Тема 2. Референтни типове данни.

доц. д-р инж. Владимир Николов

18.2.2020 г.

Съдържание

- **Референтни типове (** предаване на референции в Java**)**
  - ***class*** *- класове;*
    - създаване;
    - сравнение;
    - стрингова интерпретация;
    - интерпретация на хеширане;
  - ***interface*** *– интерфейс;*
- **Базов клас Object - основни методи;**
- **Наследяване**
- **Операции на езика**
- **Инициализация на променливите**
- **Предварителни декларации**

Референтни типове

Дефиниция: ***Типове данни, които се създават CAMO в динамичната памет (heap) и се предават чрез референция между методите.***

- ***class ;***
- ***Array - масиви;***
- ***interface – интерфейси.***

Предаване на параметри, предаване на референции в Java

Предаване на параметри в езика:

- В Java параметрите **се предават по стойност**, но за референтните типове, предаваната стойност е референция;
  - ⇒ При предаването на масиви или обекти от произволен клас между методите, техните елементи (полета) се променят от методите, към които се предават;
  - ⇒ Референциите обаче не могат да се променят в извиквания метод. **След връщане от метода, стойността им се възстановява.**

Предаване на параметри, предаване на референции в Java

Предаване на параметри:

В C++ копиране на обект от клас предаден с указател към обекта (по стойност), в който има член променливи указатели към данни от някакъв тип. Проблемът се решава с копиращ конструктор, който прави копие на данните по член променливата указател, а не само да копира указателя.

- В Java решението е:
  - За масиви - операцията System.arraycopy;
  - За обекти методът ***clone.***

  (разглеждат се по-подробно в лекцията)

Предаване на параметри, предаване на референции в Java

Пример:

```
public class ArrayPass {
    void f( int [] byValue )     //референция към масива
    {
        byValue[0] = 10;   // сменя се стойността при викащия arrayTest
        byValue = null;    // локално действие, не се отразява на arrayTest
    }
    void g()
    {
       int [] arrayTest = new int [3];     //създаване на референция към масива
       arrayTest[0] = 5;
       f(arrayTest); // arrayTest[0] има стойност 10
       System.out.print("arrayTest: ");
       for(int i=0; i<arrayTest.length; i++) {
          System.out.print(arrayTest[i]+" ");       //Какви стойности има в масива?
       }
    }

    public static void main(String[] args) {
        ArrayPass obj = new ArrayPass();
        obj.g(); // arrayTest: 10 0 0
    }
}
```

Базов клас Object

Дефиниция:

Основен клас, който се наследява от всички дефинирани от програмиста класове.

Задача: Дефинира основни методи за наследниците – важни за работата на всеки написан потребителски клас.

(Не се разглеждат методите, свързани с паралелни процеси и конкурентен достъп)

Базов клас Object

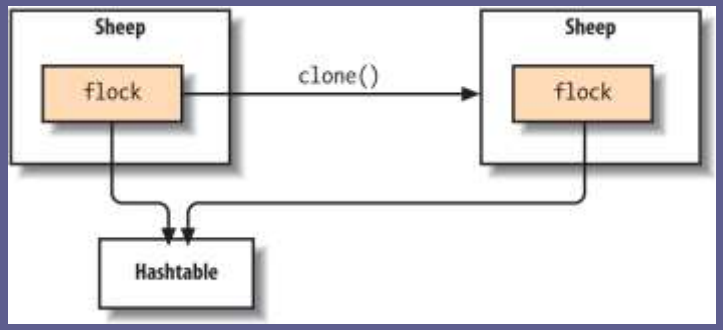
Функция	*Обяснение*
protected Object clone()	Създаване на копие на текущия обект
boolean equals(Object toCompare)	Проверка за еквивалентност на подадения toCompare с текущия.
protected void finalize()	Извиква се от менажера на паметта, когато се установи, че брояча на референции е нулев (обекта не се използва).
final Class getClass()	Връща класа наследник на базовия клас по време на изпълнение.
int hashCode()	Връща hash кода на обекта object.
String toString()	Връща низово представяне на object.

