

Документ подписан простой электронной подписью

Информация о владельце:

ФИО: Локтионова Оксана Геннадьевна

Должность: проректор по учебной работе

Дата подписания: 18.06.2023 15:27:56

Уникальный идентификатор:

0b817ca911e6668abb13a5d426d39e5f1c11eabbf73e947df4a4851fda56d089

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
Высшего образования "Юго-Западный государственный университет"
(ЮЗГУ)

Кафедра биомедицинской инженерии



ТЕОРИЯ И ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ ДЛЯ БИОТЕХНИЧЕСКИХ СИСТЕМ

Методические указания к проведению лабораторных работ для студентов
направления подготовки 12.03.04 - "Биотехнические системы и технологии»

Курск 2023

УДК 615.478

Составитель М.В. Артеменко

Рецензент

Кандидат технических наук, доцент *С. С. Шевелев*

Теория и технология программирования для биотехнических систем: методические указания к поведению лабораторных работ / ЮгоЗап. гос. ун-т; сост. М.В. Артеменко . - Курск, 2023. – 60 с.: ил. 6, табл.2.

Содержатся сведения, необходимые для выполнения лабораторных работ по теории и технологии программирования для биотехнических систем.

Методические указания по структуре, содержанию и стилю изложения материала соответствуют методическим и научным требованиям, предъявляемым к учебным и методическим пособиям.

Предназначены для студентов направления подготовки 12.03.04 очной формы обучений.

Текст печатается в авторской редакции

Подписано в печать

Усл. печ.л. . Уч. -изд.л. Тираж _____ экз. Заказ.

Юго-Западный государственный университет

305040, г.Курск, ул. 50 лет Октября, 94

Формат 60x84 1/16

Бесплатно.

СОДЕРЖАНИЕ

Лабораторная работа №1. Изучение технологии создания кросс-платформенных приложений	4
Лабораторная работа №2. Разработка внешних спецификаций и проектирование программного обеспечения: переменные, ввод-вывод, классы string, stringbuilder и character	14
Лабораторная работа №3. Объектно-ориентированное программирование	22
Лабораторная работа № 4. Абстрактные классы и интерфейсы	29
Лабораторная работа №5. Введение в разработку графического интерфейса пользователя.....	35
Лабораторная работа №6. Компоненты Java и модель делегирования событий.....	42
Лабораторная работа №7. Графический дизайн пользовательского интерфейса и адаптеры класса.....	53

Лабораторная работа №1. Изучение технологии создания кросс-платформенных приложений

1.1 Цель работы: Изучение фундаментальных блоков программ Java, программирование Java, с использованием консоли операционной системы Windows, использование IDE NetBeans и Eclipse IDE.

1.2 . Краткие теоретические сведения

Java - это язык программирования, первоначально разработанный Джеймсом Гослингом в Sun Microsystems (который с тех пор объединился с Oracle Corporation), и выпущен в 1995 году как основной компонент платформы Java Sun Microsystems. Язык унаследовал большую часть своего синтаксиса из C и C++, но имеет более простую объектную модель и возможности более низкого уровня. Приложения Java обычно скомпилированы в байт-код (файл класса), который может работать на любой виртуальной машине Java (JVM) независимо от архитектуры компьютера. Java - это универсальный, параллельный, основанный на классе, объектно-ориентированный язык, который специально разработан для того, чтобы иметь как можно меньше зависимостей в реализации. Он предназначен для того, чтобы разработчики приложений «написал один раз, запустил в любом месте», что означает, что код, который работает на одной платформе, не нужно перекомпилировать для запуска на другой. В настоящее время Java является одним из самых популярных языков программирования, особенно для клиент-серверных веб-приложений, с сообщением о 10 миллионах пользователей.

Разработчики Java решили использовать комбинацию компиляции и интерпретации. Программы, написанные на Java, скомпилированы в машинный язык, но этот язык для компьютера, который на самом деле не существует. Этот так называемый «виртуальный» компьютер известен как виртуальная машина Java или JVM. Язык для виртуальной машины Java называется байт-кодом Java. Нет причин, по которым байт-код нельзя использовать в качестве машинного языка реального компьютера, а не виртуального. Но на самом деле использование виртуальной машины делает возможным одну из основных точек продажи Java:

тот факт, что ее можно фактически использовать на любом компьютере. Все, что требуется компьютеру, - это интерпретатор для байт-кода Java. Такой интерпретатор имитирует JVM так же, как Virtual PC имитирует компьютер ПК. (Термин JVM также используется для программы интерпретатора байт-кода Java, которая выполняет симуляцию, поэтому мы говорим, что компьютеру требуется JVM для запуска программ Java. Технически было бы правильнее сказать, что интерпретатор реализует JVM, чем сказать, что это JVM.)

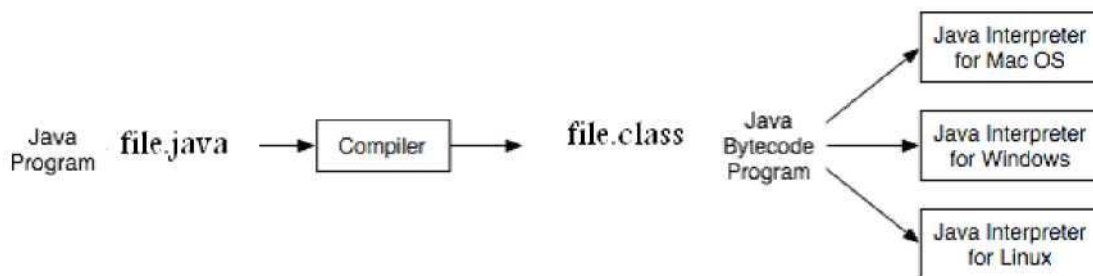


Рисунок 2.1. Создание и исполнение байт-кода

Рисунок 2.1. Иллюстрирует основные этапы создания и выполнения байтового кода. Сначала программист создает оригинальный файл Java, который должен быть сохранен в файле с расширением «.java», это текстовый файл, который можно создать с помощью любого текстового редактора. Затем исходный файл .java может быть скомпилирован в байт-код с использованием компилятора Java. В операционной системе Windows можно использовать консольную команду:

«Javac FileName.java», где «FileName.java» - это фактическое имя файла Java, созданного программистом. Приведенный байт-код после компиляции будет сохранен под тем же именем с расширением .class. Затем байт-код можно выполнить с помощью виртуальной машины Java командой: "java имя_файла".

Как правило, для создания программы, написанной на VB.Net, C ++, C #, вам нужен как минимум компилятор, интегрированный в IDE, такой как Microsoft Visual Studio 2008, 2010. Чтобы создать программу на Java, вы также можете выбрать подходящие IDE, такие как Eclipse или NetBeans или, если хотите, вы можете создавать и компилировать программу Java без какой-либо IDE, используя консольный режим операционной системы. Сегодня мы покажем, как создать простую программу Java, используя

консольный режим, затем с помощью Eclipse и, наконец, с помощью IDE NetBeans.

Во время этой лабораторной работы мы предположили, что Java System Development Kit (SDK) уже установлен в вашей операционной системе. Последняя версия Java (на момент создания этого руководства) - это Java 7.2, эта версия может отличаться от версии, установленной на вашем компьютере LAB. В вашем классе комнатный компьютер версия Java - 6.12, установленная в позиции:

C:\Program Files\Java\jdk1.6.0_16\bin.

Первая часть нашей лаборатории объяснит, как создавать и запускать программу Java без какой-либо среды, используя консольное окно операционной системы.

2.3. Порядок выполнения работы

2.3.1 . Добавить путь к java-компилятору в переменную среды операционной системы Windows. Для этого перейдите к началу, затем запустите, затем введите команду «cmd», как показано на рисунке, затем нажмите кнопку «ОК»:

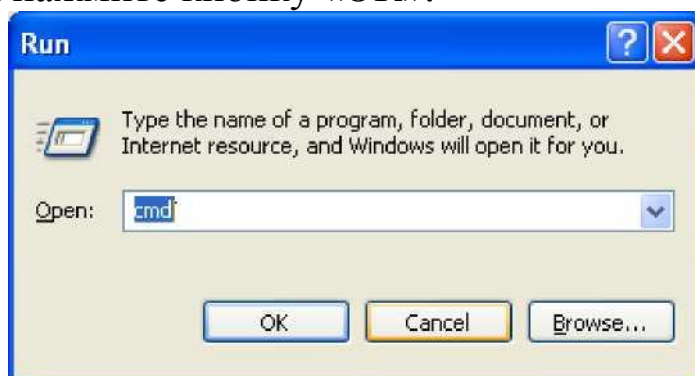


Рисунок 2.2. Выполнение команды «cmd» в Windows XP

Консольное окно операционной системы, представленное на рисунке 2.3, вы можете увидеть свое текущее местоположение в папке C: \ Documents and Settings \ admin, где admin - это имя учетной записи.

2.3.2 Посмотрите на окно консоли, текущая позиция, связанная с именем учетной записи:

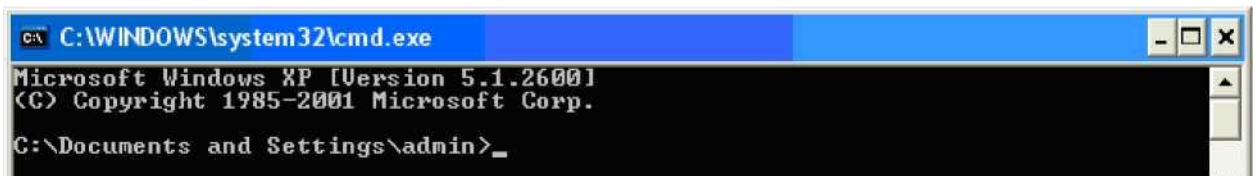


Рисунок 1.3. Окно консоли операционной

Во-первых, мы должны добавить путь компилятора Java к переменному окружению операционной системы. Можно использовать «Мой компьютер» Windows XP или «Компьютер» Windows 7, но лучше и быстрее использовать консольные команды. Команда добавления пути к Java Compiler:

Путь C: \ Program Files \ Java \ jdk1.6.0_16 \ bin, как указано на рисунке 1.4.

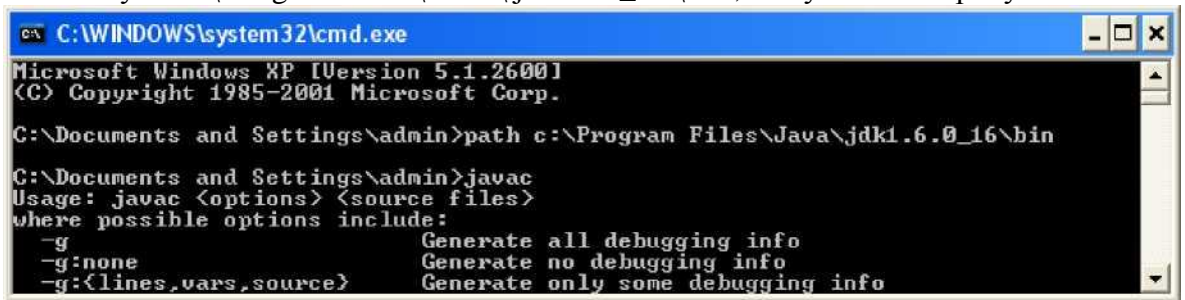


Рисунок 1.4. Добавление пути Java-компилятора к переменной окружения системы Windows.

2.3.3 Чтобы создать программу Java, мы должны сначала создать папку для хранения исходного кода программы, это также возможно с помощью консольных команд, см. Рисунок 1.5.



Рисунок 1.5. Создание новой папки с помощью консольной команды.

Если вы помните «Компьютерное мастерство», то для консоли управления командой для создания новой папки "mkdir <Имя новой папки>". Давайте назначим имя новой папки «javaproject». Чтобы войти в новую папку, мы можем использовать команду «cd javaproject». Чтобы создать программу, вы можете использовать любой текстовый редактор, такой как Microsoft world или Notepad, но убедитесь, что расширение нового файла - java, а не doc или txt.

