Тема 4. Програмни елементи.

доц. д-р инж. Владимир Николов

17.3.2020 г.

Съдържание

- Програмни оператори на езика-сравнение
- Инициализация на променливите;
- Ключова дума - final :
  - променлива;
  - аргумент на метод;
  - клас
  - метод;
- Предварителни декларации;
- Пакетите в Java.
- Структури и обединения

Програмни оператори на езика

В голямата си част имат подобно действие с тези на езика C++.

Особености:

- **Има разлика във вътрешното представяне на оператора за цикъл for, който се преобразува в еквивалентен while:**

```
for(i = 0; i < 10; i++)
{
    System.out.println(i);
}
```

Еквивалентно представяне:

```
i = 0;
while (i < 10)
{
    System.out.println(i);
    i++;
}
```

Програмни оператори на езика

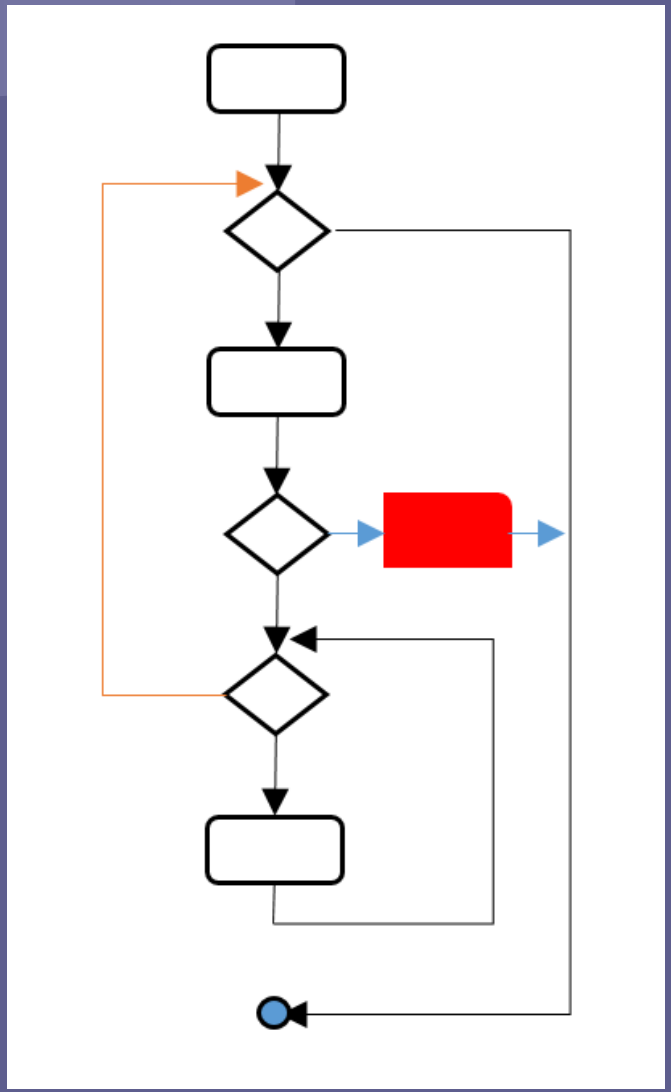
- **Не съществува оператор goto.**

Заместител на механизма за безусловно предаване на управлението е **break** *label* или **continue** *label*, който се използва за предаване на управлението от вътрешността на вложени цикли:

```
labelOne:
while ( condition ) {
   ...
   labelTwo:
   while ( condition ) {
      ... // break or continue (labelTwo, labelOne)
   }
   // after labelTwo
}
// after labelOne
```

Програмни оператори на езика

```
public class ContinueWithLabelDemo {
    public static void main(String[] args) {
        String searchMe = "Look for a substring in me";
        String substring = "sub";
        boolean foundIt = false;
        int max = searchMe.length() - substring.length();
    test:
        for (int i = 0; i <= max; i++) {
            int n = substring.length();
            int j = i;
            int k = 0;
            while (n-- != 0) {
                if (searchMe.charAt(j++) !=
                    substring.charAt(k++)) {
                    continue test;
                }
            }
            foundIt = true;
            break;
        }
        System.out.println(foundIt ? "Found":"No found");
    }
}
```



Програмни оператори на езика

- Оператори за сравнение:
  - == (равно)
  - != (различно)
  - > (по-голямо)
  - >= (по-голямо или равно)
  - < (по-малко)
  - <= (по-малко или равно)

При прости типове съответстват на операторите в езика C (C++).

