Програмни средства за Интернет

Лекции	2 ч.
Лабораторни упражнения	1,5 ч.
Форма на контрол	Изпит

Лектор:

Доц. д-р Мариана Горанова

Катедра „Програмиране и компютърни технологии“, ФКСУ
Технически университет – София
Кабинет: 2302

E-mail: mgor@tu-sofia.bg

URL: http://pct.tu-sofia.bg/MGoranova/InternetProgramming

Литература

I. Основна

1. Брус Екел, Да мислим на Java, том I и II, Софт Прес, 2001.
2. Кен Арнолд, Програмният език Java, ИнфоДАР, 2001.
3. Herbert Shild, Java 2 – ръководство за програмиста, Софт Прес, 2001.

II Допълнителна

1. H. M. Deitel, P. J. Deitel, S. E. Santry, Advanced Java 2 Platform, How to Program, Prentice Hall, 2002.
2. Ira Pohl, Charlie McDowell, Java By Dissection, Addison Wesley Longman, Inc., 2000.
3. Cay S. Horstman, Computing Concepts with Java 2 Essentials, Second Edition, Jon Wiley & Sons, Inc., 2000.
4. John Lewis, William Loftus, Java Software Solutions, Foundations of Program Design, Second Edition, Addison-Wesley Longman, Inc., 2000
5. David Flanagan, Java in a Nutshell, A Desktop Quick Reference, Third Edition, O’Reilly, 1999.
6. H. M. Deitel. P. J. Deitel, Java How to Program, Second Edition, Prentice Hall, 1998.

Въведение в езика за програмиране Java

Езикът за програмиране служи за кодиране на алгоритмите и структурите от данни в програма.

Езикът Java е създаден от Sun Microsystems през 1995 г.

Характеристики:

- **приложения – програми, изградени на основата на концепцията на обектно-ориентираното програмиране;**
- **аплети – програми от страна на клиента – активни мрежови приложения, които се вграждат в HTML документи, разпространяват се във WWW и се стартират от браузъри;**
- **сървлети – програми от страна на сървъра.**

Предимства:

- **прост и мощен;**
- **осигурява безопасност;**
- **обектно-ориентиран;**
- **надежден;**
- **интерактивен;**
- **интерпретиращ и с висока производителност**
  - **код на програмата** – име.java;
  - **компилаторът** (javac) **конвертира файла в байткод (междинно представяне)** – име.class;
  - **виртуалната машина на Java интерпретира байткода;**
- **независим от архитектурата;**
- **лесен за изучаване;**
- **притежава богата обектна среда.**

Библиотеки с класове и пакети

JDK (Java Development Kit) – средата за разработка на Java програми

Java API (Java Application Programming Interface) – стандартната библиотека с класове, групирани в пакети.

Пакет – елемент на езика, групира класове под общо име. Класовете от пакета java.lang се използват автоматично.

Импортиране на класове от пакет:

```
import <име_на_пакет>.<име_на_клас>|*;
```

* **всички класове от пакета**

```
import javax.swing.JApplet;
       име_на_пакет  име_на_клас
import javax.swing.*;  // всички класове от пакета javax.swing
```

Структура на проста програма на Java

```
import <име_на_пакет>.*;
class <име_на_потребителски_клас>
{
  public static void main (String[] args)
  {
    <тяло>
  }
}
```

import – **импортиране на класове от пакет**

class – **фундаментална концепция в Java**

main() – **метод – извиква се автоматично от интерпретатора при изпълнение на Java програма**

- **не връща резултат (void);**
- **видим и достъпен от произволен клас (модификатор на достъп public);**
- **параметърът args предава съдържанието на масива ([]) от обекти от класа String (java.lang) на командния ред;**
- **извиква се без предварително да се създава екземпляр на класа и работи само с локални и статични променливи (static).**

Пълно име на клас:

<име_на_пакет>.<име_на_клас>

Пример: Java приложение

```
import java.util.*;
public class First {
  public static void main(String[] args) {
    System.out.println("Добре дошли!");
    System.out.println("Днес е " + new Date());
  }
}
```

Резултати:

```
Добре дошли!
Днес е Tue Aug 08 17:57:00 EEST 2006
```

Клас System (java.lang)

- **стандартен изходен поток – екран (винаги е отворен) out**

  public static final PrintStream out

- **стандартен входен поток – клавиатура (винаги е отворен) in**

  public static final InputStream in

Клас String (java.lang) – представя символен низ

Клас PrintStream (java.io) – реализира изходен поток с методи за отпечатване

public void println(String s)
Отпечатва низа s и завършва с нов ред.

public void print(String s)
Отпечатва низа s, но не завършва с нов ред.