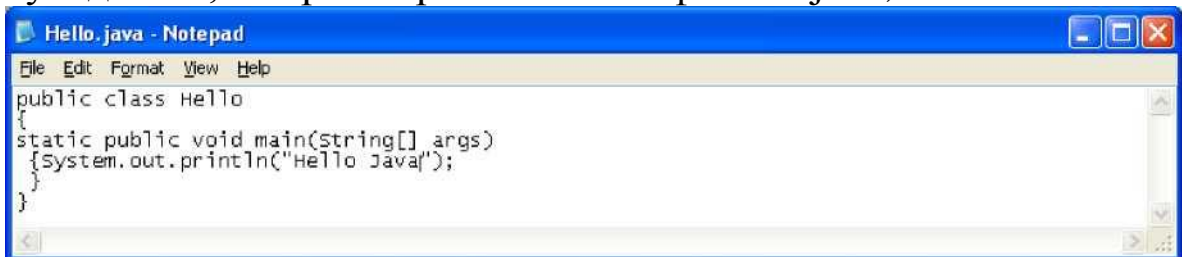


Рисунок 1.6. Создание новой программы в Notepad.

Текст файла Java, представленный на рисунке 1.6. Это простая программа, которая включает открытый класс «Hello» и основной метод с инструкцией `System.out.println`, которая может использоваться для вывода числовой или текстовой информации. Затем вы должны ввести текст программы, сохранить файл и закрыть программу «Блокнот».

Вернитесь в окно консоли и введите команды «`javac Hello.java`», затем «`java Hello`», чтобы запустить программу, как показано на рисунке 1.7.



```
C:\WINDOWS\system32\cmd.exe
D:\>mkdir javaproject
D:\>cd javaproject
D:\javaproject>javac Hello.java
D:\javaproject>java Hello
Hello Java
D:\javaproject>
```

Рисунок 1.7. Консольные команды для компиляции и запуска программы Java.

На выходе программы, представленной на рисунке 1.7, вы можете увидеть строку текста, созданную командой `System.out.println` («Hello world»);

1.3.4 Использование IDE Eclipse в Java-программировании

Eclipse - это сообщество с открытым исходным кодом, которое разрабатывает открытые платформы и продукты. Сообщество утверждает, что его проекты «направлены на создание открытой платформы разработки, состоящей из расширяемых фреймворков, инструментов и времени автономной работы для создания, развертывания и управления программным обеспечением на протяжении всего жизненного цикла». Фонд Eclipse является некоммерческой корпорацией, которая действует в качестве управляющего сообщества Eclipse. В мире программного обеспечения простое упоминание «Eclipse» обычно относится к набору для разработки программного обеспечения Eclipse (SDK). Eclipse SDK состоит из платформы Eclipse, средств разработки Java и среды разработки плагинов. Платформа Eclipse представляет собой многоязычную среду разработки программного обеспечения, включающую интегрированную среду разработки (IDE) и расширяемую систему подключаемых модулей. Он написан в

основном на Java. С помощью различных плагинов он может быть использован для разработки приложений на различных языках программирования, включая Ada, C, C ++, COBOL, Java, Perl, PHP, Python, R, Ruby (включая Ruby on Rails), Scala, Clojure , Groovy и Scheme. Его также можно использовать для разработки пакетов для программного обеспечения Mathematica. Среда разработки включает в себя инструменты разработки Java Eclipse (JDT) для Java, Eclipse CDT для C / C ++ и Eclipse PDT для PHP и другие.

Eclipse SDK (который включает инструменты разработки Java) предназначен для разработчиков Java. Пользователи могут расширить свои возможности, установив плагины, написанные для платформы Eclipse, такие как инструментальные средства разработки для других языков программирования, а также могут писать и вносить свои собственные подключаемые модули. Выпущенный на условиях публичной лицензии Eclipse, Eclipse SDK является бесплатным и открытым исходным кодом. Eclipse имеет много выпусков, последний из них (на момент написания этого текста) имеет имя «Eclipse Indigo», созданное в 2011 году. Eclipse IDE не имеет установки, все, что вам нужно, - это загрузка архива с сайта Eclipse ([www. Eclipse.org](http://www.Eclipse.org)), тогда вы должны извлечь папку «Eclipse» в любое место вашего жесткого диска, мы советуем вам использовать корень вашего диска D: поскольку раздел C: ваш компьютер LAB может быть защищен программным обеспечением Deepfreeze.

Чтобы запустить Eclipse, войдите в папку Eclipse, затем дважды щелкните файл «eclipse.exe». При первом запуске IDE на



экране появится запрос о рабочей области (см. Рисунок 1.8.):

Введите местоположение рабочей области: «D: \ JavaProjects \») и нажмите «ОК», на следующей странице нажмите «Workbench».

Теперь вы готовы создать свой первый проект Java с помощью eclipse. Перейдите в «Файл», «Создать» и нажмите «Проект Java», выберите название проекта, как показано на рисунке 1.9:

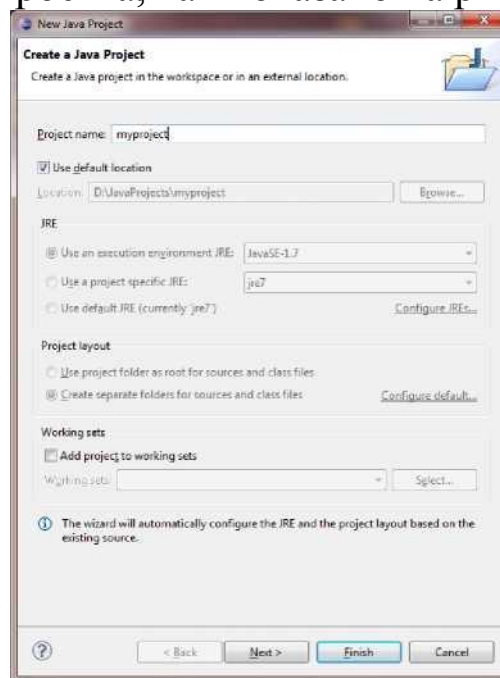


Рисунок 1.9. Создание нового проекта в Eclipse Environment

Оставьте все остальные настройки по умолчанию, затем нажмите кнопку Готово. Будет создан новый пустой проект. Любой Java-проект содержит как минимум один класс, чтобы добавить новый класс в пустой проект Java, перейдите в файл, затем «Создать», затем «Класс», дайте имя «Hello» для класса и нажмите «Готово». Введите текст своей программы:

```
public class Hello {  
  
    static public void main(String[] args)  
    {  
        System.out.println("Hello world");  
    }  
}
```

, Затем нажмите Ctrl + F11, чтобы скомпилировать и запустить проект. См. Результат на рисунке 1.10

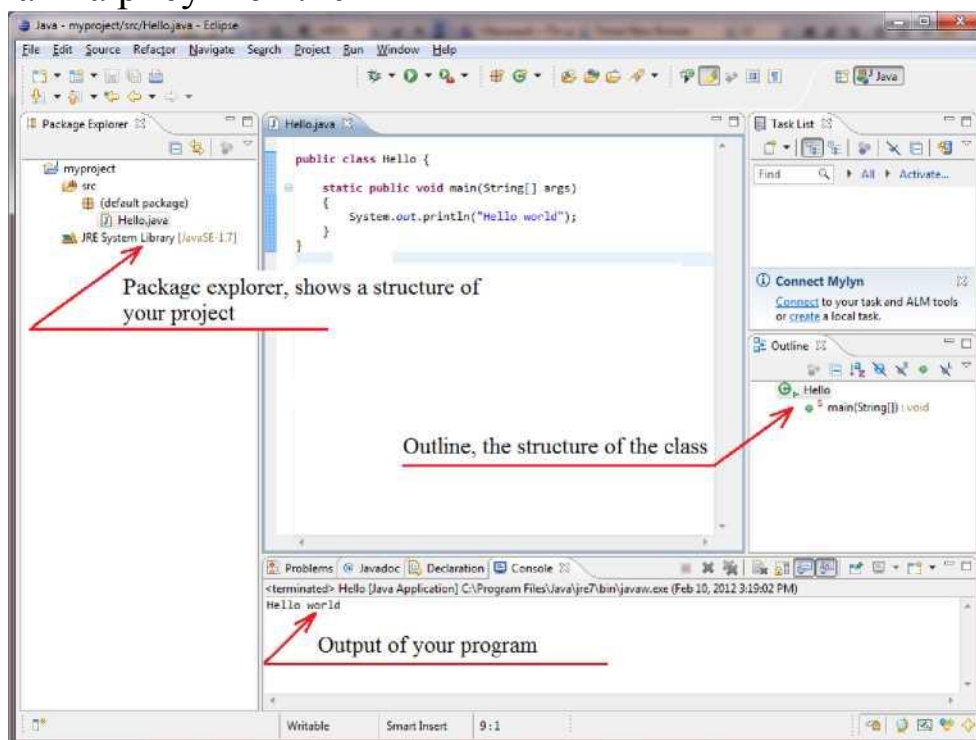


Рисунок 1.10. Основные элементы Eclipse IDE.

2.3.5 Программирование на Java с использованием среды IDE NetBeans.

Вы можете загрузить и установить последнюю версию Eclipse IDE на веб-сайте www.oracle.com. После установки и запуска IDE вы должны нажать «Файл», «Новый проект», выберите «Категории: Java», «Проект: приложение Java», затем нажмите «Далее» (см. Рис. 1.11.).

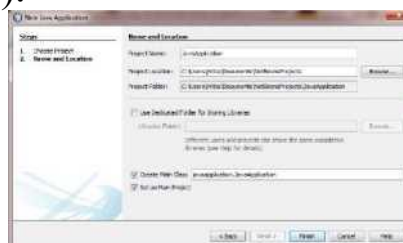


Рисунок 1.11. Создание нового приложения Java с использованием Netbeans.

Согласно рисунку 1.11. По умолчанию именем приложения Netbeans является JavaApplication, это имя класса, которое будет создано после нажатия кнопки «Готово». На рисунке 1.12 показаны

основные компоненты среды IDE NetBeans.

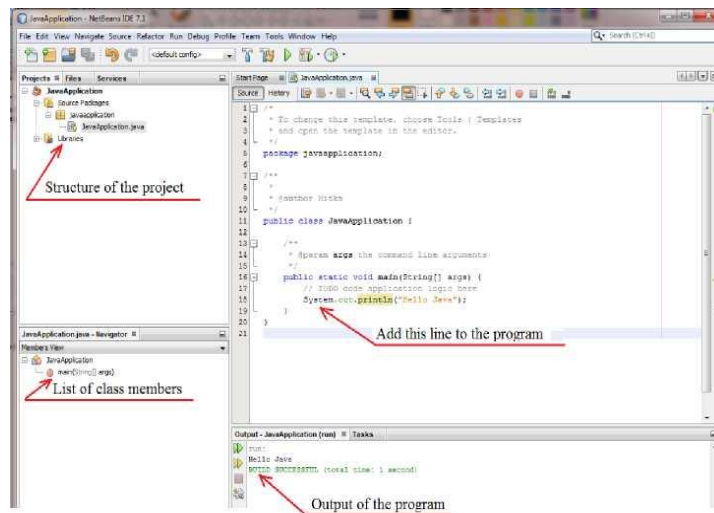


Рисунок 1.12. Основные компоненты IDE NetBeans.

Как вы можете видеть, вся структура программы создана автоматически с помощью IDE, вам просто нужно добавить в программу строку «System.out.println (« Hello Java »)». Для компиляции и запуска вашей программы нажмите F6, выход Появляется в окне вывода IDE (см. Рисунок 1.12).

1.4. Содержание отчета

1. Задание на лабораторную работу
2. Последовательность действий, произведенных при выполнении задания
3. Текст программы
4. Скриншот работы созданного приложения

Лабораторная работа №2. Разработка внешних спецификаций и проектирование программного обеспечения: переменные, ввод-вывод, классы string, stringbuilder и character

2.1. Цель работы: изучить основы разработки внешних спецификаций разрабатываемого ПО, а также технологию проектирования программного обеспечения на основе объектно-ориентированного подхода.

