Тема 1. Език за програмиране Java.

доц. д-р инж. Владимир Николов

13.2.2020 г.

Програма на дисциплината (извадка)

ФОРМА НА СЕМЕСТРИАЛНИЯ КОНТРОЛ	Точки – К1
Контролни и/или тестове по време на лекции	до 30
Препитване по раздели по време на лабораторни упражнения	до 40
Задача върху двата раздела	до 25
Активно участие по време на лабораторните упражнения	до 5
Общо	100

Форма на контрол	Точки – К2
Изпит - писмен със събеседване	100

Окончателна оценка в точки: К= 0,4 ×К1+ 0,6 ×К2

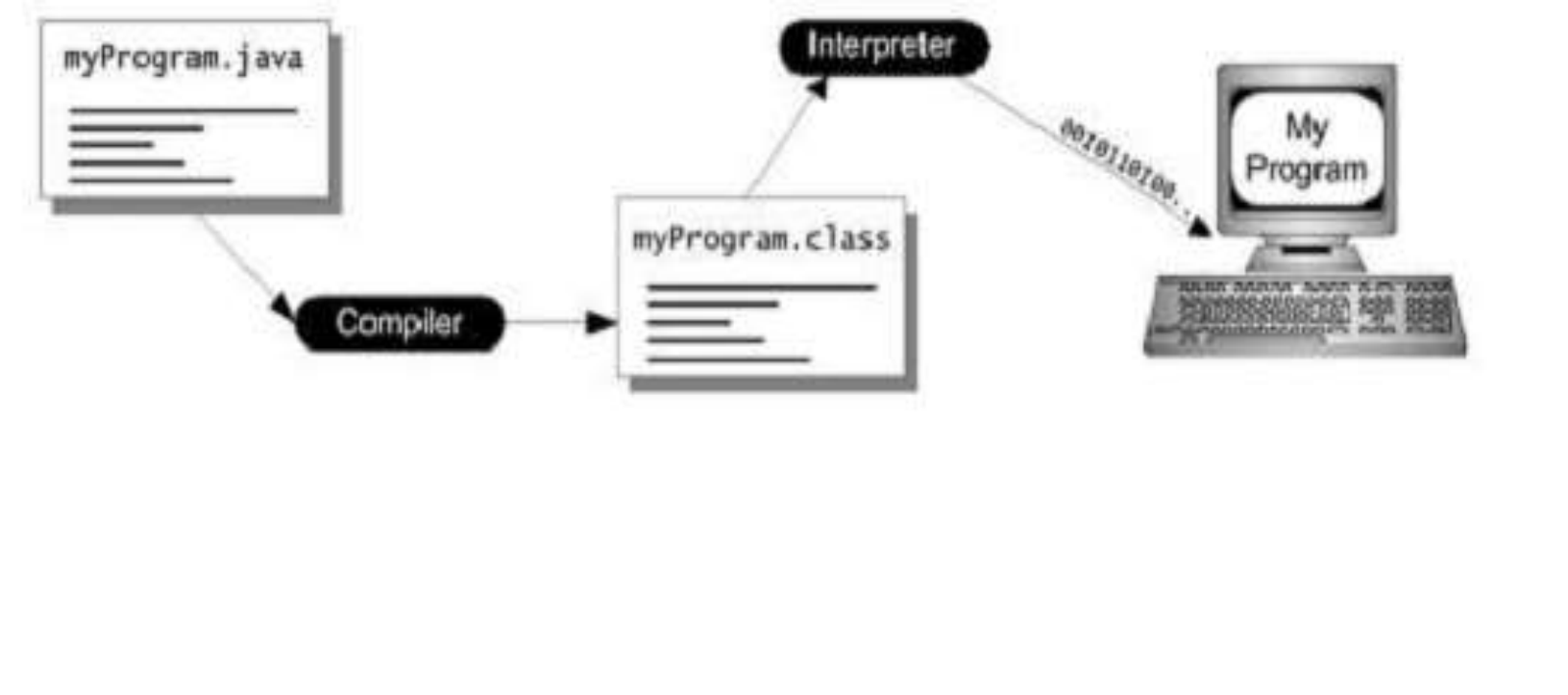
Лекция 1
Съдържание

- Характеристики на езика;
- Виртуална машина - същност;
- Главна програма на приложение
- Спецификатор static;
- Аргументи на командния ред;
- Стандартен вход/изход;
- **Прости типове-**Разлики между C++ и Java. Особености при предаване на параметри

