МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра биомедицинской инженерии

Утверждаю
Проректор по учебной работе
О.Г. Локтионова
«25» 09 2023 г.

ЯЗЫК JAVA

Методические рекомендации по выполнению лабораторных работ для студентов направления подготовки 12.03.04 – «Биотехнические системы и технологии» (бакалавр)

Курск 2023

УДК 621.(076.1)

Составители: А.А.Кузьмин

Рецензент:
Кандидат технических наук, доцент *Т.Н. Конаныхина*

Язык Java: методические рекомендации по выполнению лабораторных работ для студентов направления подготовки 12.03.04 – «Биотехнические системы и технологии» (бакалавр) / Юго-Зап. гос. ун-т; сост.: А.А.Кузьмин. - Курск, 2023. - 34 с.

Содержат методические рекомендации к проведению лабораторных работ по дисциплине «Язык Java». Методические указания по структуре, содержанию и стилю изложения материала соответствуют методическим и научным требования, предъявляемым к учебным и методическим пособиям.

Предназначены для студентов направления подготовки 12.03.04 – «Биотехнические системы и технологии» (бакалавр)

Текст печатается в авторской редакции

Подписано в печать 25.09.23 Формат 60х84 1/16
Усо.печ.л. 2,0 . Уч.-изд.л. 1,9 .Тираж 30 экз. Заказ: 1076 .Бесплатно.
Юго-Западный государственный университет.
305040. г. Курск, ул. 50 лет Октября, 94.

Лабораторная работа №1.

Технология разработки кросс-платформенных приложений

1.1 Цель работы: Изучение фундаментальных блоков программ Java, программирование Java, с использование консоли операционной системы Windows, использование IDE NetBeanse 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 пользователях.

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.)

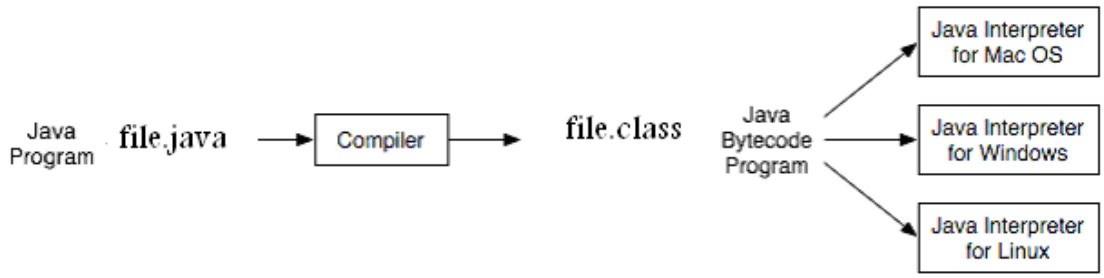

Рисунок 1.1. Создание и исполнение байт-кода

Рисунок 1.1. 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.

1.3. Порядок выполнения работы

1.3.1. 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»:

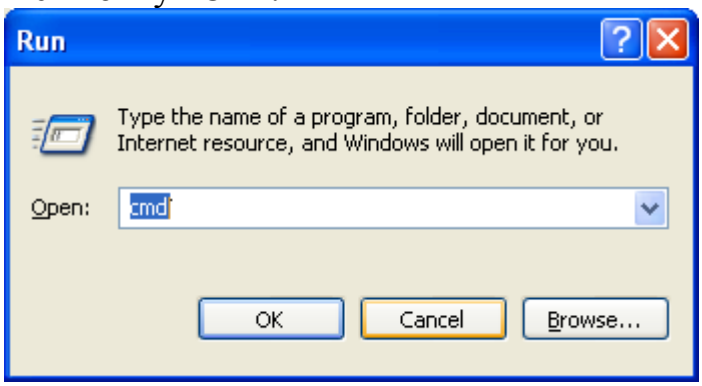

Рисунок 1.2. Выполнение команды «cmd» в Windows XP

Консольное окно операционной системы, представленное на рисунке 1.3, вы можете увидеть свое текущее местоположение в папке C: \ Documents and Settings \ admin, где admin - это имя учетной записи.

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

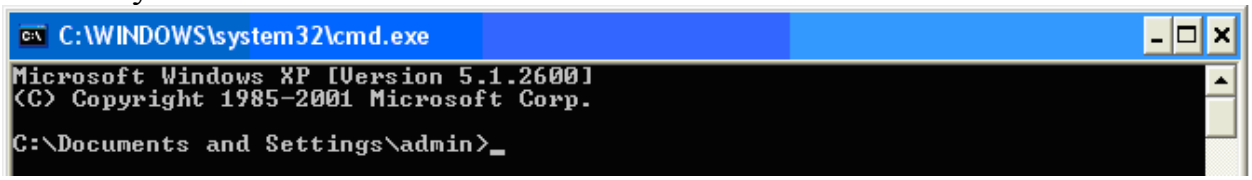

Рисунок1.3. Окно консоли операционной системы

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.

