Lecture 11

Generics

- Generics means **parameterized types**. The idea is to allow type (Integer, String, … etc., and user-defined types) to be a parameter to methods, classes, and interfaces.

- Using Generics, it is possible to create classes that work with different data types. An entity such as class, interface, or method that operates on a parameterized type is a generic entity.

Why Generics?

- The **Object** is the superclass of all other classes, and Object reference can refer to any object. These features lack type safety. Generics add that type of safety feature in later examples.

- **Generics in Java** are similar to **templates in C++**. For example, classes like HashSet, ArrayList, HashMap, etc., use generics very well. There are some fundamental differences between the two approaches to generic types.

Types of Java Generics

- **Generic Method:** Generic Java method takes a parameter and returns some value after performing a task. It is exactly like a normal function, however, a generic method has type parameters that are cited by actual type. This allows the generic method to be used in a more general way. The compiler takes care of the type of safety which enables programmers to code easily since they do not have to perform long, individual type castings.

- **Generic Classes:** A generic class is implemented exactly like a non-generic class. The only difference is that it contains a type parameter section. There can be more than one type of parameter, separated by a comma. The classes, which accept one or more parameters, ?are known as parameterized classes or parameterized types.

Generic Class

- Like C++, we use <> to specify parameter types in generic class creation. To create objects of a generic class, we use the following syntax.

- SYNTAX
  // To create an instance of generic class
  BaseType <Type> obj = new BaseType <Type>();

Generic Functions

- We can also write generic functions that can be called with different types of arguments based on the type of arguments passed to the generic method. The compiler handles each method.

Generics Work Only with Reference Types

- When we declare an instance of a generic type, the type argument passed to the type parameter must be a reference type. We cannot use primitive data types like **int**, **char.**

Test<int> obj = new Test<int>(20); // wrong

- The above line results in a compile-time error that can be resolved using type wrappers to encapsulate a primitive type.

Test<integer> obj = new Test<integer>(20); // correct

Type Parameters in Java Generics

- The type parameters naming conventions are important to learn generics thoroughly. The common type parameters are as follows:

  - T – Type
  - E – Element
  - K – Key
  - N – Number
  - V – Value

Advantages of Generics

- Programs that use Generics has got many benefits over non-generic code.

  1. Code Reuse: We can write a method/class/interface once and use it for any type we want.

  2. Type Safety: Generics make errors to appear compile time than at run time

  3. Individual Type Casting is not needed.

  4. Generics Promotes Code Reusability: With the help of generics in Java, we can write code that will work with different types of data.

  5. Implementing Generic Algorithms: By using generics, we can implement algorithms that work on different types of objects, and at the same, they are type-safe too.