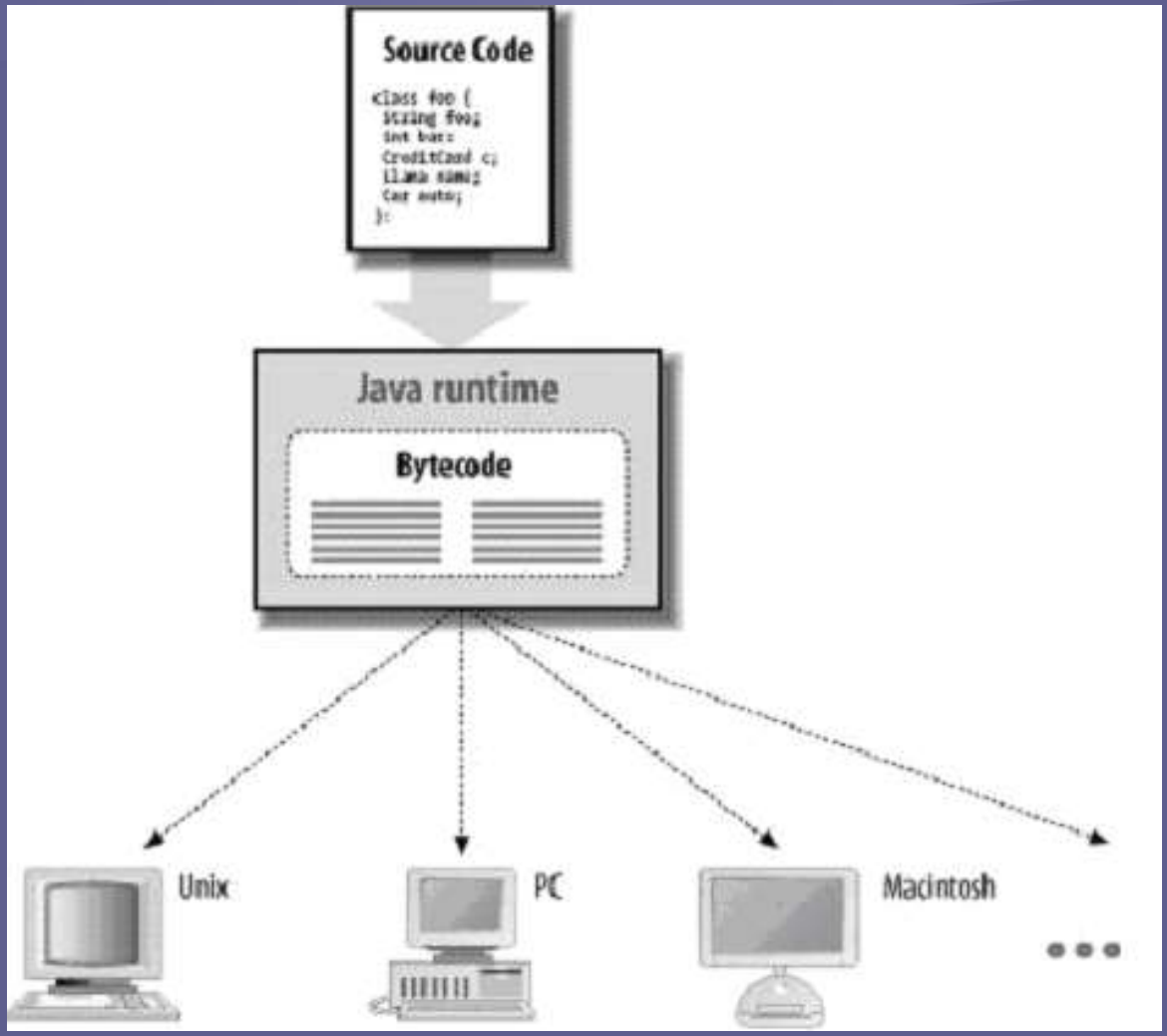
Език за програмиране Java
Характеристики на езика

