всех объектов Java, работающих под JRE, что позволяет реализовать такие функции, как многопоточность и сборщик мусора.

Абстрактные классы и интерфейсы.

Абстрактный метод – это метод, у которого отсутствует реализация. Для объявления такого метода используется ключевое слово **`abstract`**.

Например в классе `Shape` объявлены абстрактные методы `getArea()`, `draw()`:

```
abstract public class Shape {
  abstract public double getArea();
  abstract public double getPerimeter();
  abstract public void draw();
}
```

Реализация данных методов в этом классе невозможна, так как фактическая форма сферы пока неизвестна. Реализовываться абстрактные методы будут в подклассах, когда станет известна форма сферы. Т.к. абстрактный метод не имеет реализации, следовательно он не может быть вызван.

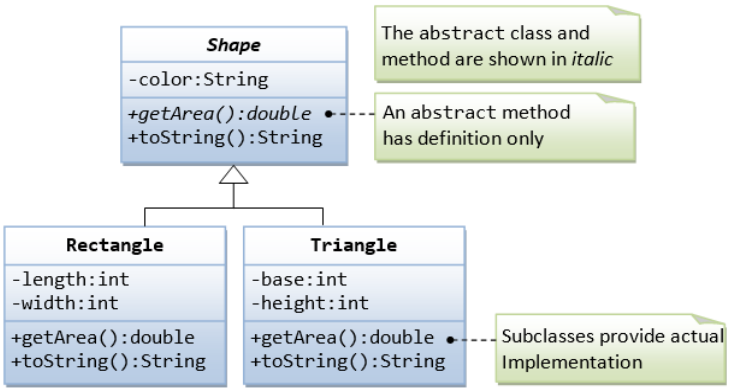


Рисунок 2.4.1 – Абстрактные классы и методы

Класс, содержащий один или более абстрактных методов, называется ***абстрактный класс***. Такой класс должен быть объявлен с модификатором `abstract`. Т.к. определение абстрактного класса не полно, не может быть создан его экземпляр.

Пример:

```
abstract public class Shape {// описывает объект сфера
  private String color;
  public Shape (String color) {//сфера
    this.color = color;}
  @Override
  public String toString() {
    return "Shape of color=\"" + color + "\"";}
  //Все подклассы класса Shape должны реализовывать метод getArea()
  abstract public double getArea();//абстрактный метод
}
```

Чтобы использовать абстрактный класс, нужно создать производный от него класс, в котором переопределить и добавить реализацию для всех абстрактных методов. Теперь подкласс будет полностью определен и можно создавать его экземпляры. (Если подкласс не обеспечивает реализацию всех абстрактных методов, он остается абстрактным.) Например, можем создать экземпляры классов `Triangle` и `Rectangle` и привести их к типу `Shape` (таким образом, чтобы программировать и управлять на уровне интерфейса), но не можем создать экземпляр класса `Shape`.

Пример:

```
public class TestShape {
  public static void main(String[] args) {
    Shape s1 = new Rectangle("red", 4, 5);
    System.out.println(s1);
    System.out.println("Area is " + s1.getArea());
    Shape s2 = new Triangle("blue", 4, 5);
    System.out.println(s2);
    System.out.println("Area is " + s2.getArea());
```

```
    // Нельзя создать экземпляр абстрактного класса
    Shape s3 = new Shape("green"); // Ошибка компиляции
  }
}
```

Таким образом абстрактный класс реализует шаблон для дальнейшей разработки. Его цель – обеспечить общий интерфейс для всех подклассов. Указывая сигнатуру абстрактных методов, мы обязываем все подклассы иметь данные методы с указанной сигнатурой. Подклассы должны обеспечить правильную реализацию методов.

Абстрактный метод не может быть определен к ключевым словом *final*, поскольку *final*-метод не может быть переопределен. С другой стороны, абстрактный метод должен быть переопределен в потомке до его использования. Абстрактный метод не может быть `private` (приведет к ошибке компиляции). Это связано с тем, что `private` метод не доступен подклассу и поэтому не может быть переопределен.

В Java ***интерфейс*** – это полностью абстрактный суперкласс, определяющий множество методов, которые подкласс должен поддерживать. Интерфейс содержит только `public` *abstract методы и, возможно,* `public static final` переменные. Чтобы определить класс как интерфейс используется ключевое слово `interface` вместо `class`. Ключевые слова `public` и `abstract` для абстрактных методов не требуются. Как и с абстрактными классами – нельзя создать экземпляр интерфейса. Необходимо создать подклассы, реализующие интерфейс и обеспечить реализацию абстрактных методов. В отличие от обычных классов вместо ключевого слова `extends` для описания производных классов используется `implements`. Интерфейс говорит, что классы должны делать, но не говорит, как это делать.

Модернизируем абстрактный суперкласс superclass `Shape` в интерфейс (рисунок 2.4.2).

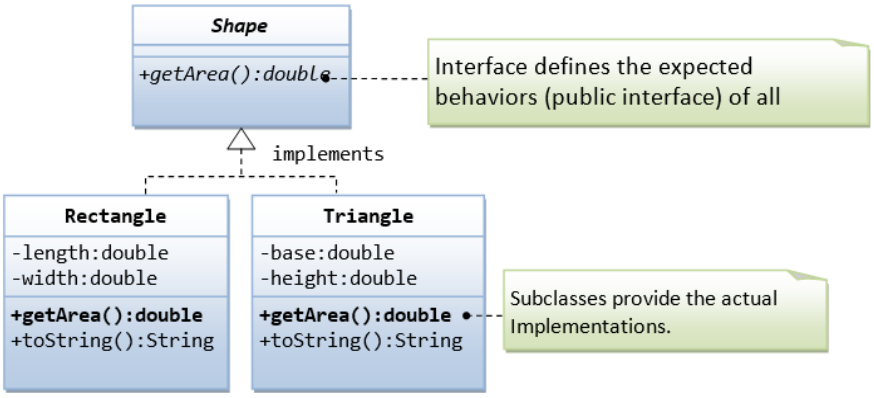


Рисунок 2.4.2 – Интерфейс Shape

Пример:

Программный код интерфейса `Shape`

```
public interface Shape {//используется ключевое слово interface вместо class
  /*список абстрактных public-методов, которые должны быть реализованы
  в подклассах */
  double getArea();
```

```
}
```

Программный код класса *Rectangle*, реализующего интерфейс *Shape*

```
public class Rectangle implements Shape {
  /* используется ключевое слово implements вместо extends*/
  private int length;//длина
  private int width;/ширина
  public Rectangle(int length, int width) {
    this.length = length;
    this.width = width;
  }
  @Override
  public String toString() {
    return "Rectangle[length=" + length + ",width=" + width + "]";
  }
  //Реализация абстрактного метода getArea(), определенного интерфейсом
  @Override
  public double getArea() {
    return length * width;
  }
}
```

Программный код класса *Triangle*, реализующего интерфейс *Shape*

```
public class Triangle implements Shape {
  private int base;//основание
  private int height;//высота
  public Triangle(int base, int height) {
    this.base = base;
    this.height = height;
  }
  @Override
  public String toString() {
    return "Triangle[base=" + base + ",height=" + height + "]";
  }
  //Реализация абстрактного метода getArea(), определенного интерфейсом
  @Override
  public double getArea() {
    return 0.5 * base * height;
  }
}
```

Программный код класса ***TestShape*, тестирующего работоспособность созданных классов и интерфейса**

```
public class TestShape {
  public static void main(String[] args) {
    Shape s1 = new Rectangle(1, 2);  // приведение к базовому типу
    System.out.println(s1);
    System.out.println("Area is " + s1.getArea());
    Shape s2 = new Triangle(3, 4);  // приведение к базовому типу
    System.out.println(s2);
    System.out.println("Area is " + s2.getArea());
    // Нельзя создать объект интерфейса
```

```
        Shape s3 = new Shape("green");  // Ошибка компиляции!!!
    }
}
```

Практические задания и методические указания

1. Изучить теоретические сведения по теме «Объявление и использование классов».

2. Выполнить индивидуальное задание № 1.

3. Изучить теоретические сведения по теме «Наследование»

4. Выполнить индивидуальное задание № 2.

5. Изучить теоретические сведения по теме «Абстрактные методы. Интерфейсы», доработать и выполнить описанные там примеры.

6. Создать отчет, включив в него описание работы по пунктам 2,4-5 ответы на контрольные вопросы.

Индивидуальные задания

Задание 1

1 Описать объект «прямоугольный треугольник» в виде класса «*rightTriangle*». Свойства: длины катетов. Методы: ввод, вывод значений свойств, вычисление гипотенузы, вычисление периметра. Использовать созданный класс для написания программы вычисления периметра по заданным длинам катетов.

2 Описать объект «прямоугольник» в виде класса «*rectangle*». Свойства: длины сторон. Методы: ввод, вывод значений свойств, вычисление площади. Использовать созданный класс для написания программы вычисления площади по заданным длинам сторон.

3 Описать объект «точка на плоскости» в виде класса «*point*». Свойства: абсцисса, ордината. Методы: ввод, вывод значений свойств, вычисление расстояния до начала координат. Использовать созданный класс для написания программы вычисления расстояния от точки до начала координат.

4 Описать объект «круг» в виде класса «*circle*». Свойства: радиус. Методы: ввод, вывод значений свойств, вычисление длины окружности. Использовать созданный класс для написания программы вычисления длины окружности по заданному значению радиуса.

5 Описать объект «куб» в виде класса «*cube*». Свойства: длина ребра. Методы: ввод, вывод значений свойств, вычисление площади поверхности. Использовать объекта для написания программы вычисления площади поверхности куба по заданной длине ребра.

6 Описать объект «товар» в виде класса «*product*». Свойства: шифр, цена за 1 килограмм, вес. Методы: ввод, вывод значений свойств, вычисление итоговой стоимости товара. Использовать созданный класс для написания программы вычисления стоимости товара.

7 Описать объект «студент» в виде класса «*student*». Свойства: номер зачетной книжки, оценки за экзамены по иностранному языку и программированию. Методы: ввод, вывод значений свойств, вычисление среднего балла за сессию. Использовать объекта для написания программы вычисления среднего балла студента.

8 Описать объект служащий в виде класса «*employee*». Свойства: количество отработанных часов за месяц, тарифная ставка. Методы: ввод, вывод значений свойств, вычисление зарплаты за месяц. Использовать созданный класс

для написания программы вычисления зарплаты за месяц.

9 Описать объект «автомобиль» в виде класса «*car*». Свойства: гос. номер, VIN, пробег. Методы: ввод, вывод значений свойств, увеличить пробег. Использовать созданный класс для написания программы, позволяющей пользователю добавлять и просматривать пробег автомобиля.

10 Описать объект телефонный номер в виде класса «*phoneNumber*». Свойства: номер, ФИО владельца, паспортные данные, тарифный план, баланс. Методы: ввод, вывод значений свойств, пополнение баланса, изменение личных данных. Использовать созданный класс для написания программы, позволяющей клиенту просмотреть баланс счета.

Задание 2 Выполнить объектно-ориентированное проектирование предметной области. Разработать программу на основании полученного проекта. Прежде, чем приступать к разработке программного кода, согласовать проект с преподавателем.

1 Учебное заведение осуществляет подготовку по двух направлениям: программисты и связисты. В конце обучения учащиеся сдают комплексный итоговый экзамен, состоящий из двух дисциплин. В качестве первой дисциплины все студенты сдают иностранный язык. Второй экзамен сдается по дисциплине специализации. Для программистов это программирование для связистов это теория электросвязи.

Написать программу для заполнения и вывода на экран экзаменационной ведомости. Предусмотреть возможность сортировки по среднему баллу.

2 На предприятии возможны два типа трудовых отношений: работа в штате и работа по договору подряда.

Зарплата сотрудника работающего в штате определяется размером оклада, надбавкой и размером премии.

Вознаграждение сотрудника работающего по договору подряда определяется размером тарифной ставки и количеством отработанных часов за месяц.

Для всех сотрудников организации из их суммы высчитываются два вида налогов: подоходный налог(12%) и налог в ФСЗН(1%).

Написать программу для расчета и вывода на экран заработной платы сотрудников.

3 Тестовые задания по программированию могут содержать два вида вопросов: выбор из n вариантов и с открытым ответом.

При использовании теста с открытым ответом студент вводит свой вариант ответа, и он должен совпасть с правильным вариантом ответа.

Написать программу для создания тестов и проведения тестирования студентов.

4 В файле находится текст. В первое строке текста находится кодовое слово, позволяющее определить, каким алгоритмом зашифрован текст. Текст может быть зашифрован одним из алгоритмов : «А1» и «А2».

Алгоритм А1. Два соседних символа меняются местами.

Алгоритм А2. Слова записаны в обратном порядке.

Написать программу позволяющую зашифровывать и расшифровывать тексты.

5 Интернет-магазин продает телефоны. Каждый товар имеет название, цену, дату поставки, производитель. Проводные телефоны характеризуются: наличием автоответчика, определителя номера. Мобильные телефоны характеризуются: стандарт связи, поддержка двух SIM-карт, платформа.

Написать программу для ведения каталога товаров, а также поиска товаров по следующим критериям: найти товары-новинки; найти товары по заданному ценовому диапазону.

6 В библиотеке хранятся книги и периодические издания: газеты и журналы. Книга характеризуется: название, авторы, издательство, год издания, кол-во страниц. Журналы и газеты характеризуется: название, год, номер, издательство.

Написать программу для ведения каталога библиотеки.

Названия выводить в следующем формате:

1. Куперштейн, В.И. Современные информационные технологии в делопроизводстве и управлении / В.И. Куперштейн. – Спб : БХВ, 2000. – 248.

2. Вести института современных знаний. 2009. -№2.

7 В ВУЗе учатся студенты, на последнем курсе они становятся студенты-дипломники. Студент характеризуется именем (указатель на строку), курсом и идентификационным номером. Студент-дипломник имеет тему дипломного проекта.

Написать программу для редактирования данных о студентах.

8 Имеются фигуры двух видов – круг и эллипс, каждая из которых определяется координатами центра.

Написать программу для вычисления площади заданных фигур.

9 На предприятии сотрудники могут работать в штате либо по договору подряда. Для штатных сотрудников заработная плата назначается исходя из оклада, размера надбавки (% от оклада) и премии, а также в случае отсутствия на работе (отпуск болезнь и др.) оклад выплачивается пропорционально отработанным дням. Для сотрудников, работающих по договору подряда заработная плата начисляется исходя из ставки за час и количества отработанных.

Написать программу для начисления заработной платы и формирования ведомости для выдачи заработной платы.

10 В магазине товары могут продаваться поштучно, а могут по весу. Для товаров продающихся поштучно магазин устанавливает цену за штуку, при заказе товара покупатель вводит количество необходимых ему единиц товара. Для товаров продающихся на развес магазин устанавливает цену за килограмм, при заказе товара покупатель указывает необходимый ему вес. На все товары магазин может устанавливать скидки (в %).

Написать программу для формирования ассортимента товаров в магазине, а также для формирования корзины заказа покупателем.

Контрольные вопросы

1. Дайте определения понятиям класс, переменная, метод.
2. Приведите синтаксис объявления классов, переменных, методов.
3. Приведите синтаксис создания экземпляров классов и вызова методов.

4. Назовите классификаторы доступа и приведите различия между ними.

5. Объясните назначение типовых методов в классах на Java (constructor, get-методы, set-методы, is-методы, toString).

6. Раскройте суть понятия наследование.

7. Приведите синтаксис описания производных классов в языке Java.

8. Объясните назначение ключевых слов *super*, *this*, *final*.

9. Объясните механизм переопределения методов.

10. Дайте определения понятиям «абстрактный класс» и «интерфейс».

Лабораторная работа №7
Работа с пакетом java.awt

Цель: Сформирования знания о средствах пакета java.awt; научить разрабатывать программы с использованием пакета java.awt.

Аппаратное, программное обеспечение: персональный компьютер, JDK, текстовый редактор (TextPad или NotePad++).

Краткие теоретические сведения

Пакет *java.awt* содержит набор классов для создания графического пользовательского интерфейса (GUI). В языке Java представлено два набора API для визуального программирования: *AWT* (Abstract Windowing Toolkit) и *Swing*. *AWT API* представлен в *JDK* 1.0. Большинство компонентов *AWT* устарели и были заменены новыми компонентами пакета *Swing*. *Swing API* был представлен как часть *Java Foundation Classes* (JFC) после выхода *JDK* 1.1. *JFC* состоит из *Swing*, *Java2D*, *Accessibility*, *Internationalization*, *Pluggable Look-and-Feel Support APIs*. Кроме *Swing Graphics API*, и *AWT*, представленных в JDK, существуют *Graphics API*, которые работают с Java, такие как *Eclipse's Standard Widget Toolkit* (SWT) (используются в *Eclipse*), *Google Web Toolkit* (*GWT*) (используются в *Android*), *3D Graphics API*.

Контейнеры и компоненты

Существуют два типа GUI элементов:

1. *Компоненты* являются элементарными объектами *GUI* (например, *Button*, *Label*, *TextField*.)

2. *Контейнеры* (например, *Frame*, *Panel*) используются для хранения компонентов в определенной компоновке (например, *flow*, *grid*). Контейнер может содержать подконтейнеры.

GUI компоненты также называются *элементы управления* (Microsoft ActiveX Control), *виджеты* (*Eclipse's Standard Widget Toolkit*, *Google Web Toolkit*), которые позволяют взаимодействовать (или управлять) с приложением через компоненты (например, нажатие на кнопку, ввод текста).

На рисунке 2.5.1 представлено три контейнера: один *Frame* и два *Panels*. *Frame* – контейнер высшего уровня AWT-программы. Он содержит заголовок (содержит иконку, название, *minimize/maximize/close buttons*), строку меню и область отображения содержимого. *Panel* - прямоугольная область, использующаяся для группировки связанных элементов графического интерфейса в определенной компоновке. Также на рисунке видим пять компонентов: *Label* (подпись), *TextField* (поле для ввода текста), и три *Buttons* (кнопка, по нажатию вызываются запрограммированные обработчики действий).

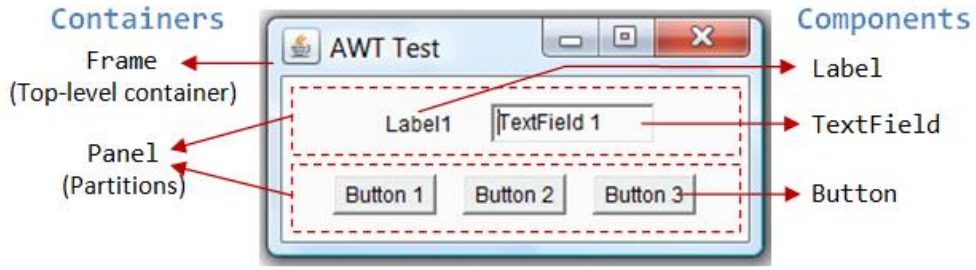


Рисунок 2.5.1 – Визуальные элементы AWT

В программе GUI компоненты должны храниться в контейнере. Поэтому сначала необходимо определить контейнер и компоненты. Далее с помощью

метода *add(Component c)* добавить компоненты в данный контейнер. Пример:

Panel panel = new Panel(); // определяем контейнер panel
Button btn = new Button("Press"); // определяем кнопку btn
panel.add(btn); // добавляем кнопку btn в контейнер panel

Контейнерные классы

Контейнеры верхнего уровня: Frame, Dialog и Applet. Каждая *GUI* программа в Java включает контейнер верхнего уровня. Наиболее используемые контейнеры верхнего уровня в AWT: *Frame*, *Dialog* и *Applet*:

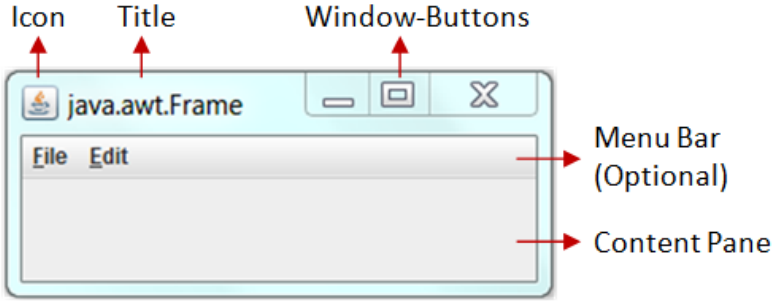


Рисунок 2.5.2 – Вид контейнера *Frame*

Класс *Frame* создает «главное окно» для GUI приложения. Чтобы написать GUI программу, начинают как правило с создания подкласса, производного от *java.awt.Frame*, чтобы унаследовать главное окно со всеми его свойствами: заглавие, иконка, кнопки, область отображения содержимого и т.д.

Пример:

```
import java.awt.Frame;  // используем класс Frame из package java.awt
public class MyGUIProgram extends Frame {
  // Конструктор для настройки GUI элементов
  public MyGUIProgram() { ..... }
  // другие методы
  public static void main(String[] args) {
     //вызов конструктора, для создания экземпляра окна
     new MyGUIProgram();
  }
}
```

AWT Dialog – «всплывающее окно», использующееся для взаимодействия с пользователем. *Dialog* включает заголовок (содержит иконку, название, кнопку закрыть) и область определения контента. Пример на рисунке 2.5.3.

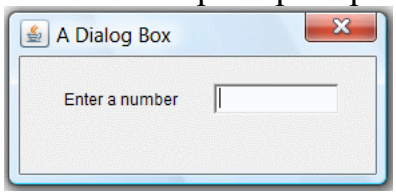


Рисунок 2.5.3 – Вид контейнера *Dialog*

AWT Applet (в пакете *java.applet*) контейнер верхнего уровня для аплетов (Java-программа, выполняется под управлением браузера.

Вторичные контейнеры: Panel and ScrollPane.

Вторичные контейнеры – контейнеры, которые размещаются внутри контейнеров верхнего уровня, либо внутри других вторичных контейнеров. В *AWT* представлены следующие вторичные контейнеры:

– *Panel*: прямоугольная область внутри контейнера верхнего уровня,

используемая для того чтобы разместить набор связанных GUI-компонентов по одному из шаблонов *grid* или *flow*.

– *ScrollPane*: обеспечивает автоматические горизонтальные и / или вертикальные прокрутки для одного дочернего элемента.

Иерархия **классов контейнера AWT** представлена на рисунке 2.5.4.

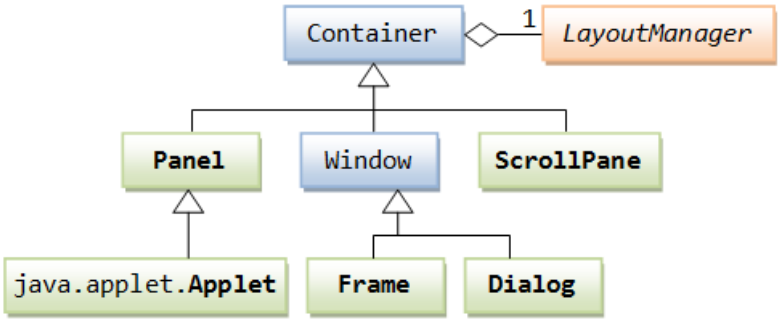


Рисунок 2.5.4 – Иерархия классов контейнера *AWT*

В AWT представлено множество готовых GUI-компонентов. Наиболее часто используются: *Button*, *TextField*, *Label*, *Checkbox*, *CheckboxGroup* (*radio buttons*), *List*, *Choice*. Их вид представлен на рисунке 2.5.5.

Рисунок 2.5.5 – Вид компонент AWT

Создание компонент и добавление их в контейнер

Чтобы создать и добавить в контейнер *GUI* компоненты необходимо пройти четыре шага:

1. Объявить компонент, задав ему идентификатор (имя);
2. Создать компонент, вызвав соответствующий конструктор с помощью оператора new;
3. Определить контейнер для хранения этого компонента.
4. Добавить компонент в контейнер с помощью оператора *aContainer.add(aComponent);*.

Класс ***java.awt.Label*** обеспечивает создание *текстового поясняющего сообщения*. Следует обратить внимание, что *System.out.println()* печатает в системную консоль, а не в окно. Поэтому необходимо использовать другие компоненты (текстовые поля) для вывода текстовые описаний. Спецификация *java.awt.Label*.

Конструкторы:

public Label(String strLabel, int alignment); – создает метку с заданным текстом (*text String*) и выравниванием (*alignment*). Для указания выравнивания в класс определены три константы: *Label.LEFT*, *Label.RIGHT*, *Label.CENTER*

public Label(String strLabel); – создает метку с заданным текстом (*text String*) и выравнивание по умолчанию по левому краю

public Label(); – создает метку с пустой строкой.

