**GENERICS**

Generics mean parameterized types. Parameterized types are important because they enable you to create classes, interfaces and methods in which the type of data upon which they operate is a specified as a parameter. With this capability, you can define a class or a method with generic types that the compiler can replace with concrete types.

A class, interface or method that operates on a parameterized type is called generic, as in generic class or generic method.

The key benefit of generics is to enable errors to be detected at compile – time rather than run – time.

It is important to understand that Java has always given the ability to create generalized class, interfaces and methods by operating through references of type **Object.** Because **Object** is the super class of all other classes, an **Object** reference can refer to any type object. But this does not provide type – safety to the program. Generics add the type safety that was lacking.

**A SAMPLE PROGRAM TO DEMONSTRATE GENERICS**

class Gen<T>

{

T obj;

Gen(T o)

{

obj = o;

}

void showType()

{

System.out.println("Type of T is:"+ obj.getClass().getName());

}

T getObj()

{

return obj;

}

}

public class GenericsDemo

{

public static void main(String[] args)

{

Gen<Integer> iob;

iob = new Gen<Integer> (80);

iob.showType();

int num = iob.getObj(); // No type cast required

System.out.println("Num: " + num);

System.out.println();

Gen<String> strob;

strob = new Gen<String> ("Hello");

strob.showType();

String str = strob.getObj(); // No type cast required

System.out.println("String: " + str);

}

}

Output

Type of T is: java.lang.Integer

Num: 80

Type of T is: java.lang.String

String: Hello

In the above example,

<T> represents a formal generic type, which can be replaced later with an actual concrete type. Replacing a generic type is called **generic instantiation**. By convention, a single capital letter **<E>** or **<T>** is used to denote a formal generic type. Whenever a type parameter is declared, it is specified with angle bracket.

**Note**

In the above program, two types are specified for Gen generics. It doesn’t mean Java creates two different versions of Gen. Instead; the compiler removes all generics type information, and substituting the necessary casts. Thus, there is really only one version of Gen that actually exists in the program. **The process of removing generic type information is called eraser**.

**Generics work only with Objects**

While declaring an instance of the generic type, the type argument passed to the type parameter must be a class type. You cannot use primitive type, such as int or char.

Gen<int> iob = new Gen<int> (80); // Error

**Generics Types Differ Based on their types arguments**

A key point to understand about generic types is that a reference of one specific version of a generic type is not type compatible with another version of the same generic type. For example, the following line of code is in error:

iob = strob;

Even though iob and strob are of type Gen<T>, they are references to different types because their parameter differ.

**Advantage of Generics**

1. **Type – Safety**: We can hold only single type of objects in generics. It doesn’t allow to store other objects.
2. **Type** – **Casting** **is not required:** There is no need to typecast the object.
3. **Compile time checking**: It is checked at compile time, so problem will not occur at run – time. The good programming strategy says it it far better to handle the problem at compile time than run – time.

**Type Parameters**

The type parameters naming conventions are important to learn generics thoroughly.

**T**: Type

**E**: Element

**K**: Key

**N**: Number

**V**: Value

**java.util.ArrayList<E>**

1. ArrayList()
2. void add(E o)
3. void add(int index, E o)
4. void clear()
5. boolean contains(Object o)
6. E get(int index)
7. int indexOf(Object o)
8. boolean isEmpty()
9. int lastIndexof(Object o)
10. boolean remove(Object o)
11. int size()
12. boolean remove(int index)
13. E set(int index, E o)