- **Прост**-ограничаване на части от семантиката на C++
- **Обектно-ориентиран**;
- **Разпределен**-реализация на мрежови приложения;
- **Интерпретира се**;
- **Ясен** (премахнати са семантични части);
- **Защитен** (JVM,JRE(Classloader), разделяне на външни от вътрешни, контрол на права за достъп, запис...);
- **Платформено независим**:
  (Портируем на различни операционни среди);
- **Може да се пренася на различни машини**:
  (Портируем на различни машини);
- **Ефективен** (интерпретатор!);
- **Много-нишков** (интеграция на паралелни процеси);
- **Динамичен** (съхранител на обекти-heap).

Виртуална машина (Java Virtual Machine-JVM)



Виртуална машина (Java Virtual Machine-JVM)



Главна програма на приложение

Особености на езика:

- **Java е напълно обектно ориентиран език;**
- **Понятието глобална функция не съществува в езика (същност на main);**
- **Същност на приложението;**
- **Базов клас - системния клас на Java "Object";**

Пример 1.1. Създаване на приложение, което се записва във файл с името на главния клас (HelloWorldApp.java)

```
/**
 * HelloWorldApp class имплементира конзолно
   приложение, извеждащо
 * "Hello World!" на стандартния изход.
 */
public class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Главна програма на приложение

```
/**
 * HelloWorldApp class имплементира приложение,
   основано на технологията swing
 * извеждащо прозорец.
 */
import javax.swing.*;
public class HelloWorldApp {
  public static void main(String[] args) {
        JFrame frame = new JFrame( "Hello, Java!" );
        JLabel label = new JLabel("Label Text!", JLabel.CENTER );
        frame.getContentPane().add( label );
        frame.setSize( 300, 300 );
        frame.setVisible( true );
  }
}
```

Главна програма на приложение

Главна програма на приложение

- Изпълнение на клас:

Полученият от компилацията файл от тип <ime>.class се стартира от виртуалната машина **Java** с команда:

Java HelloWorldApp.class

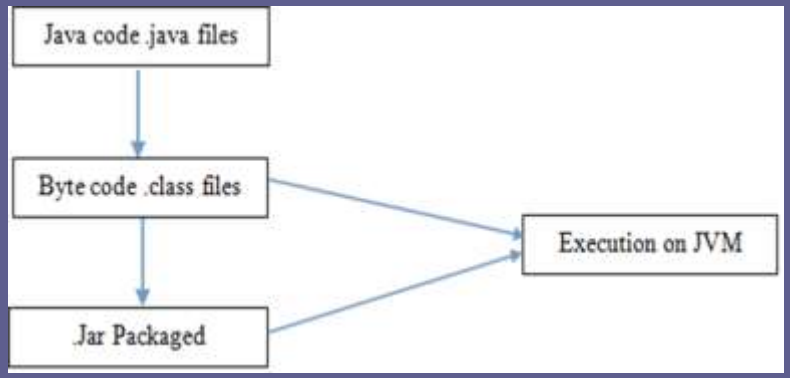
Класът, подлежащ на изпълнение трябва да е достъпен за интерпретатора. Определяне на местоположението:

- Текуща директория, ако съществува в нея;
- Чрез променливата CLASSPATH:
  - от настройките на системата (получена при инсталацията на Java или работната среда на машината);
  - CLASSPATH съдържа една или повече директории които служат за корен при търсенето на **.class** файлове.

Главна програма на приложение

- Изпълнение на Jar е подобно-има множество вх.точки:

Пряко задаване на стартовия клас:
java -cp example.jar [<пакет>.]HelloWorldApp
Чрез манифест.
1. Създаване на файл:
MANIFEST.MF (съдържание стартов клас):
Manifest-version: 1.0
Main-Class: HelloWorldApp



2. Компилиране:
javac HelloWorldApp.java
3. Архивиране заедно с класа създаване на example.jar :
jar cfm example.jar manifest.mf HelloWorldApp.class
4. Стартиране на архивата example.jar :
java -jar example.jar

Главна програма на приложение

Формиране на пътя:

- За да формира пътя от корена CLASSPATH до търсения елемент, в частност библиотечен клас, се извършва преобразуване на неговия спецификатор - започвайки от този корен интерпретаторът преобразува низа като всяка точка се заменя с наклонена черта:

Пример:

package bg.tu-varna.sit

се преобразува в:

<стойност на CLASSPATH>**bg\tu-varna\sit**

Спецификатор static

Дефиниция:

В Java се използва ключовата дума *static* която, поставена пред даден елемент, определя (по принцип) атрибут за глобална декларация (статичност).

Синтаксис:

static декларация;

Видове статични декларации **В Java** :

- Преди декларацията (дефиниция) на член променливи **(полета)** на класа. В този случай се получават **статични член променливи;**
- Преди декларацията (дефиниция) на член функция на клас. В този случай се получават **статични член функции (методи);**

Спецификатор static

Когато е поставен пред функция, ***static*** *определя* ***глобална декларация*** *на същата. Статичните методи се асоциират към класа, а не към обектите, негови инстанции.*

Статичният метод може да оперира само със статичните членове на класа!

Действието е еквивалентно на това в C++.

В Java декларацията и дефиницията на функцията съвпадат и всички член функции на класа , в Java се наричат методи имат непосредствено написани дефиниции на функции.

Поставен пред декларацията на член променливи (полета) на клас. В този случай се получават ***статични член променливи.***

Спецификатор static

Пример:

```
public class Test {
  public static double PI = 3.14; // статична(class) променлива
//  public final static double PI = 3.14; // статична(class) константа
  private double dbl = 3; // нестатична обектна променлива

  public double func (double x)  {
     return (dbl * Math.sin (x)**2);
 }
 public static double sfunc (double x) {
        return (3.0 * x);
 }
 public static double tfunc (double x) {
        return (3.0 * x * PI);
 }
 public static double wfunc (double x) {
        return (3.0 * x * dbl );// Грешка!
                    // използване на нестатична (обектна) променлива
                    // от статичен метод
 }
}
```

Спецификатор static

Сравнение на декларацията със C++:

Прилики:

- Както и в C++ статичните променливи служат за дефиниране глобална променлива към даден клас (невидима извън класа);
- При обявяването на дадена член променлива всичките създадени обекти ***използват едно и също*** място за съхраняване-обща памет;
- Това е начин, чрез който обектите на даден клас могат да "комуникират" помежду си;

Разлики:

- Не е необходимо да се дефинира допълнително извън класа(C++).
- Инициализацията-при обявяването на статичната променлива (C++).
- Статичните данни остават членове на класа, те са в обхвата на класа и могат да имат атрибути за достъп (**public**, **private**…).

Спецификатор static

Пример: Брой обекти от клас StaticSample

```
public class StaticSample {
public static int iNumStatics=0; //статична променлива
 public StaticSample(){
  iNumStatics++; // брой на създавните обекти
 }
 public static void main(String[] args) {
  StaticSample oSt1 = new StaticSample();  //обект 1
  StaticSample oSt2 = new StaticSample(); // обект 2
  System.out.println(StaticSample.iNumStatics); //брой
 }
}
```

Спецификатор static

Разлика със C++:

- Не може да се използва с разделителя за принадлежност **::**
  - **Не се допуска** синтаксиса на C++:

StaticSample::*iNumStatics*

или

Test::tfunc(…)

Аргументи на командния ред

Разлики спрямо C++ при предаване на аргументи:

Аргументите **на командния ред** се различават по:

- Броят на аргументите в двата езика е различен:
  - C и C++ се подават два аргумента
  - Java **системата подава една стойност**
- Тип на аргументите:
  - C и C++ броя на аргументите (число) и указател към масив от указатели;
  - **Java масив от обекти тип String ;**
- В C и C++ аргументите на командния ред съдържат и името на приложението.

Аргументи на командния ред

- Примерна програма на **C++**, която отпечатва аргументите на командния ред.

