# *CORE JAVA*

**Reference : https://github.com/srihas-sa/CoreJava-DSA.git**

Non-Primitive Data Types

==========]

Non-primitive data types are called reference types because they refer to objects.

The main differences between primitive and non-primitive data types are:

Primitive types in Java are predefined and built into the language, while non-primitive types are created by the programmer (except for String).

Non-primitive types can be used to call methods to perform certain operations, whereas primitive types cannot.

Primitive types start with a lowercase letter (like int), while non-primitive types typically starts with an uppercase letter (like String).

Primitive types always hold a value, whereas non-primitive types can be null.

Examples of non-primitive types are Strings, Arrays, Classes etc.

Java Type Casting

==========

Type casting is when you assign a value of one primitive data type to another type.

In Java, there are two types of casting:

Widening Casting (automatically) - converting a smaller type to a larger type size

byte -> short -> char -> int -> long -> float -> double

Narrowing Casting (manually) - converting a larger type to a smaller size type

double -> float -> long -> int -> char -> short -> byte,  ex => double d=20.d  ; int s= (int) d;

SUBCLASS MEANS EXTENDS.

Escape sequences that are valid in Java are:

========

Code Result Try it

\n New Line

\r Carriage Return

\t Tab

\b Backspace

\f

Math.abs(-2,2) ==> 2 (positive);

int randomNum = (int)(Math.random() * 101); // 0 to 100

🔁 Summary of Java Execution Flow

Step Process

1. Write Code Developer writes Java code (.java file).

2. Compilation javac compiles .java → .class (bytecode).

3. Class Loading JVM loads .class into memory.

4. Verification JVM checks for bytecode security & validity.

5. Interpretation & JIT Compilation JVM executes bytecode (interpreted/JIT compiled).

6. Memory Management JVM allocates memory for execution.

7. Execution & Output The program runs and prints output to the console.

METHOD OVERLOADING

==========

SAME CLASSname ,same return type,diff no or type of parameters

blob:https://web.whatsapp.com/f40cea27-557a-4db5-9f7a-a715e49f70d4

SUBCLASS MEANS EXTENDS.

ARRAY VS

==========

ARRAY ===> FIXED SIZE,Search entire array for element by index,can store only 1 DATATYPE.

5. When to Use What?

✓ Use int when you need better performance and don’t need null values.

✓ Use Integer when working with collections (List<Integer>, Map<Integer, String>) or when null values are required.

🚀 Conclusion:

int is a primitive (faster, no null).

Integer is an object (has methods, can be null, slower).

STRING

======

String belongs to collection framework.

Memory of strings will be assigned in heap memory

String constant pool where strings will be stored in heap .if ther eis any common string it wont create new instead points to existing one

String will be immutable

Stringg buffer is mutable and ave operations like insert,append,mincapacity

SUBCLASS MEANS EXTENDS.

String builder is similar to buffer but te only diff is related to treads

STATIC

---

```
class hello {

    static string name;
        int marks;
}

h1=new hello();
hh2=new hello();
h1.name="srihhas";
```

print(h1.name,hh2.name) ==> Both will be same as we define in static it is not stored nin instance variable it will be stored in specific heap for all OBJ it will be same

static means

**Static method

==========

Can use static variables in static function but not non static variables .

Class Loader // static block

===========

```
class details {

        static {
                string name="SRIHAS";
                }
}
details d1=new d1();
```

SUBCLASS MEANS EXTENDS.

// when the obj is created 1) Class loaded => means static name,static block will get executed

2) objects will be created.

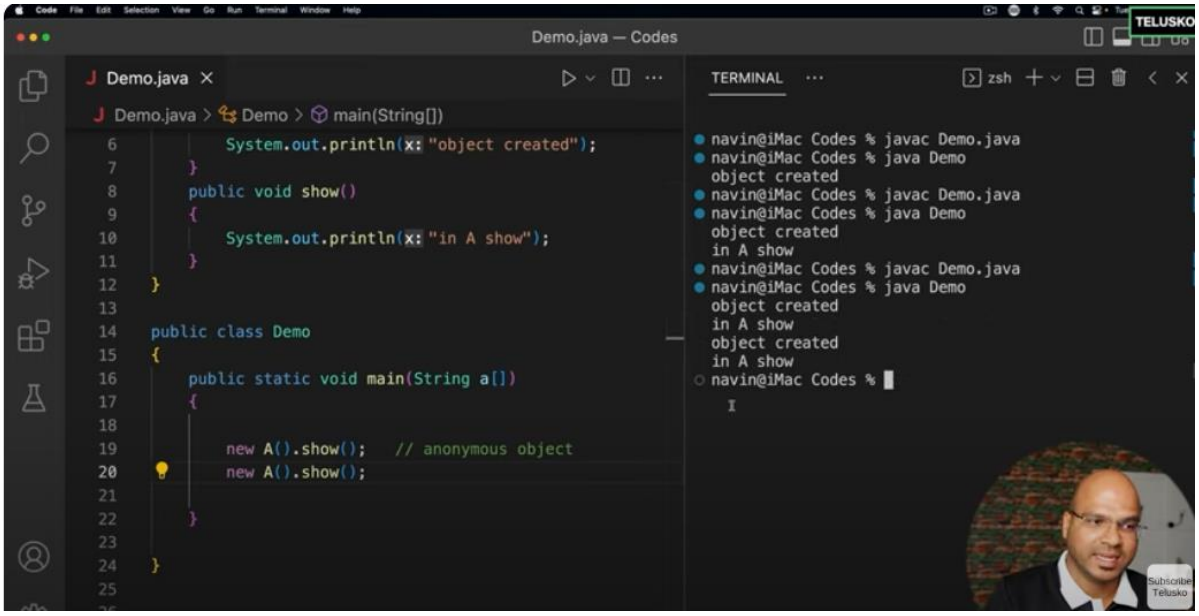
so no matter how many times you create objects the static block will be called once.

Reference vs object creation

Obj a; reference creation.

A=new Obj(); ➔ heap memory creation.

Anonymous object




Creation of object with out allocation of memory cant use twice the same obj.everytime new obhj will be created

SUBCLASS MEANS EXTENDS.

# OOPS

Inhheritance

```java
class Node {
  int val;
  Node next;
  Node prev;

  Node() {

  }

  public Node(int val, Node next, Node prev) {
    this.val = val;
    this.next = next;
    this.prev = prev;
  }

  public static void hello1() {
    System.out.print(s:"hello");
  }
}

class AdvHallo extends Node {
  public void advhello() {
    System.out.println(x:"Adv Hello");
  }
}
```

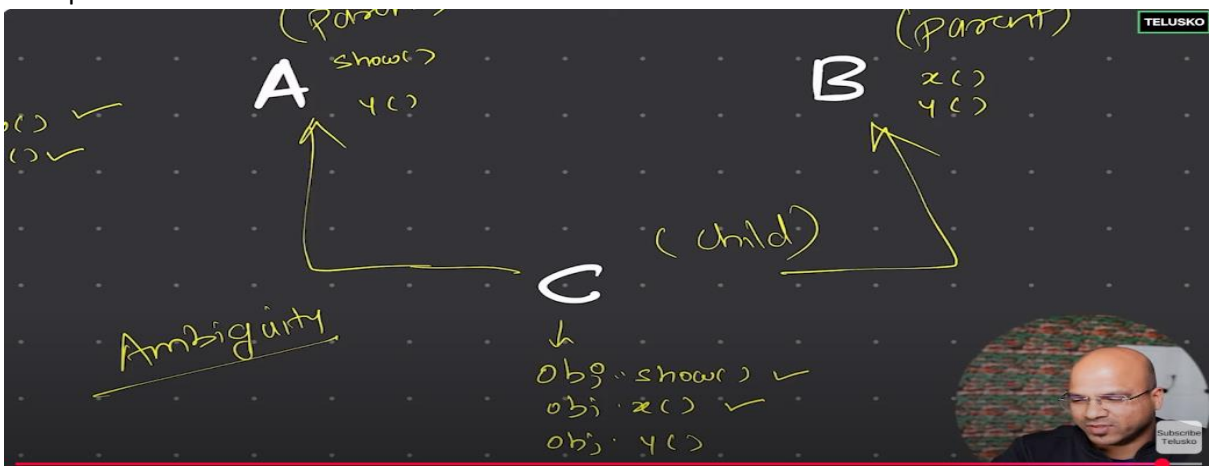
**Single level and multilevel inheritance**

AdvHello extends Node => **Single Level Inheritance**

If in case there is other class MoreAdv which extends AdvHello

MoreAdv extends AdvHello ➔ AdvHello extends Node ➔ M**ulti Level Inheritance**

Multiple Inheitance




Java Not Support Multiple inheritance as it is ambiguous to access y() function present in 2 parents.

SUBCLASS MEANS EXTENDS.

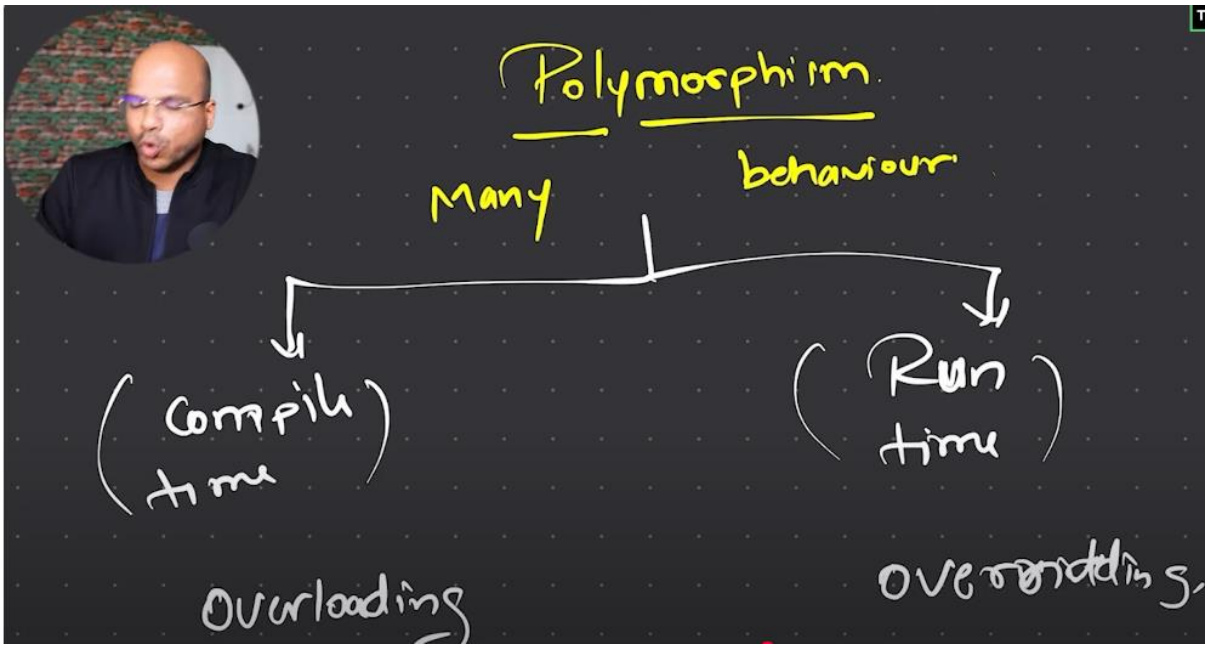
By default every constructor calls super().

```
Class A {
        A() {
            s.o.p("in A");
           }
       A(int a){
            s.o.p("Parametrised Comnstructor for A");
              }
}
Class B extends A{
            B() {
                Super();
                 Sop("in B");
               }
           B(int n) {
                Super();
                Sop('para in b');
                    }
}
```

B a=new B(); ➔ goes to default con of b then super

Ploymorphism

Many Forms




**Method Overloading**

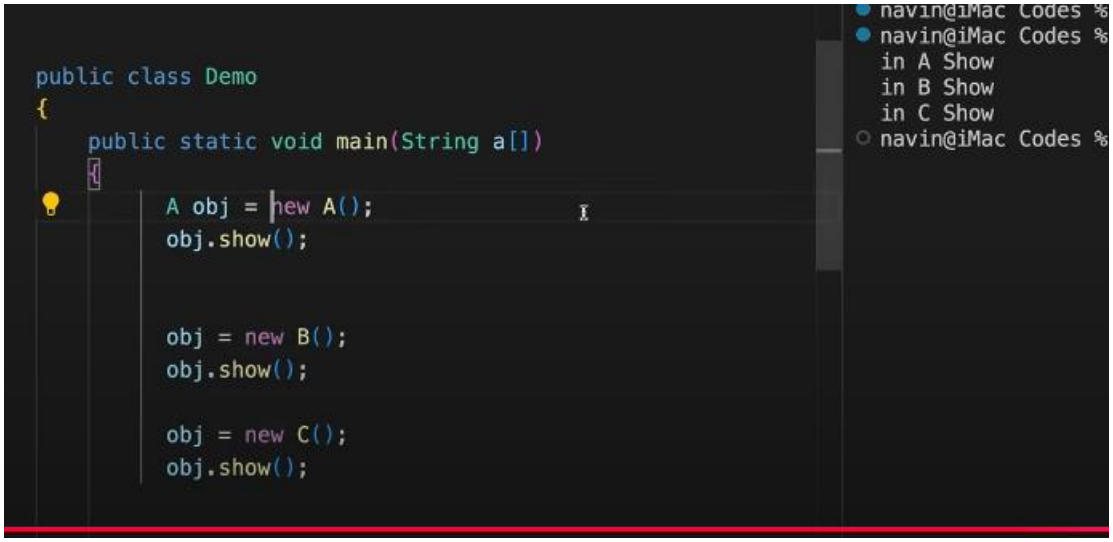
Means same method name but different in return types and no of arguments but within same class

**Method Overriding**

Same Method name and parameters but different class

```
class Calc
{
    public int add(int n1,int n2)
    {
        return n1+n2;
    }

}
class AdvCalc extends Calc
{
    public int add(int n1,int n2)
    {
        return n1+n2+1;
    }
}
```

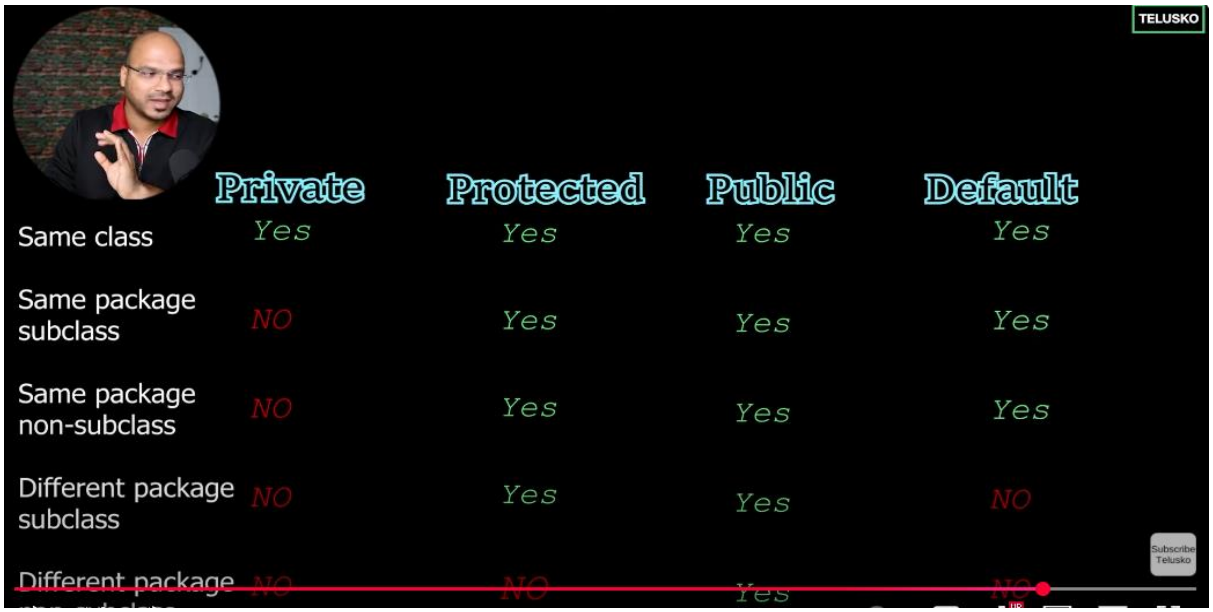
SUBCLASS MEANS EXTENDS.




DYNAMIC METHOD DISPACH OR METHOD OVERRIDING

UNTILL RUN TIME WE DON'T KNOW WHICH SHOW() METHOD IS CALLED EVERY TIME YOU ALLOCATED NEW MEMORY THAT SHOW IS CALLED .

ACCESS MODIFIERS




PRIVATE➔USED WITHIN CLASS

PUBLIC ➔USED IN SAME AND DIFF PACKAGES

PROTECTED➔USED UPTO DIFFERENT PACKAGE BUT ACCESSED IN SUBCLASS MEANS EXTENDINGclas

SUBCLASS MEANS EXTENDS.

FINAL KEYWORD

VARIABLE➔Once defined can’t change

CLASS➔ can’t be extended by other class

METHOD➔CAN’T BE OVERRIDED

WRAPPER CLASS

Java is not 100% OOPS because it has primitive data types

For collection we need classs not primitive data type so we use wrapper class

Int ➔ Integer  so on…. Fror every primitive data type

int a=2;

Integer b=a ;  ➔ Autoboxing means storing primitive dt to wrapper class

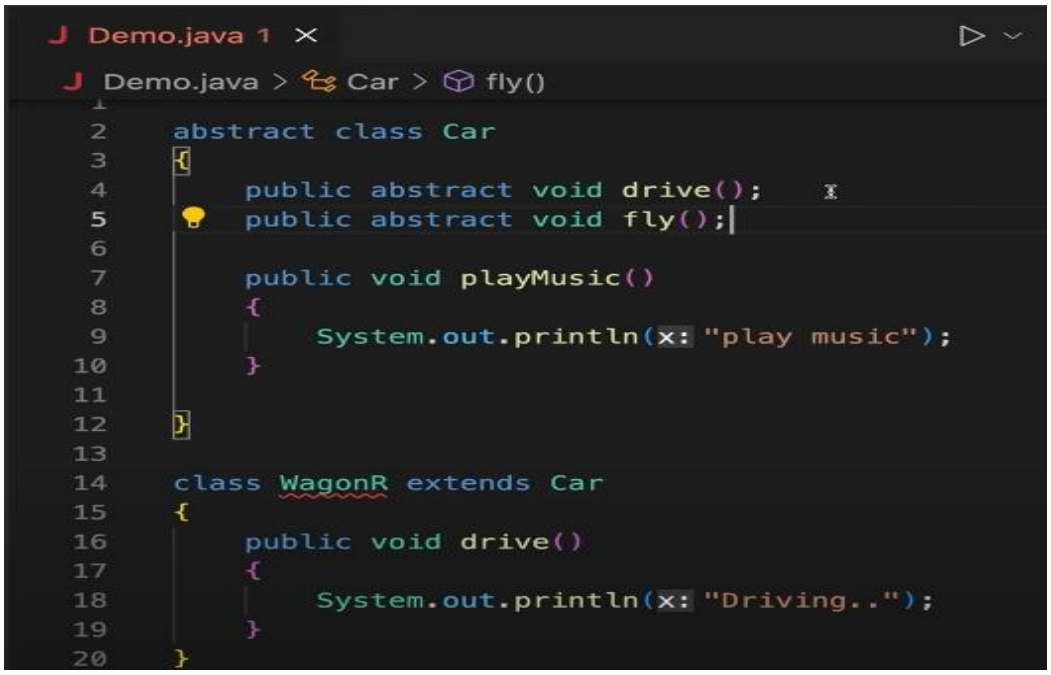
Int c= b;  ➔ AutoUnBoxing means changing from class to primitive data type;

ABSTRACT

IF YOU want to declare method but not to initiate then use abstract methos,**also can’t create ref to abClass and obj to it only by extending it we can do**

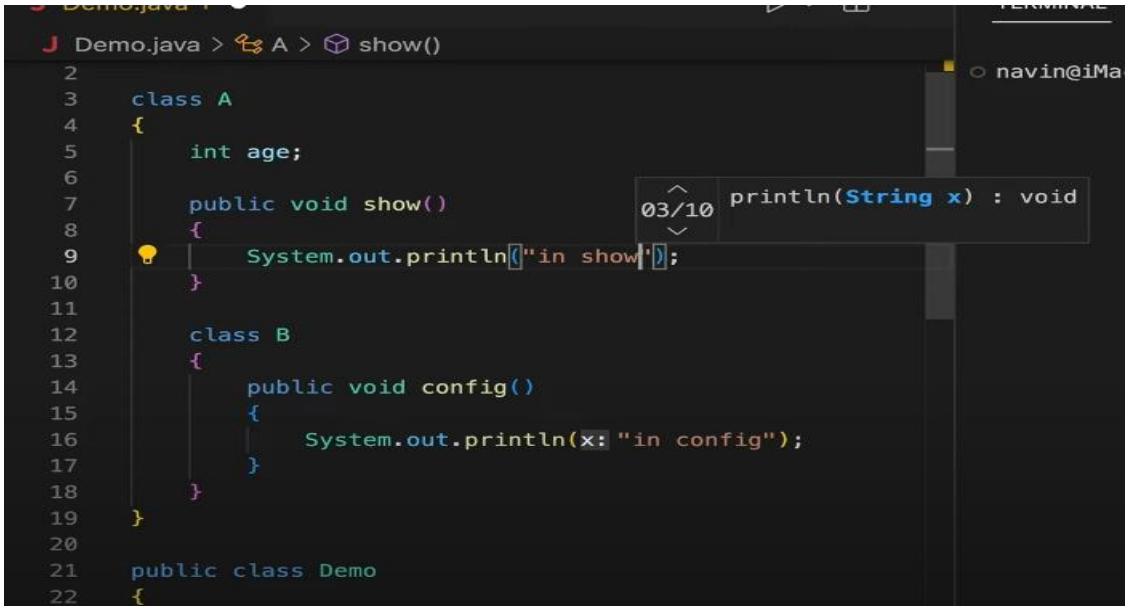
**ABSTRACT METHODS** SHOUD BE PRESENT ONLY IN **ABSTRACT CLASS**,NORMAL METHODS OR IMPLEMENTTED METHODS CAN ALSO BE PRESENT.

