Тема 9. (продължение) Параметризирани типове. Параметризирани колекции

Съдържание

- Параметризирани класове (Generics)
  - Същност;
  - Създаване на параметризиран клас-с един и два шаблонни параметъра;
  - Използване на прости типове в колекциите;
- Параметризирани колекции;

Параметризирани класове (Generics)

Дефиниция:

- Терминът generics означава шаблонен или параметризиран тип. Въведен е от Java 2 V5.0
- T е типизиран параметър, който се замества с реален тип;
- T е името на типа на параметъра;
- Това наименование на параметъра се използва като променлива за получаване на стойност от фактическия тип, която ще се използва в класа като тип, когато се създава обект от параметризирания клас.

Параметризирани класове (Generics)

- Пример за преобразуване на клас. Постановка:
  - Да се състави клас GenericClass, частно поле-референция към базовия клас **Object;**
  - Да се осигури публичен конструктор;
  - Да се осигури публичен метод за достъп (getter) до частното поле;
  - Да се осигури извеждане на конзолния изход на името на типа данни, чиято референция се съхранява в обекта от класа.

Параметризирани класове (Generics)

- Отправна точка. Създаване на класа:

```
class GenericClass {
  private Object ob;
  public GenericClass(Object o) {
    ob = o;
  }
 public Object getob() {
    return ob;
  }
  public void showType() {
   System.out.println("Type of field is " +
    ob.getClass().getName());
  }
}
```

Параметризирани класове (Generics)

- Преобразуване в шаблонен клас:

```
class GenericClass<T> {
  private T ob; // като поле
  public GenericClass(T o) { // като формален параметър
    ob = o;
  }
  public T getob() { // като връщана стойност
    return ob;
  }
  public void showType() { // тестово извеждане на типа
    System.out.println("Type of T is " +
    ob.getClass().getName());
  }
}
```

Параметризирани класове (Generics)

Предимства на параметризираните типове:

```
public class MainClass {
  public static void main(String args[]) {
    // Създаване на референция на GenericClass за Integer.
    GenericClass<Integer> iOb = new GenericClass<Integer>(88);
    iOb.showType();
    // Не се изисква преобразуване към прост тип.
    int ival = iOb.getob();
    System.out.println("value: " + ival);
    //  Създаване на референция на GenericClass за  Strings.
    GenericClass<String> strOb = new GenericClass<String>("Gen Test");
    strOb.showType();
    String str = strOb.getob();
    System.out.println("value: " + str);
  }
}
```

Параметризирани класове (Generics)

Резултат:

Type of T is **java.lang.Integer** value: 88
Type of T is **java.lang.String** value: Gen Test

Параметризирани класове. Особености

За разлика от езика C++ в Java не могат да се използват като параметри на шаблона всички типове данни.

- **Използват се само с обекти – не допускат прости типове данни**!

Пример:

```
Gen<int> strOb = new Gen<int>(53);
// Предизвиква грешка, защото се
// използва прост тип
```

Параметризирани класове. Особености

- **Прости типове данни като шаблонни 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.**

Параметризирани класове. Особености

- **Прости типове данни и съхраняването им в колекции без шаблони:**

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

Контейнерни параметризирани типове. Особености

Съществуващите в пакета java.util.* параметризирани типове са колекциите на езика.

- Могат да приемат параметри;
- Параметризирани типове са синоними на общите (generic) типове на езика Java;
- В скобите могат да се поставят само референтни типове;

Параметризирани класове. Особености

- **Използване на повече от един параметър на типа. Пример:**

```
class TwoGen<T, V> {
      private T ob1;
      private V ob2;
      public TwoGen(T o1, V o2) {
        ob1 = o1;
        ob2 = o2;
      }
     // метод за извеждане на имената на типовете
      public void showTypes() {
        System.out.println("Type of T is " + ob1.getClass().getName());
        System.out.println("Type of V is " + ob2.getClass().getName());
      }

      public T getob1() {
        return ob1;
      }

      public V getob2() {
        return ob2;
      }
}
```

Параметризирани класове. Особености

- Използване на повече от един параметър на типа. Пример за тестов клас:

```
public class MainClass {
  public static void main(String args[]) {
    TwoGen<Integer, String> tgObj = new TwoGen<Integer,
    String>(88, "Generics");
    tgObj.showTypes();

    int v = tgObj.getob1();
    System.out.println("value: " + v);

    String str = tgObj.getob2();
    System.out.println("value: " + str);
  }
}
```

Параметризирани класове. Особености

- Резултат :

Type of T is java.lang.Integer

Type of V is java.lang.String

value: 88

value: Generics

Контейнерни параметризирани типове. Особености

Формат:

List<E> myList;

Където E е променливата на типа, т.е. променливата, която се замества с конкретния тип. Особености:

- Типът, който използва типовата променлива E позволява да се постави E когато се декларира или се създава инстанция на параметризирания тип;
- Ако E е клас, може да се предава наследник на класа E;
- Ако E е интерфейс, може да се постави вместо него клас, който имплементира интерфейса E;
- За съвместимост при дефиниране на шаблонните класове за име на типовата променлива се използва единична главна буква;

Контейнерни параметризирани типове. Особености

Пример

```
import java.util.ArrayList;
import java.util.List;
public class MainClass {
    public static void main(String[] args) {
   // основен вариант
   List stringList1 = new ArrayList();

      stringList1.add("Java 5");
      stringList1.add("without generics");
// изисква типово преобразуване
      String s1 = (String) stringList1.get(0);
      System.out.println(s1.toUpperCase());//JAVA 5
```

