Программирование на языке Java в среде Eclipse

М.В. Лапенок

Уральский государственный педагогический университет,
г. Екатеринбург

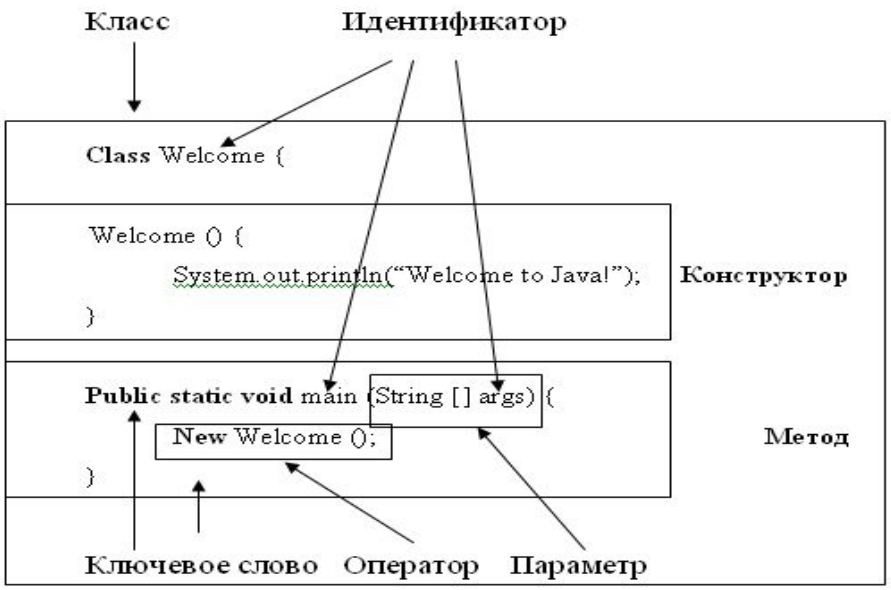
Байт-код

Язык программирования Java имеет одно существенное отличие от других распространенных языков программирования, например, Object Pascal или C++.

Компилятор Java принимает последовательность команд Java, которые образуют исходный текст программы, и в результате создает не исполняемый машинный код, а промежуточный, так называемый «байт-код».

Байт-код принимается Java-машиной, которая входит в состав всех браузеров, и затем преобразуется в исполняемый машинный код.

Стандартная структура программы



Пример программы

```
class Welcome1 {
Welcome1 () {
        System.out.println("Welcome1 to Java!");
System.out.println("5 февраля День рождения Андрюльчика!!!!!!!");
}
public static void main (String [] args){
new Welcome1 ();
}
}
```

Пример программы

```
class Welcome2 {
Welcome2 () {int sum;
        int i;

int    N=100;
        for (i=1, sum=0; i<=N; i++)
                sum+=i;
                        System.out.println(sum);

        System.out.println("5 февраля День рождения Андрюльчика!!!!!!!");

}
public static void main (String [] args){
new Welcome2 ();

}
}
```

- Java – программы состоят из одного или нескольких взаимозависимых классов.

- После запуска на исполнение программы
  Java – машина ищет метод с именем *main* и характеристиками *public static void*
  и затем запускает программу с этого места.

- Метод *main* создает объект класса программы и вызывает его конструктор.

- Конструкторы выполняют три основные функции
  - Если класс содержит метод MAIN, он выполняет работу программы.
  - Если в программе используется графика, конструктор отвечает за создание фрейма, определение его размера и т.д.
  - Если класс является основой для создания нескольких объектов, конструктор применяется для копирования начальных значений в объектные переменные.

Переменные могут быть объявлены в любом месте класса или метода в программе на языке Java, а также внутри любой группы операторов, заключенной в фигурные скобки.

Такую группу операторов называют блоком. Обычно объявление располагают в начале или в конце блока.

Идентификаторы не должны дублироваться внутри блока.

Схема объявления переменных:

- < тип > имя_перем_1;
- < тип > имя_перем_ 1, имя_перем_ 2, …, имя_перем_ n;
- < тип > перем.1 = значение 1;

Типы данных

В языке Java насчитывается 6 типов данных для определения чисел: **int, long, double, byte, short, float**

и два специальных типа **boolean** и **char**.

Обычно для объявления целых чисел используется тип **int**, хотя часто применяется и тип **long**, поскольку он по – умолчанию используется для целочисленных выражений. Для вещественных по умолчанию применяется **double**.

byte	целое со знаком	8 бит	max 127
short	целое со знаком	16 бит	32767
int	целое со знаком	32 бит	214748647
long	целое со знаком	64 бит	$\approx 10^{18}$
float	с плав. точкой	32 бит	$\approx 10^{38}$
double	целое со знаком	64 бит	$\approx 10^{308}$

Оператор присваивания

имя _ переменной = выражение;
(для встроенных и объектных переменных)

Краткая форма записи оператора присваивания:

имя _ переменной + +;
имя _ переменной - -; (только для чисел)
Выражение имя _ перем. + + можно заменить выражением
имя _ перем. + = 1
(т.е. к переменной прибавить 1)
Можно записать имя _ переменой + = 6, т.е. к переменой прибавить 6.
Эта запись называется краткой формой. Существует краткая форма также для вычитания и для умножения
имя _ перем. - = 1
имя _ перем. * = 5

Разрядность числовых констант:

- целые числа могут иметь до 10 разрядов,
- вещественные до 17 разрядов.

