ГЕНЕТИЧНО ПРОГРАМИРАНЕ

ЛЕКЦИОНЕН КУРС "ПРОГРАМИРАНЕ НА JAVA"

Distributed @learning center

СТРУКТУРА НА ЛЕКЦИЯТА

- Въведение
- Генетични класове
- Параметри с ограничени типове
- Генетични методи
- Маски
- Примери

СЦЕНАРИЙ

- Да разгледаме следния сценарий:
  - Искаме да се разработи контейнер, който ще бъде използван за да приемане на обектите в приложения
  - Типът на обектите не винаги ще бъде същият за различните приложения
  - Затова е необходимо да се разработи един контейнер, който има способността да съхранява обекти от различен тип
- За сценария, най-очевидният начин за постигане на целта ще бъде да се разработи контейнер, който има способността да съхраняват и извличат обекти от тип Object, а след това да ги преобразува в различни типове

ПРИМЕР

```
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public final class OldStyleList   {
   public static void main(String[] args)   {
      final List personList = new ArrayList();
       personList.add(new Person("Иван", new Date(), "Пловдив"));
       personList.add(new Person("Мария", new Date(), "Пловдив"));
       personList.add(new Dog("Sarah from Plovdiv"));
       for (int i = 0; i < personList.size(); i++)   {
          final Person person = (Person) personList.get(i);
          System.out.println(person.getName() + " от " + person.getCity());
       }
   }
   private OldStyleList()   {   }
}

class Dog   {
   private final String name;
   public Dog(final String name)   {  this.name = name;   }

   @Override
   public String toString()   {  return "Dog [name='" + name + "']";   }
}
```

```
Иван от Пловдив
Мария от Пловдив
Exception in thread "main" java.lang.ClassCastException: Dog cannot be cast to
Person at OldStyleList.main(OldStyleList.java:27)
```

ПРИМЕР

```
public class ObjectContainer {
    private Object obj;

    /**
     * @return the obj
     */
    public Object getObj() {
        return obj;
    }

    /**
     * @param obj the obj to set
     */
    public void setObj(Object obj) {
        this.obj = obj;
    }

}
```

- Въпреки, че контейнерът ще постигне желания резултат, няма да е най-подходящото решение за нашата цел
- Понеже има потенциала да предизвика изключения надолу по пътя
  - ✓ Не е сигурен към типовете
  - ✓ Изисква използвате явно преобразуване всеки път, когато се търси обекта

ИЗПОЛЗВАНЕ

```
ObjectContainer myObj = new ObjectContainer();

1 myObj.setObj("Test");
  System.out.println("Value of myObj:" + myObj.getObj());

2 myObj.setObj(3);
  System.out.println("Value of myObj:" + myObj.getObj());

  List objectList = new ArrayList();
  objectList.add(myObj);

3 String myStr = (String) ((ObjectContainer)objectList.get(0)).getObj();
  System.out.println("myStr: " + myStr);
```

Съхраняване на низ

Съхраняване на цяло число

Трябва да направим преобразуване към коректния тип за да избегнем ClassCast Exception

ПО-ДОБРО РЕШЕНИЕ

- Generics могат да се използват за разработване на по-добро решение
  - Като се използва контейнер, който може да има определен тип (генетичен тип), определен при инициализацията
  - Позволява създаването на инстанции на контейнера, които могат да се използват за съхранение на обекти от определени типове
- Един генетичен тип е клас или интерфейс, който може да се параметризира спрямо типа
  - Което означава, че актуалният тип може да бъде определен чрез заместване на генетичния тип с конкретен тип
- Актуалният тип ще ограничава стойностите, които ще се използват в рамките на контейнера
  - Премахва се изискването за преобразуване
  - Осигурява се по-силна проверка на типовете по време на компилация

ПРИМЕР

```
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public final class NewStyleList  {
   public static void main(String[] args)   {
      final List<Person> personList = new ArrayList<Person>();
      personList.add(new Person("Иван", new Date(), "Пловдив"));
      personList.add(new Person("Мария", new Date(), "Пловдив"));
      // personList.add(new Dog("Sarah from Plovdiv"));  // Compile-Error

      for (int i = 0; i < personList.size(); i++)   {
         final Person person = personList.get(i);
         System.out.println(person.getName() + " aus " + person.getCity());
      }
   }

   private NewStyleList()  {    }
}
```

```
Иван от Пловдив
Мария от Пловдив
```

