Документ подписан простой электронной подписью
Информация о владельце:
ФИО: Локтионова Оксана Геннадьевна
Должность: проректор по учебной работе
Дата подписания: 08.09.2026 07:48:17
Уникальный программный ключ:
0b817ca911e6668abb13a5d426d39e5f1c11eabbf73e943df4a4851fda56d089

МИНОБРНАУКИ РОССИИ

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

Кафедра биомедицинской инженерии

УТВЕРЖДАЮ
Проректор по учебной работе
_____ Локтионова О.Г.
«25» 03 _____ 2021 г.

ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ МЕДИКО-БИОЛОГИЧЕСКИХ СИСТЕМ

Методические указания по проведению практической работы для студентов специальности 30.05.03 Медицинская кибернетика

Курск 2021

УДК 615.478

Составитель: Л.В. Стародубцева

Рецензент:
Доктор технических наук, профессор *И.Е. Чернецкая*

Технология программирования медико-биологических систем: методические указания по выполнению практической работы по специальности 30.05.03 Медицинская кибернетика / ЮЗГУ ; сост. Л.В. Стародубцева - Курск: ЮЗГУ, 2021. - 66 с.

Содержатся сведения, необходимые для выполнения практических занятий по дисциплине «Технология программирования медико-биологических 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 30.05.03 Медицинская кибернетика.

Текст печатается в авторской редакции

Подписано в печать Формат 60x84 1/16
Усл. печ.л. . Уч. -изд.л. Тираж 100 экз. Заказ. Бесплатно.
Юго-Западный государственный университет
305040, г.Курск, ул. 50 лет Октября, 94

ВВЕДЕНИЕ

Основной целью проведения практических занятий является формирование умений и навыков по программированию с использованием современных технологий, включая теорию алгоритмов и программирование для медико-биологических систем

Проведению практических занятий предшествует самостоятельная работа студентов, направленная на ознакомление с соответствующим теоретическим материалом. При необходимости, студенты по заданиям преподавателей выполняют подготовительную работу, обеспечивающую более эффективный процесс закрепления умений и навыков.

Практические занятия проводятся в специализированном классе, оснащенным персональными ЭВМ не ниже Intel i3, с операционной системой не ниже Windows 7. При проведении практического занятия рекомендуется сочетание интерактивного и практического обучения.

Практическая работа №1

Динамические структуры данных

2.1. Цель работы: Введение в использование динамических структур в языке C++.

2.2. Краткие теоретические сведения

Указатель - это переменная, содержащая адрес переменной. Указатели широко применяются в Си - отчасти потому, что в некоторых случаях без них просто не обойтись, а отчасти потому, что программы с ними обычно короче и эффективнее. Указатели и массивы тесно связаны друг с другом: в данной главе мы рассмотрим эту зависимость и покажем, как ею пользоваться. Наряду с goto указатели когда-то были объявлены лучшим средством для написания малопонятных программ. Так оно и есть, если ими пользоваться бездумно. Ведь очень легко получить указатель, указывающий на что-нибудь совсем нежелательное. При соблюдении же определенной дисциплины с помощью указателей можно достичь ясности и простоты. Мы попытаемся убедить вас в этом.

Изменения, внесенные стандартом ANSI, связаны в основном с формулированием точных правил, как работать с указателями. Стандарт узаконил накопленный положительный опыт программистов и удачные нововведения разработчиков компиляторов. Кроме того, взамен char* в качестве типа обобщенного указателя предлагается тип void* (указатель на void).

Начнем с того, что рассмотрим упрощенную схему организации памяти. Память типичной машины подставляет собой массив последовательно пронумерованных или проадресованных ячеек, с которыми можно работать по отдельности или связными кусками. Применительно к любой машине верны следующие утверждения: один байт может хранить значение типа char, двухбайтовые ячейки могут рассматриваться как целое типа short, а четырехбайтовые - как целые типа long. Указатель - это группа ячеек (как правило, две или четыре), в которых может храниться адрес

Унарный оператор * есть оператор косвенного доступа. Примененный к указателю он выдает объект, на который данный указатель указывает. Предположим, что x и y имеют тип int, а ip – укаэатель на int. Следующие несколько строк придуманы специально для того, чтобы показать, каким образом объявляются указатели и как используются операторы & и *.

```
int x = 1, y = 2, z[10];
int *ip;    /* ip - указатель на int */
ip = &x;    /* теперь ip указывает на x */
y = *ip;    /* y теперь равен 1 */
*ip = 0;    /* x теперь равен 0 */
ip = &z[0]; /* ip теперь указывает на z[0] */
```

Объявления x, y и z нам уже знакомы. Объявление указателя ip

int *ip;

мы стремились сделать мнемоничным - оно гласит: "выражение *ip имеет тип int". Синтаксис объявления переменной "подстраивается" под синтаксис выражений, в которых эта переменная может встретиться. Указанный принцип применим и в объявлениях функций. Например, запись

double *dp, atof (char *);

означает, что выражения *dp и atof(s) имеют тип double, а аргумент функции atof есть указатель на char.

Вы, наверное, заметили, что указателю разрешено указывать только на объекты определенного типа. (Существует одно исключение: "указатель на void" может указывать на объекты любого типа, но к такому указателю нельзя применять оператор косвенного доступа.)

Если ip указывает на x целочисленного типа, то *ip можно использовать в любом месте, где допустимо применение x; например,

*ip = *ip + 10;

увеличивает *ip на 10.