Базов клас Object

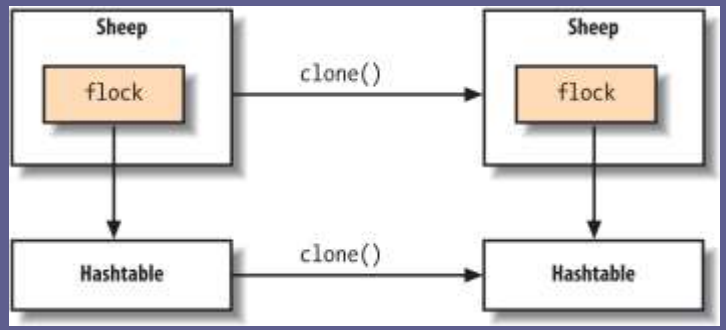
- **Object clone()** се използва за създаване на копие на обекта. Всеки клас може да декларира метода, ако допуска създаването на копие. Новият обект е има същите състояния като текущия.

Видове копиране:

Плитко:



Дълбоко:



Базов клас Object

- **Class getClass()** служи за определяне на класа по време на изпълнение.

Типовете могат да се определят по време на изпълнение подобно на езика C++. За получаване на информация за тип може да се използва следната кодова последователност:

X.getClass().getName();

където **X** е референция към паметта, която реферира обекта от класа, информация за който трябва да се получи.

Базов клас Object

Определяне на типовете по време на изпълнение downcasting

Преобразуване-проверка за съответствие (анализиране на възможностите за преобразуването и генериране на изключение (грешка в преобразуването-**bad casts**), както в C++. - ClassCastException

Пример.

```
import java.util.*; //  ArrayList, LinkedList
public class Mainapp {
    public static void test(List a) {       //предаден чрез интерфейс
      System.out.println("Testing " +
      a.getClass().getName());
    }
    public static void main(String[] args) {
      test(new ArrayList());
      test(new LinkedList());
    }
}
```

Базов клас Object

- **int hashCode()** връща hash кода на обекта; Методът се използва за получаване на кода, необходим при запис на обектите в хеширани таблици. Връщаната стойност е гарантирано консистентна по време на изпълнение на програмата;
- **void wait()** и вариантите се използват в паралелен режим на работа и са свързани с взаимно-зависими паралелни процеси.

Базов клас Object

- **boolean equals(Object toCompare)**

Същност: Интерфейсен метод за определяне на операцията сравнение за равенство на два обекта от потребителския клас-връща логическа стойност от изпълнението си.

Предназначение: За сравняване на обекти (еквивалентност).

Методът се предефинира в потребителския клас - цел съхраняването на обектите от класа в контейнери и търсене на обекти.

Допуска се като параметър да се използва обект от класа, а не базовия клас - **downcasting**.

Може да се предефинира конкретно за класа по следния начин:

```
public class CStudent {
....
  public boolean equals(Object toCompare){
    //
  }
}
```

Базов клас Object

Изисквания към функцията, реализираща метода:

- рефлексивна - за всяка референция x x.equals(x) да е винаги истина:

if (this == obj) return true;

- симетрична- за всяка референция x и y, x.equals(y) е true тогава и само тогава, когато y.equals(x) е true;
- транзитивана - за всяка референция x, y и z, ако x.equals(y) е true и y.equals(z) е true, тогава x.equals(z) е true.
- консистентна – една функция е консистентна, ако за всяка референция на обектите-x и y многократните извиквания на x.equals(y) винаги връща един и същ резултат от сравнението (не предизвиква вътрешни състояния);
- Различни класове са несравними връща се false:

if (getClass() != obj.getClass()) return false;

- Ако базовите им класове са различни, са различни:

if(!super.equals(obj)) return false;

- сравнение с нулева референция. За всяка референция към обект obj, която е ненулева obj.equals(null) е винаги false:

if (obj == null) return false;