Если размер вещественного числа превышает 17 значащих разрядов, то Java выводит число в E – формате.

Приоритеты арифметических операций

1. ( )
2. + + - - унарный + унарный -
3. * / %
4. + - конкатенация +

Класс Math

Выполнение стандартных арифметических и тригонометрических функции, которые поддерживает любой хороший калькулятор, в языке Java обеспечивается с помощью специального класса **Math.** Он принадлежит к пакету **java.Lang**, который импортируется в каждую программу

```
import java.Lang
```

Наиболее часто употребляемые функции :

1) final double PI /*константа 10 разрядов*/
2) double pow (double, double) //первый параметр //возводится в степень с показателем вторым параметром
3) double sqrt (double)
4) double atan (double)
5) double sin (double) // в радианах
6) double cos (double) // в радианах
7) double tan (double) // в радианах
8) double toDegrees (double) // взаимные преобразования
9) double toRadians (double) // значений из радиан в градусы
10) double random ( );
11) int round (float)
12) long round (double)
13) value abs (value) //значение м.б. int, long, float, double
14) value max (value, value) //значение м.б. int, long, float, double
15) value min (value, value) //значение м.б. int, long, float, double

Все методы этого встроенного класса языка Java имеют свой тип, отличный от void, т.е. являются **типизированными**.

Примеры использования класса Math

1) Программа генерирует 2 случайных числа и выводит большее из них с помощью метода max.

```
class  Primer_progr1   {
    Primer_progr1  ( ) {
    double random1, random2;
    random1 = Math. random ( );
    random2 = Math. random ( );
    System. out. println (“Числа : ” + random1 + “u” + random2);
    System. out. println (“Максим. : + ” Math. Max (random1, random2));
                                        }
    public static void main (String [ ] args) {
       new Primer_progr1 ( );
    }
    }
```

2) Для возведения в квадрат м. использовать функцию Math.pow (x, 2)

3) Тригонометрические функции вызываются Math. sin (x)

4) Округление до ближайшего целого Math.round (6,6) даст значение 7

Методы класса

Когда создан объект какого – либо класса, к нему можно применить методы, определенные внутри этого класса.

Доступ к этим методам осуществляется с помощью команды

Имя_объекта **.** имя_метода ( )

Исключением является только метод main, так как он объявлен как static, поэтому он является единственным и принадлежит самому классу. Когда виртуальная машина Java ищет метод main для запуска, она использует идентификатор.

Имя_класса**.** main ( )

Класс Math также является абстрактным.

Все его методы объявлены как static, так что для их использовании объект не создается, доступ к нему осуществляется **прямо через имя класса**.

Простые условия

Условия (выражения логического типа) могут принимать значения true и false. При формировании условий м.б. использованы в операции сравнения

= =	равно
<	меньше
!=	не равно
> =	больше или равно
>	больше
< =	меньше или равно

Результат такого логического выражения может храниться в булевой переменой

Напр.

```
boolean podrostok, pensioner;
int vozrast;
podrostok = vozrast < 18;
pensioner = vozrast > 60;
```

Для построения сложных условий используют булевы операторы

- & – и
- | – или
- ^ – исключающее или
- ! – не

В Java имеются так называемые операторы короткого замыкания для «и» && , а также для «или» ||.

В выражении

c && d

значение **d** учитывается, если значение **c** является истинным.

Так же и для логического выражения c || d

если **c** имеет истинное значение, то **d** игнорируется.

Примером употребления таких операторов служит задача выяснения, какая дата (состоящая из трех целых чисел) является более ранней.

Оператор решающий эту задачу, имеет вид:

```
boolean    rannij =    god1 < god2       ||

(god1 = = god2 && mes1 < mes2)            ||

(god1 = = god2 && mes1 = = mes2 && den1 < den2)
```

Приоритет булевых операторов

1. !
2. &
3. |
4. &&
5. ||

- Булевы операторы и операторы сравнения можно использовать только с переменными простых типов.
- Для сравнения объектов программист сам должен определить соответствующие методы.

Например, в программе ANKETA нужно сравнить зарплату работников.

Добавим метод, возвращающий значение типа boolean:

```
boolean vishe_oplata (Employee c) {
  return
       salary > c. salary;}
```

Чтобы сравнить, например, зарплату менеджера и программиста, нужно применить данный метод в форме:

```
boolean  g = Manager. vishe_oplata (Programmer);
```

Вывод данных

В Java не существует оператора для вывода данных. Вывод осуществляется с помощью специальных методов классов. Чтобы вывести какую – либо переменную, нужно знать какой метод следует вызвать.

В специальном встроенном классе ***PrintStream*** находятся методы *println* и *print.*

Внутри общедоступного класса System уже существует объект ***PrintStream*** с именем out, который связан с экраном компьютера.

Итак есть 3 элемента:

1) Класс System
2) объект **out** класса ***PrintStream***
3) методы *println* и *print*

Из них можно скомпоновать операторы, которые вызывают метод для вывода информации на экран. Для краткости их часто называют операторами вывода:

```
String   s1, s2, s3, s, sn;
System. out. println (s);
System. out. println (s1 + s2 + …+ sn);
System. out. println ( );
```

Java выполняет только вывод строк.