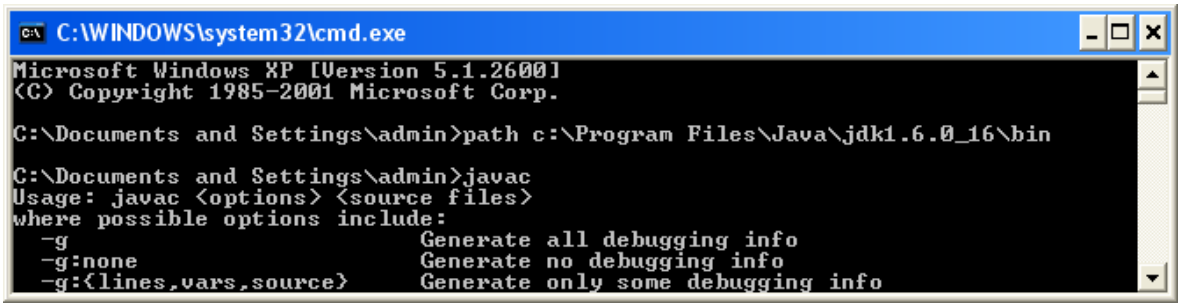

Рисунок 1.4. Добавление пути Java-компилятора к переменной окружения системы Windows.

1.3.3 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.

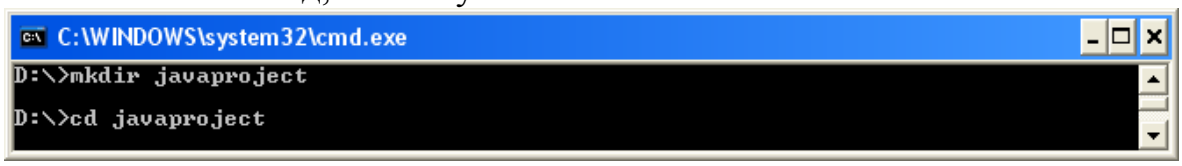

Рисунок 1.5. Создание новой папки с помощью консольной команды.

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.

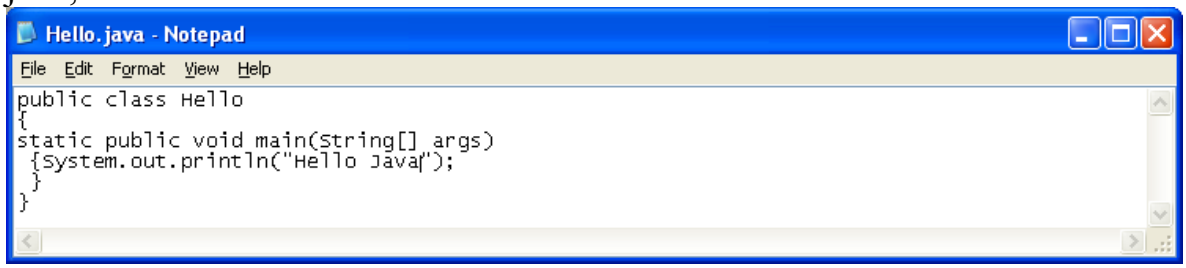

Рисунок 1.6. Создание новой программы в Notepad.

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.

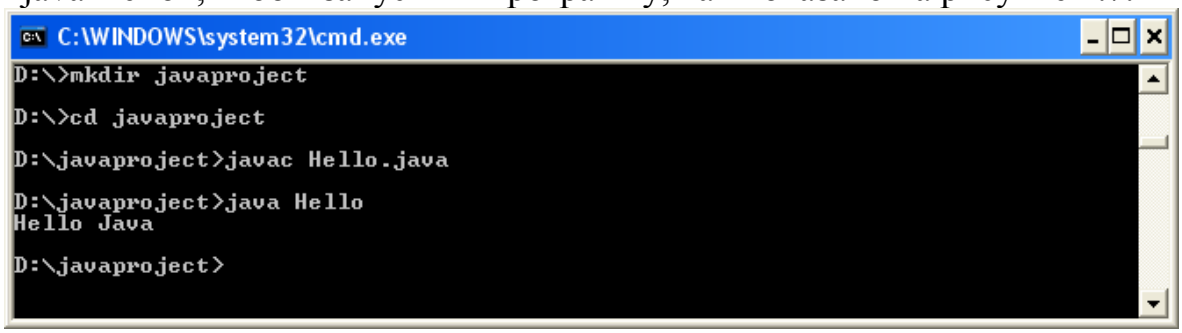

Рисунок 1.7. Консольные команды для компиляции и запуска программы Java.