2.2. Краткие теоретические сведения

Внешнее проектирование и внешние спецификации - это процесс описания планируемого поведения разрабатываемого программного изделия с точки зрения потенциальных пользователей. При разработке внешних спецификаций необходимо обеспечить концептуальную целостность проекта, которая представляет собой меру единообразия способа взаимодействия с пользователем. Система с концептуальной целостностью имеет следующую характеристику: все средства, доступные одному пользователю, должны быть доступны другим пользователям, т.е. интерфейсы пользователей должны быть одинаковы.

При разработке внешних интерфейсов пользователя необходимо решить следующие проблемы:

- 1) доведение до минимума ошибок пользователя;
- 2) обнаружение ошибок пользователя в случае их возникновения;
- 3) доведение до минимума сложности разрабатываемого программного изделия.

Разработка внешних спецификаций разбивается на две части: предварительный внешний проект и детальный внешний проект.

Предварительный внешний проект содержит описание основных компонентов, а также компонентов, из которых состоят основные компоненты, далее - внешних функций, составляющих отдельные компоненты проекта.

Детальный внешний проект каждой функции пользователя должен включать следующую информацию:

1. Описание входных данных - синтаксис, семантика, допустимые значения, формат.
2. Описание выходных данных - описание всех результатов функции, описание связи между входными и выходными данными.

3.3. Задание для лабораторной работы.

3. Преобразование системы.
4. Характеристика надёжности - описание возможных отказов функций на систему.
5. Эффективность - описание ограничений, накладываемых на эффективность функции.
6. Замечания по программированию - описание внешних спецификаций с точки зрения пользователя.

Для разработки программного обеспечения в настоящее время используется объектно-ориентированный подход. В практическом случае для любого класса компьютерного языка используется набор элементов данных и методов под именем. Элементами данных являются любые структуры данных внутри класса: переменные, такие как `int`, `float`, `double`, `short`; Массивы некоторых элементов; Объекты некоторых других классов, таких как `String`, `StringBuilder`, `ArrayList`. Методы класса - это члены функции любой функции внутри класса. Итак, просто вы можете сказать, что класс - это набор переменных и набор функций, расположенных вместе в одном месте под некоторым именем. Класс имеет имя; Это похоже на некоторый тип данных. В другой руке у нас есть объект - экземпляр класса, объект выглядит как переменная некоторого типа данных. Пример:

```
Int a; // a - переменная типа данных int;
String st; // st - объект класса String.
Пример объявления класса в Java: class MyClass {
    // field, constructor, and // method declarations
}
```

В примере выше описан новый класс с именем «MyClass». Внутри класса вы можете выделить набор переменных, массивов и объектов других классов. «class» - это ключевое слово Java. Перед этим ключевым словом вы можете написать префикс «public». Слово «public» для классов в Java означает, что этот класс доступен из других классов, из других пакетов и из других программ. Несколько раз вы можете встретить некоторые другие префиксы перед ключевым словом «class»: «abstract» означает, что некоторые методы класса, объявленные, но не определенные (см. Лабораторную работу 2, чтобы узнать подробности абстрактных классов), префикс «final» определяет последний класс (без наследования, дочерний класс не может быть создан).

Класс может иметь конструктор. Конструктор - это метод класса

(функция внутри класса), который имеет одно и то же имя с классом. Конструктор будет выполняться автоматически в любое время, когда будет создан объект класса. Класс может не иметь конструктора, может иметь один конструктор из нескольких конструкторов. Последний факт, называемый перегрузкой конструктора - ситуация, в которой у вас есть много конструкторов внутри класса с тем же именем (что совпадает с именем класса), но с разными параметрами.

Теперь вы знаете, что в Java, как в C ++, есть конструкторы. Но в отличие от C ++ в Java нет деструкторов. Деструктор в C ++ - это метод, который имеет одно и то же имя с префиксом класса плюс ~ перед именем. Деструктор в C ++ - это функция, которая будет выполняться автоматически в любое время, когда объект класса будет уничтожен (используя ключевое слово «удалить»). В Java нет указателей, исключение ключевых слов, отсутствие множества памяти и никаких деструкторов. Все объекты выделены не в множестве памяти, а в некоторой другой памяти виртуальной машины Java, называемой «сбор мусора». Это означает, что программист может создавать объект, но не может его уничтожить. Программист может сообщить о сборе мусора, что какой-то объект больше не нужен, назначив значение «null» объекту, но физически этот объект будет уничтожен последним, может быть через один час, время разрушения непредсказуемо на Java. Вот почему никакие деструкторы в Java не гарантируют, что объект будет уничтожен в будущем. Если вы хотите, то можете использовать деструктор, представленный на Java, с помощью метода «finalize», но ни один орган не может гарантировать, что деструктор будет выполнен в вашей программе, поэтому в практических программах Java нет причин использовать деструкторы.

Пример прояснит ситуацию с классами, конструкторами, деструкторами и объектами. **class** MyClass //MyClass is class name {
int a=1,b=2,c=3;//variables a,b,c look like public variables **public int**
d=4,e;//d and e are public variables

protected float f=5.5f;//f is protected variable, in Java it look like public
private String st="Hello";//This is private variable, like private in C++

public MyClass()//This is constructor. Name equal to the class name.

//No output of this function!

{


```

System.out.println("This is default constructor"); }

public MyClass(String s)//This is overloaded constructor {st=st+s;
    System.out.println("This is String constructor: "+st); }

public MyClass(float k)//This is also overloaded constructor {f=k;
    System.out.println("This is float constructor, value of f is "+f);
}

void f1()//this is method of the class {
    System.out.println("This is method f1 of the class MyClass");
}

protected void finalize()
{
System.out.println("This is destructor, but destructurs in Java are not
useful");
}
} //end of the class public class Example{//This is public class with main
method
    //The code on this page located in the file Example.java
    static public void main(String[] args)
    {MyClass obj1=new MyClass();
        obj1.a=5;
        obj1.f1();
        obj1=null;
        MyClass obj2=new MyClass("Ahmad");
        MyClass obj3=new MyClass(5.5f);
    }
}

```

Результатом этой программы будет:

```

This is default constructor
This is method f1 of the class MyClass
This is String constructor: HelloAhmad
This is float constructor, value of f is 5.5

```

Вы можете видеть, что в основном методе мы пытались уничтожить объект obj1, присвоив значение null obj1 = null ;, но на самом деле этот объект не разрушен, поэтому нет причин использовать

деструкторы в реальных Java-программах. Также вы можете видеть, что у нас есть операторы «new» в каждой строке, где созданы объекты, но не «delete» операторов, потому что все объекты, созданные в сборке мусора, а не в памяти кучи и сборке мусора, будут автоматически очищены виртуальной машиной Java после выполнения вашей программы.

Вывод программы показывает, что если объект класса, созданного с помощью пустых скобок, конструктор по умолчанию выполнен автоматически. Перегруженные конструкторы будут выполняться только в том случае, если вы передадите соответствующие параметры. В Java, как в C ++, есть наследование, это означает, что вы можете создать класс на основе некоторого класса. В следующем примере мы создадим Class1 сначала Class2 на основе Class1, чтобы показать наследование, все строки кода Java, пронумерованные, чтобы иметь ссылки в тексте:

```
class Class1 //Class1 is superclass of Class2
{int a=1;
protected float b=2;//protected look like public in Java
void f1()
{
System.out.println("Method f1 of Class1");
}
}
9
class Class2 extends Class1//Class2 is subclass of Class1
{
void f2()
{
System.out.println("Method f2 of Class2:");
System.out.println("I can call data members and
methods of Class1:");
System.out.println("a="+a+" b="+b);
f1();//We call method f1 of Class1 directly
int a=5;//from this moment we have new local variable
a
//now we can access a of Class1 using keyword super:
System.out.println("local a="+a+" but a of
Class1="+super.a);
}
```

```
}
```

```
public class Example{  
//This is public class with main method  
//The code on this page located in the file  
Example.java  
static public void main(String[] args)  
{Class2      obj=new Class2();  
obj.b=100;//b is protected but can be accessed  
    directly even outside a  
        //class  
        obj.f2();  
    }  
}
```

Результатом этой программы будет:

Method f2 of Class2

I can call data members and methods of Class1: a=1 b=2.0

Method f1 of Class1

local a=5 but a of Class1=1

Программа содержит 3 класса: Class1, Class2 и Example. Последний класс является общедоступным, поэтому вся программа находится в файле Example.java. Класс с именем «Class1» является корневым классом, созданным без наследования, пусть «from zero». Класс содержит два элемента данных a и b и один метод f1 (). Члены класса также называются в Java как «fields of the class». Начальные значения элементов данных, которые вы можете назначить непосредственно в классе.

В строке 10 нашего примера вы можете увидеть определение класса 2, созданного с использованием Class1. Вы можете увидеть ключевое слово «extends», что означает, что Class2 унаследовал все от Class1. Внутри Class2 мы имеем только один метод f2 (), но все члены данных и методы, унаследованные от Class1, также доступны в Class2. В строке 16 (method f2 () class 2) мы печатаем значения переменных a и b, объявленных в Class1. Затем в строке 17 мы назвали метод f1 (), расположенный в Class1. В строке 18 мы создали новую локальную переменную a и присвоенное значение 5, начиная с этого момента мы можем получить доступ к переменной «a» class1,

используя ключевое слово «super», что означает перейти к суперклассу (родительский класс) и получить определенный член класса после десятичной точки «.», В нашем случае это элемент данных «a» class1, см. Строку 20.

В строке 29 мы создали объект «obj» класса Class2. Затем в строке 30 мы получили доступ к защищенному элементу данных «b», объявленному в Class1. Поскольку вы видите, что «protected» элемент данных выглядит как открытый, его можно получить непосредственно в классе, где он был объявлен, или в любом другом подклассе.

2.3. Задание для лабораторной работы.

Разработайте предварительный внешний проект для описания компонентов, составляющих программу и напишите программу Java, которая печатает все реальные решения квадратичного уравнения. Решение квадратичного уравнения должно быть реализовано в классе Quadratic. Класс имеет 3 общих элемента данных **a**, **b**, **c**. Вы должны создать методы класса:

Конструктор по умолчанию класса: назначить нулевые значения: $a = 0$, $b = 0$, $c = 0$ для членов данных класса;

Void inputdata (); Этот метод попросит пользователя ввести номера a , b , c с помощью ПК-клавиатуры, все номера должны быть назначены элементам данных a , b и c .

Int checksolution (); Этот метод возвращает число корней (0, если отрицательное значение **Discriminant (D)**, 1, if $D = 0$, 2, if $D > 0$) или распечатка «Нет входных данных», если метод inputdata () не выполнен. $D = b^2 + 4ac$.

Void printoutresults (); Этот метод печатает корни квадратного уравнения. Если квадратичное уравнение не имеет корней, то распечатка метода «Нет корней», если только один корень квадратного уравнения представляет собой распечатку метода « $x =$ значение корня», если два корня будут напечатаны « $x_1 = \text{root1}$ и $x_2 = \text{root2}$ ». Этот метод распечатывает «Нет входных данных», если метод inputdata () не выполнен. Значения корней, которые вы можете вычислить, используя уравнение: $x, ? = \frac{-b \pm \sqrt{D}}{2a}$, where $D=b^2+4ac$.

1,2 2a

Запишите текст класса и основного метода, чтобы создать объект

класса Квадратичный и вычислить корни некоторых квадратичных уравнений.

1. Титульный лист
2. Задание на лабораторную работу
3. Внешняя спецификация разрабатываемой программы
4. Листинг программы
5. Скриншот работы приложения

Лабораторная работа №3. Объектно-ориентированное программирование

5.1. Цель работы: изучить основы объектно-ориентированного программирования, классов, объектов, методов, приватных, защищенных и публичных членов классов

5.2. Краткие теоретические сведения

Интересно, что нет ни одного стандартного определения термина «class» в современной литературе. В некоторых книгах он определяется как метод использования одной или нескольких переменных, которые будут использоваться в качестве основы для более подробной переменной. В других книгах определение класса предоставляет шаблон или план, который описывает данные, содержащиеся внутри, и поведение объектов, созданных в соответствии с новым типом.

