Generičko programiranje u Javi

AUDITORNE VJEŽBE

Sadržaj

Primjer metode koja vraća generički tip podatka

Primjer parametrizirane klase

Primjer korištenje zamjenskih simbola

Pitanja s certifikata

Primjer metode koja vraća generički tip podatka

- Klase koje implementiraju sučelje „Comparable” pružaju mogućnost svojim objektima da se međusobno uspoređuju korištenjem metode „compareTo”
- Ako je potrebno napisati generičku metodu koja može primati samo takve klase, onda je to moguće navesti kod same deklaracije klase korištenjem preduvjeta „extends Comparable”
- Sučelje „Comparable” također prima generički tip pa je u tom slučaju moguće koristiti i isti tip T:

```
public static <T extends Comparable<T>> T maximum(T x, T y, T z)
```

Primjer metode koja vraća generički tip podatka

```
public static <T extends Comparable<T>> T maximum(T x, T y, T z)
   {
      T max = x;

      if (y.compareTo(max) > 0)
         max = y;

      if (z.compareTo(max) > 0)
         max = z;

      return max;
   }
```

Primjer metode koja vraća generički tip podatka

```
public static void main(String[] args)
   {
      System.out.printf("Maximum of %d, %d and %d is %d%n%n", 3, 4, 5,
         maximum(3, 4, 5));
      System.out.printf("Maximum 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("Maximum of %s, %s and %s is %s%n", "pear",
         "apple", "orange", maximum("pear", "apple", "orange"));
   }
```

```
Maximum of 3, 4 and 5 is 5

Maximum of 6.6, 8.8 and 7.7 is 8.8

Maximum of pear, apple and orange is pear
```

Primjer parametrizirane klase

```
public class Stack<T>
{
    private List<T> elements;

    public Stack() {
        this(10);
    }

    public Stack(int capacity)
    {
        int initCapacity = capacity > 0 ? capacity : 10;
        elements = new ArrayList<T>(initCapacity);
    }
```

Primjer parametrizirane klase

```
    public void push(T pushValue) {
        elements.add(pushValue);
    }

    public T pop()
    {
        if (elements.isEmpty()) {
            throw new EmptyStackException("Stack is empty, cannot pop");
        }
        return elements.remove(elements.size() - 1);
    }
}
```

Primjer parametrizirane klase

```
public static void main(String[] args) {
      double[] doubleElements = {1.1, 2.2, 3.3, 4.4, 5.5};

      Stack<Double> doubleStack = new Stack<>(5);

      testPushDouble(doubleStack, doubleElements);
      testPopDouble(doubleStack);
}

private static void testPushDouble(Stack<Double> stack, double[] values) {
      System.out.printf("%nPushing elements onto doubleStack%n");
      for (double value : values) {
         System.out.printf("%.1f ", value);
         stack.push(value);
      }
   }
```

Primjer parametrizirane klase

```
private static void testPopDouble(Stack<Double> stack) {
   try {
         System.out.printf("%nPopping elements from doubleStack%n");
         double popValue;
         while (true) {
            popValue = stack.pop();
            System.out.printf("%.1f ", popValue);
         }
      }
      catch(EmptyStackException emptyStackException)
      {
         System.err.println();
         emptyStackException.printStackTrace();
      }
   }
```

Primjer korištenja zamjenskih simbola

```
public static void main(String[] args)
   {
      Integer[] integers = {1, 2, 3, 4, 5};
      List<Integer> integerList = new ArrayList<>();

     for (Integer element : integers)
         integerList.add(element);

      System.out.printf("integerList contains: %s%n", integerList);
      System.out.printf("Total of the elements in integerList: %.0f%n%n",
         sum(integerList));
```

Primjer korištenja zamjenskih simbola

```
Double[] doubles = {1.1, 3.3, 5.5};
List<Double> doubleList = new ArrayList<>();

for (Double element : doubles)
   doubleList.add(element);

System.out.printf("doubleList contains: %s%n", doubleList);
System.out.printf("Total of the elements in doubleList: %.1f%n%n",
   sum(doubleList));

Number[] numbers = {1, 2.4, 3, 4.1};
List<Number> numberList = new ArrayList<>();

for (Number element : numbers)
   numberList.add(element);
```

Primjer korištenja zamjenskih simbola

```
        System.out.printf("numberList contains: %s%n", numberList);
        System.out.printf("Total of the elements in numberList: %.1f%n",
            sum(numberList));
    }

    public static double sum(List<? extends Number> list) {
        double total = 0;

        for (Number element : list)
            total += element.doubleValue();

        return total;
    }
```

Pitanja s certifikata (1)