Класове - създаване

Пример:

```
import java.io.*;
import java.util.*;
public class Person {
  String strName="N/A";
  String strGender="N/A";
  String strTitle="N/A";
  Date lastDate=new Date();
  public Person() {
  }
  public Person( String name, String gender, String title){
    strGender=gender; strName=name; strTitle=title;
    setLastDate();
  }
  public void setLastDate() {
    Calendar calendar = Calendar.getInstance(  );
    lastDate = calendar.getTime(  );
  }
  public void setName(String name) {
     strName = name;
     setLastDate();
  }
```

Класове - създаване

```
public void setGender(String gender) {
  strGender=gender;
  setLastDate();
}
public void setTitle(String title){
  strTitle = title;
  setLastDate();
}
public String getName(){
    return strName;
}
public String getGender() {
    return strGender;
}
public String getTitle(){
   return strTitle;
}
}
```

Класове - сравнение

Пример-Предефиниране на сравнение за равенство (equals) между обектите от класа студент, която имплементира равенството на студентите по факултетен номер:

```
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    if(!super.equals(obj))
        return false;
    final Student other = (Student) obj;
    if (strFacNumer == null) {
        if (other.strFacNumer != null)
            return false;
    } else if (!strFacNumer.equals(other.strFacNumer))
        return false;
    return true;
}
```

Класове – стрингова интерпретация

- **public String toString():**

Имплементиране на стрингово преобразуване на потребителски обект:

- **Като се предефинира метода toString() на базовия клас Object;**
- **Осигурява възможност за преобразуването на информацията за неговите атрибути (полета) в стрингов еквивалент.**

Предназначение:

- За извеждане обекта на конзолния изход;
- За предаване на обекта към поток (конзола);
- За предаване на обекта в мрежата.

Класове – стрингова интерпретация

При кодирането на метода се използва подходящо преобразуване на неговите полета в стринг. Ако полетата са от прости типове или класове с имплементация на метода toString() се използват с конкатенацията на типа String:

Примерна интерпретация на метода toString за класа Person. Преобразува членовете на класа като низ и ги връща.

```
public String toString(){
      return(strTitle +"
   "+strName+"\tUpdated:"+lastDate.toString());
  }
```

Класове – интерпретация на хеширане

public int hashCode() : Имплементиране на код за хеширани таблици

- **За да се съхранява обекта от клас в хеширани таблици е необходимо да се дефинира метод за изчисление на кода за хеширане , например Person:**

```
public int hashCode() {
  final int PRIME = 31;
  int result = 1;
  result = PRIME * result + ((strName == null) ? 0 : strName.hashCode());
  result = PRIME * result + ((strGender == null) ? 0 : strGender.hashCode());
  result = PRIME * result + ((strTitle == null) ? 0 : strTitle.hashCode());
  return result;
}
```

Класове – област на създаване

Видове класове:

- **На ниво файл;**
- **На ниво пакет;**
- **Вградени (Inner Classes) в обхват {}:**
  - **В класове вътрешни класове, принадлежат към друг клас;**
  - **В методи като променлива с ограничена видимост;**
  - **Анонимни вътрешни класове-класове без име.**

  Представляват синтактична надстройка на езика. Нямат смисъла на динамичен тип, който се създава по време на изпълнение, както е в други динамични езици. (ще бъдат разгледани при възможност в следващите лекции!)

Класове – Сравнение със C++

Разлики:

C++	Java
Дефиниция на клас **class MyClass {** **//** **};**	**Дефиниция на клас** **public class MyClass {** **//** **}**
Създаване на стеков обект: **MyClass obj(…);**	**Създаване на стеков обект:** **----------**
Създаване на динамичен обект: **MyClass* pMyObj =new MyClass (…);** **MyClass& oMyObj = *new MyClass (…);**	**Създаване на динамичен обект:** **MyClass rMyObj =new MyClass (…);**

Наследяване

Същност и особености:

Java използва подразбираща се класова йерархия, при която всички обекти в основата си са наследници на класа "корен" **Object**.

