**Generics in Java**

**Generics in Java** allow us to create classes, interfaces, and methods where the type of the data is specified as a parameter. If we use generics, we do not need to write multiple versions of the same code for different data types.

**Why Use Generics?**

Generics allow you to write code that works with *different types* while providing **type safety** and **reusability**.

**Types of Java Generics**

**1.Generic Method:**

A generic Java method takes a parameter and returns some value after performing a task. The compiler takes care of the type of safety, which enables programmers to code easily sin ce they do not have to perform long, individual type castings.

**2. 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 that accept one or more parameters are known as parameterized classes or parameterized types.

// Generic classes

class Test<T> {

 T obj;

 Test(T obj) { this.obj = obj; } // constructor

 public T getObject() { return this.obj; }

}

class Geeks {

 public static void main(String[] args)

 {

 Test<Integer> = new TeiObjst<Integer>(15);

 System.out.println(iObj.getObject());

 Test<String> sObj

 = new Test<String>("GeeksForGeeks");

 System.out.println(sObj.getObject());

 }

}

We can also pass multiple Type parameters in Generic classes.

class Test<T, U>

{

 T obj1; // An object of type T

 U obj2; // An object of type U

 Test(T obj1, U obj2)

 {

 this.obj1 = obj1;

 this.obj2 = obj2;

 }

 public void print()

 {

 System.out.println(obj1);

 System.out.println(obj2);

 }

}

class Geeks

{

 public static void main (String[] args)

 {

 Test <String, Integer> obj = new Test<String, Integer>("GfG", 15);

 obj.print();

 }

}

| **Limitation** | **Example Problem** | **Typical Workaround** |
| --- | --- | --- |
| Type Erasure | No runtime type info | Avoid relying on type checks |
| No primitives as type params | List<int> invalid | Use wrappers like Integer |
| No new T() | Can't create T | Use factories or Class<T> |
| No generic arrays | new T[10] invalid | Cast from Object[] |
| No instanceof with type params | instanceof List<String> invalid | Use raw List checks |
| No static type parameter fields | static T invalid | Use static with explicit types only |
| No generic exceptions | Can't extend/throw/catch generic exceptions | Use non-generic exceptions |