Унарные операторы * и & имеют более высокий приоритет, чем арифметические операторы, так что присваивание

y = *ip + 1;

берет то, на что указывает ip, и добавляет к нему 1, а результат присваивает переменной y. Аналогично

*ip += 1;

увеличивает на единицу то, на что указывает ip; те же действия выполняют

++*ip;

и

(*ip)++;

В последней записи скобки необходимы, поскольку если их не будет, увеличится значение самого указателя, а не то, на что он указывает. Это обусловлено тем, что унарные операторы * и ++ имеют одинаковый приоритет и порядок выполнения - справа налево.

И наконец, так как указатели сами являются переменными, в тексте они могут встречаться и без оператора косвенного доступа. Например, если iq есть другой указатель на int, то

iq = ip;

копирует содержимое ip в iq, чтобы ip и iq указывали на один и тот же объект.

В Си существует связь между указателями и массивами, и связь эта настолько тесная, что эти средства лучше рассматривать вместе. Любой доступ к элементу массива, осуществляемый операцией индексирования, может быть выполнен с помощью указателя. Вариант с указателями в

общем случае работает быстрее, но разобраться в нем, особенно непосвященному, довольно трудно.

Объявление

int a[10];

Определяет массив a размера 10, т. е. блок из 10 последовательных объектов с именами a[0], a[1], ..., a[9].

Запись a[i] отсылает нас к i-му элементу массива. Если pa есть указатель на int, т. е. объявлен как

int *pa;

то в результате присваивания

pa = &a[0];

pa будет указывать на нулевой элемент a, иначе говоря, pa будет содержать адрес элемента a[0].

Теперь присваивание

x = *pa;

будет копировать содержимое a[0] в x.

Если pa указывает на некоторый элемент массива, то pa+1 по определению указывает на следующий элемент, pa+i - на i-й элемент после pa, а pa-i - на i-й элемент перед pa. Таким образом, если pa указывает на a[0], то

*(pa+1)

есть содержимое a[1], a+i - адрес a[i], а *(pa+i) - содержимое a[i].

Сделанные замечания верны безотносительно к типу и размеру элементов массива a. Смысл слов "добавить 1 к указателю", как и смысл любой арифметики с указателями, состоит в том, чтобы pa+1 указывал на следующий объект, а pa+i - на i-й после pa.

Между индексированием и арифметикой с указателями существует очень тесная связь. По определению значение переменной или выражения типа массив есть адрес нулевого элемента массива. После присваивания

pa = &a[0];

pa и a имеют одно и то же значение. Поскольку имя массива является синонимом расположения его начального элемента, присваивание pa=&a[0] можно также записать в следующем виде:

pa = a;

Еще более удивительно (по крайней мере на первый взгляд) то, что a[i] можно записать как *(a+i). Вычисляя a[i], Си сразу преобразует его в *(a+i); указанные две формы записи эквивалентны. Из этого следует, что полученные в результате применения оператора & записи &a[i] и a+i также будут эквивалентными, т. е. и в том и в другом случае это адрес i-го элемента после a. С другой стороны, если pa - указатель, то его можно использовать с индексом, т. е. запись pa[i] эквивалентна записи *(pa+i). Короче говоря, элемент массива можно изображать как в виде указателя со смещением, так и в виде имени массива с индексом.

Между именем массива и указателем, выступающим в роли имени массива, существует одно различие. Указатель - это переменная, поэтому можно написать pa=a или pa++. Но имя массива не является переменной, и записи вроде a=pa или a++ не допускаются.

Если имя массива передается функции, то последняя получает в качестве аргумента адрес его начального элемента. Внутри вызываемой функции этот аргумент является локальной переменной, содержащей адрес. Мы можем воспользоваться отмеченным фактом и написать еще одну версию функции strlen, вычисляющей длину строки.

```
/* strlen: возвращает длину строки */
int strlen(char *s)
{
   int n;
   for (n = 0; *s != '\0'; s++)
      n++;
   return n;
}
```

Так как переменная s - указатель, к ней применима операция ++; s++ не оказывает никакого влияния на строку символов функции, которая обратилась к strlen. Просто увеличивается на 1 некоторая копия указателя, находящаяся в личном пользовании функции strlen. Это значит, что все вызовы, такие как:

```
strlen("Здравствуй, мир"); /* строковая константа */
strlen(array);             /* char array[100]; */
strlen(ptr);               /* char *ptr; */
```

правомерны.

2.3. Задание на практическую работу

Составьте текст программы, которая запрашивает информацию об оценках студентов вашей группы и заносит их в динамический массив, после чего производит поиск максимального, минимального и среднего

значения, а также ранжирует оценки по признакам: отлично, хорошо, удовлетворительно и неудовлетворительно

2.4. Содержание отчета

1. Титульный лист
2. Задание на практическую работу
3. Листинг программы с динамическим массивом
4. Скриншот экрана