Програмни оператори на езика

- Особености при сравнение на реални числа:

Точността на представяне на числата има значение при ==:

```
float value = 1.0f;
System.out.println(value);
float sum = 0.1f + 0.1f + 0.1f + 0.1f + 0.1f +
   0.1f + 0.1f + 0.1f + 0.1f + 0.1f;
System.out.println(sum);
System.out.println(sum == value);// false: 1.0 != 1.000001
```

Програмни оператори на езика

- Особености при сравнение на референтни типове:
  - За променливи от тип обект, не са приложими сравненията по големина (**<**, **>**, **<=** и **>=**).
  - Оператор == не сравнява съдържанието на обектите, а само дали референциите сочат едно и също място в динамичната памет, т. е. дали сочат един и същ обект. Операторът не проверява съдържанието на обектите по стойност, както се прави от еквивалентния оператор на езика C++. За сравнение по стойност се използва метод equals;

Програмни оператори на езика

- Пример:

```
String str = "to be or not to be";
String anotherStr = str;
String part1 = " to be";
String part2 = " or not to be ";
String thirdStr = part1  + part2;

System.out.println("str = " + str);
System.out.println("anotherStr = " + anotherStr);
System.out.println("thirdStr = " + thirdStr);
System.out.println(str == anotherStr); // true
System.out.println(str == thirdStr); // false (new reference)
```

Програмни оператори на езика

- **Нова форма на цикъла for (за версия Java 5.0**)

Това е опростена версия на цикличния оператор, която се използва по подобен начин както алгоритъма (и оператора в езици като C#) **"foreach"**. Имплементира се като итерация на серия от стойности в масиви или други типове колекции (наречени в Java collection).

Общ формат на цикъла:

```
for ( varDeclaration : iterable )
   statement;
```

varDeclaration е декларация на **типа и идентификатора** на стойностите в масив или колекция.

Цикълът може да се използва за итериране на масив от произволен тип на обекти или обекти, които имплементират интерфейса java.lang.Iterable. Това са обекти от класовете на езика Java тип Collections, които се разглеждат в следващите лекции.

Програмни оператори на езика

Примери:

```
int [] arrayOfInts = new int [] { 1, 2, 3, 4 };
for( int arrValue  : arrayOfInts ) {
    System.out.println( arrValue );
}
List<String> list = new ArrayList<String>(  );
// разглеждат се в следващите лекции
    list.add("String example 1");
    list.add("String example 2");

    for( String strVar : list ) {
        System.out.println( strVar );
}
```

Типове променливи (*final*)

***В Java е д*обавена ключовата дума *final,* която, поставена пред даден елемент определя (по принцип) атрибут за непроменяемост:**

final променлива: Определя променлива, стойността на която не може да се променя след като е инициализирана (константа).

Типове променливи (*final*)

final – Предназначение и действие:

Подобна на променливата с атрибут const на C++.

Декларира именована константа. Варианти и особености:

- Прости типове данни;
- Референтни типове данни.