THE **OTHER CLASS** THAT EXTENDS **ABSTRACT CLASS** MUST IMPLEMENT THOSE **ALL ABSTRACT METHODS THAT ARE DEFINED IN ABSTRACT CLASS**.




SUBCLASS MEANS EXTENDS.

INNER CLASS




**For Non Static Inner** class we need to use outerclass objects so we create outer class obj then access

Outerrclass ou = new Outerrclass();

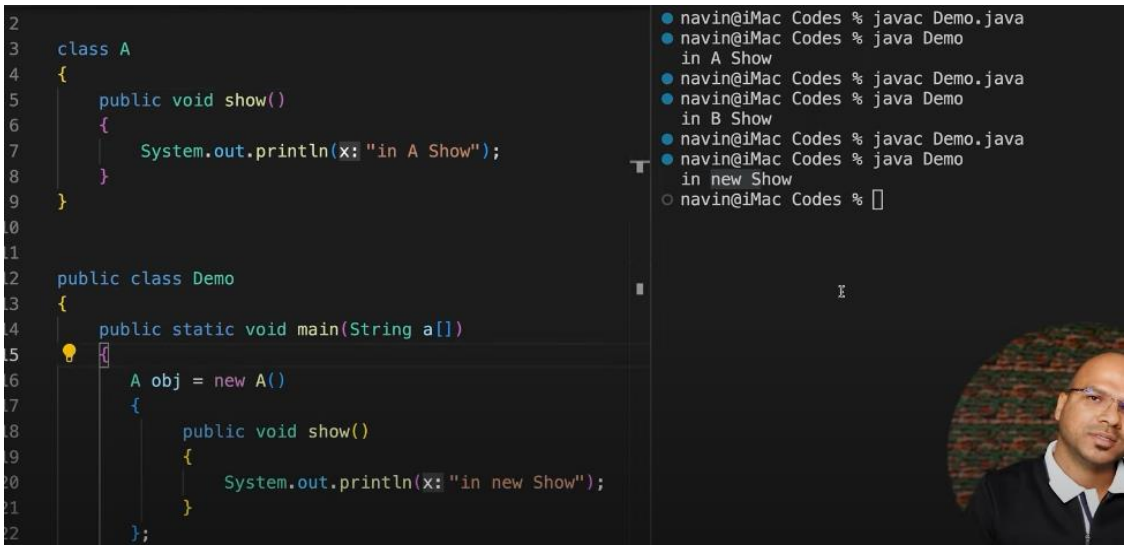
Outerrclass.Innerclass in = ou.new Innerclass();

in.show();

**But For Static Inner class** no need to create outer class obj to access as it is a static inner class

Outerrclass.Innerclass obj=new Outerrclass.Innerclass();

**Anonymous Inner class**

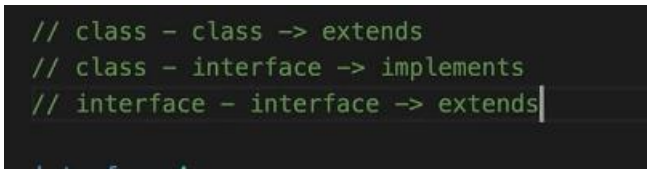



SUBCLASS MEANS EXTENDS.

INTERFACE

INTERFACE ONLY HAVE ABSTRACT METHODS

AND THE CLASS WHICH IMPLEMENTS INTERFACE SHOULD CALL ALL METHODS




```
package Interface;


interface Common {
  void hello();
}


class Inter implements Common {
  public int i = 5;


  public void hello() {
    System.out.print("In Inter");
  }


}


class Inter1 implements Common {
  public void hello() {
    System.out.print("In Inter1");
  }
}


class ii {
  public void pp(Common obj) {
    obj.hello();
```

SUBCLASS MEANS EXTENDS.

```
 }
}


public class Interf {
 public static void main(String[] args) {
  Common a = new Inter();
  Common bs = new Inter1();
  ii i = new ii();
  i.pp(a);
  // prints 5
 }
}
```

CAN IMPLENT TWO INTERFACES

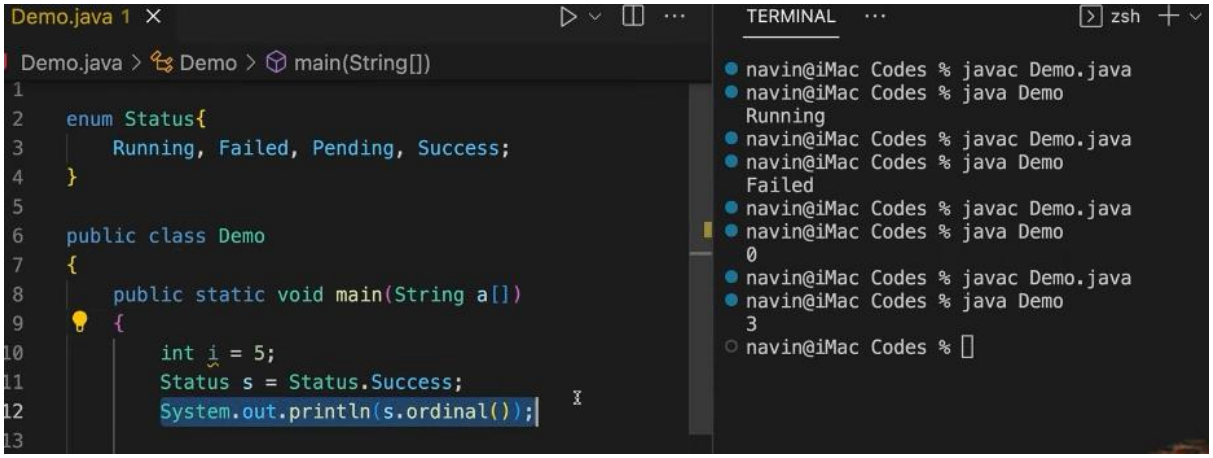
```
interface Common2 {
  void hello1();
}

interface Common1 extends Common {
  void hello1();
}

class Helo implements Common1, Common2 {
  public void hello() {
    System.out.print(s:"helo");
  }

  public void hello1() {
    System.out.print(s:"Hello World");
  }
}

class Inter implements Common {
  public int i = 5;
```

HERE unlike we cant extend 2 classes so multiple inheritance fails

- **But in Interfaces we can implement 2 interfaces as they don't have any definition there won't be ambiguity problem in interfaces**

SUBCLASS MEANS EXTENDS.

**ENUM**




Enum means it is also a class inside there will be named constants instead of using string or char.

```
public class Enum {
    Run | Debug
    public static void main(String args[]) {
        enum Laptop {
            Ideapad(price:2000), macbook(price:100), Lenovo(price:3000);

            int price;

            Laptop(int price) {
                this.price = price;
            }

        }
        ;
        Laptop e1 = Laptop.Ideapad;
        System.out.print(e1.price);
    }
}
```

SUBCLASS MEANS EXTENDS.

## Annotations

Annotation means interacting with compiler and find errore during compile time itself.Gives red line

@override ➔ Used to override the method

@ FunctionalInterface ➔ Restrict to have only one method inside interface.

## Lambda Expressions

Should be used with @ FunctionalInterface

```
package LambdaExp;

interface A {
  public void show();
}

interface B {
  public void show(int i);
}

public class Lambda {
  public static void main(String args[]) {
    // 1st--> Way of defining Lambda Expressions
    A obj = new A() {
      public void show() {
        System.out.println("Hello Lambda");
      }
    };
    obj.show();
    // 2nd--> Way of defining Lambda Expressions
    A obj1 = () -> System.out.println("Hello in Lambda2");
    obj1.show();
```

SUBCLASS MEANS EXTENDS.

```
  // Lambda expr with parameters
  B obj2 = new B() {
    public void show(int i) {
      System.out.println("Hello Lambda" + i);
    }
  };
  obj2.show(5);

  B obj3 = (i) -> System.out.println("Hello in Lambda2" + i);
  obj3.show(10);
 }

}
```

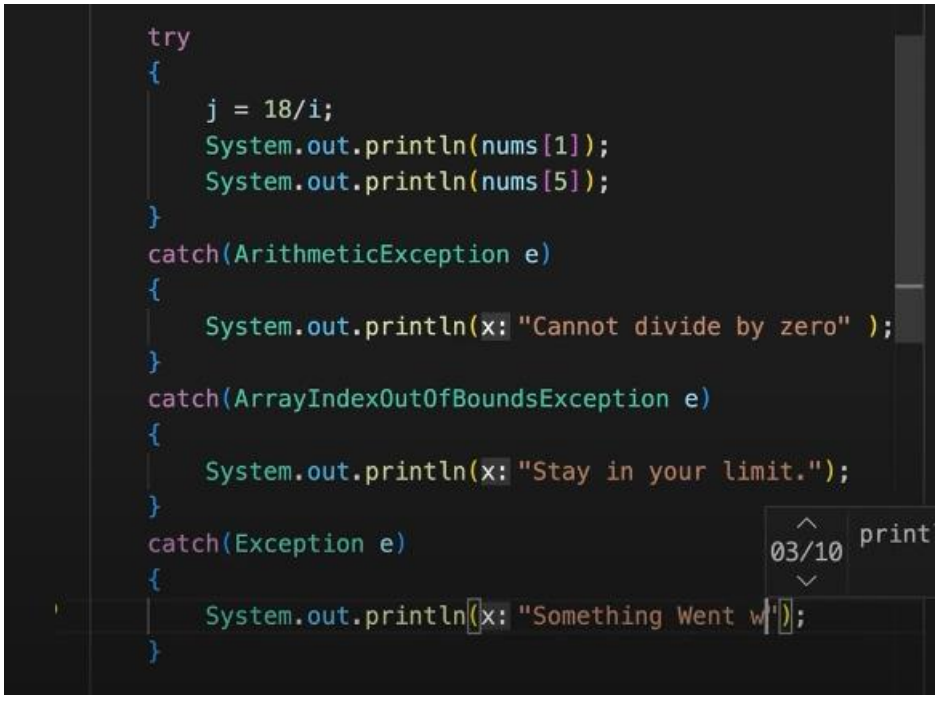
**Types of Interfaces**

Normal Interfaces ➔ Having multiople methods

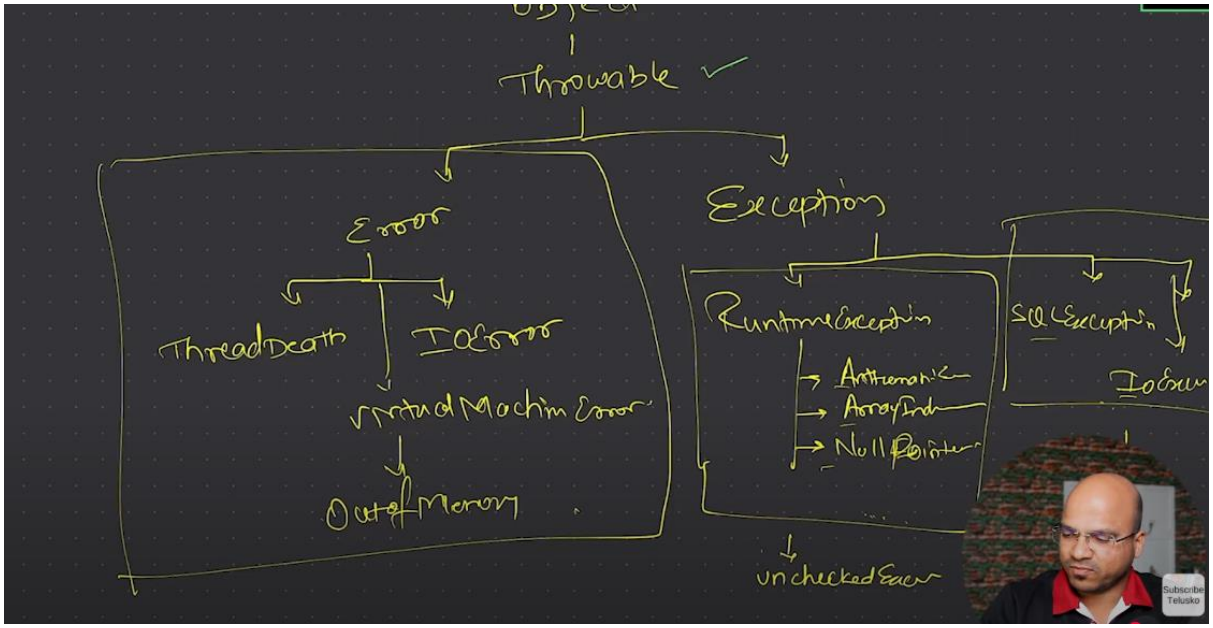
Functional Inter or SAM ➔ Single abstract Method

Marker Interfacxe ➔ No Methods used for serializing and deserializing

**Exception**




SUBCLASS MEANS EXTENDS.




In Exception we have checked and unchecked

Unchecked Exception➔Means not force you to handle the exception.

Checked Exception ➔ Means You should Handle the exception like sql exception ,etc

Errors➔ Something you can’t handle .

**For More Reference go through the git hub repository**

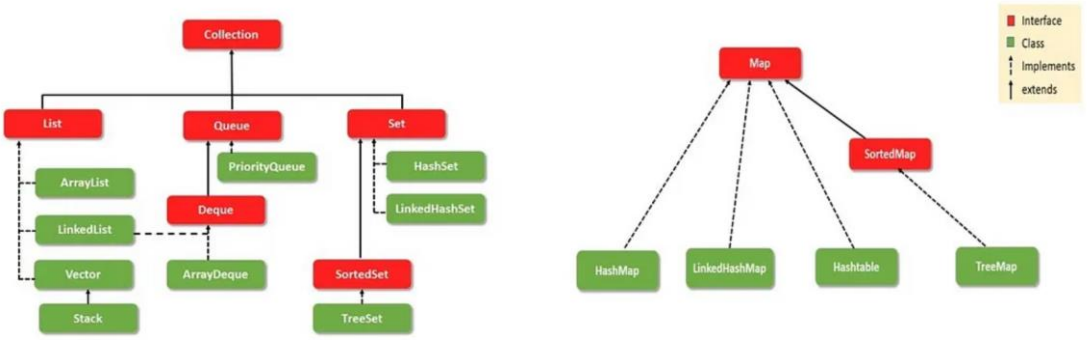
**Link**: https://github.com/srihas-sa/CoreJava-DSA.git

**Input/Output**

```java
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;

public class Inou {
    Run | Debug
    public static void main(String[] args) throws NumberFormatException, IOException {
        // 1st Way of taking input older versions
        InputStreamReader in = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(in);
        int n = Integer.parseInt(br.readLine());
        System.out.print(n);
        // @nd way
        Scanner sc = new Scanner(System.in);
        int n1 = sc.nextInt();
        sc.nextLine();
        String s = sc.nextLine();
        System.out.print(s);
    }
```

SUBCLASS MEANS EXTENDS.

COLLECTION FRAMEWORK IN JAVA




Collection is a interface which have methods like add etc which will be implemented by all classes

Collection extends by other interfaces which provide Specific Operation usefull for that Interface

****That extended Interface will be implemented by different classes no need to define or create anonymous inner class for the interface we have build in classes which will implement those methods or operations inside that perticular interface . that's why we call it Collection API

Set

Hashset ➔ Unique ,UnSorted

Treeset➔Unique,Sorted.

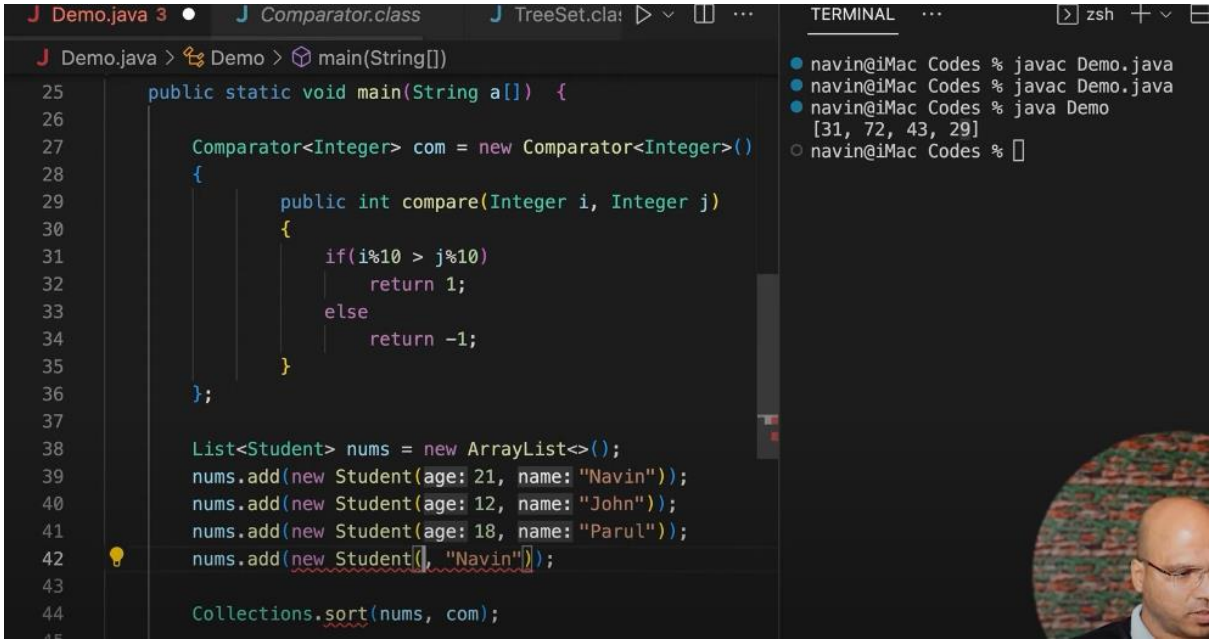
Map

HashMap➔Normal

HashTable➔Synchronous like threads

We can use Collections to sort

SUBCLASS MEANS EXTENDS.




We can use Collections to sort by default it will sort

Collections.sort(nums);

But if you want to implement your own logic like compare unit's place and sort we need to implement that method called Comparator which will be used to Sort based on our logic

SUBCLASS MEANS EXTENDS.

## ***STACK AND HEAP MEMORY MANAGEMENT AND GARBAGE COLLECTION(GC)

Void main{

Int a=2;  //Stores in Stack memeory in main method scope

Class a = new Class; // Ref will be stored in main scope but address in **heap memory**

a.call()  // calls and from here on New Scope will be created in stack.

}

Class a{

Public void call()  // Stores in scope of call method

}

**Once execution of scope is done like after closing Bracket stack will be freed in Last in First Out Manner Even references will be deleated.**

**So first call method scope will be deleated**

**Then Main method and it's Referenceees**

**Then in Heap Memory Not referencing Objects will be deleted Automatically by Guarbage collector**

System.gc();  // JVM decide when it runs even though you executed command

Based on space it will be running accordingly.

Strong Reference ➔ How we create Class a = new Class; can't be deleted until ref cleared by garbage coll

Weak Reference➔ WeekReference<Person> p=new … ➔ As soon as GC runs it will be cleared

Soft Reference➔ Similar but removed only if it is urgent like no memory .