Consider the following program:

```
class Base<T> { }

class Derived<T> { }

class Test {
     public static void main(String []args) {
             // Stmt #1
     }
}
```

Which statements can be placed in the place of //Stmt#1 and the program remains compilable (choose two):

a) `Base<Number> b = new Base<Number>();`

b) `Base<Number> b = new Derived<Number>();`

c) `Base<Number> b = new Derived<Integer>();`

d) `Derived<Number> b = new Derived<Integer>();`

e) `Base<Integer> b = new Derived<Integer>();`

f) `Derived<Integer> b = new Derived<Integer>();`

Pitanja s certifikata (2)

Which statements are true about the following program?

```
public class Q100_82 {
  public static void main(String[] args) {
    Object o = choose(991, "800");                          // (1)
    Number n1 = choose(991, 3.14);                          // (2)
    Number n2 = Q100_82.<Double>choose((double)991, 3.14);  // (3)
    int k = (int) choose(1.3, 3.14);                        // (4)
    int l = (int) (double) choose(1.3, 3.14);               // (5)
  }

  public static <T extends Comparable<T>> T choose(T t1, T t2) {
    return t1.compareTo(t2) >= 0 ? t1 : t2;
  }
}
```

Select the two correct answers.

(a) The class must be declared as a generic type:

```
public class Q100_82<T extends Comparable<T>> { ... }
```

(b) The compiler reports errors in (1).
(c) The compiler reports no errors in (2).
(d) The compiler reports no errors in (3).
(e) The compiler reports no errors in (4).
(f) The compiler reports errors in (5).

Pitanja s certifikata (3)

Which parameter declarations can be inserted at (1) so that the program compiles without warnings?

```
interface Wagger{}
class Pet implements Wagger{}
class Dog extends Pet {}
class Cat extends Pet {}

public class Q100_51 {
  public static void main(String[] args) {
    List<Pet> p = new ArrayList<Pet>();
    List<Dog> d = new ArrayList<Dog>();
    List<Cat> c = new ArrayList<Cat>();
    examine(p);
    examine(d);
    examine(c);
  }

  static void examine(_____ pets) {          // (1)
    System.out.print("Your pets need urgent attention.");
  }
}
```

Select the three correct answers.

(a) `List<? extends Pet>`
(b) `List<? super Pet>`
(c) `List<? extends Wagger>`
(d) `List<? super Wagger>`
(e) `List<?>`
(f) All of the above

Pitanja s certifikata (4)

Given:

```
interface Hungry<E> { void munch(E x); }
interface Carnivore<E extends Animal> extends Hungry<E> {}
interface Herbivore<E extends Plant> extends Hungry<E> {}
abstract class Plant {}
class Grass extends Plant {}
abstract class Animal {}
class Sheep extends Animal implements Herbivore<Sheep> {
  public void munch(Sheep x) {}
}
class Wolf extends Animal implements Carnivore<Sheep> {
  public void munch(Sheep x) {}
}
```

Which of the following changes (taken separately) would allow this code to compile? (Choose all that apply.)

A. Change the `Carnivore` interface to

```
interface Carnivore<E extends Plant> extends Hungry<E> {}
```

B. Change the `Herbivore` interface to

```
interface Herbivore<E extends Animal> extends Hungry<E> {}
```

C. Change the `Sheep` class to

```
class Sheep extends Animal implements Herbivore<Plant> {
  public void munch(Grass x) {}
}
```

D. Change the `Sheep` class to

```
class Sheep extends Plant implements Carnivore<Wolf> {
   public void munch(Wolf x) {}
}
```

E. Change the `Wolf` class to

```
class Wolf extends Animal implements Herbivore<Grass> {
  public void munch(Grass x) {}
}
```

F. No changes are necessary

Pitanja s certifikata (5)

Given a method declared as

```
public static <E extends Number> List<E> process(List<E> nums)
```

A programmer wants to use this method like this:

```
// INSERT DECLARATIONS HERE

output = process(input);
```

Which pairs of declarations could be placed at `// INSERT DECLARATIONS HERE` to allow the code to compile? (Choose all that apply.)

A.
```
ArrayList<Integer> input = null;
ArrayList<Integer> output = null;
```

B.
```
ArrayList<Integer> input = null;
List<Integer> output = null;
```

C.
```
ArrayList<Integer> input = null;
List<Number> output = null;
```

D.
```
List<Number> input = null;
ArrayList<Integer> output = null;
```

E.
```
List<Number> input = null;
List<Number> output = null;
```

F.
```
List<Integer> input = null;
List<Integer> output = null;
```

G. None of the above

Pitanja?