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

4.7 GENERIC PROGRAMMING
-GENERIC CLASSES

COMPUTER SCIENCE & ENGINEERING

Introduction of Generic Programming

- The Java Generics programming provides option for writing codes that will work for objects of many different data types.
- Generic programming enables the programmer to create classes,interfaces and methods in which type of data is specified as a parameter.
- It provides a facility to write an algorithm independent of any specific type of data.
- Generics also provide type safety. Type safety means ensuring that an operation is being performed on the right type of data before executing that operation.

Introduction of Generic Programming

- Can create a single class ,interface or method that automatically works with all types of data(Integer, String, Float etc).
- It has expanded the ability to reuse the code safely and easily.

Introduction of Generic Programming

SYNTAX:

Class_name <data type> reference_name = new Class_name<data type> ();

OR

Class_name <data type> reference_name = new Class_name<>();

Generic class

- A class that can refer to any type is known as a generic class.
- T type parameter is used to create the generic class of specific type.
- A generic class is defined with the following format:

```
class name<T1, T2, . . . , Tn> { /* . . . */ }
```

- The type parameter section, delimited by angle brackets (<>), follows the class name.
- It specifies the type parameters (also called type variables) T1, T2, . . ., and Tn.

Generic class

Example:

```
class MyGen<T>{
T obj;
void add(T obj){this.obj=obj;}
T get(){return obj;}
}
```

- 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

```
class Gen <T>
{
  T ob;  //an object of type T is declared<
  Gen(T o)  //constructor
  {
    ob = o;
  }
  public T getOb()
  {
    return ob;
  }
}
```

```
class Demo
{
  public static void main (String[] args)
  {
Gen < Integer> iob = new Gen<>(100);
//instance of Integer type Gen Class
int  x = iob.getOb();
System.out.println(x);
Gen < String> sob = new Gen<>("Hello");
//instance of String type Gen Class
 String str = sob.getOb();
 System.out.println(str);
  }
}
```

output:

100

Hello

Generic Type with more than one parameter

EXAMPLE:

```
class Gen <T1,T2>
{
  T1 name;
  T2 value;
  Gen(T1 o1,T2 o2)
  {
    name = o1;
    value = o2;
  }
  public T1 getName()
  {
    return name;
  }
  public T2 getValue()
  {
    return value;
  } }
```

```
class Demo
{
  public static void main (String[] args)
  {
        Gen < String,Integer> obj = new
Gen<>("SAIRAM",1);

    String  s = obj.getName();
    System.out.println(s);

    Integer i = obj.getValue();
    System.out.println(i);
  }
}
```

output:

SAIRAM
1

Type Parameters

- Type parameters:
    1. T - Type
    2. E - Element
    3. K - Key
    4. N - Number
    5. V - Value

Advantage of Java Generics

There are mainly 3 advantages of generics. They are as follows:

1.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

2.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.

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

Advantage of Java Generics

3.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);
```