Типове променливи (*final*)

Деклариране - ключовата дума **final :**
final int aFinalVar = 0; // деклариране и инициализация
Може да се отложи инициализацията за по-късно:
final int blankfinal;
. . .
blankfinal = 0;

но след като се направи един път, опитът за промяна предизвиква компилационна грешка.

final променливи

```
class Value {
  int m_i = 1;
}
public class FinalData {
  // Константа, образувана при компилация:
  final int finalInt = 9;   // пакетен достъп
  static final int VAL_TWO = 99;   // пакетен достъп
  // публично достъпна:
  public static final int VAL_THREE = 39;
  // Не може да бъдат декларират константи, присвоявани по-късно:
  final int finalInt4 = (int)(Math.random()*20);
  static final int finalInt5 = (int)(Math.random()*20);

  Value notFinalRef = new Value();
  final Value finalRef = new Value();
  static final Value staticFinalRef = new Value();
  // Масиви:
  final int[] a = { 1, 2, 3, 4, 5, 6 };
  public void print(String id) {
    System.out.println(
      id + ": " + "finalInt4 = " + finalInt4 +
      ", finalInt5 = " + finalInt5);
  }
```

final променливи

```
public static void main(String[] args) {
    FinalData fd1 = new FinalData();
    //! fd1.finalInt++; // Грешка – опит за смяна на стойността на константа
    fd1.finalRef.m_i++; // не е константа - OK!
    fd1.notFinalRef = new Value(); // не е константа - OK
    for(int i = 0; i < fd1.a.length; i++)
      fd1.a[i]++; //  не е константа - OK!
    //! fd1.finalRef = new Value(); // // Грешка – опит за смяна на константа стойност
    //! fd1.staticFinalRef = new Value(); // Грешка – опит за смяна на константна
     референция
    //! fd1.a = new int[3];
    fd1.print("fd1");
    System.out.println("Creating new FinalData");
    FinalData fd2 = new FinalData();
    fd1.print("fd1");
    fd2.print("fd2");
  }
}
```

final аргумент на метод

- **final като аргумент на метод:**

```
class Gizmo {
  public void spin() {}
}
public class FinalArguments {
  void with(final Gizmo g) {
    //! g = new Gizmo(); // грешка g е константна (непроменяема) променлива
  }
  void without(Gizmo g) {
    g = new Gizmo(); // OK -- g не е константна (непроменяема) променлива
    g.spin();
  }
  // void f(final int i) { i++; } // грешка i е константна (непроменяема) променлива
  // но може да се чете от константна (непроменяема) променлива:
  int g(final int i) { return i + 1; }
  public static void main(String[] args) {
    FinalArguments bf = new FinalArguments();
    bf.without(null);
    bf.with(null);
  }
}
```

final клас и метод

- **final клас: Класът не може да се наследява, т.е. да се използва като базов (*superclass*);**
- **final метод: Методът не може да се предекларира в наследника;**

“Препроцесор”

- Препроцесорът на C++ извършва основно търсене и замяна на идентификатори, които са били декларирани с директивите #define и typedef. Въпреки че повечето защитници на C++ не препоръчват използването на препроцесора, който е наследен от C, той е все още широко използван от C++ програмистите.

- **Java не предоставя препроцесор**. Тя предоставя подобна функционалност (на #define, typedef и т.н.) като тази на C++, но с доста повече контрол. Константните член-променливи се използват вместо #define директивата, а дефинициите на класове – вместо typedef. Крайният резултат е, че изходният код (source) на Java е много по-логичен и лесен за четене от

C++ кода.

“Header файлове”

- **C++ и организация на кода с отделни Header файлове**

  Преди да се започне прехвърляне на кода от C++ е важно да се разберат основните разлики между изходния код на в Java и C/C++. В C и C++ изходният код се създава като двойка от декларативен файл (.h) и файл имплементация (.c или .cpp). Това е направено с цел да се раздели декларацията на функциите от тяхната дефиниция. По този начин всеки програмист, използващ даден клас може да разбере кода, като прегледа декларативната част на функциите, абстрахирайки се от конкретната имплементация. (дефиницията на функциите).