В практическом случае для любого класса компьютерного языка используется набор элементов данных и методов под именем. Элементами данных являются любые структуры данных внутри класса: переменные, такие как `int`, `float`, `double`, `short`; Массивы некоторых элементов; Объекты некоторых других классов, таких как `String`, `StringBuilder`, `ArrayList`. Методы класса - это члены функции любой функции внутри класса. Итак, просто вы можете сказать, что класс - это набор переменных и набор функций, расположенных вместе в одном месте под некоторым именем. Класс имеет имя; Это похоже на некоторый тип данных. В другой руке у нас есть объект - экземпляр класса, объект выглядит как переменная некоторого типа данных. Пример: `Int a;` // `a` - переменная типа данных `int`;

`String st;` // `st` - объект класса `String`.

Пример объявления класса в Java:

```
class MyClass {
    // field, constructor, and
    // method declarations
}
```

В примере выше описан новый класс с именем «MyClass». Внутри класса вы можете выделить набор переменных, массивов и объектов других классов. «class» - это ключевое слово Java. Перед этим

ключевым словом вы можете написать префикс «public». Слово «public» для классов в Java означает, что этот класс доступен из других классов, из других пакетов и из других программ. Несколько раз вы можете встретить некоторые другие префиксы перед ключевым словом «class»: «abstract» означает, что некоторые методы класса, объявленные, но не определенные (см. Лабораторную работу 2, чтобы узнать подробности абстрактных классов), префикс «final» определяет последний класс (без наследования, дочерний класс не может быть создан).

Класс может иметь конструктор. Конструктор - это метод класса (функция внутри класса), который имеет одно и то же имя с классом. Конструктор будет выполняться автоматически в любое время, когда будет создан объект класса. Класс может не иметь конструктора, может иметь один конструктор из нескольких конструкторов. Последний факт, называемый перегрузкой конструктора - ситуация, в которой у вас есть много конструкторов внутри класса с тем же именем (что совпадает с именем класса), но с разными параметрами.

Теперь вы знаете, что в Java, как в C ++, есть конструкторы. Но в отличие от C ++ в Java нет деструкторов. Деструктор в C ++ - это метод, который имеет одно и то же имя с префиксом класса плюс ~ перед именем. Деструктор в C ++ - это функция, которая будет выполняться автоматически в любое время, когда объект класса будет уничтожен (используя ключевое слово «удалить»). В Java нет указателей, исключение ключевых слов, отсутствие множества памяти и никаких деструкторов. Все объекты выделены не в множестве памяти, а в некоторой другой памяти виртуальной машины Java, называемой «сбор мусора». Это означает, что программист может создавать объект, но не может его уничтожить. Программист может сообщить о сборе мусора, что какой-то объект больше не нужен, назначив значение «null» объекту, но физически этот объект будет уничтожен последним, может быть через один час, время разрушения непредсказуемо на Java. Вот почему никакие деструкторы в Java не гарантируют, что объект будет уничтожен в будущем. Если вы хотите, то можете использовать деструктор, представленный на Java, с помощью метода «finalize», но ни один орган не может гарантировать, что деструктор будет выполнен в вашей программе, поэтому в практических программах Java нет причин использовать деструкторы.

Пример прояснит ситуацию с классами, конструкторами, деструкторами и объектами. **class** MyClass //MyClass is class name {
int a=1,b=2,c=3;//variables a,b,c look like public variables **public int**
d=4,e;//d and e are public variables

protected float f=5.5f;//f is protected variable, in Java it look like public

private String st="Hello";//This is private variable, like private in C++

public MyClass();//This is constructor. Name equal to the class name.

//No output of this function!

{

System.out.println("This is default constructor");

}

public MyClass(String s)//This is overloaded constructor {st=st+s;

System.out.println("This is String constructor: "+st);

}

public MyClass(**float** k)//This is also overloaded constructor {f=k;

System.out.println("This is float constructor, value of f is "+f);

}

void f1();//this is method of the class {

System.out.println("This is method f1 of the class MyClass");

}

protected void finalize()

{

System.out.println("This is destructor, but destructurs in Java are not useful");

}

//end of the class **public class** Example{//This is public class with main
method //The code on this page located in the file Example.java **static**

public void main(String[] args) {MyClass obj1=new MyClass();

obj1.a=5;

obj1.f1();

obj1=null;

MyClass obj2=new MyClass("Ahmad");

MyClass obj3=new MyClass(5.5f);

}


```
}
```

Результатом этой программы будет:

This is default constructor

This is method f1 of the class MyClass

This is String constructor: HelloAhmad

This is float constructor, value of f is 5.5

Вы можете видеть, что в основном методе мы пытались уничтожить объект `obj1`, присвоив значение `null obj1 = null ;`, но на самом деле этот объект не разрушен, поэтому нет причин использовать деструкторы в реальных Java-программах. Также вы можете видеть, что у нас есть операторы «new» в каждой строке, где созданы объекты, но не «delete» операторов, потому что все объекты, созданные в сборке мусора, а не в памяти кучи и сборке мусора, будут автоматически очищены виртуальной машиной Java после выполнения вашей программы.

Вывод программы показывает, что если объект класса, созданного с помощью пустых скобок, конструктор по умолчанию выполнен автоматически. Перегруженные конструкторы будут выполняться только в том случае, если вы передадите соответствующие параметры. В Java, как в C ++, есть наследование, это означает, что вы можете создать класс на основе некоторого класса. В следующем примере мы создадим Class1 сначала Class2 на основе Class1, чтобы показать наследование, все строки кода Java, пронумерованные, чтобы иметь ссылки в тексте:

```
1  class Class1 //Class1 is superclass of Class2
2  {int a=1;
3  protected float b=2;//protected look like public in Java
4  void f1()
5  {
6      System.out.println("Method f1 of Class1");
7  }
8  }
9
10 class Class2 extends Class1//Class2 is subclass of Class1
11 {
12     void f2()
13     {
14         System.out.println("Method f2 of Class2:");
```

```

15     System.out.println("I can call data members and
16 methods of Class1:");
17     System.out.println("a="+a+" b="+b);
18     f1();//We call method f1 of Class1 directly
20     int a=5;//from this moment we have new local variable
21 a
22     //now we can access a of Class1 using keyword super:
23     System.out.println("local a="+a+" but a of
24 Class1="+super.a);
25 }
26
27 }
28
29 public class Example{
30     //This is public class with main method
31     //The code on this page located in the file
32 Example.java
33     static public void main(String[] args)
34     {Class2 obj=new Class2();
        obj.b=100;//b is protected but can be accessed directly even
        outside a
        //class
        obj.f2();
        }
    }

```

Результатом этой программы будет:

Method f2 of Class2

I can call data members and methods of Class1: a=1 b=2.0

Method f1 of Class1

local a=5 but a of Class1=1

Программа содержит 3 класса: Class1, Class2 и Example. Последний класс является общедоступным, поэтому вся программа находится в файле Example.java. Класс с именем «Class1» является корневым классом, созданным без наследования, пусть «from zero». Класс содержит два элемента данных a и b и один метод f1 (). Члены класса также называются в Java как «fields if the class». Начальные значения элементов данных, которые вы можете назначить непосредственно в классе.

В строке 10 нашего примера вы можете увидеть определение класса 2, созданного с использованием Class1. Вы можете увидеть ключевое слово «extends», что означает, что Class2 унаследовал все от Class1. Внутри Class2 мы имеем только один метод f2 (), но все члены данных и методы, унаследованные от Class1, также доступны в Class2. В строке 16 (method f2 () class 2) мы печатаем значения переменных a и b, объявленных в Class1. Затем в строке 17 мы назвали метод f1 (), расположенный в Class1. В строке 18 мы создали новую локальную переменную a и присвоенное значение 5, начиная с этого момента мы можем получить доступ к переменной «a» class1, используя ключевое слово «super», что означает перейти к суперклассу (родительский класс) и получить определенный член класса после десятичной точки «.», В нашем случае это элемент данных «a» class1, см. Строку 20.

В строке 29 мы создали объект «obj» класса Class2. Затем в строке 30 мы получили доступ к защищенному элементу данных «b», объявленному в Class1. Поскольку вы видите, что «protected» элемент данных выглядит как открытый, его можно получить непосредственно в классе, где он был объявлен, или в любом другом подклассе.

Напишите программу Java, которая печатает все реальные решения квадратичного уравнения. Решение квадратичного уравнения должно быть реализовано в классе Quadratic. Класс имеет 3 общих элемента данных **a**, **b**, **c**. Вы должны создать методы класса:

Конструктор по умолчанию класса: назначить нулевые значения: $a = 0$, $b = 0$, $c = 0$ для членов данных класса;

Void inputdata (); Этот метод попросит пользователя ввести номера a, b, c с помощью ПК-клавиатуры, все номера должны быть назначены элементам данных a, b и c.

Int checksolution (); Этот метод возвращает число корней (0, если отрицательное значение **Discriminant (D)**, 1, if $D = 0$, 2, if $D > 0$) или распечатка «Нет входных данных», если метод inputdata () не выполнен. $D = b^2 + 4ac$.

Void printoutresults (); Этот метод печатает корни квадратного уравнения. Если квадратичное уравнение не имеет корней, то распечатка метода «Нет корней», если только один корень квадратного уравнения представляет собой распечатку метода «x = значение корня», если два корня будут напечатаны «x1 = root1 и x2 = root2». Этот метод распечатывает «Нет входных данных», если

метод `inputdata ()` не выполнен. Значения корней, которые вы можете вычислить, используя уравнение: $x = \frac{-b \pm \sqrt{D}}{2a}$, where $D=b^2+4ac$.

^{1.2} $2a$

Запишите текст класса и основного метода, чтобы создать объект класса Квадратичный и вычислить корни некоторых квадратичных уравнений.

3.4 Содержание отчета

1. Титульный лист
2. Задание на лабораторную работу
3. Листинг программы
4. Скриншот работы приложения

Лабораторная работа № 4. Абстрактные классы и интерфейсы

4.1. Цель работы: Расширенные возможности объектно-ориентированного программирования в Java: абстрактные классы, полиморфизм, интерфейсы классов Java.

4.2. Краткие теоретические сведения:

В реальном программировании несколько раз вы можете встретить ситуации, в которых вы хотите определить суперкласс, который объявляет некоторые методы, не предоставляя полную реализацию каждого метода. В этом случае эти суперметоды имеют имя «абстрактный». Итак, другими словами, абстрактный класс является таким классом, в котором какой-либо метод (или более одного метода) объявлен, но не определен. В этом случае метод является виртуальным с точки зрения C++ и абстрактным с точки зрения Java. Абстрактные методы не имеют реализации (body). Определение методов (creation the body) вы дадите в подклассе абстрактного класса. Если какой-то класс является абстрактным, вы не можете создать объект этого класса, другими словами этот класс не может быть создан, но должен быть унаследован.

Например, предположим, что мы хотим что-то создать: может быть автомобиль, может быть вертолет, может быть велосипед, может быть какой-то другой автомобиль. Конечно, мы можем сказать, что в общем случае вертолет, автомобиль и велосипед являются транспортным средством, и у каждого транспортного средства есть некоторые общие черты, такие как количество колес, цвет и некоторые инструкции по использованию этого автомобиля. Но, к сожалению, инструкции «идут» на бицикл, для автомобиля и для вертолета разные. Любой водитель автомобиля не может управлять вертолетом, а пилот вертолета не может управлять автомобилем. Теперь вы можете видеть, что некоторые особенности наших автомобилей распространены (например, автомобиль, велосипед и вертолет имеют цвет и количество колес), некоторые другие функции, такие как «как управлять», абсолютно разные. Как описать создание наших автомобилей с точки зрения объектно-ориентированного программирования? Эту ситуацию можно решить с помощью абстрактных классов. Предположим, что абстрактный класс «Автомобиль» описывает общие черты автомобиля и вертолета, такие как количество колес и цвет. Также автомобиль содержит метод «go ()», чтобы описать набор

инструкций для вождения автомобиля или управления вертолетом. Мы не можем предсказать, что такое подкласс класса «Транспортное средство», и мы не можем сказать, что представляет собой набор инструкций по вождению, поэтому мы можем оставить метод «go ()» пустым, без выполнения инструкций по вождению. Другими словами: сначала мы должны создать абстрактный класс «Транспортное средство», в котором содержится абстрактный метод «go ()», затем, используя наследование, мы создадим классы «Автомобиль» и «Вертолет» с использованием абстрактного класса «Транспортное средство». В классе «Автомобиль» мы реализуем метод «go ()» с использованием набора инструкций по вождению автомобиля, в классе «Вертолет» мы будем использовать другие инструкции в методе «go», которые подходят для пилотирования вертолетов. Иерархия классов объясняется на рисунке 5.1.