Heap memory space Young age,old generation  and metadata

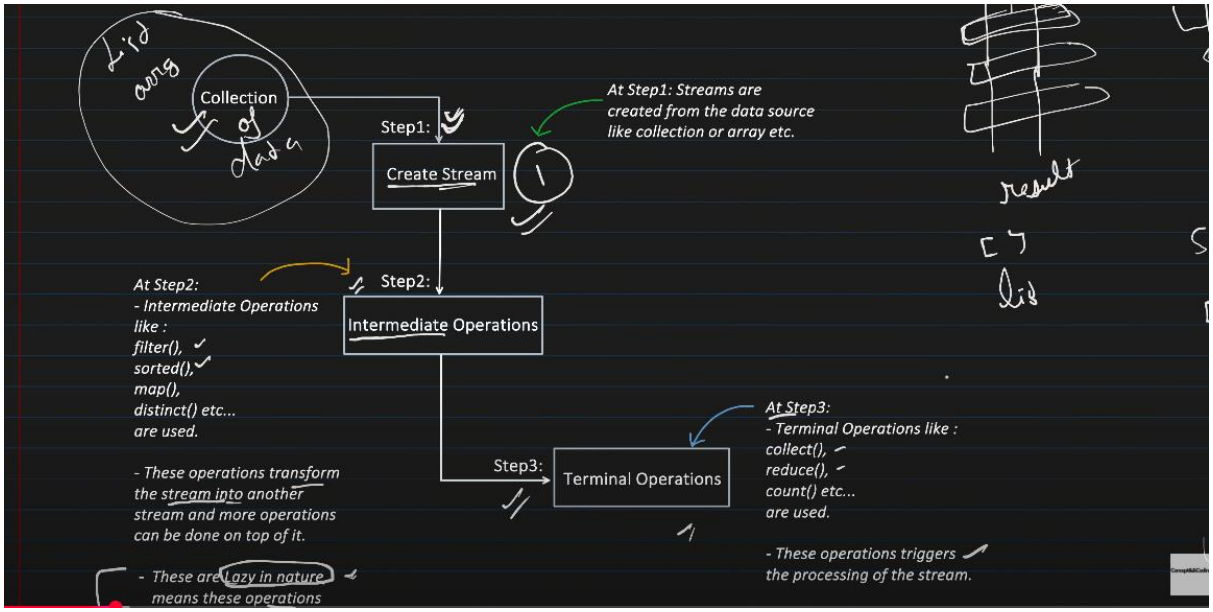
Means if obj is present in longer time it is stored in old or else in new

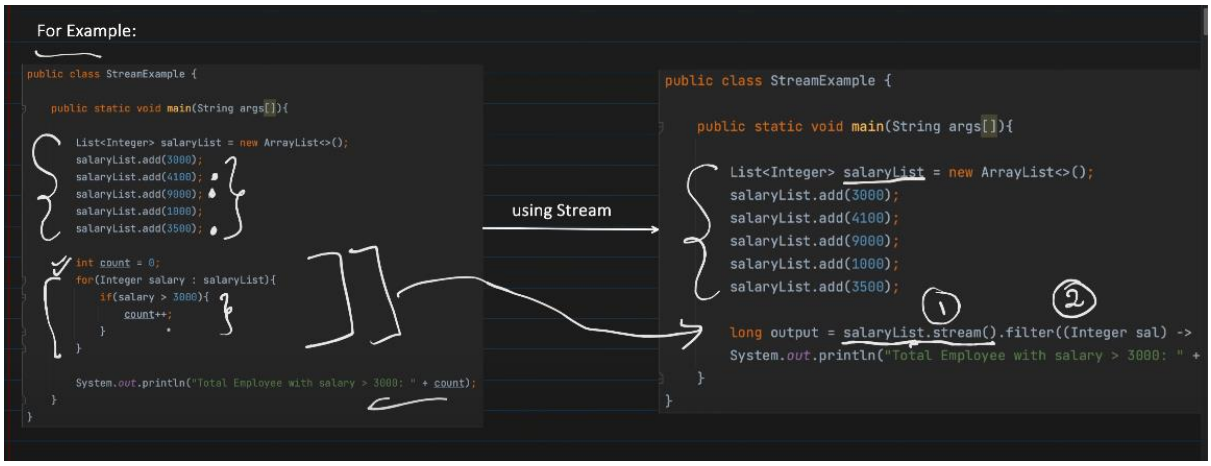
Mark and Sweep algo, in young gen gc will run more freq than old gen

Meta data ➔ All static data,class data will be stored,also called class loader

SUBCLASS MEANS EXTENDS.

STREAMS

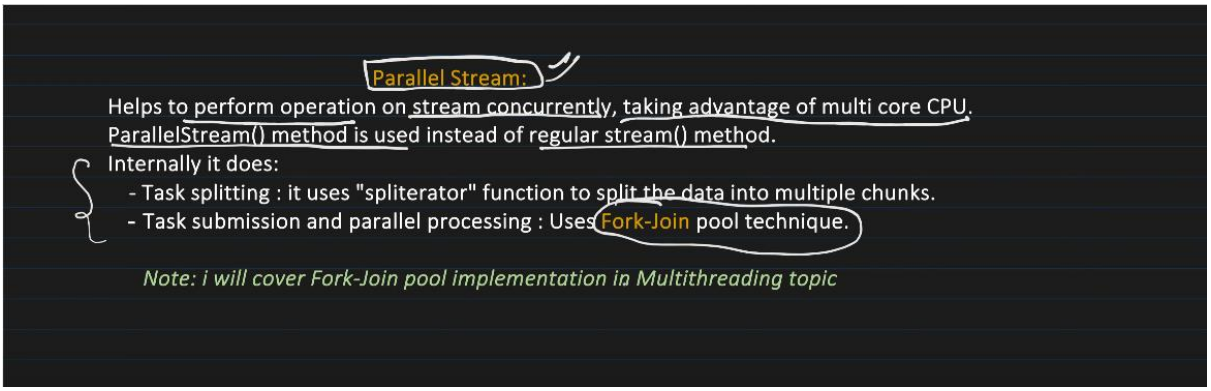



***Only be executed once the terminal operation is called***

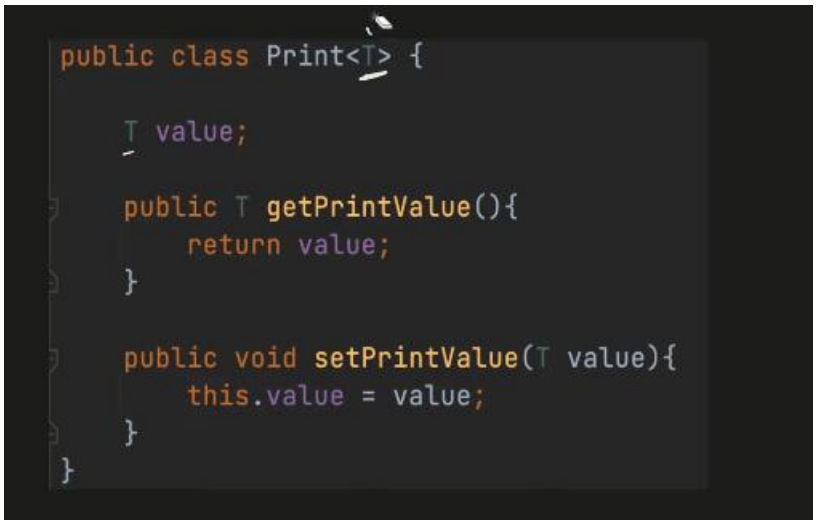
***Filter,map, these all are intermediate operations they are called lazy because they wont be executed without terminal operations like count,toArray() ect …***


SUBCLASS MEANS EXTENDS.




***Work faster than Normal Stream because it will divide task into sub tasks.***

***GENERIC CLASSES***



```
public class Print<T> {

    T value;

    public T getPrintValue(){
        return value;
    }

    public void setPrintValue(T value){
        this.value = value;
    }
}
```

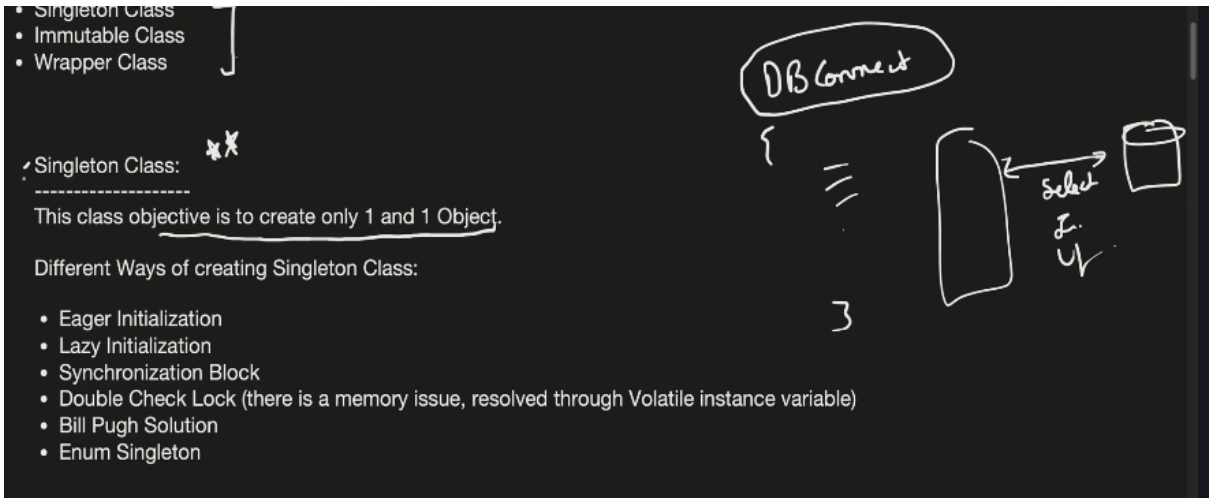
***Print<Integer> ii=new Print<>();***

SUBCLASS MEANS EXTENDS.

# ****SINGLETON CLASS****

***Have Single class and Single Object***

***For Db Connection we just need one instance don't want multiple obj so then we use these ,.***




## *Eager Initialization*

***Here we create static method to create new object and make constructor as private so we can't create object with new keyword***

***During loading itself it will load the static methods and create object whether you use it or not that is disadvantage.***

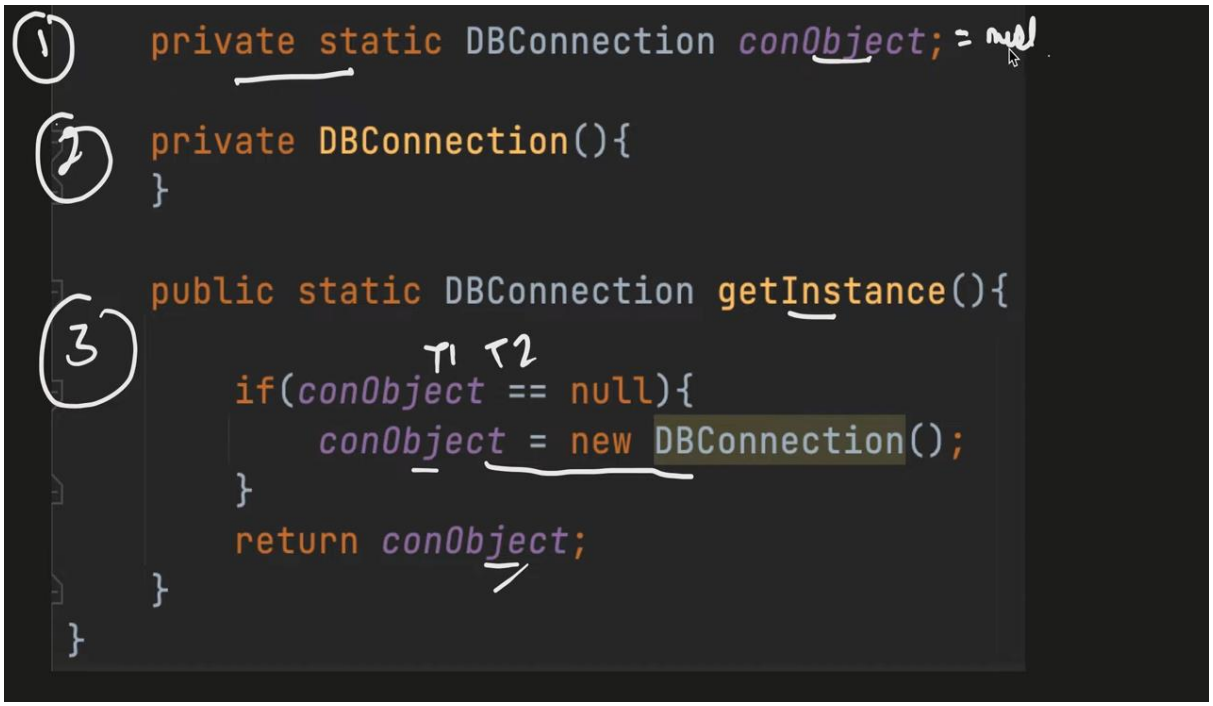
```
public class DBConnection {

    private static DBConnection conObject = new DBConnection();

    private DBConnection(){
    }

    public static DBConnection getInstance(){
        return conObject;
    }
}
```

```
public class Main {

    public static void main(String args[]) {

        DBConnection connObject = DBConnection.getInstance();
    }

}
```

SUBCLASS MEANS EXTENDS.

## Lazy Initialization

**Object is Created Not in static block in main method if it is Not Null then New Obj will be created**




Advantages

Object is not created Before head

Disadvantages

If Multiple Thread come at a time and if obj is not already Created both will create New Objects();

## Synchronization Block

Same as above but Just as Synchronized Before the method getInstance() so only one thread ast a time can use that function

Lock and Unlock Method so no multiple threads can use

**Advantages**

Will not create more than one object

**Disadvantages**

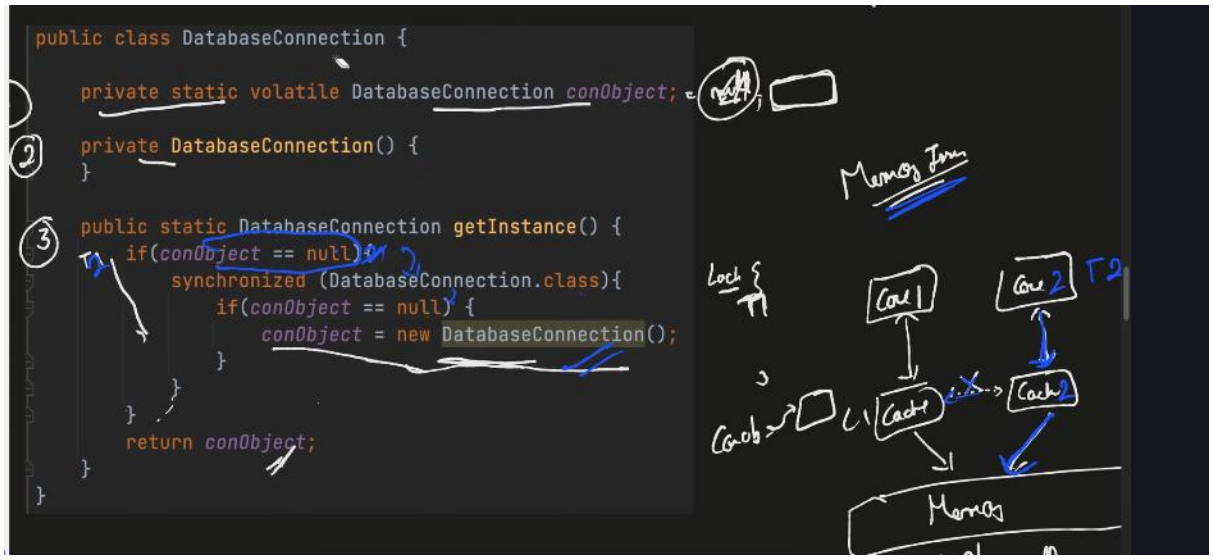
Too Slow because of locking and unlocking.

SUBCLASS MEANS EXTENDS.


Here We use Double check even if two threads enters to method 1st only one will go to inside creation.

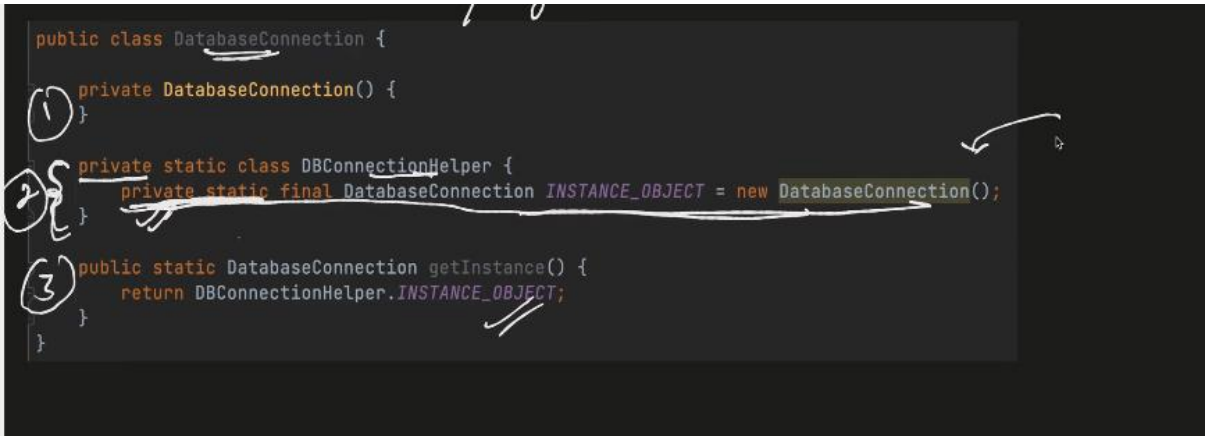
Issue with Memory

Here in above without volatile keyword the memory allocation will happen in catch not in main memory so it will create 2 obj in two catch so to avoid catch and directly work with main memory we use VOLATILE




SUBCLASS MEANS EXTENDS.

## Bill Plug Solution

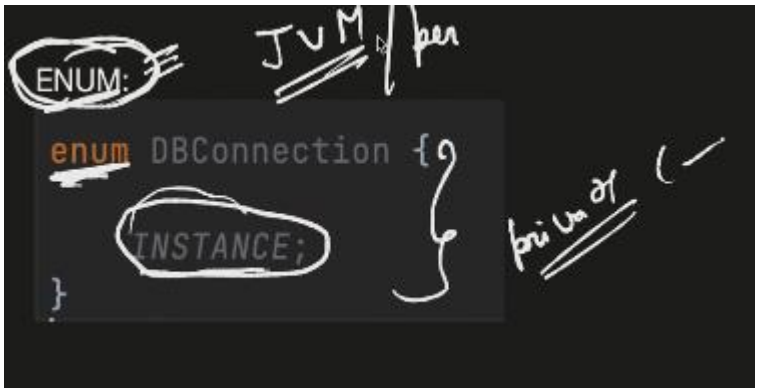



Here we use **static inner class**

Whenever Getinstance() is called for the 1st time static inner class will be executed and create new object

As it is static inner class it will load directly with out creating again.

## ENUM

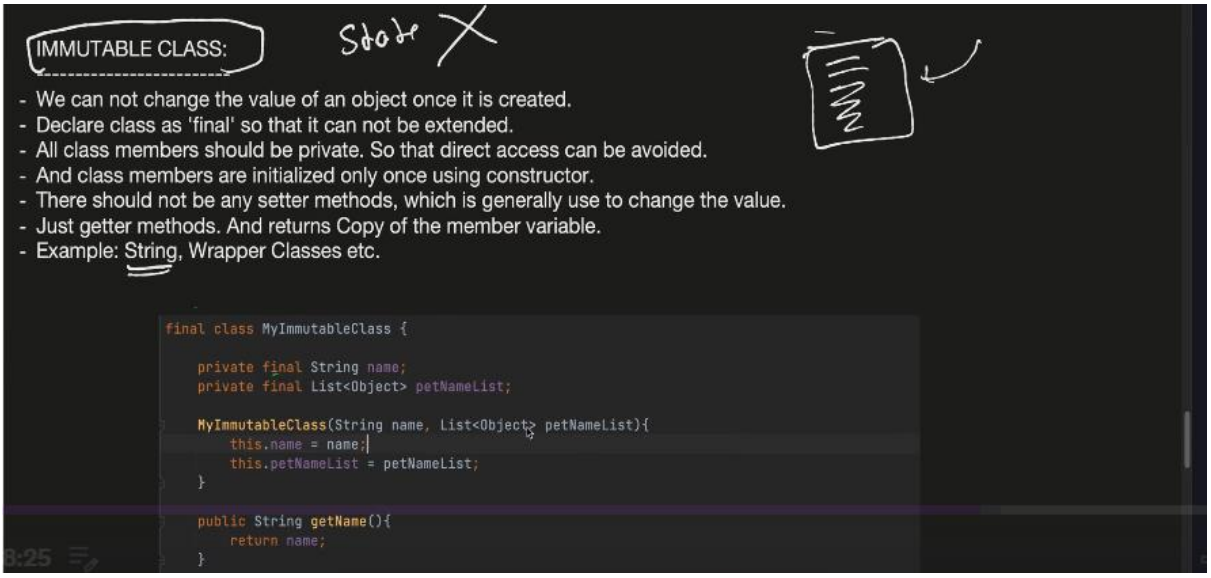



By default Enum has Private Constructor and only one instance will be present

So Enum is SingleTon Class.

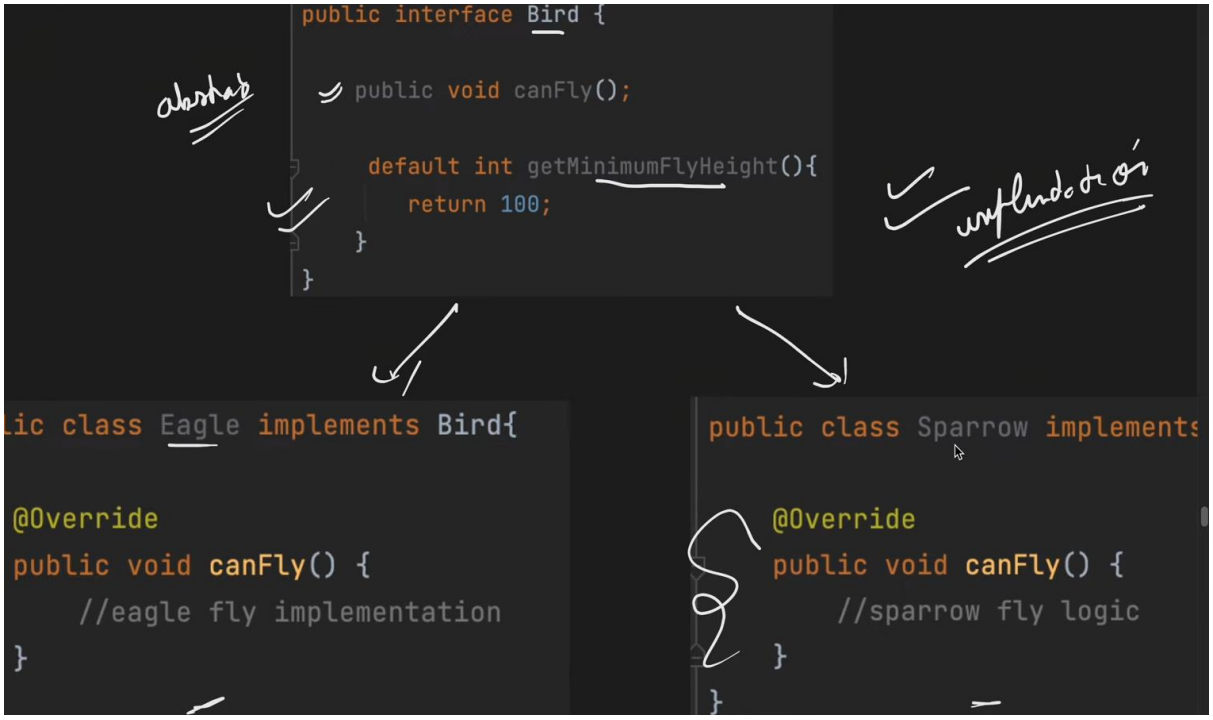
SUBCLASS MEANS EXTENDS.