“Header файлове”

- **В Java не съществуват отделни Header файлове**

  При Java има само един изходен код, написан във файл (.java) за всяка логическа програмна единица. Класовете на Java съдържат информацията за класовата декларация, която може да се извлече от изходния код с използването на класа javap, предоставен от разработващата среда (Java Development Kit). **Затова не е необходимо разделянето на декларативния h. файл от имплементацията.**

“Атрибут **inline**”

- **В езика няма методи с атрибут inline. Компилаторът на Java определя кой от методите е такъв и няма контрол от страна на програмиста. За да се предизвика компилатора да определи метода като макрос inline се използва ключовата дума final пред метода. Самостоятелните функциите тип inline изобщо са възможни само на C++**

“Атрибут **inline**”

- **В декларативната част на C/C++ кода могат да се поставят коментари за действието на отделните членове, т.е. документирането на кода. За облекчаване на документирането на кода, написан на Java се предоставя приложението javadoc, с което се генерира документация като HTML от коментарите, написани в кода на програмата.**

Преобразуване на C++ код

Ако е нужен клас от друга библиотека се използва директивата **import** и името (спецификатора) на библиотеката. Не съществуват и препроцесорни макроси, подобни на тези в C и C++.

Пример. Показва C++ включен файл (.h) за класа "топка"

Пример. C++ класа «топка»

```
// константи
    #define COLOR_RED 1
    #define COLOR_YELLOW 2
    #define COLOR_BLUE 3
    #define MATERIAL_RUBBER 1
    #define MATERIAL_LEATHER 2

    class ball {
// променливи
      float diameter;
      int color;
      int material;
    };
```

Преобразуване на C++ код

- За да се преобразува този клас в клас на Java е нужно:
  - да се премахнат препроцесорните директиви #define и от точката и запетая на края на класа. Препроцесорните директиви се заменят като се декларират член-променливи на Java класа с атрибути ***static*** *и* ***final***. Както е показано по-горе в Java типовете данни с ключовата дума ***static*** **означават, че от тях има само едно копие за целия клас**, а тези с ключовата дума ***final*** **определят, че са константни**, което обикновено е основният мотив за използване на #define в кода на C/C++.

Пример за трансформацията на Java клас. Java версията не се записва във включен файл, защото такъв не се поддържа. В преобразуваната програма на Java **дефинициите и декларациите са комбинирани в един файл**, което е .java изходния файл (source).

Преобразуване на C++ код

Java клас «топка». Недостъпни константи:

```
class ball {      // пакетен достъп
      // Константи
     static final int COLOR_RED = 1;
     static final int COLOR_YELLOW = 2;
     static final int COLOR_BLUE = 3;
     static final int MATERIAL_RUBBER = 1;
     static final int MATERIAL_LEATHER = 2;

      // Променливи
      float diameter;          // пакетен достъп
      int color;             // пакетен достъп
      int material;          // пакетен достъп
}
```

За да се осигури достъп до константите, трябва да са public, например:

```
public static final int COLOR_RED = 1;
```

Достъпът от други класове става с клас принадлежност,например:

```
int color = ball.COLOR_YELLOW;
```

Преобразуване на C++ код

- Модификаторът ***final*** се използва в Java за да се осигурят константни, само за четене стойности. **Този модификатор не осигурява ограничаването на промяната на обектите, когато те се предават като аргументи, ограничаването на връщаните стойности да бъдат само за четене и ограничаването на методите да модифицират обектите, с които работят.**
- Този пропуск е по-малкият проблем в Java спрямо C++, главно поради разликата между класа ***String*** и масивът от символи, но може да стане източник на грешки.
- Няма начин да се гарантира, че метод, който не трябва да променя обект, няма да го промени по невнимание на програмиста.

Пакетите в Java (Packages)

Предназначение

Пакетите се използват за лесно намиране, използване и премахване на колизии при използването на класовете. Те свързват класовете и интерфейсите в свързани групи, които осигуряват защита на достъпа и именни пространства.

Дефиниция:

Пакет – Съвкупност от логически свързани класове и интерфейси, осигуряващи защита от колизии и управление на именните пространства (в C++ се използва namespace).

За да се опрости търсенето и използването на класовете както и контрола на достъпа, програмистът трябва да групира свързаните класове и интерфейси в пакети.

Проблемите с дублирането на имената на програмните елементи и тяхното групиране в именни пространства, решавани с namespace в C++, в Java се решават с обединяването на класовете в пакети с наименования.