В списке вывода можно использовать операцию + (конкатенацию) строк.

Числа при выводе преобразуются в строки.

Ввод данных

- Программе часто приходится запрашивать извне значения для своих переменных. Необходимые программе данные могут извлекаться из таблиц, хранящихся на диске, а также представлять собой ответы на вопросы или списки значений.

- Термин поток применяется для обозначения последовательности данных, которые поступают из одного источника (например, с клавиатуры или из файла на диске).

Интерактивный ввод данных
Потоки ввода

Внутри общедоступного класса **System** уже существует объект абстрактного класса **InputStream** с именем **in**, который связан с клавиатурой.

Ввод данных в программу с клавиатуры удобнее всего осуществлять через буфер, для этого следует использовать встроенный класс BufferedReader.

Чтение строк

В классе BufferedReader имеется типизированный метод readLine, который присваивает строку вводимого с клавиатуры текста строковой переменной

```
String s=in.readLine();
```

Чтение чисел

В классе **Double** имеется типизированный метод **parseDouble**, который преобразует строковое значение к числовому простейшему типу **double**.

```
String s=in.readLine();
double d=Double.parseDouble(s);
```

Аналогично, в классах Float и Integer имеются типизированные методы parseFloat и parseInt, которые преобразует строковое значение к простейшим типам float и int, соответственно.

```
String s=in.readLine();
float f=Float.parseFloat(s);
```

```
String s=in.readLine();
int k=Integer.parseInt(s);
```

Объявление потока для ввода с клавиатуры

```
import java.io.*;
```

Объект **System.in** должен передаваться в качестве параметра конструктору другого класса, InputStreamReader, в результате чего создается объект.

```
InputStreamReader s;
s = new      InputStreamReader (System.in);
BufferedReader in;
in=new BufferedReader (s);
```

Вместо четырех строк в программе кратко записывают:

```
BufferedReader in=new BufferedReader (new
    InputStreamReader (System.in));
```

В процессе ввода/вывода часто могут возникать ошибки, связанные с внезапным завершением данных или наличием у них неправильного формата.

Такие события называют **исключениями**.

Java требует, чтобы в каждом методе указывались **все возможные** исключения.

Поэтому, прежде чем приступить к считыванию/записи данных, необходимо добавить команду throws IOException после объявления каждого метода, в котором осуществляется чтение данных, и любого метода, который вызывает его, и не обрабатывает исключение.

Для примера рассмотрим программу со встроенным приветствием и просьбой пользователей ввести их имена.

```
import java. io. *;
class Privetstvie {
Privetstvie () throws IOException  {
BufferedReader in = new BufferedReader
(new InputStreamReader (System. in));
System. out. print (“Как вас зовут?”);
String imja = in. readLine ();
System. out. println (“Здравствуй, “+imja);
                                                    }
public static void main (String args [ ])
throws IOException { new Privetstvie ();
                                  }
                                  }
```

Условный оператор.

В Java существует два оператора выбора:

if и switch.

Синтаксис оператора if

if (условие) {операторы1; } else {операторы2; }

Часть else не является обязательной. Если она отсутствует, то когда условие не выполняется, будет выполнен оператор, следующий за оператором if.

Условие – это булево выражение.

Например: 1) day! = 29

2) (age >=16) & (age < 75)

3) if (number >0) {

System. out. println (“Positive”)

}

else {

System. out. println (“Negative”)

4) if (day = = 25) {

System. out. println (“Christmas, Hooray”);

}

Switch – отбор с использованием ключа

Синтаксис оператора Switch

```
switch (выражение целого типа или типа char) {
case значение1 : оператор; break;
case значение2 : оператор; break;
. . .
default : оператор; break;
}
```

Switch – отбор с использованием ключа

Вычисляется значение выражения и сравнивается с перечисляемыми ниже значениями.

Если вычисленное значение выражения совпадает с одним из перечисленных ниже, то вначале выполняется соответствующий оператор, а затем **все остальные операторы**, пока не встретится break либо конец оператора switch.

Если совпадения значений нет, то выполняется оператор, определенный по умолчанию (относящийся к default).

В теле оператора каждое значение может быть указано только один раз.

Виды циклов

Цикл for

for (int имя_перем = нач_зн; условие продолжения цикла; шаг_изменения) {
тело цикла }

Переменная имя_перем инициализируется значением выражения нач_зн.

Второй параметр содержит условие, зависящее от значения переменной имя_перем.

Если проверка показала, что условие истинно, программа переходит к выполнению тела цикла.

Каждый раз при прохождении цикла значение переменной меняется в соответствии с параметром шага изменения.

После чего снова выполняется проверка условия.

Цикл повторяется, пока условие не станет ложным.

Следующая конструкция выводит на экран 5 рядов звездочек

```
for  (int i=0; i<5; i++) {
System.out.println (“*****”);   }
```

Примеры использования цикла FOR.

1) Обратный отсчет.

Может выполняться с помощью оператора --.

Например: for (int n = 10; n > = -6; n --)[

{ System.out.println (n+” “); }

Итог работы

10 9 8 7 6 5 4 3 2 1 0 -1 -2 -3 -4 -5 -6

2) Пустой цикл.

Когда начальное условие превосходит конечное тело цикла совсем не выполняется.

3) Вложенные циклы, т.е. циклы, расположенные внутри других циклов.