Практическая работа №2

Графический дизайн пользовательского интерфейса и адаптеры класса

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

2.2 Краткие теоретические сведения

Во время последней практической работы 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.getX();
  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
}
```

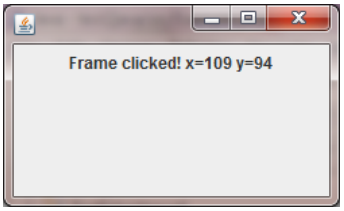


Рисунок 2.1 – GUI примера (1), описанного выше

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)

MouseReleased - напротив mousePressed event. Это событие генерируется, когда пользователь отпускает кнопку мыши. Например, одно событие mouseClicked, предсказанное событиями mousePressed и mouseReleased.

Для обработки события «mouseClicked» мы использовали интерфейс MouseListener, который содержит 5 событий. Даже если вам нужно только одно событие (метод «mouseClicked»), 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 mouseClicked из MouseListener или обработчик mouseDragged 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 2.1 показаны несколько классов адаптера java.awt.event и интерфейсы, которые они реализуют.

Таблица 2.1. Интерфейсы и класс-адаптеры

N	**Название интерфейса**	**Название адаптера**	**Описание событий**
1	ComponentListener	Component Adapter	События компонента. Способ регистрации: addcomponentListener () или добавить компонентный адаптер. Интерфейс и адаптер класса содержат набор методов:componentHidden();componentMoved(); componentResized();componentShown();
2	ContainerListener	ContainerAdapter	События, связанные с добавлением и удалением компонентов в / из класса контейнера. Способ регистрации:addContainerListener() или add ContainerAdapter. Contains 2 methods: componentAdded() и componentRemoved()
3	FocusListener	FocusAdapter	События, связанные с фокусом (когда компонент контролирует интерфейс пользователя). Способ регистрации:addFocusListener() или addFocusAdapter()
4	KeyListener	KeyAdapter	Ответственный за нажатие и отпускание клавиш. Способ

			регистрации:addKeyListener или addKeyAdapter. Contains 3 methods: keyPressed(); keyTyped(); keyReleased().
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 интерфейс содержат 2 метода:mouseDragged() и mouseMoved()
7	WindowListener	WindowAdapter	События из окна. Метод регистрации addWindowListener или добавьте WindowAdapter. Содержит 7 событий:windowActivated(); windowClosed(); windowClosing(); windowDeactivated(); windowDeiconofied(); windowIconofied(); windowOpened().

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
}
```

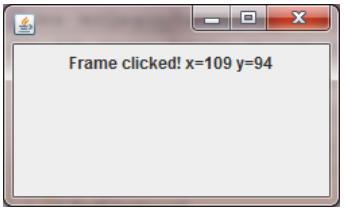


Рисунок 2.2 – GUI примера (2), описанного выше

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 FlowLayout());//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
}
```

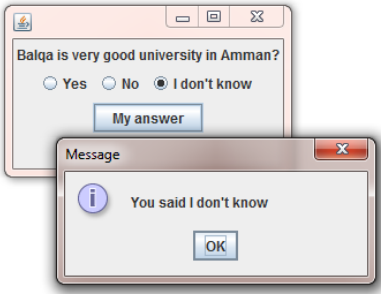
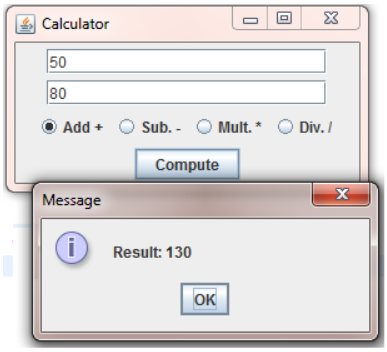


Рисунок 2.3 – GUI примера (3)

2.3. Задание для практической 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 evemt «щелчок» кнопки.

2.4 Содержание отчета

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

Практическая работа №3

Изучение массивов

3.1 Цель работы: изучить и получить практические навыки проектирования программ с использованием массивов данных

3.2. Краткие теоретические сведения

Java предоставляет структуру данных, массив, в котором хранится последовательный набор фиксированного размера элементов того же типа. Массив используется для хранения коллекции данных, но часто бывает полезно придумать массив как набор переменных того же типа.

Вместо объявления отдельных переменных, таких как number0, number1, ... и number99, вы объявляете одну переменную массива, такую как числа, и используете числа [0], числа [1] и ..., цифры [99] для представления отдельных переменных.

В этом руководстве описывается, как объявлять переменные массива, создавать массивы и массивы процессов с помощью индексированных переменных.

Объявление переменных массива:

Чтобы использовать массив в программе, вы должны объявить переменную для ссылки на массив, и должны указать тип массива, который может ссылаться на переменную. Вот синтаксис объявления переменной массива:

```
dataType[] arrayRefVar;   // preferred way.
```

or

```
dataType arrayRefVar[];  //  works but not preferred way.
```

Примечание. Стиль **dataType [] arrayRefVar** является предпочтительным.

Стиль **dataType arrayRefVar []** исходит из языка C / C ++ и был принят на Java для размещения программистов на C / C ++.

Создание массивов:

Вы можете создать массив, используя «новый» оператор со следующим синтаксисом:

```
ArrayRefVar = new dataType [arraySize];
```

Вышеприведенное утверждение делает две вещи:

Он создает массив с использованием нового dataType [arraySize];

Он присваивает ссылку вновь созданному массиву переменной arrayRefVar.

Объявление переменной массива, создание массива и назначение ссылки массива переменной могут быть объединены в один оператор, как показано ниже:

```
DataType [] arrayRefVar = new dataType [arraySize];
```

В качестве альтернативы вы можете создавать массивы следующим образом:

```
DataType [] arrayRefVar = {value0, value1, ..., valuek};
```

Элементы массива получают доступ через индекс. Индексы массивов основаны на 0; То есть они начинаются от 0 до **arrayRefVar.length-1.**

Примеры правильного и неправильного объявления и создания массивов:

```
int[] k;//Declaring  array k of int data type
k[0]=5;//Mistake,array declared but not created
k=new int[5];//Creation of array, k, length equal 5 elements
k[0]=2;//No mistake, array was created, cell 0 assigned
int[] k1=new int[5];//No mistake, array k1 declared and created
                //using one line of Java code
k1[0]=4;//No mistake, array created, you can
        //assign a value to the cell 0
```

Обработка массивов:

При обработке элементов массива мы можем использовать любой цикл для доступа ко всем элементам массива, которые имеют одинаковый тип данных. Язык Java-компьютера имеет 4 разных цикла, которые можно использовать для доступа к элементам массива: для while и для каждого цикла:

```
for (int i=0;i<10;i++) {/*body of the loop*/}
while (/*condition*/) {/*body of the loop*/}
do {/*body of the loop*/} while(/*condition*/);
for(int i:name_of_array){/*body of the loop*/}
```

В следующем примере сначала мы создадим массив целых чисел, тогда мы будем использовать разные циклы для доступа и распечатки всех элементов заданного массива:

```
public class ArrayTest
{
public static void main(String[] args) {
int[] k1={5,4,3,2,1,6,8,9};
//k1 is array of 8 integer numbers
//Example of "for" loop:
for (int i=0;i<k1.length;i++) System.out.print(k1[i]);
System.out.println();//go to next line
//Example of "while" loop:
int j=0;//create integer variable j, set value of j to zero
while(j<k1.length)
{
        System.out.print(k1[j]);
        j++;

}
System.out.println();//go to next line
//Example of "do while" loop:
```

```
int a=0;
do
{
        System.out.print(k1[a]);
        a++;
}
while(a<k1.length);
System.out.println();//go to next line
//Example of "for each" loop:
for(int b:k1) System.out.print(b);
System.out.println();//go to next line
        }//End of method “main”
}//End of the class ArrayTest
```

Результатом этой программы будет:

54321689
54321689
54321689
54321689

Вы можете видеть, что каждый цикл предоставляет один и тот же вывод, и в будущем вы можете использовать любой цикл по своему вкусу для доступа к элементам массива, но обычно наиболее популярными циклами в Java являются цикл «for» и «for each» цикла. В приведенной выше программе имя массива k1, вы можете использовать переменную «length », если хотите знать длину массива: k1.length - это целое число, которое возвращает фактическую длину массива.

3.3. Задание на практическую работу

Составьте текст программы, которая запрашивает информацию об оценках студентов вашей группы, после чего производит поиск максимального, минимального и среднего значения, а также ранжирует оценки по признакам: отлично, хорошо, удовлетворительно и неудовлетворительно

3.4. Содержание отчета

1. Титульный лист
2. Задание на практическую работу
3. Листинг прораммы
4. Скриншот экрана

Практическая работа №4
Java: объектно-ориентированное программирование

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

4.2. Краткие теоретические сведения

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.

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.

4.3. Задание для практической работы.

Создать класс, который позволяет хранить оценки студентов вашей группы. В классе должна быть структура для хранения, а так же набор методов, обеспечивающих следующие функции:

1. Ввод данных
2. Сортировка
3. Печать данных
4. Поиск минимума, максимума и среднего

4.4 Содержание отчета

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

Практическая работа №5

Типы данных, а также обзор классов и методов, реализующих работу со строками

5.1. Цель работы: практическое освоение разделов программирования, связанных с созданием переменных, изучение типов данных, массивов, а также освоение стандартных классов, реализующих работу со строками.

5.2 Краткая теория:

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 5.1.

Таблица 5.1 – 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.

5.2.2 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
```

5.2.3 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».

5.2.4. 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

5.2.5. 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
```

5.3. 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:

А) 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

5.4 Содержание отчета

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

Практическая работа №6
Технология разработки кросс-платформенных приложений

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

6.2. 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.)

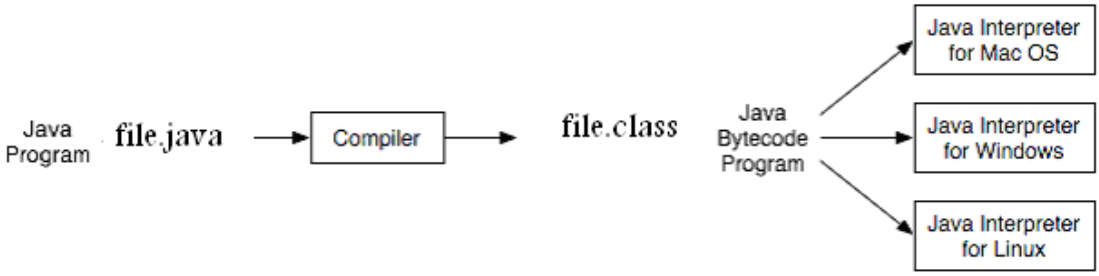


Рисунок 6.1 – Создание и исполнение байт-кода

Рисунок 6.1 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.

6.3. Порядок выполнения работы

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»:

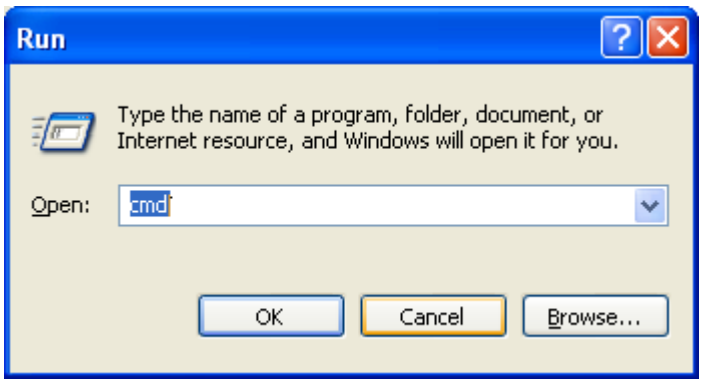


Рисунок 6.2 – Выполнение команды «cmd» в Windows XP

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.

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

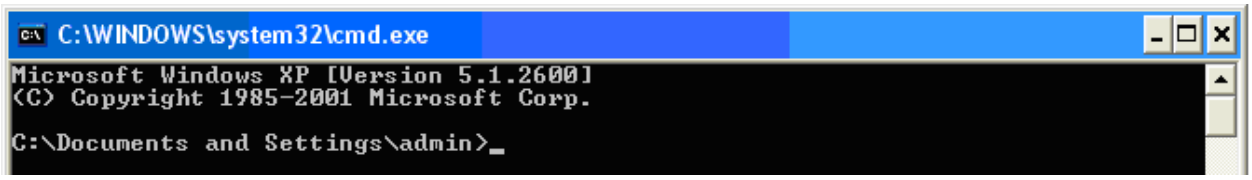


Рисунок 6.3 – Окно консоли операционной системы

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 6.4.

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\admin>path c:\Program Files\Java\jdk1.6.0_16\bin

C:\Documents and Settings\admin>javac
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
  -g:none                    Generate no debugging info
  -g:{lines,vars,source}     Generate only some debugging info
```