Декларации за импортиране и пакет

Ключова дума "импортиране" **import** – указва на компилатора какви пакети и класове са необходими на класовете, расположени в текущия исходен файл (source *.java). Действието е подобно на #include в C++.;

Ключова дума "пакет" **package** – определя библиотека от класове, имащи "глобално уникално име"; за уникалност на пространствата от имена на класовете (в Java именните пространства съдержат само класове) често се използват (като префикс) обърнати имена на домейни;

- С тези ключови думи се указва, какви външни класове и от кои пакети са нужни за класовете, описвани в текущия изходен файл и към какъв пакет те ще принадлежат (последното не е задължително – default package…);
- В един изходен файл (с расширение .java) може да се опише и повече от един клас, един от които по име трябва да съвпада с името на файла и извън пакета може да се достъпи само този, обявен с public. Останалите класове са с вътрешно пакетна видимост.
- Имената на стандартните пакети на java започват с java. (…).Пакетът java.lang се импортира автоматично.

Пакетите в Java, директивата import

- С концепцията за пакетирането при Java се осигурява адресиране на имената, при което се разграничава пространството, в което е разположен класа. Еднаквите имена в различните пакети са абсолютно уникални. Основното, което трябва да се осигури е отделянето на свързаните класове в общ пакет.
- **Java използва пакет вместо именно пространство.** Пакетите обединяват също и компонентите на библиотека с определено **име на библиотеката**. За използването на библиотеката в дадена програма се използва директивата **import** и компилаторът осигурява достъп до тези компоненти. Класовете и интерфейсите на езика, които са част от платформата на Java са членове на различни пакети, които свързват класове и функции. Те са базови, намиращи се в java.lang, такива за четене и запис, намиращи се в java.io и др.

Пакетите в Java, директивата import

- **Компилационна единица**

Изходният код класа на езика Java е организиран в компилационна единица. Най-простата компилационна единица, съдържаща един клас се образува като класът се създаде (напише му се код) във файл с име, съвпадащо с името на класа. Например, дефиницията на класа с име MyClass, трябва да се включи във файл с име MyClass.java. В повечето случаи компилационната единица е файл с разширение .java, но теоретично в средите за проектиране, тя може да се разглежда като по широко понятие. За опростяване на разглеждането се приема, че компилационната единица е файл.

Разделянето на класовете в отделни файлове е важно, тъй като компилаторът използва тази постановка при търсенето и достъпа до имената на класовете основавайли се на този факт. Вкючването на повече от един клас във файла е допустимо при определени ограничения, които се разглеждат допълнително при дефинициите на класовете.

Пакетите в Java. Именоване

- Пакетите се именоват йерархично (път в дърво);
- Върховете се представят (именоват) с низове и определят позиция (място) за разполагане на класове или интерфейси;
- Разделителят точка «.» служи за връзка между обозначените наименования;
- Компонентите на името на пакета съставят уникален път до съответното обозначение, който се използва от компилатора и системата за изпълнение:
  - Не се налага да се създават отношения между пакетите по други принципи;
  - Не съществува понятието "подпакет", което означава, че пакетът трябва еднозначно да определя мястото на класа в йерархията;
  - Конвенция за имената:
    - ✓ Имената на пакетите започват с малка буква;
    - ✓ Имената на класовете – с малка.

Пакетите в Java. Деклариране

- **Собствен пакет се декларира на първия (ефективен) ред на компилационна единица;**
- В една компилационна единица (файл) може да има само една команда за дефиниране на пакет;
- За декларация на пакет се използва командата **package**

Пример: Пакет за клас, реализиращ четене на данни от входен файл:

- Име на пакета- bg.tu_varna или bg.tu_varna.cs
- Избира се уникален за приложението идентификатор-низ с разделители:

package bg.tu_varna; **(bg.tu_varna.cs)**

Пакетите в Java . Имена и деклариране.

```
package bg.tu_varna;          (bg.tu_varna.cs)
import java.util.*; // пакет на класа Date
import java.io.*;    // пакет на класа System и файлов вход/изход
public class FileTest {
public void fileRead()
{
 String str;
 try { // C:\\jdev\\EclipseWorkspace\\TestApp\\FileApp\\FilePackage\\
    RandomAccessFile fin = new RandomAccessFile ("<path>\\FileTest.java","r");
     while((str = fin.readLine()) != null ) {
    System.out.println(str);
     }
     fin.close();
  } catch(EOFException e) {
     System.out.println( "End of file encountered");
  }
  catch(FileNotFoundException e) {
    System.out.println(
    "FileNotFound");
  }
  catch(IOException e) {
    System.out.println(        "IOException encountered");
   }
}
```

Пакетите в Java . Имена и деклариране.

```
public static void main(String[] args) {
    System.out.println("Hello, it's: ");
    System.out.println(new Date()); // използване на клас от пакета java.util
// използване на клас от пакет bg.tu_varna
    FileTest oFile = new FileTest();
    oFile.fileRead();
 }
}
Hello, it's:
Mon Feb 19 13:26:19 EET 2018
<съдържание на файла>
```

bg.tu_varna е пакет в който е дефиниран клас FileTest.

Класът FileTest може да се реферира от външните за компилационната единица класове. В една компилационна единица може да се дефинира само един публичен клас и файлът се именова с неговото име.

Пакетите в Java. Основни принципи.

Основни принципи при определяне на достъпа:

- За скриване на останалите вътрешни за даден клас други класове, пакетът създава подсистема със строго дефиниран интерфейс от видими за външните класове части и детайли (класове), които не се предоставят. Това са класовете, които се използват само от класовете на пакета. Те се явяват "частни", т.е. видими само за пакета класове и имат пакетна видимост. В този смисъл тези класове са "приятелски" за всички класове в пакета и невидими за външните класове.
- Всички дефинирани в отделни файлове класове и интерфейси с публичен достъп създават **интерфейса на пакета;**
- Класовете и интерфейсите **без атрибут за достъп** private или protected се подразбират като вътрешно-пакетни класове и интерфейси.
- Ако една компилационна единица няма дефиниран пакет, тя получава подразбиращ се такъв (default).

Пакетите в Java. Основни принципи.

- Класовете в пакета могат да се достъпват помежду си само по име. Когато един клас е в друг пакет, трябва да се образува адрес, включващ пълната му спецификация.

Например:

bg.tu_varna.FileTest

- За да не се налага да се пишат тези адреси на класа е предвидена директивата за включване на пакет **import** Такива директиви могат да се включат в началото на компилационната единица (файла), след което могат да се използват класовете, включени в импортирания пакет. Например:

```
import java.util.*; // пакет на класа Date (*)
import java.io.*;   // пакет на класа System и файлов вход/изход
```

- По същия принцип се получава достъпа до дефинираните в пакета публични класове. Използване на име на домейн:

```
import bg.tu_varna.*
```

- Java не решава общия проблем на колизиите между имена.

Пакетите в Java. Основни принципи.

Избор на имена на пакетите

- Имената на пакетите са пряко свързани с пътя за разполагането им във файловата система. Това правило се използва при създаване на име на пакетите. Използват се уникални идентификатори на пакетите, което предизвиква образуване на уникална директория за тяхното разполагане и липса на колизии. За уникално наименование често се използва домейн адреса на организацията, написано в обратен ред. Например:

bg.tu_varna

- За да може да се открие от компилатора в коя директория се намира файла с изходния код на програмата се използва директивата на компилатора classpath, на която се подава параметър поддиректорията на пакета. Например, ако абсолютният път на пакета е c:\jdev\src:

При създаване на пакета се образува

<позиция на пакета,classpath><позиция на класа в пакете>,напр: c:\jdev\src\bg\tu_varna

javac –classpath **c:\jdev\src c:\jdev\src\FileTest.java**

Стартиране на създаденото приложение, клас: **FileTest**

java –cp **c:\jdev\src\FileTest**

Пакетите в Java. Статични пакети.