Public методы:

public String getText();

public void setText(String strLabel);

public int getAlignment();

public void setAlignment(int alignment);

Пример создания и добавления метки:

```
Label lblInput;    // Объявляем компонент Label с именем lblInput
lblInput = new Label("Enter ID");   // создаем компонент lblInput
add(lblInput);      //добавляем компонент в контейнер lblInput
// this.add(lblInput) - "this" is typically a subclass of Frame
lblInput.setText("Enter password"); // Устанавливаем текст
```

Можно создать метку без указания идентификатора, называемую анонимный экземпляр. В этом случае компилятор Java присвоит анонимный идентификатор для размещаемого объекта, в результате не будет возможности сослаться на анонимный экземпляр, после того как он будет создан. Это правило хорошо для меток, т.к. часто нет необходимости ссылаться на них после создания. Пример:

add(new Label("Enter Name: ", Label.RIGHT));

Класс ***java.awt.TextField*** создает *однострочное текстовое поле*, в которое пользователь может вводить текст. (Многострочное текстовое поле – *TextArea*.) Нажатие клавиши <ENTER> в *TextField* вызывает обработчик события. Спецификация класса:

Конструкторы:

public TextField(String strInitialText, int columns); – создает текстовое поле ввода с заданным по умолчанию текстом *strInitialText* и количеством символов *columns.*

public TextField(String strInitialText); – создает текстовое поле ввода с заданным по умолчанию текстом *strInitialText.*

public TextField(int columns); – создает текстовое поле ввода с заданным количеством символов *columns.*

Public-методы:

public String getText();

public void setText(String strText);

public void setEditable(boolean editable); – устанавливает поле как редактируемое (read/write) или нередактируемое (read-only).

Пример создания, добавления в контейнер текстового поля ввода, а также записи и чтения данных из него

```
/*объявляем и создаём текстовое поле ввода tfInput размером 30 символов*/
TextField tfInput = new TextField(30);
add(tfInput); // добавить в контейнер tfInput
/*объявляем и создаём поле ввода  tfResult*/
TextField tfResult = new TextField();
tfResult.setEditable(false) ; // сделать tfResult нередактируемым
add(tfResult); // добавить tfResult в контейнер
/*читаем данные из поля tfInput в переменную number*/
int number = Integer.parseInt(tfInput.getText());
number *= number;
/*записываем данные из переменной number в поле tfResult*/
```

```
tfResult.setText(number + "");
```

Следует обратить внимание, что методы *getText()/SetText()* оперируют с типом данных *String*. Конвертировать значение типа *String* в примитивные типы (*int, double*) можно через *static* методы *Integer.parseInt()* или *Double.parseDouble()*. Чтобы конвертировать примитивный тип в тип *String*, можно просто сложить (+) примитивный тип с пустой строкой.

Класс ***java.awt.Button*** позволяет создавать кнопки, при нажатии на которые вызываются обработчики действий. Спецификация класса:

Конструкторы:

public Button(String buttonLabel); – создает кнопку с заданной подписью.

public Button(); **–** создает кнопку без подписи

Public-методы:

public String getLabel();

public void setLabel(String buttonLabel);

public void setEnable(boolean enable); – включить или отключить эту кнопку. Отключенная кнопка не может быть нажата.

Пример создания и добавления кнопки

```
Button btnColor = new Button("Red"); // объявить и создать кнопку  btnColor
add(btnColor); // добавить кнопку btnColor в контейнер
btnColor.setLabel("green"); // установить надпись кнопки
/*создать анонимную кнопку, на которую нельзя ссылать вдальнейшем*/
add(Button("Blue"));
```

События

Java использует так называемую *Event-Driven* модель обработки событий, аналогичную большинству языков визуального программирования (JavaScript, Delphi) – в ответ на действие пользователя вызывается программный код, обрабатывающий это действие. Классы обработки событий *AWT* хранятся в пакете *java.awt.event*. В обработке событий задействовано три объекта: источник (*source)*, слушатель (*listener*(s)) и событие (*event)*. Объект источник (например, кнопка или текстовое поле) взаимодействует с пользователем. После срабатывания события источник создает объект событие, который будет передан всем зарегистрированным слушателям (*registered listener)*, и подходящий обработчик события будет вызван. Срабатывание события у источника вызывает событие для всех его слушателей и вызывает соответствующий обработчик слушателя (ов).

Нажатие кнопки запускает так называемый *ActionEvent* и вызывает определенное запрограммированное действие – обработчик события. Нажатие кнопки (или нажатие клавиши «Enter» на *TextField*) запускает *ActionEvent* для всех слушателей *ActionEvent*. Класс *ActionEvent* должен реализовывать интерфейс *ActionListener*, в котором объявлян один абстрактный метод *actionPerformed()*, задающий алгоритм обработчика события.

Пример 1: AWTCounter

Создадим простую GUI-программу, обеспечивающую работу счетчика. Она включает контейнер верхнего уровня, который содержит три компонента – метку «Counter», нередактируемое текстовое поле для отображения счетчика, кнопка «Count». В текстовом поле при инициализации отображается 0. Каждый раз при нажатии на кнопку значение счетчика увеличивается на 1. Вид интерфейса

представлен на рисунке 2.5.6.

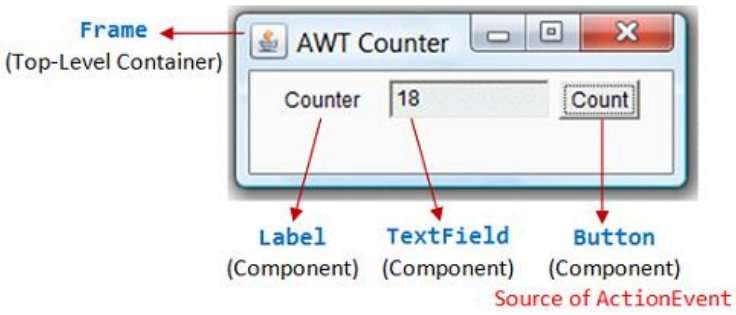


Рисунок 2.5.6 – Вид интерфейса счетчика (*AWTCounter*)

Пример 1. AWTCounter

```
import java.awt.*; // контейнерные и компонентные классы
/*классы обработки событий и интерфейс ActionListener*/
import java.awt.event.*;
public class AWTCounter extends Frame implements ActionListener {
  //класс наследует Frame и применяет интерфейс ActionListener
  private Label lblCount; // объявлена метка
  private TextField tfCount; // для вывода значения счетчика
  private Button btnCount; // объявлена кнопка
  private int count = 0; // значение счетчика
  /*конструктор для установки визуальных элементов //и обработчиков событий*/
  public AWTCounter () {
    setLayout(new FlowLayout());
    /*"super" Frame устанавливает диспетчер компоновки элементов в окне FlowLayout - компоненты будут упорядочены слева направо и перетекать в следующий ряд сверху вниз*/
    lblCount = new Label("Counter"); // создаем метку lblCount
    add(lblCount);  // добавляем ее к фрэйму "super" Frame
    /*создаем текстовое поле ввода tfCount для счетчика размером 10 символов, при инициализации выводится 0*/
    tfCount = new TextField("0", 10);
    tfCount.setEditable(false); // делаем поле tfCount нередактируемым
    add(tfCount); //добавляем tfCount к "super" Frame
    /*создаем кнопку tfCount для управления счетчиком*/
    btnCount = new Button("Count");
    add(btnCount);// добавить btnCount на "super" Frame
    btnCount.addActionListener(this);// добавить обработчик событий
      /* btnCount - это исходный объект, который запускает ActionEvent при нажатии. Он добавляет экземпляр this в качестве слушателя ActionEvent, который предоставляет метод обработки события  actionPerformed(). Т.о. нажав на кнопку btnCount вызывается actionPerformed() */
    setTitle("AWT Counter"); //устанавливаем заголовок "super"
    setSize(250, 100); // устанавливаем размер окна "super" Frame
    // Для отладки можно вывести информацию об объектах
    // System.out.println(this);
      /*выведется AWTCounter[frame0,0,0,250x100, invalid, hidden, layout=java.awt.FlowLayout, title=AWT Counter, resizable,normal]*/
    // System.out.println(lblCount);
```

```
    // System.out.println(tfCount);
    // System.out.println(btnCount);
    setVisible(true); // сделать "super" Frame видимым
    // System.out.println(this);
    // System.out.println(lblCount);
    // System.out.println(tfCount);
    // System.out.println(btnCount);
  }
  public static void main(String[] args) { // Метод main()
      /*Вызываем конструктор, чтобы настроить компоненты графического
интерфайса и создать экземпляр app */
    AWTCounter app = new AWTCounter();
    // или просто "new AWTCounter();" для анонимного экземпляра
  }
      /*Переопределяем обработчик события, который вызовется при
нажатии на кнопку*/
  @Override
  public void actionPerformed(ActionEvent evt) {
    ++count; // увеличить счетчик
    // Отобразить значение счетчика в поле tfCount
    tfCount.setText(count + ""); // при этом конвертировать int в String
  }
  }
```

Можно создать анонимный обработчик события для кнопки.

Пример. Анонимный обработчик события для кнопки

```
 btnCount.addActionListener(new ActionListener() {
  @Override
  public void actionPerformed(ActionEvent evt) {
     count--;
     tfCount.setText(count + "");
  }
 });
```

Такой подход удобен, если на форме несколько кнопок, то можно для каждой создать свой обработчик. (Альтернативный вариант – создать обработчики для каждой кнопки и запрограммировать когда, какой вызывать в методе *actionPerformed()*.)

Программный код необходимо сохранить в файле «AWTCounte.java», скомпилировать и запустить на выполнение. Как видно, для того чтобы выйти из программы, необходимо закрыть консоль (*cmd*), либо принудительно завершить выполнение приложения средствами *Eclipse*. Так происходит потому что не запрограммирован обработчик события «нажать на кнопку закрыть окно».

Пример. Обработчик события «нажать на кнопку закрыть окно»

```
 addWindowListener(new WindowAdapter() {
  public void windowClosing(WindowEvent windowEvent){
     System.exit(0);
  }
 });
```

WindowEvent запускается (для всех его слушателей *WindowEvent*), когда

окно (например, *Frame*) было открыто / закрыто, активировано / деактивировано, минимизировано через 3 кнопки в правом верхнем углу или другими способами. Источником *WindowEvent* должен быть оконный контейнер верхнего уровня, например *Frame*. Слушатель *WindowEvent* должен реализовать интерфейс *WindowListener*, в котором объявлено 7 абстрактных методов:

public void windowClosing(WindowEvent evt) – вызывается, когда пользователь пытается закрыть окно, нажав кнопку закрытия окна

public void windowOpened(WindowEvent evt) – вызывается когда окно первый раз становится видимым.

public void windowClosed(WindowEvent evt) – вызывается, когда окно было закрыто в результате вызова в окне.

public void windowActivated(WindowEvent evt) – вызывается когда окно становится активным

public void windowDeactivated(WindowEvent evt) – вызывается когда окно становится неактивным

public void windowIconified(WindowEvent evt) – вызывается, когда окно изменяется от нормального к минимизированному состоянию.

public void windowDeiconified(WindowEvent evt) – вызывается, когда окно изменяется с минимального до нормального состояния.

Таким образом, чтобы разрабатывать JAVA GUI-приложение необходимо:

– импортировать контейнерные и компонентные классы пакета *java.awt* и классы обработки событий и интерфейс *ActionListene*;

– создать контейнер верхнего уровня, обычно это подкласс класса *Frame*, от которого наследуются базовые стандартные компоненты окна;

– определить конструктор, в котором настроить и инициализировать GUI компоненты;

– задать с помощью метода *setLayout()* диспетчер компоновки элементов в окне. Можно выбрать одно из следующий значений: `FlowLayout`, `GridLayout`, `BorderLayout`, `GridBagLayout`, `BoxLayout`, `CardLayout`;

– создать компоненты интерфейса, вызвав соответствующий конструктор;

– добавить компоненты в контейнер с помощью метода *add()*;

– установить свойства контейнера с помощью методов *setSize()* и *setTitle()*;

– вызывать метод *setVisible (true)*, чтобы отобразить дисплей;

– установить обработчик события с помощью метода *addActionListener()* и переопределить метод *actionPerformed()* – задать алгоритм обработки события;

– в методе *main()* создать экземпляр класс, вызвав конструктор, который инициализирует компоненты интерфейса и установит механизм обработки события. Далее программа будет ждать действий пользователя.

Пример 2: AWTAccumulator

В данном примере контейнер верхнего уровня также *java.awt.Frame*, содержит 4 компонента: Label «Enter an Integer», TextField для ввода данных пользователя, Label «The Accumulated Sum is», нередактируемое поле TextField для вывода результата, компоновка элементов – *FlowLayout*. Вид окна представлен на рисунке 2.5.7.

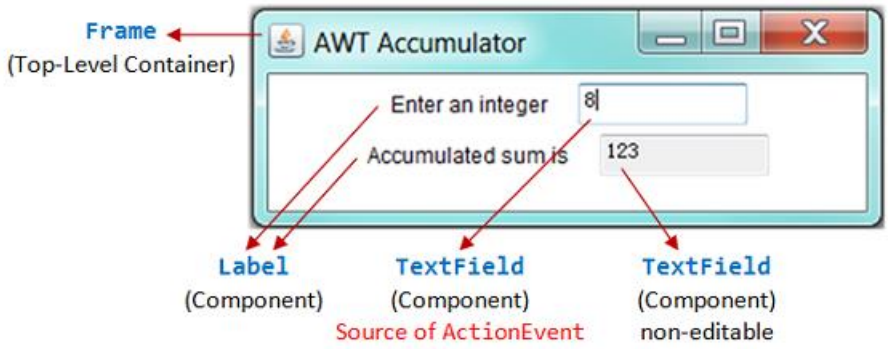


Рисунок 2.5.3 – Вид окна приложения **AWTAccumulator**

Программа должна накапливать сумму чисел, введенных пользователей через *TextField* для ввода, и отображать сумму в *TextField* для вывода.

Пример 2: AWTAccumulator

```
import java.awt.*;
import java.awt.event.*;
public class AWTAccumulator extends Frame implements ActionListener {
  private Label lblInput;
  private Label lblOutput;
  private TextField tfInput;
  private TextField tfOutput;
  private int sum = 0; // Накопитель суммы
  public AWTAccumulator() {
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent windowEvent){
        System.exit(0);}  });
    setLayout(new FlowLayout());
    lblInput = new Label("Enter an Integer: "); // Конструктор Label
    add(lblInput); // в окно "super" Frame добавляем Label
    tfInput = new TextField(10); // Конструктор TextField
    add(tfInput); // в окно "super" Frame добавляем TextField
    tfInput.addActionListener(this);// добавить обработчик события
    lblOutput = new Label("The Accumulated Sum is: ");
    add(lblOutput);
    tfOutput = new TextField(10);
    tfOutput.setEditable(false);
    add(tfOutput);
    setTitle("AWT Accumulator");
    setSize(350, 120);
    setVisible(true);
  }
  public static void main(String[] args) {
    // Вызов конструктора для настройки GUI
    new AWTAccumulator();
  }
  // Переопределяем обработчик события,
  // который вызовется при нажатии клавиши ввода на TextField
  @Override
  public void actionPerformed(ActionEvent evt) {
    // Получить введенную через tfInput значение-строку
```

```
    // и конвертировать ее к int
    int numberIn = Integer.parseInt(tfInput.getText());
    sum += numberIn; // Прибавить введенное значение к sum
    tfInput.setText(""); // очистить поле TextField
    // Отображаем sum в поле TextField, конвертировав к String
    tfOutput.setText(sum + "");
  }
}
```

tfInput (TextField) – объект-источник, который запускает ActionEvent по нажатиию на клавишу Enter. *tfInput* добавляет этот экземпляр как обработчик *ActionEvent handler*. В класс-слушателе (this или AWTAccumulator) необходимо реализовать интерфейс *ActionListener*, запрограммировав метод *actionPerformed()*. Когда пользователь нажмет *Enter* в поле ввода *tfInput* - вызовется *actionPerformed()*.

Пример 3: AWTChoice

Класс *Choice* используется для создания выпадающего списка выбора. Компонент *Choice* занимает ровно столько места, сколько требуется для отображения выбранного в данный момент элемента, когда пользователь щелкает мышью на нем, раскрывается меню со всеми элементами, в котором можно сделать выбор. Каждый элемент меню – это строка, которая выводится, выровненная по левой границе. Элементы меню выводятся в том порядке, в котором они были добавлены в объект *Choice*. Метод *countItems* возвращает количество пунктов в меню выбора. Вы можете задать пункт, который выбран в данный момент, с помощью метода *select*, передав ему либо целый индекс (пункты меню перечисляются с нуля), либо строку, которая совпадает с меткой нужного пункта меню. Аналогично с помощью методов *getSelectedItem* и *getSelectedIndex* можно получить, соответственно, строку-метку и индекс выбранного в данный момент пункта меню.

Choice() – создает экземпляр выпадающего списка.

void add(String item) – добавляет новый элемент в список.

Пример 3: AWTChoice

```
import java.awt.*;
import java.awt.event.*;
public class AWTChoice{
  private Frame mainFrame;
  private Label headerLabel;
  private Label statusLabel;
  private Panel controlPanel;
  public AWTChoice(){
    mainFrame = new Frame("Java AWT Examples");
    mainFrame.setSize(400,400);
    mainFrame.setLayout(new GridLayout(3, 1));
    mainFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent windowEvent){
        System.exit(0);
      }
    });
```

```
        headerLabel = new Label();
        headerLabel.setAlignment(Label.CENTER);
        statusLabel = new Label();
        statusLabel.setAlignment(Label.CENTER);
        statusLabel.setSize(350,100);
        controlPanel = new Panel();
        controlPanel.setLayout(new FlowLayout());
        headerLabel.setText("Элемент управления: Choice");
        final Choice fruitChoice = new Choice();
        fruitChoice.add("Яблоко");
        fruitChoice.add("Виноград");
        fruitChoice.add("Манго");
        fruitChoice.add("Банан");
        Button showButton = new Button("Показать");
        showButton.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
             String data = "Выбран фрукт: "
             + fruitChoice.getItem(fruitChoice.getSelectedIndex());
             statusLabel.setText(data);
          }
        });
        controlPanel.add(fruitChoice);
        controlPanel.add(showButton);
        mainFrame.add(headerLabel);
        mainFrame.add(controlPanel);
        mainFrame.add(statusLabel);
        mainFrame.setVisible(true);
      }
      public static void main(String[] args){
        AWTChoice  awtControlDemo = new AWTChoice();
      }
    }
```

Пример 4: AWTImage

Класс *Image* предназначен для работы с изображениями в java-приложении.

Пример 4: AWTImage

```
import java.awt.*;
import java.awt.event.*;
import java.awt.Image;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
public class AWTImage {
  private Frame mainFrame;
  private Label headerLabel;
  private Label statusLabel;
  private Panel controlPanel;
  public AWTImage(){
    mainFrame = new Frame("Java AWT Examples");
    mainFrame.setSize(400,400);

    mainFrame.setLayout(new GridLayout(3, 1));
    mainFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent windowEvent){
         System.exit(0);
      }
    });
    headerLabel = new Label();
    headerLabel.setAlignment(Label.CENTER);
    statusLabel = new Label();
    statusLabel.setAlignment(Label.CENTER);
    statusLabel.setSize(350,100);
    controlPanel = new Panel();
    controlPanel.setLayout(new FlowLayout());
    mainFrame.add(headerLabel);
    mainFrame.add(controlPanel);
    mainFrame.add(statusLabel);
    mainFrame.setVisible(true);
    headerLabel.setText("Элемент управления: Image");
    controlPanel.add(new
ImageComponent("IMG_20160712_124046.jpg"));
    mainFrame.setVisible(true);
  }
  public static void main(String[] args)throws Exception{
    AWTImage  awtControlDemo = new AWTImage();
  }
  class ImageComponent extends Component {
    BufferedImage img;
    public void paint(Graphics g) {
       g.drawImage(img, 0, 0, null);
    }
    public ImageComponent(String path) {
       try {
          img = ImageIO.read(new File(path));
       } catch (IOException e) {
```

```
            e.printStackTrace();
        }
    }
    public Dimension getPreferredSize() {
        if (img == null) {
            return new Dimension(100,100);
        } else {
            return new Dimension(img.getWidth(), img.getHeight());
        }
    }
  }
}
```

Практические задания и методические указания

1. Изучить краткие теоретические сведения по теме «Работа с пакетом Java.awt», разобрать и выполнить пример 1 и пример 2.

2. Модифицировать программный код примера 1 в соответствии со следующими заданиями:

a. Сделать поле *tfCount*.

b. Добавить текстовое поле *Result*, сделать его редактируемым.

c. Изменить подпись кнопки с *Count* на *Copy*.

d. Изменить алгоритм работы обработчика события нажатия на кнопку (функция *public void actionPerformed(ActionEvent evt)*) на следующий: при нажатие на кнопку «*Copy*» содержимое из текстового поля «*Counter*» копируется в текстовое поле «*Result*».

e. Переименовать кнопку «*Copy*» в «*Analize*». Изменить обработчик события нажатия на кнопку (функция *public void actionPerformed(ActionEvent evt)*) на следующий: если значение поля «*Counter*»больше 0, то в поле «*Result*» вывести «*positve*», если значение поля «*Counter*» равно 0, то в поле «*Result*» вывести «*zero*», если значение поля «*Counter*» меньше 0, то в поле «*Result*» вывести «*negative*». Примечание. Содержимое поля «*Counter*» – строка. Для корректной работы операций сравнения ее необходимо примести к типу *Integer*.

f. Изменить порядок компоновки элементов в окне. В строке *setLayout(new FlowLayout()*);

FlowLayout() заменить на *GridLayOut()*.

3. Используя интерфейсные элементы пакета JAVA.AWT запрограммировать задание 1.1 из лабораторной работы «Основные конструкции языка Java».

4. Разобрать и выполнить пример 3.

5. Используя интерфейсные элементы пакета JAVA.AWT запрограммировать задание 1.3 из лабораторной работы «Основные конструкции языка Java».

6. Разобрать и выполнить пример 4.

7. Вставить логотип в программный код, разработанный в результате выполнения пункта 5.

8. Создать отчет, включив в него описание работы по пунктам 1-7, ответы на контрольные вопросы.

Контрольный вопросы

1. Для чего предназначены пакеты JAVA.AWT и JAVA.AWT.EVENT?
2. Раскройте суть понятий Component, Container.
3. Какие интерфейсные элементы (GUI components) использовались при выполнении лабораторной работы. Назовите назначение каждого из них.
4. Приведите типовую структуру производного от класса Frame класса для создания оконного приложения.
5. Изобразите в виде блок-схемы алгоритм добавления компонента на фрэйм.
6. Какие свойства фрэйма, а также текстового поля Вы устанавливали в ходе выполнения данной лабораторной работы.
7. Какой параметр определяет порядок расположения элементов на фрэйме?
8. Какая модель обработки событий используется в Java-приложениях?
9. Изобразите в виде блок-схемы алгоритм добавления события объекту.
10. Какой код нужно добавить в программу, чтобы обеспечить закрытие окно по нажатию на крестик.

Лабораторная работа №8
Работа с пакетом java.awt.event

Цель: сформировать умения использовать средства пакета javax.awt.event при разработке программ.

Аппаратное, программное обеспечение: персональный компьютер, JDK, JDBC-драйвер, текстовый редактор (TextPad или NotePad++).

Краткие теоретические сведения

В программировании графического пользовательского интерфейса (GUI) Событие - это объект, описывающий изменение состояния источника события. Взаимодействие с элементами GUI приводит к генерации события. Любые действия с объектами GUI такие как, нажатие на кнопку, щелчок мыши, нажатие на клавиши клавиатуры приводят к генерации событий. Поэтому прямое или косвенное взаимодействие с элементами GUI, клавиатурой, мышью генерирует Событие. События могут генерироваться и в других случаях таких как, истечение времени по таймеру, начало и прекращение взаимодействия с экземпляром приложения, подтверждение или откат транзакции и другие.

Java AWT связывается при возникновении каких-либо действий с програмой, используя События и вызывая соответствующие методы для их обработки. Ранняя версия - Java 1.0 AWT управляла событиями, используя два метода - action() и handleEvent(). Action() - метод, имеющий три параметра - событие, x и y координаты в которых произошло событие. handleEvent() метод, вызываемый для объекта. Объект события создается и передается во время исполнения. Однако, сейчас используется новая Модель Делегирования Событий, где Событие может передаваться от источника к слушателю.

Источник события - это объект, который генерирует события. Когда внутреннее состояние объекта изменяется, он генерирует событие. Один источник событий может генерировать несколько видов Событий. Источник может регистрировать слушателей. Слушателю дается право на прием событий определенного типа. Источник событий имеет набор методов для регистрации одного или нескольких слушателей. Каждый тип событий имеет свой метод для регистрации.