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), 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.):

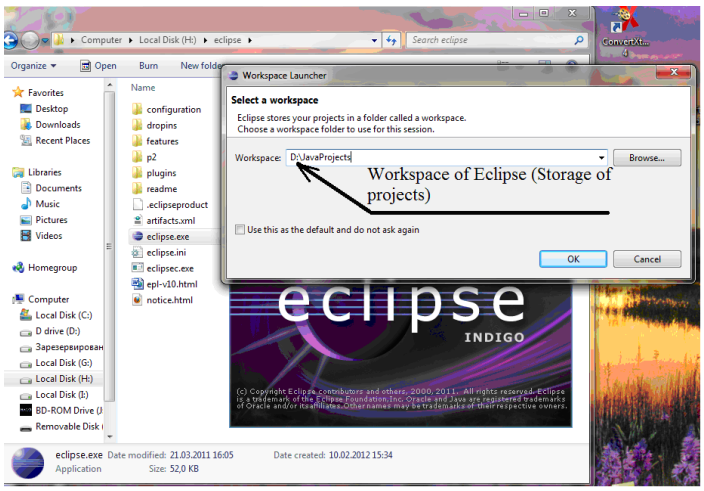

Рисунок 1.8. Eclipse Environment, запрос рабочего пространства.

Введите местоположение рабочей области: «D: \ JavaProjects \» и нажмите «OK», 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:

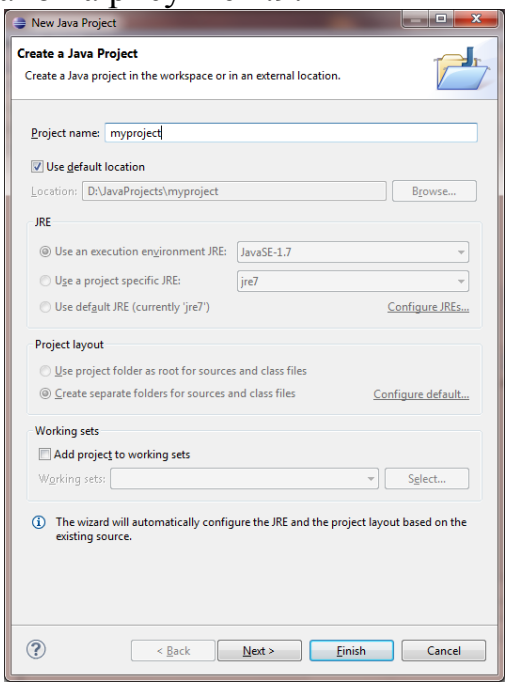

Рисунок 1.9. Создание нового проекта в Eclipse Environment

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

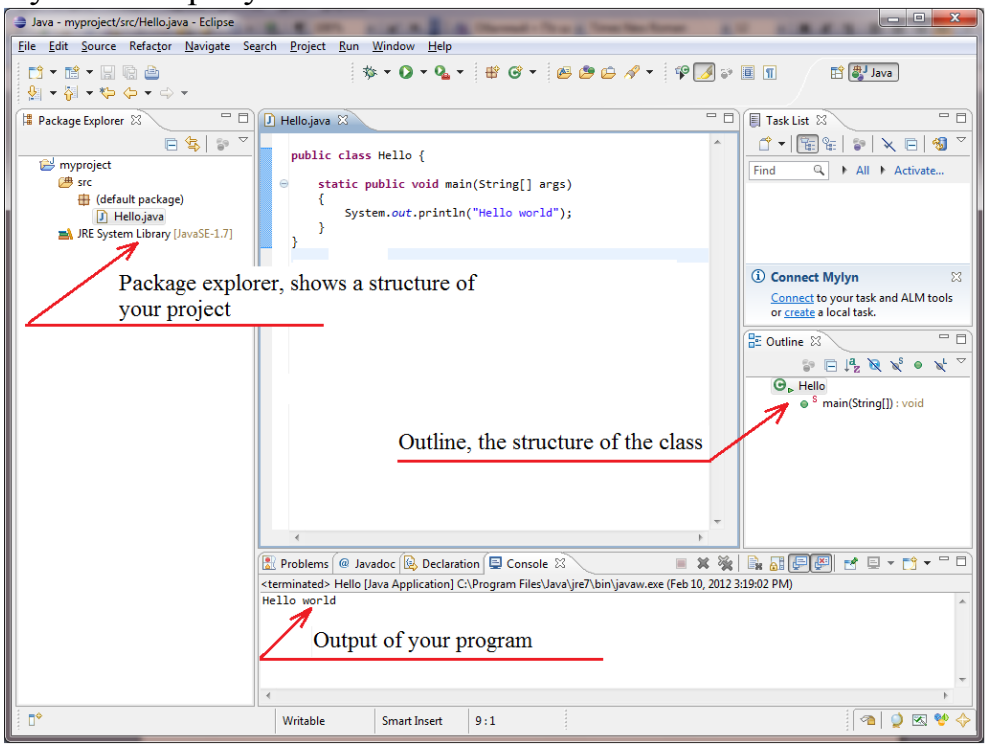

Рисунок 1.10. Основные элементы Eclipse IDE.