Клас InputStream (java.io) – представя входен поток от байтове.

Обект – елемент, който се управлява в програмата чрез извикване на методи.

Извикване на метод:

<обект>.<име на метод>(<параметри>)

```
System.out.println("Добре дошли!");
```

обект метод

Клас Date (java.util) – представя дата

Създаване на обект:

new <име_на_клас>(<параметри>)

new Date()

Изпълнение на приложение

1. **Редактиране**

   Името на файла First.java **трябва да съвпада с името на класа.**
2. **Компилация**

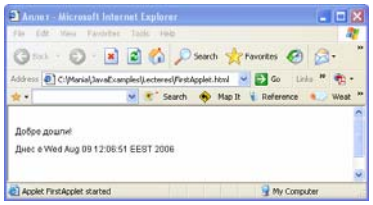
   javac First.java

   Създава се файл с байткода First.class**.**
3. **Изпълнение**

   java First

Пример: Java аплет

```
import javax.swing.JApplet;
import java.awt.Graphics;
import java.util.Date;
public class FirstApplet extends JApplet {
  public void paint(Graphics g) {
    g.drawString("Добре дошли!", 0, 25);
    g.drawString("Днес е " + new Date(), 0, 50);
  }
}
```



Аплет

- **наследява класа** JApplet **(**javax.swing**) чрез клаузата** extends**;**
- **има модификатор на достъп** public**.**

Браузърът изпълнява аплет, като извиква автоматично метода paint **на класа** JApplet**.**

Клас JApplet **(**javax.swing**)**

Аплет е малка програма, която не се изпълнява самостоятелно, а се вгражда в друго приложение. JApplet **е суперклас за всеки аплет, вграден в Web страница.**

public void paint(Graphics g)

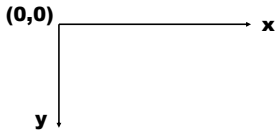
Отпечатва компонентата.

Клас Graphics **(**java.awt**) – съдържа методи за изчертаване на графични примитиви.**

public void drawString(String str, int x, int y)

Изчертава низа str **от точката (**x,y**) надясно.**

Горният ляв ъгъл на екрана е с координати (0,0).



Изпълнение на аплет

1. **Редактиране –** FirstApplet.java
2. **Компилация**

   javac FirstApplet.java
3. **Вграждане на аплета в HTML документ**

   Името на файла с разширение .html **не е необходимо да съвпада с името на класа.**

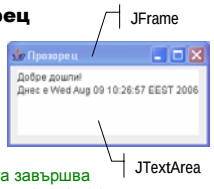
```
<! FirstApplet.html>
<HTML>
  <HEAD>
    <TITLE>Аплет</TITLE>
  </HEAD>
  <BODY>
    <APPLET CODE="FirstApplet.class" WIDTH=250 HEIGHT=100>
    </APPLET>
  </BODY>
</HTML>
```

4. **Стартиране на аплет**
   - **в браузър**

     File ⇒ OpenFile ⇒ FirstApplet.html
   - **с AppletViewer**

     appletviewer FirstApplet.html

Пример: Приложение с прозорец




```
import javax.swing.*;
import java.awt.*;
import java.util.*;
public class FirstFrame {
  public static void main(String[] args) {
    // Създава фрейм
    JFrame frame = new JFrame("Прозорец");
    // При затваряне на прозореца програмата завършва
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    // Получава съдържанието на контейнера
    Container container = frame.getContentPane();
    // Създава компоненти и ги поставя във фрейма
    JTextArea text = new JTextArea(5,10);
    text.setText("Добре дошли!" + "\n" + "Днес е " + new Date());
    container.add(text);
    // Установява размера и изобразява фрейма
    frame.setSize(250, 130);
    frame.setVisible(true);
  }
}
```

Клас JFrame **(**javax.swing**) – представя прозорец**

public JFrame(String title)
Конструкторът създава прозорец със заглавие title**.**

public void setDefaultCloseOperation(int operation)
Установява операцията при затваряне на прозореца.

operation =
- JFrame.DO_NOTHING_ON_CLOSE — **нищо**
- JFrame.HIDE_ON_CLOSE — **скрий прозореца**
- JFrame.DISPOSE_ON_CLOSE — **освободи прозореца**
- JFrame.EXIT_ON_CLOSE — **завърши приложението**

public Container getContentPane()
Връща обект от класа Container **със съдържанието на фрейма.**

public void setContentPane(Container contentPane)
Установява съдържанието на фрейма.