```
int main(int argc, char* argv[])
{
  for(int i = 1; i < argc; i++)
    cout << argv[i] << "\n";
}
```

Тази програма преминава през всеки аргумент, използвайки for цикъл като отпечатва всички на стандартния поток за изход. Цикълът стартира от 1, за да се избегне отпечатването на името на програмата!

Аргументи на командния ред

Пример за извеждане на аргументите на Java

```
public class ArgPrint {
      public static void main(String[] args) {
        for (int i = 0; i < args.length; i++)
          System.out.println(args[i]);
      }
    }
```

Методът main приема за параметър масив от обекти от тип String:

- Всички масиви на Java имат статична член-променлива за брой елементи length, която съхранява размера на масива;
- В този случай length определя броя параметри, които са предадени.

Стандартен вход/изход

- Извеждане на конзолния изход - клас System:

System.out.println(...);
System.out.print(...);

Разлика и особености:

- Няма оператор за извеждане в поток. Той се подменя от **статичен метод за извеждане** System.out.print или System.out.println (с нов ред);
- Както при оператор << на C++, **методите за извеждане на конзолата могат да получават произволен тип данни**.

Стандартен вход/изход.

Пример:

```
// извеждане на String:
   System.out.print("test string");
// извеждане на цяло число:
System.out.print(16);
// извеждане на реално число (израз):
System.out.print(5.5 * .2);
```

Стандартен вход/изход.

Особености:

- При извеждането може да се използва operator + за конкатенация на извежданите данни;
- Ако не всички операнди са от тип String:
  - Преобразуват се в String (при необходимост);
  - Създава се нов String чрез конкатенация на операндите.

Стандартен вход/изход.

Примери:

```
int x = 20, y = 10;
  System.out.println("x: " + x + "\ny: " + y);
```

Изход:

```
x: 20
y: 10
```

Стандартен вход/изход.

Аргумент на println е израз, съставен от операндите:

op1 + op2 + op3 + op4

- Операторът + се изпълнява отляво надясно. Ако има други оператори с по-висок приоритет се изпълняват преди това. Например:

"x: " + x

има ляв операнд String, десния се преобразува от int в String и се конкатенира, образувайки стойност:

"x: 20"

- Изпълнението продължава със следващите операции + “\n” е символа за нов ред :

x: 20

y: 10

Стандартен вход/изход.

Рядко се използват методите за четене :

System.in

// За четене на потребителския вход се използват производни класове, например Scanner:

```
Scanner kb = new Scanner(System.in);
//Създава обект от класа с конзолен вход
System.out.println("Enter data ...");
String input = kb.next(); //сканира входа
char choice = input.charAt(0);
```

Стандартен вход/изход.

```
Boolean bTest;
// анализ на входа по типове:
bTest = kb.hasNextDouble();
bTest = kb.hasNextInt();
//четене на входа по типове
double dbl = kb.nextDouble();
int integer = kb.nextInt();
```

Стандартен вход/изход. (5.0+)

- Чрез клас StreamTokenizer могат да се разграничават следните видове синтактични елементи tokens:
  1. Числа;
  2. Стрингове;
  3. Думи;
  4. Коментари;
  5. Празни символи - Whitespace

Форматиран вход/изход. (5.0+)

Пример:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.StreamTokenizer;
public class MainClass {
  public static void main(String[] av) throws IOException {
    StreamTokenizer tf = new StreamTokenizer(new BufferedReader(new InputStreamReader(System.in)));
    String s = null;
    int i;
    while ((i = tf.nextToken()) != StreamTokenizer.TT_EOF) {
      switch (i) {
      case StreamTokenizer.TT_EOF:   System.out.println("End of file");         break;
      case StreamTokenizer.TT_EOL:   System.out.println("End of line");         break;
      case StreamTokenizer.TT_NUMBER:  System.out.println("Number " + tf.nval);         break;
      case StreamTokenizer.TT_WORD:  System.out.println("Word, length " + tf.sval.length() + "->" + tf.sval);     break;
      default:  System.out.println("What is it? i = " + i);
      }
    }
  }
}
```