ПРИМЕР

```
public class GenericContainer<T> {
   private T obj;

   public GenericContainer() {
   }

   public GenericContainer(T t) {
      obj = t;
   }
   /**
    * @return the obj
    */
   public T getObj() {
      return obj;
   }

   /**
    * @param obj the obj to set
    */
   public void setObj(T t) {
      obj = t;
   }
}
```

- Най-съществените различия са, че дефиницията на класа съдържа <T> и полето obj не е вече от тип Object, а по-скоро от генетичния тип T
- Скобите в дефиницията на класа заграждат секцията на тип-параметрите, които ще бъдат използвани в рамките на класа
- T е параметър, свързан с генетичния общ тип, който е дефиниран в този клас

ИЗПОЛЗВАНЕ

```
public class GenericContainer<T>  {
   private T obj;

   public GenericContainer()  {
   }

   public GenericContainer(T t)  {
      obj = t;
   }
   /**
    * @return the obj
    */
   public T getObj()  {
      return obj;
   }

   /**
    * @param obj the obj to set
    */
   public void setObj(T t)  {
      obj = t;
   }
}
```

```
GenericContainer<Integer> myInt =  new GenericContainer<Integer>();

myInt.setObj(3);  // OK
myInt.setObj("Int"); // няма да се компилира
```

- За да използваме генетичния контейнер, трябва да присвоим тип на контейнера чрез спецификация, дадена в скобите
- В примера, инициализираме контейнера за цели числа
- При записване на низ компилаторът ще възрази

ПОЛЗИ ОТ GENERICS

- Някои от основните ползи от използване на генетични типове
  - По-строги проверки на типовете е един от най-важните, защото това спестява време, като се предотвратява възникването на ClassCastExceptions, които могат да бъдат изхвърлени по време на изпълнение
  - Премахване на преобразуването, което означава използване по-малко код, тъй като компилаторът знае точно какъв тип ще се съхраняват в една колекция
  - Могат да се разработят генерични алгоритми, които могат да бъдат персонализирани за решаване на конкретната задача

ПРИМЕР

```
List myObjList = new ArrayList();

for(int x=0; x <=10; x++) {
   ObjectContainer myObj = new ObjectContainer();
1  myObj.setObj("Test" + x);
   myObjList.add(myObj);
}

for(int x=0; x <= myObjList.size()-1; x++) {
   ObjectContainer obj = (ObjectContainer) myObjList.get(x);
2  System.out.println("Object Value: " + obj.getObj());
}

List<GenericContainer> genericList = new ArrayList<GenericContainer>();

for(int x=0; x <=10; x++) {
   GenericContainer<String> myGeneric = new GenericContainer<String>();
3  myGeneric.setObj(" Generic Test" + x);
   genericList.add(myGeneric);
}

for(GenericContainer<String> obj:genericList) {
   String objectString = obj.getObj();
4  System.out.println(objectString);
}
```

Съхраняване инстанции на ObjectContainer

За получаване на обекти е необходимо преобразуване

Съхраняване инстанции на GenericContainer

За получаване на обекти не е необходимо преобразуване

ПРИМЕР

```
List myObjList =

for(int x=0; x <=
    ObjectContainer myObj = new ObjectContainer();
    myObj.setObj("Test" + x);
    myObjList.add(myObj);
}

for(int x=0; x <=
   ObjectContaine
   System.out.prir
}

List<GenericCont

for(int x=0; x <=1
    GenericContainer<String> myGeneric = new GenericContainer<String>();
    myGeneric.setObj(" Generic Test" + x);
    genericList.add(myGeneric);
}

for(GenericContainer<String> obj:genericList) {
    String objectString = obj.getObj();
    System.out.println(objectString);
}
```

Разлики между съхраняване на обекти в Object контейнер и съхраняване в GenericContainer

- При използване на ArrayList, ние сме в състояние да определим вида на колекцията при създаването ѝ, като се използва означението в скобите (<GenericContainer>)
- Колекцията ще бъде в състояние да съхранява само GenericContainer обекти (или на подкласове на GenericContainer), като не е необходимо преобразуване при извличане на обекти от колекцията

ИЗПОЛЗВАНЕ НА GENERICS

- Съществуват различни възможности за използване на generics
- В примера беше разгледан случай за генериране на генетични типове
  - Това е една добра отправна точка за изучаване синтаксиса на generics на ниво класове и интерфейси
  - Сигнатурата клас съдържа раздел за тип-параметри, зададен ъглови скоби (<>) след името на класа
  - Напр., public class GenericContainer<T> { ...