Рисунок 6.4 – Добавление пути Java-компилятора к переменной окружения системы Windows.

Чтобы создать программу Java, мы должны сначала создать папку для хранения исходного кода программы, это также возможно с помощью консольных команд, см. Рисунок 6.5.

```
C:\WINDOWS\system32\cmd.exe
D:\>mkdir javaproject

D:\>cd javaproject
```

Рисунок 6.5 – 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.

```
Hello.java - Notepad
File  Edit  Format  View  Help
public class Hello
{
static public void main(String[] args)
 {System.out.println("Hello Java");
 }
}
```

Рисунок 6.6 – 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 6.7.

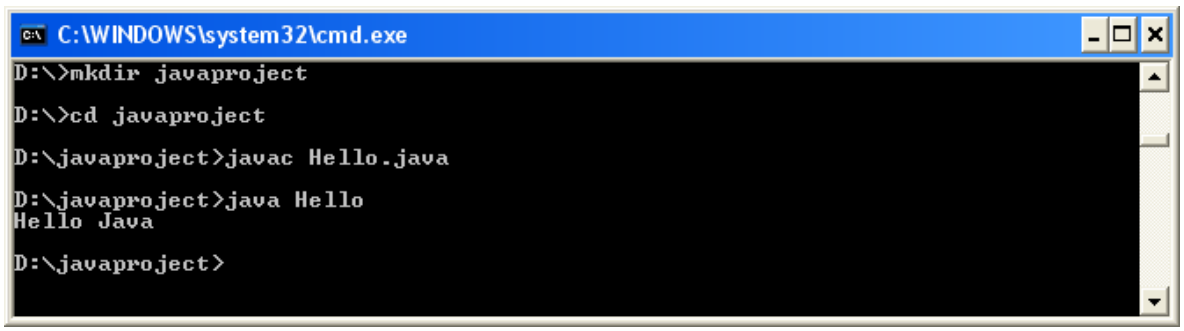


Рисунок 6.7 – Консольные команды для компиляции и запуска программы Java

На выходе программы, представленной на рисунке 6.7, вы можете увидеть строку текста, созданную командой System.out.println («Hello world»);

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.):

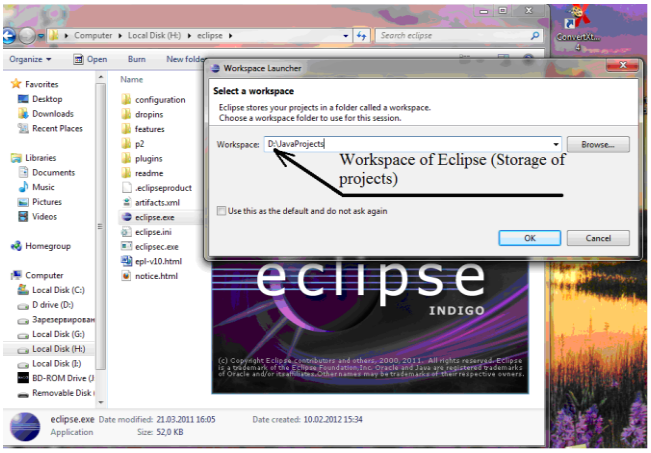


Рисунок 6.8 – Eclipse Environment, запрос рабочего пространства.

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 6.9:

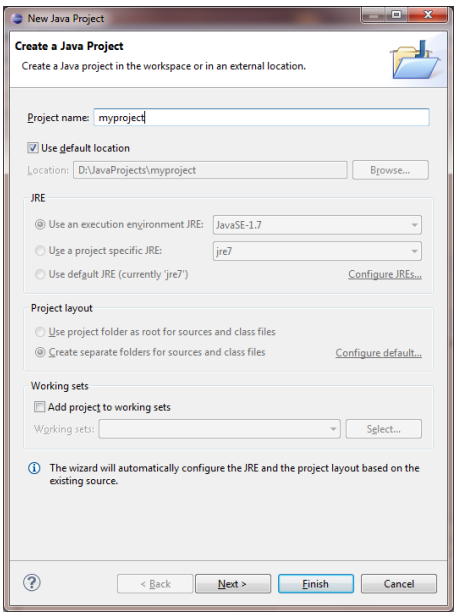


Рисунок 6.9 –Создание нового проекта в Eclipse Environment

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 6.10.