В C++, наследяването може да стартира от произволен базов клас (гора с различни корени);

Наследяване

Синтактични разлики:

- Ключовата дума **extends** за да се определи наследяване от базов клас;
- Ключовата дума **super** за определяне на методи, които се извикват от базовия клас, които имат предекларация (същата спецификация) в наследника. Това може да се прави само за достъп до методи, разположени с едно ниво по ниско в йерархията (C++);
- Конструкторът на базовия клас също може да се извика с ключова дума **super**.

Класове - наследяване

Пример Потребителска класова йерархия: CPerson, CStudent. Нека да създадем наследник на класа с нов атрибут (поле) за факултетен номер.

```
package tu_varna;
public class Student extends Person implements Comparable {
public String strFacNumer="N/A";
    public Student(String strFacNumer) {
        super();
        this.strFacNumer = strFacNumer;
    }
    public Student() {
        super();
    }
    public Student(String name, String gender, String title, String strFN) {
        super(name, gender, title);
        strFacNumer=strFN;
    }
    public String getStrFacNumer() {
        return strFacNumer;
    }
    public void setStrFacNumer(String strFacNumer) {
        this.strFacNumer = strFacNumer;
        super.setLastDate();
    }
}
```

Класове - наследяване

За да се преобразува референция към обект от базов клас към референция към клас наследник се използва подобно типово преобразуване като в C++:

derived d = (derived)base;

За примера:

Student oStudent = (Student) oPerson;

Невъзможност за преобразуване довежда до изключение - ClassCastException

Наследяване

Пример :

```
public class Derivative extends Base {
   public Derivative(String msg) {
      super(msg); // извиква конструктор на
    //базовия клас  Base
   }
   public void baz(int i) {
// предефиниране  на съществуващ метод на
// базовия клас
    super.baz(i); // извиква метода в базовия  Base
   }
}
```

Наследяване

- 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.

Наследяване

- Пример:

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:

import bg.tu-varna.*

- Java не решава общия проблем на колизиите между имена.

Наследяване

Особености на наследяването:

- Наследяването в Java не променя нивото на достъп до членовете на базовия клас. Не може да се специфицира тип (ниво) на наследяване **public, private или protected,** както се прави в C++;
- Предефинираните методи в наследника не могат да редуцират нивата на достъп в базовия клас. Например, ако метод е **public** в базовия клас и го предефинира наследника, предефинирания трябва да бъде също **public.** Това ще се провери от компилатора.

interface *– интерфейси*

- Java предоставя нова ключова дума интерфейс (**interface**), която създава еквивалент на абстрактен базов клас, в който има само абстрактни методи, без полета. В Java има **недвусмислена** разлика между клас, дефиниран като съвкупност от абстрактни функции и промяна на съществуваща функционалност чрез наследяване, която се декларира с ключова дума **extends**.

interface – интерфейси

Заедно с интерфейсите езикът не се нуждае от механизъм като виртуалните базови класове на C++. Пример за създаване на интерфейс, който може да се инстанциира:

```
public interface Face {
    public void smile(); // абстрактен метод
}
public class Student extends Person implements Face {
    public void smile( ) {
        System.out.println("Interface smile");
    }
}
```

interface – интерфейси

- **Имплементиране на сравнение за подредба**:

  В дефиницията на класа, който ще се съхранява **в сортирана колекция** трябва да се обяви интерфейса, с който да се сравняват обектите при подредбата им в контейнера. Това става с обявяването на интерфейса **Comparable:**

public class Student extends Person implements Comparable

Функцията от този интерфейс, която трябва да се предефинира е **compareTo: compareTo е с резултат int връща :**

- **-1 (по-малко);**
- **1 (по-голямо);**
- **0 (равно).**

interface – интерфейси

Пример за предефиниране за сравняване на студенти по факултетен номер в нарастващ и намаляващ ред. Използва се сравнението на тип String, което е организирано на същия принцип. Добавят се методи за сравнението:

Примери.

```
public int compareTo(Object obj) {
  return (strFacNumer.compareTo(((Student)obj).strFacNumer));
}
```

За подредба в нарастващ ред може да се използва обратното сравнение:

```
public int compareTo(Object obj) {
  return (((Student)obj).strFacNumer.compareTo(strFacNumer));
}
```

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;

Разглеждат се по-подробно в следващата лекция!

