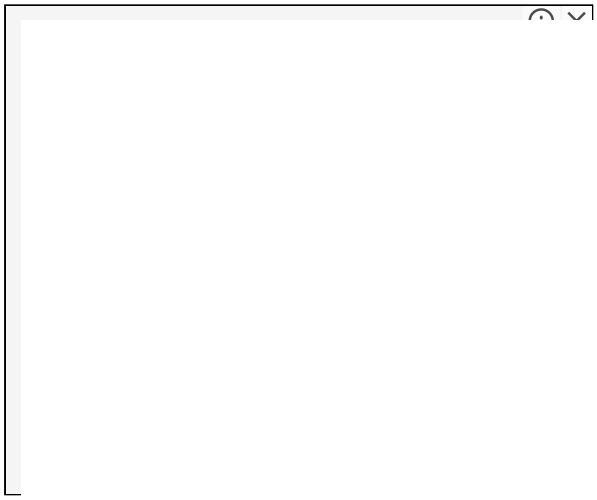

Output

```
Printing Integer Array
10
20
30
40
50
Printing Character Array
J
A
V
A
T
P
O
I
N
T
```

Wildcard in Java Generics

The ? (question mark) symbol represents the wildcard element. It means any type. If we write <? extends Number>, it means any child class of Number, e.g., Integer, Float, and double. Now we can call the method of Number class through any child class object.

We can use a wildcard as a **type of a parameter, field, return type, or local variable. However, it is not allowed to use a wildcard as a type argument for a generic method invocation, a generic class instance creation, or a supertype**.

Let's understand it by the example given below:

```
import java.util.*;
abstract class Shape{
abstract void draw();
}
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle");}
}
class Circle extends Shape{
void draw(){System.out.println("drawing circle");}
}
class GenericTest{
//creating a method that accepts only child class of Shape
public static void drawShapes(List<? extends Shape> lists){
for(Shape s:lists){
s.draw();//calling method of Shape class by child class instance
}
}
public static void main(String args[]){
List<Rectangle> list1=new ArrayList<Rectangle>();
list1.add(new Rectangle());

List<Circle> list2=new ArrayList<Circle>();
list2.add(new Circle());
list2.add(new Circle());

drawShapes(list1);
drawShapes(list2);
```

```
}}
```

Output

```
drawing rectangle
drawing circle
drawing circle
```

Upper Bounded Wildcards

The purpose of upper bounded wildcards is to decrease the restrictions on a variable. It restricts the unknown type to be a specific type or a subtype of that type. It is used by declaring wildcard character ("?") followed by the extends (in case of, class) or implements (in case of, interface) keyword, followed by its upper bound.

Syntax

```
List<? extends Number>
```

Here,

? is a wildcard character.

extends, is a keyword.

Number, is a class present in java.lang package

Suppose, we want to write the method for the list of Number and its subtypes (like Integer, Double). Using **List<? extends Number>** is suitable for a list of type Number or any of its subclasses whereas **List<Number>** works with the list of type Number only. So, **List<? extends Number>** is less restrictive than **List<Number>**.

Example of Upper Bound Wildcard

In this example, we are using the upper bound wildcards to write the method for List<Integer> and List<Double>.

```
```

```
import java.util.ArrayList;

public class UpperBoundWildcard {


   private static Double add(ArrayList<? extends Number> num) {

      double sum=0.0;

      for(Number n:num)
      {
         sum = sum+n.doubleValue();
      }

      return sum;
   }

   public static void main(String[] args) {

      ArrayList<Integer> l1=new ArrayList<Integer>();
      l1.add(10);
      l1.add(20);
      System.out.println("displaying the sum= "+add(l1));

      ArrayList<Double> l2=new ArrayList<Double>();
      l2.add(30.0);
      l2.add(40.0);
      System.out.println("displaying the sum= "+add(l2));


   }

}
```

Test it Now

Output

```
displaying the sum= 30.0
displaying the sum= 70.0
```

Unbounded Wildcards

The unbounded wildcard type represents the list of an unknown type such as List<?>. This approach can be useful in the following scenarios: -

- When the given method is implemented by using the functionality provided in the Object class.
- When the generic class contains the methods that don't depend on the type parameter.

Example of Unbounded Wildcards

```
import java.util.Arrays;
import java.util.List;

public class UnboundedWildcard {

   public static void display(List<?> list)
   {

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

   }


   public static void main(String[] args) {
```

```
    List<Integer> l1=Arrays.asList(1,2,3);
    System.out.println("displaying the Integer values");
    display(l1);
    List<String> l2=Arrays.asList("One","Two","Three");
      System.out.println("displaying the String values");
        display(l2);
    }

}
```

Test it Now

Output

```
displaying the Integer values
1
2
3
displaying the String values
One
Two
Three
```

Lower Bounded Wildcards

The purpose of lower bounded wildcards is to restrict the unknown type to be a specific type or a supertype of that type. It is used by declaring wildcard character ("?") followed by the super keyword, followed by its lower bound.

Syntax

```
List<? super Integer>
```

Here,

? is a wildcard character.

super, is a keyword.

Integer, is a wrapper class.

Suppose, we want to write the method for the list of Integer and its supertype (like Number, Object). Using **List<? super Integer>** is suitable for a list of type Integer or any of its superclasses whereas **List<Integer>** works with the list of type Integer only. So, **List<? super Integer>** is less restrictive than **List<Integer>**.

Example of Lower Bound Wildcard

In this example, we are using the lower bound wildcards to write the method for List<Integer> and List<Number>.

```
import java.util.Arrays;
import java.util.List;

public class LowerBoundWildcard {

   public static void addNumbers(List<? super Integer> list) {

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



   }
public static void main(String[] args) {

   List<Integer> l1=Arrays.asList(1,2,3);
     System.out.println("displaying the Integer values");
   addNumbers(l1);
```

```
    List<Number> l2=Arrays.asList(1.0,2.0,3.0);
      System.out.println("displaying the Number values");
    addNumbers(l2);
}

}
```

Test it Now

Output

```
displaying the Integer values
1
2
3
displaying the Number values
1.0
2.0
3.0
```

← Prev

Next →

Feedback

- Send your Feedback to feedback@javatpoint.com