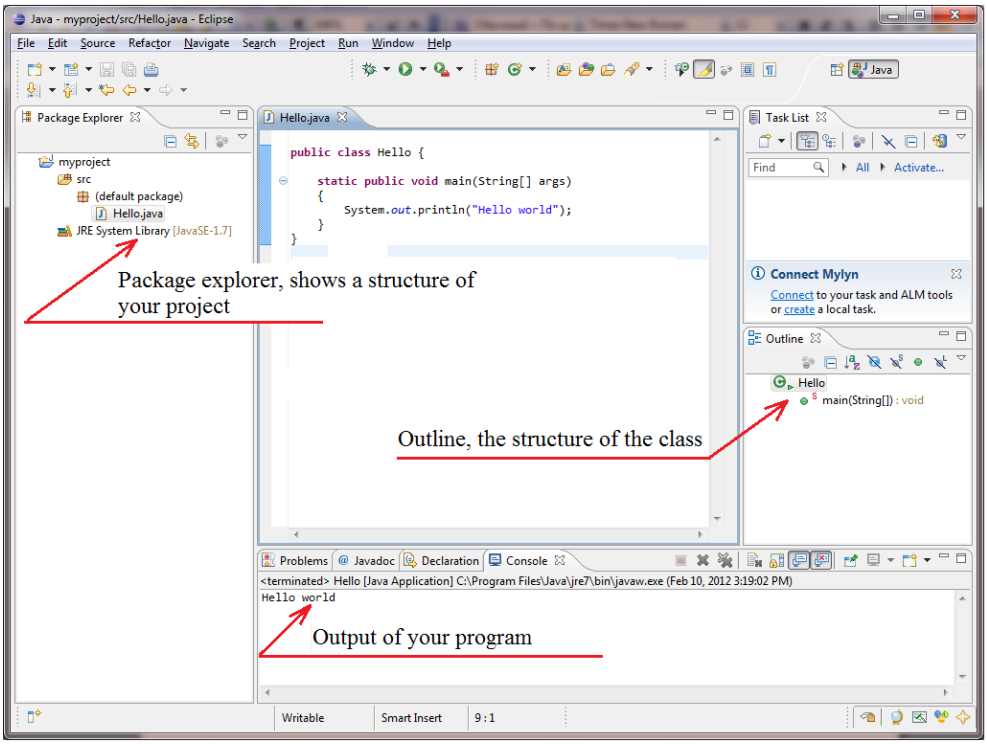


Рисунок 6.10 – Основные элементы Eclipse IDE.

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. 6.11.).

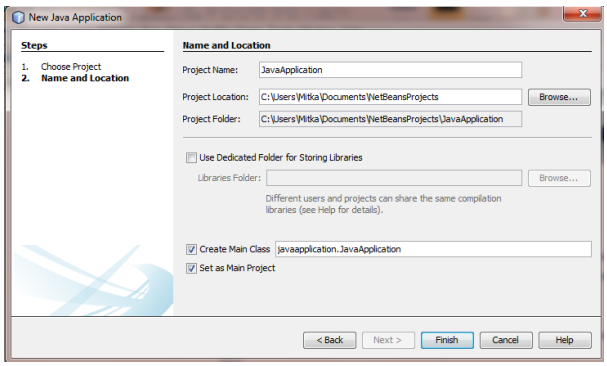


Рисунок 6.11– Создание нового приложения Java с использованием Netbeans.

Согласно рисунку 6.11. По умолчанию именем приложения Netbeans является JavaApplication, это имя класса, которое будет создано после нажатия кнопки «Готово». На рисунке 6.12 показаны основные компоненты среды IDE NetBeans.

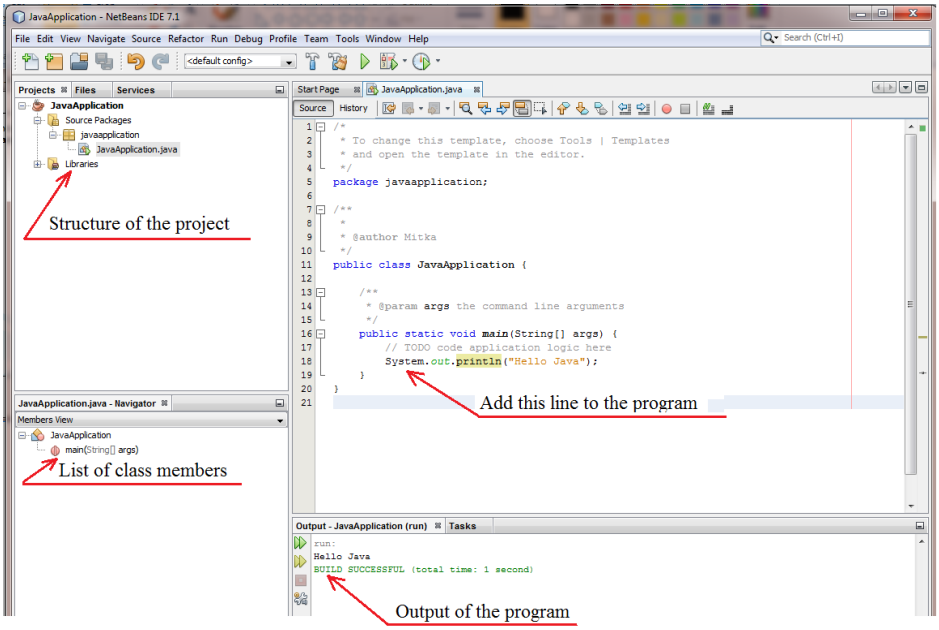


Рисунок 6.12 – Основные компоненты IDE NetBeans

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 6.12).

