

Generics

Chapter 20 and 21 of Java How to Program

Edited by Ehsan Edalat and Amir Kalbasi

Generics vs. Non-Generic

- ▶ Consider the following code:

```
List list = new ArrayList();  
list.add("hello");  
String s = (String) list.get(0);
```

- ▶ Needs casting! If done incorrectly, results in runtime error, which are difficult to correct.

- ▶ With Generics:

```
List<String> list = new ArrayList<String>();  
list.add("hello");  
String s = list.get(0); // no cast needed
```

Coding to the interface

Generics

- ▶ Java **Generic** methods and generic classes enable programmers to specify, with a single method declaration or with a single class declaration, **a set of related types**, respectively.
- ▶ Using Java Generic concept, we might write a generic method for sorting an array of objects, then invoke the generic method with Integer arrays, Double arrays, String arrays and so on, to sort the array elements

Generic Classes

- ▶ A generic class also known as **parameterized classes or parameterized types** looks like a non-generic class except that the **class name** is followed by a **type parameter** section in **< >**.
- ▶ The type parameter section can have **one or more** type parameters separated by **commas**.

Generic Class Example

```
1. public class Box<T> {  
2.     private T t;  
3.     public void add(T t) {  
4.         this.t = t;  
5.     }  
6.     public T get() {  
7.         return t;  
8.     }  
9.     public static void main(String[] args) {  
10.        Box<Integer> integerBox = new Box<Integer>();  
11.        Box<String> stringBox = new Box<String>();  
12.        integerBox.add(new Integer(10));  
13.        stringBox.add(new String("Hello World"));  
14.        System.out.printf("Integer Value :%d\n\n", integerBox.get());  
15.        System.out.printf("String Value :%s\n", stringBox.get());  
16.    }  
17. }
```

Output

Integer Value :10

String Value :Hello World

Generic Methods

- ▶ You can write a single generic method declaration that can be called with arguments of different types.
- ▶ Based on the types of the arguments passed to the generic method, the **compiler** handles each method call appropriately.

Rules to define Generic Methods

- ▶ All generic method declarations have a **type parameter section** delimited by angle brackets (< and >) that precedes the method's return type .

Example:

```
public static < E > void foo( E inputparam )
```

- ▶ Each type parameter section contains one or more type parameters separated by **commas**. A type parameter, also known as a type variable, is an identifier that specifies a generic type name.

Rules to define Generic Methods

- ▶ The type parameters can be used to declare the **return type** and act as placeholders for the types of the **arguments** passed to the generic method, which are known as actual type arguments.
- ▶ A generic method's body is declared like that of any other method. Note that type parameters can represent only **reference types**, not primitive types (like int, double and char).

Generic Methods Example

```
1. public class GenericMethodTest {  
2.     // generic method printArray  
3.     public static < E > void printArray( E[] inputArray ) {  
4.         // Display array elements  
5.         for(E element : inputArray) {  
6.             System.out.printf("%s ", element);  
7.         }  
8.         System.out.println();  
9.     }
```

Generic Methods Example

```
10. public static void main(String args[]) {  
11.     // Create arrays of Integer, Double and Character  
12.     Integer[] intArray = { 1, 2, 3, 4, 5 };  
13.     Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };  
14.     Character[] charArray = { 'H', 'E', 'L', 'L', 'O'};  
15.  
16.     System.out.println("Array integerArray contains:");  
17.     printArray(intArray); // pass an Integer array  
18.     System.out.println("\nArray doubleArray contains:");  
19.     printArray(doubleArray); // pass a Double array  
20.     System.out.println("\nArray characterArray contains:");  
21.     printArray(charArray); // pass a Character array  
22. }  
23. }
```

Generic Methods Example

Output

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

Bounded Type Parameters

- ▶ There may be times when you'll want to **restrict the kinds of types** that are allowed to be passed to a type parameter.
- ▶ To declare a bounded type parameter, list the type parameter's name, followed by the **extends** keyword, followed by its upper bound.

Generic Class with Bounded Type

```
public class Zoo<A extends Animal> {  
    private ArrayList<A> animals = new ArrayList<A>();  
    public void add(A animal) {  
        animals.add(animal);  
    }  
    public void display() {  
        System.out.println(animals.toString());  
    }  
}
```

Generic Class with Bounded Type

```
public class ZooTest {  
    public static void main(String[] args) {  
        Animal africanZebra = new Animal("African Zebra");  
        Lion persianLion = new Lion("Persian Lion");  
        Tiger caspianTiger = new Tiger("Caspian Tiger");  
  
        Zoo<Animal> tehranZoo = new Zoo<Animal>();  
        tehranZoo.add(africanZebra);      tehranZoo.add(persianLion);  
tehranZoo.add(caspianTiger);  
        tehranZoo.display();  
  
        Zoo<Lion> smallZoo = new Zoo<Lion>();  
        Lion africanLion = new Lion("African Lion");  
        Puma puma = new Puma("American Puma");  
  
        smallZoo.add(persianLion);      smallZoo.add(africanLion);      smallZoo.add(puma);  
        //          smallZoo.add(africanZebra);      //          not acceptable  
        smallZoo.display();  
        //          Zoo<Object> impossibleZoo = new Zoo<Object>();      //          not  
acceptable  
    }  
}
```

Example on Generic Method with Bounded Type

```
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  
    }  
}
```

Example on Generic Method with Bounded Type

```
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("Max 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"));  
}
```

Output

Max of 3, 4 and 5 is 5

Max of 6.6,8.8 and 7.7 is 8.8

Max of pear, apple and orange is pear