Key Points about Bounded Types

=> As the type parameter we can use any valid java identifier but it convention to use T always

eg: class Test<T>{}

class Test<iNeuron>{}

=> We can pass any no of type parameters ,need not be one.

eg: class HashMap<K,V>{}

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

Eg : Bounded\_With\_Multiple\_Types

Which of the following are valid?

class Test <T extends Number&Runnable> {} //valid

Number -> class

Runnable-> interface

class Test<T extends Number&Runnable&Comparable> {} //valid

Number -> class

Runnable-> interface

Comparable -> interface

class Test<T extends Number&String> {} //invalid

we can't extends more than one class at a time.

class Test<T extends Runnable&Comparable> {} //valid

Runnable-> interface

Comparable -> interface

class Test<T extends Runnable&Number> {} //invalid

Runnable-> interface

Number -> class

rule: first inherit and the implement so invalid

GenericClass

==============

class: Type parameter

Can we apply TypeParameter at MethodLevel?

Ans.Yes, it is possible.

Generic methods and wild-card character (?) also called wild card pattern

(or)

Generic method applied at method argument level

=====================================

Type1 : adding data of specific type

methodOne(ArrayList<String> aI) :

This method is applicable for ArrayList of only String type.

Eg: Generic\_Method\_With\_Wild\_Card\_Pattern

Within the method we can add only String type of objects and null to the List.

Type 2:

methodOne(ArrayList <? > I):

We can use this method for ArrayList of any type but within the method we can't add anything to the List except null.

Eg: Generic\_Method\_With\_Wild\_Card\_Pattern\_Eg2

This method is useful whenever we are performing only read operation.

This logic will perform printing operation irrespective of whether the element is String ( or) Integer . where we use ? we perform read operation . so when we want to perform these type of operations we use that .

3. methodOne(ArrayList <? extends X> al)

X -> class, we can make a call to method by passing ArrayList of X type or its Child type.

X -> interface, we can make a call to method by passing ArrayList of X type or its Implementation class.

Best suited only for read operation.

Eg: Generic\_Method\_With\_Wild\_Card\_Pattern\_Eg3

4. methodOne(ArrayList <? super X> al)

X -> class, we can make a call to method by passing ArrayList of X type or its super class

X-> interface, we can make a call to method by passing ArrayList of X type or its super class of implementation class of x.

methodOne(ArrayList <? super X> al){

al.add(X);

al.add(null);

}

Eg: Generic\_Method\_With\_Wild\_Card\_Pattern\_Eg4

Note:

Which of the following declarations are allowed?

1. ArrayList<String> I1 = new ArrayList<String>();//valid

2. ArrayList <? > 12 = new ArrayList<String>();//valid

3. ArrayList <? > 13 = new ArrayList<Integer>();//valid

4. ArrayList <? extends Number> 14 = new ArrayList<Integer>();//valid

5. ArrayList <? extends Number> I5 = new ArrayList<String>();//invalid

// wild card pattern (<? extends Number>) should always be in the left hand side because they are the collecting one . here wild card pattern is on both sides which leads to compile time error.

6. ArrayList <? > 16 = new ArrayList <? extends Number>(); //invalid

7. ArrayList <? > 17 = new ArrayList <? >(); //invalid

TypeParameter at Method level :

(or)

Generic method applied at method level

|=> TypeParameter

class Demo<T>{

|=> Type parameter defined just before the return type

public <T> void m1(T t){

}

}

We can give the type parameter as the return type of the method also . but it should be given before void. This is used for normal datatype classes .

It controls the type of the parameter that needs to be passed .

generic method wild card pattern is used for the collection

Which of the following declarations are allowed?

public <T> void methodOne1(T t){} //valid

public<T extends Number> void methodOne2(T t){} //valid

public<T extends Number&Comparable> void methodOne3(T t){} //valid

// first class and then interface .

public<T extends Number&Comparable&Runnable> void methodOne4(T t){} //valid

// class, interface and interface valid

public<T extends Number&Thread> void methodOne(T t){} //invalid

public<T extends Runnable&Number> void methodOne(T t){} //invalid

public<T extends Number&Runnable> void methodOne(T t){} //valid

Eg: Generic\_Method\_With\_Return\_Type

Eg: Generic\_Method\_With\_Return\_Type\_Eg2

Eg: Generic\_Method\_With\_Return\_Type\_Eg3

Note: with wild card pattern you cant extend class and interface at a time , it is possible at class and type parameter at method level.

Communication with non generic code

To provide compatibility with old version sun people compromised the concept of generics in very few area's the following is one such area.

Eg: Non\_Generic\_Declaration

Eg: Generic\_NonGeneric\_Compatability

Generic : both the syntax are valid for declaring generic

ArrayList<String> I1=new ArrayList();

ArrayList<String> I2=new ArrayList<String>();

Conclusion :

Generics concept is available only at compile time , at runtime there is no such concept .

At the time of compilation , as the last step generics concept is removed , hence for jvm generics syntax wont be available

Hence the following declarations are equal. ( non generic syntax’s)

ArrayList al = new ArrayList<String>();

ArrayList al = new ArrayList<Integer>();

ArrayList al = new ArrayList<Double>();

All are equal at the runtime because compiler will remove these generics syntax

ArrayList al = new ArrayList();

Eg: Generics\_At\_Compiler\_Level\_Eg1

Eg: Generics\_At\_Compiler\_Level\_Eg2

Eg: Generics\_At\_Compiler\_Level\_Eg3

Comparable vs Comparator :

Eg: Comparable\_Using\_Treeset\_Basic\_Example

public TreeSet();

| => when we use the above constructor , jvm will internally use comparable interface method to sort the objects based on the default natural sorting order.

What is Comparable interface?

It is a functional interface present in java.lang package.

This interface is internally used by TreeSet object during sorting process

@FunctionalInterface

public interface java.lang.Comparable<T> {

// what ever input type you give in Comparable constructor on that input type compareTo() is called.

public abstract int compareTo(T);

}

Eg: Comparable\_Using\_Treeset\_Eg2

Note : if we are keeping the data inside the TreeSet Object , then the data should be

1. Homogenous - > because it uses compareTo() to sort the object
2. The object should compulsorily implement the interface called “Comparable”

If we fail to do so it would result in the ClassCastExcaption.

Eg: Comparable\_Using\_Treeset\_Eg3

All the wrapper classes and String class has implemented Comparable interface ,

StringBuffer also implemented Comparable interface so no compile time error .

Check it in javap java.lang.StringBuffer