1.3.5 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.).

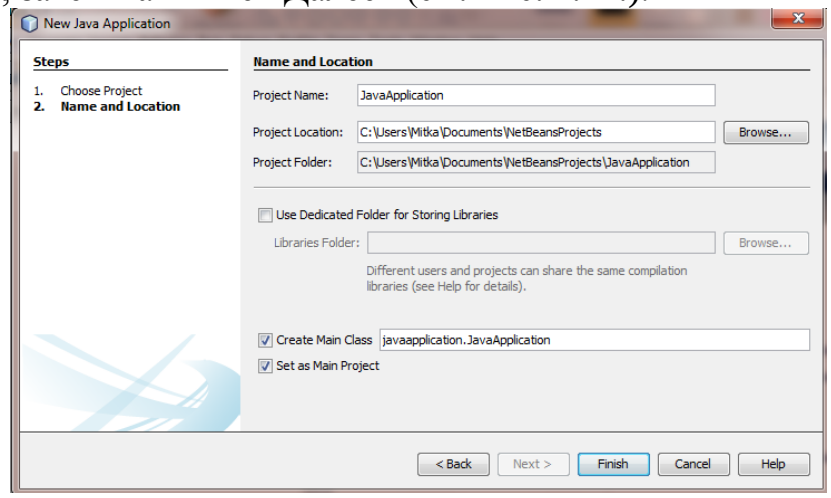

Рисунок 1.11. Создание нового приложения Java с использованием Netbeans.

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.

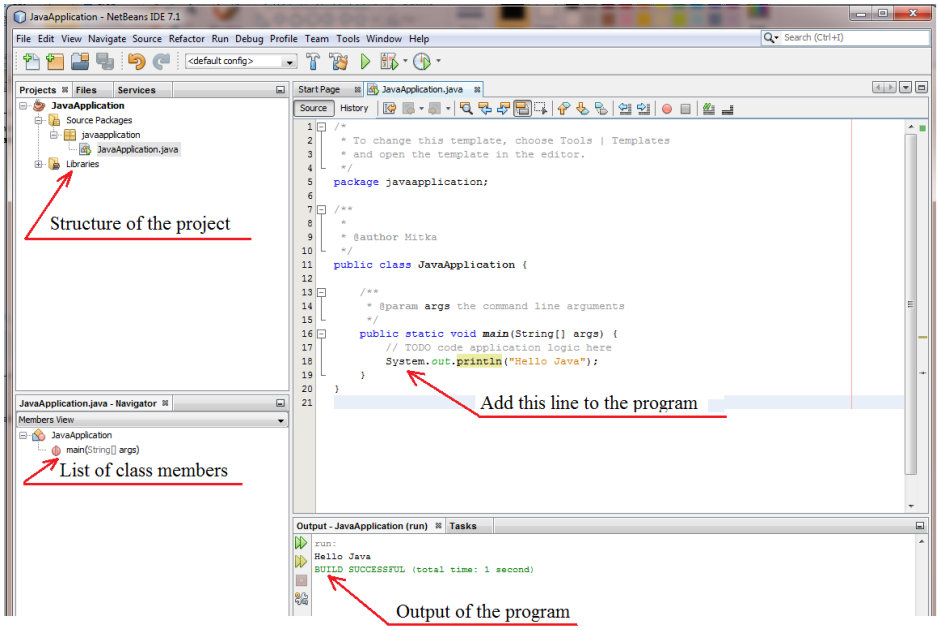

Рисунок1.12. Основные компоненты IDE NetBeans.

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).

4. Содержание отчета

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

Лабораторная работа 2

Основы Java: переменные, ввод и вывод, классы String, String Builder и Character

2.1.Цель работы: Введение в основные понятия программирования Java: обзор переменных Java и наиболее популярных классов, таких как String, String Builder и Character; Ввод и вывод; Арифметические операции и контрольные утверждения.

2.2 Краткая теория:

Язык программирования Java - это строго типизированный язык, что означает, что каждая переменная и каждое выражение имеют тип данных, который известен во время компиляции. Типы данных ограничивают значения, которые может удерживать переменная или что может выражать, ограничивают операции, поддерживаемые этими значениями, и определяют значение этих операций. Сильная типизация помогает обнаруживать ошибки во время компиляции.

2.2.1.Типы данных.