ТИПОВЕ ПРОМЕНЛИВИ

- Тип-параметрите (наричат се също типови променливи) се използват като пазители на места, които индикират, че даден тип ще бъде назначен в runtime
- Може да има един или повече тип-параметри, които могат да бъдат използвани в рамките на един клас при необходимост
- Според използваната конвенция, тип-параметрите се означават с една единствена главна буква, която показва типа на дефинирания параметър
- Стандартни означения:
  - E: елемент
  - K: ключ
  - N: число
  - T: тип
  - V: стойност
  - S, U, V, … : втори, трети, четвърти параметър в един списък

ПОВЕЧЕ ГЕНЕТИЧНИ ТИПОВЕ

- В определени случаи е полезно да има възможност да се използват повече от един генетичен тип в един клас или интерфейс
- Повечето тип-параметрите могат да се използват в клас или интерфейс чрез поставяне разделен със запетаи списък на типове между ъглови скоби

ПРИМЕР

```
public class MultiGenericContainer<T, S> {
    private T firstPosition;
    private S secondPosition;

    public MultiGenericContainer(T firstPosition, S secondPosition){
        this.firstPosition = firstPosition;
        this.secondPosition = secondPosition;
    }

    public T getFirstPosition(){
        return firstPosition;
    }

    public void setFirstPosition(T firstPosition){
        this.firstPosition = firstPosition;
    }

    public S getSecondPosition(){
        return secondPosition;
    }

    public void setSecondPosition(S secondPosition){
        this.secondPosition = secondPosition;
    }

}
```

- MultiGenericContainer съхранява два типа обекти, всеки от тях се определя при инициализацията
- Означенията – в съответствие с конвенцията

ИЗПОЛЗВАНЕ

```
public class MultiGenericContainer<T, S> {
    private T firstPosition;
    private S secondPosition;

    public MultiGenericContainer(T firstPosition, S secondPosition){
        this.firstPosition = firstPosition;
        this.secondPosition = secondPosition;
    }

    public T getFirstPosition(){
        return firstPosition;
    }

    public void setFirstPosition(T firstPosition){
        this.firstPosition = firstPosition;
    }

    public S getSecondPosition(){
        return secondPosition;
    }

    public void setSecondPosition(S secondPosition){
        this.secondPosition = secondPosition;
    }

}
```

```
MultiGenericContainer<String, String> mondayWeather =
        new MultiGenericContainer<String, String>("Monday", "Sunny");
MultiGenericContainer<Integer, Double> dayOfWeekDegrees =
        new MultiGenericContainer<Integer, Double>(1, 78.0);

String mondayForecast = mondayWeather.getFirstPosition();

double sundayDegrees = dayOfWeekDegrees.getSecondPosition();
```

UNBOXING & AUTOBOXING

1 Коментар?

- Не е възможно да се използват примитивни типове в generetics
- Могат да се използват само референцирани типове
- Autoboxing и Unboxing дава възможност да се съхраняват и извличат стойности до и от примитиви при работа с генетични обекти

```
MultiGenericContainer<String, String> mondayWeather =
      new MultiGenericContainer<String, String>("Monday", "Sunny");
MultiGenericContainer<Integer, Double> dayOfWeekDegrees =
      new MultiGenericContainer<Integer, Double>(1, 78.0);

String mondayForecast = mondayWeather.getFirstPosition();

double sundayDegrees = dayOfWeekDegrees.getSecondPosition();
```

autoboxing

unboxing

ОПЕРАТОР „ДИАМАНТ“

1 Коментар?

