Wildcard Generics

There are three types of wild card generics

A) Upper bounded: To declare an upper bounded wildcard use the wildcard character <?> followed by extends keyword by its upper bound.

Syntax: <? extends classname>

Source code without upper bounded

```
package org.techhub;
import java.util.*;
public class UpperBoundedApp {

        public static void main(String[] args) {
         List  <Integer>intList=new ArrayList<Integer>();
         intList.add(10);
         intList.add(20);
         intList.add(30);
         calSum(intList); //call calSum function
         List <Double>dblList= new ArrayList<Double>();
         dblList.add(5.4);
         dblList.add(6.5);
         dblList.add(7.6);
         dblList.add(3.4);
         calSum(dblList);
        }
        public static void calSum(List<Integer> list)
        {       for(Integer val:list)
                 {
                        System.out.println(val);
                 }
        }

}
```

If we think about above code then it will generate the compile time error because in calSum(List<Integer> list) this method can accept the only list of integer but from main method we pass the list of double in calSum() from calling point so it is not acceptable here because strictly specify in calSum() List only works with Integer parameter so if we want to pass any kind of Number means if we want to pass Integer in list, float in list,double list etc then we can use the upper bounded generics in java. So as per our example we need to use the Number class with List using upper bounded generics Because in java Wrapper classes Number class is parent of Integer ,Float and Double etc means when we use the Number with upper bounded in List then any child of Number class is acceptable in List Collection

Source Code with Upper Bounded Generics

```
package org.techhub;
import java.util.*;
public class UpperBoundedApp {

	public static void main(String[] args) {
	 List  <Integer>intList=new ArrayList<Integer>();
	 intList.add(10);
	 intList.add(20);
	 intList.add(30);
	 System.out.println("Integer list is ");
	 calSum(intList); //call calSum function
	 List <Double>dblList= new ArrayList<Double>();
	 dblList.add(5.4);
	 dblList.add(6.5);
	 dblList.add(7.6);
	 dblList.add(3.4);
	 System.out.println("Float list is");
	 calSum(dblList);
	}
	public static void calSum(List<? extends Number> list)
	{
		for(Number val:list)
		{System.out.println(val);
		}
	}
```

}

b) Lower bounded: The way upper bounded wildcard restricts the unknown type to be a specific type or subtype of that type same way lower bounded wildcard restrict the unknown type to be a specific type or super type of that type.

Syntax: <? Super classname>: here we can use the all super classes of specified child mention in Lower bounded generics.

Example

```
package org.techhub;
import java.util.*;
public class LowerBoundedApp {
   public static void main(String[] args) {
         List <Number> numList=Arrays.asList(10,20,30,40,50);
         System.out.println("Number with list");
         calSum(numList);
         System.out.println("Object with list");
         List<Object> objList=Arrays.asList(5.6,6.5);
         calSum(objList);
        }
        public static void calSum(List<? super Integer> list)
        {       for(Object val:list)
                {
                        System.out.println (val);
                }
        }

}
```

c) Unbounded wild cards

Unbounded wild cards as name suggest wild card without any upper or lower bound. Unbounded wildcard type is specific using the wildcard character <?> for Example List<?> This is called of List of unknown type

There are two scenarios where an unbounded wildcard are useful

1) If you are writing method that can be implemented using any functionality provided by in Object class.

2) When the code using methods in Generics class That don’t depend on the type parameter e.g. List.clear (), List.isEmpty ()

Source code Example

```
package org.techhub;
import java.util.*;
public class WildCardGenApp {
   public static void main(String[] args) {
          List <Integer> numList=Arrays.asList(10,20,30,40,50);
          System.out.println("Number with list");
          calSum(numList);
          System.out.println("Object with list");
          List<String> objList=Arrays.asList("good","bad","test");
          calSum(objList);
         }
         public static void calSum(List<?> list)
         {

                  for(Object val:list)
                  {
                           System.out.println(val);
                  }
         }

}
```