Типы данных языка программирования Java делятся на две категории: примитивные типы и ссылочные типы. Примитивными типами данных являются булевский тип и числовые типы. Числовые типы - такие типы данных, как байты, короткие, int, long и char, а типы с плавающей запятой плавают и удваиваются. Ссылочными типами являются типы классов, интерфейсов и массивов. Набор примитивных типов данных Java, представленных в таблице 2.1.

Таблица 2.1. Примитивные типы данных Java.

Data Type	Description	Size	Default Value
boolean	true or false	1-bit	false
char	Unicode Character	16-bit	\u0000
byte	Signed Integer	8-bit	0
short	Signed Integer	16-bit	0
Int	Signed Integer	32-bit	0
long	Signed Integer	64-bit	0
float	Real number	32-bit	0.0f
double	Real number	64-bit	0.0

Объявление переменных в Java аналогично языку C ++; Во-первых, вы должны определить тип данных переменной, а затем имя этой переменной, а затем значение. Например:

```
int i;//create variable i, default value is zero
int a,b,c=5;//create variables a,b,c; c=5, a and b are zeros
long k=100;//create long integer variable k, assign value 100
```

```
float r=56.5f;// create float variable r that equal to 56.5
double ahmad=67;// create variable "ahmad", data type is double, value is 67
```
Арифметические и логические операции Java похожи на операции в C ++, но есть только одно отличие: Java не может использовать разные типы данных автоматически, но C ++ может. Кастинг - это преобразование между различными типами данных, и если вы помните тему «Компьютерное мастерство2», вы можете присвоить значение дробного числа целой переменной без каких-либо проблем на C ++, но вы не можете сделать то же самое в Java. Следующий пример прояснит ситуацию:
```
int i=5;
short b=6;
i=b;/*Correct! Data type int can be equal short
    because i has 4 bytes, b has 2 bytes*/
b=i;//Mistake! Short can not be equal int
b=(short)i; //Correct! Data type int casted by Java code
          // to short data type
```
Теперь вы можете увидеть операцию распределения в последней строке нашего примера b = (short) i. Переменная b имеет тип данных short (2 байта), но переменная i - int (4 байта). Вот почему попытка назначить 4 байта на 2 байта ячейки памяти вызывает ошибку компиляции. В общем случае литье имеет форму, представленную ниже:

<new data type>=(<new data type>)<old data type>;
Другой пример распределения для дробных чисел:
```
int i=5;
double b=6.7;
float f=10.5f;
f=i;//Correct, float can be equal to int
b=f;//Correct, double can be equal to float
f=b;//Mistake, no casting by default in Java
f=(float)b;//Correct, casting provided by Java code
i=(int)b;//Correct, casting provided by Java code
```
Тип данных char. Char - это один символ, то есть буква, цифра, знак препинания, вкладка, пробел или что-то подобное. Символьный литерал - это один символ, заключенный в одиночные кавычки, подобные этому:
```
char c1='A';
char c2=65;
```
Здесь вы можете увидеть, что тип данных char может хранить целое число (в диапазоне от 0 до 65535) или символ английского или любого другого алфавита. Выход переменной char вызывает вывод символической информации:

```
public class Example{
  public static void main(String args[]){
    char c1='A';
```

```
    char c2=65;
    int i=c1;
    System.out.println(c1+"  "+c2+"  "+i);
   }
}
```

В приведенной выше программе показаны манипуляции с типом данных char. Во-первых, мы присвоили символу «A» переменную c1, затем номер 65 переменной c2, затем значение переменной char c1 для целочисленной переменной i. Вывод программы даст вам:
A A 65
Теперь вы можете видеть, что символ «A» имеет номер юникода 65.

2.2.2 Ввод и вывод в Java

Вывод в программах Java, представленных классом с именем «out», который находится в классе «System». Класс «out» содержит три самых популярных метода вывода: print, println и printf. Методы print и println аналогичны методу «out» в C ++, метод printf аналогичен printf на C ++. В приведенной ниже программе показан вывод в Java:

```
public class Example{
  public static void main(String args[]){
   int i=5;
   double d=56.78;
   System.out.print("i= ");
   System.out.println(i);
   System.out.print("d= ");
   System.out.println(d);
   System.out.println("i="+i+" d="+d);
   System.out.printf("Output is: i=%d, d=%f",i,d);
  }
}
```

Результатом этой программы будет:
i= 5
d= 56.78
i=5 d=56.78
Output is: i=5, d=56,780000
Ввод в Java, представленный классом Scanner, который находится в пакете-утилита, использует пакет, расположенный в пакете java. Чтобы сначала использовать класс Scanner, вы должны открыть библиотеку:

```
Import java.util.Scanner;
```

Во-вторых, вы должны создать объект класса Scanner:

```
Ввод Scanner = new Scanner (System.in);
```