- Тип-интерфейс: способност на Java компилатора да определи автоматично тип-параметрите, позовавайки се на декларацията и извикването на метода
- Вместо да декларираме типовете повторно можем да използваме оператора “диамант”

```
MultiGenericContainer<String, String> mondayWeather =
      new MultiGenericContainer<>("Monday", "Sunny");
MultiGenericContainer<Integer, Double> dayOfWeekDegrees =
      new MultiGenericContainer<>(1, 78.0);
```

ОГРАНИЧЕНИ ТИПОВЕ

- В определени случаи, посочваме общия тип, но искаме да контролирате типове, които могат да бъдат специфицирани, вместо достъпът да бъде широко отворен
- Ограничени типове
  - Използват се за ограничаване на границите на генетичния тип
  - Използват се ключовите думи extends и super
  - Напр.:
    - <T extends UpperBoundType>
    - <T super LowerBoundType>

ПРИМЕР

```
public class GenericNumberContainer <T extends Number> {
  private T obj;

  public GenericNumberContainer(){
  }

  public GenericNumberContainer(T t){
     obj = t;
  }
  /**
   * @return the obj
   */
  public T getObj() {
     return obj;
  }

  /**
   * @param obj the obj to set
   */
  public void setObj(T t) {
     obj = t;
  }
}
```

GenericNumberContainer специфицира, че генетичният тип наследява Number

ИЗПОЛЗВАНЕ

1 Коментар?

```
public class GenericNumberContainer <T extends Number> {
  private T obj;

  public GenericNumberContainer(){
  }

  public GenericNumberContainer(T t){
     obj = t;
  }
  /**
   * @retu
   */
  public T
     return
  }

  /**
   * @param obj the obj to set
   */
  public void setObj(T t) {
     obj = t;
  }
}
```

```
GenericNumberContainer<Integer> gn = new GenericNumberContainer<Integer>();
gn.setObj(3);

GenericNumberContainer<String> gn2 = new GenericNumberContainer<String>();
```

ГЕНЕТИЧНИ МЕТОДИ

- Възможно е, да не знаем типа на един аргумент, който се предава на метод
- Generics могат да се използват на ниво методи за решаване на такива ситуации
- Методите могат да съдържат:
  - Тип-аргументи
  - Тип-резултати (връщани стойности)

ПРИМЕР

```
public class Calculator {

    public static Integer addInteger(Integer a, Integer b)  {
        return a + b;
    }

    public static Float addFloat(Float a, Float b)  {
        return a + b;
    }

    public static <N extends Number> double add(N a, N b)  {
        double sum = 0;
        sum = a.doubleValue() + b.doubleValue();
        return sum;
    }
 }
```

Не-генетични методи

Генетичен метод

ИЗПОЛЗВАНЕ

```
public class CalculatorExample {
  public static void main(String[] args) {

      float floatValue = Calculator.addFloat(2f, 3f);
      System.out.println("Float Value: " + floatValue);

      int intValue = Calculator.addInteger(3, 4);
      System.out.println("Integer Value: " + intValue);

      double genericValue1 = Calculator.add(3, 3f);
      System.out.println("The int + float result: " + genericValue1);

      double genericValue2 = Calculator.add(7.54, 174);
      System.out.println("The double + int result: " + genericValue2);

      // Causes a ClassCastException because String is not a subtype of java.lang.Number
      //double genericValue3 = Calculator.add("Not valid", 3f);
      //System.out.println("The invalid result: " + genericValue3);
  }
}
```

```
Float Value: 5.0
Integer Value: 7
The int + float result: 6.0
The double + int result: 181.54
```

МАСКИ

- В някои случаи е полезно да се пише код, който специфицира неизвестни типове
- Въпросителният знак “?” (символ за маска) може да се използва за представяне от неизвестен тип в генетичен код
- Маски могат да бъдат използвани с:
  - Параметри
  - Полета
  - Локални променливи
  - Типове на резултати

ПРИМЕР

```
import java.util.ArrayList;
import java.util.List;
public class WildcardExample  {
   public static void main(String[] args)  {
      List<Integer> intList = new ArrayList<Integer>();
      intList.add(2);
      intList.add(4);
      intList.add(6);

      List<String> strList = new ArrayList<String>();
      strList.add("two");
      strList.add("four");
      strList.add("six");

      List<Object> objList = new ArrayList<Object>();
      objList.add("two");
      objList.add("four");
      objList.add(strList);

      printList(intList);
      printList(strList);
      printList(objList);

      checkList(intList, 3);
      checkList(objList, strList);
      checkList(strList, objList);
      checkNumber(intList, 3);
}
```