Клас Container **(**java.awt**) – представя контейнер – компонента, която може да съдържа други компоненти**

public Component add(Component comp)
Добавя компонента comp **към контейнера.**

Клас Component **(**java.awt**) – представя компонента**

public void setSize(int width, int height)
Установява размера на компонентата с ширина width **и височина** height**.**

public void setVisible(boolean b)
Изобразява/скрива компонентата според стойността на b **(**true/false**).**

Класът JFrame **наследява класовете** Component **и** Container **и техните методи.**

Клас JTextArea **(**javax.swing**) – представя текстова област**

public JTextArea(int rows, int columns)
Конструкторът създава празна текстова област с rows **реда и** columns **стълба.**

public void setText(String t)
Установява текста t **в текстовата компонента.**

Лексически елементи

1. **Множество от символи**
   - **а) главни и малки латински букви и символ за подчертаване**
     A÷Z a÷z _
   - **б) арабски цифри**
     0÷9
   - **в) шестнадесетични цифри**
     0÷9 A÷F a÷f
   - **г) служебни символи**
     .,:;?!<>()[]+-*/\=%&|~"{}#$
     празни символи: интервал, нов ред, табулация, коментари

2. **Резервирани думи – запазени имена за вградени типове, модификатори и средства за управление изпълнението на програмите**
   - **не могат да се използват за имена на променливи, класове и методи.**

   int public for return

3. Идентификатор – име на:
- **пакет;**
- **клас;**
- **интерфейс;**
- **поле на клас;**
- **метод на клас;**
- **параметър на метод на клас;**
- **променлива;**
- **етикет;**
- **константа.**

Идентификатор е комбинация от букви (латински), цифри и знак за подчертаване (_).
а) винаги започва с главна буква или _ , не се допускат интервали;
б) различен от резервираните думи;
в) главните и малките букви се различават.
name Name NAME sum

Типове имена:
- **Прости имена – единствен идентификатор;**
- **Квалифицирани имена – няколко идентификатора, между които има точка (.).**

Правила за имената:
- **Пакети – квалифицирани имена**

  java.lang javax.swing
- **Класове и интерфейси – съществителни, комбинация от главни и малки букви с първа главна буква на всяка дума**

  JTextArea System
- **Методи – глаголи, комбинация от главни и малки букви с първа малка буква, а следващите думи започват с главна буква**

  getContentPane setVisible toString

- **Полета – съществителни, комбинация от главни и малки букви с първа малка буква, а следващите думи започват с главна буква**

  firstName
- **Константи в интерфейси и** final **променливи – последователност от думи с главни букви и разделени със символа за подчертаване (_)**

  MIN_VALUE MAX_VALUE EXIT_ON_CLOSE
- **Локални променливи и параметри – къса последователност от малки букви, без да образува думи**

  buf len i j

4. Коментар

а) С коментар

```
/* Коментарът може да обхваща няколко реда
*/
```

б) C++ коментар

```
// Коментар до края на реда
```

в) за документиране (javadoc**)**

```
/** Коментарът може да обхваща няколко реда и
   се използва за създаване на документация */
```

Променливи, константи и присвояване

1. Променлива – име, чрез което се обръщаме към паметта, където е съхранена стойност на данна.

Декларация на променлива

<модификатор> <тип> <идентификатор> = израз |
инициализатор на масив, ...

<идентификатор> **– име на променливата**
<тип> **– тип на променливата**
<модификатор> **– достъп до променливата**

```
public int a, b;
public int sum = 0;
```

2. Константа – съдържа дадена стойност през цялото време на съществуването си.

Декларация на константа

final <тип> <идентификатор> = <израз> | <инициализатор на масив>, ...

<идентификатор> **– с главни букви**
<модификатор> – final

```
final int DIMENSION = 100;
```

3. Оператор за присвояване =

<идентификатор> = <израз>;
- **изчислява изразът от дясната страна;**
- **резултатът се запазва в паметта;**
- **достъпът до тази памет се осъществява чрез името на променливата от лявата страна.**

```
sum = a + b;
```

Типове данни

Типът данни е множество от стойности, които имат общи характеристики:

- **област от стойностите на данните;**
- **операциите, които се изпълняват върху тях;**
- **заеманата памет.**

Основни типове данни:

1. **Примитивни типове**
   - **числа и символи;**
   - **операциите се изпълняват чрез оператори.**
2. **Сложни типове данни или обекти**
   - **дефинират се чрез клас;**
   - **операциите се представят чрез методи, които се изпълняват върху обектите от този клас.**

Символен низ

Символният низ е обект – дефиниран е клас String (java.lang). Низовата константа се заградя в " ".