6.4. Содержание отчета

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

Практическая работа №7
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
16 of Class1:");
17    System.out.println("a="+a+" b="+b);
18    f1();//We call method f1 of Class1 directly
19    int a=5;//from this moment we have new local variable a
20    //now we can access a of Class1 using keyword super:
21    System.out.println("local a="+a+" but a of
22 Class1="+super.a);
23   }
24
25  }
26
27  public class Example{
28      //This is public class with main method
29      //The code on this page located in the file
30 Example.java
31    static public void main(String[] args)
32    {Class2 obj=new Class2();
33     obj.b=100;//b is protected but can be accessed directly
34 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.

7.3. 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:

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

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

7.4 Содержание отчета

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

Практическая работа №8
Абстрактные классы и интерфейсы

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

8.2. 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 8.1.

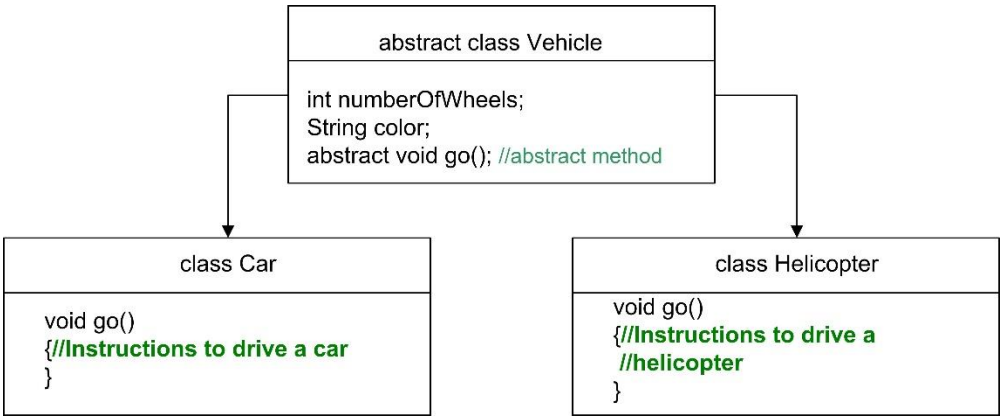


Рисунок 8.1 – Иерархия классов

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 OOP 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:

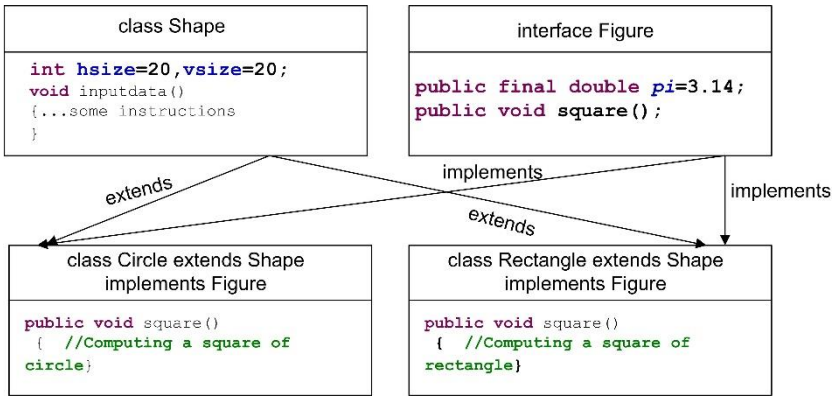


Рисунок 8.2 – Иерархия классов

```
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.

8.3. 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.

8.4 Содержание отчета

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

Практическая работа №9
Введение в разработку графического интерфейса пользователя

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

9.2 Краткие теоретические сведения

Графический пользовательский интерфейс (GUI) представляет собой удобный для пользователя механизм взаимодействия с приложением. Графический интерфейс (произносится как «GOO-ee») 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 siwng 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

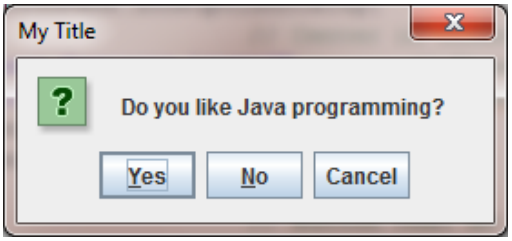
Таблица 9.1 – Связь между последним параметром showMessageDialog и видом значка

Parameter	Icon
JOptionPane.ERROR_MESSAGE	
JOptionPane.WARNING_MESSAGE	
JOptionPane.INFORMATION_MESSAGE	
JOptionPane.QUESTION_MESSAGE	
JOptionPane. PLAIN_MESSAGE	none

Метод JOptionPane.showConfirmDialog может использоваться для запроса некоторого вопроса от пользователя. В приведенном ниже коде показан простой пример:

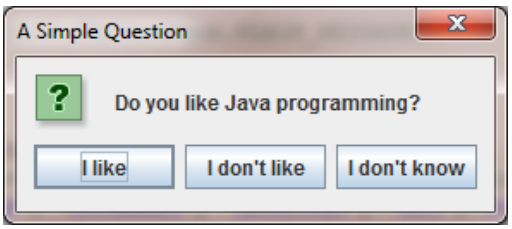
```
int n = JOptionPane.showConfirmDialog(null,
"Do you like Java programming?","My Title",
```