В таких случаях переменные циклов должны быть разными.

4) Бесконечный цикл.

Чтобы цикл for выполнялся без остановки, можно пропустить часть, в которой проверяется условие.

for (int i=0; ; i++)

5) Прерванный цикл.

Для остановки цикла в процессе итерации можно использовать оператор break, передающий управление в конец цикла. Он используется в месте с условным оператором if.

Циклы с выходом по условию

Цикл с пост_условием

do {операторы}
while (условие);

Операторы выполняются 1 раз.

Затем проверяется условие продолжения цикла, если оно не выполняется, то цикл повторяется.

Если логическое выражение ложно, т. е. условие не выполняется, то цикл заканчивается, и начинает выполняться следующий за ним оператор.

Циклы с выходом по условию

Цикл с пред_условием

while (условие)

{операторы}

Проверяется условие.

Если оно истинно, выполняются операторы, после чего условие проверяется снова.

Этот процесс повторяется до тех пор, пока условие не станет ложным.

В этом случае управление передается оператору, следующему после цикла.

Классы-оболочки

Значения встроенных (простейших) типов и объекты не должны смешиваться. Примитивные типы: char, int, long, float, double, boolean.

Пакет Java.lang имеет классы-оболочки, ассоциированные с каждым из примитивных типов: Character, Integer, Long, Float, Double и Boolean.

Если необходимо передать стандартному пакету объект, а не значение одного из простейших типов, то значение вначале помещается в класс-оболочку и становится объектом.

Классы-оболочки поддерживают разнообразные методы преобразования.

Методы преобразования между Integer, int и String

```
Integer   (int   i);      // конструктор, v-переменная
Integer  valueOf (String  s);
int    intValue();
int   parseInt  (String      s);
String toString (int i);
```

Конструктор и методы экземпляра предоставляют возможность свободно перемещаться между типами и классами.

Поэтому, если i – это тип int, то

Integer Iobj= new Integer (i);

делает из i объект. Чтобы получить int назад для вывода на печать, надо использовать

System.out.println(Iobj.intValue());

Методы преобразования между Double, double и String

```
Double   (double   v);      // конструктор, v-переменная
Double valueOf (String  s);
double doubleValue();
double parseDouble (String      s);
String toString (double v);
```

Форматированный вывод чисел

Абстрактный класс NumberFormat предоставляет методы, позволяющие получить объект, форматирующий числа, т.е. осуществляющий структурирование чисел, придавая данным на выходе удобочитаемый вид.

Для настройки объектов форматирования чисел можно использовать следующие методы:

NumberFormat getInstance()

// Системные настройки форматирования по умолчанию

Void setMaximumIntegerDigits(int k);

// Максимальное количество цифр целой части числа

Void setMinimumIntegerDigits(int k);

// Минимальное количество цифр целой части

Void setMaximumFractionDigits(int k);

// Максимальное количество цифр дробной части числа

Void setMinimumIntegerDigits(int k);

// Минимальное количество цифр дробной части

В программе используется конструкция:

```
NumberFormat Nd;
double d;
Nd= NumberFormat.getInstance()
Nd.setMaximumIntegerDigits(5);
Nd.setMinimumIntegerDigits(2);
Nd.setMaximumFractionDigits(3);
Nd.setMinimumFractionDigits(3);

System.out.println(“После форматирования получим: ”+Nd.format(d))
```

С учетом установок выводимое на экран вещественное число в целой части будет содержать от 2-х до 5-ти цифр, а в дробной части ровно 3 цифры

Примеры программ. 1. Программа «Служащие компании»

```
//класс Person определяет только имя человека
public class Person {
String name;
int age;
public String Signature()
{
      return name;
      }
}
// Класс Employee наследует классу Person и расширяет его
class Employee extends Person {
      public int yearsWithCompahy; // Количество лет
                        // в компании
      public int salary;  // зарплата
      public int position; // должность
      int Bonus () { // Метод вычисления премии
        return 0;
                }
      public String Signature()
      {
        return name;
        }
                        }
```

Примеры программ. 1. Программа «Служащие компании»

```
// Класс Manager описывает  менеджеров в компании
// Он выведен из класса Employee и переопределяет метод Bonus
class Manager extends Employee {
        int Bonus () {
          if (yearsWithCompahy > 2)
          return salary/100 * 15 ;
          else
          return salary/10;
                  }
                          }
//Класс Programmer описывает программистов
// Он выведен из класса Employe и переопределяет метод Bonus
class Programmer extends Employee {
        int Bonus () {
          if (yearsWithCompahy > 5)
          return salary/20 ;
          else
          return 0;
                  }

//Программа CalculateTotal вычисляет суммарную премию и
//                          затем ее печатает
public static int CalculateTotal (Employee[] empArray) {
        int sum = 0;
        for (int i = 0; i < empArray.length; i++) {
          sum += empArray[i].Bonus();
          System.out.println(i+"-я премия= "+ empArray[i].Bonus()+"\t");
                                          }
        return sum;
        }
```

Примеры программ. 1. Программа «Служащие компании»