Оператор за слепване на низове +

```
String s="Добре "+"дошли!";
```

Типовете данни, върху които се извършва операторът (+), определят дали ще се извърши събиране или слепване на низове.

Пример: Слепване на низове

```
System.out.println("25 и 12 са слепени: " + 25+12);
System.out.println("25 и 12 са събрани: " + (25 + 12));
```

Резултати:

```
25 и 12 са слепени: 2512
25 и 12 са събрани: 37
```

Пример: Въвеждане и извеждане на символен низ

```
import java.io.*;
class IOString {
  public static void main(String[] args) throws IOException {
    BufferedReader stdin;
    stdin = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("Въведете трите си имена: ");
    String name = stdin.readLine();
    System.out.println("Вашето име е " + name);
  }
}
```

Прехвърля изключението IOException, без да го обработва

Чете символи

Чете байтове от клавиатурата

Чете ред от текст

Резултати:

```
Въведете трите си имена: Иван Георгиев Петров
Вашето име е Иван Георгиев Петров
```

Въвеждане и извеждане на низ

Клас BufferedReader (java.io) – чете последователност от символи от входен поток с буфериране.

```
public BufferedReader(Reader in)
```

Създава буфериран символен входен поток.

```
public String readLine()
```

Чете ред от текст, връща обект от тип String и при възникване на входно/изходна грешка предизвиква изключението IOException, което не се обработва, ако в main() се използва клаузата throws.

Клас InputStreamReader (java.io) – като наследник на класа Reader чете байтове от входния поток и ги преобразува в символи.

```
public InputStreamReader(InputStream in)
```

Създава InputStreamReader.

Примитивни типове данни и изрази

Тип	Памет [бита]	Минимална стойност	Максимална стойност	Стойност по подразбиране
byte	8	-128	127	0
short	16	-32768	32767	0
int	32	-2147483648	2147483647	0
long	64	-9E18	9E18	0L
float	32	≈-3.4E+38 7 зн.ц.	≈3.4E+38 7 зн.ц.	0.0f
double	64	≈-1.7E+308 15 зн.ц.	≈1.7E+308 15 зн.ц.	0.0d
char	16	'\u0000' (0)	'\uffff' (65,535)	'\u0000'
boolean *		false	true	false

Паметта за примитивните типове данни се отделя в стека (RAM памет). Размерът на всеки тип е еднакъв за всички хардуерни платформи. Всички числени типове са със знак.

*** не е точно дефиниран**

Класовете Byte, Short, Integer, Long, Float, Double, Character и Boolean са външни обвивки на примитивните типове данни.

Методите parseXXX преобразуват низ в съответния примитивен тип и могат да предизвикат NumberFormatException.

- public static byte parseByte(String s)
- public static short parseShort(String s)
- public static int parseInt(String s)
- public static long parseLong(String s)
- public static float parseFloat(String s)
- public static double parseDouble(String s)
- public static boolean parseBoolean(String s)
- public char charAt(int index)

Методът charAt на класа String връща символа в зададената позиция index от низ.

Въвеждане на примитивни типове данни

```
import java.io.*;
BufferedReader stdin;
stdin = new BufferedReader(new InputStreamReader(System.in));
byte b = Byte.parseByte(stdin.readLine());
short s = Short.parseShort(stdin.readLine());
int x = Integer.parseInt(stdin.readLine());
long l = Long.parseLong(stdin.readLine());
float f = Float.parseFloat(stdin.readLine());
double d = Double.parseDouble(stdin.readLine());
boolean bool = Boolean.parseBoolean(stdin.readLine());
char c = stdin.readLine().charAt(0);
```

Целите числа се представят чрез фиксирана запетая след младшия разряд.

Реалните числа имат цяла и дробна част с фиксирано място на десетичната точка или с мантиса и порядък в експоненциален вид. Представят се с две части: двоична дроб и двоичен порядък (представяне с плаваща точка).

При въвеждане целите и реалните числа се разделят с интервал, табулация или нов ред.

