SNS COLLEGE OF ENGINEERING

Coimbatore-107
An Autonomous Institution

COURSE NAME : 23CSB201 & Object Oriented Programming

II YEAR/ III SEMESTER

UNIT – IV I/O, GENERICS, STRING HANDLING

Topic: Generic Programming

Dr.P.Poonkodi

Assistant Professor(SG)

Department of Computer Science and Technology

Introduction

- Allow to create classes, Interfaces and methods that can work with any data type
- Generics does not work with primitive types (int, float, char, etc)
- Enables type safety , Code reusability and reduces casting

Example

```
// Generic class
class Box<T>
{
        private T content;
        public Box(T content)
        {
                this.content = content;
        }
        public T getContent()
        {
                return content;
        }
}
```

Example

```
public class Main
{
        public static void main(String[] args)
        {
            // Use with String
               Box<String> stringBox = new Box<>("Hello");
               System.out.println(stringBox.getContent());
            // Use with Integer
               Box<Integer> integerBox = new Box<>(123);
               System.out.println(integerBox.getContent());
        }
}
```

Example - Explanation

- T used inside the angle bracket <> indicates the type parameter. Inside the Main class, we have created two objects of GenericsClass **Box**

- **integerBox** - Here, the type parameter T is replaced by Integer. Now, the GenericsClass **Box** works with integer data
- **stringBox** - Here, the type parameter T is replaced by String. Now, the GenericsClass **Box** works with string data

Generics Method

- Similar to the generics class, we can also create a method that can be used with any type of data
- Such a class is known as Generics Method
- Here's is how we can create a generics method in Java
-

Example

```
class Main
{
  public static void main(String[] args)
  {
        // initialize the class with Integer data
       DemoClass demo = new DemoClass();
        // generics method working with String
       demo.<String>genericsMethod("Java Programming");
        // generics method working with integer
       demo.<Integer>genericsMethod(25);
    }
}
```

Example

```
class DemoClass
{
        // creae a generics method
        public <T> void genericsMethod(T data)
{
                System.out.println("Generics Method:");
                System.out.println("Data Passed: " + data);
        }
}
```

Example - Explanation

- created a generic method named genericsMethod
- public <T> void genericMethod(T data) {...}
- the type parameter <T> is inserted after the modifier public and before the return type void
- We can call the generics method by placing the actual type <String> and <Integer> inside the bracket before the method name
- demo.<String>genericMethod("Java Programming");
- demo.<Integer>genericMethod(25);

Bounded Type

- the **type parameter** can accept any data types (except primitive types)
- if we want to use generics for some specific types (such as accept data of number types) only, then we can use bounded types
- we use the extends keyword
- <T extends A>
  - This means T can only accept data that are subtypes of A

Example

```
class GenericsClass <T extends Number>
{
        public void display()
        {
                System.out.println("This is a bounded type generics class.");
        }
}
class Main
{
        public static void main(String[] args)
        {
                // create an object of GenericsClass
                 GenericsClass<Integer> intobj = new GenericsClass<>();
                GenericsClass<String> stringobj = new GenericsClass<>();
        }
}
```

Example - Explanation

- we have created a class named GenericsClass. Notice the expression, notice the expression

  <T extends Number>

- GenericsClass is created with bounded type
- This means GenericsClass can only work with data types that are children of Number (Integer, Double, and so on)
- However, we have created an object of the generics class with String In this case, we will get the following error

```
GenericsClass<String> obj = new GenericsClass<>();
                                                ^
    reason: inference variable T has incompatible bounds
      equality constraints: String
      lower bounds: Number
  where T is a type-variable:
    T extends Number declared in class GenericsClass
```

Key Aspects

Advantages

- Type Parameters (like T, E, K, V)
- Compile time type checking
- Code reusability
- Bounded type parameters (Specific)
- Wildcards (?)

Restrictions & Limitations

- type erasure, which removes type information at runtime
- Cannot Instantiate Generic Types with **Primitive Types**
- Cannot Create **Instances** of Type Parameters
- Cannot Declare **Static Fields** Whose Types are Type Parameters
- Cannot Use **Casts** or instanceof With Parameterized Types
- Cannot Create **Arrays** of Parameterized Types

References

- Java : the complete Reference (Eleventh Edition), Herbert Schildt, 2018.


Thank You!