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


        Employee[] workForse = new Employee[3];
        workForse[0] = new Manager();
        workForse[0].salary = 100000;
        workForse[0].yearsWithCompahy = 14;
        workForse[1] = new Programmer();
        workForse[1].salary = 30000;
        workForse[1].yearsWithCompahy = 34;
        workForse[2] = new Programmer();
        workForse[2].salary = 40000;
        workForse[2].yearsWithCompahy = 6;
        int sum = CalculateTotal (workForse);
        System.out.println(" Total bonus = " + sum);
}
}
```

Результаты работы

0-я премия= 15000

1-я премия= 1500

2-я премия= 2000

Total bonus = 18500

Примеры программ. 2. Программа «Телефон»

```
import java.awt.*;
import java.awt.event.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
class Call extends Frame implements ActionListener
{
 Button b[] = new Button[11];
 Button b1,b2,b3,b4;
 Button LastButton;
 int i;
 Label l;
 String s;
public Call()

 {
 setResizable(false);
 l = new Label();
 l.setVisible(true);
 l.setBounds(49,30,140,90);//окошко вызова положение и размер
 add(l);
```

Примеры программ. 2. Программа «Телефон»

```
b1 = new Button();
b2 = new Button();
b3 = new Button();
b4 = new Button();
b1.setVisible(true); b2.setVisible(true);
b3.setVisible(true); b4.setVisible(true);
b1.setSize(65,25);//размеры кнопок
b2.setSize(60,25);
b3.setSize(40,25);
b4.setSize(40,25);
b1.setLocation(35,125);// положения кнопок
b2.setLocation(140,125);
b3.setLocation(60,225);
b4.setLocation(140,225);
b1.setBackground(Color.gray); // цвет кнопок
b2.setBackground(Color.gray); // цвет кнопок
b3.setBackground(Color.gray);
b4.setBackground(Color.gray);
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
b4.addActionListener(this);
b1.setLabel("вызвать");
b2.setLabel("очистить");
b3.setLabel("*");
b4.setLabel("#");
add(b1);add(b2);
add(b3);add(b4);
LastButton = new Button();
```

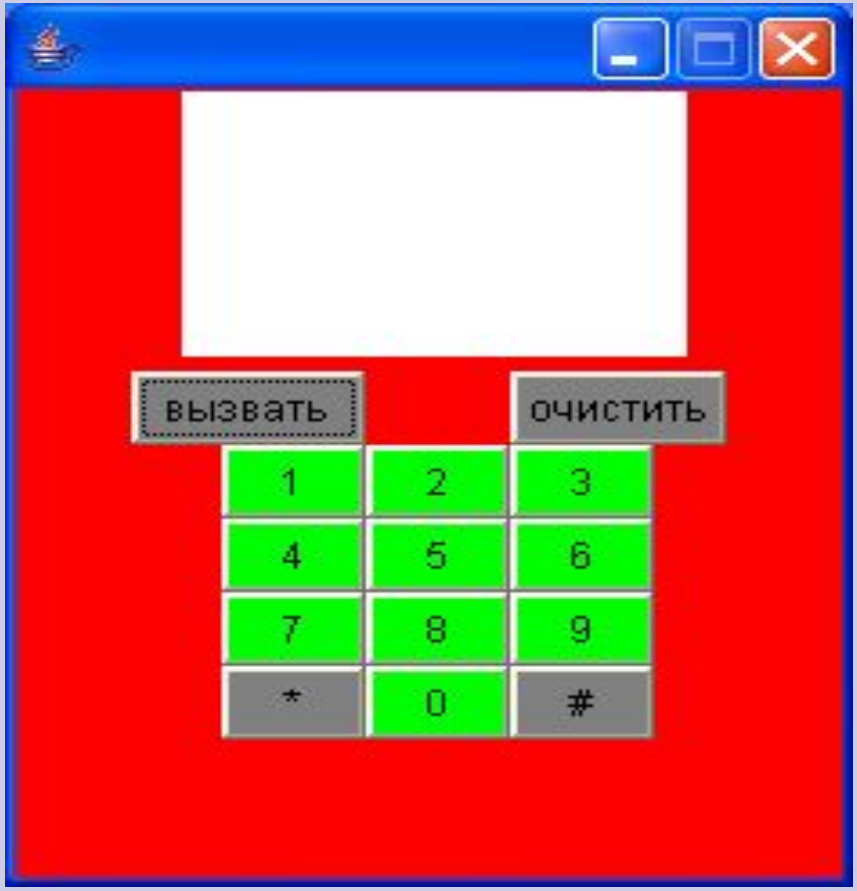
Примеры программ. 2. Программа «Телефон»

```
for (i=1;i<=10;i++)
 {
 b[i] = new Button();
 b[i].setCursor(getCursor());
 b[i].setLabel(""+(i-1));
 b[i].setBackground(Color.green);// цвет кнопок с цифрами
 b[i].addActionListener(this);
 add(b[i]);
 b[i].setSize(40,25);
 b[i].setLocation(30+(i*40),150);
 b[i].setVisible(true);
 }
 b[2].setLocation(60,150); // кнопка 1
 b[3].setLocation(100,150); // кнопка 2
 b[4].setLocation(140,150); // кнопка 3
 b[5].setLocation(60,175);  // кнопка 4
 b[6].setLocation(100,175); // кнопка 5
 b[7].setLocation(140,175); // кнопка 6
 b[8].setLocation(60,200);  // кнопка 7
 b[9].setLocation(100,200); // кнопка 8
 b[10].setLocation(140,200);// кнопка 9
 b[1].setLocation(100,225); // кнопка 0
   //
 setVisible(true);
 setBounds(300,100,235,300);//размеры и положение формы
 setBackground(Color.red); // цвет формы
 setCursor(HAND_CURSOR);
 //
add(LastButton);
 LastButton.setVisible(false);
```

Примеры программ. 2. Программа «Телефон»

