**Generic Class.**

In our last article we saw the internal implementation for ArrayList class where it was a kind of parameterized with a type parameter

class ArrayList<T> if you remember such classes are known as generic classes. We can create a generic class of our own too.

How to create a generic class, let us understand easy way.

**class** GenericClass<T>{

T t;

**public** GenericClass(T t) {

**this**.t=t;

}

//show type of T

**public** **void** show() {

System.***out***.println(t.getClass().getName());

}

//return t

**public** T get() {

**return** t;

}

}

**public** **class** GenericClassTest {

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

GenericClass<Integer> gen=**new** GenericClass<Integer>(3);

gen.show();

System.***out***.println(gen.get());

GenericClass<String> gen1=**new** GenericClass<String>("Generics");

gen1.show();

System.***out***.println(gen1.get());

}

}

In the above code observe how a single class can be used in a generic way to create object for two different types.

What if you are writing a functionality in your generic class that go well only with certain types. Let us say a generic class calculator which performs mathematical operations on numbers.

Will that class go down well with String or any other type except numbers!!!

For the above problem there is a solution which bounds the type parameter by giving it a range. The keyword that bounds it is **extends** sounds familiar!!!

class Calculator<T extends Number>{

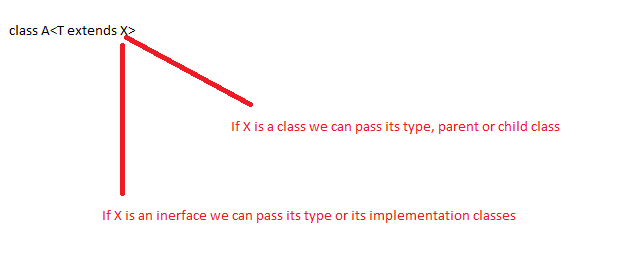
public T add(T a,T b){return a+b}

public T multilply(T a,T b){return a\*b}

}

So now the type parameter is bounded to only take Number or its subclass. What if you want to bound it by giving an interface or its implemented class, for that case also the keyword is **extends**.

The only keyword used to bound the type parameter is **extends**.



Bound can be given in combinations too.

class A<T extends X & Y> (This syntax is valid Y must be an interface)

class A<T extends X & Y & Z> (This syntax is also valid Y and Z must be interface)

class A<T extends Runnable & Number> (This syntax is invalid as Runnable is an interface and Number is a class, the order must be first class and then interface same as in inheritance.)

class A<T extends Number & Thread> (This syntax is invalid as two classes cannot be extended at the same time, same as in inheritance)

The next point to keep in mind is we can have multiple type parameter as well separated with commas.

class<T,X,Y> //Valid syntax.

Think of Maps in Java. Map<Character,Integer> map=new HashMap<Character,Integer>();

**Note- To represent a Type parameter any valid Java identifier can be used instead of T.**