Общая форма методов: public void addListener (TypeListener e).

Здесь, ТипСобытия имя события, параметр e - объектная ссылка на слушателя событий. Например, addKeyListener () метод регистрирующий события клавиатуры. Когда слушатели зарегистрированы, они получают уведомление, когда источник событий генерирует событие. Это и есть широковещание события. Некоторые источники позволяют зарегистрировать, только одного слушателя. В противном случае произойдет выброс исключения, чтобы этого не произошло необходимо использовать метод set<TunСобытия>Listener . В этом случае вещание будет производится только одному слушателю.

Примечание: Слушатель событий - это объект, которому сообщается о генерации событий источником. Слушатель событий - это объект класса, который реализует специальный интерфейс - интерфейс слушателя.

Резюме работы источника событий в модели делегирования событий.

– Источник событий - объект, регистрирующий одного или нескольких

слушателей.
– Источник событий посылает оповещение всем зарегистрированным слушателям о случившемся событии.

– Слушатели событий используют объект События для получения дополнительной информации и определения действий по реакции на событие.

Рассмотрим пример, когда пользователь нажал на кнопку. После этого генерируется событие action event и передается слушателю action listener. Необходим класс, реализующий соответствующий интерфейс. В данном случае необходимо реализовать интерфейс Action Listener. Это класс-слушатель, который определяет все методы, объявленные в интерфейсе ActionListener, для его реализации. Также класс-слушатель должен определять методы объявленные в интерфейсе ActionListener и работать с объектами Событий. В рассматриваемом случае, это всего один метод ActionPerformed, который имеет один параметр - объектная ссылка на объект класса ActionEvent. Рассмотрим часть кода, описывающую структуру слушателя.

```
public class MyEventTest extends JPanel
{
.  .
Jbutton myButton = new Jbutton( Blue );
// create Panels or other GUI objects   .  .
MyListener myAction = new MyListener()
MyButton.addActionListener(myListener);
private class MyListener implements ActionListener
    {
public MyListener()
    {
.  .  .  .  . // инициализация
    }
public void actionPerformed(ActionEvent event)
    {
.  .  .  .  . //Действия, происходящие при наступлении //события
    } } }
```

Когда пользователь нажмет кнопку, объект - кнопка создаст объект класса ActionEvent и вызовет listener.actionPerformed(event).

Пример MyEventTest можно разбить на следующие шаги:

1. Создание объекта класса Button.
2. Создание объекта класса MyListener.
3. Вызов метода addActionListener для добавления слушателя MyListener к источнику событий myButton.
4. Вызов объектом класса Button метода actionPerformed слушателя.

Пример 1: Событие нажатия на кнопку

```
import java.awt.*;
import java.awt.event.*;
public class MyButton
  {
public static void main(String[] args)
    {
MyButtonFrame frm = new MyButtonFrame();
frm.show();
```

```
    }
   }
/* A frame with a panel of buttons */
class MyButtonFrame extends Frame
{
public MyButtonFrame()
{
setTitle ( "To Test Button Event");
setSize(300, 200);
MyButtonPanel panel = new MyButtonPanel();
add(panel);
    }
}
/* A panel of three buttons. */
class MyButtonPanel extends Panel
{
public MyButtonPanel()
    {
// create buttons
Button bButton = new Button ( "Blue" ) ;
Button rButton = new Button( "Green" );
Button eButton = new Button( "Exit" );
// Add buttons to panel
add(bButton);
add(rButton);
add(eButton);
// Create button actions
    MyListenerAction bAction = new
MyListenerAction(Color.blue);
    MyListenerAction rAction = new
MyListenerAction(Color.green);
    MyListenerAction eAction = new
MyListenerAction(Color.red);
    // Add Listener object to Buttons
    bButton.addActionListener(bAction);
    rButton.addActionListener(rAction);
    eButton.addActionListener(rAction);
    }
//Action listener Class which is used to set background color
private class MyListenerAction implements ActionListener
    {
public MyListenerAction (Color c)
        {
bgColor = c;
        }
public void actionPerformed(ActionEvent event)
        {
setBackground(bgColor) ;
repaint();
if(event.getActionCommand() == "Exit" )
System.exit(0);
        }
private Color bgColor;
    }
}
```

Блоки прослушивания Listener представляют собой объекты классов, реализующих интерфейсы прослушивания событий, определенных в пакете java.awt.event. Соответствующие методы, объявленные в используемых интерфейсах, необходимо явно реализовать при создании собственных классов прослушивания. Эти методы и являются обработчиками события. Передаваемый источником блоку прослушивания объект-событие является аргументом обработчика события. Объект класса – блока прослушивания события необходимо зарегистрировать в источнике методом

Источником событий могут являться элементы управления: кнопки (JButton, JCheckbox, JRadioButton), списки, кнопки-меню. События генерируются окнами при развертке, сворачивании, выходе из окна. Каждый класс-источник определяет один или несколько методов addСобытиеListener() или наследует эти методы

Когда событие происходит, все зарегистрированные блоки прослушивания уведомляются и принимают копию объекта события. Таким образом источник вызывает метод-обработчик события, определенный в классе, являющемся блоком прослушивания, и передает методу объект события в качестве параметра. В качестве блоков прослушивания на практике используются внутренние классы. В этом случае в методе, регистрирующем блок прослушивания в качестве параметра, используется объект этого внутреннего класса.

AWT различает события низкого уровня и семантическими события. Если событие описывает действие пользователя (например, нажатие кнопки), - это семантическое событие; так событие ActionEvent - семантическое. События низкого уровня - события, которые делают это возможным. В пакете java.awt.event существуют четыре семантических события: ActionEvent. AdjustmentEvent, ItemEvent, TextEvent. Каждый класс события имеет конструктор и методы.

В таблице приведены некоторые интерфейсы и их методы, которые должны быть реализованы в классе прослушивания событий, реализующем соответствующий интерфейс:

Интерфейсы:	Обработчики события:
– ActionListener	– actionPerformed(ActionEvent e)
– AdjustmentListener	– adjustmentValueChanged(AdjustmentEvent e)
– ComponentListener	– componentResized(ComponentEvent e) – componentMoved(ComponentEvent e) – componentShown(ComponentEvent e) – componentHidden(ComponentEvent e)
– ContainerListener	– componentAdded(ContainerEvent e) – componentRemoved(ContainerEvent e)
– FocusListener	– focusGained(FocusEvent e) – focusLost(FocusEvent e)
– ItemListener	– itemStateChanged(ItemEvent e)

– KeyListener	– keyPressed(KeyEvent e) – keyReleased(KeyEvent e) – keyTyped(KeyEvent e)
– MouseListener	– mouseClicked(MouseEvent e) – mousePressed(MouseEvent e) – mouseReleased(MouseEvent e) – mouseEntered(MouseEvent e) – mouseExited(MouseEvent e)
– MouseMotionListener	– mouseDragged(MouseEvent e) – mouseMoved(MouseEvent e)
– TextListener	– textValueChanged(TextEvent e)
– WindowListener	– windowOpened(WindowEvent e) – windowClosing(WindowEvent e) – windowClosed(WindowEvent e) – windowIconified(WindowEvent e) – windowDeiconified(WindowEvent e) – windowActivated(WindowEvent e)

Событие, которое генерируется в случае возникновения определенной ситуации и затем передается зарегистрированному блоку прослушивания для обработки, – это объект класса событий. В корне иерархии классов событий находится суперкласс EventObject из пакета java.util. Этот класс содержит два метода: getSource(), возвращающий источник событий, и toString(), возвращающий строчный эквивалент события. Абстрактный класс AWTEvent из пакета java.awt является суперклассом всех AWT-событий, связанных с компонентами. Метод getID() определяет тип события, возникающего вследствие действий пользователя в визуальном приложении. Ниже приведены некоторые из классов событий, производных от AWTEvent, и расположенные в пакете java.awt.event:

ActionEvent – генерируется: при нажатии кнопки; двойном щелчке клавишей мыши по элементам списка; при выборе пункта меню;

AdjustmentEvent – генерируется при изменении полосы прокрутки;

ComponentEvent – генерируется, если компонент скрыт, перемещен, изменен в размере или становится видимым;

FocusEvent – генерируется, если компонент получает или теряет фокус ввода;

TextEvent – генерируется при изменении текстового поля;

ItemEvent – генерируется при выборе элемента из списка.

Класс InputEvent является абстрактным суперклассом событий ввода (для клавиатуры или мыши). События ввода с клавиатуры обрабатывает класс KeyEvent, события мыши – MouseEvent.

Чтобы реализовать методы-обработчики событий, связанных с клавиатурой, необходимо определить три метода, объявленные в интерфейсе KeyListener. При нажатии клавиши генерируется событие со значением KEY_PRESSED. Это приводит к запросу обработчика событий keyPressed(). Когда клавиша

отпускается, генерируется событие со значением KEY_RELEASED и выполняется обработчик keyReleased(). Если нажатием клавиши сгенерирован символ, то посылается уведомление о событии со значением KEY_TYPED и вызывается обработчик keyTyped().

Для регистрации события приложение-источник из своего объекта должно вызвать метод addKeyListener(KeyListener el), регистрирующий блок прослушивания этого события. Здесь el – ссылка на блок прослушивания события.

Класс ActionEvent: имеет два конструктора:
ActionEvent(Object src, int type,String cmd)
ActionEvent(Object src, int type,String cmd,int modifiers)

В обоих определениях src - источник события, объектная ссылка на объект, генерирующий это событие. Второй параметр- type определяет тип события и параметр cmd показывает, какие клавиши (ALT, CTRL, META, и SHIFT) были нажаты при генерации события. Для определения, какие клавиши были нажаты, используйте метод getModifier().

Класс AdjustementEvent: имеет один конструктор:
AdjustmentEvent(Adjustable src, int id, int type, int data)

Здесь, src - источник события, объектная ссылка на объект, генерирующий это событие, id содержит значение целого типа, путем сравнения его с константой ADJUSTMENT_VALUE_CHANGED можно узнать, какого рода изменение произошло. Этот класс содержит метод getAdjustable(), который возвращает объект, сгенерировавший событие (синтаксис: Adjustable getAdjustable()).

Класс ComponentEvent: Этот класс имеет один конструктор, один параметр - источник события, другой-тип события.

ComponentEvent(Component src, int type).

Метод getComponent() возвращает компонент, сгенерировавший событие. Его синтаксис: Component getComponent().

Класс ComponentEvent суперкласс для ContainerEvent, FocusEvent, KeyEvent, MouseEvent и WindowEvent.

Класс ContainerEvent: Конструктор этого класса:

ContainerEvent(Component src, int type, Component comp)

Поскольку этот класс наследовался от класса ComponentEvent, то он содержит все параметры конструктора класса ComponentEvent. Конструктор имеет еще один параметр – это объектная ссылка на объект, который был добавлен или удален из контейнера (src). Объектную ссылку на контейнер (src) можно получить, используя метод getContainer(). Общий синтаксис, Container getContainer().

Класс FocusEvent: Конструктор этого класса:
FocusEvent(Component src, int type)
FocusEvent(Component src, int type,boolean temporaryFlag)

Здесь, src - источник события, объектная ссылка на объект, генерирующий это событие. temporaryFlag показывает потерян ли фокус.

Класс InputEvent: Это абстрактный класс и подкласс ComponentEvent. Он имеет два подкласса: KeyEvent и MouseEvent.

Объекты класса KeyEvent генерируются при работе с клавиатурой. Существуют три типа событий, которые соответствуют константам: KEY_PRESSED, KEY_RELEASED, KEY_TYPED.

Класс KeyEvent имеет два конструктора:
KeyEvent(Component src, int type,long when, int modifiers, int code)
KeyEvent(Component src, int type,long when, int modifiers, int code, char ch)

src - источник события. Тип события определяется параметром type. Системное время, показывающее, когда произошло событие, - when, modifier - определяет какая дополнительная - управляющая клавиша была нажата при наступлении события. Код нажатой клавиши - code и символ нажатой клавиши - ch.

Пример2.

```
import java.awt.*;
import java.awt.event.*;
public class MyRect extends JApplet {
private Rectangle rect =
new Rectangle(20, 20, 100, 60);
private class AppletMouseListener//блок обработки событий
implements MouseListener {
/* реализация всех пяти методов интерфейса MouseListener */
public void mouseClicked(MouseEvent me) {
int x = me.getX();
int y = me.getY();
if (rect.contains(x, y)) {
showStatus(
"клик в синем прямоугольнике");
} else {
showStatus("клик в белом фоне");
}
}

// реализация остальных методов интрефейса пустая
public void mouseEntered(MouseEvent e) {}
public void mouseExited(MouseEvent e) {}
public void mousePressed(MouseEvent e) {}
public void mouseReleased(MouseEvent e) {}
}
public void init() {
setBackground(Color.WHITE);
```

```
/* регистрация блока прослушивания */
addMouseListener(new AppletMouseListener());
}
public void paint(Graphics g) {
g.setColor(Color.BLUE);
g.fillRect(rect.x, rect.y,
rect.width, rect.height);
}
}
```

Пример3(доработать):

```
public interface MouseListener {
  public void mousePressed(MouseEvent evt);  // Called back upon mouse-button pressed
  public void mouseReleased(MouseEvent evt); // Called back upon mouse-button released
  public void mouseClicked(MouseEvent evt);  // Called back upon mouse-button clicked (pressed and released)
  public void mouseEntered(MouseEvent evt);  // Called back when mouse pointer entered the component
  public void mouseExited(MouseEvent evt);   // Called back when mouse pointer exited the component
}
// An example of MouseListener, which provides implementation to the handler methods
class MyMouseListener implement MouseListener {
  @Override
  public void mousePressed(MouseEvent e)  {
    System.out.println("Mouse-button pressed!");
  }
  @Override
  public void mouseReleased(MouseEvent e) {
    System.out.println("Mouse-button released!");
  }
  @Override
  public void mouseClicked(MouseEvent e)  {
    System.out.println("Mouse-button clicked (pressed and released)!");
  }
  @Override
  public void mouseEntered(MouseEvent e)  {
    System.out.println("Mouse-pointer entered the source component!");
  }
  @Override
  public void mouseExited(MouseEvent e)   {
    System.out.println("Mouse exited-pointer the source component!");
  }
}
```

Практические задания и методические указания

1. Изучить теоретические сведения по теме.

2. Разобрать и выполнить примеры, пример3 доработать (добавить метод main()).

3. Создать отчет, включив в него описание работы по пункту2, результаты выполнения индивудуального задания согласно варианту и ответы на контрольные вопросы и выполнение индивидуального задания.

Задания для индивидуальной работы:

Вариант1. Использовать макет формы, представлены на рисунке1. Перемещать фокус на соответствующий RadioButton, при нажатии на стрелки «↑», «↓».

Вариант2. Использовать макет формы, представлены на рисунке1. Изменять цвет фона формы в соответствии с выбранным RadioButton.

Вариант3. Использовать макет формы, представлены на рисунке1. При наведении на Label курсора изменять форму кнопки.

Вариант4. Использовать макет формы, представлены на рисунке1. При нажатии на кнопку поочередно изменять содержимое RadioButton.

Вариант5. Использовать макет формы, представлены на рисунке1. При наведении курсора вне формы на форме пропадают RadioButton, при возвращении курсора в зону формы отображать RadioButton.

Вариант6. Использовать макет формы, представлены на рисунке2. При вводе текста в textField, добавлять его к содержимому Label.

Вариант7. Использовать макет формы, представлены на рисунке2. При нажатии на стрелки «→», «←» изменять цвет содержимого textField.

Вариант8. Использовать макет формы, представлены на рисунке2. При наведении курсора на textField изменять цвет текста кнопки.

Вариант9. Использовать макет формы, представлены на рисунке2. При нажатии на кнопку проверять содержимое textField на положительное число. Соответствующее значение сравнения отображать в Label.

Вариант10. Использовать макет формы, представлены на рисунк2. При нажатии на кнопку записывать в textField по одному слову из Label (содержит не менее 10 слов).

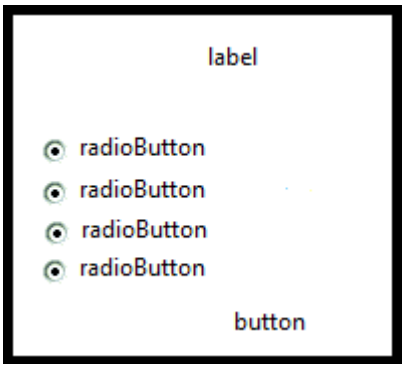


Рисунок1

Рисунок2

Контрольные вопросы

1. Для чего предназначен пакет JAVA.AWT.EVENT?
2. В чем суть модели событийного программирования?
3. Виды интерфейсов типа Listener.
4. Перечислить все источники событий, привести 5 примеров.

Лабораторная работа №9
Работа с пакетом javax.swing

Цель: сформировать умения использовать средства пакета javax.swing при разработке программ.

Аппаратное, программное обеспечение: персональный компьютер, JDK, JDBC-драйвер, текстовый редактор (TextPad или NotePad++).

Краткие теоретические сведения

Библиотека компонентов Swing содержит богатый набор управляющих элементов графических интерфейсов пользователя (*graphical user interfaces – GUIs*). Компоненты спроектированы таким образом, чтобы обеспечить настолько близкие черты внешнего облика и поведения GUI на всех платформах, насколько это возможно. Этим компоненты Swing существенно отличаются от компонентов AWT, которые «приспосабливаются» к особенностям конкретной платформы: если кнопка AWT выглядит как обычная кнопка Windows в системе Windows и как кнопка Mac в системе Macintosh, то кнопка Swing выглядит единообразно на всех платформах.

Компоненты Swing построены на основе той же модели событий, что и компоненты AWT и JavaBeans, хотя набор событий в данном случае несколько расширен.

Пакет javax.swing содержит целый ряд вложенных пакетов, позволяющих определять различные объекты графических интерфейсов, изменять их внешний вид, поддерживать средства редактирования кода HTML и текста, а также управлять некоторыми более сложными экранными объектами – такими как таблицы и деревья.

Четыре шага для создания GUI:

1. Создаем окно (JFrame):
JFrame frame = new JFrame();

2. Создаем компонент (кнопка, поле ввода и т. д.):
JButton button = new JButton("click me");

3. Добавляем компонент внутрь фрейма:
frame.getContentPane().add(BorderLayout.EAST, button);

4. Выводим его на экран (задаем ему размер и делаем видимым):
frame.setSize(300,300);
frame.setVisible(true);

Компоненты Swing

Объекты, которые вы вставляете в GUI и которые пользователь видит и использует: поля ввода, кнопки, прокручиваемые списки, переключатели и т. д., – все это компоненты. По сути, они все наследуют класс *javax.swing.JComponent.*

Практически все компоненты Swing начинаются с буквы J (JFrame, JTable, JMenu). Названия всех компонентов очевидны, и сходны с теми, которые использовались в AWT. К примеру, если в AWT в роли окна верхнего уровня использовалось Frame, в Swing используется в аналогичной роли JFrame.

В Swing практически все компоненты могут включать в себя другие компоненты. Иными словами, можно вставить почти что угодно во что-то еще. Но

большую часть времени вы будете вставлять *интерактивные* компоненты в *фоновые* компоненты (*JFrame* – фрейм, *JPanel* – панель). За исключением JFrame, разница между интерактивными и фоновыми компонентами довольно условна. JPanel, например, обычно используется как фон для группирования остальных компонентов, но даже он может быть интерактивным. Как и с другими компонентами, вы можете привязывать к JPanel события, в том числе щелчки кнопкой мыши или нажатие клавиш.

Простые Swing-виджеты:

JLabel – метка

JButton – кнопка

JTextField – текстовое поле

JFrame – фрэйм или контейнер верхнего уровня

JComboBox – ниспадающий список

JCheckBox/*JRadioButton* – варианты для выбора

JMenu/*JMenuItem*/*JMenuBar* – системы меню

JTextArea – несколько строк

JList – варианты для выбора

JTable – электронная таблица

Диспетчеры компановки

Диспетчер компоновки – это Java-объект, связанный с определенным компонентом, почти всегда фоновым. Диспетчер компоновки управляет компонентами, которые содержатся внутри него и с которыми он связан.

Допустим, панель включает в себя пять объектов. Даже если каждый из них обладает собственным диспетчером компоновки, размер и расположение пяти объектов на панели контролируются ее диспетчером компоновки. Если эти пять объектов, в свою очередь, включают другие объекты, то те располагаются в соответствии с диспетчером компоновки содержащего их объекта.

Когда мы говорим «содержит», то в действительности имеем в виду «добавлена». Панель содержит кнопку, потому что кнопка добавлена с помощью определенного кода, например:

my Panel.add (button);

Диспетчеры компоновки бывают нескольких типов, и любой фоновый компонент может иметь собственный диспетчер компоновки. Кроме того, они должны следовать определенным правилам при построении схем размещения.

Диспетчеры компоновки: *BorderLayout, FlowLayout, BoxLayout, GridLayout.*

Диспетчер ***BorderLayout*** делит фоновый компонент на пять областей: *east, west, north, south* и *center*. В каждую область, управляемую BorderLayout, вы можете добавить только один компонент. Компоненты, размещенные этим диспетчером, обычно не имеют предпочтений по размерам. BorderLayout – по умолчанию диспетчер компоновки для фрейма.

Диспетчер ***FlowLayout*** работает с компонентами наподобие текстового процессора. Каждый компонент имеет желаемый размер, и все они размещаются слева направо, в порядке добавления, с возможностью переноса на новую строку. Когда компонент не помещается по горизонтали, он переносится на следующую строку в компоновке. FlowLayout – по умолчанию диспетчер компоновки для панели.

Диспетчер ***BoxLayout*** похож на FlowLayout тем, что все его компоненты

получают собственный размер и располагаются в порядке добавления. Однако, в отличие от FlowLayout, BoxLayout позволяет располагать компоненты вдоль одной из осей – вертикально (*PAGE_AXIS, Y_AXIS)* или горизонтально *(LINE_AXIS, X_AXIS)*.

Диспетчер ***GridLayout*** – располагает компоненты в таблице.