```
b[10].setSize(40,25);//размеры кнопки №9
  b[10].setLocation(140,200);//положение кнопки №9
  addWindowListener(new WindowAdapter()
  {
   public void windowClosing(WindowEvent e)
   { System.exit (0); }
  });
}
public void actionPerformed(ActionEvent e)
{ Object source = e.getSource();
 if (source == b)
 { System.exit(0);
 }
 if (source==b[1]) l.setText(l.getText()+b[1].getLabel());
 if (source==b[2]) l.setText(l.getText()+b[2].getLabel());
 if (source==b[3]) l.setText(l.getText()+b[3].getLabel());
 if (source==b[4]) l.setText(l.getText()+b[4].getLabel());
 if (source==b[5]) l.setText(l.getText()+b[5].getLabel());
 if (source==b[6]) l.setText(l.getText()+b[6].getLabel());
 if (source==b[7]) l.setText(l.getText()+b[7].getLabel());
 if (source==b[8]) l.setText(l.getText()+b[8].getLabel());
 if (source==b[9]) l.setText(l.getText()+b[9].getLabel());
 if (source==b[10]) l.setText(l.getText()+b[10].getLabel());
 if (source==b1) l.setText("      Пошел вызов ;)))))");
 if (source==b2) l.setText("");
 if (source==b3) l.setText(l.getText()+b3.getLabel());
 if (source==b4) l.setText(l.getText()+b4.getLabel());
}
public static void main (String[] args)
{
 new Call();
}
```

Примеры программ. 2. Программа «Телефон»
Результаты работы



- Из демонстрационных программ видно, что JAVA представляет собой объектно-ориентированную среду. Объекты хранят информацию о предметах из реального мира (людей, служащих компании), а каждый класс объектов определяет, какая функция должна быть применена к ним, например, при вычислении размера премии (функция BONUS).
- Все демонстрационные программы (Служащие компании и Телефон) содержат несколько классов. Метод main служит в них “начальной точкой” конструктора программы: с этого места начинается её начальная работа.

Лекция 2

Даты, календари и время

Класс Date

Класс Date входит в пакет java.util, представляет даты и время в виде, независимом от системных установок.

Рассмотрим структуру класса.

Date ();
Date (**long** date);
boolean after (Date when);
boolean before (Date when);
boolean compareTo (Date d);
boolean equals (Object obj);
long getTime();
long setTime(long time);
String toString();

Например, при использовании:

Date today = new Date ();

Будет создан объект, содержащий сегодняшнюю дату и системное время.

Для вывода на экран

System.out.println(today);

Формат вывода:

День_недели Месяц День Часы:Мин:Сек Зона

Для получения конкретной даты нужен класс Calendar

Обработка Исключений

На случай возникновения непредвиденных ситуаций (разрыв связи у пользователей, ошибочное удаление файла, неверны ввод данных) у Java есть специальный объект – исключение.

Исключение – это объект, который сигнализирует о том, что возникла нестандартная ситуация.

Хотя в Java существует много заранее определенных объектов, связанных с исключениями, программист, пользователь может создать и свой собственный.

Примеры исключений, генерируемых JAVA

- ArithmeticException
- FileNotFoundException
- IOException
- NumberFormatException
- ArrayIndexOutBoundsException

Иногда посредством исключений удобнее управлять программой, чем посредством оператора IF

Обработка Исключений

Чтобы программа отреагировала на заранее определенную исключительную ситуацию, такую как EOFException, надо определить две части оператора try – catch.

- try – указать блок операторов, в котором вызов любого метода или выполнение другой операции может привести к возникновению исключительной ситуации

  (данный блок должен предваряться словом try)

- catch – снабдить блок одним или несколькими обработчиками и поставить перед ними ключевое слово catch.

  Обработчик сам по себе является блоком, который содержит операторы, связанные с сообщением об исключительных ситуациях или ошибках.

Обработка Исключений

Схема использования

```
try   { операторы, при выполнении которых
        может возникнуть исключение}
catch  (тип исключение e1)
        {операторы, которые реагируют на
         исключение и исправляют ситуацию}

catch  (тип исключение e2)
        {операторы, которые реагируют на
         исключение и исправляют ситуацию}
…остальные операторы catch.
```

Обработка Исключений

Если исключение возникает в блоке try и соответствует какому – либо типу в списке параметров одного из блоков catch,

то управление передается этому блоку catch и выполняются его операторы.

При отсутствии подходящих блоков catch исключение передается следующему выполняемому методу.

Обработка Исключений

Например.

```
try    {
   for (;;)
 number = Stream. readInt (fin);
 total += number;
          }

 catch (EOFException   e)   {
System. out. println (“Все данные прочитаны”);
```

Обработка Исключений

Схема объявления нового исключения.

