UNIT I

Parametrized Generic Classes

Generic method: Syntax

- A method that can refer to any data type is known as a generic method

The syntax for declaring a generic method is as follows:

```
<access specifier> <return type> mName(<type list> )
{
 //Body of the method
}
```

Example : A generic method for printing

```
class DemoClass        {
// Defining a generic method to print any data type  void genericPrint (T t) {
System.out.println (t);
}
public static void main(String[] args) {
DemoClass aObj;        // Creating an object of the class
aObj.genericPrint(101); // Calling generic method with int argument
aObj.genericPrint("Joy with Java"); // Calling generic method with String
aObj.genericPrint(3.1412343); // Calling generic method with double
}
}
```

Generic method versus method overloading

Note:

1.You can readily understand the similarity between method overloading and generic method. Both the concepts have the same objective, but in their own ways.

2.The main difference is that in case of method overloading, we have to build code for each overloaded method, whereas, with generic method, same code can work for the different type of data.

3.Further, with generic method, theoretically you can pass any type of data as argument; however, with method overloading only a limited number of arguments are allowed.

4.According to the class encapsulation, method overloading and method overriding are also applicable to generic methods.

Example: Static generic method

```
class StaticGenericMethodDemo{
// Defining a static generic method to print any data type  static <T> void genericPrint (T t)
{
//The following statement print which type parameter T this method is handling
 System.out.println (t.getClass().getName() + ":" + t);
}

public static void main(String[] args){
genericPrint(101); // Calling generic method with integer argument  genericPrint("Joy with Java");
 // Calling generic method with String argument  genericPrint(3.1412343);
// Calling generic method with double argument
}
}
```

Parameter passing

- Type(s) of parameter(s) in a method definition is an issue

Note:
Parameter(s) should be class type(s)

- Generic method for Integer swap operation

```
class SwapTest1{
public static void swap(T x, T y){   T temp;
t = x;   x = y;   y = t;
}
public static void main(String args[]){  Integer x = new Integer(99);  Integer y = new
   Integer(66);
System.out.println("x = “ + x + " “ + "y = “ + y);  swap(x, y);
System.out.println("x = “ + x + " “ + "y = “ + y);
}
}
```

Generic method for String swap operation

```
class SwapTest3{
public static void swap(T x, T y){   T temp;
t = x;   x = y;   y = t;
}
public static void main(String args[]){  String x ="99";
String y = "66";
System.out.println("x = “ + x + " “ + "y = “ + y);  swap(x, y);
System.out.println("x = “ + x +" “ + "y = “ + y);
}
}
```

Declaration of “varargs” methods

1. Using an array

```
gMethod1(T[] t);
```

2. Using ellipsis (three dots)

```
gMethod2(T ... t);
```

3. Using Object class

```
gMethod3(Object[] o);
```

varargsmethods using array

- You can define a varargs method with an argument an array (of any type).
- In other words, the values which you want to pass to a method, store them in an array and then pass the array to the method

```
class VarargsMethodDemo1 {
static void varargsMethod1(int v[])  {
System.out.print("Number of args: " + v.length +" Elements: ");  for(int x : v)
System.out.print(x + " ");
System.out.println();
}
public static void main(String args[]) {
// Following arrays are created for test...  int x[] = { 1, 3, 5, 7 };
int y[] = { 2, 4};
int z[] = { };
varargsMethod1 (x); // Passed 4 values to the method  varargsMethod1 (y); // Passed
    2 values to the method  varargsMethod1 (z); // Passed no argument to the method
}
}
```

Thank You