***Immutable Classes*** caan't be changed and modified and inherited




No Setter ,Final class,Final Variables

## Java8 Features Default Method in Interfaces




Now we can provide **implementation to the Method in Interface using default Keyword.**

SUBCLASS MEANS EXTENDS.

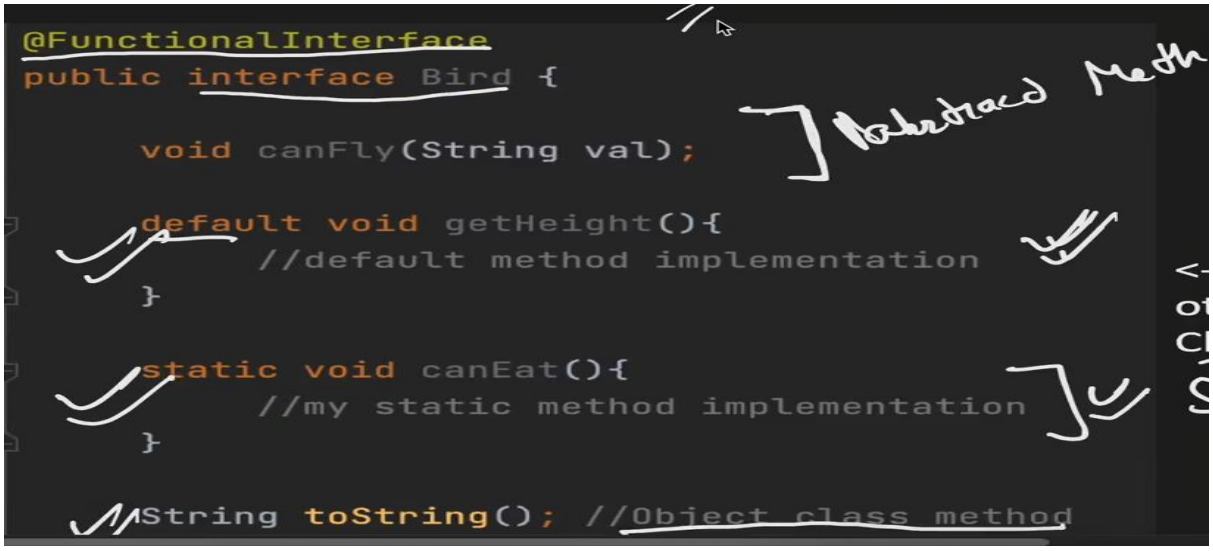
## Interface

### 3. Private Method and Private Static method (Java9):

- We can provide the implementation of method but as a private access modifier in interface.
- It brings more readability of the code. For example if multiple default method share some code, t
- It can be defined as static and non-static.
- From Static method, we can call only private static interface method.
- Private static method, can be called from both static and non static method.
- Private interface method can not be abstract. Means we have to provide the definition.
- It can be used inside of the particular interface only.

```
public interface Bird {

    void canFly(); //this is equivalent to public abstract void canFly();

    public default void minimumFlyingHeight() {
        myStaticPublicMethod();  //calling static method
        myPrivateMethod(); //calling private method
        myPrivateStaticMethod();   //calling private static method
    }

    static void myStaticPublicMethod() {
        myPrivateStaticMethod(); //from static we can call other static method
    }

    private void myPrivateMethod(){
        // private method implementation
    }

    private static void myPrivateStaticMethod(){
        // private static method implementation

    }
```

Functional Interfaces




***Functional Interfaes should have one abstract method and n number of default and static methods as they will have implementation they will no longer be abstract methods.

Types of Functional Interfaces


Left one is Consumer

Right One is Supplier =>get from method

1)Consumer ➔Accepts Value Not return

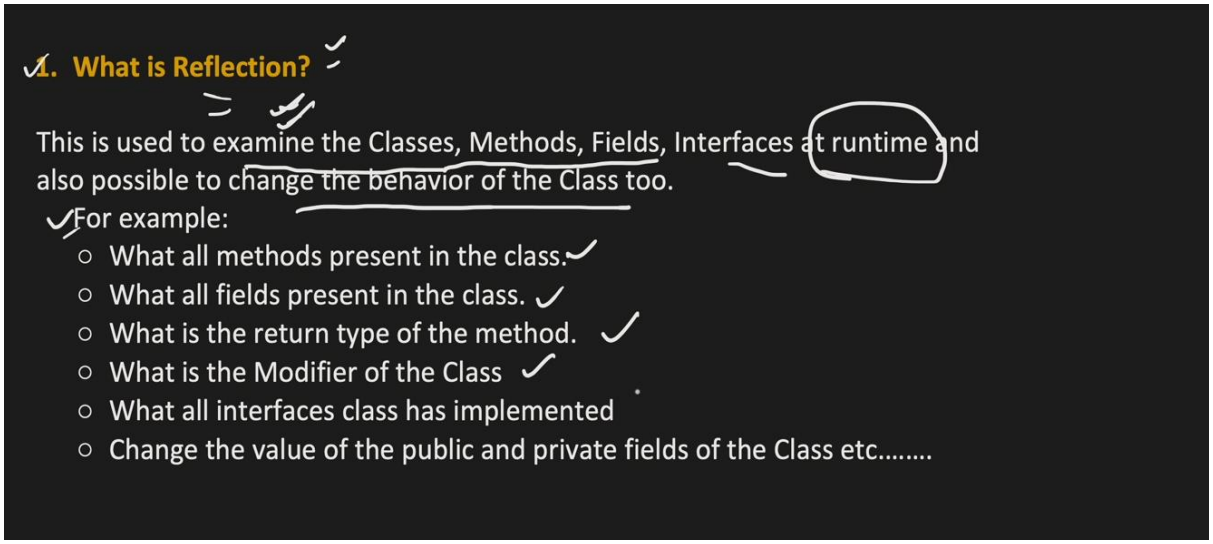
2)Supplier➔Returns Value

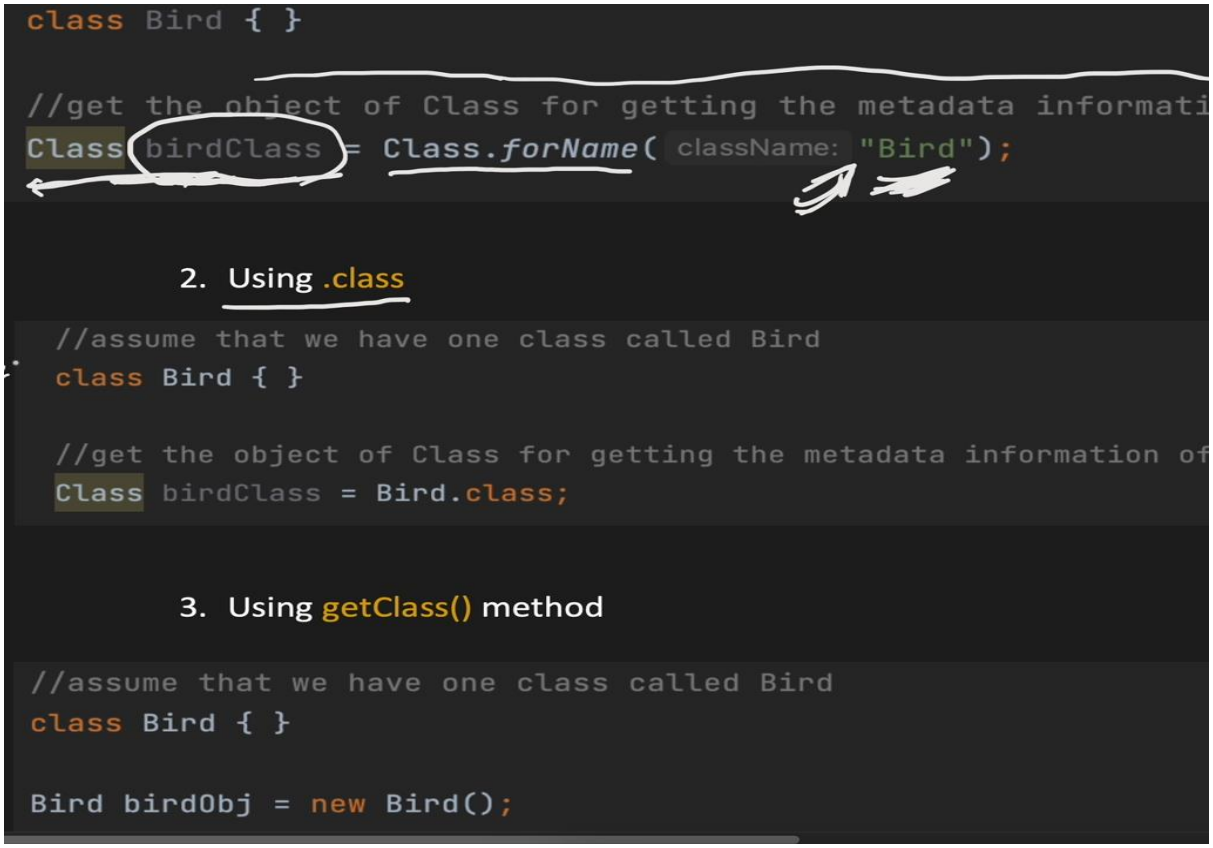
3)Function➔Accepts and return value

4)Predicate➔Return Boolean

SUBCLASS MEANS EXTENDS.

Reflection Class




Reflection Classes Shows the methods ,parameters,fields thhta are present in the class

We can create the Refletion class by using Class Object.

Internally for Every Class one Class obj will be created we can access by above methods.

SUBCLASS MEANS EXTENDS.


By using reflector class you can even change and access private methods and variables

SO you don't have access specifications in relection

That's why singleton CLASS ALSO NOT APPLICATE AS ITS CONSTRUCTOR IS PRIVATE WE CAN CALL IT BY CRETING CLASS AND CHANGE ACCCESS SPECIFICATION AND CALL THAT CONSTRUCTOR WHICH IS PRIVATE.

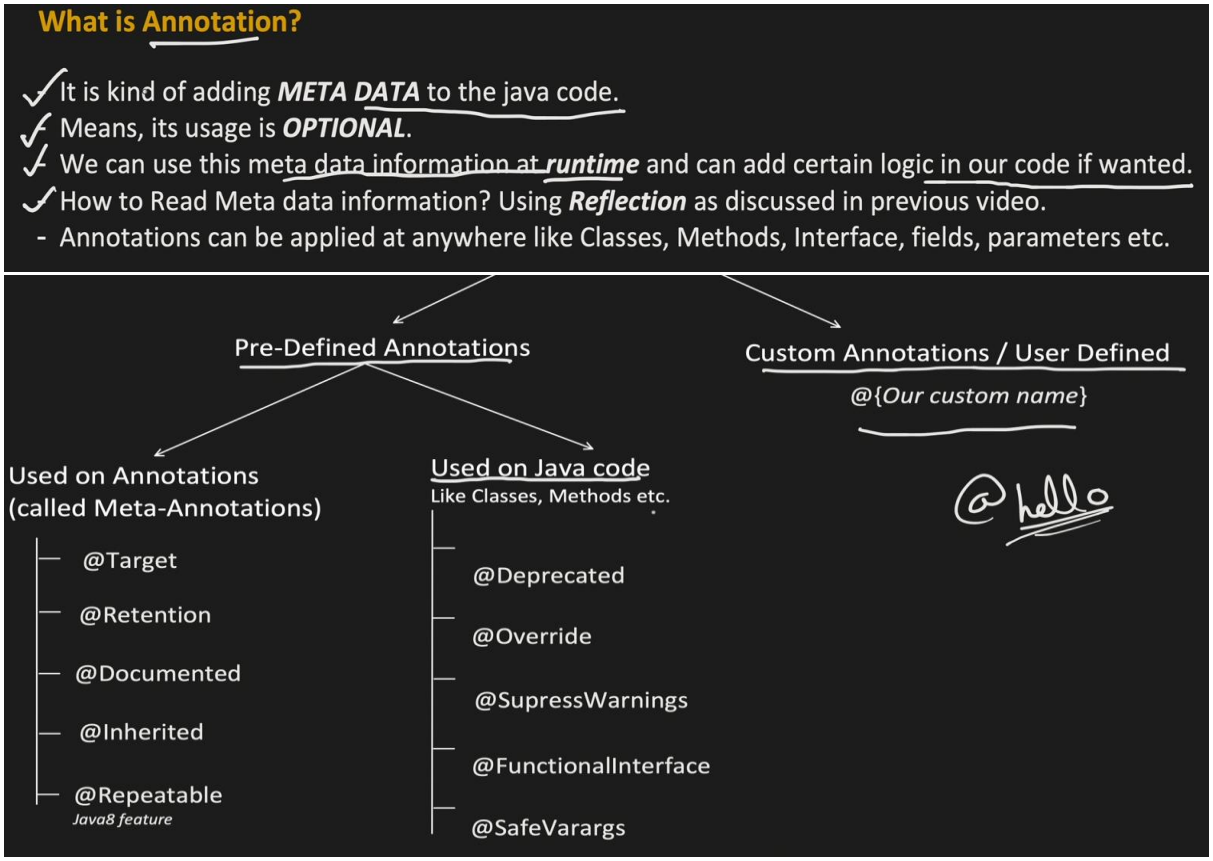
NOT RECOMMEND USING IT

BREAK SINGLETON BY BREAKING PRIVATE

---

SUBCLASS MEANS EXTENDS.

ANNOTATION




@Deprecated : Shows Warning when ever you used that particular method,filed,class etc..

@SupressWarning : Do not give warnings including depreciated warnings

Meta Annotation

Annotation over Annotatin

Like

@Target

@Override

Target=@ElementType.Method ➔means used only on method

@Retension (RetentionPolicy.**Source**)➔ Present only in **Source code** not in class file

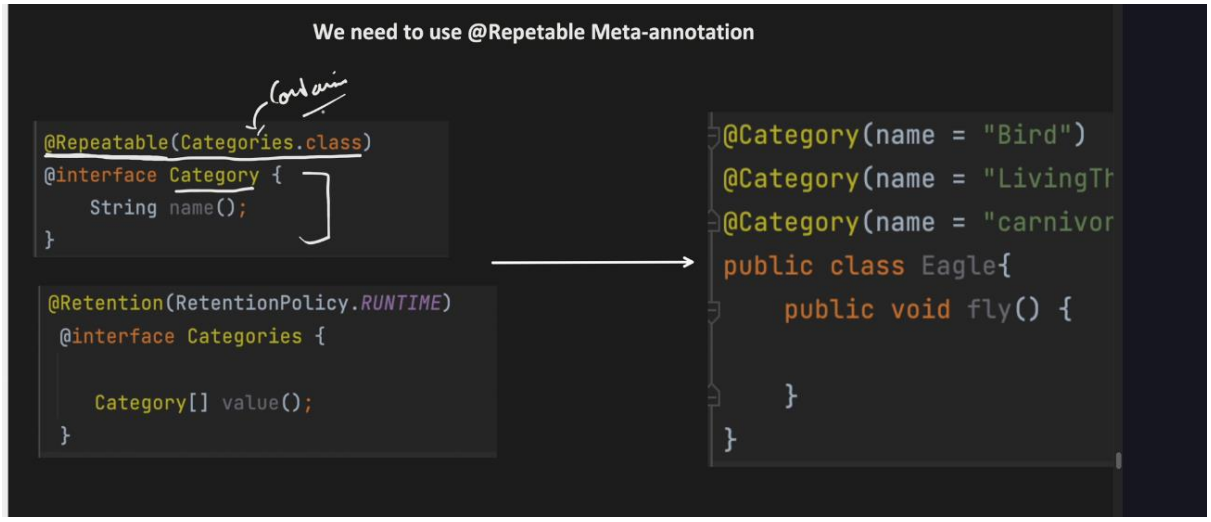
(RetentionPolicy.**Class**)➔ Present **Class file but ignored by jvm at RunTime**.

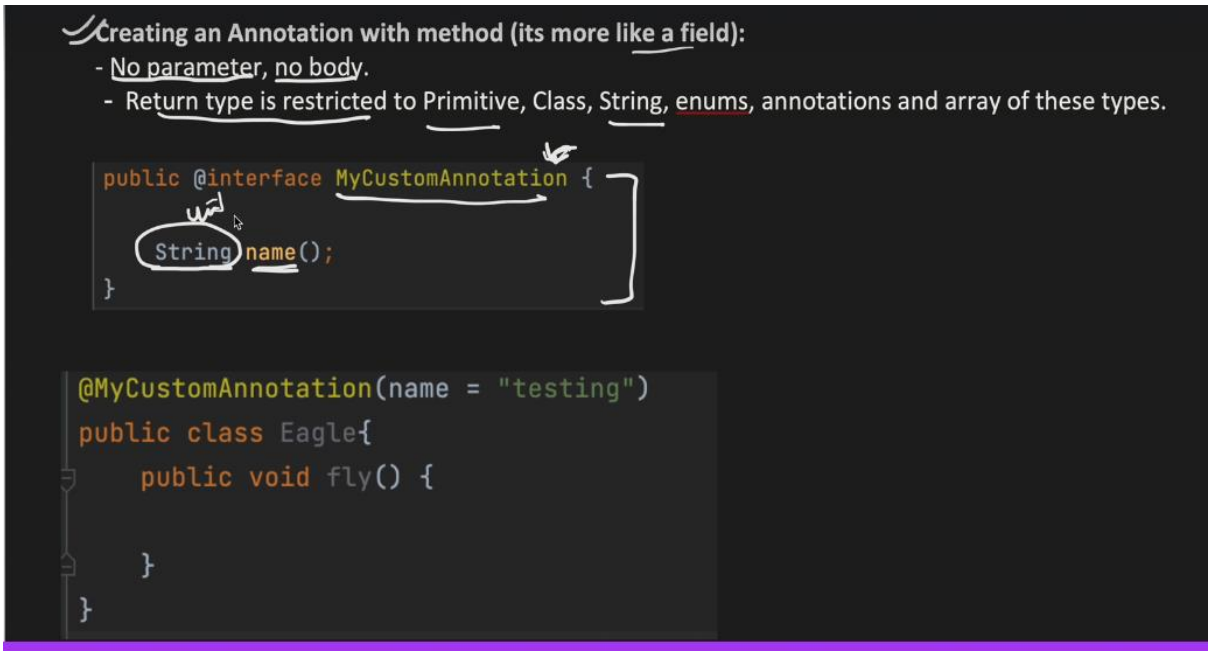
@Retension (RetentionPolicy.**RunTime**)➔During Run time also it will be present.

@Documented➔If you want to show annotation in java Document use it.

@Inherited➔Child class also can able to use the parent class Annotations.

SUBCLASS MEANS EXTENDS.




| .NO | METHODS | Available in | USAGE |
|---|---|---|---|
| 1. | size() | Java1.2 | It returns the total number of elements present in the collection. |
| 2. | isEmpty() | Java1.2 | Used to check if collection is empty or has some value. It return true/false. |
| 3. | contains() | Java1.2 | Used to search an element in the collection, returns true/false. |
| 4. | toArray() | Java1.2 | It convert collection into an Array. |
| 5. | add() | Java1.2 | Used to insert an element in the collection. |
| 6. | remove() | Java1.2 | Used to remove an element from the collection. |
| 7. | addAll() | Java1.2 | Used to insert one collection in another collection. |
| 8. | removeAll() | Java1.2 | Remove all the elements from the collections, which are present in the collection passed in the parameter. |
| 9. | clear() | Java1.2 | Remove all the elements from the collection. |
| 10. | equals() | Java1.2 | Used to check if 2 collections are equal or not. |
| | stream() | | |

SUBCLASS MEANS EXTENDS.

COLLECTIONS

QUEUE ➔DEQUEUE ➔Double Sided Queue

```java
package Collections;

import java.util.ArrayDeque;
import java.util.Collection;
import java.util.Deque;
import java.util.Queue;
import java.util.concurrent.DelayQueue;

public class Dequeue {
    Run | Debug
    public static void main(String[] args) {
        Deque<Integer> cl = new ArrayDeque<>();
        cl.add(e:1);
        cl.addFirst(e:0);
        cl.addLast(e:2);
        cl.removeFirst();
        cl.removeLast();
        cl.getFirst();
        cl.getLast();
        System.out.println(cl);
    }
}
```

Insertion T.C ➔ O(1) Mostly except when size got full it copies to new deque with double size then it wil be O(n);

Deletion ➔O(1)

Search ➔O(1)

**List**

CAN BE INSERTED AND DELETED IN MIDDLE IT IS INDEX BASED UNLIKE QUEUE .

INTERNALLY USES ARRAY DATA STRUCTURE

```java
ListIterator<Integer> li = cl1.listIterator();
while (li.hasNext()) {
    System.out.println(li.next());

}
while (li.hasPrevious()) {
    System.out.println(li.previous());
}

// if you want to write your own logic for sorting then
System.out.println(cl1);
```

SUBCLASS MEANS EXTENDS.

Can travel backward through ListIterator()

**LinkedList**

---

Implement Both Queue and List

Can addfirst,last,search,middle,set,get ….

