**Wildcard Arguments With An Unknown Type:**

The syntax for declaring this type of wildcard arguments is,

GenericType<?>

The arguments which are declared like this can hold any type of objects. For example, Collection<?> or ArrayList<?> can hold any type of objects like **String**, **Integer**, **Double** etc.

**Simple Wildcard With An Unknown Type Example**

import java.util.ArrayList;

import java.util.Collection;

import java.util.HashSet;

import java.util.LinkedList;

/\*\*

\* Wildcard Arguments With An Unknown Type

\* @author javaguides.net

\*

\*/

public class WildCardSimpleExample {

public static void printCollection(Collection<?> c) {

for (Object e : c) {

System.out.println(e);

}

}

public static void main(String[] args) {

Collection<String> collection = new ArrayList<>();

collection.add("ArrayList Collection");

printCollection(collection);

Collection<String> collection2 = new LinkedList<>();

collection2.add("LinkedList Collection");

printCollection(collection2);

Collection<String> collection3 = new HashSet<>();

collection3.add("HashSet Collection");

printCollection(collection3);

}

}

Output:

ArrayList Collection

LinkedList Collection

HashSet Collection

**2. Upper Bounded Wildcards**

To specify an upper bound for wildcards, use this syntax,

GenericType<? extends SuperClass>

This specifies that a wildcard argument can contain ‘SuperClass’ type or it’s subclasses. Remember that *extends* clause is an inclusive bound. i.e ‘SuperClass’ also lies in the bound.

**Upper Bounded Wildcards Example**

In the above example, if you want the *processElements()* method to work with only numbers, then you can specify an upper bound for wildcard argument.

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Wildcard Arguments With An Upper Bound Demo

\* @author javaguides.net

\*

\*/

public class WildCardWithUpperBoundExample {

static void processElements(List<? extends Number> a) {

for (Object element : a) {

System.out.println(element);

}

}

public static void main(String[] args) {

// ArrayList Containing Integers

List<Integer> a1 = new ArrayList<>();

a1.add(10);

a1.add(20);

a1.add(30);

processElements(a1);

// ArrayList Containing Longs

List<Long> a2 = new ArrayList<>();

a2.add(100L);

a2.add(200L);

a2.add(300L);

processElements(a2);

// Arraylist containing Doubles

List<Double> a3 = new ArrayList<>();

a3.add(21.35);

a3.add(56.47);

a3.add(78.12);

processElements(a3);

// Arraylist containing Strings

List<String> a4 = new ArrayList<>();

a4.add("One");

a4.add("Two");

a4.add("Three");

// This will not work

//processElements(a4); // Compile time error

}

}

Output:

10

20

30

10

20

30

21.35

56.47

78.12

**3. Lower Bounded Wildcards**

A lower bounded wildcard is expressed using the wildcard character ('?'), following by the super keyword, followed by its lower bound: <? super A>.

GenericType<? super SubClass>

**Lower Bounded Wildcards Example**

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Wildcard Arguments With An Lower Bound Demo

\* @author javaguides.net

\*

\*/

public class WildCardWithLoweroundExample {

static void processElements(List<? super Integer> a) {

for (Object element : a) {

System.out.println(element);

}

}

public static void main(String[] args) {

// ArrayList Containing Integers

List<Integer> a1 = new ArrayList<>();

a1.add(10);

a1.add(20);

a1.add(30);

processElements(a1);

// ArrayList Containing Longs

List<Long> a2 = new ArrayList<>();

a2.add(100L);

a2.add(200L);

a2.add(300L);

processElements(a2); // compiler error

}

}