**Generics – Part\_03**

* **Bounded Types:**

We can bound the type parameter for a particular range by using extends keyword, such types are called “bounded types”.

class Test<T>{

}

As a type parameter we can pass any type and there are no restrictions and hence it is unbounded type.

Test<Integer> t1= new Test<Integer>();

Test<String> s1 = new Test<String>();

* **Syntax for bounded type:**

class Test<T extends X>{

}

X can be either class or interface.

If X is a class then as the type-parameter we can pass either X type or its child classes.

If X is an interface then as the type-parameter we can pass either X type or its implementation classes.

* **Example:**

class Test<T extends Number> {

}

Test<Integer> t1 = new Test<Integer>();// Valid

Test<String> t2 = new Test<String>(); // Invalid

CE: Type parameter java.lang.String is not within its bound.

class Test<T extends Runnable>{

}

Test<Runnable> t1 = new Test<Runnable>(); // Valid

Test<Thread> t2 = new Test<Thread>(); // Valid

Test<Integer> t3 = new Test<Integer>(); // Invalid

CE: type parameter java.lang.Integer is not within its bound.

* **Combined bounded types:**

We can define bounded types even in combination also.

class Test<T extends Number & Runnable>{

}

As the type-parameter we can take anything which should be child class of Number and should implements Runnable interface.

class Test<T extends Runnabe & Comparable>

class Test<T extends Number & Runnable & Comparable>

class Test<T extends Runnable & Number> // Invalid

Because we have to take class first followed by interface next.

class Test<T extends Number & Thread> // Invalid

Because we can’t extend more than one class simultaneously.

* **Conclusions:**

1. We can define bounded types only by using extends keyword and we can’t use implements and super keywords. But we can replace implements keyword purpose with extends keyword.

class Test<T extends Number> // Valid

class Test<T implements Runnable> // Invalid

class Test<T extends Runnable> // Valid

class Test<T super String> // Invalid

1. As a type-parameter “T” we can take any valid Java identifier but it is convention to use “T”. 🡪 (T)ype-parameter.

class Test<T> // Valid class Test<A> // Valid

class Test<X> // Valid class Test<durga> // Valid

1. Based on our requirement we can declare any number of type-parameters and all these type-parameters should be separated with comma ( ,)

class Test<A, B> {} // Valid

class Test<X, Y, Z> {} // Valid

class HashMap<K,V> {} // Valid

K 🡪 Key type, V 🡪 Value type.

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