Insertion T.C ➔ O(1);

Search ➔ O(n);

VECTOR

---

Same As ArrayList

But ThreadSafe

Have locks

****HASHMAP******

---

Internal Implementation of Hashmap

Node {  hash:,key:,val;next}

Internally there will be ARRAY [16]

Map.put(1,20) ➔ 1 will go to hasing ➔12002(hashValue)➔HashVal%**LenArray➔Store in that index**

**If you get Same val with other key then it will go the same index but now it already have value so it will store in .next (LINKED LIST) .**

**Internally it uses load Factor if all are filled then 2x the storage and refactor again**

**If there were More than theroshold val linked in a linked list then**

**Linked List will be converted to Balanced Binary tree so from O(n) to O(logn)**

**Best Case Or Average Case**

---

**Insertion,deletion,searh:O(n)**

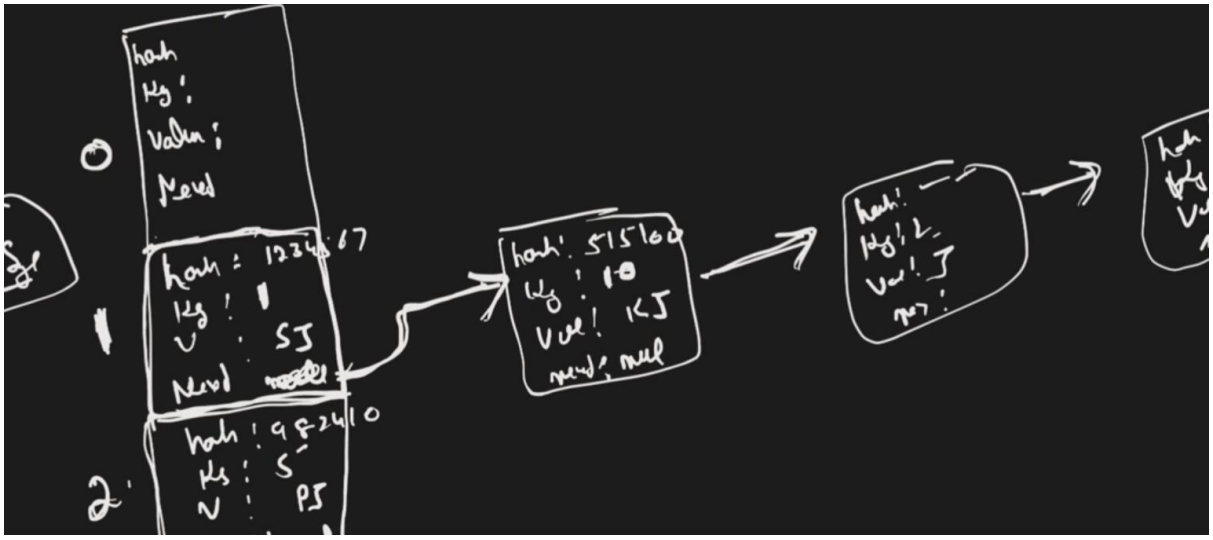
**Worst**

---

**Initally until thereshold in Linked List ==O(n)**

**With Balanced Binary tree ==O(logn) only traverse any one side**

SUBCLASS MEANS EXTENDS.



**LINKEDHASHMAP(Not Thread Safe)**

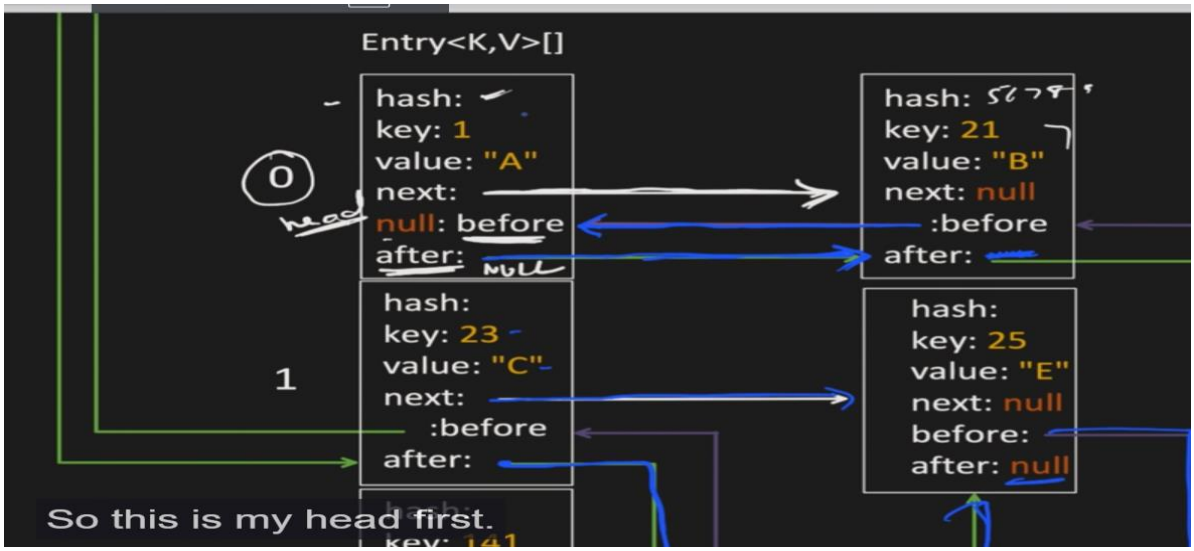
---

**Similar to hash may in terms of Architecture but**

**1)Stores insertion Order (or) Store by low Frq order to high freq order**

**2)Internally Architecture is same as Hashmap but 2 more fields will be added in addition to existing hashcode.. those are after,before to store insertion order (or) Access logic.**

**3)Uses Doubly LINKED LISTl;**




```
Run | Debug
public static void main(String[] args) {
    Map<Integer, String> LHP = new LinkedHashMap<>(initialCapacity:0, loadFactor:0, accessOrder:true); //access Order Freq used ele at end;
    Map<Integer, String> LHP1 = new LinkedHashMap<>(); // Noral Linked HashMap which follow insertion order by DOUBLY LINKED LIST
    LHP.put(key:1, value:"srihas");
    LHP.putIfAbsent(key:2, value:"Srihas12");
    System.out.println(LHP);
    System.out.println(LHP.get(key:1));
    System.out.println(LHP);
}
```

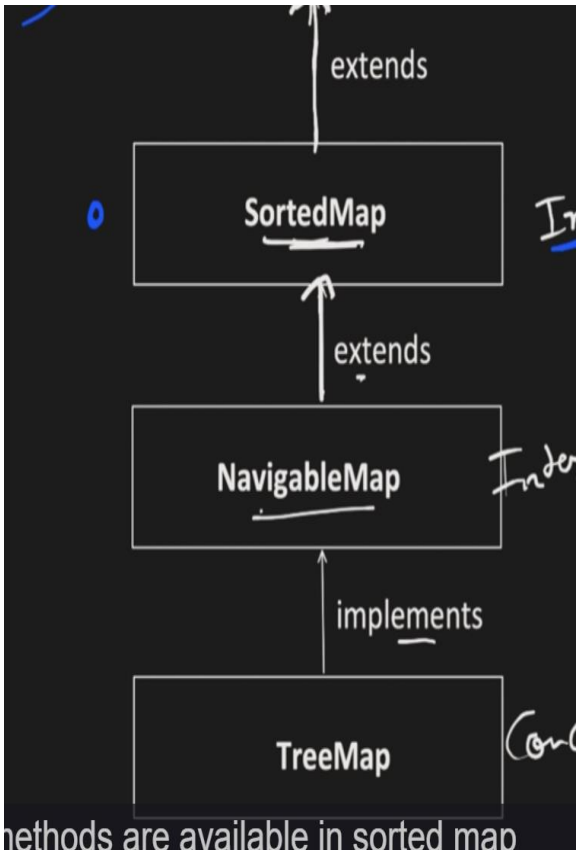
SUBCLASS MEANS EXTENDS.

TreeMap

Simimlar to Map but store in ascending or desending based on comparator it will stor the Key .

Stores in Form of Balanced Binary Tree

Have Useful Methods




```java
public class TreeMap1 {
  Run | Debug
  public static void main(String[] args) {
    NavigableMap<Integer, String> hp = new TreeMap<>();
    hp.put(key:1, value:"srihas");
    hp.put(key:2, value:"srihas");
    hp.put(key:0, value:"srihas");
    System.out.println(hp);
    hp.ceilingEntry(key:2);
    System.out.println(hp.descendingMap());
    NavigableSet<Integer> ns = hp.descendingKeySet();
    for (Integer integer : ns) {
      System.out.println(integer);
    }
  }
}
```

Time Complexity😋 : O(logN)

SUBCLASS MEANS EXTENDS.

SET O(1)

---

SIMilar To HashMap

Store like Entry<val,dummy obj >

LINKEDHAshSET O(1)

---

Similar to LINKED HASHMAP store like DOUBLY LINKED LIST

**But Store only insertion order** not access order.

TREESET(=O(logn)

---

Similar to TREESet

SUBCLASS MEANS EXTENDS.

# MultiThreading

**Process and Thread**

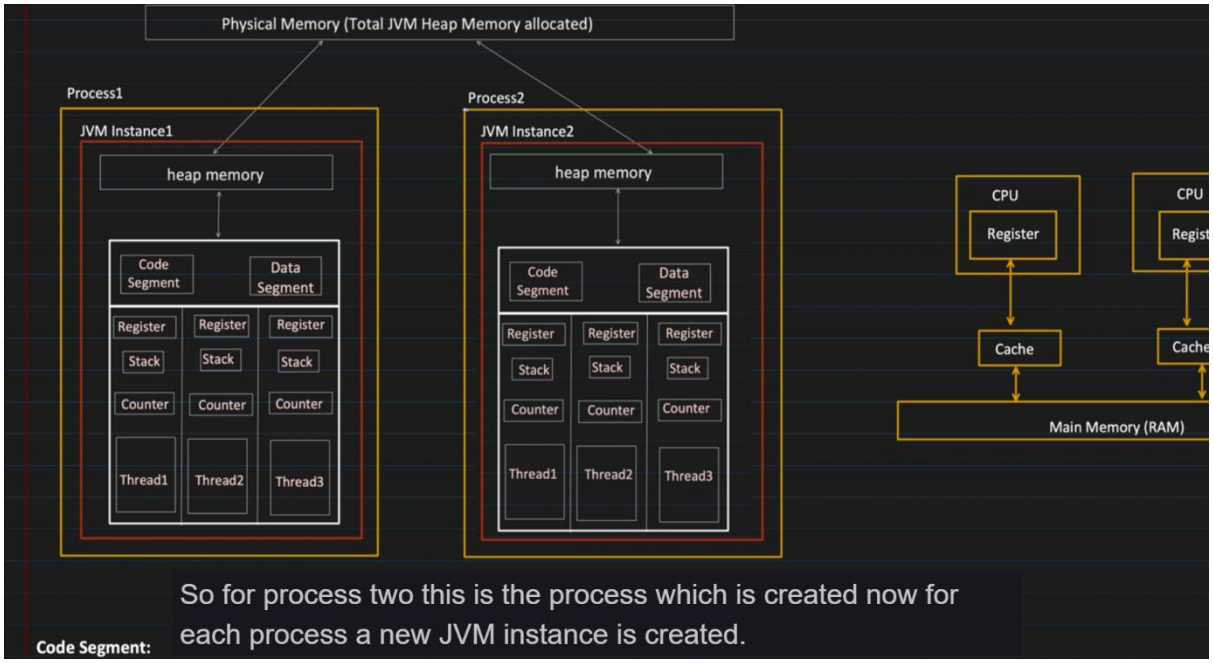
**Process will have multiple Threads**

**Process will be created once we run java class**

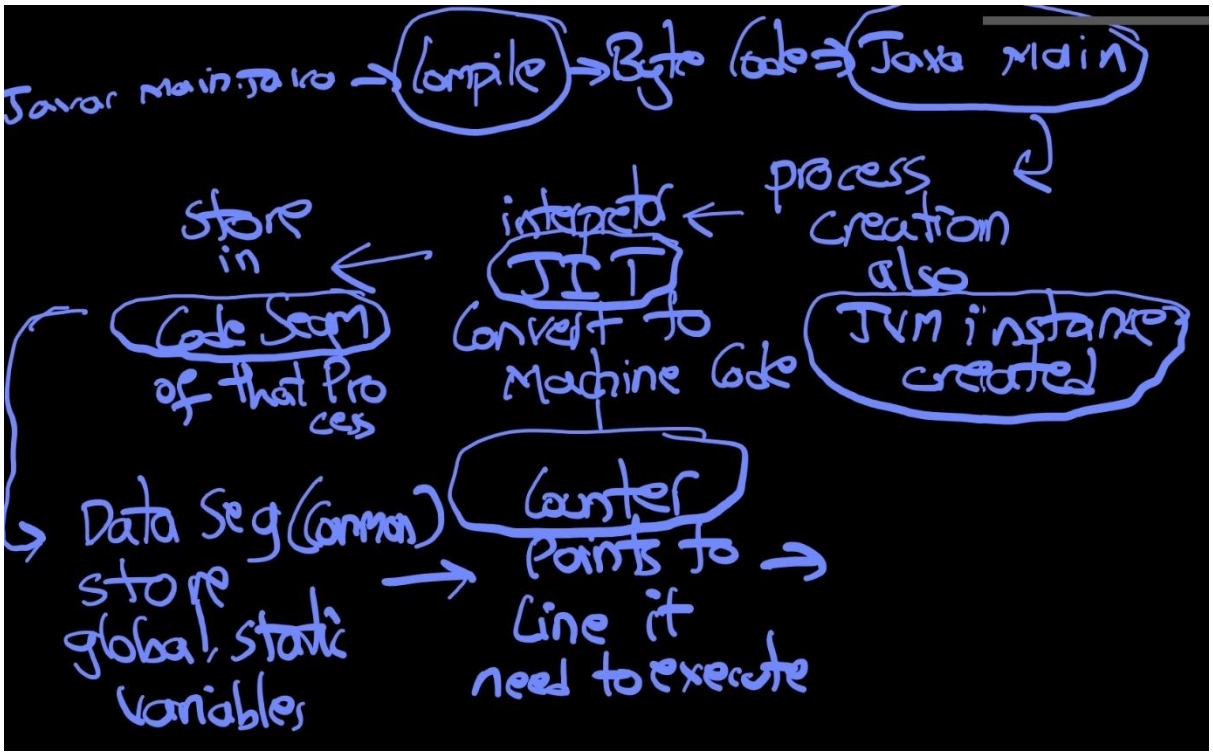
**Javac srihas.java➔Compiled➔Byte Code ➔java srihas➔Process will be created with its own heap memory.**

**Initially in main it will create main thread and accordingly it will create multiple threads**

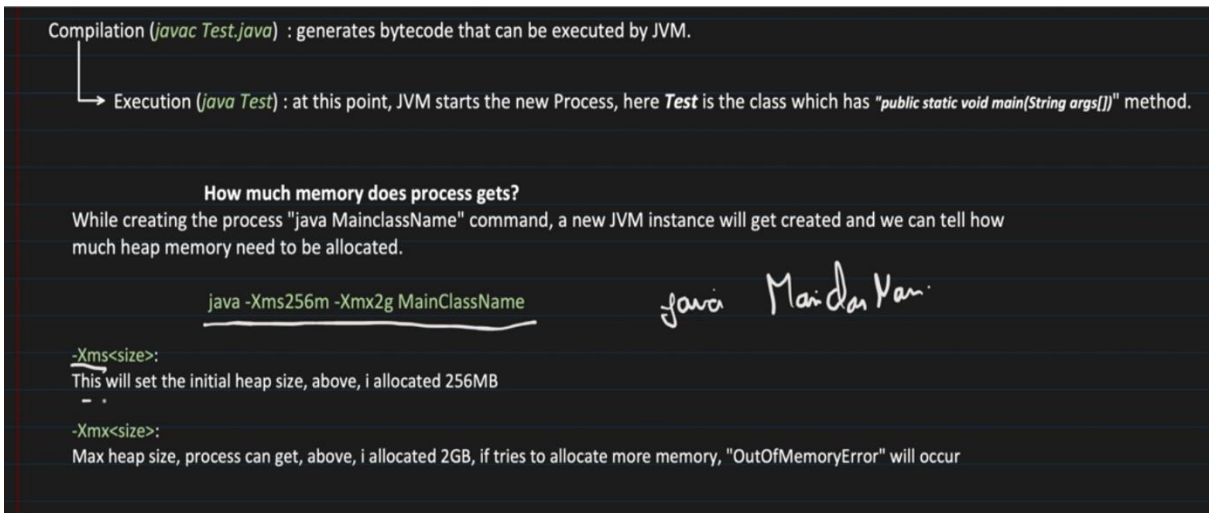
**Thread is a light weight process or specific set of machine understandable code. Executed by CPU**



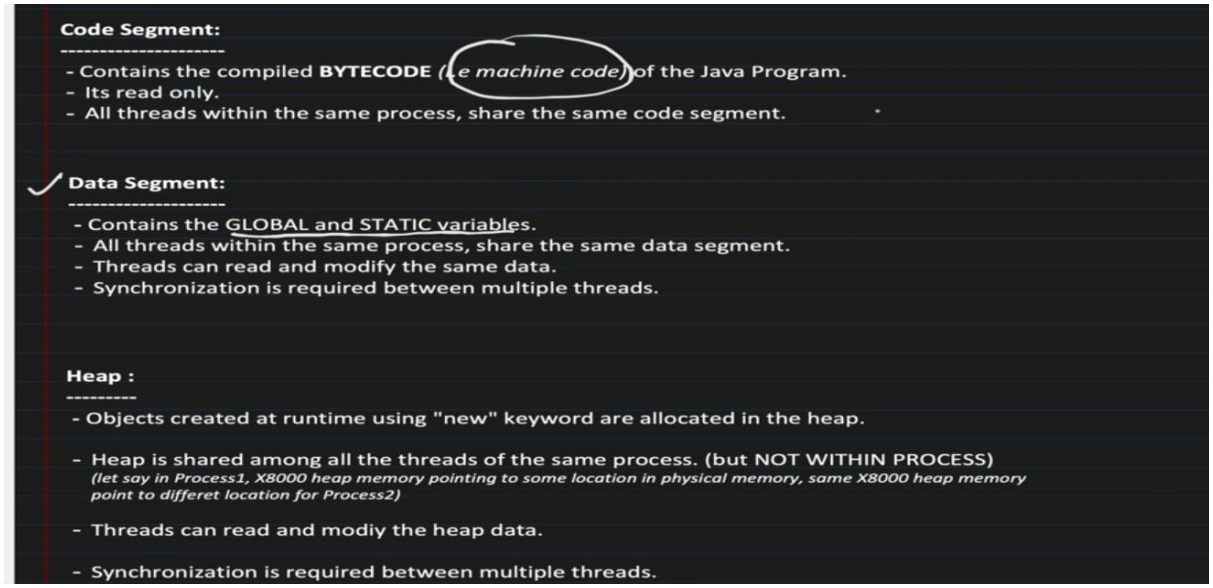

SUBCLASS MEANS EXTENDS.

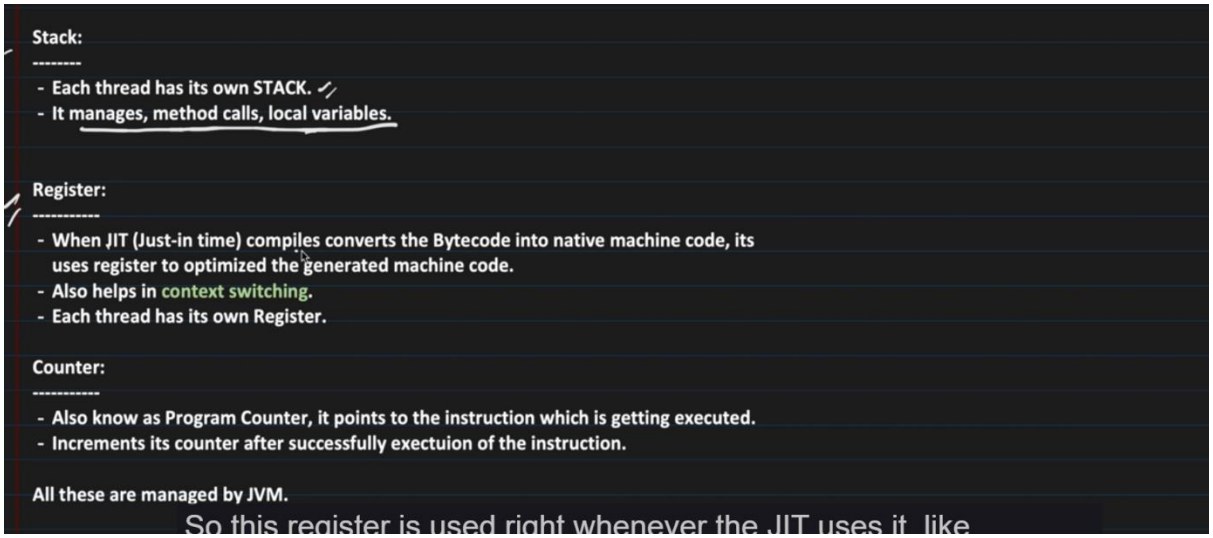



With in the specified jvm space only the memory will be allocated during creation of process itself we can specify size it should consume.




SUBCLASS MEANS EXTENDS.