```
JOptionPane.YES_NO_CANCEL_OPTION);
```



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  =  {"Mohammad",  "Abdallah",  "Ahmad",
"Hakeem"};
  //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
```

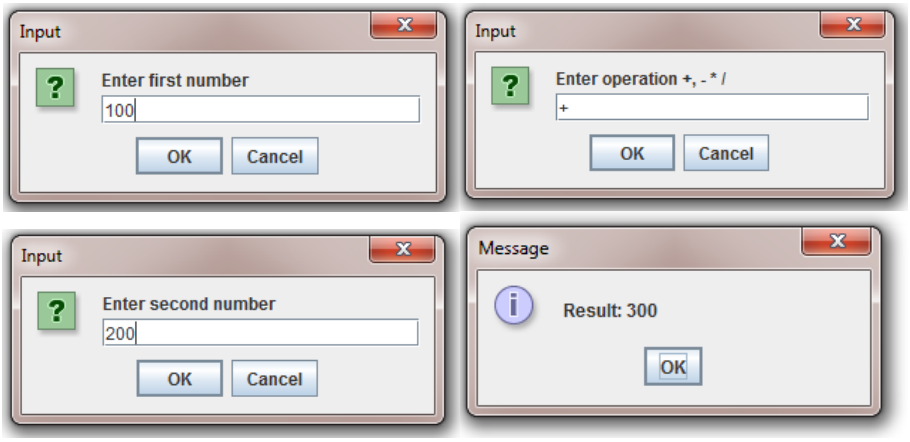
Результат этой программы представлен ниже:

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.

9.3 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:



Подсказка: преобразование из String в int

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

Преобразование из int в String

```
int i=123;
String st1=String.valueOf(i);
```

5.4 Содержание отчета

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

Практическая работа №10
Компоненты Java и модель делегирования событий

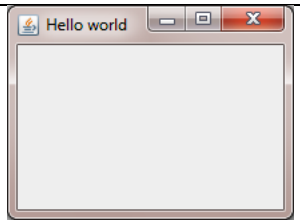
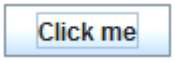
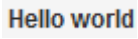
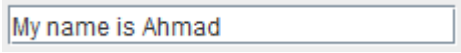
10.1 Цель работы: Создание приложений форм с использованием компонентов JAVA. Обработка событий и событий с использованием интерфейсов Java.

10.2 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 10.1 показаны несколько базовых компонентов GUI Swing, которые мы обсуждаем сегодня.

Таблица 10.1 – Список компонентов графического интерфейса пользователя

Название компонента	**Описание**	**Внешний вид**
JFrame	Это класс-контейнер, база программы, окно, содержащее все остальные компоненты программы. В .NET (Microdoft 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 имеет имя "TextBox"	

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, JTextFiel:

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
}
```

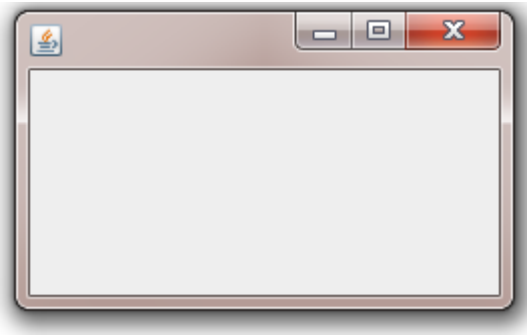


Рисунок 10.1 –Форма (окно), созданная кодом

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 командLocation (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 10.2 – 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 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
}
```

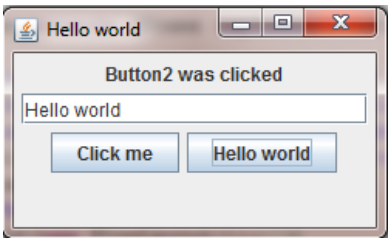


Рисунок 10.3 – Внешний вид программы и выполнение обработчиков событий

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.

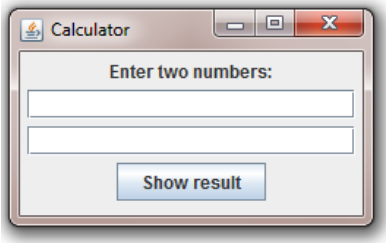
10.3 Задание для лабораторной работы.

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 Eclips.

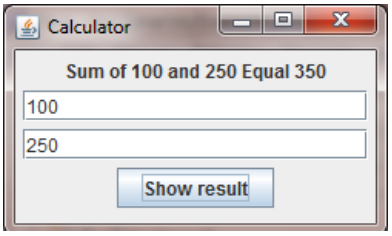
```
import javax.swing.*;
import java.awt.event.*;

class MyFrame extends JFrame implements MouseMotionListener
{
 public MyFrame()
 {   super("Virtual frog");//5
    this.addMouseMotionListener(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:



10.4 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] : методические указания по выполнению лабораторной работы по курсу «Интернет-технологии» для студентов, обучающихся по направлению подготовки 230400.62 «Информационные системы и технологии» / 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.