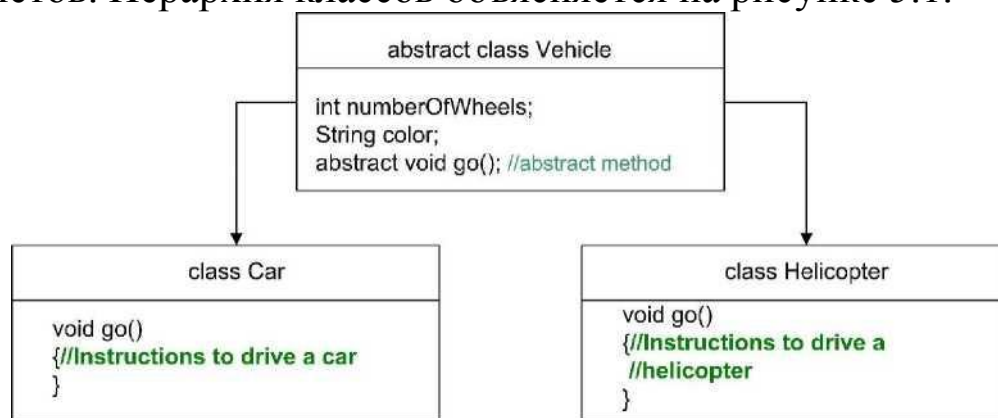


Рисунок 5.1. Иерархия классов

Программа иллюстрирует создание абстрактного класса и получение подклассов с использованием абстрактного суперкласса.

```

abstract class Vehicle//This is abstract class { int numberOfWheels=4;
  String color;//Color of our vehicle
  abstract void go();//this is abstract method "go()" }
  
```

```

class Car extends Vehicle //Class that creates a car from vehicle {
  void go()
  {
    System.out.println("Set of instructions to drive a car");
  } }
  
```

```

class Helicopter extends Vehicle//class that creates a helicopter {
  void go()
  
```

```
{  
    System.out.println("Here instructions to flight your helicopter");  
}
```

```
public class Test{  
    static public void main(String[] args)  
    {Vehicle myvehicle=new Car();//Create a car  
    System.out.println("Car created");  
    myvehicle.go();  
    System.out.println();  
    myvehicle=new Helicopter();//Create a helicopter  
    System.out.println("Helicopter created");  
    myvehicle.go();  
    } }
```

Результатом этой программы будет: Car created
Set of instructions to drive a car

Helicopter created
Here instructions to flight your helicopter

Теперь вы можете увидеть что-то интересное, обратите внимание на первую строку основного метода:

Транспортное средство `myvehicle = new Car ()`; объект `myvehicle` является объектом класса `Vehicle`, созданного с использованием класса `Car ()`. Похоже, что `myvehicle` является «Автомобилем» «Автомобиль» в текущем случае. Затем строка `myvehicle.go ()` иллюстрирует вызов метода `go ()` объекта `myvehicle ()`; Поскольку этот объект создан как автомобиль, будет выполнен метод класса «car», который даст вам выход «Набор инструкций для вождения автомобиля». Затем в строке `myvehicle = new Helicopter ()` вы можете увидеть, что тот же объект «`myvehicle`» воссоздан с использованием класса «`Helicopter`». Таким образом, объект `myvehicle` может быть автомобилем или вертолетом по вашему выбору. Эта функция называется ООР как «полиморфизм», это имя означает, что «`myvehicle`» имеет много форм: в нашем примере это может быть автомобиль и вертолет. Теперь вызовите метод `myvehicle.go ()`, который вызывает выполнение метода `go ()` класса `Helicopter`. Это также полиморфизм, потому что метод «`go ()`» иногда содержит инструкции для вождения автомобиля или иногда инструкции для вождения вертолета.

В отличие от C++ в Java нет множественного наследования. Множественное наследование - это ситуация, когда подкласс создается с использованием более одного суперкласса. Чтобы использовать множественное наследование в Java, вы можете использовать некоторую другую функцию, называемую «Java-интерфейсы». Интерфейс выглядит как абстрактный класс, но интерфейс «абсолютно абстрактный», все методы в интерфейсе не имеют тела. Все члены данных интерфейса имеют префикс «final». Другими словами, интерфейс Java является абстрактным классом, который содержит только абстрактные методы и конечные члены данных. Следующий пример иллюстрирует создание программы, которая вычисляет квадрат прямоугольника и круга с использованием и высоты фигур. Ниже представлена иерархия классов:

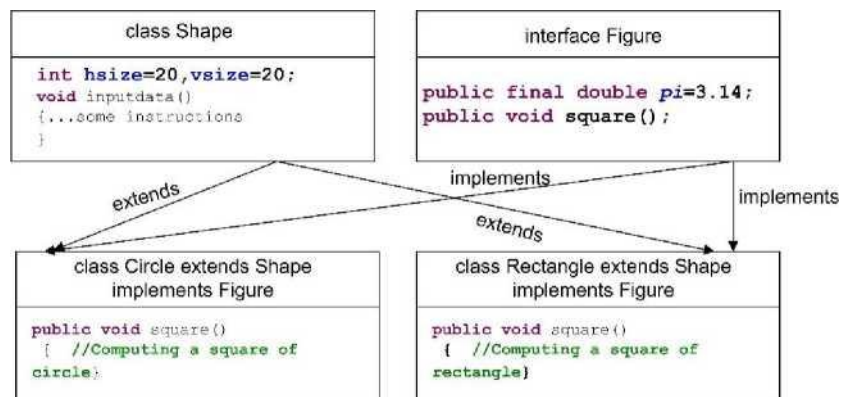


Рисунок 5.1. Иерархия классов

```

interface Figure //This is general interface for all figures {
    public final double pi=3.14;
    public void square();
}
  
```

```

class Shape //This is root class Shape with datamembers and method {
    int hsize=20, vsize=20; //default size of the figure

    void inputdata(int sizex, int sizey)
    { hsize=sizex; //Example of method with 2 parameters vsize=sizey; }
}
  
```

```

class Circle extends Shape implements Figure
  
```



```

{
    public void square()//Computing a square of circle { double r=vsize/2;
        double result=r*r*pi;
        System.out.println("Square of the circle="+result);
    }
}

```

class Rectangle **extends** Shape **implements** Figure

```

{
    public void square()//Computing a square of rectangle
    { double result=hsize*vsize;
        System.out.println("Square of the rectangle="+result);
    } }

```

```

public class Test{
    static public void main(String[] args)
    {Rectangle rec=new Rectangle();
    rec.square();
    Figure myfigure=new Rectangle();
    myfigure.square();
    myfigure=new Circle();
    myfigure.square();
    }
}

```

Результатом этой программы будет:

Square of the rectangle=400.0

Square of the rectangle=400.0

Square of the circle=314.0

В основной функции нашей программы мы можем создать объект «rec» класса Rectangle, затем вызывается метод square (). Чтобы использовать класс Circle, вы также можете создать объект этого класса. С другой стороны, классы Circle и Shape имеют общий интерфейс Figure, это означает, что вы можете создать один объект «myfigure» типа интерфейса «Figure» и использовать этот объект в качестве экземпляра классов Circle и Shape у них:

Figure myfigure = new Rectangle (); Эта строка означает, что объект myfigure () является прямоугольником, а метод square возвращает

квадрат прямоугольника. В следующей строке кода Java `Myfigure = new Circle ()` объект для переназначения на новый экземпляр класса `Circle`. Как вы уже знаете, эта функция имеет полиморфизм имен.

4.3. Задание для лабораторной работы.

Напишите java-программу для создания абстрактного класса с именем `Shape`, который содержит пустой метод с именем **`numberOfSides ()`**. Пропустите три класса с именем `Triangle`, `Square` и `Line`, чтобы каждый из классов расширил класс `Shape`. Каждый из классов содержит только метод **`numberOfSides ()`**, который показывает количество сторон в данных геометрических фигурах.

4.4. Содержание отчета

1. Титульный лист
2. Задание на лабораторную работу
3. Листинг программы
4. Скриншот работы приложения

Лабораторная работа №5. Введение в разработку графического интерфейса пользователя

5.1 . Цель работы: Введение в программирование графического интерфейса. Операции ввода-вывода и взаимодействия с пользователем с использованием класса `JoptionPane`, статические методы `JoptionPane`.

5.2 Краткие теоретические сведения

Графический пользовательский интерфейс (GUI) представляет собой удобный для пользователя механизм взаимодействия с приложением. Графический интерфейс (произносится как «ГОО-ее») дает приложению отличительный «внешний вид». Графические интерфейсы создаются из компонентов GUI. Иногда они называются элементами управления или виджетами - для оконных гаджетов. Компонент GUI - это объект, с которым пользователь взаимодействует с помощью мыши, клавиатуры или другой формы ввода. Многие IDE (программы, которые позволяют графический интерфейс проектирования) предоставляют инструменты проектирования графического интерфейса, с помощью которых вы можете указать точный размер и местоположение компонента визуально. IDE генерирует для вас код GUI. Хотя это значительно упрощает создание графических интерфейсов, вы не можете понять все свойства и события компонентов. По этой причине мы написали GUI-код руками.

При разработке Java-программы важно выбрать соответствующие компоненты графического интерфейса пользователя (GUI) Java. Есть два основных набора компонентов, с которыми вы, скорее всего, будете строить свои Java-программы. Эти две группы компонентов называются абстрактным инструментарием окна (AWT) и Swing. Обе эти группы компонентов являются частью Java Foundation Classes (JFC). Оба пакета содержат много классов, которые позволяют создавать графические интерфейсы в ваших программах. Чтобы использовать его, вы должны открыть соответствующие пакеты: `import javax.swing. *;` Для компонентов `swing` и импорта `java.awt. *;` Для компонентов `awt`.

Какие компоненты лучше? Ниже мы обобщили оба пакета для описания преимуществ и недостатков каждого из них.

Обзор AWT

AWT означает абстрактное окно ToolKit. Инструмент Abstract Window Toolkit поддерживает графическое программирование на Java. Это портативная графическая библиотека для автономных приложений и / или апплетов. Инструментарий Abstract Window Toolkit обеспечивает соединение между вашим приложением и собственным GUI. AWT обеспечивает высокий уровень абстракции для вашей Java-программы, поскольку он скрывает вас от базовых деталей графического интерфейса, в котором будет работать ваша программа. AWT включает в себя: богатый набор компонентов пользовательского интерфейса; Надежная модель обработки событий; Графические и визуальные инструменты.

Swing реализует набор компонентов GUI, которые основаны на технологии AWT и обеспечивают удобный внешний вид. Возможности Swing включают в себя: все функции AWT, 100% чистые Java сертифицированные версии существующего набора компонентов AWT (Button, Scrollbar, Label и т.д.); Богатый набор компонентов более высокого уровня (например, древовидный вид, окно списка и панели с вкладками). Компоненты Swing не зависят от операционной системы для обеспечения их функциональности. Таким образом, эти компоненты часто называют «легкими» компонентами.

В общем, компоненты AWT подходят для простой разработки или разработки апплетов, ориентированных на определенную платформу (например, программа Java будет работать только на одной платформе).

Для большинства других разработок Java GUI вы захотите использовать компоненты Swing. Также обратите внимание, что компоненты Oracle NetBeans IDE и Borland с добавленной стоимостью, включенные в JBuilder, например, dbSwing и JBCL, основаны на компонентах Swing, поэтому, если вы хотите использовать эти компоненты, вы захотите основать свою разработку на Swing.

