Generics

1. Introduction
2. Generic classes
3. Bounded type
4. Generic method
5. Wild card character (?)
6. Communication with non-generic code
7. Conclusion.

To provide type-safety
To resolve type-casting problem.

Case-1] Type Safety

→ Arrays are type safe we can give gurantee for the type of element present in the array.

```
String s = new String();
s[1] = "Durga";          ✓
s[2] = new Integer();    ✗
```

→ Collection are not type safe.

```
ArrayList l = new ArrayList();
l.add("Durga");          ✓
l.add(new Integer(10));  ✓
```

`String name1 = s[0];` → Type casting not required.

`String name = l.get(0);` (Incompatible types)

`String name = (String) l.get(0);` → Type Casting is Mandatory

To overcome above problem.

↳ Introduce generics concept in 1.5V

① To provide type safety

② To resolve type-casting problems.

To hold only string object.

```
AL <String> l = new AL <String>();
l.add("durga");
l.add("Ravi");
l.add(new Integer(10));   ---- C.E. (x)
```

Through generics we are getting type safety & type casting.

```
String name = l.get(0);
```

↳ Type casting is not Required.

Base type → AL <String> l = new AL <String>(); ← Parameter type

✓ List <String> l = new AL <String>();

✓ Collection <String> l = new AL <String>();

✗ AL <object> l = new AL <String>();

↳ Incompatible type

→ polymorphism is allowed for only Base type not for parameter type.

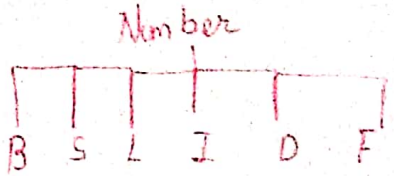
→ class Account <T>
{

}

Account <gold> a1 = new Account <gold> ();

Account <platinum> a2 = new Account <platinum> ();

→ class Test <T extends Number> ← Bounded types.
{
  Public void m1()
  {
    T a, b;
    Sout (a+b);
    Sout (a+b);
    Sout (a/b);
  }
}



→ We can bound our parameter to number only

→ class Test <T>
{
.
.
.
}
✓

class Test <T extends Number>
{
.
.
.
}
✓

class test <T implements Runnable>
{
.
.
.
}
✗ instead of implements we can extend interface.

---

class test <T super String>
{
.
.
.
} ✗

→ class Test<T extends Number & Runnable>
{
    // Now it can bounded with Number & Runnable.
}

→ class Test<T extends Number, & Runnable & comparable>

× class Test <T extends Runnable & Number>

↳ Bcoz we have to take class first then interface.

× class Test <T extends Number & Thread>

↳ Bcoz we can't extend more than one class simultaneously.

Multiple parameter type

HashMap<Integer, String> h = HashMap<Integer, String>();

Generics Methods & Wild Card character (?)

→ AL <String> l = new AL <String> ();
m1(l);

→ AL <Integer> l = new AL <Integer> ();
m1(l);

→ AL <Double> l = new AL <Double> ();
m1(l);

→ AL <Student> l = new AL <Student> ();
m1(l)

→ For every types considering.

```
m1(AL<?>)
{

}
```

Any type

→ m1(AL <? extends Number> l)

→ m1(AL <? super X> l)

→ Between the method we can't add anything to list except null.

→ Best suitable for read only operation.

→ We can add anything & null.

→ Super can be used in method only

→ Interface can also be used