Форматиран вход/изход. (5.0+)

Въвеждане, може да се използва до EOF:

```
abcd 123 2.45
```

резултат:

```
Word, length 4->abcd
Number 123.0
Number 2.45
```

Форматиран вход/изход. (5.0+)

- ***Използване на синтактични анализатори на регулярни изрази (Regular Expressions)***

Позволяват да се прилага теорията на формалните граматики и езици на практика.Включват дефиниция на азбука, терминални и нетерминални символи и правила за анализ, извличане на елементи и получаването им в променливи.

Форматиран вход/изход. (5.0+)

Пример, разделяне на низ с разделител “,”:

```
public class MainClass {
   public static void main( String args[] )
   {
      String secondString = "1, 2, 3, 4, 5, 6, 7, 8";
      String output = "String split at commas: [";
      String[] results = secondString.split( ",\\s*" ); //рег. израз
      for ( String string : results ) {
         output += "\"" + string + "\", ";
      }
      output = output.substring( 0, output.length() - 2 ) + "]";
      System.out.println( output );
   }
}
```

Форматиран вход/изход.

- Известния от езика С форматен изход:

Създаден е метод printf( ) за PrintStream и PrintWriter.

PrintStream, printf( ) има две форми на реализация:

1. PrintStream printf(String fmtString, Object ... args)

извежда args на стандартния изход по формат, указан чрез fmtString, като използва подразбиращите се локални настройки.

2. PrintStream printf(Local loc, String fmtString, Object ... args)

Дава възможност за смяна на локалните настройки.

Форматиран вход/изход. (5.0+)

Пример за числа:

```
public class MainClass {
  public static void main(String[] av) throws IOException {
    System.out.printf("%d %(d %+d %05d\n", 3, -3, 3, 3);

    System.out.printf("Default floating-point format: %f\n", 1234567.123);
    System.out.printf("Floating-point with commas: %,f\n", 1234567.123);
    System.out.printf("Negative floating-point default: %,f\n", -1234567.123);
    System.out.printf("Negative floating-point option: %,(f\n", -1234567.123);

    System.out.printf("Line-up positive and negative values:\n");
    System.out.printf("% ,.2f\n% ,.2f\n", 1234567.123, -1234567.123);
  }
}
```

Прости типове

Прости типове:

Дефиниция: тези, които се предават по стойност, са подобни на C++:

- **boolean-1 bit;**
- **byte-8bit;**
- **short-8bit;**
- **char-16 bit;**
- **int-32 bit;**
- **long-64 bit;**
- **float-32 bit;**
- **double-64 bit;**

Променливите от прости типове се използват като стеково базирани при предаване на параметрите.

Няма еквивалент създаването на стоково базираните променливи на C++, освен за простите типове . Простите типове могат да се създават само в стека, без да се използва new.

Прости типове

Разлики:

- При символен тип (char) се използва международния 16-bit Unicode на символите, като той може автоматично да представя повечето национални символи.
- Простият тип символ както и другите прости типове (int, float, double,unsigned…) се преобразува в тип String преди отпечатване на конзолата;
- Необходимо е експлицитно преобразуване при потенциална загуба на точност, например:

```
int i = 13;
byte b = i; //  грешка
byte b = (byte) i; // OK
```

Прости типове

C++ има допълнителни:

- signed char и signed int (синоним на int);
- unsigned char и unsigned int;
- long double.