Здесь ввод является объектом класса Scanner. Этот объект содержит много полезных методов для ввода целочисленных, дробных чисел или объектов String:

NextInt () - обеспечивает ввод целочисленного числа (тип данных int) с клавиатуры
NextShort () - то же самое, но он возвращает тип данных short
NextDouble () - этот метод подходит для дробных чисел
NextLine () - этот метод возвращает объект

Пример программы, показывающей ввод, приведён ниже:

```
import java.util.Scanner;
public class Example{
  public static void main(String args[]){
Scanner input=new Scanner(System.in);
int num1,num2,sum;
System.out.print("Enter first number: ");num1=input.nextInt();
System.out.print("Enter second number: ");num2=input.nextInt();
sum=num1+num2;
System.out.print("Sum of numbers equal "+sum);
  }
}
```

Вывод этой программы представлен ниже:

```
Enter first number: 5
Enter second number: 6
Sum of numbers equal 11
```

2.2.3 Класс Character

Класс Character предлагает ряд полезных статических методов для манипулирования символами. Статический метод означает, что вы можете использовать его напрямую, даже если объект класса не создан.
Ниже приведен список наиболее важных методов, которые реализуют все подклассы класса Character:

```
char c1='A';
boolean b;
```

b = Character.isLetter (c1); Поскольку вы можете видеть, что метод принимает символ Char и возвращает «true», если аргумент представляет букву английского или любого другого алфавита. В текущем примере b является «истинным», поскольку переменная c1 содержит символ «A», который является первой буквой английского алфавита.
b = Character.isDigit (c1); Метод «isDigit» возвращает «true», если входной аргумент находится в диапазоне от 0 до 9 или от ٠ to ٩, в противном случае вывод метода будет ложным.
b = Character.isUpperCase (c1); Этот метод возвращает «true», если входной аргумент c1 представлен символом верхнего регистра (A ... Z), иначе метод возвращает «false». Для арабского алфавита вывод «false».
b = Character.isLowerCase (c1); Согласно содержанию входного аргумента c1, метод возвращает «true», если c1 содержит строчный символ (a ... z).

c1 = toUpperCase (c1); Эти методы возвращают прописную форму указанного значения char. Если c1 является «a», то выход метода будет «A».
c1 = toLowerCase (c1); Возвращает строчную форму указанного значения char. Если значение c1 равно «A», метод возвращает «a».
В приведенной ниже программе Java показан пример типа char и класса Character.

```
public class Example{
  public static void main(String args[]){
   char c1='Z';
   if (Character.isUpperCase(c1)) c1=Character.toLowerCase(c1);
   else c1=Character.toUpperCase(c1);
   System.out.println(c1);
  }
}
```

Программа создает переменную c1 и сохраняет символ «Z» в переменной. Затем метод isUpperCase (c1) используется для проверки символа, хранящегося в переменной c1. Если символ строчный, он будет преобразован в верхний регистр, если символ в верхнем регистре, то будет преобразован в нижний регистр. Результатом программы является строчный символ «z».

2.2.4. Класс String

Строки, которые широко используются в Java-программировании, представляют собой последовательность символов. На языке программирования Java строки являются объектами класса String. Самый прямой способ создать строку - написать:

```
String st="Hello world!";
```

Всякий раз, когда он встречает строковый литерал в вашем коде, компилятор создает объект String со своим значением в этом случае «Hello world!». Как и любой другой объект, вы можете создавать объекты String с помощью нового ключевого слова и конструктора.

```
String st1= new String("Hello world!");
```

В примерах выше st и st1 являются объектами класса String, оба объекта представляют собой текст «Hello world». Строковые объекты поддерживают конкатенацию, например: «Hello» + «world» = «Hello world».
Наиболее популярные методы класса String, представленные ниже:
Char charAt (int index) - Возвращает символ по указанному индексу. Например:

```
String st="Hello world";
char c=st.charAt(0);
System.out.println(c);
```

Результатом этого кода будет: «H», потому что char в позиции 0 строки является символом «H».
Int compareTo (String anotherString) - сравнивает две строки лексикографически. Если строки равны, метод возвращает ноль, если одна

строка больше другой строки, метод возвращает отрицательное число, в противном случае - положительное число. Например:

```
String st="Hello World";
String st1="Hello Ahmad";
int i=st1.compareTo(st);
System.out.println(i);
```

Результат равен -22, это отрицательное число, потому что «Hello Ahmad» <«Hello World»

Int compareToIgnoreCase (String str). То же, что и метод compareTo, но случай не важен для этого метода.

Boolean endsWith (String suffix). Тесты, если эта строка заканчивается указанным суффиксом, возвращает true или false.