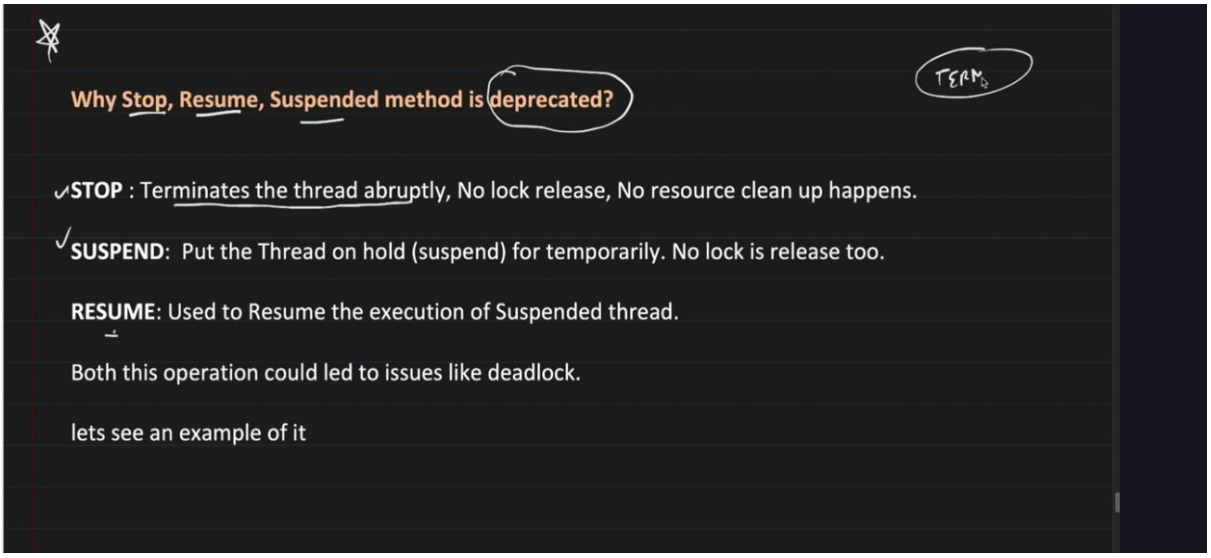

*****

In Cpu the working will be like Thread with specified regid will go to cpu then os will allocate thread to cpu and if any thread time limit reached it will use context switching and store until that part in register and when it again allocated from that point it will be executed.If thread are less and cpu is more then it is multithreading or it is just using context-swiching but looks like multithreading.

Multitasking→creating new process without common memory.

Multithreading→within Process have common memory and independent memory

SUBCLASS MEANS EXTENDS.



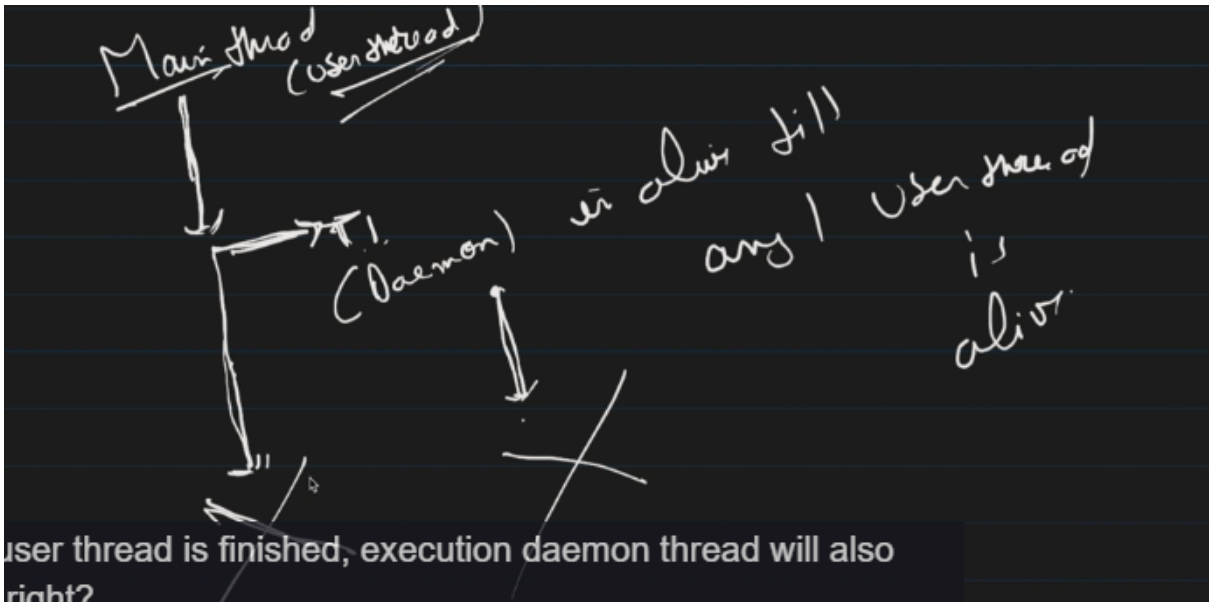

Stop and Suspend wont release locks unlike wait,so deadlock situation wll occur so these methods are depreciated

Resume is used to continue the thread which is suspended hence suspend itself is depreciated resume also is.

## Thread Priority

Never Guarantee that this Thread should execute 1st iit may be work or not so in production we wont use Thread priority.

## Deamon Thread




Normal User Thread


Once all user Thraed are stopped Deamon Thread will be stopped unlike User Thread it wont be running even if the corresponding user thread is finished.

******

Synchronized work with Monitor lock

Obj1 {

Critical section

}

So if both threads have Same obj then Synchronized will work because It will keep Monitor lock on obj .

But What if different Obj Trying to access same critical Section then Monitor lock and Synchronized wont work because there are multiple diff obj trying to access

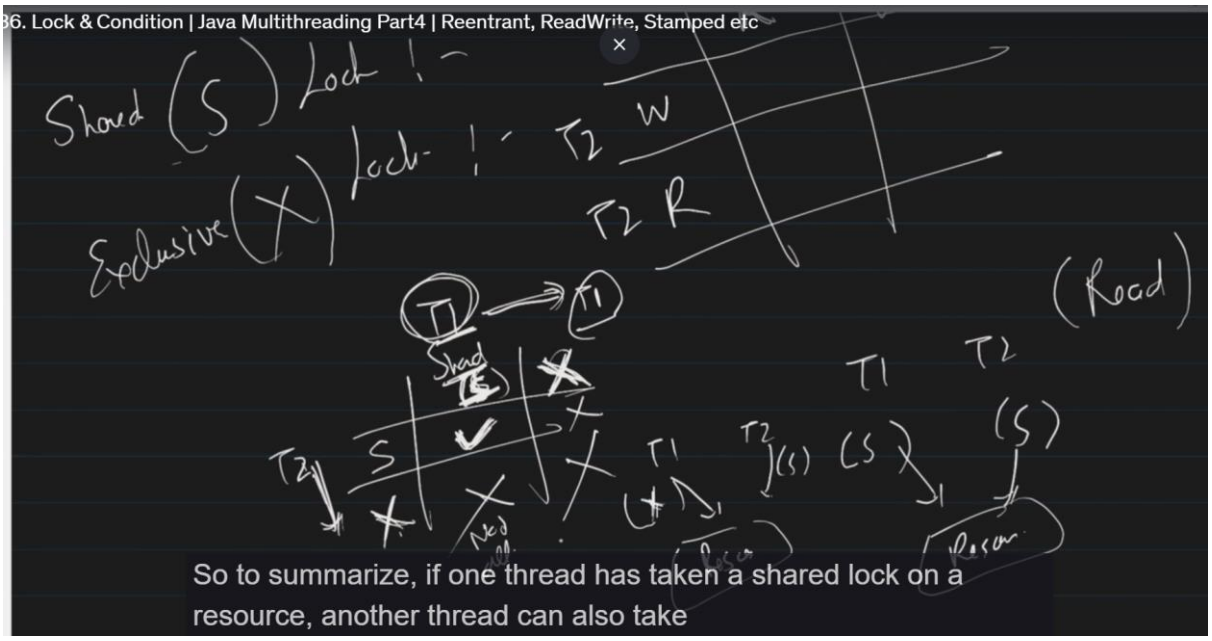
So in these Situation Custom Lock's Comes in

## Types of Custom Locks

1)ReentantLock

Wont have Synchronized but have lock and pass that lock to that function and it will work.

2)ReadWrite Lock



SharedLock ➔ read Exclusive Lock➔ read,write

3)STAMPED LOCK

---

Support read,lick lock and also support **optimistic lock**

**Optimistic Lock➔Don't have lock unlock but have stamp value during locking if that stamp value is changed by other function then it wont update.**

**4)Stemapone Lock**

---

**Similar but can provide how many threads can access critical section**

**Stemaphone sm =new Stemaphone(2);**

**Atomicity**

---

**By CAS(Compare and Swap) operatiom**

**Similar to Optimistic lock where compare the version and if the version is same then only need to update values . Works on the same principle.**

**Don't use lock, efficient but used only in COUNTER++ like get value, compare, update**

**Int counter=1; Atomic, Thread safe**

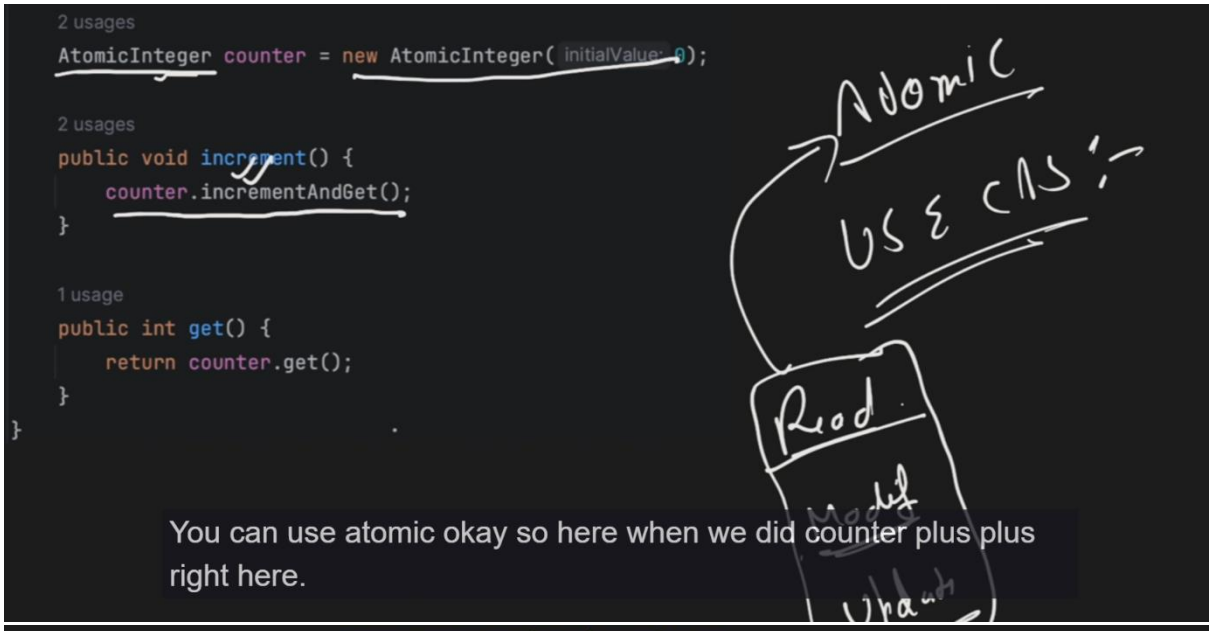
**Class function { void counter(){coun++} Not threadsafe multiple thread can come and update value so chance of data loss**

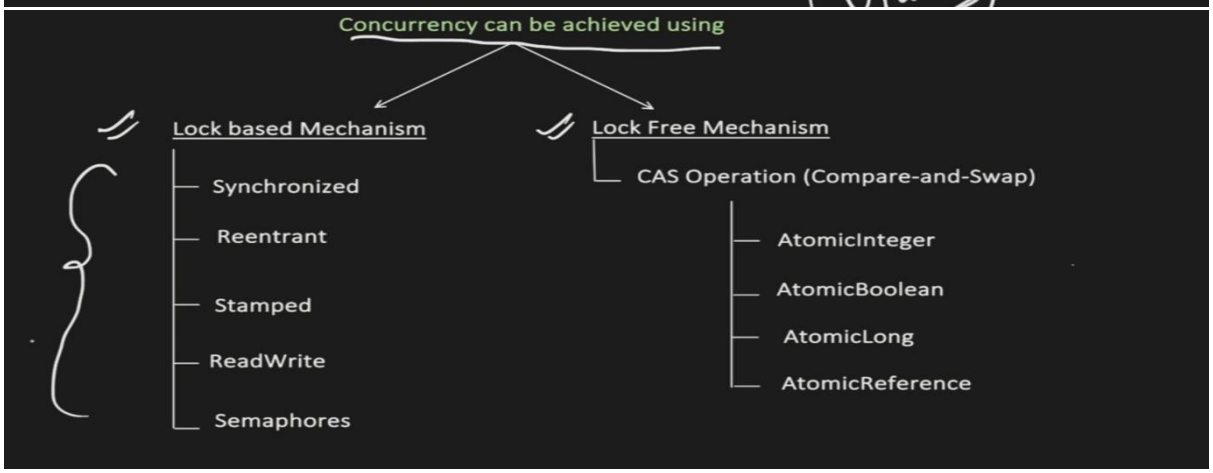
**Can solve by using Synchronized**

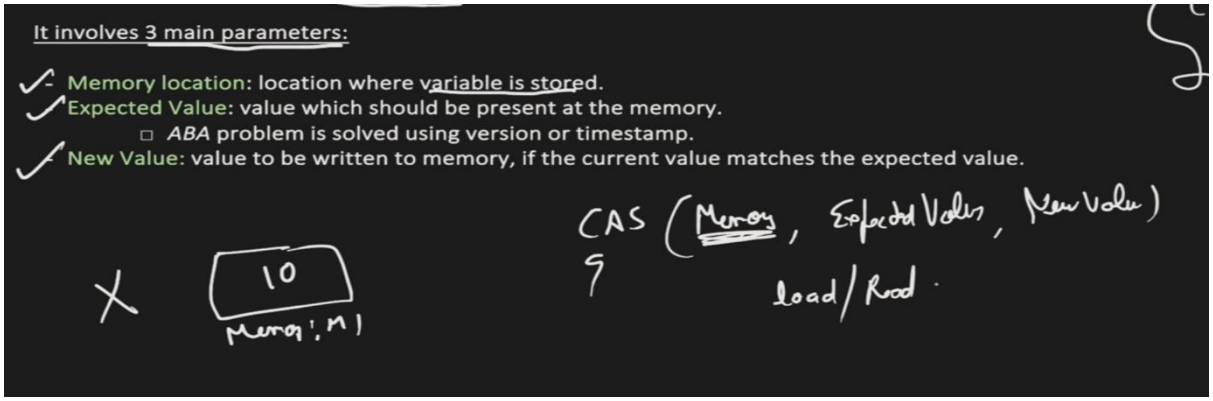
**Also by Atomicity**

## Flow

**Check for value in memory by volatile(Means update value directly to memory not catch for data safety), if value in memory and expected is same update value in memory ,else fail and again initialize memry .**




SUBCLASS MEANS EXTENDS.

## ThreadPool

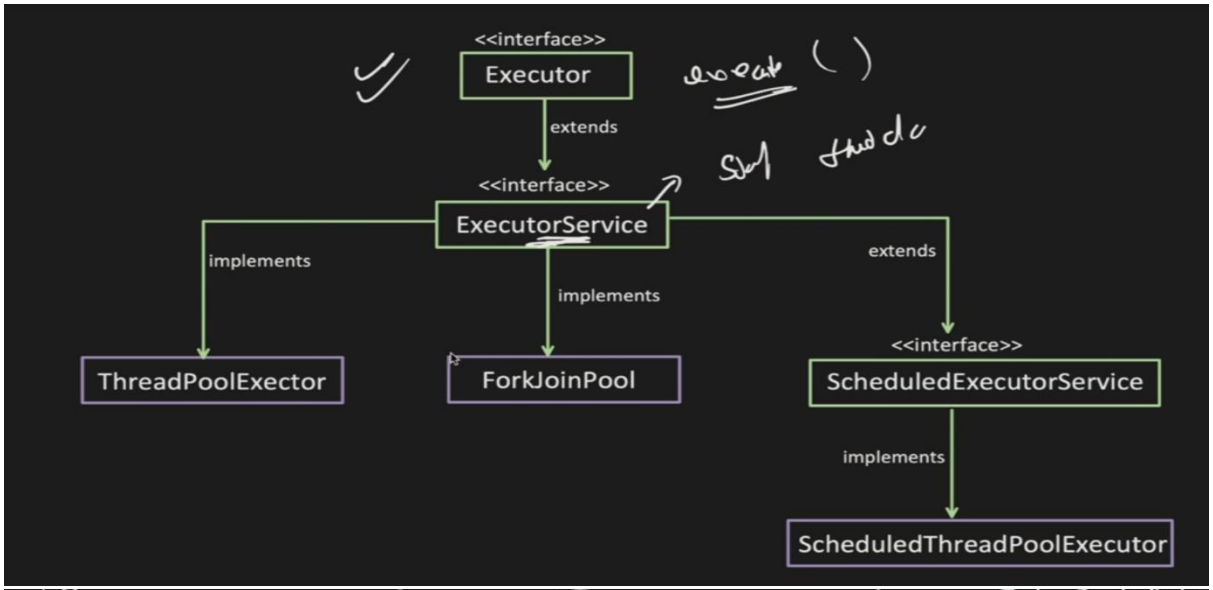
Multiple threads present inside this

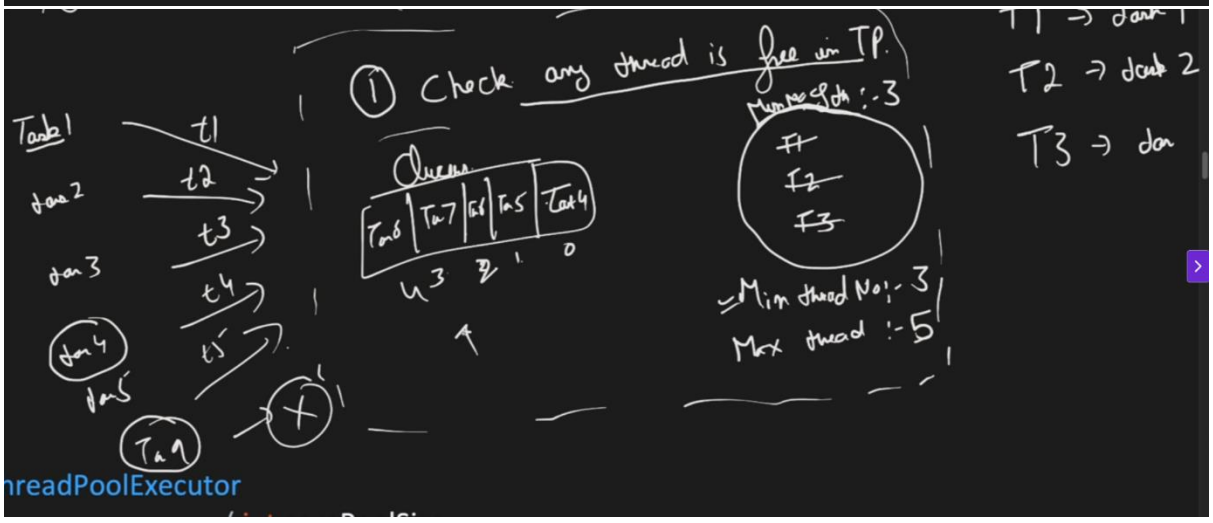
Advan: 1) No need to create Thread

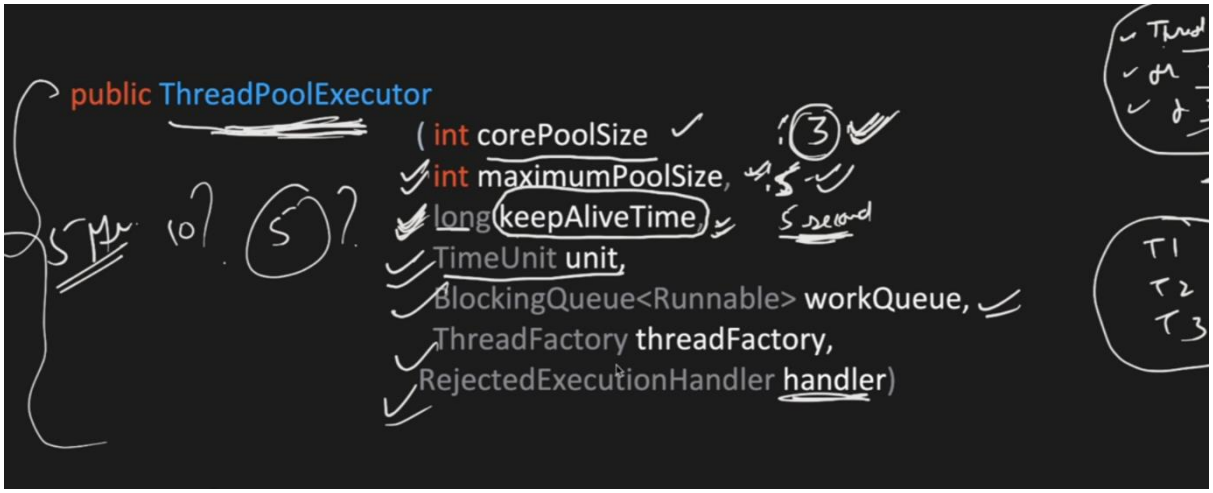
2)No Need to manage thread life cycle like create,wait,etc

3)Not create multiple threads for every task it will resuse the thread so less context switching and cpu wont be idle for longer.