В C++ логическият тип bool се дефинира като разширение на езика:

enum bool {false, true};

boolean е вграден тип в Java (не е int)

Прости типове

Пример:

```
public class ArrayCharSamples {
    public static void main(String[] args) {
        char ch = 'a';
// Особености на извеждането на данни на конзолата:
        System.out.println("ch="+ch);
        char uniChar='\u0000';
// Кодова таблица на печатаемите символи от 16 кодови таблици по 256 символа:
        System.out.println("Array:");
        for(int i=32; i<16*256; i++) {
            uniChar = (char)i;    // преобразуване в Unicode
            System.out.print(uniChar);
        }
// масив от char с размер 5:
        char[] charArray ={ 'a', 'b', 'c', 'd', 'e' };
        System.out.println();
        System.out.print("charArray[]=");
// извеждане с използване на дължината на масива (5)
        for(int i=0; i<charArray.length; i++ )
            System.out.print(charArray[i]);
    }
}
```

Прости типове

Изход:

ch=a

Array:

!"#$%&'()*+,-./0123456789:;<=>?@

ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`

abcdefghijklmnopqrstuvwxyz...

charArray[]=abcde

Прости типове

- Статичните константни символни низове се преобразуват автоматично към обекти String (следва разглеждане на низ);

```
String strVal = "constant string...";
String strDinamic = new String("string2");
```

- Няма еквивалентен на статични символни масиви от C и C++;

```
char C_Array[30]; // не се допуска
char javaArray[]= new char[30];
```

Прости типове

Статични осмични константи

int i = **0**1230; // 664(10)

Статични шестнадесетични константи

int i = **0x**FFFF; // 65535(10)

Статични long константи:

long lng= 13L

Прости типове (Java 7+)

Статични двоични константи 0b

```
byte one = (byte)0b00000001;
byte two = (byte)0b00000010;
byte four = (byte)0b00000100;
byte sixteen = (byte)0b00001000;
long lots_o_ones =
   (long)0b11111111111111111111111111111111
   1111111L;
```

Числови константи:

```
double e = 3.00e+8; // подразбиращ се тип
float g = 3.00e+8F;     // експлицитна към float
float pi = 3.14_159_265_358; // форматиране _
```

Прости типове

Заграждащи класове (wrapper classes) :

- java.lang.Boolean;
- java.lang.Byte;
- java.lang.Short;
- java.lang.Character;
- java.lang.Integer;
- java.lang.Long;
- java.lang.Float;
- java.lang.Double.

Създават се динамично като референтен тип.
boxing (unboxing)

Прости типове

Всеки от класовете има експлицитен конструктор за създаване от простия тип и метод за получаване на простия тип;

Примери:

```
Integer classInt = new Integer(20);
int simpleInt = classInt.intValue(); //20
// логически
Boolean bClassVal=new Boolean(true);
boolean simpleBoolean = bClassVal.booleanValue(); //true
// символен
Character classChar = new Character('\u0032');
char simpleChar = classChar.charValue();
```

Прости типове

Връзка между прости типове и класовете им:

```
Float pi = new Float( 3.14 );
Float pi = new Float( "3.14" ); //! преобразуване
```

Преобразуващи методи между класовете и простите типове:

```
Double size = new Double ( 32.76 );
double d = size.doubleValue( );     // 32.76
float f = size.floatValue( );       // 32.76
long l = size.longValue( );         // 32
int i = size.intValue( );           // 32
```

Прости типове

Предназначение на класовете

- за да може да се предава стойността им при изискване от референтни типове - предаването им като параметри на методите. **Параметрите са НЕПРОМЕНЯЕМИ!**;
- когато параметър на метода е обект от базовия клас Object;
- когато се съхраняват в контейнери (колекции) или масиви от референтни типове.

Прости типове

Примери:

```
// съхраняване в динамичен списък:
List myNumbers = new ArrayList( );
Integer thirtyThree = new Integer( 33 );
myNumbers.add( thirtyThree );
// Използване
Integer theNumber = (Integer) myNumbers.get(0);
int num = theNumber.intValue( );          // 33
```

Прости типове

Примери:

```
// съхраняване в динамичен списък:
List myNumbers = new ArrayList( );
Integer thirtyThree = new Integer( 33 );
myNumbers.add( thirtyThree );
// Използване
Integer theNumber = (Integer) myNumbers.get(0);
int num = theNumber.intValue( );          // 33
```

Прости типове (инициализация)

Тип	стойност
boolean	false
char	'\u0000'
byte, int, short, long, float, double	0
Референция	null

Въпроси ?