Представяне на:

1. **Цели числа**
   - **десетично;**
     123
   - **шестнадесетично (0x или 0X пред числото);**
     0xa5 0XF2
   - **осмично (0 пред числото).**
     034
2. **Реални числа**
   - **десетично;**
     1234.56
   - **експоненциално.**
     1234.56e-6

Пример: Препълване при работа с цели числа

```
class Test {
  public static void main(String[] args) {
    int j = 2147483647;
    System.out.println(j + " " + (j+1) + " " + (j+2));
    int i = 1000000;
    System.out.println(i * i);
    long l = i;
    System.out.println(l * l);
    System.out.println(20296 / (l - i));
  }
}
```

Резултати:

```
2147483647 -2147483648 -2147483647
-727379968
1000000000000
Exception in thread "main" java.lang.ArithmeticException: / by zero
at Test.main(Test.java:7)
```

Пример: Грешки при закръгляване с реални числа

```
class Test {
  public static void main(String[] args) {
    float a, b, c;
    a = 2.0e6f + 1.0f;
    b = a - 2.0e6f;
    System.out.println(b);
    a = 2.0e7f + 1.0f;
    b = a - 2.0e7f;
    System.out.println(b);
    c = 1.0f/41;
    System.out.println(c*41);
  }
}
```

Резултати:

```
1.0
0.0
0.99999994
```

Символният тип char е последователност от символи в определен ред. Java използва символната последователност Unicode, която представя символ чрез 16 бита и поддържа 65536 символа.

Символна константа – загражда се с литерали ' '.

```
char letter = 'A', digit = '7';
```

Специални символи (escape последователности)

\b	**връщане позиция назад**
\t	**табулация**
\n	**нов ред**
\r	**връщане на каретката**
\"	**"**
\'	**'**
\\	****

Логическият тип boolean **има две стойности:** true **и** false**, т.е. истина или лъжа. Никой друг тип не може да се конвертира в** boolean**.**

```
boolean state = true;
boolean cond = 7>5;
```

Изрази

Изразът се състои от операнди свързани с операции. Операндите могат да бъдат константи, променливи, методи или комбинация от тях. Операциите биват: унарни (с един операнд) и бинарни (с два операнда).

Резултатът от изпълнението на израз в Java може да бъде:

- **променлива (в C се нарича lvalue);**
- **стойност;**
- **нищо –** void **израз (извикване на метод, който не връща стойност).**

Ако изразът не може да се изпълни нормално, се получава изключение, изразът прекъсва и цялата програма прекъсва.

Напр. при целочислено деление или целочислен остатък, ако стойността на десният операнд е 0, операторът прехвърля ArithmeticException**.**

Видове изрази

1. **Постфиксни**
2. **Унарни**
3. **Мултипликативни**
4. **Адитивни**
5. **Преместване**
6. **Отношения**
7. **Равенство**
8. **Побитови**
9. **Логически**
10. **Оператори за присвояване**

Постфиксни изрази

Постфиксен оператор увеличаване/намаляване с 1

<идентификатор>++

<идентификатор>--

<идентификатор> **– променлива от числен тип**

Към стойността на променливата се добавя/изважда 1 и сумата/разликата се запазва в променливата. Резултатът е стойността на променливата (а не променлива) преди запазване на новата стойност.

```
int x = 5, y;
y = x++;
System.out.println("x=" + x + " y=" + y);   // x=6 y=5
```

Унарни оператори + - ++ -- ~ ! (тип)

Асоциативността им е отдясно наляво.

Префиксен оператор увеличаване/намаляване с 1

++<идентификатор>

--<идентификатор>

<идентификатор> **– променлива от числен тип**

Към стойността на променливата се добавя/изважда 1 и сумата/разликата се запазва в променливата. Резултатът е стойността на променливата (а не променлива) след запазване на новата стойност.

```
int x=5, y;
y=++x;
System.out.println("x=" + x + " y=" + y);              // x=6 y=6
```

Унарен +

+<идентификатор>

Унарен -

-<идентификатор>

Побитово унарно отрицание ~

~идентификатор

```
x       00101010
~x      11010101        // всички битове се инвертират
```

Логическо унарно отрицание НЕ !