Пример1. Использование компоновщика BorderLayout:

```
import javax.swing.*;
public class cl1 {
  public static void main(String[] args) {
    JFrame frame = new JFrame();
    JButton east = new JButton("East");
    JButton west = new JButton("West");
    JButton north = new JButton ("North") ;
    JButton south = new JButton("South");
    JButton center = new JButton("Center");
    frame.getContentPane().add(BorderLayout.EAST, east);
    frame.getContentPane().add(BorderLayout.WEST, west);
    frame.getContentPane().add(BorderLayout.NORTH, north);
    frame.getContentPane().add(BorderLayout.SOUTH, south);
    frame.getContentPane().add(BorderLayout.CENTER, center);
    frame.setSize(300,300) ;
    frame.setVisible(true);
  }
}
```

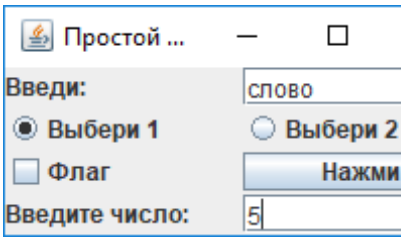
Пример 2. Использование компонентов swing

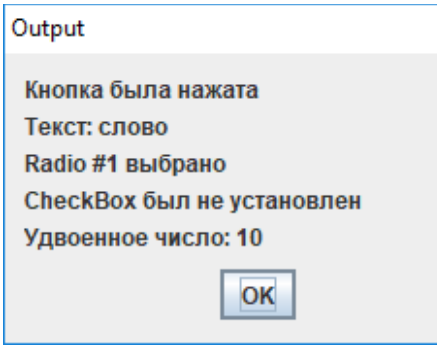
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class class1 extends JFrame {
  private JButton button = new JButton("Нажми");
  private JTextField input = new JTextField("", 5);
  private JLabel label = new JLabel("Введи:");
  private JRadioButton radio1 = new JRadioButton("Выбери 1");
  private JRadioButton radio2 = new JRadioButton("Выбери 2");
  private JCheckBox check = new JCheckBox("Флаг", false);
  public class1() {
    super("Простой пример");
    this.setBounds(100,100,250,100);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    Container container = this.getContentPane();
    container.setLayout(new GridLayout(3,2,2,2));
    container.add(label);
    container.add(input);
    ButtonGroup group = new ButtonGroup();
    group.add(radio1);
    group.add(radio2);
    container.add(radio1);
    radio1.setSelected(true);
```

```
    container.add(radio2);
    container.add(check);
    button.addActionListener(new ButtonEventListener());
    container.add(button);
  }
 class ButtonEventListener implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    String message = "";
    message += "Кнопка была нажата\n";
    message += "Текст: " + input.getText() + "\n";
    message += (radio1.isSelected()?"Radio #1":"Radio #2")  + " выбрано\n";
    message += "CheckBox был " + ((check.isSelected())?"установлен":
    "не установлен");
    JOptionPane.showMessageDialog(null,message,"Output",
    JOptionPane.PLAIN_MESSAGE);
  }
 }
 public static void main(String[] args) {
  class1 app = new class1();
  app.setVisible(true);
 }
 }
```

Практические задания и методические указания

1. Изучите краткие теоретические сведения.
2. Выполните пример1 и пример2 из теоретической части.
3. Измените последнюю программу, согласно приведенным скриншотам выполнения (для преобразования текста в число можно использовать, например Integer.parseInt())





4. Используя интерфейсные элементы пакета Swing запрограммировать задание 1.1 из лабораторной работы «Основные конструкции языка Java».
5. Используя интерфейсные элементы пакета Swing запрограммировать задание 1.2 из лабораторной работы «Основные конструкции языка Java».
6. Создать отчет, включив в него описание работы по пунктам 4-5, ответы на контрольные вопросы.

Контрольный вопросы

1. Для чего предназначены пакеты JAVAX.SWING?
2. Опишите работу диспетчеров компановки.
3. Перечислите простые Swing-виджеты.

4. Какие компоненты использовались при выполнении лабораторной работы. Назовите назначение каждого из них.
5. Опишите четыре шага для создания GUI.

Лабораторная работа №10
Программирование баз данных в Java

Цель: сформировать умения компиляции и запуска программ на языке Java с использованием MySQL JDBC Driver. Сформировать умения разрабатывать программы на языке Java с использованием запросов к СУБД на выборку и модификацию данных.

Аппаратное, программное обеспечение: персональный компьютер, JDK, JDBC-драйвер, текстовый редактор (TextPad или NotePad++).

Краткие теоретические сведения

Вместо того чтобы использовать клиент *mysql* для доступа к MySQL серверу напишем свое приложение на языке Java, которое сможет подключаться к серверу на заданном IP-адресе, по заданному порту (по умолчанию он использует для работы порт 3306), посылать SQL-команды, получать обработанные данные.

Установка JDBC-драйвера

Для выполнения работы необходимо, чтобы был установлен MySQL-сервер, а также JDK и среда разработки. Для взаимодействия с СУБД, необходимо установить подходящий *JDBC* (*Java Database Connectivity*) драйвер. JDBC драйвер для СУБД MySQL называется «*MySQL Connector/J*» и доступен на сайте MySQL.

Инструкция по установке драйвера для *ОС Windows*:

1. Загрузить последнюю версию JDBC-драйвера по ссылке http://dev.mysql.com/downloads ⇒ «MySQL Connectors» ⇒ «Connector/J» ⇒ Connector/J 5.1.{*xx*} ⇒ выбрать «Platform Independent» ⇒ ZIP Archive (e.g., «`mysql-connector-java-5.1.{xx}.zip`», где {*xx*} номер версии).
2. Разархивировать загруженный файл во временную папку.
3. Из временной папки скопировать JAR-файл `mysql-connector-java-5.1.{xx}-bin.jar` в директорию для расширений JDK `<JAVA_HOME>\jre\lib\ext` (где `<JAVA_HOME>` – директория установки JDK), «`c:\program files\java\jdk1.8.0_{xx}\jre\lib\ext`».

В *Mac OS X*:

1. Загрузить последнюю версию JDBC-драйвера по ссылке http://www.mysql.com/downloads ⇒ MySQL Connectors ⇒ Connector/J ⇒ Connector/J 5.1.{*xx*} ⇒ выбрать «Platform Independent» ⇒ Compressed TAR Archive (e.g., `mysql-connector-java-5.1.{xx}.tar.gz`, где {*xx*} номер версии).
2. Разархивировать загруженный файл во временную папку.
3. Из временной папки скопировать JAR-файл "`mysql-connector-java-5.1.{xx}-bin.jar`" в директорию для расширений JDK «`/Library/Java/Extensions`».

Можно скомпилировать программу без JDBC-драйвера. Но при запуске программы JAR-файл с JDBC драйвером должен находится в директории для расширений JDK, либо путь к нему может быть включен в переменную окружения `CLASSPATH`, либо задан с помощью опции `-cp` в командной строке.

Для ОС Windows:
> java -cp .;/path/to/mysql-connector-java-5.1.{xx}-bin.jar JDBCClassToBeRun

---

Для Macs/Unixes:

> java -cp .:/path/to/mysql-connector-java-5.1.{xx}-bin.jar JDBCClassToBeRun

Создание рабочей базы данных

Прежде чем приступать к программированию, необходимо создать базу данных, с которой будет осуществляться работа в программе. Пусть база данных имеет название «`ebookshop`», и она содержит таблицу «`books`», состоящую из 5 колонок.

Пример:

Database: **ebookshop**

Table: **books**

id (INT)	**title** (VARCHAR(50))	**author** (VARCHAR(50))	**price** (FLOAT)	**qty** (INT)
1001	Java for dummies	Tan Ah Teck	11.11	11
1002	More Java for dummies	Tan Ah Teck	22.22	22
1003	More Java for more dummies	Mohammad Ali	33.33	33
1004	A Cup of Java	Kumar	44.44	44
1005	A Teaspoon of Java	Kevin Jones	55.55	55

Подключиться к MySQL-сервер и проверить номер порта.

В ОС Windows:

cd {path-to-mysql-bin}

mysqld --console

В ОС Mac OS X:

Использовать «System Preferences» ⇒ MySQL

Запустить MySQL-клиент (предположим, что имя пользователя *`myuser`*, пароль *`xxxx`*):

Для ОС Windows

cd {path-to-mysql-bin} // Check your MySQL installed directory

mysql -u myuser -p

Для Mac OS X

cd /usr/local/mysql/bin

./mysql -u myuser -p

Выполнить следующие SQL-операторы, чтобы создать саму базу данных и таблицы в ней.

Пример:

```
create database if not exists ebookshop;
use ebookshop;
drop table if exists books;
create table books (
  id int, title varchar(50),
  author varchar(50),
  price float, qty int,
  primary key (id));
insert into books values (1001, 'Java for dummies', 'Tan Ah Teck', 11.11, 11);
```

insert into books values (1002,'More Java for dummies','Tan Ah Teck', 22.22, 22);
insert into books values (1003, 'More Java for more dummies', 'Mohammad Ali', 33.33, 33);
insert into books values (1004, 'A Cup of Java', 'Kumar', 44.44, 44);
insert into books values (1005, 'A Teaspoon of Java', 'Kevin Jones', 55.55, 55);
*select * from books;*

Программа JDBC (Java Database Connectivity) включает следующие шаги:

1. Создать объект *Connection* для подключения к базе данных.
2. Создать объект `Statement` в созданном объекте *Connection*..
3. Написать SQL-запрос и выполнить его, используя методы объекта `Statement` и созданное соединение.
4. Обработать результаты запроса.
5. Закрыть соединение, чтобы освободить ресурсы.

Схематично процесс выполнения и обработки результатов SQL-запроса представлен на рисунке 2.6.1.

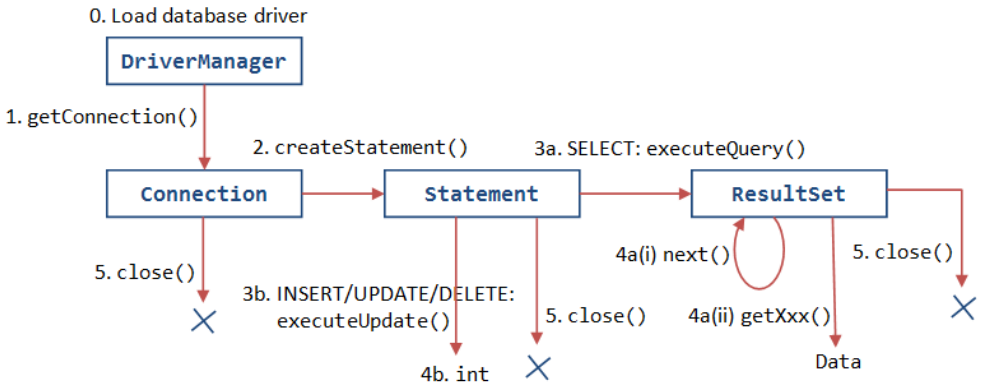


Рисунок 2.6.1 – Процесс выполнения и обработки результатов SQL-запроса в JDBC-программе

Процесс напоминает программирование с использованием баз данных на языке PHP.

Рассмотрим приемы программирования с использованием СУБД на практических примерах. (Работает для версии JDK 1.7 и выше.)

SELECT-запросы

Напишем программный код, который выполняет запрос на выборку данных и выводит на экран полученные в результате записи таблицы базы данных.

Пример:

```
import java.sql.*;  // используем классы пакета java.sql package
public class JdbcSelectTest {  // Сохранить как "JdbcSelectTest.java"
  public static void main(String[] args) {
    try (
      // Шаг 1: Создать объект conn класса Connection
      // и установить соединение с MySQL-сервером
      Connection conn = DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/ebookshop?useSSL=false", "myuser", "xxxx");
      // Шаг 2: Создать объект stmt класса Statement
      // для выполнения SQL-запроса
      Statement stmt = conn.createStatement();
    ) {
       // Шаг 3: Выполнить SELECT-запрос
       // результат будет возвращен в объект rset класса ResultSet
       String strSelect = "select title, price, qty from books";
       // вывод запроса на экран для отладки
       System.out.println("The SQL query is: " + strSelect);
       System.out.println();
       ResultSet rset = stmt.executeQuery(strSelect); //выполнение запроса
       // Шаг 4: Обработка объекта ResultSet прокручивая курсор вперед
       // с использованием метода next().
       // Для каждой строки получаем содержимое ячеек
       // методом get[типДанныхСтолбца]([имяСтолбца])
       System.out.println("The records selected are:");
       int rowCount = 0;
       while(rset.next()) { // Перемещаем курсор на следующую строку
       String title = rset.getString("title");
       double price = rset.getDouble("price");
       int qty = rset.getInt("qty");
       // вывод данных полученной строки на экран
       System.out.println(title + ", " + price + ", " + qty);
       ++rowCount;
      }
      System.out.println("Total number of records = " + rowCount);
    } catch(SQLException ex) {
    ex.printStackTrace();
    }
    // Шаг 5: Освободить ресурсы. Выполняется автоматически
  }
}
```

Сохранить программный код в файле ***JdbcSelectTest.java,*** **скомпилировать и запустить на выполнение.**

Таким образом, для разработки программ на Java с использованием запросов к СУБД необходимо задействовать классы «*Connection*», «*Statement*» и «*ResultSet*» из пакета *java.sql*. Нет необходимости знать детали осуществления поиска и извлечения данных, обмена ими между Java-приложением и СУБД, достаточно использовать методы, определенные *API*.

Соединение с СУБД устанавливается с помощью метода
DriverManager.getConnection("jdbc:mysql://localhost:{port}/{db-name}", "{db-user}", "{password}");

Создание запроса осуществляется методом `createStatement()`.

Отправка запроса и получение результирующего набора данных осуществляется методом `executeQuery()`. Данный метод возвращает объект класса `ResultSet`, который моделирует возвращенную таблицу, доступ к которой возможен через курсор строки. Курсор первоначально позиционируется перед первой строкой в результирующем наборе. Метод *next()* перемещает курсор в первую строку. Получить значения столбцов строки можно методами `getType`(*columnName*), где `Type` – тип данных, а *columnName* – название столбца таблицы базы данных из которого данные извлечены. Метод *next()* возвращает значение *false* в последней строке, которое вызовет завершение цикла *while*.

С помощью метода `getString`(*columnName*) можно извлечь значения любых типов. Для избегания ошибок столбцы в каждой строке результирующего набора следует читать в порядке слева направо, и каждый столбец следует читать только один раз. В JDK 7 появилась новая функция, называемая *try-with-resources*, которая автоматически закрывает все открытые ресурсы в блоке *try*, в нашем случае используемые объектами *Connection* и *Statement*.

UPDATE-запросы

Для того чтобы выполнить запрос на модификацию данных, необходимо использовать метод `executeUpdate()` класса `Statement`. Метод возвращает целочисленное значение, показывающее количество записей, подвергшихся изменению. Обратите внимание, что для `SELECT`-запроса использовался метод `executeQuery()`, который возвращал таблицу, содержащую результирующий набор данных, запросы `UPDATE|INSERT|DELETE` не возвращают набор данных, а возвращают количество записей, подвергшихся изменению.

Напишем программный код, демонстрирующий выполнение `UPDATE-запрос к СУБД`. *(Работает для версии JDK 1.7 и выше.)*

Пример:

```
import java.sql.*;
public class JdbcUpdateTest {
  public static void main(String[] args) {
    try (
      // Шаг 1: Создать объект conn класса Connection
      // установить соединение с MySQL-сервером
      Connection conn = DriverManager.getConnection(
         "jdbc:mysql://localhost:3306/ebookshop?useSSL=false", "myuser", "xxxx"); //
         MySQL
      // Шаг 2: Создать объект stmt класса Statement
      // для выполнения SQL-запроса
      Statement stmt = conn.createStatement();)
    {
```

```
      // Шаг 3 и 4: Выполнение запроса с помощью метода executeUpdate()
      // Увеличить цену на 7% и qty на 1 для записи с id=1001
      String strUpdate = "update books set price = price*0.7,
           qty = qty+1 where id = 1001";
      // вывод на экран для отладки
      System.out.println("The SQL query is: " + strUpdate);
      // выполнение запроса, результат – количество измененных записей
      // помещается в переменную countUpdated
      int countUpdated = stmt.executeUpdate(strUpdate);
      System.out.println(countUpdated + " records affected.");
    } catch(SQLException ex) {
      ex.printStackTrace();
    }
    // Шаг 5: Освободить ресурсы. Выполняется автоматически
  }
}
```

Проверить изменения в базе данных можно выполнив через MySQL-клиент, используемый в предыдущем пункте при создании БД, запрос

*select * from books where id = 1001;*

INSERT и DELETE-запросы

Для выполнения запросов на добавление и удаление данных используется тот же метод `executeUpdate()`.

Пример:

```
import java.sql.*;
public class JdbcInsertTest {
  public static void main(String[] args) {
    try (
      // Шаг 1: Создать объект conn класса Connection
      // установить соединение с MySQL-сервером
      Connection conn = DriverManager.getConnection(
          "jdbc:mysql://localhost:3306/ebookshop?useSSL=false", "myuser",
```

```
            "xxxx"); // MySQL
        // Шаг 2: Создать объект stmt класса Statement
        // для выполнения SQL-запроса
        Statement stmt = conn.createStatement();)
    {
      // Шаг 3 и 4: Выполнение запросов
      // с использованием метода executeUpdate()
      // Удалить записи с id>=3000 и id<4000
      String sqlDelete = "delete from books where id>=3000 and id<4000";
      // вывод на экран для отладки
      System.out.println("The SQL query is: " + sqlDelete);
      int countDeleted = stmt.executeUpdate(sqlDelete);
      System.out.println(countDeleted + " records deleted.\n");
      // добавить запись
      String sqlInsert = "insert into books " // необходим пробел
         + "values (3001, 'Gone Fishing', 'Kumar', 11.11, 11)";
      System.out.println("The SQL query is: " + sqlInsert);
      int countInserted = stmt.executeUpdate(sqlInsert);
      System.out.println(countInserted + " records inserted.\n");
      // добавить несколько записей
      sqlInsert = "insert into books values "
         + "(3002, 'Gone Fishing 2', 'Kumar', 22.22, 22),"
         + "(3003, 'Gone Fishing 3', 'Kumar', 33.33, 33)";
      System.out.println("The SQL query is: " + sqlInsert);
      countInserted = stmt.executeUpdate(sqlInsert);
      System.out.println(countInserted + " records inserted.\n");
      // добавить одну запись, указаны значения не для всех полей
      sqlInsert = "insert into books (id, title, author) "
         + "values (3004, 'Fishing 101', 'Kumar')";
      System.out.println("The SQL query is: " + sqlInsert);
      countInserted = stmt.executeUpdate(sqlInsert);
      System.out.println(countInserted + " records inserted.\n");
    } catch(SQLException ex) {
      ex.printStackTrace();
    }
    // Шаг 5: Освободить ресурсы. Выполняется автоматически
  }
}
```

Проверить изменения в базе данных можно выполнив через MySQL-клиент, используемый в предыдущем пункте при создании БД, запрос

*select * from books;*

Практические задания и методические указания

1. Изучить краткие теоретические сведения. Установить MySQL, MySQL JDBC Driver, выполнить тестовые примеры.

2. Модифицировать приведенный в описании примеры по следующему заданию:

   a. Выполнить и вывести результаты следующих запросов

   – SELECT * FROM books

- SELECT title, author, price, qty FROM books WHERE author = 'Tan Ah Teck' OR price >= 30 ORDER BY price DESC, id ASC
- Delete all books with id > 8000

b. Увеличить на 50% цену «A Cup of Java»
c. Установить значение `qty` 0 для «A Teaspoon of Java».
d. Добавить записи (8001, 'Java ABC', 'Kevin Jones', 15.55, 55) и (8002, 'Java XYZ', 'Kevin Jones', 25.55, 55)

3. Выполнить индивидуальное задание согласно варианту.
4. Создать отчет, включив в него описание работы по пунктам 1-3, ответы на контрольные вопросы.

Индивидуальные задания

1. Создать таблицу «Рабочие» в базе данных «Бригада». Распечатать данные о рабочих, чьи фамилии начинающиеся с заданной буквы «А» с указанием их месячной зарплаты.

2. Создать таблицу «Учащиеся» в базе данных «Академия». Распечатать список учеников, фамилии которых начинаются с заданной буквы с указанием даты их рождения.

3. Создать таблицу «Учащиеся» в базе данных «Академия». Распечатать список учеников, родившихся в заданном месяце.

4. Создать таблицу «Учащиеся» в базе данных «Музыкальная школа». Распечатать список учеников музыкальной школы, которые учатся играть на скрипке. Указать также, сколько лет они занимаются музыкой и принимали ли участие в каких-либо конкурсах

5. Создать таблицу «Спортсмены» в базе данных «Соревнования». Из заданного списка спортсменов распечатать сведения о тех, кто занимается плаванием. Указать возраст, сколько лет они занимаются спортом.

6. Создать таблицу «Преподаватели» в базе данных «Академия». Распечатать список преподавателей, которые преподают математику, указать их стаж работы и недельную нагрузку.

7. Создать таблицу «Участники» в базе данных «Олимпиада» (один участник может участвовать в олимпиаде по одному направлению). Распечатать анкетные данные учеников, участвовавших в олимпиаде по программированию и заработавших не менее 30 балов.

8. Создать таблицу «Граждане» в базе данных «Перепись населения». Напечатать фамилии, имена и подсчитать общее количество жителей, родившихся после 1990 года.

9. Создать таблицу «Служащий», состоящую из полей фамилия, дата рождения, профессия, в базе данных «Организация». Описать переменную служащий,. Выдать сведения о служащем, который имеет заданную профессию.

10. Создать таблицу «Расписание», содержащую день недели, время начала и конца занятия, название предмета, фамилию преподавателя, в базе данных «Академия». Вывести полную информацию о занятиях по программированию.

Контрольные вопросы

1. Опишите, для чего используется MySQL JDBC Driver.
2. Назовите этапы включает в себя JDBC-программа?
3. Какие классы и методы использовались в данной работе?

4. В чем отличие в выполнении и обработке результата запроса на выборку и модификацию данных?

Лабораторная работа №11
Этапы работы с программой на языке C#

Цель: сформировать первоначальные умения по разработке программ на языке C#.

Аппаратное, программное обеспечение: персональный компьютер, Visual Studio.

Краткие теоретические сведения

Для разработки профессионального програмного обеспечения в среде .NET зачастую используется среда разработки производства компании Microsoft, котороя называется *Visual Studio* и доступна по адресу *www.visualstudio.com*. Этот продукт представляет собой полностью интегрированную и наиболее функционально насыщенную IDE-среду. Она спроектирована таким образом, чтобы делать процесс написания кода, его отладки и компиляции как можно более простым. На практике это означает, что Visual Studio является достаточно сложным приложением и каждая версия продукта предоставляет свой уникальный набор функциональных возможностей.

После установки Visual Studio можно приступать к созданию первого проекта, для этого можно создать новый проект и указать Visual Studio приложение какого типа требуется построить. Среда разработки автоматически сгенерирует файлы и исходный код C#, составляющие каркас программы.

Создание нового проекта

Для создания простого приложения требуется выбрать в меню *File (Файл) – New Project (Создать - Проект)*.

В появившемся диалоговом окне необходимо:

1. Выбрать тип создаваемого приложения *Windows Forms App* (Приложение Windows Forms).
2. В поле *Name* (Имя) указать наименование создаваемого пректа.
3. Указать *Location* (Путь) каталога нового проекта. Это расположение, где мастер будет помещать создаваемые файлы (и подкаталоги).
4. Задать *Solution name* (Имя решения). Решение может объединять в себе несколько различных проектов. По умолчанию имя решения задаётся таким же, как и имя проекта.
5. Из выпадающего списка *Framework* выбрать версию *NET Framework (2.0, 3.x или 4.0)*, для которой должно создаваться приложение.
6. Нажать кнопку *OK*.

Пример. Автоматически создаваемый файл Program.cs содержит следующие строки кода:

```
//использование имён заданных пространств имён
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;
//объявление нового пространства имён
namespace WindowsFormsApp2
{
   //объявление нового класса
   static class Program {
      /// <summary>
      /// The main entry point for the application
      /// </summary>
      [STAThread]
      //точка входа в программу
      static void Main() {
         Application.EnableVisualStyles();
         Application.SetCompatibleTextRenderingDefault(false);
         //запуск приложения
         Application.Run(new Form1());
      }
   }
}
```

Файл Form1.cs описывает основные методы главного окна программы. Для того, чтобы переключиться от дизайнера к файлу кода требуется нажать *F7*.

Пример. По умолчанию код формы

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace WindowsFormsApp2
{
   public partial class Form1 : Form
   {
      //конструктор формы
      public Form1()
      {
         //инициализация формы
         InitializeComponent();
      }
   }
}
```

Текстовый редактор позволяет готовить исходные тексты программ на языке C# и обладает достаточно широкими возможностями. К примеру, он автоматически компонует вводимый текст программы на странице, создавая между строками необходимые отступы, выравнивая открывающие и закрывающие скобки блоков кода и выделяя цветом ключевые слова. Кроме того, он выполняет проверку исходного кода на предмет синтаксических ошибок и подчеркивает фрагменты, вызывающие ошибки при компиляции, что также называется отладкой на стадии проектирования. Дополнительно редактор поддерживает технологию *IntelliSense*, которая обеспечивает автоматическое отображение имен классов, полей или методов при начале их ввода, а также списки параметров, которые поддерживают все доступные перегруженные версии методов при начале ввода параметров для методов. Для вывода подсказок *IntelliSense* используется комбинация клавиш *Ctrl+Space*.

Visual Studio имеет встроенный отладчик кода и позволяет создавать точки остановки, а также отслеживать значения переменных, не покидая среду разработки. Во ремя работы с текстовым редактором есть возможность выделить ключевое слово в коде и нажав клавишу <F1> вызвать раздел справки, относящийся именно к этому ключевому слову. Аналогично, если требуется узнать, что означает та или иная ошибка компиляции, необходимо выделить сообщение с ошибкой и нажать <F1>.

Управление составом проекта

Утилита *Solution Explorer* (Обозреватель решений, Ctrl+Alt+L), предназначена для просмотра всех элементов, внешних сборок, проектов и файлов, водящих в состав текущего решения.

В дереве элементов также отображаются все классы с их полями, свойствами и методами. Сгенерированные автоматически средой разработки и созданые вручную. Новый класс в программу вводят в случае, когда необходимо добавить функционал, который можно выделить в отдельный модуль. Вынесение класса в отдельные модули позволяет не загромождать побочным кодом основной модуль программы.

Для *добавления нового класса* в программу следует:

1. Щёлкнуть правой кнопкой мыши по имени проекта в *Solution Explorer* и выбрать *Add (Добавить) – New Item (Новый элемент)*.
2. В диалоговом окне выбрать *Class (Класс)*.
3. Указать имя класса в строке *Name*.
4. Нажать кнопку *Add*.

В результате файл с расширением .cs будет автоматически сгенерирован и добавлен в дерево элементов. В созданном файле требуется описать структуру класса: поля, конструкторы, свойства, методы и т.д.