Void getChars (int srcBegin, int srcEnd, char [] dst, int dstBegin) Копирует символы из этой строки в целевой массив символов.

Int indexOf (String st) Возвращает индекс внутри этой строки первого вхождения указанного символа или строки. Этот метод можно использовать для поиска подстроки st в строке. Метод возвращает позицию подстроки, если она найдена или -1 в противном случае. Например:

```
String st="Hello World";
int i=st.indexOf("Wo");
System.out.println(i);
```

Результат равен 6, это индекс подстроки «Wo» в строке «Hello World».

Int length (). Этот метод возвращает фактическую длину строки.

String replaceAll (String regex, String replacement) и String replace (String regex, String replacement). Это аналогичные методы, заменяет каждую подстроку этой строки, которая соответствует данному регулярному выражению с указанной заменой. Например:

```
String st="Hello world";
String st1=st.replace("o","Ahmad");
System.out.println(st1);//st1 is HellAhmad wAhmadrld, st no change
st1=st.replaceAll("o", "Ahmad");
System.out.println(st1);//st1 is HellAhmad wAhmadrld, st no change
```

Boolean startsWith (String prefix) и метод boolean endsWith (String prefix) показывает, будет ли заданная строка начинаться (заканчивается) конкретными символами, например:

```
String st="Hello";
System.out.println(st.startsWith("He"));
System.out.println(st.endsWith("lo"));
```

Результат этого кода истинен, потому что «Hello world» начинается с «He» и заканчивается «lo».

String substring(int beginIndex, int endIndex). Возвращает новую строку, которая является подстрокой этой строки.

String toLowerCase (). Преобразует все символы в этой строке в нижний регистр, используя правила локали по умолчанию.

String toUpperCase (). Преобразует все символы в этой строке в верхний регистр, используя правила локали по умолчанию.
String trim(). Возвращает копию строки, при этом опущенные пробелы ведущего и конечного.
static String valueOf(primitive data type x). Возвращает строковое представление аргумента переданного типа данных. Этот метод можно использовать для преобразования из числового формата в объект String. Например:

```
double d1=123.456;
String st1=String.valueOf(d1);
System.out.println(st1);
```

Результатом этого кода будет 123.456.
Преобразование из строкового в числовой формат, иллюстрируемое кодом Java ниже:

```
String st1="100";
String st2="123.456";
int i=Integer.valueOf(st1);
double d=Double.valueOf(st2);
System.out.println("i="+i+" d="+d);
```

Результатом кода будет: i=100 d=123.456

2.2.5. Класс StringBuilder

Объекты StringBuilder похожи на объекты String, за исключением того, что они могут быть изменены. Внутренне эти объекты обрабатываются как массивы переменной длины, которые содержат последовательность символов. В любой момент длина и содержание последовательности могут быть изменены посредством вызова метода.
Строки всегда должны использоваться, если постройка строк не имеет преимущества в отношении более простого кода или более высокой производительности. Например, если вам нужно объединить большое количество строк, добавление к объекту StringBuilder более эффективно.
Класс StringBuilder, как и класс String, имеет метод length (), который возвращает длину последовательности символов в постройке.
В отличие от строк, каждая постройка строк также имеет емкость, количество выделенных пространств символов. Емкость, возвращаемая методом capacity (), всегда больше или равна длине (обычно больше) и автоматически расширяется по мере необходимости для дополнения дополнений к построителю строк.
Методы класса StringBuilder аналогичны методам класса String, но кроме того, класс StringBuilder имеет некоторые методы, связанные с длиной и емкостью, которые не имеет класса String:
Void setLength (int newLength) Устанавливает длину последовательности символов. Если newLength меньше длины (), последние символы в последовательности символов усекаются. Если newLength больше длины (), в конце последовательности символов добавляются нулевые символы.

Void protectCapacity (int minCapacity) Обеспечивает, чтобы емкость была как минимум равна указанному минимуму.
StringBuilder append (String s) Добавляет аргумент в этот построитель строк. Данные преобразуются в строку перед выполнением операции добавления.
StringBuilder delete (int start, int end) или StringBuilder deleteCharAt (int index) Первый метод удаляет подпоследовательность от начала до конца-1 (включительно) в последовательности символов StringBuilder. Второй метод удаляет символ, расположенный по индексу.
StringBuilder insert (int offset, String s) Вставляет второй аргумент в построитель строк. Первый целочисленный аргумент указывает индекс, перед которым данные должны быть вставлены. Данные преобразуются в строку до начала операции вставки.
Метод **setCharAt(index,char);** Этот метод позволяет заменить какой-либо символ в желаемом положении новым символом.
Например:

```
creates empty builder, capacity 16 by default
StringBuilder sb = new StringBuilder();//Create an empty object sb
System.out.println(sb.capacity());//Default capacity is 16
System.out.println(sb.length());//Length of empty string is zero
sb.append("Greetings");//Add word Greeting to the string
System.out.println(sb.capacity());//Print capasity
System.out.println(sb.length());//Print length
System.out.println(sb.toString());//print content
```