JOptionPane (пишется как есть) является частью java swing library. Для этого требуется оператор импорта в верхней части программы. Вот что такое утверждение: **import** javax.swing.JOptionPane;

or

import javax.swing.*;

Статические методы в классе JOptionPane позволяют легко создавать модальные диалоги для отображения сообщений JOptionPane.showMessageDialog, чтобы рассказать пользователю о чем-то, что произошло;

JOptionPane.showConfirmDialog, чтобы задать подтверждающий вопрос, например yes / no / cancel.

JOptionPane.showInputDialog, чтобы ввести некоторую информацию, такую как текст или цифры;

JOptionPane.showOptionDialog, чтобы выбрать один из множества кнопок;

Каждый из этих методов возвращает int, определяющую, какая кнопка была нажата, или String, определяющая выбранную опцию.

Подробное описание всех статических методов класса JOptionPane, расположенного ниже:

JOptionPane.showMessageDialog (parentComponent, message);

Отображает модальное диалоговое окно с одной кнопкой, которая помечена как «ОК». Вы можете легко указать сообщение, значок и название, отображаемое диалоговым окном. В простейшем случае метод принимает два параметра: parentComponent (или null вместо него) и String constant «message», который является отображаемым текстом. Первый параметр «parentComponent» определяет компонент, который должен быть родителем этого диалогового окна. Он используется двумя способами: кадр, который содержит его, используется в качестве родителя кадра для диалогового окна, а его координаты экрана используются при размещении диалогового окна. В общем случае диалоговое окно размещается чуть ниже компонента. Этот параметр может быть нулевым, и в этом случае в качестве родителя используется кадр по умолчанию, и диалог будет центрироваться на экране.

Пример: кода

```
JOptionPane.showMessageDialog(  
null, "This is text Message \n to be displayed");
```

Дает вам сообщение, представленное ниже:



Вы можете выделить текст, используя одну строку или много строк, используя \ n разделитель. Также вы можете настроить заголовок сообщения и значок, который будет представлен на нем. В следующем примере используется полный синтаксис метода:

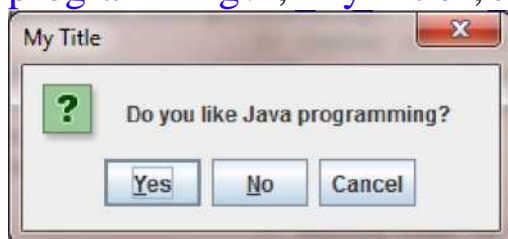
```
JOptionPane.showMessageDialog(  
null, "My message", "My title", JOptionPane.WARNING_MESSAGE);
```

Создает сообщение с текстом «My message», заголовок «My title» и значок «!». Согласно параметру

Таблица 6.1. Связь между последним параметром `showMessageDialog` и видом значка

Parameter	Icon
<code>JOptionPane.ERROR_MESSAGE</code>	X
<code>JOptionPane.WARNING_MESSA</code>	A
<code>JOptionPane.INFORMATION_ME</code>	@
<code>JOptionPane.QUESTION_MESSA</code>	s
<code>JOptionPane.PLAIN_MESSAGE</code>	none

Метод `JOptionPane.showConfirmDialog` может использоваться для запроса некоторого вопроса от пользователя. В приведенном ниже коде показан простой пример: `int n = JOptionPane.showConfirmDialog(null, "Do you like Java programming?", "My Title", JOptionPane.YESNOCANCELOPTION);`



Код создает диалог, пользователь может нажать кнопку «Да», выход этого метода (переменная «n») в этом случае будет равен нулю. Если пользователь нажимает «Нет», значение n будет равно 1. Если пользователь нажимает «Отмена», значение «n» будет равно 2, в противном случае, если пользователь нажимает клавишу «ESC» или нажимает X, n будет равным -1, который указывает на ситуацию, когда ни одна из кнопок не нажата. Последний параметр метода содержит стандартный набор кнопок (например, да, нет), но если вы хотите, вы можете создать несколько нестандартных кнопок, как в примере ниже:

```
String[] buttons = { "I like", "I don't like", "I don't know" };
```

```

int n = JOptionPane.showOptionDialog(null, "Do you like Java
programming?");//Message
"A Simple Question",//Title
JOptionPane.YES_NO_OPTION, //Will be ignored
JOptionPane.QUESTION_MESSAGE, //Icon null, //no custom icon
buttons, //the titles of buttons
buttons[0]); //default button title

```



Как вы можете видеть, текст внутри кнопок генерируется в соответствии с данными строковых массивов «Кнопки». По аналогичному методу вы можете создать любое количество кнопок, содержащих любой текст. Выходной сигнал метода будет присвоен переменной «n», которая содержит число нажатой кнопки (отсчитывается с нуля) или -1, если ни одна из кнопок не нажата.

Метод `JOptionPane.showInputDialog` может использоваться для чтения любых данных от пользователя. Это может быть набор символов или цифр. Следующий пример иллюстрирует простую программу, которая сначала спрашивает пользователя о его имени, а затем показывает приветственное сообщение: **import** javax.swing.*;

```

public class Test

```

```

{
static public void main(String[] args)
    {String st=JOptionPane.showInputDialog("What is your name?");
      JOptionPane.showMessageDialog(null,"Hello my dear "+st);
    }
}

```



Результат этой программы для ввода «Mohammad» представлен ниже:

Несколько раз вы можете дать пользователю сделать выбор из

многих вопросов. В этом случае вы можете использовать другой синтаксис метода `showMessageDialog`. Следующая программа показывает пользователю список имен, затем просит выбрать любое любимое имя, а затем показывает, какое имя выбрано.

```
import javax.swing.*;
public class Test
{ static public void main(String[] args)
  {
    String[] questions "Hakeem"}; {"Mohammad", "Abdallah", "Ahmad",
    //questions is array of String
    String st;//st is String variable
    st=(String)JOptionPane.showInputDialog(null, "Choose your favorite name",//Text of dialog "Customized Dialog",//Title
    JOptionPane.ERROR_MESSAGE,//Icon null,//no custom icon
    questions,//array of String questions[0]);//Default question
    JOptionPane.showMessageDialog(null,st);
  }
}
```



Результат этой программы представлен ниже:

Последним важным методом является `JOptionPane.showOptionDialog`, наиболее универсальный по сравнению с предыдущими методами. Этот метод отображает модальный диалог с указанными кнопками, значками, сообщением, заголовком и т. Д. С помощью этого метода вы можете изменить текст, который появляется на кнопках стандартных диалогов. Вы также можете выполнять многие другие виды настройки.

Следующий пример иллюстрирует этот метод

```
String[] options = {"Yes, please", "No, thanks", "I don't like Java!!!"};
int n = JOptionPane.showOptionDialog(null,
"Do you want new books about Java?",//Message
"A Silly Question",//Title
JOptionPane.YES_NO_CANCEL_OPTION,//default buttons ignored
JOptionPane.QUESTION_MESSAGE,//Icon null,//Custom icon ignored
```


options, //Array of String options[2]); //Default answer

Результат этой программы представлен ниже:



Согласно клику пользователя значение переменной «n» будет равно 0,1,2 или -1, если ни одна из кнопок не выбрана.

Следующая страница руководства представляет собой задание для этой лабораторной работы.

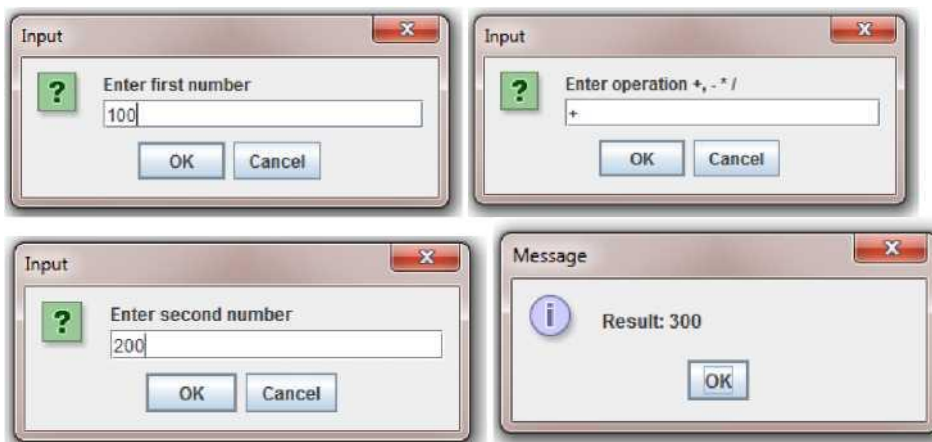
5.3 Задание для лабораторной работы.

Сделать простой калькулятор Java. Калькулятор - это программа, которая позволяет пользователю вводить первое число, затем математическую операцию (+, -, *, /), затем второй оператор затем вычисляет и показывает результат:

```
String st1=String.valueOf(i);
```

5.4 Содержание отчета

1. Титульный лист



Подсказка: преобразование из String в int

```
String st="5";  
int i=Integer.valueOf(st);
```

Преобразование из int в String

```
int i=123;
```

2. Задание на лабораторную работу
3. Листинг программы
4. Скриншот работы приложения

Лабораторная работа №6. Компоненты Java и модель делегирования событий

Цель работы: Создание приложений форм с использованием компонентов JAVA. Обработка событий и событий с использованием интерфейсов Java.


6.1 Краткие теоретические сведения

Java включает библиотеки, обеспечивающие многоплатформенную поддержку объектов графического интерфейса пользователя. «Мультиплатформенный» аспект заключается в том, что вы можете написать программу для разных операционных систем, таких как Linux, Unix, Mac OS, и у вас появятся одни и те же графические объекты, когда программа будет запущена в UNIX или Windows или Mac OS. Компоненты GUI Java включают метки, текстовые поля, текстовые области, кнопки, всплывающие меню и т.д. В SWING Toolkit также входят контейнеры, которые могут включать эти компоненты. Контейнеры включают фреймы (окна или формы), холсты (которые используются для рисования) и панели (которые используются для группировки компонентов). Панели, кнопки и другие компоненты могут быть размещены либо непосредственно в кадрах, либо в панелях внутри фреймов. Эти компоненты GUI автоматически рисуются всякий раз, когда отображается окно, в котором они находятся. Таким образом, нам не нужно явно вводить команды для их рисования. Действия с этими компонентами GUI обрабатываются с использованием модели событий Java. Когда пользователь взаимодействует с компонентом (щелкает по кнопке, вводит в поле, выбирает из всплывающего меню и т.д.), Событие генерируется компонентом, с которым вы взаимодействуете. Для каждого компонента вашей программы программист должен указывать один или несколько объектов для «прослушивания» событий из этого компонента. Если обнаружено какое-либо событие, код, связанный с этим событием, должен выполняться автоматически. Название кода, которое будет выполнено в случае какого-либо события, - это «обработчик событий».

Остальная часть этой лабораторной работы обсуждает некоторые компоненты GUI, которые позволяют разработчикам

приложений создавать полную программу. В таблице 7.1 показаны несколько базовых компонентов GUI Swing, которые мы обсуждаем сегодня.

Таблица 4.1. Список компонентов графического интерфейса пользователя

Название компонента	Описание	Внешний вид
JFrame	Это класс-контейнер, база программы, окно, содержащее все остальные компоненты программы. В .NET (Microsoft Visual Studio) аналогичный класс имеет имя «Форма».	
JButton	JButton - это класс в пакете javax.swing, который представляет кнопки на экране. Этот компонент запускает событие при щелчке мышью. Аналогичный компонент в .NET имеет имя «Button».	
JLabel	Отображает неотредактируемый текст и / или значки. В .NET подобный класс имеет имя "Label"	
JTextField	Обычно получает вход от пользователя, строку текста. В .NET подобный	

Компонент JFrame является базой программы, он выглядит как окно, в котором расположены кнопки, метки и другие компоненты. Размер JFrame, цвет, местоположение можно контролировать с помощью свойств и методов JFrame. Свойства - это некоторые переменные внутри класса, которые отвечают за его внешнее представление. Методы - это команды, которые должны выполняться объектом класса. Наиболее популярными методами класса JFrame являются:

- Show();** - показать форму на экране;
- GetWidth ();** - этот метод возвращает ширину формы, количество пикселей;
- SetSize (int, int);** - настроить новый размер формы. Два целочисленных параметра описывают соответственно ширину и высоту формы;

GetHeight (); - этот метод возвращает фактическую ширину формы, представленной в пикселях;

SetLocation (int, int); - установить новое положение формы на экране, два целочисленных параметра - X и Y формы;

SetTitle (string); - этот метод изменит название формы, как показано в таблице 1, название «Hello world» .