Пример. Структура класса Child

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace L11VS01 //объявление пространства имён
{
   class Child
   {
   // скрытые поля класса
      private int age;
      private string name;
      // конструктор по умолчанию
      public Child()
      {
         name = "N/A";
      }
      // конструктор с двумя параметрами
      public Child(string name, int age)
      {
         this.name = name;
         this.age = age;
      }
      // метод для вывода информации на консоль
      public void PrintChild()
      {
         Console.WriteLine("{0}, {1} years old.", name, age);
      }
   }
}
```

Добавление ссылок на внешние сборки

Внутри каталога *References (Ссылки)* отображается список внешних сборок, на которые в проекте были добавлены ссылки.

Для добавления новых сборок в проект необходимо щёлкнуть правой кнопкой мыши на каталоге *References* и выбрать в контекстном меню пункт *Add Reference (Добавить ссылку)*.

В диалоговом окне *Add Reference* можно выбрать необходимые сборки, в частности, на вкладке *Assemblies* отображаются наиболее востребованные из них.

В Visual Studio также есть утилита, позволяющая изучить множества доступных сборок – *Object Browser* (Браузер объектов, Ctrl+Alt+J).

Просмотр свойств проекта

В окне утилиты *Solution Explorer* находится кнопка *Properties (Свойства)*, позволяющая открыть окно редактора конфигураций проекта – *Project Properties (Свойства проекта)*. В этом окне возможно установить параметры безопасности приложения, задать имя сборки, развернуть прилоение, установить дополнительные ресурсы приложения и конфигурировать события, которые должны происходить перед и после компиляции сборки.

В поле *Output type (Тип выходных данных)* указывается способ вывода

информации для пользователя:

– *Windows Application* (Приложение Windows)
– *Console Application* (Консольное приложение)
– *Class Library* (Библиотека классов)

При работе с консольным приложением дла ввода-вывода данных следует использовать стандартные потоки, представленные свойствами *Console.In, Console.Out* и *Console.Error*.

Практические задания и методические указания

1. Изучить теоретические сведения.
2. Создать приложение *Windows Forms* и установить значение *Console Application* в свойстве *Output type*.
3. Разработать графический интерфейс для реализации индивидуального задания.
4. Выполнить индивидуальное задание согласно варианту.
5. Каждый разрабатываемый класс должен содержать следующие элементы: закрытые поля, конструкторы с параметрами и без параметров.
6. Выводить в консоль информацию о вызываемом конструкторе.
7. Создать отчет, включив в него блок-схему разработанного алгоритма, ответы на контрольные вопросы.

Индивидуальные задания

1. Создать класс Point для работы с точками а плоскости, разработав следующие элементы класса:
   - Поля:
     - int x, y;
   - Конструкторы, позволяющие создать экземпляр класса:
     - с нулевыми координатами;
     - с заданными координатами.
2. Создать класс Rectangle для работы с прямоугольниками, разработав следующие элементы класса:
   - Поля:
     - int a,b;
   - Конструкторы, позволяющие создать экземпляр класса:
     - со сторонами равными 1;
     - с заданными сторонами.
3. Создать класс Complex для работы с комплексными числами, разработав следующие элементы класса:
   - Поля:
     - double re, im;
   - Конструкторы, позволяющие создать экземпляр класса:
     - с мнимой и действительными частями равными 0;
     - с заданными мнимой и действительной частями.
4. Создать класс Triangle для работы с треугольниками, разработав следующие элементы класса:
   - Поля:
     - int a,b,c;

- Конструкторы, позволяющие создать экземпляр класса:
  - со сторонами равными 1;
  - с заданными сторонами.

5. Создать класс Quadratic для работы с квадратными уравнениями, разработав следующие элементы класса:
   - Поля:
     - double a,b,c;
   - Конструкторы, позволяющие создать экземпляр класса:
     - с нулевыми коэффициентами;
     - с заданными коэффициентами.
6. Создать класс Circle для работы с окружностями на плоскости, разработав следующие элементы класса:
   - Поля:
     - int x,y,r;
   - Конструкторы, позволяющие создать экземпляр класса:
     - с центром в начале координат и единичным радиусом;
     - с заданным центром и радиусом.
7. Создать класс Vector для работы с вектором на плоскости, разработав следующие элементы класса:
   - Поля:
     - int x,y;
     - double l;
   - Конструкторы, позволяющие создать экземпляр класса:
     - с началом в начале координат и единичной длиной;
     - с заданным началом и длиной.
8. Создать класс Ellipse для работы с элипсами, разработав следующие элементы класса:
   - Поля:
     - double r1, r2;
   - Конструкторы, позволяющие создать экземпляр класса:
     - с единичными радиусами;
     - с заданными радиусами.
9. Создать класс Linear для работы с линейными уравнениями от трёх неизвестных, разработав следующие элементы класса:
   - Поля:
     - int a, b, c;
   - Конструкторы, позволяющие создать экземпляр класса:
     - с единичными радиусами;
     - с заданными радиусами.
10. Создать класс Figure для работы с равносторонними многоугольниками, разработав следующие элементы класса:
    - Поля:
      - int n;
      - double a;
    - Конструкторы, позволяющие создать экземпляр класса:
      - с тремя сторонами равными 1;
      - с заданным количеством сторон и их длиной.

Контрольные вопросы

1. В чём состоит назначение директивы using?
2. Что является точкой входа в программу?
3. Приведите синтаксис объявления нового пространства имён.
4. Для чего необходима утилита Solution Explorer?
5. Как добавить ссылки на внешние сборки в проект?

Лабораторная работа №12
Основные конструкции языка C#

Цель работы: сформировать умения разработки программ на языке C# с применением основных конструкций.

Аппаратное, программное обеспечение: персональный компьютер, Visual Studio.

Краткие теоретические сведения

Microsoft.NET (.NET Framework) – программная платформа, состоящая из общеязыковой среды выполнения (Common Language Runtime, CLR) и библиотеки классов .NET.

Описание и инициализация переменных в C# выполняются с использованием конструкции:

Тип_переменной имя_переменной [=значение];

Например:

```
int x;          //объявление переменной x
x=100;          //инициализация переменной x
byte y, z=100;  //объявление переменных y и z и инициализация z
byte w=100*z;   //объявление с динамической //инициализацией
```

C# – язык программирования со строгой проверкой на соответствие типу данных. Все элементарные типы данных имеют собственное зарезервированное обозначение (таблица 3.1)

Таблица 3.1 – Элементарные типы данных и их псевдонимы в C#

Псевдоним C#	*Системный тип*	*Диапазон*	*Пояснения*
void	Void	Пусто	Применяется для соблюдения синтаксиса
Типы значений			
sbyte	SByte	-128 – 127	Знаковое 8 бит
byte	Byte	0 – 255	Беззнаковое 8 бит
short	Int16	-32 768 – 32 767	Знаковое 16 бит
ushort	UInt16	0 – 65 535	Беззнаковое 16 бит
int	Int32	-2 147 483 648 – 2 147 483 647	Знаковое 32 бит (4 байта)
uint	UInt32	0 - 4'294'967'295	Беззнаковое 32 бит
long	Int64	-9 223 372 036 854 775 808 – 9 223 372 036 854 775 807	Знаковое 64 бит (8 байт)
ulong	UInt64	0 – 18'446'774'073'709'551'615	Беззнаковое 64 бит (8 байт)

Продолжение таблицы 3.1

Псевдоним C#	*Системный тип*	*Диапазон*	*Пояснения*
char	Char	U+0000 – U+FFFF	Для описания только одного символа Unicode 16 бит
float	Single	1.5x10-45 - 3.4x1038	Знаковое с

			плавающей точкой 32 бит (4 байта)
double	Double	5.0x10-324 - 1.7x10308	Знаковое с плавающей точкой 64 бит (8 байт)
bool	Boolean	true или false	Логическое
decimal	Decimal	100 – 1028	Знаковое целое (16 байт)
Ссылочные типы			
string	String	Ограничено только системной памятью	Для описания строки символов Unicode любой длины
object	Object	Практически все что угодно	Все типы происходят от System.Object, поэтому объектом является все Класс, базовый для всех типов в . NET

Значимые типы данных

Экземпляры элементарных типов, структур и перечислений представлены переменными, которые являются самими данными. Такие типы называют значимыми. Если локальная переменная представляет значимый тип, то все представленные ею данные размещаются в стеке.

Объекты значимого типа поддерживают два вида синтаксиса:

1. Статическое объявление предполагает неявный вызов конструктора по умолчанию только при объявлении полей класса.

int x;

Point point;

2. Динамическое объявление предполагает одновременно создание и инициализацию объекта. Такое объявление допускается и внутри метода.

int x = new int();

Point point = new Point();

Упаковка и распаковка значимых типов

Класс *object* является родительским для всех типов данных в C#. Хранение значимого типа с его адресом выполняется в ссылочной переменной, которая будет иметь тип *object*. Такой процесс называется упаковкой значимого объекта:

object refValue = point;

Среда выполнения сама резервирует место в динамической области памяти и копирует туда все данные значимого типа.

Упакованный объект можно передать как ссылочный параметр любому внешнему методу. По ссылке упакованного объекта нельзя обратиться к его членам. Для этого объект нужно распаковать, что позволит получить значимый объект по его ссылке:

Point point = (Point)refValue;

Ссылка на упакованный объект приводится с помощью явного преобразования к самому объекту того же типа. Типы упакованного и распакованного объектов должны быть совместимы, т.е. иметь одинаковую структуру данных и функций.

Область видимости

Область видимости определяет, где можно использовать переменную, начинается в точке описания переменной и длится до конца блока. ***Блок*** – это код, заключённый в фигурные скобки.

Примеры:

```
if (length>10) //область видимости переменных внутри блока if
  {
    int area = length* length;
  }
private string message; //область видимости переменной в блоке класса
void SetString()
  {
    Message = "Hello, World!";
  }
```

Использование одинаковых имён переменных в разных частях программного кода допустимо только в случае, если области видимостей этих переменных не перекрываются. Однако C# различает переменные, объявленные на уровне типа (поля класса) и переменные объявленные в методе (локальные переменные).

Пример:

```
using System;
namespace ExampleApp
{
  class Program {
     static int i = 10;       //переменная класса существует
     //пока существует сам класс
     public static void Main()
       {
         int i = 20;  //локальная переменная метода
         //скрывает переменную класса
         Console.WriteLine(i); //выводится 20
         return;
       }
    }
        }
```

Пространство имён определяет область объявления, что позволяет хранить каждый набор имён отдельно от других. Переменные, объявлённые в одном пространстве имён, не конфликтуют с такими же переменными, объявлёнными в пространстве имён с другим имененм.

Для того, чтобы использовать переменные пространств имён без указания его полного имени, применяется ключевое слово *using*:

using System.Linq;

using TextLib = System.Text; //использование псевдонима для имени

Свойства в C#

Свойство в C# – это член класса, который предоставляет доступа к полю класса (чтение поля и запись). При использовании свойства программа обращается к нему как к полю класса, но на самом деле компилятор использует вызов *аксессора (accessor)*. Существуют два типа аксессора:

– *get* – для получения данных,
– *set* – для записи свойства.

Объявление свойства имеет структуру:

```
[модификатор доступа] [тип] [имя_свойства]
{
   get
        {
        // тело аксессора для чтения из поля
        }
   set
        {
        // тело аксессора для записи в поле
        }
}
```

Пример использования свойств

```
class Student
 {
  private int year; //объявление закрытого поля
  public int Year //объявление свойства
   {
    get // аксессор чтения поля
     {
      return year;
     }
    set // аксессор записи в поле
     {
      if (value < 1)
       year = 1;
      else if (value > 5)
       year = 5;
      else year = value;
     }
   }
 }
class Program
 {
  static void Main(string[] args)
   {
    Student st1 = new Student();
    st1.Year = 0; // записываем в поле, используя аксессор set
    // читаем поле, используя аксессор get, выведет 1
    Console.WriteLine(st1.Year);
    Console.ReadKey();
   }
}
```

Свойство может предоставлять доступ только на чтение или только на запись поля в зависимости от постановки задачи.

Пример:

```
class Student
 {
  private int year;
  public Student(int y) // конструктор
   {
    year = y;
   }
  public int Year //свойство на получение данных
   {
    get
     {
      return year;
     }
   }
 }
class Program
 {
  static void Main(string[] args)
   {
    Student st1 = new Student(2);
    Console.WriteLine(st1.Year); // чтение
    st1.Year = 5; // ошибка, свойство только на чтение
    Console.ReadKey();
   }
 }
```

Автоматическое свойство – это простое свойство, которое определяет место в памяти создавая неявное поле, но не задаёт логику доступа к этому полю.

Общая структура автоматического свойства:

[модификатор доступа] [тип] [имя_свойства] { get; set; }

Пример использования свойства без логики доступа

```
class Student
 {
  public int Year { get; set; } //используется автоматическое свойство
 }
class Program
 {
  static void Main(string[] args)
   {
    Student st1 = new Student();
    //при доступе к свойству используется set и get по усолчанию
    st1.Year = 0;
    Console.WriteLine(st1.Year);
    Console.ReadKey();
   }
 }
```

Автоматически реализуемые свойства используются в случае, когда на значения поля свойства не требуется установки дополнительных ограничений.

Практические задания и методические указания

1. Изучить теоретические сведения.

2. Выполнить индивидуальное задание согласно варианту и модифицировать приложение для демонстрации работы.

3. В консоль выводить имя и время вызова свойства.

4. Создать отчет, включив в него блок-схему разработанного алгоритма, ответы на контрольные вопросы.

Индивидуальные задания

1. Для класса Point разработать следующие элементы:
   - Свойства:
     - получить-установить координаты точки (доступное для чтений и записи);
     - позволяющие умножить координаты точки на скаляр (доступное только для записи).
2. Для класса Rectangle разработать следующие элементы:
   - Свойства:
     - получить-установить стороны прямоугольника (доступное для чтений и записи);
     - позволяющие умножить стороны на число (доступное только для записи).
3. Для класса Complex разработать следующие элементы:
   - Свойства:
     - получить-установить составные части комплекссного числа (доступное для чтений и записи);
     - позволяющие умножить мнимую часть а число (доступное только для записи).
4. Для класса Triangle разработать следующие элементы:
   - Свойства:
     - получить-установить стороны треугольника (доступное для чтений и записи);
     - позволяющие умножить стороны на число (доступное только для записи).
5. Для класса Quadratic разработать следующие элементы:
   - Свойства:
     - получить-установить коэффициенты уравнения (доступное для чтений и записи);
     - позволяющие умножить первый и второй коэффиуиенты на число (доступное только для записи).
6. Для класса Circle разработать следующие элементы:
   - Свойства:
     - получить-установить радиус и координаты центра (доступное для чтений и записи);
     - позволяющие умножить радиус на число (доступное только для записи).
7. Для класса Vector разработать следующие элементы:
   - Свойства:
     - получить-установить начало внектора (доступное для чтений и записи);

- позволяющие умножить вектор на скаляр (доступное только для записи).

8. Для класса Ellipce разработать следующие элементы:
   - Свойства:
     - получить-установить радиусы эллипса (доступное для чтений и записи);
     - позволяющие умножить меньший радиус на число (доступное только для записи).
9. Для класса Linear разработать следующие элементы:
   - Свойства:
     - получить-установить коэффициенты уравнения (доступное для чтений и записи);
     - позволяющие умножить первый коэффициент на число (доступное только для записи).
10. Для класса Figure разработать следующие элементы:
    - Свойства:
      - получить-установить колличество сторон (доступное для чтений и записи);
      - позволяющие умножить длину стороны на число (доступное только для записи).

Контрольные вопросы

1. Что означает аббревиатура CLR?
2. Приведите синтаксис объявления нового пространства имён.
3. Приведите обобщённый синтаксис объявления и инициализации строковой переменной.
4. Что понимается под областью видимости переменной?
5. Благодаря чему можно избежать коллизий имён переменных?

Лабораторная работа №13
Организация подпрограмм в C#

Цель работы: сформировать знания разработки пользовательских подпрограмм. Сформировать практические навыки составления и использования подпрограмм для упрощения отладки и возможной модификации программ.

Аппаратное, программное обеспечение: персональный компьютер, Visual Studio.

Краткие теоретические сведения

В C# в роли подпрограмм выступают методы, вызываемые из других методов. В качестве примера использования подпрограмм можно рассмотреть метод *Main()* (точка входа в программу), который вызывает другие методы этого же класса (подпрограммы).

Примеры методов:

```
static void Method01() // статический метод, не возвращающий значений
{
   Console.WriteLine("Hell to World ");
}
string Method02(int n) //метод, возвращающий строку
{
   return "Number of Items = " + n;
}
```

Если метод не возвращает значения вызывающей ее программе, то в качестве типа значений указывается *void* (пустой тип). Иначе нужно указывать тип возвращающего значения и обязательно использовать оператора *return* в теле метода. Если метод возвращает значение – оно может быть выведено на экран или присвоено переменной/свойству с сопоставимым типом данных.

Вызов методов

Чтобы вызвать метод в программе необходимо указать его имя и значения параметров в скобках. Существует два способа передачи параметров:

1. по значению;
2. по ссылке.

При передаче параметров по значению, на место параметров обязательно передавать значения соответствующих типов.

Пример:

```
static void Main(string[] args)
{
   int x = 10, y = 15;
   Addition(x, y); // вызов метода
   Console.WriteLine(x); // результат равен 10
   Console.ReadLine();
}
// определение метода
static void Addition(int x, int y)
{
   x += y;
}
```

При передаче параметра по значению метод получает не саму переменную,

а её копию. Следовательно, модификации, выполняемые в методе, не повлияют на исходную переменную.

Передача параметров по ссылке

При передаче параметров по ссылке параметры передаются с модификатором *ref* (указывается как при объявлении, так и при вызове метода). Такой вариант передачи данных позволяет методу получить адрес переменной в памяти. И, если в методе изменяется значение параметра, передаваемого по ссылке, то также изменяется и значение переменной, которая передается на его место.

Пример:

```
static void Main(string[] args)
{
    int x = 10, y = 15;
    Addition(ref x, y); // вызов метода
    Console.WriteLine(x); // результат равен 25
    Console.ReadLine();
}
// определение метода
static void Addition(ref int x, int y)
{
    x += y;
}
```

При работе с параметрами, передаваемыми по ссылке, для возвращения значения можно использовать выходные параметры с модификатором out (указывается как при объявлении, так и при вызове метода). В данном случае отпадает необходимость использования параметра return.

Пример:

```
static void Main(string[] args)
{
    int x = 10, y = 15, z;
    Addition(x, y, out z); // вызов метода
    Console.WriteLine(z); // результат равен 25
    Console.ReadLine();
}
// определение метода
static void Addition(ref int x, int y, out int sum)
{
    sum = x + y;
}
```

Оптимизация программ

Высокая производительность программ достигается за счёт оптимизации кода, для достижения желаемого результата наиболее эффективным путём.

Пример:

Неоптимизированный код	Оптимизированный код

`void Method01(int a, int b)`
`{`
`  int a_ = a;`
	`void Method01(int a, int b)`
	`{`
	`Console.WriteLine("{0} + {1} = {2}", a,`

int b_ = b; *int c_ = a + b;* *Console.WriteLine("{0} + {1} = {2}", a_, b_, c_);* *}*	*b, a+b);* *}*

Главная цель оптимизации программы – получение лёгкой для понимания программы с простой структурой.

Разработку программы и каждого её метода стоит начинать с определения исходных данных и итоговых результатов, следующий шаг – определение последовательности действий, с помощью которых будет достигнут ожидаемый результат. Имена методов и переменных должны отражать их смысл.

После написания программы её следует отредактировать, добавив комментарии к основным блокам и отступы между структурными частями.

Порядок выполнения работы

1. Изучить теоретические сведения по теме «Подпрограммы в C#.».
2. Выполнить индивидуальное заданий согласно варианту и модифицировать приложение для демонстрации работы.
3. В консоль выводить имя и время вызова метода.
4. Создать отчет, включив в него блок-схему разработанного алгоритма, ответы на контрольные вопросы.

Индивидуальные задания

1. Для класса Point разработать следующие элементы:
   - Методы:
     - вывести координаты точки на экран;
     - рассчитать расстояние от начала координат до точки;
     - переместить точку на плоскости на вектор (a, b).
2. Для класса Rectangle разработать следующие элементы:
   - Методы:
     - вывести длину сторон прямоугольника на экран;
     - рассчитать площадь прямоугольника;
     - поменять размеры сторон прямоугольника на заданные.
3. Для класса Complex разработать следующие элементы:
   - Методы:
     - вывести комплексное число на экран;
     - рассчитать комплексно сопряжённое число;
     - возвести комплексное число в квадрад.
4. Для класса Triangle разработать следующие элементы:
   - Методы:
     - вывести длину сторон треугольника на экран;
     - рассчитать площадь треугольника;
     - поменять размеры сторон треугольника на заданные.
5. Для класса Quadratic разработать следующие элементы:
   - Методы:
     - вывести квадратное уравнение на экран;
     - рассчитать корни квадратного уравнения;
     - установисть третий коэффициент равным нулю.
6. Для класса Circle разработать следующие элементы:

- Методы:
  - вывести радиус и точку центра на экран;
  - рассчитать длину дуги;
  - сместить центр окружности по оси Ox.

7. Для класса Vector разработать следующие элементы:
   - Методы:
     - вывести вектор на экран;
     - рассчитать координаты конца вектора;
     - сместить вектор по оси Oy.
8. Для класса Ellipce разработать следующие элементы:
   - Методы:
     - вывести радиусы на экран;
     - рассчитать площадь фигуры;
     - увеличить стороны в два раза.
9. Для класса Linear разработать следующие элементы:
   - Методы:
     - вывести линейное уравнение на экран;
     - найти решение линейного уравнения;
     - установисть третий коэффициент равным нулю.
10. Для класса Figure разработать следующие элементы:
    - Методы:
      - вывести параметры фигуры на экран;
      - рассчитать периметр фигуры;
      - увеличить стороны в два раза.

Контрольные вопросы

1. Что такое подпрограмма в C#?
2. Как вызвать метод, возвращающий значеине?
3. Какие есть способы передачи параметров метода?
4. Для чего используется модификатор ref?
5. Для чего необходима оптимизация программы?

Лабораторная работа №14
Использование структурных типов данных

Цель работы: познакомиться с понятие структуры как способа описания объектов и создания типизированных файлов, применить полученные навыки на практике.

Аппаратное, программное обеспечение: персональный компьютер, Visual Studio.

Краткие теоретические сведения

Структуры являются одними из основных форм данных в языках программирования высокого уровня. Понятие структуры используется при машинной обработке различных документов, таблиц, баз данных. Запись – это структура, состоящая из фиксированного числа компонент, называемых полями.

В одном поле данные имеют один и тот же тип, а в разных полях могут иметь разные типы, за исключением функций:

```
 struct {
type1 id11, id12,.., id1n;
type2 id21, id22,.., id1m;
…….
typei idk1, idk2,.., idkp; } описатель [описатель];
```

Здесь idij — идентификаторы полей; typei — типы полей; описатель — имя переменной с заданной структурой.

Пример описания переменных date1 и date2 (каждая переменная содержит два поля):

```
struct date {
int year;
short day; } date1, date2;
```

Для описания структуры удобно использовать шаблоны. Описание шаблона идет без последующего списка переменных.

Формат шаблона следующий:

```
struct имя_типа_структуры {
список описаний;
};
```

Описание шаблона является описанием нового типа данных.

Далее можно описывать переменные, используя имя шаблона. Пример описания шаблона для даты (день, месяц, год):

```
struct data{
int day;
char [] month;
int year; }; struct data d1, d2, d3;//описание переменных
```

Структура не может содержать в качестве элемента структуру такого же типа, но может включать указатель на структуру этого типа, при условии, что в объявлении структуры указано имя типа. Это позволяет создавать связанные списки структур.

Пример описания узла бинарного дерева:

```
struct tree {
```

```
int number;
struct tree left;
struct tree right;
};
```

Доступ к элементу структуры осуществляется с помощью символа ‘.’, обозначающего операцию получения элемента структуры.

Примеры обращения к элементам структур, описанных выше:

```
d1.day
d2.year
```

Пример1: создание структуры, описывающей книги

```
 struct Book{
 string author;
 string title;
 int year;
public void set_book(string a, string t, int y){
author=a;
title=t;
year=y;
}
public void input(){
 Console.WriteLin("Введите автора:");
author=Console.ReadLine();
 Console.WriteLine("Введите название:");
 title=Console.ReadLine();
 Console.WriteLine("Введите год:");
 year=Convert.ToInt32(Console.ReadLine ());
 }

public void print(){
 Console.WriteLine("СВЕДЕНИЯ О КНИГЕ");
 Console.WriteLine("Автор: "+author);
 Console.WriteLine("Название: "+title);
 Console.WriteLine("Год: "+year);
 }
}
```

Использовать заданный структурный тип данных можно с предварительной установкой параметров или с вводом данных с клавиатуры:

```
static void Main (string [] args){
 Book b=new Book();
 b.input();//ввод данных с клавиатуры
 b.print();
 Console.ReadLine();
}
```

Либо:

```
 static void Main(string [] args){
 Book b=new Book();
```

```
b.set_book("Толстой Л.Н","Война и мир",1900);
b.print();
Console.ReadLine();
}
```

Можно также использовать массив структур:

```
static void Main(string[] args){
 int n=3; //количество книг
 int i;
 Book [] b=new Book[n];
 for(i=0;i<n;i++){
 b[i].input ();   }
 for(i=0;i<n;i++){
 b[i].print();
 }
 Console.ReadLine();
 }
```

Порядок выполнения:

1) Ознакомиться с теоретическими сведеньями.

2) Разобрать и выполнить пример с тремя различными способами вызова структур.

3) Создать отчет, включив в него описание работы по пункту 2, результаты выполнения заданий для индивидуальной работы согласно варианту и ответы на контрольные вопросы.

Задание для индивидуальной работы:

Создать программу, задающую структурный объект согласно варианту. Определить для выбранной структуры поля (не менее 5), среди которых есть данные разного типа (например, типы string, int, float, char). Внутри описания структуры создать 2 обязательных метода: заполнение полей структуры и вывод полей структуры – и 1 метод, характерный для данного объекта.

Вариант1: структурный объект Фильм.
Вариант2: структурный объект Группа студентов.
Вариант3: структурный объект Автомобиль.
Вариант4: структурный объект Животное.
Вариант5: структурный объект Лекарство.
Вариант6: структурный объект Игрушка.
Вариант7: структурный объект Книга.
Вариант8: структурный объект Блюдо.
Вариант9: структурный объект Учебное заведение.
Вариант10: структурный объект Семья.

Контрольные вопросы:

1. Что такое структурный тип данных?
2. Приведите шаблон описания структур.
3. Как получить доступ к полям структуры?
4. Как создать типизированный файл?

Лабораторная работа №15
Использование коллекций в C#

Цель работы: сформировать знание о структуре коллекций в C#. Сформировать навык разработки приложений с использованием коллекций.

Аппаратное, программное обеспечение: персональный компьютер, Visual Studio.

Краткие теоретические сведения

Коллекции удобно использовать при создании группы связанных объектов, в случае если нам не известно количество объектов, которые нужно хранить.

Основные классы коллекций содержатся в следующих пространствах имён:

1. *System.Collections* – простые необобщенные классы коллекций;
2. *System.Collections.Generic* – обобщенные или типизированные классы коллекций;
3. *System.Collections.Specialized* – специальные классы коллекций.

Коллекции предназначены для стандартизированной обработки групп объектов в программе. Под стандартизированной обработкой понимается использование таких известных динамических структур данных как: стеки, очереди, линейные списки, хеш-таблицы. Кроме того, в коллекциях обеспечиваются распространенные операции обработки массивов данных, например, сортировка.

Создание коллекций

Необобщенные или простые коллекции определены в пространстве имен *System.Collections*, их функциональные возможности описываются в интерфейсах, которые также находятся в этом пространстве имен.

К необобщённым коллекциям относится класс *ArrayList*, представляющий собой простые коллекции объектов, используется если нужно хранить вместе объекты разного типа (числа, строки и т.д.).

Классы обобщенных коллекций находятся в пространстве имен *System.Collections.Generic*. *List<T>* относится к таким обобщённым классам и представляет простейший список однотипных объектов, где *T* – это тип хранимых объектов.

Объявление и инициализация коллекции выполняется с использованием конструктора.

Пример:

```
// необобщенная коллекция ArrayList
    ArrayList list = new ArrayList();
// необобщенная коллекция ArrayList с созданием элемента
    ArrayList objectList = new ArrayList() { 1, 2, "string", 'c', 2.0f };
// обобщенная коллекция List с созданием элемента
List<string> countries = new List<string>() { "Россия", "США", "Великобритания",
"Китай" };
```

Добавление данных в коллекцию

Добавление значений в коллекцию выполняется последовательно с использованием двух методов:

1. *Add* – добавляет одиночный объект,

2. *AddRange* – добавляет набор объектов (массив, коллекция).

Пример добавления данных в список:

```
list.Add(2.3); // заносим в список объект типа double
list.Add(55); // заносим в список объект типа int
// заносим в список строковый массив
list.AddRange(new string[] { "Hello", "world" });
```

Для обобщенной коллекции *List* доступно добавление данных через метод *Insert* с указанием индекса вставки:

countries.Insert(0, "Беларусь");

Перебор коллекции

Через циклы *for* или *foreach* можно пройтись по всем объектам, хранящимся в списке. Число элементов коллекции можно получить через свойство *Count*. В качестве типа перебираемых объектов можно использовать тип *object*, это позволит получать разнородные объекты из списка.

Пример:

```
// перебор значений с использованием for
for (int i = 0; i < list.Count; i++)  Console.WriteLine(list[i]);
// перебор значений с использованием foreach
foreach (object o in list)  Console.WriteLine(o);
```

Удаление данных из коллекции

Метод *Remove()* удаляет первый элемент, который встретился с заданным значением, *RemoveAll()* удаляет все элементы:

list.Remove(55); //удаление элемента со значением 55

Метод *RemoveAt()* удаляет элемент в заданной позиции:

list.RemoveAt(1); //удаление второго элемента

Чтобы из списка удалить диапазон элементов, нужно использовать метод *RemoveRange()*. Метод принимает два параметра. Первый параметр – индекс, из которого начинается удаление. Второй параметр – количество элементов, которые удаляются:

list.RemoveRange(2,2); //удаление двух элементов начиная со второго

Метод *Clear()* удаляет все элементы из коллекции.

list.Clear();

Использование индексаторов

Индексаторы позволяют создавать специальные коллекции объектов и использовать их как массивы. По своей форме индексаторы напоминают свойства с методами *get* и *set*, позволяющими установить и вернуть значения.

Пример:

```
class Vector //класс с одномерным массивом
{
   private int[] numbers = new int[,] {1, 2, 4};
   public int this[int i] //индексатор с одним индексом
   {
      get //метод get для получения данных
      {  return numbers[i];}
      set //метод set для добавления новых данных
      {  numbers[i] = value;}
   }
```

```
}
class Matrix //класс с двумерным массивом
{
   private int[,] numbers = new int[,] { { 1, 2, 4}, { 2, 3, 6 }, { 3, 4, 8 } };
   public int this[int i, int j]
   {
      get
      {  return numbers[i,j];}
      set
      {  numbers[i, j] = value;}
   }
}
```

После объявления индексаторов доступ к объектам класса должен реализовываться как к массиву.

Пример:

```
Vector vector= new Vector();
Matrix matrix = new Matrix();
vector[0] = 1;
matrix[0, 0] = 111;
Console.WriteLine("vector[0, 0] = {0}", vector[0]);
Console.WriteLine("matrix[0, 0]", matrix[0, 0]);
```

Порядок выполнения работы

1. Изучить теоретические сведения по теме «Коллекции».
2. Для класса, реализованного в лабораторной работе «Использование структурных типов данных» создать коллекцию, хранящию экземляры разработанного класса, добавить индексаторы.
3. Модифицировать приложение для демонстрации работы
4. Создать отчет, включив в него блок-схему разработанного алгоритма, ответы на контрольные вопросы.

Контрольные вопросы

1. Какое главное преимущество коллекций?
2. Какими типами данных оперируют необобщенные коллекции?
3. В каком пространстве имен объявлены специальные коллекции?
4. Приведите пример пербора данных из коллекции.
5. Для чего используются индексаторы?

Лабораторная работа №16
Использование событий

Цель работы: сформировать понятие о расширенных возможностях языка C#. Сформировать навык управления событиями средствами C#.

Аппаратное, программное обеспечение: персональный компьютер, Visual Studio.

Краткие теоретические сведения

События основаны на модели делегата. Модель делегата соответствует шаблону разработки наблюдателя, который позволяет подписчику зарегистрироваться у поставщика и получать от него уведомления. Отправитель события отправляет уведомление о событии, а приемник событий получает уведомление и определяет ответ на него.

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

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

delegate int Operation(int x, int y);

delegate void GetMessage();

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. Также делегаты могут быть параметрами методов.

Наиболее сильная сторона делегатов состоит в том, что они позволяют создать функционал методов обратного вызова, уведомляя другие объекты о произошедших событиях.

Многоадресность – это способность делегата хранить несколько ссылок на различные методы, что позволяет при вызове делегата инициировать эту цепочку методов. Для создания цепочки методов необходимо создать экземпляр делегата, и пользуясь операторами + или += добавлять методы к цепочке. Для удаления метода из цепочки используется оператор - или -=. Делегаты, хранящие несколько ссылок, должны иметь тип возвращаемого значения void.

Использование делегатов можно описать четырьмя шагами.

– Определение объекта делегата с сигнатурой, точно соответствующей сигнатуре метода, который пытаемся связать.

– Определение метода с сигнатурой делегата, определенного на первом шаге

– Создание объекта делегата и связывание их с методами

– Вызов связанного метода с помощью объекта делегата

Пример1. (показывает шаги в реализации одного делегата и четырех классов)

namespace ConsoleApplication1{

```
// Шаг 1. Определение делегата с сигнатурой связываемого метода
public delegate void MyDelegate(string input);
// Шаг 2. Определение метода с сигнатурой определенного делегата
class MyClass1{
   public void delegateMethod1(string input){
      Console.WriteLine("This is delegateMethod1 and the input to the method is {0}",input);
   }
   public void delegateMethod2(string input){
      Console.WriteLine("This is delegateMethod2 and the input to the method is {0}",input);
   }
}
// Шаг 3. Создание объектов делегата и связывание с методами
class MyClass2{
   public MyDelegate createDelegate(){
      MyClass1 c2 = new MyClass1();
      MyDelegate d1 = new MyDelegate(c2.delegateMethod1);
      MyDelegate d2 = new MyDelegate(c2.delegateMethod2);
      MyDelegate d3 = d1 + d2;
      return d3;
   }
}
// Шаг 4. Вызов метода с помощью делегата
class MyClass3{
   public void callDelegate(MyDelegate d, string input){
      d(input);
   }  }
class Driver{
   static void Main(string[] args){
      MyClass2 c2 = new MyClass2();
      MyDelegate d = c2.createDelegate();
      MyClass3 c3 = new MyClass3();
      c3.callDelegate(d, "calling the delegate");
      Console.ReadKey();
   }
}
}
```

Однако C# для той же цели предоставляет более удобные и простые конструкции под названием события, которые сигнализируют системе о том, что произошло определенное действие.

Событие – это сообщение, посланное объектом, чтобы сообщить о совершении действия.Это действие может быть вызвано взаимодействием с пользователем, например, при нажатии кнопки, или другой логикой программы, например, изменением значения свойства.

В контексте C# событие - это способ, с помощью которого один класс оповещает другой (другие) класс о чем-то произошедшем. Иногда говоря, что механизм событий использует идеологию «публикация/подписка». Какой-то класс публикует свои события, а другие классы подписываются на те события, которые им интересны.

Объект, вызывающий событие, называется отправителем событий. Отправителю событий не известен объект или метод, который будет получать (обрабатывать) созданные им события.Обычно событие является членом отправителя событий; например, событие Click — член класса Button, а

событие PropertyChanged — член класса, реализующего интерфейс INotifyPropertyChanged.

События объявляются в классе с помощью ключевого слова event, после которого идет название делегата:

public delegate void MyEventHandler(object sender, MyEventArgs e);

Первый параметр (sender) определяет объект, который издает событие. Второй параметр (e) содержит данные, которые должны быть использованы обработчиком события. Класс MyEventArgs должен быть производным от класса EventArgs. EventArgs является базовым классом для более специализированных классов, таких как MouseEventArgs, ListChangedEventArgs и т.д.

Для GUI события можно применять объекты этих специализированных классов без создания своего собственного. Однако, для остальных событий необходимо создать свой класс и держать в нем данные, которые нужно передать делегату.

Пример2.

```
namespace ConsoleApplication1{
   // Шаг 1 Определение делегата с сигнатурой связываемого метода
   public delegate void MyHandler1(object sender, MyEventArgs e);
   public delegate void MyHandler2(object sender, MyEventArgs e);
   // Шаг 2 Определение метода с сигнатурой определенного делегата
   class A{
      public const string m_id = "Class A";
      public void OnHandler1(object sender, MyEventArgs e){
         Console.WriteLine("I am in OnHandler1 and MyEventArgs is {0}",e.m_id);
      }
      public void OnHandler2(object sender, MyEventArgs e){
         Console.WriteLine("I am in OnHandler2 and MyEventArgs is {0}",e.m_id);
      }
      // Шаг 3 Создание объектов делегата и связывание с методами
      public A(B b){
         MyHandler1 d1 = new MyHandler1(OnHandler1);
         MyHandler2 d2 = new MyHandler2(OnHandler2);
         b.Event1 += d1;
         b.Event2 += d2;
      }
   }
   // Шаг 4 Вызов метода с помощью делегата
   class B{
      public event MyHandler1 Event1;
      public event MyHandler2 Event2;
      public void FireEvent1(MyEventArgs e){
         if (Event1 != null) {
            Event1(this, e);
         }
      }
      public void FireEvent2(MyEventArgs e){
         if (Event2 != null) {
            Event2(this, e);
         }
      }
   }
// наследование класса EventArgs
   public class MyEventArgs : EventArgs{
```

```
    public string m_id;
  }
  public class Driver{
    public static void Main(){
      B b = new B();
      A a = new A(b);
      MyEventArgs e1 = new MyEventArgs();
      MyEventArgs e2 = new MyEventArgs();
      e1.m_id = "Event args for event 1";
      e2.m_id = "Event args for event 2";
      b.FireEvent1(e1);
      b.FireEvent2(e2);
      Console.ReadKey();
    }
  }
}
```

Пример3.

Первый класс будет считать до 100. Два других класса будут ждать, когда в первом классе счетчик досчитает до 14, и после этого каждый вызовет свой метод.

```
namespace ConsoleApplication1
{
  class ClassCounter  // класс, в котором производится счет.
  {
    public delegate void MethodContainer();
    //Событие OnCount с типом делегата MethodContainer.
    public event MethodContainer onCount;
    public void Count() {
      for (int i = 0; i < 100; i++){
        if (i == 14){
          if (onCount != null){
            onCount();}
        }  }  }  }
  class Handler_I //Это первый класс, реагирующий на событие (счет равен 14) записью строки в консоли.
  {
    public void Message(){
      Console.WriteLine("Пора действовать, ведь уже 14!");
    }
  }
  class Handler_II //Это второй класс, реагирующий на событие (счет равен 14) записью строки в консоли.
  {
    public void Message(){
      Console.WriteLine("Точно, уже 14!");
    }
  }
  class Program{
    static void Main(string[] args){
      ClassCounter Counter = new ClassCounter();
      Handler_I Handler1 = new Handler_I();
      Handler_II Handler2 = new Handler_II();
      //Подписка на событие
      Counter.onCount += Handler1.Message;
      Counter.onCount += Handler2.Message;
```

```
            //Запуск счетчика
            Counter.Count();
            Console.ReadKey();
        }
    }
}
```

Порядок выполнения работы

1. Изучить теоретические сведения по теме «Делегаты, лямбда выражения, события».

2. Для класса, реализованного в лабораторной работе 15 создать события, реагирующие на:

– добавление данных в коллекцию;

– удаление данных из коллекции.

3. В консоль выводить информацию о срабатывании события.

4. Модифицировать приложение для демонстрации работы

5. Создать отчет, включив в него блок-схему разработанного алгоритма, ответы на контрольные вопросы.

Контрольные вопросы

1. Приведите общую форму записи делегата.
2. Дя чего используются делегаты?
3. Какие типы лямбда-выражений существуют?
4. Для чего используется командное сова event?
5. Что такое обработчик событий?

Лабораторная работа №17
Организация наследования

Цель работы: сформировать знание о механизмах и особенностях наследования и полиморфизма. Сформировать навык использования механизма наследования при разработке приложений.

Аппаратное, программное обеспечение: персональный компьютер, Visual Studio.

Краткие теоретические сведения

Наследование (inheritance) позволяет создавать новый класс (*наследник*) на базе другого (*базового*) класса с сохранением полей, свойств и функциональности базового класса. Единственное, что не передается при наследовании, это конструкторы базового класса.

Объявление нового класса наследника выглядит следующим образом:

```
class [имя_класса] : [имя_базового_класса]
{
  // тело класса
}
```

где:

имя_класса – имя класса наследника,

имя_базового_класса – имя класса родителя (базового класса).

Пример использования наследования:

```
class Animal {
    public string Name { get; set; }
}
class Dog : Animal {
    public void Guard() {
        Console.WriteLine("{0} охраняет дом", Name);
    }
}
class Cat : Animal {
    public void CatchMouse() {
        Console.WriteLine("{0} ловит мышей", Name);
    }
}
class Program {
    static void Main(string[] args) {
        Dog dog = new Dog();
        dog.Name = "Барбос"; // называем пса
        Cat cat = new Cat();
        cat.Name = "Барсик"; // называем кота
        dog.Guard(); // отправляем пса охранять
        cat.CatchMouse(); // отправляем кота на охоту
    }
}
```

Объекты *Dog* и *Cat* также являются и объектами *Animal*, поэтому мы имеем право объявлять следующие переменные:

Animal dog = new Dog();

Animal cat = new Cat();

Язык C# не поддерживает множественное наследование. Это значит, что у класса наследника может быть только один базовый класс.

Модификаторы доступа

При создании класса наследника учитывается модификатор доступа к базовому классу, к его полям и методам.

В C# существуют следующие модификаторы доступа:

– ***public***: публичный, общедоступный класс или член класса. Такой член класса доступен из любого места в коде, а также из других программ и сборок.

– ***private***: закрытый класс или член класса. Представляет полную противоположность модификатору public. Такой закрытый класс или член класса доступен только из кода в том же классе или контексте.

– ***protected***: такой член класса доступен из любого места в текущем классе или в производных классах. При этом производные классы могут располагаться в других сборках.

– ***internal***: класс и члены класса с подобным модификатором доступны из любого места кода в той же сборке, однако он недоступен для других программ и сборок (как в случае с модификатором public).

– ***protected internal***: совмещает функционал двух модификаторов. Классы и члены класса с таким модификатором доступны из текущей сборки и из производных классов.

– ***private protected***: такой член класса доступен из любого места в текущем классе или в производных классах, которые определены в той же сборке.

Важно помнить, что тип доступа к классу наследнику может быть таким же, как у базового класса, или более строгим.

Например, если базовый класс объявлен с модификатором *internal*, то класс наследник может быть либо *internal*, либо *private*, но не может быть *public*.

Пример доступа к членам базового класса из класса-наследника:

```
class Animal {
    privat string privatname;
    public string Name {
        get {return privatname;};
        set {privatname = value;};
    }
}
class Dog : Animal {
    public void Guard() {
        Console.WriteLine("{0} охраняет дом", Name);
    }
}
class Program {
    static void Main(string[] args) {
```

```
        Dog dog = new Dog();
        //недопустимо и приводит к ошибке
        // privatname скрытое поле
        // dog. privatname = "Барбос";
        //доступ к скрытому полу может быть реализован через
        //открытое свойство или метод
        dog.Name = "Барбос"; // называем пса
        dog.Guard(); // отправляем пса охранять
    }
}
```

Класс наследник может иметь доступ только к тем членам базового класса, для которых определены модификаторы доступа *public*, *internal*, *protected* и *protected internal*.

Если класс был объявлен с модификатором *sealed*, то наследование от него запрещено. Например, у следующего класса не может быть наследников:

```
sealed class Admin
{
   //тело класса
}
```

Вызов базового метода

Для доступа к членам базового класс из класса наследника используется ключевое слово base, позволяющее:

– вызывать методы базового класса, переопределённые в наследнике другим методом;

– определять конструктор базового класса, который будет вызываться при создании экземпляра производного класса.

При использовании слова base сначала будет выполнен метод базового класса, а затем метод класса наследника.

Использовать ключевое слово *base* в статическом методе недопустимо.

Пример:

```
class Animal {
    privat string privatname;
    //конструктор базового класса
    public Anumal (string privatname) {
        Name = privatname;
    }
    public string Name {
        get {return privatname;};
        set {privatname = value;};
    }
}
class Dog : Animal {
    public string breed;
```

```
    //конструктор класса наследника вызывает конструктор базового класса
    public Dog(string breed, string privatname) : base(privatname){
        this.breed = breed;
    }
}
class Program {
    static void Main(string[] args) {
        Dog dog = new Dog("Шарпей", "Барбос");
    }
}
```

Полиморфизм

Полиморфизм – это способность обьекта использовать методы производного класса, который не существует на момент создания базового. Полиморфизм реализовывается с использованием переопределения виртуальных и абстрактных методов.

Виртуальный метод – это метод, который помечается в базовом классе ключевым словом *virtual* и можно переопредеить в классе-наследнике с модификатором *override*. Переопределенный в классе-наследнике метод должен иметь одинаковую сигнатуру с виртуальным методом базового класса.

Общая форма *объявления виртуального метода*:

[модификатор доступа] virtual [тип] [имя метода] ([аргументы])
{
// тело метода
}

Общая форма *переопределения виртуального метода*:

[модификатор доступа] override [тип] [имя метода] ([аргументы])
{
// новое тело метода
}

Пример переопределения виртуального метода:

```
class Person {
    public string Name { get; set; }
    public Person(string name) {
        Name = name;
    }
    //объявляем виртуальный метод,
    //который будет переопределён в классе-наследнике
    public virtual void Display() {
        Console.WriteLine($"Hello, {Name}");
    }
}
class Employee : Person {
    public string Company { get; set; }
    public Employee(string firstName, string lastName, string company)
    : base(firstName, lastName) {
```

```
        Company = company;
    }
        public override void Display() {
        Console.WriteLine($"{Name} работает в {Company}");
    }
}
static void Main(string[] args) {
    Person p1 = new Person("Bill");
    p1.Display(); // вызов метода Display из класса Person
    Employee p2 = new Employee("Tom", "Microsoft");
    p2.Display(); // вызов метода Display из класса Employee
    Console.ReadKey();
}
```

Абстрактный метод – это метод, который обязательно должен быть реализован в классе-наследнике. В отличии от виртуального метода, абстрактный не может иметь своей реализации в базовом классе. Абстрактые методы, а также абстрактые классы определяются с модификатором доступа *abstract.*

Пример объявления абстрактного класса:

```
abstract class Person {
    //свойства абстрактного класса
    public string FirstName { get; set; }
    public string LastName { get; set; }
    //конструктор абстрактного класса
    public Person(string name, string surname) {
        FirstName = name;
        LastName = surname;
    }
    //абстрактный метод
    public abstract void Display();
}
```

Конструктор абстрактного класса не может быть использован для создания объектов этого класса, но может быть использован в классе-наследнике.

Пример реализации класса-наследника для абстрактного класса:

```
class Client : Person {
    public int Sum { get; set; }    // сумма на счету
    public Client(string name, string surname, int sum) : base(name, surname) {
        Sum = sum;
    }
    public override void Display() {
Console.WriteLine($"{FirstName} {LastName} имеет счет на сумму {Sum}");
    }
}
```

Интерфейс

Интерфейс (interface) представляет собой именованный набор абстрактных членов, он объявляется за пределами класса, с использованием ключевого слова interface:

interface ISomeInterface

{
  // тело интерфейса
}

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

Рассмотрим пример объявления интерфейса:

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

Реализация интерфейса должна производиться в классах. Чтобы указать, что класс реализует интерфейс, необходимо указать имя интерфейса после двоеточия:

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

Класс, реализующий интерфейс, обязательно должен реализовывать все члены интерфейса.

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

Пример. Реализация двух интерфейсов в одном классе