ПРИМЕР

```
public static <T> void printList(List<T> myList) {
    for(Object e:myList) {
        System.out.println(e);
    }
}

public static <T> void checkList(List<?> myList, T obj) {
    if(myList.contains(obj)) {
        System.out.println("The list " + myList + " contains the element: " + obj);
    } else {
        System.out.println("The list " + myList + " does not contain the element: " + obj);
    }
}

public static <T> void checkNumber(List<? extends Number> myList, T obj) {
    if(myList.contains(obj)) {
        System.out.println("The list " + myList + " contains the element: " + obj);
    } else {
        System.out.println("The list " + myList + " does not contain the element: " + obj);
    }
}
}
```

ПРИМЕР

```
public static <T> void printList(List<T> myList){
    for(Object e:myList){
        System.out.println(e);
    }
}

public static <T> void checkList(L
    if(myList.contains(obj)){
        System.out.println("The list " +
    } else {
        System.out.println("The list " +
    }
}

public static <T> void checkNuml
    if(myList.contains(obj)){
        System.out.println("The list " +
    } else {
        System.out.println("The list " + myList + " does not contain the element: " + obj);
    }
}
}
```

```
2
4
6
two
four
six
two
four
[two, four, six]
The list [2, 4, 6] does not contain the element: 3
The list [two, four, [two, four, six]] contains the element: [two, four, six]
The list [two, four, six] does not contain the element: [two, four, [two, four, six]]
The list [2, 4, 6] does not contain the element: 3
```

ОЩЕ ПРИМЕРИ

```
public class GenericMethodTest
{
  // generic method printArray
  public static < E > void printArray( E[] inputArray )
  {
    // Display array elements
      for ( E element : inputArray ){
        System.out.printf( "%s ", element );
      }
      System.out.println();
  }

  public static void main( String args[] )
  {
    // Create arrays of Integer, Double and Character
    Integer[] intArray = { 1, 2, 3, 4, 5 };
    Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
    Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };

    System.out.println( "Array integerArray contains:" );
    printArray( intArray  ); // pass an Integer array

    System.out.println( "\nArray doubleArray contains:" );
    printArray( doubleArray ); // pass a Double array

    System.out.println( "\nArray characterArray contains:" );
    printArray( charArray ); // pass a Character array
  }
}
```

1 Какъв резултат?

```
Array integerArray contains:
1 2 3 4 5

Array doubleArray contains:
1.1 2.2 3.3 4.4

Array characterArray contains:
H E L L O
```

ОЩЕ ПРИМЕРИ

1 Какъв резултат?

```
public class MaximumTest
{
  // determines the largest of three Comparable objects
  public static <T extends Comparable<T>> T maximum(T x, T y, T z)
  {
    T max = x; // assume x is initially the largest
    if ( y.compareTo( max ) > 0 ){
      max = y; // y is the largest so far
    }
    if ( z.compareTo( max ) > 0 ){
      max = z; // z is the largest now
    }
    return max; // returns the largest object
  }
  public static void main( String args[] )
  {
    System.out.printf( "Max of %d, %d and %d is %d\n\n",
            3, 4, 5, maximum( 3, 4, 5 ) );

    System.out.printf( "Maxm of %.1f,%.1f and %.1f is %.1f\n\n",
            6.6, 8.8, 7.7, maximum( 6.6, 8.8, 7.7 ) );

    System.out.printf( "Max of %s, %s and %s is %s\n","pear",
      "apple", "orange", maximum( "pear", "apple", "orange" ) );
  }
}
```

Max of 3, 4 and 5 is 5

Maxm of 6.6,8.8 and 7.7 is 8.8

Max of pear, apple and orange is pear

ОБОБЩЕНИЕ

- Генетичните методи и генеричните класове в Java позволяват на програмистите да специфицира с една декларация на метод набор от свързани с тях методи или с декларация на един клас
  - Група от свързани типове респективно
- Generics осигуряват също безопасност по отношение на типовете по време на компилация
  - Позволява на програмистите да установяват невалидни видове по време на компилация

БЛАГОДАРЯ ЗА ВНИМАНИЕТО!

КРАЙ “ГЕНЕТИЧНО ПРОГРАМИРАНЕ”

Distributed @learning center