=> Generics :-

-Generics means parametrized types which means that we can provide any type of parameter to the classes, interfaces or methods

-Generics were introduced in JDK 1.5 version

-Generics are represented by angular braces - < >

The main objective of Generics are :-

1. To provide type safety
2. To resolve type casting problem

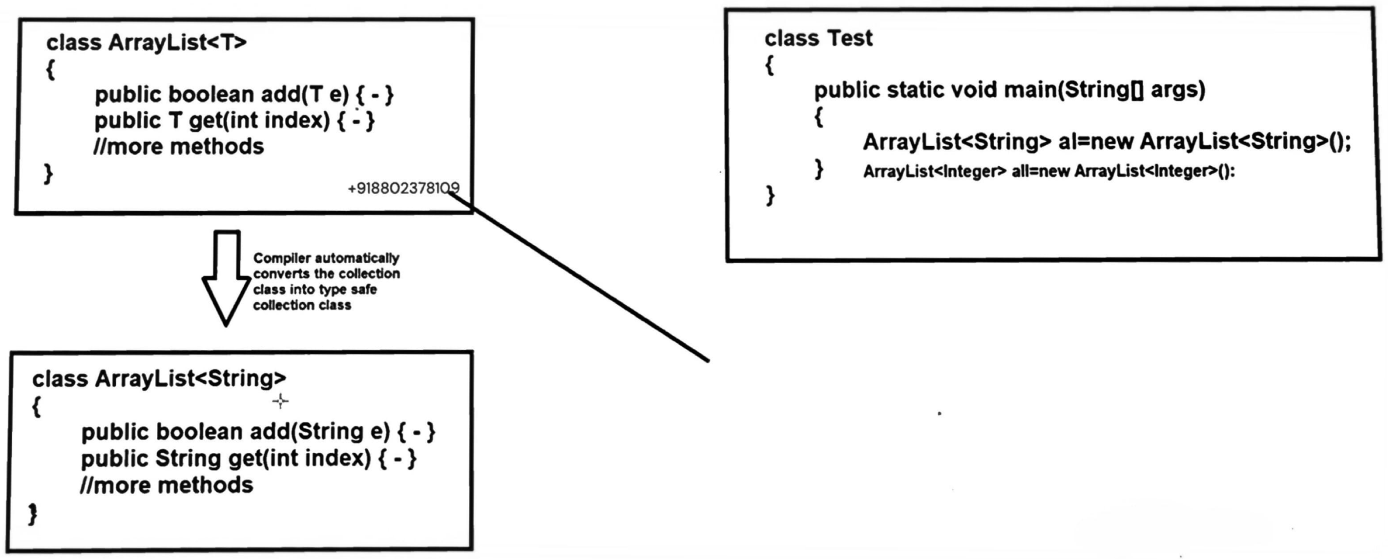
-By default "arrays are type safe"

-Now for collections, till JDK 1.4 version, collections were not generic types

-In JDK 1.5 version, Generic Collections were introduced

-NOTE : We can only provide non-primitive values in generics

```java
/**
 *
 * @author Arun
 */
public class Test
{
    public static void main(String[] args)
    {
        String[] strarr=new String[3];
        strarr[0]="Arun";
        strarr[1]="amit";
        strarr[2]="rahul";

        String name=strarr[1];      //No need of typecasting
        System.out.println(name);

        //-------------------------------------------

        ArrayList al=new ArrayList();

        al.add("Arun");
        al.add(101);
        al.add(10.0f);

        String name1 = (String)al.get(0);    //Type casting is required since it return Object
        int aa = (int)al.get(1);
        System.out.println(name1);
        System.out.println(aa);

        //-----------------------------------------

        ArrayList<Integer> all=new ArrayList<Integer>();
        all.add("Arun");
        all.add("rahul");
        all.add("Pramod");

        String name2=all.get(0);        //No need of typecasting
    }
}
```


Compiler automatically converts the collection class into type safe collection class

```
class ArrayList<T>
{
      public boolean add(T e) { - }
      public T get(int index) { - }
      //more methods
}
```

```
class ArrayList<String>
{
      public boolean add(String e) { - }
      public String get(int index) { - }
      //more methods
}
```

```
class Test
{
      public static void main(String[] args)
      {
            ArrayList<String> al=new ArrayList<String>();
      }     ArrayList<Integer> all=new ArrayList<Integer>():
}
```

=> Generic Classes :-

-If any class is declared with type parameters then it is known as Generic Class

-Generic classes can be user-defined classes or predefined classes (collections classes)

-Generic type can be any valid identifier name

-We can provide any number of parameters in generics

```
class A<T>
{
    T a;
    A(T a)
    {
        this.a=a;
    }
    void show()
    {
        System.out.println(a);
    }
}
public class Demo
{
    public static void main(String[] args)
    {
        A<String> ob1=new A<String>("deepak");
        ob1.show();

        A<Integer> ob2=new A<Integer>(101);
        ob2.show();
    }
}
```

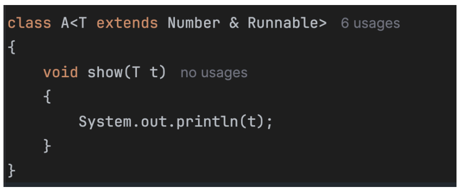
=> Generic Bounded Types :-

-We can bound the type parameter for a particular range by using extends keyword. And this concept is known as Generic Bounded Type Concept

-**Syntax :** class A<T extends X> (X can be any class or interface)

Cases :-

1. We can only use extends keyword, not implements keyword
2. We can only use Non-Primitive data types
3. class A<T extends X & Y>\



```
class A<T extends Number & Runnable>  6 usages
{
    void show(T t)  no usages
    {
        System.out.println(t);
    }
}
```

```
class A<T extends Number>
{
    void show(T t)
    {
        System.out.println(t);
    }
}
public class Test
{
    public static void main(String[] args)
    {
        A<Integer> ob=new A<Integer>();
        ob.show(101);

        A<Float> ob2=new A<Float>();
        ob2.show(100f);

        A<String> ob3=new A<String>(); // Error, since A Class is Bound for numeric Type only

    }
}
```

Generic Methods and Generic WildCard(?) :

```
class A
{
    void show(ArrayList<?> al)  // if generic wildcard(?) is not used then it will give error in case of Integer is (ArrayList<String> al) is provided here
    {
        System.out.println(al);
    }
}
public class Test
{
    public static void main(String[] args)
    {
        A ob1=new A();

        ArrayList<String> al=new ArrayList<String>();
        ob1.show(al);

        ArrayList<Integer> all=new ArrayList<Integer>();
        ob1.show(all);
    }
}
```