!<идентификатор>

Резултатът е true, ако операндът има стойност false; резултатът е false, ако операндът има стойност true.

```
boolean x = true;
boolean y = !x;          // x=true, y=false
```

Принудително преобразуване на типа

(<тип>)<израз>

Резултатът е стойността на израза, преобразуван в указания тип.

```
int i = 7;
System.out.println ((float)i/2);     // 3.5
```

Мултипликативни оператори * / %

/ и % може да предизвикат ArithmeticException само при целочислени операнди.

```
14/4        // 3
14%4        // 2

5%3         // 2      5.0%3.0       // 2.0
5%(-3)      // 2      5.0%(-3.0)    // 2.0
(-5)%3      // -2     (-5.0)%3.0    // -2.0
(-5)%(-3)   // -2     (-5.0)%(-3.0) // -2.0
```

Адитивни оператори + -

Ако единият операнд е от тип String, то операторът + е слепване.

```
1+2+" sum"      // "3 sum"
"sum " +1+2     // "sum 12"
```

Оператори за преместване << >> >>>

```
35>>2       // 00100011    35
            // 00001000    8
-8>>1       // 11111000    -8
            // 11111100    -4
-8>>>1      // 11111000    -8
            // 01111100    124
```

Оператори за отношения < > <= >= instanceof

Типът на израза е boolean.

<име_на_обект> instanceof <име_на_клас>

Оператори за равенство == !=

Типът на израза е boolean.

Побитови и булеви оператори & ^ |

Побитови оператори

Двата операнда и резултатът са от числен тип.

& побитово И
^ побитово изключващо ИЛИ
| побитово ИЛИ

```
0xff00 & 0xf0f0     // 0xf000
0xff00 ^ 0xf0f0     // 0x0ff0
0xff00 | 0xf0f0     // 0xfff0
```

Булеви оператори

Двата операнда и резултатът са от boolean тип.
Винаги се изчисляват и двата операнда.

& булево И
^ булево изключващо ИЛИ
| булево ИЛИ

```
true & false        // false
true ^ false        // true
true | false        // true
```

Логически оператори && ||

Условен И оператор &&

Използва се за бърза оценка на израз.

Двата операнда и резултатът са от тип boolean.

Резултатът е true, ако и двата операнда имат стойност true, в противен случай – false. Ако левият операнд има стойност false, десният операнд не се изчислява.

Условен ИЛИ оператор ||

Използва се за бърза оценка на израз.

Двата операнда и резултатът са от тип boolean.

Резултатът е true, ако поне един от двата операнда има стойност true, в противен случай – false. Ако левият оператор има стойност true, десният оператор не се изчислява.

Условен оператор ?:

<условен_израз> ? <израз1> : <израз2>

<условен_израз> – **тип** boolean

<израз1>, <израз2> – **числен, логически или тип клас.**

```
int a, b, c;
a = 5;
b = 10;
c = (a>b) ? a : b;          // 10
```

Оператори за присвояване = *= /= %= += -= <<= >>= >>>= &= ^= |=

1. **Оператор за присвояване**

<операнд1> = <операнд2>

2. **Комбиниран оператор за присвояване**

<операнд1> o= <операнд2>

≡

<операнд1> = <операнд1> o <операнд2>

o – * / % + - << >> >>> & ^ |

Приоритет на операторите

Операции	Асоциативност
. [] () ++ -- (постфиксно)	отляво надясно
+ - ~ ! ++ -- (<тип>)	отдясно наляво
new (<тип>)	отдясно наляво
* / %	отляво надясно
+ -	отляво надясно
<< >> >>>	отляво надясно
< <= >= > instanceof	отляво надясно
== !=	отляво надясно
&	отляво надясно
^	отляво надясно
|	отляво надясно
&&	отляво надясно
||	отляво надясно
:	отдясно наляво
=	отдясно наляво

Ред на изчисляване

1. **Първо се изчислява левият операнд на бинарните оператори.**

```
int i = 2;
int j = (i = 3) * i;          // j=9
int a = 9;
a += (a = 3);                 // a=12
```

2. **Операндите се изчисляват преди операцията (с изключение на условните оператори &&, // и ?:).**
3. **Първо се изчисляват изразите в скоби () и операторите с най-висок приоритет.**
4. **Аргументите в списък от аргументи се изчисляват отляво надясно.**
5. **Редът на изчисление е специфичен при следните изрази:**

- **изрази за създаване екземпляр на клас new;**
- **изрази за създаване на масив new;**
- **изрази за извикване на методи .;**
- **изрази за достъп до масиви [];**
- **присвоявания, включващи компоненти на масиви.**