- **Статични пакети**

Във версия 5.0. е добавена нова директива, с която могат да се импортират статични членове на класовете в кода на програмата и да се използват направо с имената на статичните им членове. Например, в математическият пакет на езика java.lang има предефинирани статични константи. По този начин се придобива илюзията за вграждане на математическите константи в езика, например:

```
import static java.lang.Math.*;
// използване
double circumference = 2 * PI * radius;
double length = sin( theta ) * side;
int bigger = max( a, b );
int positive = abs( num );
```

Пакетите в Java. Библиотечни класове (пакети)

Java поддържа различни множества от предефинирани библиотечни класове (пакети)

Средата на Java съдържа различни пакети, имплементиращи множество от фундаментални класове. Това дава добър старт в бързото програмиране на смислени приложения. Някои от основните класове са включени в следните пакети:

- **java.awt.** Голяма част от приложенията изискват създаването на интерфейс GUI. Java предоставя абстрактен прозоречен инструментариум abstract window toolkit (awt), който позволява на програмистите на интерфейси графични обекти GUI, които могат да се използват без значение от платформата, върху която ще се изпълнява програмата. Програмите могат да се стартират автоматично на всички поддържани от езика платформи;
- **java.applet. Аплет** в контекста на Java, е част от голяма програма, акцентираща към предоставянето предимно на някаква форма на свързана с изобразяване на данни върху браузър. Той е подклас на компонентите awt и предоставя възможности за обслужване на динамични изображения, анимации, звукови и видео обекти.
- **java.io. Пакетът** java.io поддържа класове за четене и запис от/към потоци, файлове и програмни тръби (pipes);

Пакетите в Java. Библиотечни класове (пакети)

- **java.lang.** Базовите класове на Java за прости типове данни: Class, Object, Boolean, Float, Double, Integer, String и други, плюс връзките с останалата обкръжаваща системна и операционна среда.
- **java.net.** Класове за мрежово програмиране. Включва работата със сокети, интернет адреси, диаграми на мрежата, унифицирани позиции на ресурсите (URLs) и управляващи методи за URL.
- **java.util.** Контейнерните класове речници, хеширани таблици с данни, дати, стекове, битови низове и низове. Непрекъснато се разширява възможностите като дава подходяща и спестяваща време имплементация на най-често използваните класове.
- **java.sql.** Класове за работа с бази данни, основана на връзка JDBC/ODBC. Осигурява универсален интерфейс към бази от данни, свързани с драйвер към ODBC.
- **java.rmi.** Класове за работа с класове и интерфейси, разположени в мрежата. Могат да се стартират отдалечени методи, използвайки техните именовани положения (адреси) в глобалната мрежа-технология RMI и CORBA.

“Структури и обединения”

Структури и обединения

В C/C++ има три типа съставни типове данни :

- Класове;
- Структури;
- Обединения.

Java поддържа само един от тези типове данни – класовете. Причини:

- Принуждава програмистите да използват класове, когато им е нужна функционалността на структурите или обединенията: по-последователен подход, защото класовете могат да представят поведението на структури и обединения;
- Поддръжката на структури и обединения се счита за отстъпление от цялата концепция на Java като обектно-ориентиран език;
- Създателите на Java всъщност са искали да запазят езика по-лесен, затова са намерили смисъл в премахването на аспектите на езика, които се припокриват.

“Структури и обединения”

Превръщането на структури и обединения в Java класове е тривиално. Следващият пример показва преобразуването на C структурата “полярни координати”.

Пример: Структура на езика C - «полярни координати».

```
typedef struct polar {
      float angle;
      float size;
    } POLAR;
```

Тази структура използва typedef за да се създаде типа polar. Декларациите typedef не се поддържа в Java, защото всичко в езика е обект с уникален тип. Затова Java не поддържа и концепцията на структурите.

Пример: клас на Java «полярни координати».

```
public class polar {
      public float angle;
      public float size;
    }
```

Точката и запетаята не са нужни при дефинициите на класове в Java!

ВЪПРОСИ?