Контейнерни параметризирани типове. Особености

Пример

```
    // параметризиран вариант
       List<String> stringList2 = new ArrayList<String>();
       stringList2.add("Java 5");
       stringList2.add("with generics");
     // не изисква типово преобразуване
       String s2 = stringList2.get(0);
       System.out.println(s2.toUpperCase()); //JAVA 5
      }
}
```

Вложени контейнерни параметризирани типове

- Параметризираните типове са типове на езика Java и могат да се влагат като типови променливи. Например, може да се декларира списък за съхраняване на списък от стрингове:

```
List<List<String>> myListOfListsOfStrings;
```

За да се получи достъп до първия стринг на първия списък от обекта myListOfListsOfStrings се използва:

```
String s = myListOfListsOfStrings.get(0).get(0);
```

Вложени контейнерни параметризирани типове

Пример за използване:

```
import java.util.ArrayList;
import java.util.List;
public class MainClass {
 public static void main(String[] args) {
   List<String> listOfStrings = new ArrayList<String>();
   listOfStrings.add("Hello again");
   List<List<String>> listOfLists = new ArrayList<List<String>
    >();
   listOfLists.add(listOfStrings);
   String s = listOfLists.get(0).get(0);
   System.out.println(s); // изход "Hello again"
 }
}
```

Контейнерни параметризирани типове с повече параметри

Пример е интерфейса Map и класа HashMap:

```
import java.util.HashMap;
import java.util.Map;

public class MainClass {
    public static void main (String[] args) {
        Map<String, String> map = new HashMap<String,
    String>();
        map.put ("key1", "value1");
        map.put ("key2", "value2");
        String value1 = map.get("key1");
    }
}
```

Непроверени типове. Сигурност на кода

За да се осигури преминаването към параметризирани типове и съвместимостта с предходните версии Java позволява да се пропуснат параметрите на типа. Това създава непроверена референция към типа:

Пример:

```
class Gen<T> {
  T ob;

  Gen(T o) {
    ob = o;
  }

  T getob() {
    return ob;
  }
```

(продължение)

Непроверени типове. Сигурност на кода

```
public static void main(String args[]) {
 Gen<Integer> iObject = new Gen<Integer>(88);
 Gen<String> strObject = new Gen<String>("Generics Test");
 String str = strObject.getob();  // шаблонен
 Gen raw = new Gen(new Double(98.6)); // непроверено по тип
  конструиране на обект

 // Необходимо е преобразуване към (Double) поради неизвестния тип
 double d = (Double) raw.getob();
 System.out.println("value: " + d);

 strObject = raw; // потенциална грешка (предупреждение)

// припокриване на типа
raw = iObject; // потенциална грешка
d = (Double) raw.getob();
  }
}
```

Непроверени типове. Сигурност на кода

Резултат от примерната програма:

value: 98.6

Exception in thread "main" java.lang.ClassCastException:
java.lang.Double cannot be cast to java.lang.String
at Gen.main(Gen.java:23)

Примерно използване на имплементациите и техните интерфейси

Принципи за използване на интерфейсите при създаване на обекти от класовете:

Използване на интерфейси като полета в класа:

- Лесна поддръжка на типовете като обекти;
- Използване на интерфейси при предаване на параметри и връщане на резултати от методите;
- Използване на възможно най-базовия интерфейс при възможни алтернативи:
  - Например:
    - **Collection**
    - **List**

Примерно използване на имплементациите и техните интерфейси

Примери за използване на интерфейсите при създаване на обекти от класовете:

```
package tu_varna;
import java.io.*;
import java.util.*;
public class CollectionSamples {
//public Collection oColl = new ArrayList(); // Създаване на колекция от ArrayList
//public Collection oColl = new Vector(); //Създаване на колекция от Vector
//public Collection oColl = new Stack();  // Създаване на колекция от Stack
// public Collection oColl = new LinkedList();//Създаване на колекция от LinkedList

//   public List oColl = new ArrayList();  // Създаване на списък от класа  ArrayList
//   public List oColl = new Vector(); // Създаване на списък от класа       Vector
//   public List oColl = new Stack(); // Създаване на списък от класа        Stack
//   public List oColl = new LinkedList(); //Създаване на списък от класа LinkedList

//   public Set oColl = new HashSet(); //Създаване на множество от клас HashSet
```

Примерно използване на имплементациите и техните интерфейси

```
public Set oColl = new TreeSet();  // Създаване на множество от клас  TreeSet
public CollectionSamples(String fileName ) {
 try { // използване на Scanner с файлов поток
   Scanner sc = new Scanner(new File(fileName));
   while( sc.hasNextLine() )  {
     Object oPersonExpl= new Student(sc.next(), sc.next(), sc.next(), sc.next());
     oColl.add(oPersonExpl);
  }
  sc.close();
} catch (FileNotFoundException e){
  System.out.println("File Not Found ...");
 }
}
```

Примерно използване на имплементациите и техните интерфейси

```
// извеждане  на конзолен изход с итератор
public void printColl(Collection c) {
    for(Iterator It = c.iterator(); It.hasNext();)
       System.out.println(It.next().toString());
  }

// стрингова интерпретация
 public String toString() {
     return oColl.toString();
 }
// главна функция
 public static void main(String[] args) {
    CollectionSamples obj = new CollectionSamples("Input.txt");
    obj.printColl(obj.oColl);
   }
}
```

Въпроси?