```
class имя_искл extends Exception {
public имя_искл () {     }
public имя_искл ( String s) {
Super (s)
    }
}
```

имя_искл – имя, которое будет присвоено новому исключению. Второй конструктор передает любую специально определенную строку обработчику исключений, благодаря чему она будет доступна для метода getMessage.

Это объявление должно быть сделано там, где его смогут увидеть операторы try и catch, генерирующие и улавливающие его.

Обработка Исключений

Объявив класс исключений имя_искл, мы можем **генерировать** объекты этого класса по следующей схеме:

throw new имя_искл («сообщение»);

Это сообщение объясняет суть исключительной ситуации и может быть использовано оператором, обрабатывающим исключения (catch), для предоставления информации пользователю путем вызова метода getMessage или сообщение может быть пустым.

МАССИВЫ

В Java массивы представляют собой объекты, которые могут создаваться и передаваться в другие методы.

Процесс создания и использования массивов в Java можно условно разделить на 3 основных этапа

1) Объявление массива

2) Выделение памяти для элементов массива

3) Инициализация элементов массива

Использование массивов

Объявление массива происходит аналогично объявлению переменной в Java.

Если синтаксис объявления переменной в Java выглядит, например, следующим образом:

```
double  myVar;
```

То объявление массива осуществляется аналогично:

```
double  myVar[ ];
```

Выделение памяти для вновь объявленного массива также осуществляется аналогично выделению памяти для объекта с использованием оператора new:

```
double  myVar [ ];
myVar = new double [40];
```

Эти два оператора можно объединить в один следующим образом:

```
double  myVar [ ] = new double [40];
```

Использование массивов

Java также осуществляет поддержку многомерных массивов, в этом случае синтаксис языка имеет следующий вид.

```
int myArr [ ] = new int [10] [10];
```

Обязательным требованием при работе с массивами является то, что индекс массива должен быть целочисленного типа int. Т.е. максимальное число элементов массива может быть 2 147 483 647, однако на практике использование большого числа элементов в массиве приводит к ошибке нехватки памяти.

Использование массивов

Другим требованием является использование в качестве начального индекса 0. Если в процессе компиляции Java программы определяется выход за пределы объявленного массива, то компилятор формирует ошибку,

например: double x;

```
double  myVar [ ];
myVar = new double [40];
x = myVar [40];
// Ошибка! Последний элемент
// myVar [39]
```

Использование массивов

После объявления и выделения памяти для массива необходимо произвести его инициализацию, другими словами, заполнить его значениями.

Инициализация массива является необязательным требованием, т.к. при выделении памяти происходит инициализация по умолчанию.

Это означает, что элементам массива присваиваются значения в соответствии с типом данных, используемым при объявлении массива, например 0 для целочисленных типов.

Использование массивов

Инициализация элементов в синтаксисе Java имеет следующий вид:

```
myVar [0] = 10,24;
myVar [1] = 17,27;
```

или

```
for (int i=0; i<40; i++)
    myVar [i] = i/10;
```

Использование массивов

Инициализировать массив можно также используя разделитель фигурных скобок,

например:

```
float myVar [4] = {47.34, 17.6, 14.9, 8.75};
int myArr [3] [3] = {{1, 5, 7}, {5, 9, 12}, {44, 78, 11}};
```

При этом можно даже не указывать размерность массива, т.к. она автоматически определится компилятором, т.е.

```
int myArr [ ] [ ] = {{1, 5, 7}, {5, 9, 12}, {44, 78, 11}};
```

Использование массивов

Следует отметить также, что в Java поддерживается так называемый альтернативный синтаксис объявления массивов. Это касается расположения разделителя [ ], который может размещаться как после имени переменной, так и после указателя типа массива.

Например