Наиболее популярные свойства и методы компонентов JButton, JLabel, JTextField: **GetText ();** - метод, возвращающий текст, отображаемый компонентом;

SetText ("String object"); Определяет текст, который будет отображаться компонентом;

SetCursor (Cursor cursor); Определяет форму курсора, когда указатель мыши будет находиться в границах компонента;

setLocation (x, y); Определяет новое положение компонента в форме согласно параметрам x и y. Этот метод действителен, если ранее не были определены макеты. Тема о макетах (распределение компонентов в форме) будет поднята в следующей лабораторной работе.

getLocation (); Возвращает точку (переменная с типом данных «point»), которая содержит информацию о местоположении компонента, x и y

Все другие методы будут рассмотрены в будущем.

Теперь мы можем создать наше первое графическое приложение, как уже упоминалось ранее, JAVA-программа является дочерней (подклассом) класса JFrame, чтобы создать форму, которую вы должны создать подкласс класса JFrame. Следующий пример иллюстрирует создание окна вашей программы: **import** java.awt.*;

import javax.swing.*;

class MyFrame **extends** JFrame

{

public MyFrame()

 { //Default constructor of your class //you can place any code here

 } }

public class Example

{ //The class-driver

public static void main(String[] args)

{

```

MyFrame f = new MyFrame(); // create object of the class
f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
f.setSize(250,150); //Set size of the frame
f.setVisible( true ); // display frame
} // end main
}

```



Рисунок 4.1. Форма (окно), созданная кодом

Во-первых, в программе мы создали класс «MyFrame», используя наследование от класса «JFrame». Затем в основном методе класса «Example» был создан объект «f» класса «MyFrame», этот объект является нашей программой. Вы можете управлять размером формы, используя метод `setSize` (ширина, высота), заголовок, используя метод `setTitle` («String object»), положение в соответствии с набором `KOMaHgLocation` (x, y) и многими другими функциями вашей программы. инструкция `f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);` Требуется, чтобы виртуальная машина Java закрывала программу в случае нажатия X формы или нажатия клавиш ALT + F4.

Потому что любая профессиональная программа содержит некоторые другие компоненты, выделенные в Форме. В следующем примере показано, как создать программу, содержащую форму (класс `JFrame`), две кнопки (`JButton`), метку (`JLabel`) и поле для ввода текста (`JTextField`).

(`JButton`), label (`JLabel`) and field to type a text (`JTextField`).

```

import java.awt.*;
import javax.swing.*;

```

```

class MyFrame extends JFrame {
    JButton button1=new JButton("Click me");//1

```

```

    JButton button2=new JButton("Hello world");//2
    JLabel label1=new JLabel("This is a label");//3
    JTextField text1=new JTextField("Type text here",20);//4 public
MyFrame() { super("Hello world");//5
    setLayout(new FlowLayout());//6
    this.add(label1);//7
    this.add(text1);//8
    this.add(button1);//9
    this.add(button2);//10
}
}

public class Example
{ //The class-driver
public static void main( String[] args    )
{
    MyFrame f =new MyFrame(); // create object of the class
    f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    f.setSize(250,150);//Set size of the frame
    f.setVisible( true ); // display frame
} // end main
}

```



Рисунок 4.2. Графический интерфейс, созданный программой

Все важные строки программы нумеруются с использованием комментариев. Строки 1-4 иллюстрируют определение и создание кнопок, меток и текстовых полей. Строка 5, расположенная внутри конструктора класса, вы можете увидеть инструкцию `super` («Hello words»), которая используется здесь для определения названия формы («Hello world»). Строка 6 `setLayout` (новый `FlowLayout` ()) создает раскладку кадра. Макет - это метод, описывающий, как

компоненты будут распределены по форме. Планирование потока означает, что все компоненты распределены слева направо и сверху вниз. Детали макетов будут обсуждаться в следующей лабораторной работе. Строки 7-10 иллюстрируют инструкции по добавлению компонентов в форму: `This.add («Название компонента»);` Распределение компонентов, показанных на рисунке 7.2.

Теперь пользователи вашей программы могут вводить текст внутри текстового поля или кнопки, но никаких действий, связанных с нажатием кнопки, не происходит. Другими словами у нас нет обработчиков событий для нажатия кнопки. Это означает, что очень важной частью графического интерфейса является программирование обработчиков событий.

Событие - это действие, обнаруженное программой. Существует много разных действий, например, нажатие кнопки мыши или перемещение мыши внутри формы или ввод текста с клавиатуры. Иногда некоторые события требуют выполнения некоторого Java-кода. В этом случае код должен быть выделен в специальной функции, которая имеет имя «Обработчик событий». Как правило, в Java-программах существует два способа создания обработчика событий: использование интерфейсов и использование адаптеров классов. Сегодня мы покажем вам, как обрабатывать события с помощью интерфейсов. Интерфейс в Java - это класс, который содержит только конечные переменные и абстрактные методы. В Java существует много интерфейсов, предназначенных для обработки событий, потому что существует много групп событий. Ниже мы покажем вам наиболее популярные интерфейсы, которые могут быть полезны для вас в будущем.

Интерфейс **ActionListener** находится в пакете `java.awt.event`. Этот интерфейс позволяет обнаруживать действия, связанные с нажатием кнопки. Класс, который заинтересован в обработке события, реализует этот интерфейс, а объект, созданный с помощью этого класса, регистрируется компонентом с использованием метода «`addActionListener`» компонента. Когда происходит событие действия, вызывается метод «`actionPerformed`» этого объекта:

Public void actionPerformed (ActionEvent e);

Интерфейс **KeyListener** расположен в пакете `java.awt.event`. *. Этот интерфейс предназначен для приема нажатий клавиш. Интерфейс содержит четыре метода, которые должны быть реализованы внутри

вашего класса.

Void keyTyped (KeyEvent e) Вызывается при вводе ключа

Void keyPressed (KeyEvent e) Вызывается при нажатии клавиши.

Void keyReleased (KeyEvent e) Вызывается, когда ключ был освобожден.

Интерфейс MouseListener. Этот интерфейс прослушивателя для получения «интересных» событий мыши (нажмите, отпустите, нажмите, введите и выйдите) на компонент. Ваш класс должен реализовать 5 методов, если вы хотите использовать этот интерфейс:

Void mouseClicked (MouseEvent e); Вызывается, когда кнопка мыши была нажата (нажата и отпущена) на компоненте.

Void mousePressed (MouseEvent e); Вызывается, когда мышь нажата и удерживается
Void mouseReleased (MouseEvent e); Вызывается, когда мышь выпущена

Void mouseEntered (MouseEvent e); - Вызывается, когда мышь входит в компонент.

Void mouseExited (MouseEvent e); - Вызывается, когда мышь выходит из компонента.

Интерфейс MouseMotionListener. Этот интерфейс для приема событий движения мыши на компоненте. Для кликов и других событий мыши используйте интерфейс «MouseListener». Затем объект-слушатель, созданный из этого класса, регистрируется компонентом с использованием метода «addMouseMotionListener» компонента. Событие движения мыши генерируется, когда мышь перемещается или перетаскивается. (Многие такие события будут сгенерированы). Когда происходит событие движения мыши, вызывается соответствующий метод в объекте прослушивателя, и ему передается «MouseEvent». Этот интерфейс содержит два метода:

Void mouseDragged (MouseEvent e); Вызывается, когда кнопка мыши нажата на компоненте, а затем перетаскивается.

Void mouseMoved (MouseEvent e); Вызывается, когда курсор мыши перемещен на компонент, но кнопки не были нажаты.

Следующий пример иллюстрирует программу, которая позволяет нажимать кнопки1 и кнопку2. Если нажата кнопка1, в окне отображается сообщение «Button1 clicked», если кнопка2 нажата, на

метке появится сообщение «Button2 clicked».

```
import java.awt.*;
```

```
import javax.swing.*;
```

```
import java.awt.event.*;
```

```
class MyFrame extends JFrame implements ActionListener {
```

```
    JButton button1=new JButton("Click me");//1
```

```
    JButton button2=new JButton("Hello world");//2
```

```
    JLabel label1=new JLabel("This is a label");//3
```

```
    JTextField text1=new JTextField("Type text here",20);//4
```

```
    public MyFrame()
```

```
    { super("Hello world");//5
```

```
      setLayout(new FlowLayout());//6
```

```
      this.add(label1);//7
```

```
      this.add(text1);//8
```

```
      this.add(button1);//9
```

```
      this.add(button2);//10
```

```
      button1.addActionListener(this);//11
```

```
      button2.addActionListener(this);//12
```

```
    }
```

```
    public void actionPerformed(ActionEvent event) //13
```

```
    { if (event.getSource()==button1)//14
```

```
      label1.setText("Button1 clicked");//15
```

```
      if (event.getSource()==button2)//16
```

```
        label1.setText("Button2 was clicked");//17
```

```
        text1.setText(event.getActionCommand());//18
```

```
    }
```

```
  }
```

```
public class Example
```

```
{ //The class-driver
```

```
public static void main( String[] args )
```

```
{
```

```
    MyFrame f = new MyFrame(); // create object of the class
```

```
    f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
```

```
    f.setSize(250,150);//Set size of the frame
```

```
    f.setVisible( true ); // display frame
```

```
} // end main  
}
```



Рисунок 4.3. Внешний вид программы и выполнение обработчиков событий

В приведенной выше программе содержится обработчик событий, связанный с интерфейсом «ActionListener». Внутри интерфейса есть один метод `actionPerformed`, который будет выполняться автоматически, если кнопка будет нажата мышью. Строки № 11 и 12 предоставляют регистрацию обработчика событий для кнопок 1 и 2 кнопок. Внутри обработчика событий с именем `actionPerformed` вы должны распознать объект, который генерирует это событие. Это может быть `кнопка1` или `кнопка2`. Обработчик событий содержит параметр «событие», поле «`getSource ()`» этого параметра возвращает объект, который выполняет это событие. Строки 14 и 16 сравнивают «`getSource ()`» с кнопкой 1 и кнопкой2, если условие становится равным «`true`», объект распознается. Кроме того, вы можете использовать поле `getActionCommand ()`, чтобы получить заголовок нажатой кнопки.

6.2 Задание для лабораторной работы.

1. В приведенном ниже коде представлена простая Java-игра под названием «виртуальная лягушка». Форма пуста, но если вы попытаетесь щелкнуть ее мышью, она перейдет в случайную позицию. Класс «`MyFrame`» реализует интерфейс `MouseMotionListener`, который позволяет обнаруживать движение мыши. Интерфейс содержит два метода `mouseDragged` и `mouseMoved`, оба метода должны быть реализованы в классе, даже если вы не используете какой-либо метод. Протестируйте эту программу с помощью Eclipse.

```
import javax.swing.*;
```

```
import java.awt.event.*;
```

```
class MyFrame extends JFrame implements MouseMotionListener {  
    public MyFrame()
```

```

{ super("Virtual frog");//5
  this.addMouseListener(this);
}
public void mouseDragged(MouseEvent arg0) { }

public void mouseMoved(MouseEvent arg0) {
double x=Math.random()*800;
double y=Math.random()*600;
this.setLocation((int)x,    (int)y);
} }

```

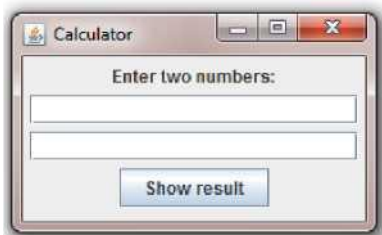
```

public class Example
{public static void main( String[] args )
{
  MyFrame f = new MyFrame(); // create object of the class
  f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  f.setSize(250,150);//Set size of the frame f.setVisible( true );
  // display frame

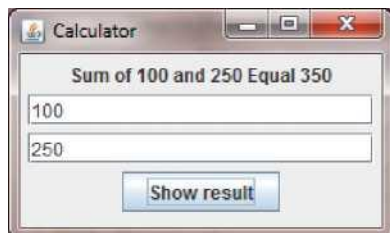
} // end main
}

```

2. Создайте простой калькулятор Java, графический интерфейс программы, представленный ниже.