## How to use?






Corepoolsize:min thread,

allowcorethreadTimeout: obj where it will check for keepAliveTime and if it is idle more than that time it will terminate the thread.and time that we mention is defined in units whether it is sec,hr,min

BlockingQueue: Can be fixed Blocking queue means array,and unblocking queue like linked list ,recommemded to use array one.

ThreadFactory: customise the thread whether name,priority and define it as demon thread etc.

RejectedExecutionHandler: To handle the rejected task if queue is full and threads are occupied they are types of it .

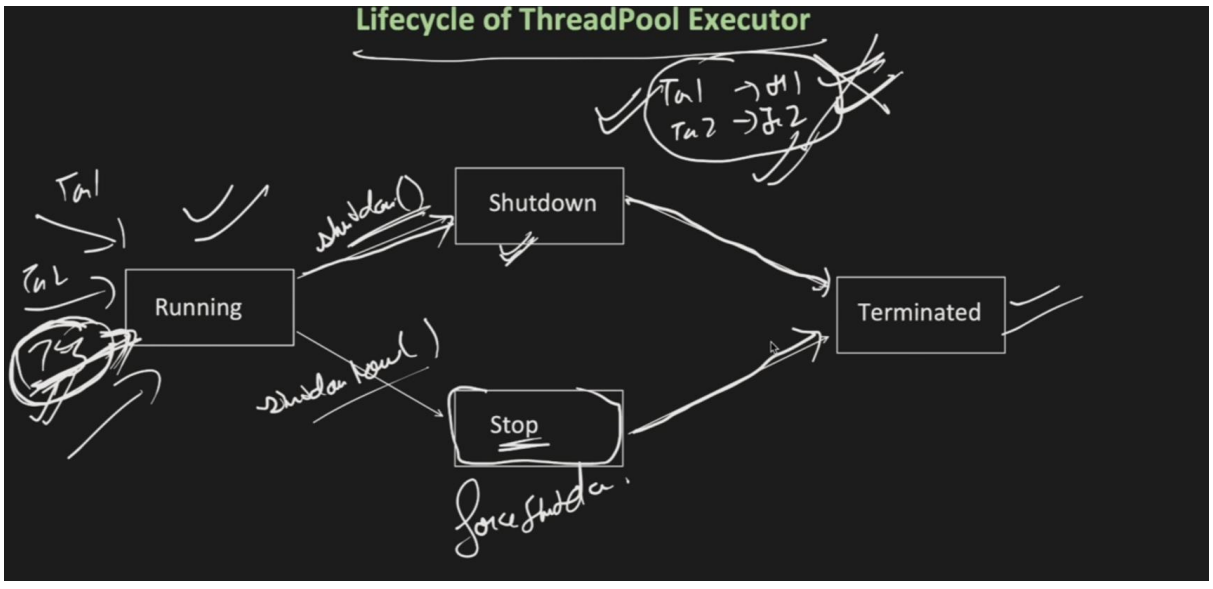
SUBCLASS MEANS EXTENDS.


Abort policy:just throw exception

CallerRunThread: send the task to the caller like main thread to execute

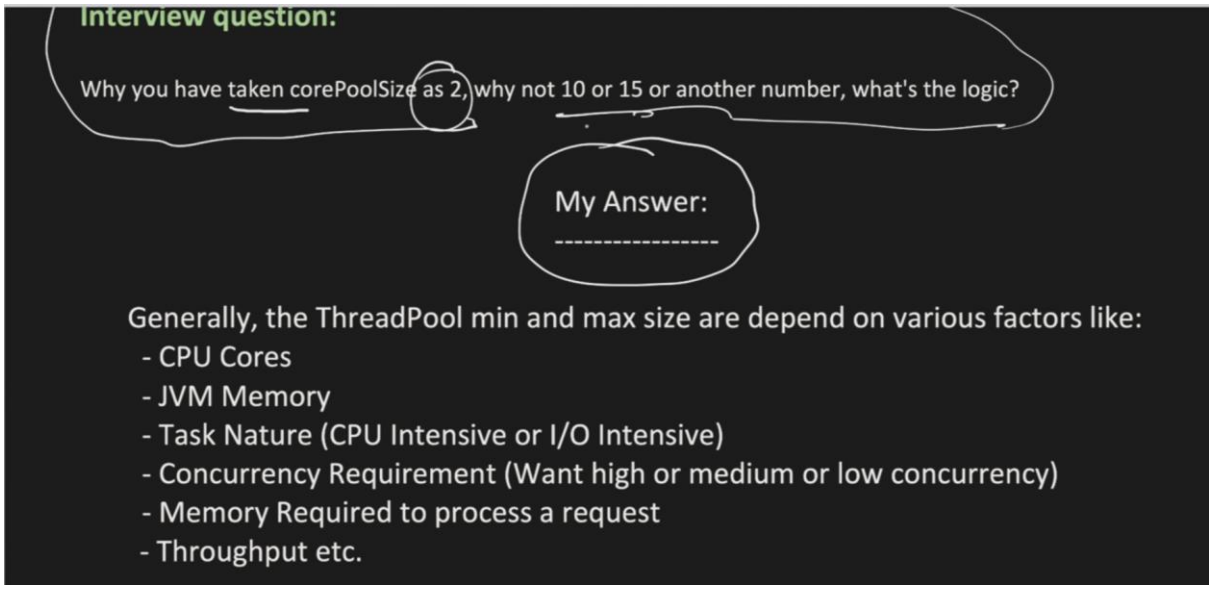
DiscardPolicy:silently reject it without exception.

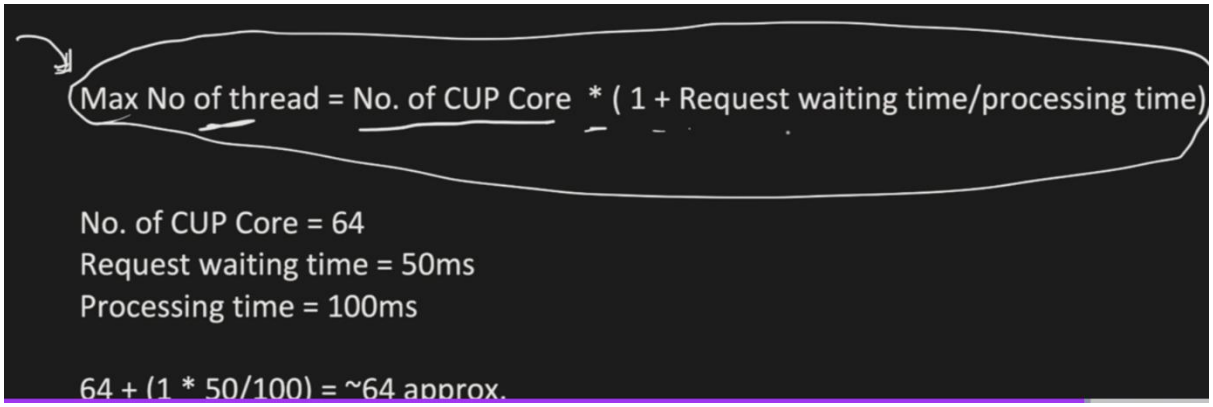
DiscardOldestPolicy:Remove older task in queue and replace with recent rejected task.

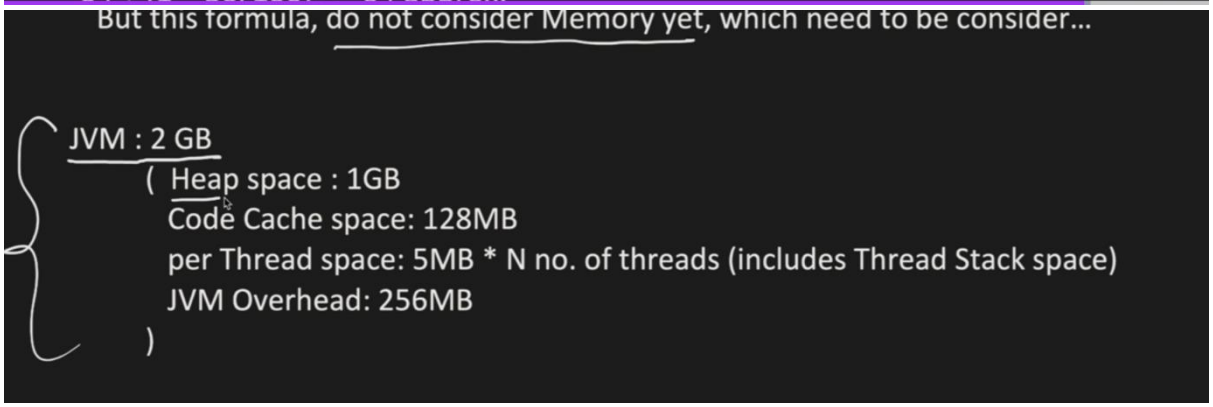



Running means accepts tasks,shutdown means wont accept new tasks and finish exesting ones,stop even running threads.

SUBCLASS MEANS EXTENDS.




Need to consider all factors if more tasks then context switching will be more so check cpu

Then space utilization for each thread some number of space should be allocated if only 500 mb is there and each thread take 10mb then only 50 threads be created also look for heap space.

SUBCLASS MEANS EXTENDS.

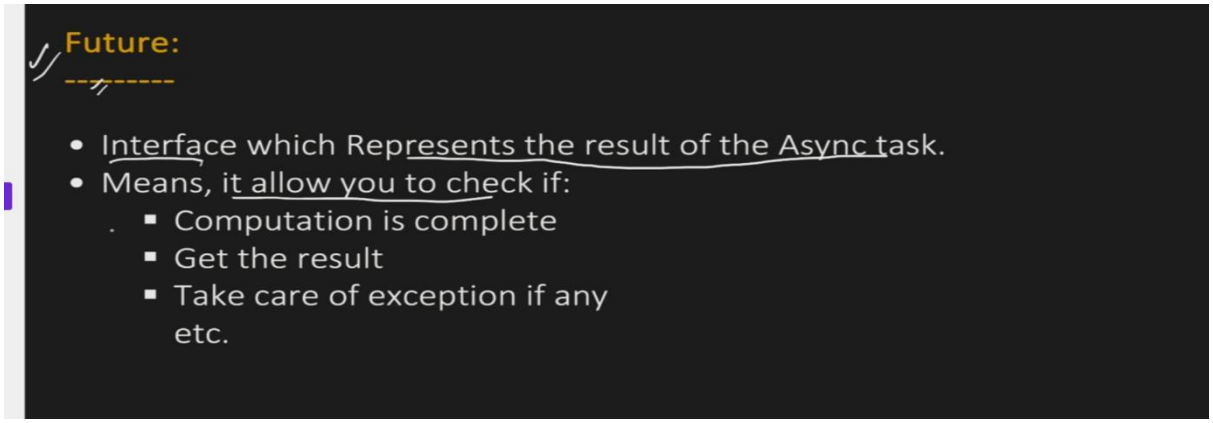
## Future,Collable,CompletableFuture

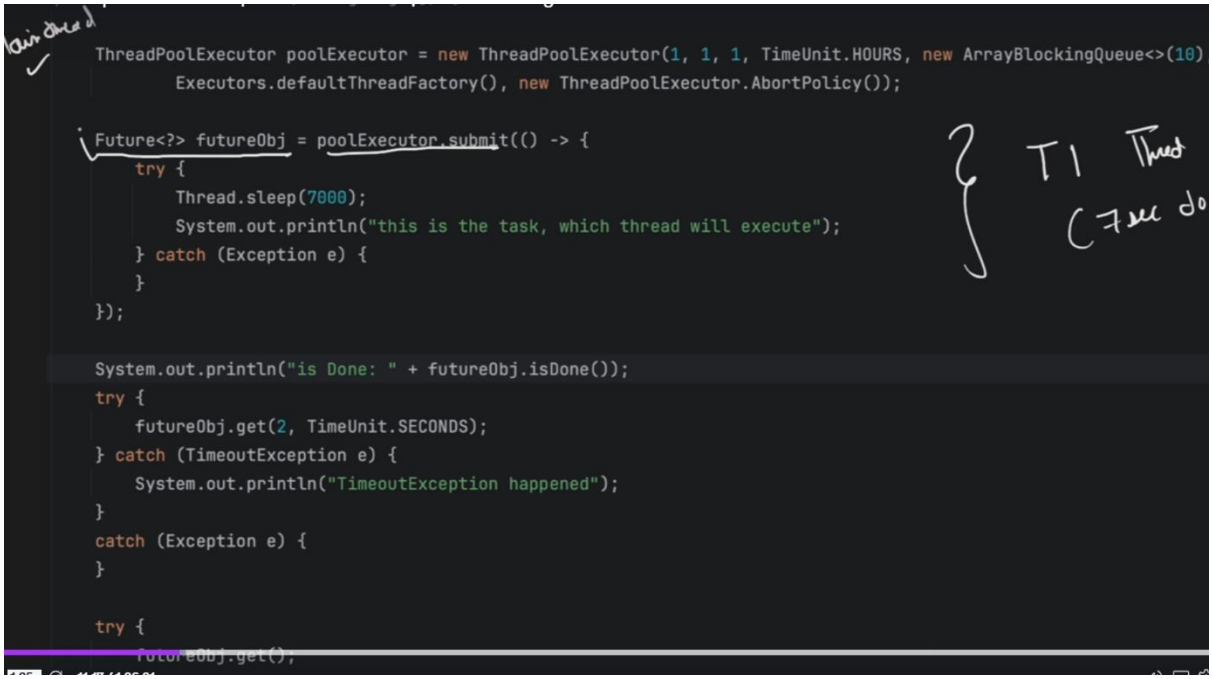
Future➔To know state of Thread like is it done,cancelled,wait for completion

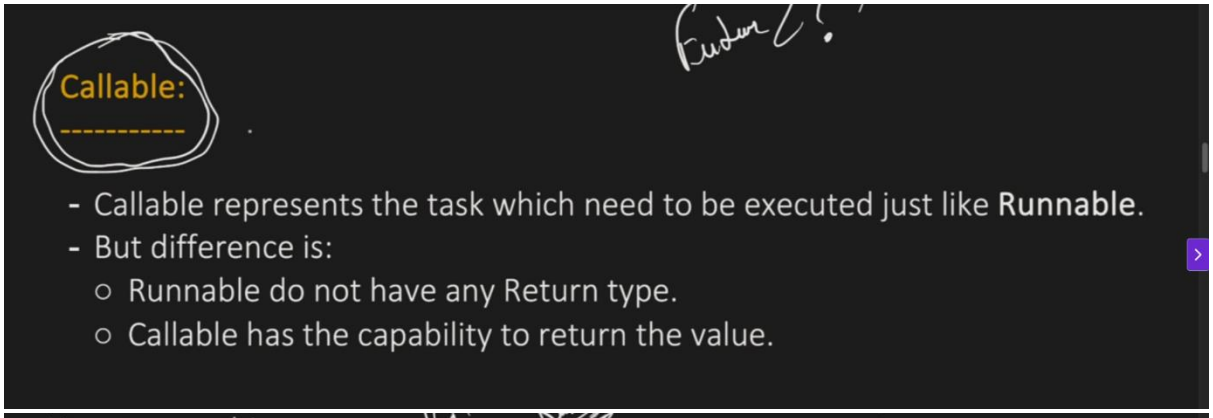
Collable➔Similar but have return type unlike Future

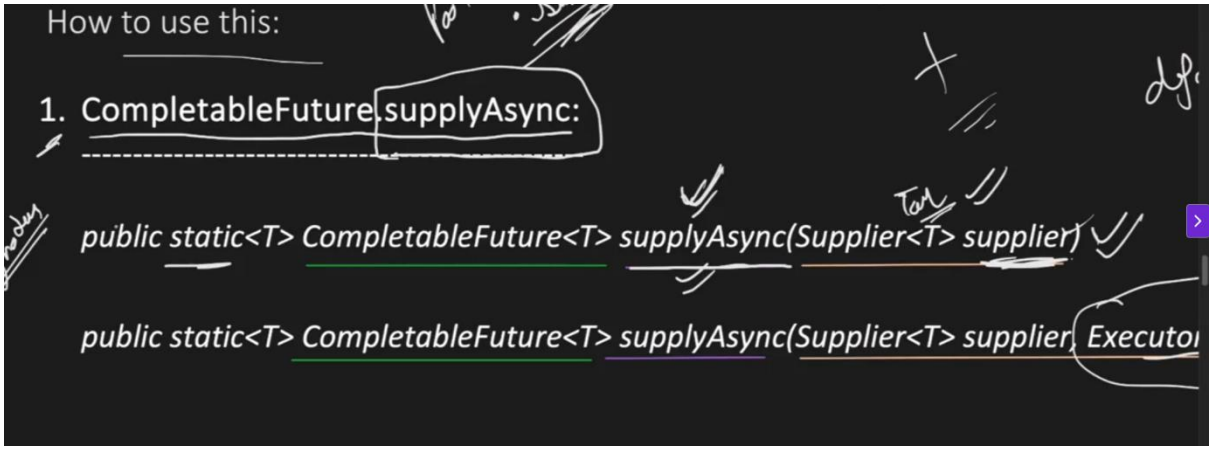
CompletableFuture➔Implements Future have all methods and also used for chaining

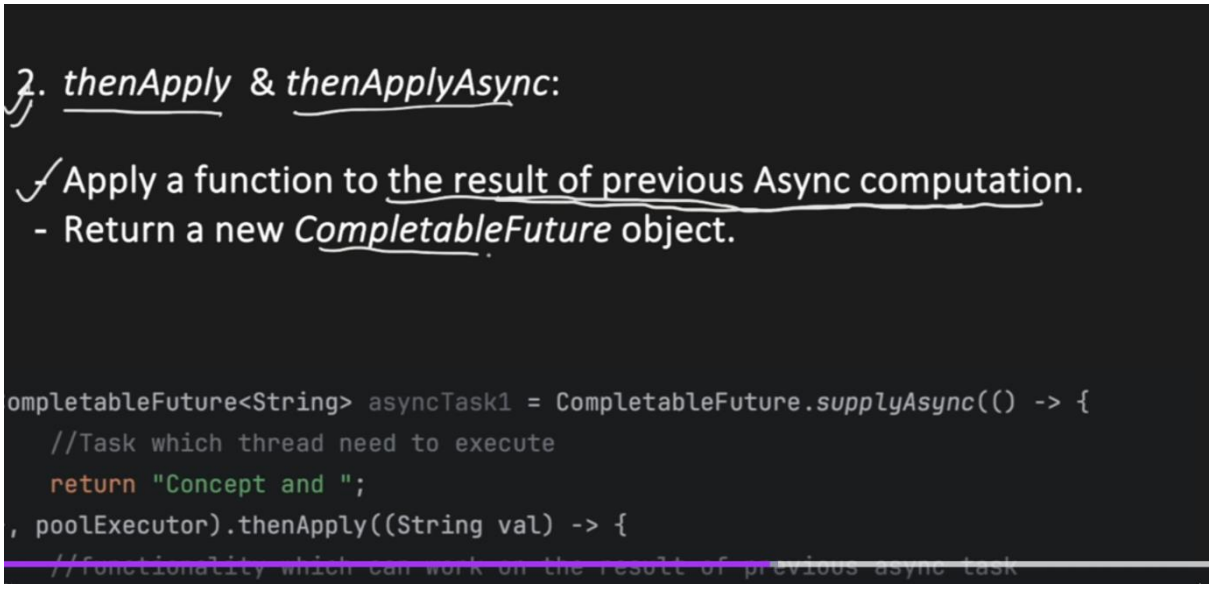
By methods like thissync,thenAccept,thencompose,



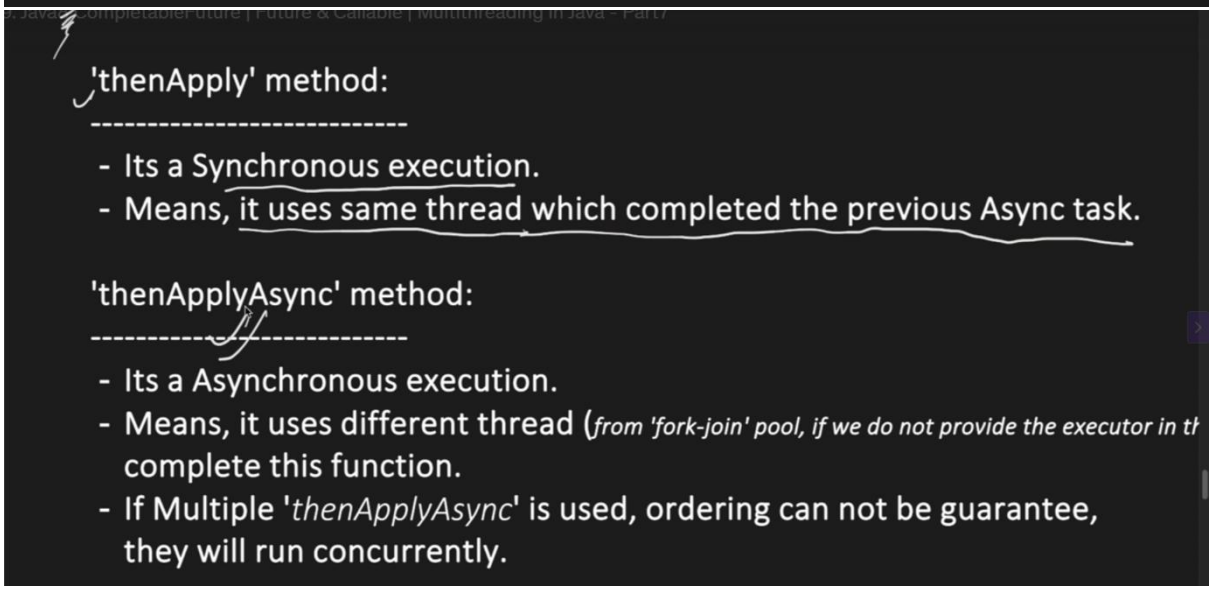



SUBCLASS MEANS EXTENDS.

```java
CompletableFuture<String> asyncTask1 = CompletableFuture.supplyAsync(() -> {
    //Task which thread need to execute
    return "Concept and ";
}, poolExecutor).thenApply((String val) -> {
    //functionality which can work on the result of previous async task
    return val + "Coding";
});
```




SUBCLASS MEANS EXTENDS.