```
interface IDrawable {
   void Draw();
}
interface IGeometrical {
   void GetPerimeter();
   void GetArea ();
}
class Rectangle : IGeometrical, IDrawable {
   public void GetPerimeter() {
      Console.WriteLine("(a+b)*2");
   }
   public void GetArea() {
      Console.WriteLine("a*b");
   }
   public void Draw() {
      Console.WriteLine("Rectangle");
   }
}
class Circle : IGeometrical, IDrawable {
   public void GetPerimeter() {
      Console.WriteLine("2*pi*r");
   }
   public void GetArea() {
      Console.WriteLine("pi*r^2");
   }
   public void Draw() {
      Console.WriteLine("Circle");
   }
}
```

Порядок выполнения работы

1. Изучить теоретические сведения по теме «Наследование».
2. Для класса, реализованного в лабораторной работе «Исползование событий в C#»
   - описать базовый класс (возможно, абстрактный), в котором с помощью виртуальных или абстрактных методов и свойств задается интерфейс для производных классов.
   - во всех классах следует переопределить метод Equals, чтобы обеспечить сравнение значений, а не ссылок.
3. Модифицировать приложение для демонстрации работы
4. При срабатывании переопределённого метода Equals выводить в консоль наименование класса-источника.
5. Создать отчет, включив в него блок-схему разработанного алгоритма, ответы на контрольные вопросы.

Контрольные вопросы

1. Что такое наследование?Как реализовывается полиморфизм в C#?
2. Чем отличаются виртуальные и абстрактные методы?
3. Как вызвать метод базового класса в классе-наследнике?
4. Что такое интерфейс в C#

Лабораторная работа №18
Работа с файлами

Цель работы: сформировать знание о возможностях языка C# при работе с файлами и потоками. Сформировать практический навык работы с файловыми потоками.

Аппаратное, программное обеспечение: персональный компьютер, Visual Studio.

Краткие теоретические сведения

Для чтения и записи данных в файл, открытия и закрытия файлов в файловой системе используется класс *FileStream*, поддерживающий синхронное или асинхронное выполнение операций чтения и записи. Для работы с потоками существует библиотека *System.IO*.

Класс FileStream позволяет организовать поток для ввода/вывода информации из текстового или бинарного файла. Представляет возможности по считыванию из файла и записи в файл. Он позволяет работать как с текстовыми файлами, так и с бинарными.

Таблица 3.2 – Методы и свойства класса *FileStream*

Наименование	Описание
Length	Возвращает длину потока в байтах
Position	Возвращает текущую позицию в потоке
Read()	Возвращает количество успешно считанных байтов.
Seek()	Устанавливает позицию в потоке со смещением на количество байт
Write()	Записывает в файл данные из массива байтов

FileStream используется чаще всего при работе с бинарными файлами, имеющими определенную структуру и использует режим доступа (*FileMod*) к данным. ***FileMod*** может принимать значения:

– ***Append***: если файл существует, то текст добавляется в конец файл. Если файла нет, то он создается. Файл открывается только для записи.

– ***Create***: создается новый файл. Если такой файл уже существует, то он перезаписывается

– ***CreateNew***: создается новый файл. Если такой файл уже существует, то он приложение выбрасывает ошибку

– ***Open***: открывает файл. Если файл не существует, выбрасывается исключение

– ***OpenOrCreate***: если файл существует, он открывается, если нет - создается новый

– ***Truncate***: если файл существует, то он перезаписывается. Файл открывается только для записи.

Рассмотрим примере считывания-записи в текстовый файл с использованием класса *FileStream*.

Пример:

```
tring text = "Строка для записи в файл";
// запись в файл
//создаём новый поток fstream
using (FileStream fstream = new
      FileStream(@"C:\SomeDir\noname\note.txt", FileMode.OpenOrCreate)) {
   // преобразуем строку в байты
   byte[] array = System.Text.Encoding.Default.GetBytes(text);
   // запись массива байтов в файл
   fstream.Write(array, 0, array.Length);
}
//поток fstream автоматически закрывается после блока using
// чтение из файла
using (FileStream fstream =
      File.OpenRead(@"C:\SomeDir\noname\note.txt")) {
   // преобразуем строку в байты
   byte[] array = new byte[fstream.Length];
   // считываем данные
   fstream.Read(array, 0, array.Length);
   // декодируем байты в строку
   string textFromFile = System.Text.Encoding.Default.GetString(array);
   Console.WriteLine("Текст из файла: {0}", textFromFile);
}
```

Если во время выполнения программы поток не был закрыт, изменения вносимые в файл не сохраняются. Для явного закрытия потока используется метод Close(), например: *fstream.Close();*

Класс StreamReader используется для считывания всех текстовых данных или отдельные строки из текстового файла.

При использовании конструктора класса *StreamReader* указывается путь к файлу, из которого нужно считать данные, и, при необходимости, кодировка файла.

Таблица 3.3 – Методы класса *StreamReader*

Наименование	Описание
Close()	Закрывает считываемый файл и освобождает все ресурсы
Peek()	Возвращает следующий доступный символ, если символов больше нет, то возвращает -1
Read()	Считывает и возвращает следующий символ в численном представлении
ReadLine()	Считывает одну строку в файле
ReadToEnd()	Считывает весь текст из файла

Пример считывания *всего файла*:

```
using (StreamReader sr = new StreamReader(@"C:\SomeDir\hta.txt")) {
   Console.WriteLine(sr.ReadToEnd()); }
```

Пример считывания *файла по строкам*:

```
using (StreamReader sr = new StreamReader(@"C:\SomeDir\hta.txt")),
```

```
        System.Text.Encoding.Default)) {
    string line;
    while ((line = sr.ReadLine()) != null) {
        Console.WriteLine(line);
    }
}
```

Пример считывания *файла по блокам*:

```
using (StreamReader sr = new StreamReader(@"C:\SomeDir\hta.txt")),
        System.Text.Encoding.Default)) {
    char[] array = new char[4];
    // считываем 4 символа
    sr.Read(array, 0, 4);
    Console.WriteLine(array);
}
```

Класс StreamWriter используется для записи данных в текстовый файл.

Таблица 3.4 – Методы класса *StreamReader*

Наименование	Описание
Close()	Закрывает записываемый файл и освобождает все ресурсы
Flush()	Записывает в файл оставшиеся в буфере данные и очищает буфер
Write()	Записывает в файл данные простейших типов
WriteLine()	Записывает данные, только после записи добавляет в файл символ окончания строки

Пример записи данных в файл *с перезаписью*:

```
using (StreamWriter sw =
        new StreamWriter(@"C:\SomeDir\ath.txt", //адрес файла
        false, //файл будет очищен перед записью
        System.Text.Encoding.Default)) //кодировка по умолчанию
{
    sw.WriteLine(text);
}
```

Пример записи данных *в конец файла*:

```
using (StreamWriter sw =
        new StreamWriter((@"C:\SomeDir\ath.txt",
        true, //данные будут записаны в конец файла
        System.Text.Encoding.Default))
{
    sw.WriteLine("Дозапись");
    sw.Write(4.5);
}
```

Класс BinaryWriter предназначен для записи данных в бинарный файл.

Таблица 3.5 – Основные метода класса *BinaryWriter*

Наименование	Описание

Close()	Закрывает поток и освобождает все ресурсы
Flush()	Записывает в файл оставшиеся в буфере данные и очищает буфер
Seek()	Устанавливает позицию в потоке
Write()	Записывает данные в поток

Пример записи данных *в бинарный файл*:

```
// создаем объект BinaryWriter
using (BinaryWriter writer =
      new BinaryWriter(File.Open(path, FileMode.OpenOrCreate)))
{
   // записываем в файл значения
   writer.Write("Беларусь");
   writer.Write("Россия");
}
```

Класс StreamReader используется для считывания всех текстовых данных или отдельные строки из текстового файла.

Таблица 3.6 – Основные метода класса *BinaryReader*

Наименование	Описание
Close()	Закрывает поток и освобождает все ресурсы
ReadBoolean()	Считывает значение *bool* и перемещает указатель на один байт

Продолжение таблицы 3.6

Наименование	Описание
ReadByte()	Считывает один байт и перемещает указатель на один байт
ReadChar()	Считывает значение *char*, то есть один символ, и перемещает указатель на столько байтов, сколько занимает символ в текущей кодировке
ReadDecimal()	Считывает значение *decimal* и перемещает указатель на 16 байт
ReadDouble()	Считывает значение *double* и перемещает указатель на 8 байт
ReadInt16()	Считывает значение *short* и перемещает указатель на 2 байта
ReadInt32()	Считывает значение *int* и перемещает указатель на 4 байта
ReadInt64()	Считывает значение *long* и перемещает указатель на 8 байт
ReadSingle()	Считывает значение *float* и перемещает указатель на 4 байта
ReadString()	Считывает значение *string*. Каждая строка предваряется значением длины строки, которое представляет 7-битное целое

	число

Пример чтения данных из *бинарного файла*:

```
// создаем объект BinaryReader
using (BinaryReader reader = new BinaryReader(
      File.Open(@"C:\SomeDir\states.dat", FileMode.Open)))
{
   string name;
   // пока не достигнут конец файла
   // считываем каждое значение из файла
   while (reader.PeekChar() > -1)
   { name = reader.ReadString();
      Console.WriteLine("Страна: {0}", name);
   }
}
```

Класс FileInfo предоставляет свойства и методы для создания, копирования, удаления, перемещения и открытия файлов.

Таблица 3.7 – Методы и свойства класса *FileInfo*

Наименование	Описание
AppendText()	Создает объект *StreamWriter* и добавляет текст в файл
CopyTo()	Копирует существующий файл в новый файл
Create()	Создает новый файл и возвращает объект *FileStream* для взаимодействия с вновь созданным файлом
CreateText()	Создает объект *StreamWriter*, записывающий новый текстовый файл

Продолжение таблицы 3.7

Наименование	Описание
Delete()	Удаляет файл, к которому привязан экземпляр *FileInfo*
Directory	Получает экземпляр родительского каталога
DirectoryName	Получает полный путь к родительскому каталогу
Length	Получает размер текущего файла или каталога
MoveTo()	Перемещает указанный файл в новое местоположение, предоставляя возможность указать новое имя файла
Name	Получает имя файла
Open()	Открывает файл с различными привилегиями чтения/записи и совместного доступа
OpenRead()	Создает доступный только для чтения объект FileStream
OpenText()	Создает объект *StreamReader* и читает из существующего текстового файла
OpenWrite()	Создает доступный только для записи объект *FileStream*

Большинство методов класса *FileInfo* возвращают специфический объект

ввода-вывода (*FileStream* и *StreamWriter*), позволяющий записывать и читать данные из файла в разнообразных форматах.

Пример создания файла с использованием метода *FileInfo.Create()*:

```
static void NewFile()
{
   // Создаем новый файл
   FileInfo f = new FileInfo(@"C:\Test.dat");
   FileStream fs = f.Create();
   // Закрыть файловый поток
   fs.Close();
}
```

Рассмотрим несколько примеров работы с файлами через экземпляр класса FileInfo.

Примеры:

Пример получения информации о файле

```
string path = @"C:\apache\hta.txt";
FileInfo fileInf = new FileInfo(path);
if (fileInf.Exists) {
   Console.WriteLine("Имя файла: {0}", fileInf.Name);
   Console.WriteLine("Время создания: {0}", fileInf.CreationTime);
   Console.WriteLine("Размер: {0}", fileInf.Length);
}
```

Пример удаления файла:

```
string path = @"C:\apache\hta.txt";
FileInfo fileInf = new FileInfo(path);
if (fileInf.Exists){
   fileInf.Delete();
   // альтернатива с помощью класса File
   // File.Delete(path);
}
```

Пример получения информации о файле:

```
string path = @"C:\apache\hta.txt";
string newPath = @"C:\SomeDir\hta.txt";
FileInfo fileInf = new FileInfo(path);
if (fileInf.Exists) {
   fileInf.MoveTo(newPath);
   // альтернатива с помощью класса File
   // File.Move(path, newPath);
}
```

Пример копирования файла:

```
string path = @"C:\apache\hta.txt";
string newPath = @"C:\SomeDir\hta.txt";
FileInfo fileInf = new FileInfo(path);
if (fileInf.Exists) {
   fileInf.CopyTo(newPath, true);
   // альтернатива с помощью класса File
   // File.Copy(path, newPath, true);
}
```

Класс File предоставляет функциональность, почти идентичную типу *FileInfo*, и поддерживает методы *AppendText(), Create(), CreateText(), Open(), OpenRead(), OpenWrite()* и *OpenText()*. Кроме этого *File* также поддерживает несколько дополнительных методов.

Таблица 3.8 – Дополнительные методы *File*

Наименование	Описание
ReadAllBytes()	Открывает указанный файл, возвращает двоичные данные в виде массива байт и затем закрывает файл
ReadAllLines()	Открывает указанный файл, возвращает символьные данные в виде массива строк, затем закрывает файл
ReadAllText()	Открывает указанный файл, возвращает символьные данные в виде System.String(), затем закрывает файл
WriteAllBytes()	Открывает указанный файл, записывает в него байтовый массив и закрывает файл
WriteAllLines()	Открывает указанный файл, записывает в него массив строк и закрывает файл
WriteAllText()	Открывает указанный файл, записывает в него данные из указанной строки и закрывает файл

Порядок выполнения работы

1. Изучить теоретические сведения.
2. Для класса, реализованного в лабораторной работе «Организация наследования» реализовать вывод данных списка объектов в файл.
3. Модифицировать приложение для демонстрации работы.
4. В консоль выводить сообщения о:
   - создании или наличии в котологе файла;
   - о успешной или неуспешной записи данных в файл.
5. Создать отчет, включив в него блок-схему разработанного алгоритма, ответы на контрольные вопросы.

Контрольные вопросы

1. Для чего предназначен класс FileStream?
2. Чем отличаются классы StreamWrite и StreamRead?
3. Для чего закрывать поток при записи данных в файл?
4. Приведите пример записи информации в коне файла.\
5. Для чего предназначен класс File?

Лабораторная работа №19
Организации обработки исключений

Цель работы: сформировать знание о жизненном цикле объектов C# и обработке исключений. Сформировать практический навык по обработке исключений средствами языка C#.

Аппаратное, программное обеспечение: персональный компьютер, Visual Studio.

Краткие теоретические сведения

Исключительная ситуация или исключение – это возникновение аварийного события, которое может порождаться некорректным использованием аппаратуры или неправильной работой программы, например делением на ноль или переполнением. Обычно эти события приводят к завершению программы с системным сообщением об ошибке. C# дает программисту возможность восстановить работоспособность программы и продолжить ее выполнение. Исключения C# не поддерживают обработку асинхронных событий, таких как ошибки оборудования или прерывания, например нажатие клавиш Ctrl+C. Механизм исключений предназначен только для событий, которые могут произойти в результате работы самой программы и указываются явным образом. Исключения возникают тогда, когда некоторая часть программы не смогла сделать то, что от нее требовалось. При этом другая часть программы может попытаться сделать что-нибудь иное.

Исключения позволяют логически разделить вычислительный процесс на две части – обнаружение аварийной ситуации и ее обработка. Это важно не только для лучшей структуризации программы. Главное то, что функция, обнаружившая ошибку, может не знать, что предпринимать для ее исправления, а использующий эту функцию код может знать, что делать, но не уметь определить место возникновения. Это особенно актуально при использовании библиотечных функций и программ, состоящих из многих модулей.

Другое достоинство исключений состоит в том, что для передачи информации об ошибке в вызывающую функцию не требуется применять возвращаемое значение или параметры, поэтому заголовки функций не разрастаются.

В принципе, ничто не мешает рассматривать в качестве исключений не только ошибки, но и нормальные ситуации, возникающие при обработке данных, но это не имеет преимуществ перед другими решениями, не улучшает структуру программы и не делает её понятнее.

Исключения генерирует либо среда выполнения, либо программист с помощью оператора throw. На рисунке1 приведены наиболее часто используемые стандартные исключения, генерируемые средой. Они определены в пространстве имен System. Все они являются потомками класса Exception, а точнее, потомками его потомка SystemException.Исключения обнаруживаются и обрабатываются в операторе try.

Имя	Описание
ArithmeticException	Ошибка в арифметических операциях или преобразованиях (является предком DivideBeZeroException и OverFlowException)
ArrayTypeMismatchException	Попытка сохранения в массиве элемента несовместимого типа
DivideByZeroException	Попытка деления на ноль
FormatException	Попытка передать в метод аргумент неверного формата
IndexOutOfRangeException	Индекс массива выходит за границы диапазона
InvalidCastException	Ошибка преобразования типа
OutOfMemoryException	Недостаточно памяти для создания нового объекта
OverFlowException	Переполнение при выполнении арифметических операций
StackOverFlowException	Переполнение стека

Рисунок1 – Стандартные исключения, генерируемые средой

Иногда при выполнении программы возникают ошибки, которые трудно предусмотреть или предвидеть, а иногда и вовсе невозможно. Например, при передаче файла по сети может неожиданно оборваться сетевое подключение. такие ситуации называются исключениями. Язык C# предоставляет разработчикам возможности для обработки таких ситуаций. Для этого в C# предназначена конструкция try...catch...finally.

Оператор try

Оператор try содержит три части:

– контролируемый блок - составной оператор, предваряемый ключевым словом try. В контролируемый блок включаются потенциально опасные операторы программы. Все функции, прямо или косвенно вызываемые из блока, также считаются ему принадлежащими;

– один или несколько обработчиков исключений – блоков catch, в которых описывается, как обрабатываются ошибки различных типов;

– блок завершения finally выполняется независимо от того, возникла ошибка в контролируемом блоке или нет.

Синтаксис оператора try:

try блок [блоки catch] [блок finally]

Отсутствовать могут либо блоки catch, либо блок finally, но не оба одновременно.

Обработчики исключений должны располагаться непосредственно за блоком try. Они начинаются с ключевого слова catch, за которым в скобках следует тип обрабатываемого исключения. Можно записать один или несколько обработчиков в соответствии с типами обрабатываемых исключений. Блоки catch просматриваются в том порядке, в котором они записаны, пока не будет найден соответствующий типу выброшенного исключения.

Синтаксис обработчиков напоминает определение функции с одним параметром - типом исключения. Существуют три формы записи:

```
catch( тип имя ) { ... /* тело обработчика */  }
catch( тип ) {.../* тело обработчика */ }
catch { ... /* тело обработчика */ }
```

Первая форма применяется, когда имя параметра используется в теле обработчика для выполнения каких-либо действий, например вывода информации об исключении.

Вторая форма не предполагает использования информации об исключении, играет роль только его тип.

При возникновении исключения среда CLR ищет блок catch, который может обработать данное исключение. Если такого блока не найдено, то пользователю отображается сообщение о необработанном исключении, а дальнейшее выполнение программы останавливается. И чтобы подобной остановки не произошло, и надо использовать блок try..catch.

Третья форма применяется для перехвата всех исключений. Так как обработчики просматриваются в том порядке, в котором они записаны, обработчик третьего типа (он может быть только один) следует помещать после всех остальных.

Оператор throw

До сих пор мы рассматривали исключения, которые генерирует среда выполнения C#, но это может сделать и сам программист. Для генерации исключения используется оператор throw с параметром, определяющим вид исключения. Параметр должен быть объектом, порожденным от стандартного класса System.Exception. Этот объект используется для передачи информации об исключении его обработчику.

Оператор throw употребляется либо с параметром, либо без него:

throw[выражение];

Форма без параметра применяется только внутри блока catch для повторной генерации исключения. Тип выражения, стоящего после throw, определяет тип исключения, например:

throw new DivideByZeroException( );

Здесь после слова throw записано выражение, создающее объект стандартного класса «ошибка при делении на 0» с помощью операции new.

При генерации исключения выполнение текущего блока прекращается и происходит поиск соответствующего обработчика с передачей ему управления. Обработчик считается найденным, если тип объекта, указанного после throw, либо тот же, что задан в параметре catch, либо является производным от него.

Пример 1: Применение блока try…catch…finaly…

```
static void Main(string[] args){
   int[] a = new int[4];
   try{
      a[5] = 4; // исключение, так как в массиве только 4 элемента
      Console.WriteLine("Завершение блока try");
   }
   catch (Exception ex){
      Console.WriteLine("Ошибка: " + ex.Message);
   }
   finally{
      Console.WriteLine("Блок finally");
   }
   Console.ReadLine();
}
```

При использовании блока try...catch..finally вначале выполняются все инструкции между операторами try и catch. Если между этими операторами вдруг возникает исключение, то обычный порядок выполнения останавливается и

переходит к инструкции catch. В данном случае явно возникнет исключение в блоке try при попытке присвоить значение шестому элементу массива в то время, как в массиве всего 4 элемента. И дойдя до строки a[5] = 4;, выполнение программы остановится и перейдет к блоку catch.

В данном случае объявляется переменная ex, которая имеет тип Exception. Но если возникшее исключение не является исключением типа, указанного в инструкции catch, то оно не обрабатывается, а программа просто зависает или выбрасывает сообщение об ошибке.

Однако так как тип Exception является базовым классом для всех исключений, то выражение catch (Exception ex) будет обрабатывать практически все исключения. Вся обработка исключения в нашем случае сводится к выводу на консоль сообщении об исключении, которое в свойстве message класса Exception.

Далее в любом случае выполняется блок finally. Однако этот блок необязательный, и его можно при обработке исключений опускать. Если же в ходе программы исключений не возникнет, то программа не будет выполнять блок catch, сразу перейдет к блоку finally, если он имеется.

При необходимости можно разграничить обработку различных типов исключений путем ввода дополнительных блоков catch.

Пример 2: Пример ввода дополнительных блоков catch.

```
class Program{
    static int MyDel(int x, int y){
        return x / y;
    }
    static void Main(){
        try{
            Console.Write("Введите x: ");
            int x = int.Parse(Console.ReadLine());
            Console.Write("Введите y: ");
            int y = int.Parse(Console.ReadLine());
            int result = MyDel(x, y);
            Console.WriteLine("Результат: " + result);
        }
        // Обрабатываем исключение возникающее при делении на ноль
        catch (DivideByZeroException ex){
            Console.WriteLine("Деление на 0 detected!!!\n");
            Console.WriteLine("ОШИБКА: " + ex.Message + "\n\n");
            Main();
        }
        // Обрабатываем исключение при некорректном вводе числа в консоль
        catch (FormatException ex){
            Console.WriteLine("Это НЕ число!!!\n");
            Console.WriteLine("ОШИБКА: " + ex.Message + "\n\n");
            Main();
        }
        Console.ReadLine();
    }
}
```

Если возникает исключение определенного типа, то оно переходит к соответствующему блоку catch. При этом более частные исключения следует помещать в начале, и только потом более общие классы исключений. Например,

сначала обрабатывается исключение IOException, и только потом Exception (так как IOException наследуется от класса Exception).

Оператор throw:

Чтобы сообщить о выполнении исключительных ситуаций в программе, можно использовать оператор throw. То есть с помощью этого оператора можно создать исключение и вызвать его в процессе выполнения. Например, в программе происходит ввод строки, и по условие, когда длина строки будет больше 6 символов, возникает исключение:

Пример 3:

```
static void Main(string[] args){
try{
            string message = Console.ReadLine();
          if (message.Length > 6){
            throw new Exception("Длина строки больше 6 символов");
          }
       }
     catch (Exception e){
        Console.WriteLine("Ошибка: " + e.Message);
       }
   Console.ReadLine();
}
```

Исключения имеют следующие свойства:

– Исключения имеют типы, в конечном счете являющиеся производными от System.Exception.

– Следует использовать блок try для заключения в него инструкций, которые могут выдать исключения.

– При возникновении исключения в блоке try поток управления немедленно переходит к первому соответствующему обработчику исключений, присутствующему в стеке вызовов. В языке C# ключевое слово catch используется для определения обработчика исключений.

– Если обработчик для определенного исключения не существует, выполнение программы завершается с сообщением об ошибке.

– Не перехватывайте исключение, если его нельзя обработать, и оставьте приложение в известном состоянии. При перехвате System.Exception вновь инициируйте это исключение с использованием ключевого слова throw в конце блока catch.

– Если в блоке catch определяется переменная исключения, ее можно использовать для получения дополнительной информации о типе произошедшего исключения.

– Исключения могут явно генерироваться программной с помощью ключевого слова throw.

– Объекты исключения содержат подробные сведения об ошибке, такие как состояние стека вызовов и текстовое описание ошибки.

– Код в блоке finally выполняется, даже при возникновении исключения. Блок finally используется для освобождения ресурсов, например, для закрытия потоков или файлов, открытых в блоке try.

Порядок выполнения:

1) Ознакомиться с теоретическими сведеньями.
2) Разобрать и выполнить примеры.
3) Создать отчет, включив в него описание работы по пункту 2, результаты выполнения заданий для идивидуальной работы согласно варианту и ответы на контрольные вопросы.

Задания для индивидуальной работы:

При выполнении индивидуальных заданий для всех вариантов предусмотреть обработку как минимум трех исключительных ситуаций различными способами.

Вариант 1.

Создать файл данных, компонентами которого являются вещественные числа. Найти сумму квадратов компонентов файла. Найти частное полученной суммы квадратов и введенного пользователем числа.

Вариант 2.

Создать файл данных, компонентами которого являются целые числа. Вывести на экран все четные числа данного файла и записать их в массив. Найти в массиве элементы, которые делятся без остатка на введенное пользователем число. Вывести их на экран.

Вариант 3.

Создать файл данных, компонентами которого являются вещественные числа. Вывести на экран все числа этого файла, которые больше данного числа X и меньше данного числа У. Преобразовать полученные числа в единую строку. Вывести на экран полученную строку и её длину.

Вариант 4.

Создать файл данных, компонентами которого являются вещественные числа. Найти разность между первой и последней компонентой файла. Создать одномерный массив, размерность которого равна этой разности. Заполнить массив числами и вывести на экран.

Вариант 5.

Создать файл данных, компонентами которого являются вещественные числа. Поменять местами первую и последнюю компоненту данного файла, если их сумма больше 100. В ином случае удалить обе компоненты. Содержимое полученного файла вывести на экран.

Вариант 6.

Создать файл данных, компонентами которого являются целые числа. Заменить в этом файле все четные числа на 0, а из нечетных чисел сформировать массив. Вывести содержимое нового файла и массива на экран. Найти корень каждого элемента массива.

Вариант 7.

Создать файл данных, компонентами которого являются целые числа. Найти сумму нечетных чисел этого файла. Сформировать строку из случайных чисел, длина которой равна полученной сумме. Вывести строку на экран.

Вариант 8.

Создать файл данных, компонентами которого символы латинского алфавита (не в алфавитном порядке и не обязательно все). Поменять местами первую и 5-ую компоненты местами и записать все символы в массив. Вывести каждый третий символ и посчитать их количество.

Вариант 9.

Создать файл и заполнить его натуральными числами, которые вводит пользователь. Найти сумму введенных чисел и извлечь из нее корень. Создать массив случайных чисел, размерность которого равна рассчитанному значению.

Вариант 10.

Создать файл данных, компонентами которого являются натуральные числа. Определить количество компонент, которые делятся на 3 без остатка. Сформировать новый массив, размерность которого равна полученному количеству. Заполнить его случайными числа и проверить их на знакоположительность.

Контрольные вопросы:

1) Что такое исключение?
2) Какие стандартные исключения генерируются средой?
3) Из каких частей состоит оператор try?
4) Какие существуют формы записи обработчиков исключений?
5) Для чего применяется оператор throw?

Лабораторная работа №20
Разработка многопоточных приложений на C#

Цель работы: ознакомиться с понятием многопоточность, изучить принципы разработки многопоточных приложений в C#. Применить полученные знания на практике.

Аппаратное, программное обеспечение: персональный компьютер, Visual Studio.

Краткие теоретические сведения

Одной из очень важных особенностей языка C# является то, что он имеет встроенную поддержку многопоточного программирования. Особенность многопоточной программы в том, что она может состоять из нескольких частей, каждая из которых выполняет свою часть поставленной задачи. Таким образом, части выполняются параллельно. Такая часть программы называется потоком.

Среда .NET Framework, в свою очередь, содержит ряд классов, предназначенный для гибкой реализации многопоточных приложений. Многопоточность может быть ориентированная на потоки и процессы. Здесь важно понимать разницу, так как процесс, по сути, является отдельно выполняемой программной. Т.е. здесь многопоточность основана на том, что выполняются две или более программы.

В среде **.NET Framework** существует два типа потоков: основной и фоновый (вспомогательный). В целом отличие между ними одно – если первым завершится основной поток, то фоновые потоки в его процессе будут также принудительно остановлены, если же первым завершится фоновый поток, то это не повлияет на остановку основного потока – тот будет продолжать функционировать до тех пор, пока не выполнит всю работу и самостоятельно не остановится. Обычно при создании потока ему по умолчанию присваивается основной тип.

Стандартно в проектах Visual Studio существует только один основной поток – в методе **Main**. Всё, что в нём выполняется – выполняется последовательно строка за строкой. Но при необходимости можно “распараллелить” выполняемые процессы при помощи потоков.

Что бы начать работать с потоками, необходимо подключить пространство имен System.Threading.

Любой поток в C# это функция. Функции не могут быть сами по себе, они обязательно являются методами класса. Поэтому, чтобы создать отдельный поток, понадобится класс с необходимым методом. Самый простой вариант метода возвращает void и не принимает аргументов: void MyThreadFunction() { ... }

Пример запуска такого потока:

```
Thread thr = new Thread(MyThreadFunction);
thr.Start();
```

После вызова метода Start() у объекта потока, управление вернется сразу, но в этот момент уже начнет работать новый поток. Новый поток выполнит тело функции MyThreadFunction и завершится. После вызова Start(), управление передается дальше, при этом созданный поток может работать еще длительное

время. Что бы обмениваться данными между потоками, можно пользоваться переменными класса.

Пример 1:

```
using System;
using System.Threading;
namespace ThreadsExample{
   class Program{
      static void Main(string[] args){
         //Создание объекта потока
         Thread thread = new Thread(ThreadFunction);
         //Запускаем поток
         thread.Start();
         //Вывод 3 раза на экран заданного текста
         int count = 3;
         while (count > 0){
            Console.WriteLine("Это главный поток программы!");
            --count;    }
         Console.Read();    }
      //Функция потока
      static void ThreadFunction(){
         //Аналогично главному потоку выводит три раза текст
         int count = 3;
         while (count > 0){
            Console.WriteLine("Это дочерний поток программы!");
            --count;
         }     }
 }     }
```

У метода потока присутствует модификатор static, это нужно для того, чтобы к ней можно было напрямую обратиться из главного потока приложения. Обычно, порядок вывода сообщений в консоль при каждом запуске разный. Планировщик задач операционной системы по-разному распределяет процессорное время, поэтому порядок вывода разный.

В таблице1 приводятся методы, с помощью которых можно управлять отдельными потоками.

Таблица 1– Методы, управляющие потоками

Метод	Действие
Start()	Запускает поток.
Sleep()	Приостанавливает поток на определенное время.
Suspend()	Приостанавливает поток, когда он достигает безопасной точки.
Abort()	Останавливает поток, когда он достигает безопасной точки.
Resume()	Возобновляет работу приостановленного потока
Join()	Приостанавливает текущий поток до тех пор, пока не будет завершен другой поток. При заданном времени ожидания этот метод возвращает значение True при условии, что другой поток закончится за это время.

Когда в программе фигурирует несколько потоков, выбор процессором следующего потока для выполнения не является рандомным. Дело в том, что у каждого потока имеется значение приоритета, чем выше приоритет потока, тем важнее для процессора предоставить время и ресурсы для выполнения именно ему. Если же приоритет потока не слишком высокий, значит он может и подождать в очереди, пока выполняются более приоритетные потоки.

Всего существует пять вариантов приоритетов потоков в C#:

– Highest – самый высокий

– AboveNormal – выше среднего

– Normal – стандартный

– BelowNormal – ниже среднего

– Lowest – самый низкий

Как уже было сказано, чем выше приоритет, тем быстрее он выполнится, опережая потоки с меньшим значением. Если у потоков одинаковые приоритеты, они выполняются по очереди.

По умолчанию все создаваемые потоки в C# имеют стандартный приоритет Normal. Однако приоритеты потоков можно и менять, делается это так:

[Имя_потока].Priority = ThreadPriority.[вариант_приоритета];

Приоритет метода Main изменить нельзя. Он всегда будет в позиции Normal.

Потоки также имеют несколько полезных свойств, которые приведены в таблице2.

Таблица 2 – Свойства для работы с потоками

Свойство	Значение
IsAlive	Содержит значение True, если поток активен.
IsBackground	Возвращает или задает логическое значение, которое указывает, является ли поток (должен ли являться) фоновым потоком.
Name	Возвращает или задает имя потока. Наиболее часто используется для обнаружения отдельных потоков при отладке.
Priority	Возвращает или задает значение, используемое операционной системой для установки приоритетов потоков.
ApartmentState	Возвращает или задает потоковую модель для конкретного потока.
ThreadState	Содержит значение, описывающее состояние или состояния потока.

Пример 2: управление приоритетом потоков

```
static void mythread1(){
        for (int i = 0; i < 10; i++){
            Console.WriteLine("Поток 1 выводит " + i);
        }
    }
    static void mythread2(){
```

```
        for (int i = 10; i < 20; i++){
            Console.WriteLine("Поток 2 выводит " + i);
        }
    }
    static void mythread3(){
        for (int i = 20; i < 30; i++){
            Console.WriteLine("Поток 3 выводит " + i);
        }
    }
    static void Main(string[] args){
        Thread thread1 = new Thread(mythread1);
        Thread thread2 = new Thread(mythread2);
        Thread thread3 = new Thread(mythread3);
        thread1.Priority = ThreadPriority.Highest;
        thread2.Priority = ThreadPriority.AboveNormal;
        thread3.Priority = ThreadPriority.Lowest;
        thread1.Start();
        thread2.Start();
        thread3.Start();
        Console.ReadLine();
    }
}
```

В примере2 имеется три потока и три метода, в которых они выполняются. Первый метод выводит на экран числа от 0 до 9, второй – от 10 до 19, третий – с 20 по 29.

В методе Main задаются приоритеты потокам. Так как надо сначала вывести числа из первого потока, необходимо установить ему самый высокий приоритет. Следующий за ним второй поток имеет приоритет чуть ниже, поэтому будет выполняться вторым, третий поток имеет самый низкий приоритет, поэтому будет выполняться после всех остальных приоритетов.

При создании потока он по умолчанию становится основным. Однако его тип можно также, как и приоритет, поменять. Для того, чтобы сделать основной поток фоновым, нужно изменить его свойство IsBackground:

[имя_потока].IsBackground = true;

Если установить данное свойство в значение true, то поток будет работать как фоновый, если в значение false - как основной.

Однако, если в программе используется несколько потоков, то не всегда можно понять, какой поток к какому типу относится. Но всегда можно легко это узнать, например:

bool a = mythread.IsBackground;

Console.WriteLine(a);

Надо присвоить переменной типа bool значение данного свойства, и, если необходимо, вывести полученное значение на экран. Суть такая же: если выведется true, значит поток фоновый, иначе – основной.

Пример 3.

```
static void mythread1(){
        for (int i = 0; i < 1000000; i++){
            Console.WriteLine("Поток 1 выводит " + i);
        }
    }
```

```
    static void Main(string[] args){
      Thread thread1 = new Thread(mythread1);
      thread1.IsBackground = true;
      thread1.Start();
      Thread.Sleep(100);
    }
  }
```

В данной программе присвоивается потоку thread1 тип – фоновый, в строке:

thread1.IsBackground = true;

А так как данный поток вызывается в методе **Main**, значит он будет полностью зависеть от потока в этом методе. Поток **thread1** вызывается из метода **Main** и начинает работу в методе **mythread1**, который должен выводить на экран числа от 0 до 999999. Однако загвоздка в том, что практически после старта работа метода **Main** прекращается (нет строки **Console.ReadLine();**). Поток **thread1** никак не успеет вывести все 999999 чисел за столь короткий промежуток времени, но так как в программе он является фоновым, то принудительно завершится вместе с завершением потока в методе **Main**. В строке Thread.Sleep(100); приостанавливается приоритетный поток (Main) на 100 мс , чтобы успеть увидеть результат данной программы.

Порядок выполнения:

1) Ознакомится с теоритическими сведеньями;

2) Разобрать и выполнить примеры;

3) Создать отчет, включив в него описание работы по пункту 2, результаты выполнения заданий для идивидуальной работы согласно варианту и ответы на контрольные вопросы.

Задания для индивидуальной работы:
Задание1

Вариант1: модифицировать пример3 из данной лабораторной работы: организовать запись чисел в файл. Вывести его содержимое на экран.

Вариант2: модифицировать пример2 из данной лабораторной работы: создать массив, элементами которого будут: произведение первых 10 чисел и сумма первых 10 чисел.

Вариант3: модифицировать пример1 из данной лабораторной работы: получить максимум информации о дочернем потоке и записать её в файл.

Вариант4: модифицировать пример3 из данной лабораторной работы: организовать запись чисел в массив. Вывести его содержимое на экран.

Вариант5: модифицировать пример2 из данной лабораторной работы: записать в файл первые 10 чисел.

Вариант6: модифицировать пример1 из данной лабораторной работы: записать в коллекцию информацию об основном потоке.

Вариант7: модифицировать пример3 из данной лабораторной работы: организовать запись чисел в коллекцию. Вывести её содержимое на экран.

Вариант8: модифицировать пример2 из данной лабораторной работы: записать в коллекцию первые 10 чисел.

Вариант9: модифицировать пример1 из данной лабораторной работы: добавить третий поток и установить приоритеты для всех потоков.

Вариант10: модифицировать пример3 из данной лабораторной работы: организовать вывод букв латинского алфавита как строчных, так и прописных.

Задание2

Вариант1: поиск 5 букв в файле. Обработка каждой буквы в отдельном потоке.

Вариант2: 5 шахтеров добывают золото равными порциями из одной шахты, задерживаясь в пути на случайное время, до ее истощения. Работа каждого шахтера реализуется в отдельном потоке.

Вариант3: для нескольких файлов (разного размера) требуется вычислить контрольную сумму (сумму кодов всех символов файла). Обработка каждого файла выполняется в отдельном потоке.

Вариант4: вычислить 10-ю степень двойки в различных потоках: сложением, умножением и возведением в степень.

Вариант5: создать два потока. Первый поток производит запись в файл случайных данных. Второй производит чтение данных из этого файла и вывод их на экран.

Вариант6: организовать сортировку массива данных и отображение сортировки на экране. Первый поток производит сортировку по возрастанию, второй по убыванию. После каждого перемещения элемента производится вывод текущего состояния сортировки.

Вариант7: рассчитать значение графика функции $y = 2*x^2–3$, с шагом x=0.01 на промежутке (-10;10). Первый поток производит расчет данных функции и добавляет их в конец массива данных. Второй поток извлекает из массива данных значения и производит выводит функции на экран.

Вариант8: Создать два потока. Первый ищет числа Фибоначчи (каждое последующее число равно сумме двух предыдущих чисел), второй простые числа. Результат работы каждого потока сохраняются в отдельный файл. После остановки потока – программа показывает количество найденных чисел Фибоначчи и простых чисел.

Вариант9: параллельно происходит поиск корней функций: y=sin(x), $y=x^2-2*x+3$, $y= \ln(x^2)/x^3$, при которых значение y равно числу, введенному пользователем.

Вариант10: координаты 3 шариков изменяются на случайную величину по вертикали и горизонтали. При выпадении шарика за нижнюю границу допустимой области шарик исчезает. Изменение координат каждого шарика производить в отдельном потоке.

Контрольные вопросы:

1) В чем разница между процессом и потоком?
2) В чем заключается суть многопоточности?

3) Как получить сведения о потоке?
4) Какие существуют приоритеты потоков?

Лабораторная работа №21
Программирование баз данных на C#

Цель работы: Сформировать понятие о средствах и технологиях языка C# для работы с базами данных. Научить применять средства языка C# при разработке проекта с использованием базы данных реляционного типа.

Аппаратное, программное обеспечение: персональный компьютер, Visual Studio.

Краткие теоретические сведения

Для работы с базами данных необходимо предварительно создать базу на сервере. Это можно сделать с использованием любых удобных инструментов. К примеру, в *Visual Studio* имеются проекты баз данных *SQL*. *Visual Studio Database Project* предоставляет гибкие возможности для создания нового проекта базы данных из существующей БД с помощью нажатия кнопки или возможность создания проекта базы данных с нуля.

Создание новой базы данных

Для создания в *Visual Studio* нового проекта баз данных требуется:

Шаг 1: Выбрать пункт *File* > *New* > *Project*…

Шаг 2: Выбрать *SQL Server*, потом *SQL Server Database Project*

Шаг 3: Ввести название базы данных и выбрать место расположения файла

Шаг 4: Нажать *OK*.

Создание новой таблицы

После выполненных действий будет создан проект новой базы. Для создания таблицы в базе требуется:

Шаг 1. Нажать правой кнопкой мыши щелкаем по *Project* и выберем *Add* > *Table*.

Шаг 2. Ввести название таблицы.

Шаг 4: Нажать *Add*.

После завершения добавления таблицы появится окно дизайнера таблицы. Здесь мы можем добавлять столбцы с типом данных. Эти действия нужно повторить для создания необходимого количества таблиц в базе.

Публикация базы данныз в SQL Server

Для сохранение информации в *SQL Server* требуется:

Шаг 1: в окне *Solution Explorer* счёткнуть правой кнопкой мыши по проекту базы данных и выбирать *Properties*.

Шаг 2: в параметрах проекта выбрать версию *SQL Server*.

Шаг 3: нажать на кнопку *Edit* и изменить свойства соединения в соответствии с параметрами ПК.

Шаг 4: ввести имя сервера, имя пользователя, пароль и выбрать базу данных.

Шаг 5: нажать OK и сохранить все изменения.

Шаг 6: щёлкнуть правой кнопкой мыши на проекте БД и выбрать *Publish*.

Шаг 7: отредактировать настройки “*Target Database*”.

Шаг 8: нажать на кнопку *Publish*.

Теперь можно запустить *SQL Server Management Studio* и проверить, есть ли новая база данных.

Подключение к проекту базы данных

После того, как база данных была создана на сервере, мы можем к ней подключаться из кода программы.

Первым делом нужно определить строку подключения и передать ей информацию о базе данных и сервере, к которым предстоит установить подключение.

Строка подключения представляет набор параметров в виде пар *ключ=значение*.

Таблица 3.18 – Основные параметры подключения:

Наименование	Описание
Data Source	Указывает на название сервера. По умолчанию это ".\SQLEXPRESS". Поскольку в строке используется слеш, то в начале строки ставится символ @. Если имя сервера базы данных отличается, то соответственно его и надо использовать.
Initial Catalog	Указывает на название базы данных на сервере
Integrated Security	Устанавливает проверку подлинности

Приведем пример строки подключенпия к базе данных.

Пример:

```
class Program {
  static void Main(string[] args) {
    string connectionString =
    @"Data Source=.\SQLEXPRESS;Initial
    Catalog=usersdb;Integrated Security=True";
  }
 }
```

При использовании различных систем управления базами данных, различных провайдеров данных .NET строка подключения может отличаться.

Класс DbConnection устанавливает соединение клиентского приложения с источником данных. С помощью свойств этого класса можно задать тип источника данных, его местоположение и некоторые другие атрибуты.

Таблица 3.19 – Основные свойства и методы класса *DbConnection*:

Наименование	Описание
LinterDbConnection()	Инициализация объекта класса *LinterDbConnection*.
ConnectionString	Значение строки подключения с источником данных.
Database	Имя БД источника данных
DataSource	Имя сервера источника данных, с которым установлено соединение.
BeginTransaction()	Начинает транзакцию по заданному соединению.
ChangeDatabase()	Меняет текущее соединение с источником данных для последующей установки нового соединения.

Close()	Закрывает соединение с текущим источником данным.
CreateCommand()	Создает объект *DbCommand*, связанный с текущим соединением.
GetSchema()	Предоставляет список (коллекцию) всех поддерживаемых источником данных объектов БД в текущем соединении.

Продолжение таблицы 3.19

Наименование	Описание
Open()	Открывает соединение с источником данных в соответствии с параметрами, указанными в строке подключения.

Свойства типа *DbConnection* предназначены в основном только для чтения и поэтому нужны, если требуется получить характеристики подключения во время выполнения.

Создание команды

Тип *SqlCommand* (порожденный от *DbCommand*) представляет собой объектно-ориентированное представление *SQL*-запроса, имени таблицы или хранимой процедуры. Тип команды указывается свойством *CommandType*, которое принимает значения из перечисления *CommandType*.

Таблица 3.20 – Основные значения *CommandType*:

Наименование	Пример запроса
Text	string select = "SELECT ContactName FROM Customers"; SqlCommand cmd = new SqlCommand(select, cn);

StoredProcedure	SqlCommand cmd = new SqlCommand("CustOrd", conn);
	cmd.CommandType = CommandType.StoredProcedure;
	cmd.Parameters.AddWithValue("@CustomerID", "QUICK");

TableDirect	OleDbCommand cmd = new OleDbCommand("Categories", conn); cmd.CommandType = CommandType.TableDirect;

Пример создания команды

```
using (SqlConnection cn = new SqlConnection()) {
  cn.ConnectionString = connect.ConnectionString;
  try {
    //Открыть подключение
```

```
    cn.Open();
    // Создание объекта команды с помощью конструктора
    string strSQL = "Select * From Inventory";
    SqlCommand myCommand = new SqlCommand(strSQL, cn);
    // Создание еще одного объекта команды с помощью свойств
    SqlCommand testCommand = new SqlCommand();
    testCommand.Connection = cn;
    testCommand.CommandText = strSQL;
   }
   catch (SqlException ex) {
    // Протоколировать исключение
    Console.WriteLine(ex.Message);
   }
   finally {
    // Гарантировать освобождение подключения
    cn.Close();
   }
  }
```

Выполненние команд

После определения команды она доступна для выполнения. Методы классов *Command* и *SqlCommand* для выполнения запросов представлены в таблице 3.21.

Таблица 3.21 – Методы классов *Command* и *SqlCommand*

Наименование	Описание
ExecuteNonQuery()	Выполняет команду, но не возвращает вывода.
ExecuteReader()	Выполняет команду и возвращает типизированный *IDataReader*;
ExecuteScalar()	Выполняет команду и возвращает значение из первого столбца первой строки любого результирующего набора.
ExecuteXmlReader()	Выполняет команду и возвращает объект *XmlReader*, который может быть использован для прохода по фрагменту *XML*, возвращенному из базы данных.
ExecuteNonQuery()	Для выполнения операторов *UPDATE, INSERT* или *DELETE*, где единственным возвращаемым значением является количество обработанных строк.
ExecuteReader()	Выполняет команду и возвращает типизированный объект-читатель данных, в зависимости от используемого поставщика. Возвращенный объект может применяться для итерации по возвращенным записям.

Приведем пример применения метода ExecuteNonQuery() и метода ExecuteReader().

Примеры:

Применение метода *ExecuteNonQuery()*:

```
  string strSQL = "UPDATE Customers SET
```

```
LastName = 'Johnson' WHERE LastName = 'Walton'";
SqlCommand myCommand = new SqlCommand(strSQL, cn);
int i = myCommand.ExecuteNonQuery();
```

применения метода *ExecuteReader():*

```
string strSQL = "SELECT * FROM Inventory";
SqlCommand myCommand = new SqlCommand(strSQL, cn);
SqlDataReader dr = myCommand.ExecuteReader();
while (dr.Read())
  Console.WriteLine("ID: {0} Car Pet Name: {1}",dr[0],dr[3]);
```

Вставка, удаление, обновление записей в базе данных

Метод *ExecuteNonQuery()* предназначен для выполнения операторов *SQL*, модифицирующих таблицу данных. Рассмотрим примеры вставки новых значений в таблицу, удаления и изменения.

Пример:

вставка новых значений в таблицу:

```
public void InsertAuto(int id, string color, string make, string petName) {
  // Оператор SQL
  string sql = string.Format("Insert Into Inventory" +
       "(CarID, Make, Color, PetName)
  Values(@CarId, @Make, @Color, @PetName)");
  using (SqlCommand cmd = new SqlCommand(sql, this.connect)) {
   // Добавить параметры
   cmd.Parameters.AddWithValue("@CarId", id);
   cmd.Parameters.AddWithValue("@Make", make);
   cmd.Parameters.AddWithValue("@Color", color);
   cmd.Parameters.AddWithValue("@PetName", petName);
   cmd.ExecuteNonQuery();
  }
 }
```

удаление данных из таблицы:

```
public void DeleteCar(int id) {
  string sql = string.Format("Delete from Inventory where CarID = '{0}'", id);
  using (SqlCommand cmd = new SqlCommand(sql, this.connect)) {
   try {
    //выполняем запрос
    cmd.ExecuteNonQuery();
   }
   //если возникло сообщение об ошибке – выводим текст
   catch (SqlException ex) {
    Exception error = new
    Exception("К сожалению, машина заказана!", ex);
    throw error;
   }
  }
 }
```

изменение данных в таблице:

```
public void UpdateCarPetName(int id, string newpetName) {
  string sql = string.Format("Update Set PetName = '{0}' Where CarID = '{1}'",
       newpetName, id);
  using (SqlCommand cmd = new SqlCommand(sql, this.connect)) {
   cmd.ExecuteNonQuery();
  }
 }
```

Порядок выполнения работы

1. Изучить теоретические сведения.
2. Для класса, реализованного в лабораторной работе «Разработка многопоточных приложений на C#» реализовать сохранение данных в базу.
3. Модифицировать приложение для демонстрации работы.
4. В консоль выводить информацию:
   - о подключении к серверу
   - о выборе БД
   - о записи данных
5. Создать отчет, включив в него блок-схему разработанного алгоритма, ответы на контрольные вопросы.

Контрольные вопросы

1. Что такое поток?
2. Для чего нужно пространство имен System.Threading?
3. Для чего используется делегат ThreadStart?
4. Какие приоритеты потоков описаны в перечислении ThreadPriority?

Для чего используется делегат ParameterizedThreadStart?