В программе есть метка, которая указывает сообщение «Введите два номера», два поля (JTextField) и кнопку «Показать результат». Пользователь может набирать числа в текстовых полях, затем нажимает кнопку, программа вычисляет сумму и показывает результат в метке, см. Рисунок ниже:



6.3 Содержание отчета

1. Титульный лист
2. Задание на лабораторную работу
3. Листинг программы
4. Скриншот работы приложения

Лабораторная работа №7. Графический дизайн пользовательского интерфейса и адаптеры класса

7.1 . Цель работы: Создание обработчиков событий с адаптерами классов; Отношения между адаптерами классов и интерфейсами. Обзор компонентов JRadioButton и JCheckBox.

7.2 Краткие теоретические сведения

Во время последней лабораторной работы (лабораторная работа 7) обработчики событий в вашей программе были созданы с использованием интерфейсов Java. Любой интерфейс - это класс, содержащий только абстрактные методы; Это означает, что вы должны реализовать все методы, объявленные в интерфейсе, если вы хотите использовать этот интерфейс в своем классе. Этот способ создания обработчиков событий имеет недостаток: если вы хотите использовать только один метод некоторого интерфейса, вы должны реализовать все методы этого интерфейса. Например, вы хотите создать фрейм с меткой, если пользователь нажимает рамку в некотором положении, метка укажет положение мыши (x и y) внутри этого фрейма. Пример (1) ниже иллюстрирует эту программу:

```
import javax.swing.*;
```

```
import java.awt.*;
```

```
import java.awt.event.*;
```

```
class MyFrame extends JFrame implements MouseListener {  
    JLabel label1=new JLabel("");//1
```

```
    public MyFrame() //2  
    { setLayout(new FlowLayout());//3  
      this.add(label1);//3  
      this.addMouseListener(this);//4  
    }
```

```
    public void mouseClicked(MouseEvent e) {  
        int x=e.getX();//5  
        int y=e.getY();  
        label1.setText("Frame clicked! "+"x="+x+"    y="+y);  
    }
```

```
    public void mouseEntered(MouseEvent e){ }//6
```

```

public void mouseExited(MouseEvent e) {}//7
public void mousePressed(MouseEvent e){}//8
public void mouseReleased(MouseEvent e){}//9
}

public class Example
{ //The class-driver
public static void main( String[] args )
{
    MyFrame f = new MyFrame(); // create object of the class
    f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    f.setSize(250,150);//Set size of the frame
    f.setVisible( true ); // display frame
} // end main
}

```



Рисунок 5.1. GUI примера (1), описанного выше

Основной кадр программы, определенный в классе «Myframe», используя наследование от суперкласса «JFrame» и наследование от интерфейса `MouseListener`. Интерфейс `MouseListener`, ответственный за события, связанные с кнопками мыши и положением мыши в сторону компонентов, этот интерфейс содержит 5 методов: `MouseClicked` - это событие появляется в любое время, когда пользователь нажимает на какой-либо компонент с помощью кнопки мыши

`MouseEntered` - щелкнуть компонент во-первых, вы должны перемещать мышь внутри компонента (в нашем случае компонент представляет собой фрейм «MyFrame»). Событие `mouseEntered`, сгенерированное в тот момент, когда мышь появляется внутри компонента

`MouseExited` - обратный случай `mouseEntered`. Событие `mouseExited` генерируется автоматически каждый раз, когда мышь покидает компонент (появляется вне компонента).

`MousePressed` - событие, которое будет генерироваться в любое

время, когда вы нажимаете кнопку мыши (и удерживайте ее) `MouseEvent` - напротив `MouseEvent` event. Это событие генерируется, когда пользователь отпускает кнопку мыши. Например, одно событие `MouseEvent`, предсказанное событиями `MouseEvent` и `MouseEvent`.

Для обработки события «`MouseEvent`» мы использовали интерфейс `MouseListener`, который содержит 5 событий. Даже если вам нужно только одно событие (метод «`MouseEvent`»), вы должны реализовать все методы используемого интерфейса, см. Строки 6-9 примера.

Многие интерфейсы `event-listener`, такие как `MouseListener` и `MouseMotionListener`, содержат несколько методов. Не всегда желательно объявлять каждый метод в интерфейсе `event-listener`. Например, для приложения может понадобиться только обработчик `MouseEvent` из `MouseListener` или обработчик `MouseEvent` из `MouseMotionListener`. Интерфейс `WindowListener` задает семь методов обработки событий окна. Для многих интерфейсов слушателя, которые имеют несколько методов, пакеты `java.awt.event` и `javax.swing.event` предоставляют классы адаптера событий-слушателей. Класс адаптера реализует интерфейс и предоставляет стандартную реализацию (с пустым телом метода) каждого метода в интерфейсе.

В таблице 5.1 показаны несколько классов адаптера `java.awt.event` и интерфейсы, которые они реализуют.

Таблица 7.1. Интерфейсы и класс-адаптеры

N	Название интерфейса	Название адаптера	Описание событий
1	<code>ComponentListener</code>	<code>ComponentAdapter</code>	События компонента. Способ регистрации: <code>addComponentListener()</code> или <code>addComponentListener()</code> или <code>addMouseListener()</code>
2	<code>ContainerListener</code>	<code>ContainerAdapter</code>	События, связанные с добавлением и удалением компонентов в / из класса контейнера.

3	FocusListener	FocusAdapter	События, связанные с фокусом (когда компонент контролирует интерфейс
4	KeyListener	KeyAdapter	Ответственный за нажатие и отпускание клавиш. Способ регистрации: addKeyListener или addKeyAdapter.
5	MouseListener	MouseAdapter	Некоторые события мыши, интерфейс (и класс адаптера) содержат 5 методов: mouseClicked(), mouseEntered(), mouseExited(), mousePressed(), mouseReleased().
6	MouseMotionListener	MouseMotionAdapter	События, связанные с движением мыши. Этот класс и
7	WindowListener	WindowAdapter	События из окна. Метод регистрации addWindowListener или добавьте WindowAdapter. Содержит 7

Вы можете расширить класс адаптера, чтобы наследовать реализацию по умолчанию каждого метода и впоследствии переопределять только методы, необходимые для обработки событий.

Пример ниже (2) - пример (1), где обработка событий реализована с использованием класса адаптера «MouseAdapter» вместо интерфейса «MouseListener»:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```

```
class MyFrame extends JFrame
{
    class MyClick extends MouseAdapter//inner class
    {
        public void mouseClicked(MouseEvent e) {
            int x=e.getX();//5
            int y=e.getY();
```



```

        label1.setText("Frame clicked! "+"x="+x+"    y="+y);
    }
}
JLabel label1=new JLabel("");//1

public MyFrame() //2
{ setLayout(new FlowLayout());//3
  this.add(label1);//4
  this.addMouseListener(new MyClick());//5
}
}

public class Example
{ //The class-driver
public static void main( String[] args )
{
    MyFrame f = new MyFrame(); // create object of the class
    f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    f.setSize(250,150);//Set size of the frame
    f.setVisible( true ); // display frame

} // end main
}

```



Рисунок 7.1. GUI примера (2), описанного выше

Теперь вы можете видеть, что код примера 2 простой и короткий по сравнению с примером 1, что является причиной использования классов адаптеров вместо интерфейсов.

Компоненты Swing GUI содержат два типа кнопок состояния: классы `JCheckBox` и `JRadioButton`. `JRadioButton` отличается от `JCheckBox` тем, что обычно несколько `JRadioButtons` сгруппированы вместе и являются взаимоисключающими - только в одной группе можно выбрать в любое время, точно так же, как кнопки на автомобильном радио. Элементы данных и методы классов

JCheckBox и JRadioButton схожи, поэтому мы обсудим здесь класс JRadioButton.

Радио кнопки (объявленные с классом JRadioButton) аналогичны флажкам, поскольку они имеют два состояния - выбраны и не выбраны (также называемые отмененными). Однако радиокнопки обычно отображаются как группа, в которой может быть выбрана только одна кнопка за раз (см. Вывод на рисунке 8.2). Выбор другого переключателя заставляет все остальные отменять выбор. Радио-кнопки используются для представления взаимоисключающих опций (т. Е. Одновременно невозможно выбрать несколько параметров в группе). Логическая связь между переключателями поддерживается объектом ButtonGroup (пакет javax.swing), который сам по себе не является компонентом GUI. Объект ButtonGroup организует группу кнопок и сам не отображается в пользовательском интерфейсе. Скорее, отдельные объекты JRadioButton из группы отображаются в графическом интерфейсе. В приложении ниже (пример 3) используются переключатели, допускающие только один ответ на вопрос:

```
import javax.swing.*;
```

```
import java.awt.*;
```

```
import java.awt.event.*;
```

```
class MyFrame extends JFrame //1
```

```
{
```

```
    class MyClick extends MouseAdapter
```

```
    {
```

```
        public void mouseClicked(MouseEvent e) {
```

```
            if (rb1.isSelected())
```

```
                JOptionPane.showMessageDialog(null, "You said 'Yes'"); //2
```

```
            if (rb2.isSelected())
```

```
                JOptionPane.showMessageDialog(null, "You said 'No!'); //3 if
```

```
            (rb3.isSelected())
```

```
                JOptionPane.showMessageDialog(null, "You said  
"+rb3.getText()); //4
```

```
        }
```

```
    }
```

```
JLabel label1=new JLabel("Balqa is very good university in  
Amman?"); //5
```

```
JRadioButton rb1=new JRadioButton("Yes"); //6
```

```

JRadioButton rb2=new JRadioButton("No");//7
JRadioButton rb3=new JRadioButton("I don't know");//8
ButtonGroup bg=new ButtonGroup();//9
JButton but1=new JButton("My answer");//10

public MyFrame() //constructor of MyFrame
{
    setLayout(new BorderLayout());//11
    this.add(label1);//12
    this.add(rb1);this.add(rb2);//13
    this.add(rb3);//14
    this.add(but1);//15
    bg.add(rb1);bg.add(rb2);bg.add(rb3);//16
    but1.addMouseListener(new MyClick());//17
}
}

public class Example
{
    //The class-driver
    public static void main( String[] args )
    {
        MyFrame f = new MyFrame(); // create object of the class
        f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        f.setSize(250,150);//Set size of the frame f.setVisible( true );
        // display frame

    } // end main
}

```

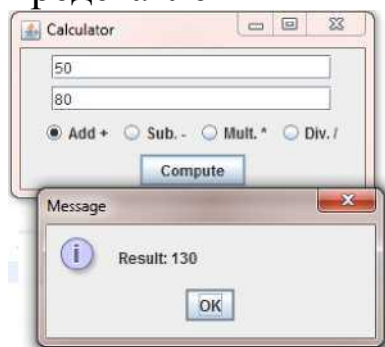


Рисунок 5.2. GUI примера (3)

7.3 . Задание для лабораторной работы:

Создайте простой калькулятор Java, графический интерфейс

программы,
представленный ниже:



GUI содержит 2 объекта `JTextField`, которые представляют собой переключатели номер 1 и номер 2, 4, каждый переключатель обозначает некоторую арифметическую операцию и кнопку для вычисления результата. По выбору пользователя программа показывает сумму числа 1 и числа 2 или продукт или любой другой результат операции. Используйте адаптер класса `MouseAdapter` для обработки event «щелчок» кнопки.

7.4 Содержание отчета

1. Титульный лист
2. Задание на лабораторную работу
3. Листинг программы
4. Скриншот работы приложения