Sri SAI RAM ENGINEERING COLLEGE
INSTITUTE OF TECHNOLOGY
West Tambaram, Chennai - 44

SAIRAM DIGITAL RESOURCES

YEAR II SEM III

CS8392

OBJECT ORIENTED PROGRAMMING
(Common to CSE, EEE, EIE, ICE, IT)

UNIT NO 4

MULTITHREADING AND GENERIC PROGRAMMING

4.8 GENERIC METHODS

COMPUTER SCIENCE & ENGINEERING

UNIT - IV
Generic Methods in Java

TOPICS PLAN

- Introduction of Generics in Java
- Advantage of Java Generics
- Generic class
- Example of Generic class
- Generic Methods
- Example of Generic Method
- Type Parameter
- Bounded Type Parameter
- Example of Bounded Type Parameter
- Video Link
- MCQ Link

Introduction of Generics in Java

- The Java Generics programming is introduced in J2SE 5 to deal with type-safe objects. It makes the code stable by detecting the bugs at compile time.
- Before generics, we can store any type of objects in the collection, i.e., non-generic. Now generics force the java programmer to store a specific type of objects.

Advantage of Java Generics

- There are mainly 3 advantages of generics. They are as follows:
- **Type-safety:** We can hold only a single type of objects in generics. It doesn't allow to store other objects.
  - **Without Generics, we can store any type of objects.**
  - List list = **new** ArrayList();
  - list.add(10); list.add("10");
  - **With Generics, it is required to specify the type of object we need to store.**
  - List<Integer> list = **new** ArrayList<Integer>();
  - list.add(10);
  - list.add("10");// compile-time error

Advantage of Java Generics

- **Type casting is not required: There is no need to typecast the object.**
  - **Before Generics, we need to type cast.**
  - List list = new ArrayList();
  - list.add("hello");
  - String s = (String) list.get(0);//typecasting
  - **After Generics, we don't need to typecast the object.**
  - List<String> list = new ArrayList<String>();
  - list.add("hello");
  - String s = list.get(0);

Advantage of Java Generics

- **Compile-Time Checking:** It is checked at compile time so problems will not occur at runtime.
- The good programming strategy says it is far better to handle the problem at compile time than runtime.

```
1.List<String> list = new ArrayList<String>();
2.list.add("hello");
3.list.add(32);//Compile Time Error
```

Generic class

- A class that can refer to any type is known as a generic class. Here, we are using the T type parameter to create the generic class of specific type.
- Let's see a simple example to create and use the generic class.
- Creating a generic class:

```
class MyGen<T>{

T obj;

void add(T obj){this.obj=obj;}

T get(){return obj;}

}
```

- The T type indicates that it can refer to any type (like String, Integer, and Employee). The type you specify for the class will be used to store and retrieve the data.

Example of Generic class

- **Let's see the code to use the generic class.**

```
class TestGenerics3{

public static void main(String args[]){

MyGen<Integer> m=new MyGen<Integer>();

m.add(2);

//m.add("vivek");//Compile time error

System.out.println(m.get());  }

}
```

Output

```
2
```

Type Parameters

- The type parameters naming conventions are important to learn generics thoroughly.
- The common type parameters are as follows:
    1. T - Type
    2. E - Element
    3. K - Key
    4. N - Number
    5. V - Value

Generic Methods

- Generic methods are methods that introduce their own type parameters.
- **Let's see a simple example of java generic method to print array elements. We are using here E to denote the element.**

```
public class TestGenerics4{
    public static < E > void printArray(E[] elements) {
      for ( E element : elements) {
         System.out.println(element );
       }
       System.out.println();
   }
}
```

Generic Method

```
public static void main( String args[] )
{
 Integer[] intArray = { 10, 20, 30, 40, 50 };
        Character[] charArray = { 'J', 'A', 'V', 'A" };
        System.out.println( "Printing Integer Array" );
        printArray( intArray  );
        System.out.println( "Printing Character Array" );
        printArray( charArray );
}
```

Output

```
Printing Integer Array
10
20
30
40
50
Printing Character Array
J
A
V
A
```

Rules to define Generic Method

- 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.

Rules to define Generic Method

- 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).

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.
- 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 −**

Bounded Type Parameters

```
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
  }
```

Bounded Type Parameters

```
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

VIDEO LINK

VIDEO LINK FOR GENERIC METHODS

https://www.youtube.com/watch?v=22AUtQnTZkY

MCQ LINK

MCQ LINK FOR CHARACTER STREAM

https://forms.gle/UieZugJPiwVZcM1S9