Операции на езика

- **В Java е добавена операция за изместване надясно** с три символа >:
  - за извършване на логическо изместване на битове надясно **>>>** с вмъкване на нула накрая;
  - операцията (позната от C++) **>>** запазва знаковия бит при изместването (т.е. Реализира "аритметично" изместване **– цялочислено деление на 2**).

Операции на езика

- **В Java е премахнат операторът за обхват operator ::** .

  - За тази цел в Java се използва операторът точка . , което следва от факта, че всичко се дефинира в класа и няма декларация и дефиниция, като в C++. Другата причина, за което е нужен този оператор е при обръщение към статичните методи. Вместо използвания в C++ оператор за обхват, отново се използва операторът точка, например:

ClassName.methodName( );

независимо от това, че методът е статичен.

Операции на езика

- Имената на пакетите (package), заместващи именните пространства, също се идентифицират с точка, например:

```
import java.awt.*;
```

Операции на езика

- **Оператор instanceof**

Дефиниция: *Бинарен оператор, който връща true ако обектът от лявата част е инстанция на клас (или имплементация на интерфейс) специфициран в дясната част. Операторът връща false :*

- *Ако обектът не е инстанция на специфицирания в дясната част;*
- *Ако не имплементира специфицирания интерфейс;*
- *Ако обектът е нулева референция null.*

Операции на езика

- Използване на **instanceof:**

Операторът се използва за определяне на типа на обекта по време на изпълнение. С него се определя дали даден обект (лява страна) е от типа на указания аргумент на оператора (дясна страна), както и негов базов клас.

- Определя дали обект (лява страна) може да се присвои на променлива от типа на аргумента (дясна страна);
- Аргументи на оператора могат да бъдат референтен тип (клас, интерфейс или масив);
- Резултатът от изпълнението е логически съвместимост на типовете.

Операции на езика

Примери:

```
 Boolean bTypeRes;
    String str = "foo";
    bTypeRes = ( str instanceof String );   // true-str е от тип  String
    bTypeRes = ( str instanceof Object );
// true, String има базов тип Object
    bTypeRes = ( str instanceof Date );
// несъвместими типове str и Date
    bTypeRes =  ( foo instanceof byte[] );
// когато стойността на обекта е null, той не може да се определи
    като тип на обекта:
String strNull = null;
    if ( strNull instanceof String )
//  false, null  не е инстанция на никакъв тип.
```

Инициализация на променливите

Особености при инициализация на членовете (полета):

- Паметите за съхраняване на членове на клас (полета) винаги се инициализират с null. Инициализацията на прости типове като членове на клас е гарантирано в Java. Ако програмистът не инициализира дадена променлива експлицитно, тя получава подразбираща се стойност (нула или еквивалентна на нула).
- За разлика от C++, член променливата (поле) на класа може да се инициализира направо при обявяването, както и в конструктора;
- Синтаксисът за инициализация е по-логичен от този на C++ и може да се прилага както за статични, така и за не статични членове на класа;
- Не е необходимо да се дефинира отделно статичната член променлива извън декларацията на класа, както е в C++.

Предварителни декларации на променливи

- В Java не се налагат предварителни декларации на данните-прости или референтни.
- Ако трябва да се използва клас или метод преди да се дефинира, трябва само да се използва. Компилаторът предполага че дефиницията е налична. Затова не са нужни предварителни декларации, както е в C++.

ВЪПРОСИ?