Преобразуване на данни

1. **Разширяващо преобразуване**

От	В
byte	short, int, long, float, double
short	int, long, float, double
char	int, long, float, double
int	long, float, double
long	float, double
float	double

2. **Стесняващо преобразуване**

От	В
byte	char
short	byte, char
char	byte, short
Int	byte, short, char
long	byte, short, char, int
float	byte, short, char, int, long
double	byte, short, char, int, long, float

Преобразуването на типовете

1. **При присвояване – само разширяващо преобразуване**

```
float money;
int leva;
money = leva;                // ако leva=25, то money=25.0
```

2. **Аритметично преобразуване – операндите се модифицират чрез разширяващо преобразуване**

```
float sum;
int count;
result = sum / count;        // result е от тип float
```

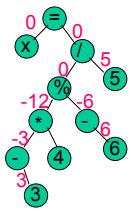
3. **Принудително преобразуване на типа**

```
int sum = 15, count = 2;
float result;
result = (float)sum / count; // result=7.5
result = sum / count;        // result=7.0
```

```
int x;
x = - 3 * 4 % - 6 / 5;
```

Резултат:

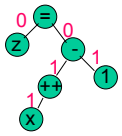
x = (-3) * 4 % (-6) / 5
x = ((-3) * 4) % (-6) / 5
x = (((-3) * 4) % (-6)) / 5
x = ((((-3) * 4) % (-6)) / 5)
x = (((-12) % (-6)) / 5)
x = (0 / 5)
x = 0




```
int x, z;
x = 1;
z = x ++ - 1;
```

Резултат:

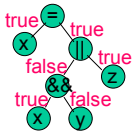
z = (x++) – 1
z = ((x++) -1)
z = (1 – 1), x = 2
z = 0




```
boolean x = true, y = false, z = true;
x = x && y || z;
```

Резултат:

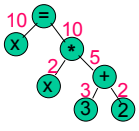
x = (x && y) || z
x = ((x && y) || z)
x = ((true && y) || z)
x = ((true && false) || z)
x = (false || z)
x = (false || true)
x = true




```
int x = 2;
x *= 3 + 2;
```

Резултат:

x *= (3 + 2)
x *= 5
x = 2*5
x = 10



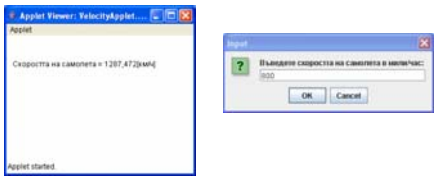
Пример: Преобразуване на скорост от мили/час в километри/час.

```
import java.io.*;
public class Velocity {
  private static final float MILES_INTO_KILOMETERS = 1.60934f;
  public static void main(String[] args) throws IOException {
  BufferedReader stdin;
  stdin = new BufferedReader(new InputStreamReader(System.in));
  System.out.print ("Въведете скоростта на самолета в мили/час: ");
  float velocity_mph = Float.parseFloat(stdin.readLine());
  float velocity_kmph = MILES_INTO_KILOMETERS * velocity_mph;
  System.out.println("Скоростта на самолета = " + velocity_kmph + "[км/ч]");
  }
}
```

Резултати:

Въведете скоростта на самолета в мили/час: 800
Скоростта на самолета = 1287.4719[км/ч]

Пример: Преобразуване на скорост от мили/час в километри/час чрез аплет.



```
import javax.swing.*;
import java.awt.*;
import java.text.DecimalFormat;
public class VelocityApplet extends JApplet {
  private static final float MILES_INTO_KILOMETERS = 1.60934f;
  private float velocity_mph;
  private float velocity_kmph;
  public void init() {
    String input = JOptionPane.showInputDialog
                    ("Въведете скоростта на самолета в мили/час: ");
    velocity_mph = Float.parseFloat(input);
    velocity_kmph=MILES_INTO_KILOMETERS*velocity_mph;
    this.setSize(300,200);
  }
  public void paint(Graphics g) {
    DecimalFormat fmt = new DecimalFormat("0.###");
    g.drawString("Скоростта на самолета = " + fmt.format(velocity_kmph) +
                 "[км/ч]", 10, 50);
  }
}
```

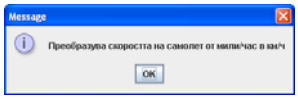
Клас JOptionPane (javax.swing)

Представя стандартен диалогов прозорец.

