Java - Generics

It would be nice if we could write a single sort method that could sort the elements in an Integer array, a String array, or an array of any type that supports ordering.

Java **Generic** methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods, or with a single class declaration, a set of related types, respectively.

Generics also provide compile-time type safety that allows programmers to catch invalid types at compile time.

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 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. Following are the 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 (< E > in the next example).
- 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.
- 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).

Example

Following example illustrates how we can print an array of different type using a single Generic method −

Live Demo

```
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
   }
}
```

This will produce the following result −

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. For example, a method that operates on numbers might only want to accept instances of Number or its subclasses. This is what bounded type parameters are for.

To declare a bounded type parameter, list the type parameter's name, followed by the extends keyword, followed by its upper bound.

Example

Following example illustrates how extends is used in a general sense to mean either "extends" (as in classes) or "implements" (as in interfaces). This example is Generic method to return the largest of three Comparable objects −

Live Demo

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

This will produce the following result −

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
```

Generic Classes

A generic class declaration looks like a non-generic class declaration, except that the class name is followed by a type parameter section.

As with generic methods, the type parameter section of a generic class can have one or more type parameters separated by commas. These classes are known as parameterized classes or parameterized types because they accept one or more parameters.

Example

Following example illustrates how we can define a generic class −

Live Demo

```
public class Box<T> {
   private T t;

   public void add(T t) {
      this.t = t;
   }

   public T get() {
      return t;
   }

   public static void main(String[] args) {
      Box<Integer> integerBox = new Box<Integer>();
      Box<String> stringBox = new Box<String>();

      integerBox.add(new Integer(10));
      stringBox.add(new String("Hello World"));

      System.out.printf("Integer Value :%d\n\n", integerBox.get());
      System.out.printf("String Value :%s\n", stringBox.get());
   }
}
```

This will produce the following result −

Output

```
Integer Value :10
String Value :Hello World
```