```
float myVar [40];
```

или

```
float [40] myVar;
```

В качестве типов данных объявляемых массивов могут выступать классы Java – программы.

Сортировка массивов

Одной из самых распространенных **операций обработки массивов** является их **сортировка.** Единственного эффективнейшего алгоритма сортировки нет, ввиду множества параметров оценки эффективности:

- **Время** — основной параметр, характеризующий быстродействие алгоритма. Называется также вычислительной сложностью.

Для упорядочения важны *худшее*, *среднее* и *лучшее* поведение алгоритма в терминах размера списка (n).

Для типичного алгоритма хорошее поведение —

это O(n log n) и плохое поведение — это $O(n^2)$.

Идеальное поведение для упорядочения — O(n). Алгоритмы сортировки, использующие только абстрактную операцию сравнения ключей всегда нуждаются по меньшей мере в O(n log n) сравнениях в среднем;

Сортировка массивов

- **Память** — ряд алгоритмов требует выделения дополнительной памяти под временное хранение данных. При оценке используемой памяти не будет учитываться место, которое занимает исходный массив и независящие от входной последовательности затраты, например, на хранение кода программы.
- **Устойчивость** (stability) — устойчивая сортировка не меняет взаимного расположения равных элементов.
- **Естественность поведения** — эффективность метода при обработке уже упорядоченных, или частично упорядоченных данных. Алгоритм ведёт себя естественно, если учитывает эту характеристику входной последовательности и

Сортировка массивов

Алгоритмы устойчивой сортировки

- Сортировка пузырьком (англ. *Bubble sort*) — сложность алгоритма: O(n2); для каждой пары индексов производится обмен, если элементы расположены не по порядку
- Сортировка перемешиванием (Шейкерная, Cocktail sort, bidirectional bubble sort) — Сложность алгоритма: O(n2)
- Сортировка вставками (Insertion sort) — Сложность алгоритма: O(n2); определяем где текущий элемент должен находится в упорядоченном списке и вставляем его туда
- Блочная сортировка (Корзинная сортировка, Bucket sort) — Сложность алгоритма: O(n); требуется O(k) дополнительной памяти
- Сортировка подсчётом (Counting sort) — Сложность алгоритма: O(n+k); требуется O(n+k) дополнительной памяти

Сортировка массивов

- Сортировка слиянием (Merge sort) — Сложность алгоритма: O(n log n); требуется O(n) дополнительной памяти; выстраиваем первую и вторую половину списка отдельно, а затем — сливаем упорядоченные списки
- In-place merge sort — Сложность алгоритма: O(n2), O(n log2 n) или O(n log n) в зависимости от применяемого алгоритма слияния.
- Двоичное дерево сортировки (Binary tree sort) — Сложность алгоритма: O(n log n); требуется O(n) дополнительной памяти
- Цифровая сортировка (Сортировка по отделениям, Pigeonhole sort) — Сложность алгоритма: O(n+k); требуется O(k) дополнительной памяти
- Лексикографическая или поразрядная сортировка (Radix sort) — Сложность алгоритма: O($n \cdot k$); требуется O(n) дополнительной памяти

Сортировка массивов

Алгоритмы неустойчивой сортировки

- Сортировка выбором (Selection sort) — Сложность алгоритма: O($n2$); поиск наименьшего или наибольшего элемента и помещения его в начало или конец упорядоченного списка
- Сортировка Шелла (Shell sort) — Сложность алгоритма: O($n \log n$); попытка улучшить сортировку вставками
- Сортировка расчёской (Comb sort) — Сложность алгоритма: O($n \log n$)
- Пирамидальная сортировка (Сортировка кучи, Heapsort) — Сложность алгоритма: O($n \log n$); превращаем список в кучу, берём наибольший элемент и добавляем его в конец списка
- Плавная сортировка (Smoothsort) — Сложность алгоритма: O($n \log n$)
- Быстрая сортировка (Quicksort) — Сложность алгоритма: O($n \log n$) — среднее время, O($n2$) — худший случай; широко известен как быстрейший из известных для упорядочения больших случайных списков; с разбиением исходного набора данных на две половины так, что любой элемент первой половины упорядочен относительно любого элемента второй половины; затем алгоритм применяется рекурсивно к каждой половине
- Introsort — Сложность алгоритма: O($n \log n$)
- Patience sorting — Сложность алгоритма: O($n \log n + k$) — наихудший случай, требует дополнительно O($n + k$) памяти, также находит самую длинную увеличивающуюся подпоследовательность

Сортировка массивов

Рассмотрим программу, реализующую 11 алгоритмов сортировки массива.

Алгоритм поразрядной сортировки:

Каждое число представляется в двоичном виде. Поиск движется в 2 направлениях: снизу ищем 0, сверху - 1. Разбиваем на 2 интервала, чтобы предотвратить неправильную сортировку и сортируем по 2 биту. По старшему биту ищем сверху 1, а снизу 0, меняем местами. Находим все. Если найдены все, переходим на второй бит и так до тех пор пока не кончатся биты. Требует меньше попыток, быстрее даже чем быстрая сортировка.

Сортировка массивов

Быстрая сортировка (англ. *quicksort*) — широко известный алгоритм сортировки, разработанный английским информатиком Чарльзом Хоаром. Самый быстрый из известных универсальных алгоритмов сортировки массивов (в среднем O(n log n) обменов при упорядочении n элементов).

Быстрая сортировка использует стратегию «разделяй и властвуй». Шаги алгоритма таковы:

- Выбираем в массиве некоторый элемент, который будем называть *опорным элементом*. С точки зрения корректности алгоритма выбор опорного элемента безразличен. С точки зрения повышения эффективности алгоритма выбираться должна медиана, но без дополнительных сведений о сортируемых данных её обычно невозможно получить. Известные стратегии: выбирать либо средний по положению элемент (элемент с индексом [n/2]), либо элемент со случайно выбранным индексом.

Сортировка массивов

- Операция *разделения* массива: реорганизуем массив таким образом, чтобы все элементы, меньшие или равные опорному элементу, оказались слева от него, а все элементы, большие опорного — справа от него. Обычный алгоритм операции:
  - два индекса — l и r, приравниваются к минимальному и максимальному индексу разделяемого массива соответственно
  - индекс l последовательно увеличивается до r или до тех пор, пока l-й элемент не окажется больше опорного;
  - индекс r последовательно уменьшается до l или до тех пор, пока r-й элемент не окажется меньше опорного;
  - если r = l — найдена середина массива — операция разделения закончена, оба индекса указывают на опорный элемент;
  - если l < r — найденную пару элементов нужно обменять местами, после чего продолжить операцию разделения с тех значений l и r, которые были достигнуты.

Сортировка массивов

1. Рекурсивно упорядочиваем подмассивы, лежащие слева и справа от опорного элемента.
2. Базой рекурсии являются наборы, состоящие из одного или двух элементов. Первый возвращается в исходном виде, во втором, при необходимости, сортировка сводится к перестановке двух элементов. Все такие отрезки уже упорядочены в процессе разделения.

Сортировка массивов