The output of the code:

```
16
0
16
9
Greetings
```

2.3. Задание для лабораторной работы: во время текущей работы вы должны создать 2 Java-программы в соответствии с описаниями:
1. Создайте программу, которая сначала попросит пользователя ввести первый номер, второе число и математическую операцию (+, -, /, *), а затем вычислить результат в соответствии с операцией, введенной пользователем:

```
Enter first number: 5
Enter second number: 6
Enter operation: *
Result: 30
```

2. Создайте программу, которая позволяет вводить строку, затем вычисляет:
A) Число цифр в строке
B) Количество заглавных букв
C) Количество пробелов в строке

Например:
Enter the string: Hello 1 world 2 My Name is 5 Vasya
Result: 3 digits 4 capital chars 8 spaces

2.4 Содержание отчета

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

Лабораторная работа №3

Java: объектно-ориентированное программирование

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

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

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.

3.3. Задание для лабораторной работы.

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 = b2 + 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:

$x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}$, where $D=b^2+4ac$.

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 суперклеты 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.

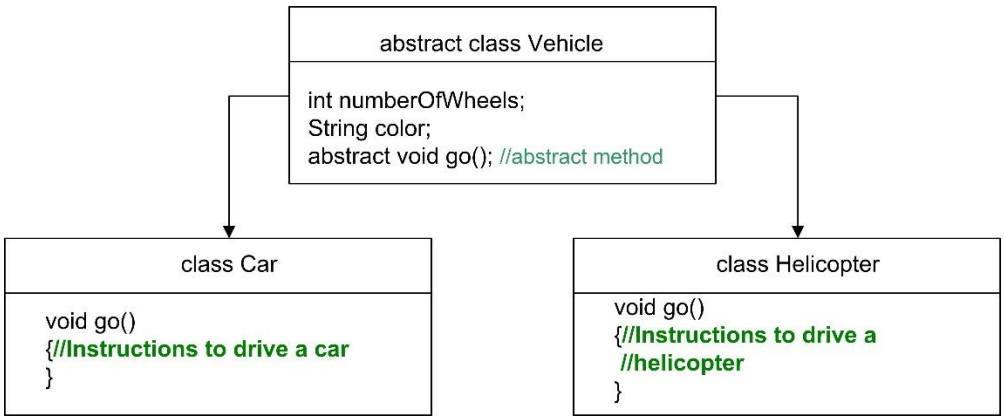

Рисунок 5.1. Иерархия классов

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:

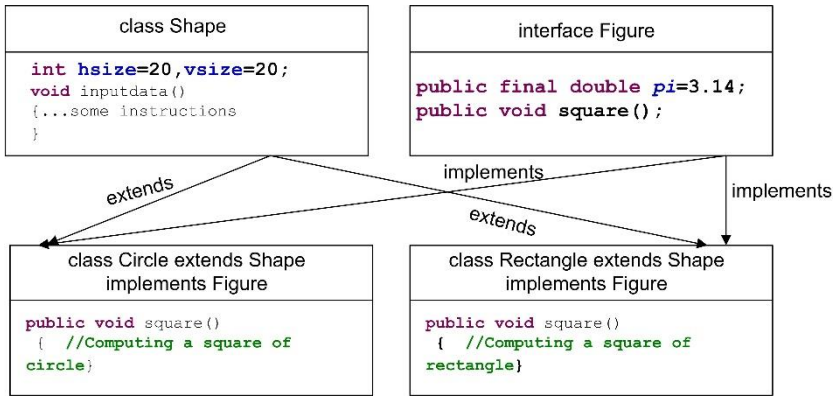

Рисунок 5.1. Иерархия классов

```
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

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Программирование REST-сервисов на языке Java [Электронный ресурс] : методиче-ские указания по выполнению лабораторной работы по курсу «Интернет-технологии» для сту-дентов, обучающихся по направлению подготовки 230400.62 «Информационные системы и тех-нологии» / Юго-Западный государственный университет, Кафедра информационных систем и технологий ; ЮЗГУ ; сост. М. В. Бородин. - Курск : ЮЗГУ, 2013. - 18 с.

2. Программирование Web-сервисов на языке Java [Электронный ресурс] : методические указания по выполнению лабораторной работы по курсу «Интернет-технологии» для студентов, обучающихся по направлению подготовки 230400.62 «Информационные системы и технологии» / ЮЗГУ ; сост.: Е. А. Титенко, М. В. Бородин. - Курск : ЮЗГУ, 2013. - 22 с.