Метод showMessageDialog

Изобразява модален диалогов прозорец с един бутон OK. Може да изобрази съобщение, икона и заглавие на прозореца.

```
JOptionPane.showMessageDialog
(this, "Преобразува скоростта на самолет от мили/час в км/ч");
```



Метод showConfirmDialog

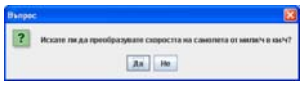
Изобразява модален диалогов прозорец, който задава въпрос на потребителя.

```
JOptionPane.showConfirmDialog(this,
"Искате ли да преобразувате скоростта на самолета от мили/ч в км/ч?",
"Въпрос", JOptionPane.YES_NO_OPTION);
```

Метод showOptionDialog

Изобразява модален диалогов прозорец с бутони, икони, съобщение и заглавие, определени от потребителя.

```
Object[] options = {"Да", "Не"};
JOptionPane.showOptionDialog(this,
"Искате ли да преобразувате скоростта на самолета от мили/ч в км/ч?",
"Въпрос", JOptionPane.YES_NO_OPTION,
JOptionPane.QUESTION_MESSAGE,
null,                  // не използва потребителска икона
options,               // заглавията на бутоните
options[0]);           // заглавие на подразбиращия се бутон
```



Метод showInputDialog

Изобразява модален диалогов прозорец за потребителски вход.

```
String input = JOptionPane.showInputDialog
                ("Въведете скоростта на самолета в мили/час: ");
float velocity_mph = Float.parseFloat(input);
```

Клас DecimalFormat **(**java.txt**)**

Форматира десетични числа.

public DecimalFormat(String pattern)

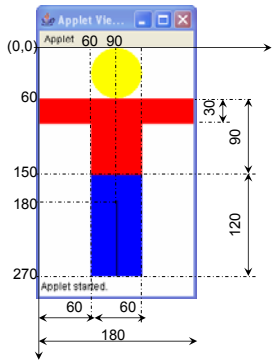
Създава DecimalFormat **обект с определен шаблон** pattern**.**

patern ="0.###" – **поне една цифра ще бъде отпечатана наляво от десетичната точка и дробната част ще бъде закръглена до 3 цифри**

String format(double number)

Връща низ, съдържащ определеното число number**, форматирано според шаблона.**

Пример: Изчертава фигура



Клас Graphics **(**java.awt**)**

public void setColor(Color c)

Установява текущия цвят на графичното съдържание.

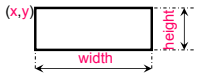
public void drawLine(int x1, int y1, int x2, int y2)

Изчертава линия между точките (x1**,**y1**) и (**x2**,**y2**).**

(x1,y1) —————— (x2,y2)

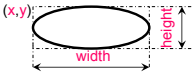
public void drawRect(int x, int y, int width, int height)
public void fillRect(int x, int y, int width, int height)

Изчертава/запълва правоъгълник с горен ляв ъгъл (x**,**y**) и размери** width **и** height**.**



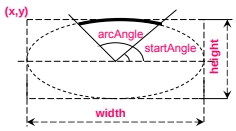
public void drawOval(int x, int y, int width, int height)
public void fillOval(int x, int y, int width, int height)

Изчертава/запълва елипса, вписана в правоъгълник с горен ляв ъгъл (x**,**y**) и размери** width **и** height**.**



void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)
void fillArc (int x, int y, int width, int height, int startAngle, int arcAngle)

Изчертава/запълва с текущия цвят дъга/сектор от елипса, ограничена от правоъгълника с горен ляв ъгъл (x**,**y**) и размери** width **и** height**. Дъгата/секторът започва от ъгъл** startAngle **и завършва на разстояние, дефинирано от** arcAngle**.**



public void clearRect(int x, int y, int width, int height)

Изчиства зададения правоъгълник, като го запълва с фоновия цвят на текущата повърхност за изчертаване.

```
import javax.swing.JApplet;
import java.awt.*;
public class StickFigure extends JApplet {
  public void init() {
    setSize(180, 270);
  }
  public void paint(Graphics g) {
    // Изчертава главата
    g.setColor(Color.YELLOW);
    g.fillOval(60,0,60,60);
    // Изчертава ризата
    g.setColor(Color.RED);
    g.fillRect(0,60,180,30);
    g.fillRect(60,60,60,90);
    // Изчертава панталоните
    g.setColor(Color.BLUE);
    g.fillRect(60, 150, 60, 120);
    g.setColor(Color.BLACK);
    g.drawLine(90, 180, 90, 270);
  }
}
```