3. thenCompose and thenComposeAsync:

- Chain together dependent Aysnc operations.
- Means when next Async operation depends on the result of the previou
  We can tied them together.
- For aysnc tasks, we can bring some Ordering using this.

```
ompletableFuture<String> asyncTask1 = CompletableFuture
        .supplyAsync(() -> {
            System.out.println("Thread Name which runs 'supplyAsync': " + Thread.currentThread
            return "Concept and ";
        }, poolExecutor)
```

```
CompletableFuture<String> asyncTask1 = CompletableFuture
        .supplyAsync(() -> {
            System.out.println("Thread Name which runs 'supplyAsync': " + Thread.currentThread
            return "Concept and ";
        }, poolExecutor)
        .thenCompose((String val) -> {
            return CompletableFuture.supplyAsync(() -> {
                System.out.println("Thread Name which runs 'thenCompose': " + Thread.currentTh
                return val + "Coding";
            });
        });
```

4)thenAccept final stage don’t have return type

5)thenCombine combines two CompletableFutures.

# Types of Executors or Custom defined Thread Pools

**'newFixedThreadPool' method creates a thread pool executor with a fixed no. of threads.**

| | |
|---|---|
| Min and Max Pool | Same |
| Queue Size | Unbounded Queue |
| Thread Alive when idle | Yes |
| When to use | Exact Info, how many Async task is needed |
| Disadvantage | Not good when workload is heavy, as it will lead to limited concurrency |

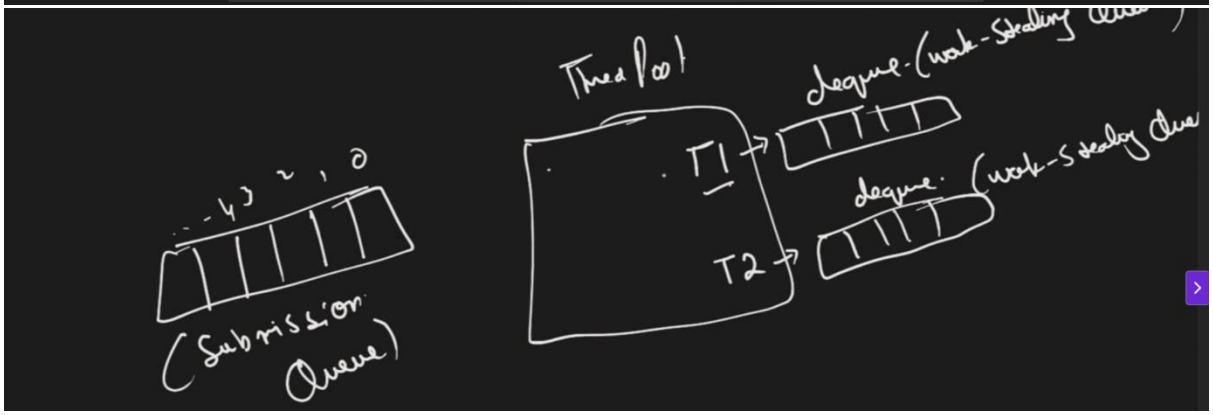
```
//fixed thread pool executor
ExecutorService poolExecutor1 = Executors.newFixedThreadPool( nThreads: 5);
poolExecutor1.submit(() -> "this is the async task");
```

| | |
|---|---|
| Min and Max Pool | ✓ Min : 0<br>Max: Integer.MAX_VALUE |
| Queue Size | Blocking Queue with Size 0 |
| Thread Alive when idle | 60 SECONDS |
| When to use | Good for handling burst of short lived tasks. |
| Disadvantage | Many long lived tasks and submitted rapidly, ThreadPool can create so many threads which might lead to increase memory usage. |

SUBCLASS MEANS EXTENDS.

**'newSingleThreadExecutor' creates Executor with just single Worker thread.**

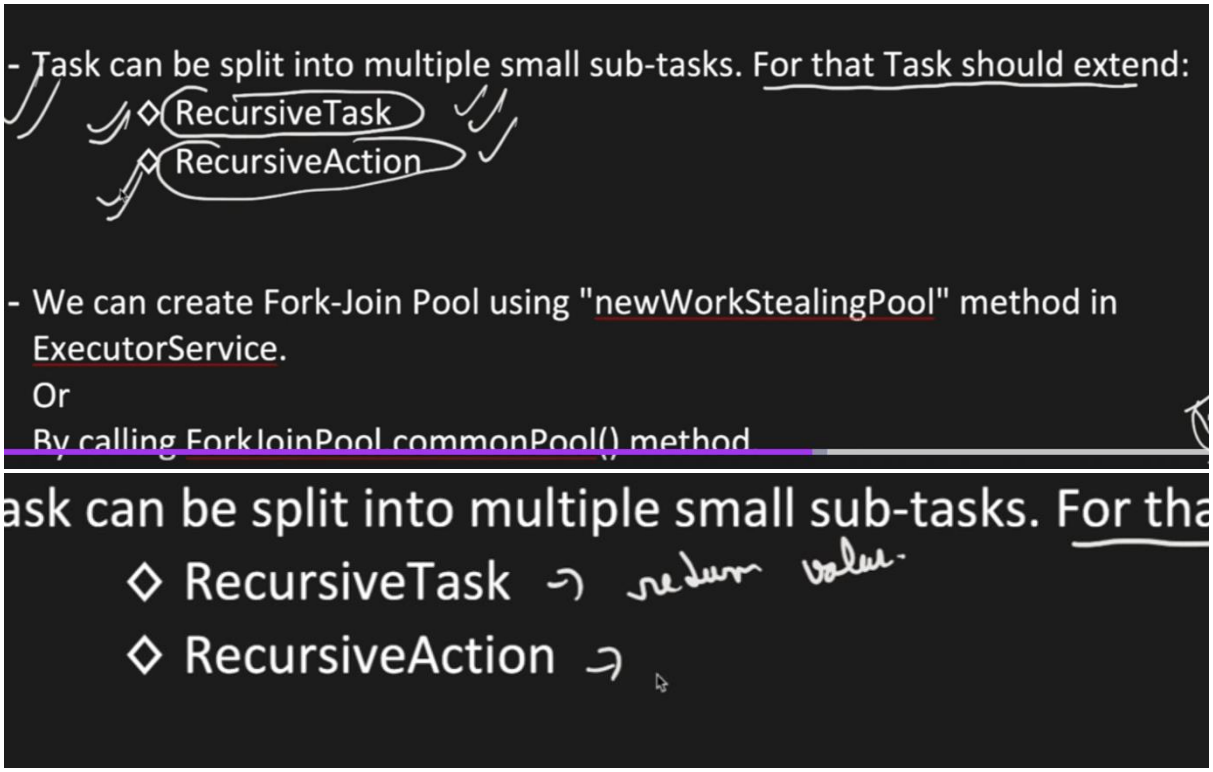
| | |
|---|---|
| Min and Max Pool | Min : 1<br>Max: 1 |
| Queue Size | Unblocking Queue |
| Thread Alive when idle | Yes |
| When to use | When need to process tasks sequentially |
| Disadvantage | No Concurrency at all |



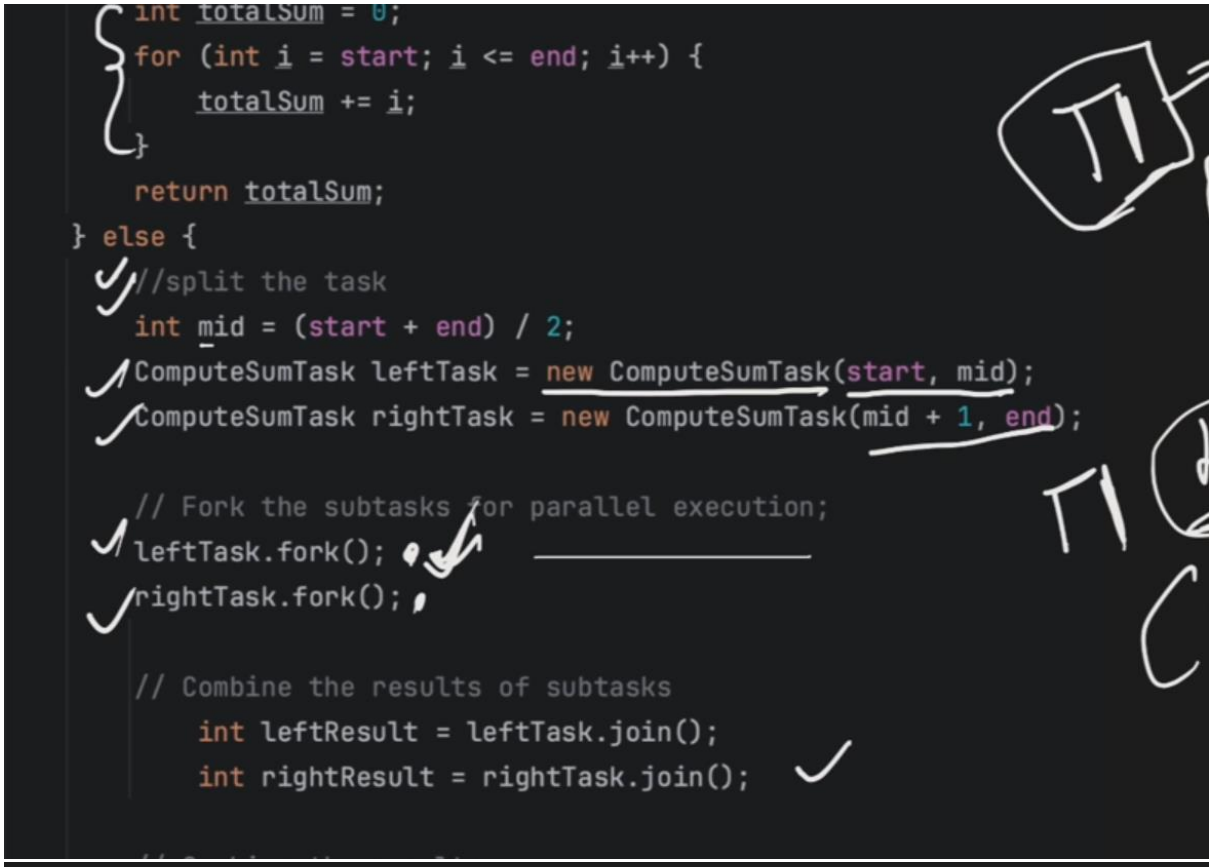

Steps:

- If all threads are busy, task would be placed in "Submission Queue". (or whenever we call submit() method, tasks goes into submission queue only)
- Lets say task1 picked by ThreadA. And if 2 subtasks created using fork() method. Subtask1 will be executed by ThreadA only and Subtask2 is put into the ThreadA work-stealing queue.
- If any other thread becomes free, and there is no task in Submission queue, it can "STEAL" the task from the other thread work-stealing queue.

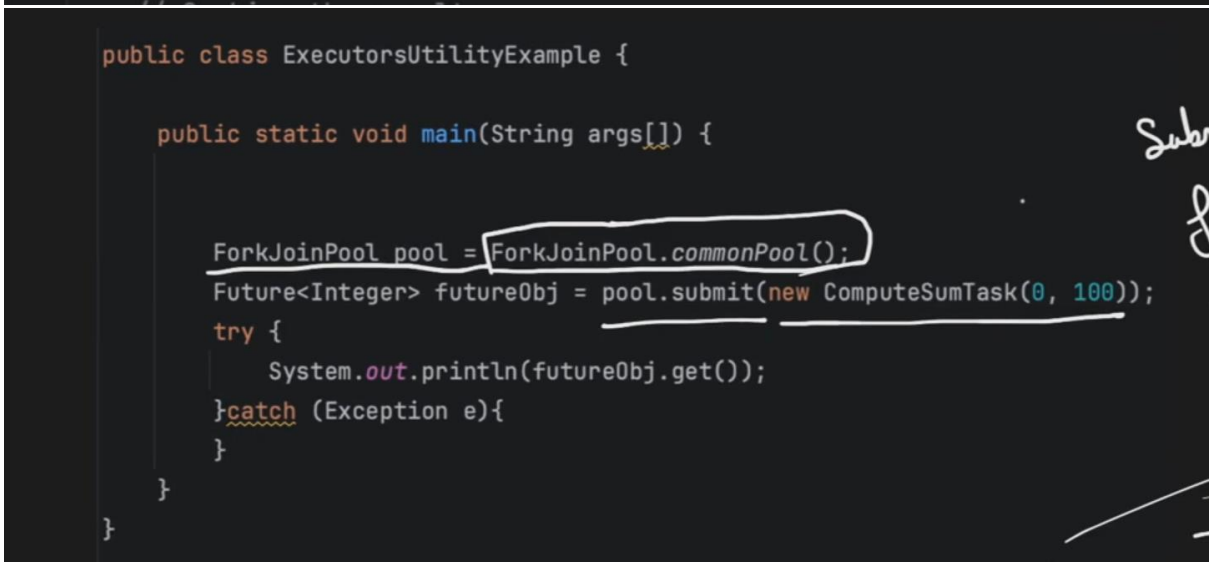
SUBCLASS MEANS EXTENDS.




SUBCLASS MEANS EXTENDS.



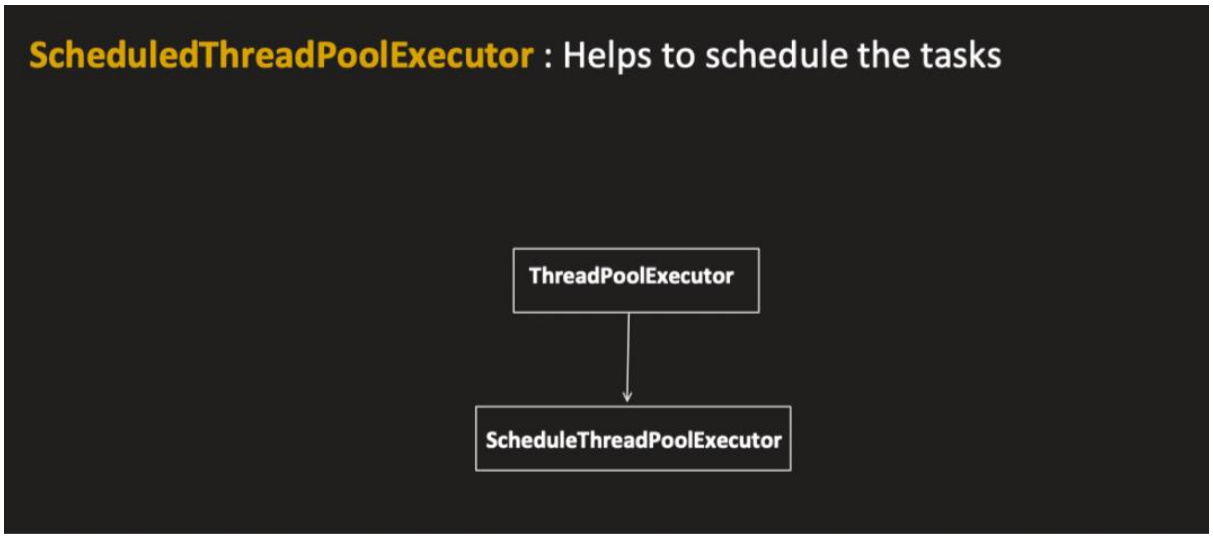
```
        int totalSum = 0;
        for (int i = start; i <= end; i++) {
            totalSum += i;
        }
        return totalSum;
    } else {
        //split the task
        int mid = (start + end) / 2;
        ComputeSumTask leftTask = new ComputeSumTask(start, mid);
        ComputeSumTask rightTask = new ComputeSumTask(mid + 1, end);

        // Fork the subtasks for parallel execution;
        leftTask.fork();
        rightTask.fork();

        // Combine the results of subtasks
            int leftResult = leftTask.join();
            int rightResult = rightTask.join();
```



```
public class ExecutorsUtilityExample {

    public static void main(String args[]) {


        ForkJoinPool pool = ForkJoinPool.commonPool();
        Future<Integer> futureObj = pool.submit(new ComputeSumTask(0, 100));
        try {
            System.out.println(futureObj.get());
        }catch (Exception e){
        }
    }
}
```

SUBCLASS MEANS EXTENDS.

# ScheduledPoolExecutor



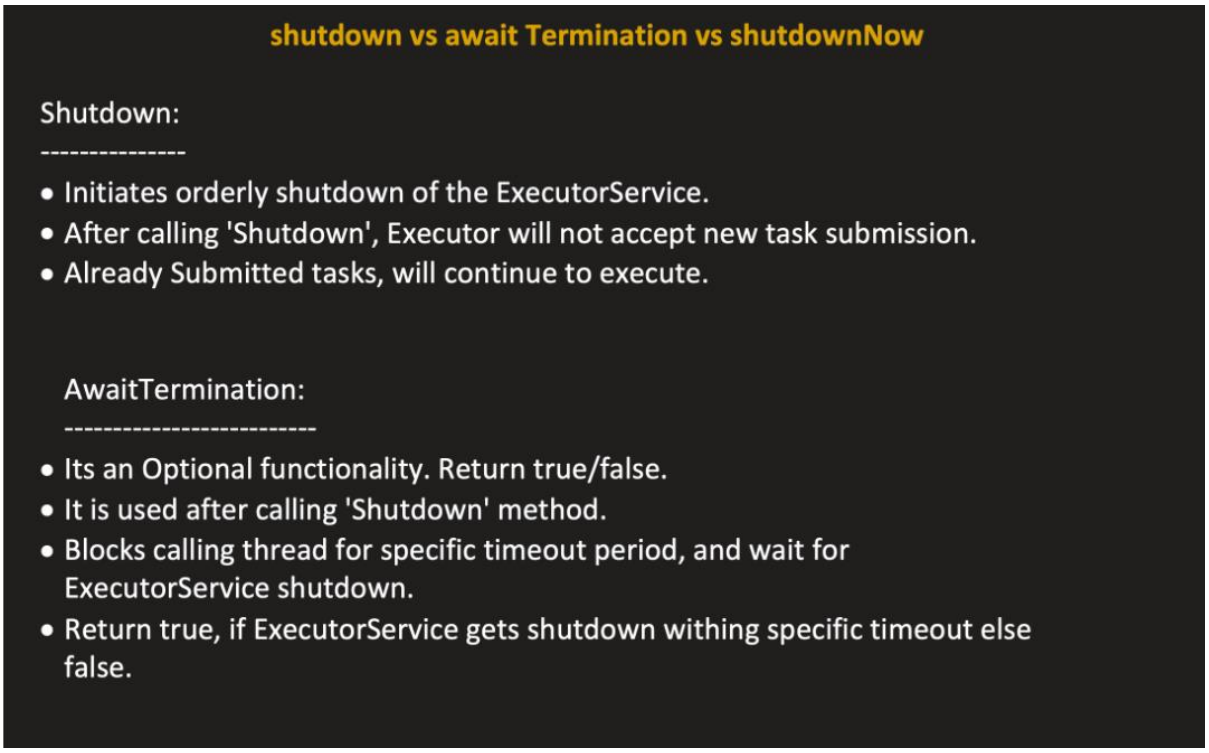

| S.No. | Method Name | Description |
|---|---|---|
| 1. | schedule(Runnable command, long delay, TimeUnit unit) | Schedules a Runnable task after specific delay.<br><br>Only one time task runs. |
| 2. | schedule(Callable<V> callable, long delay, TimeUnit unit) | Schedules a Callable task after specific delay.<br><br>Only one time task runs. |
| 3. | scheduleAtFixedRate(Runnable command, long initialDelay, long period, TimeUnit unit)<br><br>`public static void main(String args[]) {`<br>`ScheduledExecutorService scheduledExecutorServiceObj = new ScheduledThreadPoolExecutor( corePoolSize: 15);`<br>`Future<?> scheduledObj = scheduledExecutorServiceObj.scheduleAtFixedRate(() -> {`<br>`System.out.println("task going to start by : " + Thread.currentThread().getName());`<br>`try{`<br>`Thread.sleep( millis: 10000);`<br>`}catch (Exception e){`<br>`}`<br>`System.out.println("New task");`<br>`}, initialDelay: 5, period: 5, TimeUnit.SECONDS);` | Schedules a Runnable task for repeated execution with fixed rate.<br><br>We can use cancel method to stop this repeated task.<br><br>Also lets say, if thread1 is taking too much time to complete the task and next task is ready to run, till previous task will not get |
| 4. | scheduleWithFixedDelay(Runnable command, long initialDelay, long delay, TimeUnit unit) | Schedules a Runnable task for repeated execution with a fixed delay<br>(Means next task delay counter start only after previous one task completed) |

Schedule:will schedule after that delay time is over

ScheduleAt FixedRate:Schedules initially after delay after that based on the period it will schedule continuously.

SUBCLASS MEANS EXTENDS.

ScheduleAtFixeddelay:Only after compleion of previos thread only delay will be done after that new schedule will happen




Shutdown : will not accept new tasks and finishes already running tasks.

ShutdownNow: Not finishes running task also

Await termination :check if threadpool is shutdown.

## **Virtual Thread vs Normal Thread**

Normal thread or Platform Thread are the threads that we are using until now where  creation on thread takes time and internally that thread is created in os and jvm wraps around it and gives access.

Disadvantages of platform or normal threa:

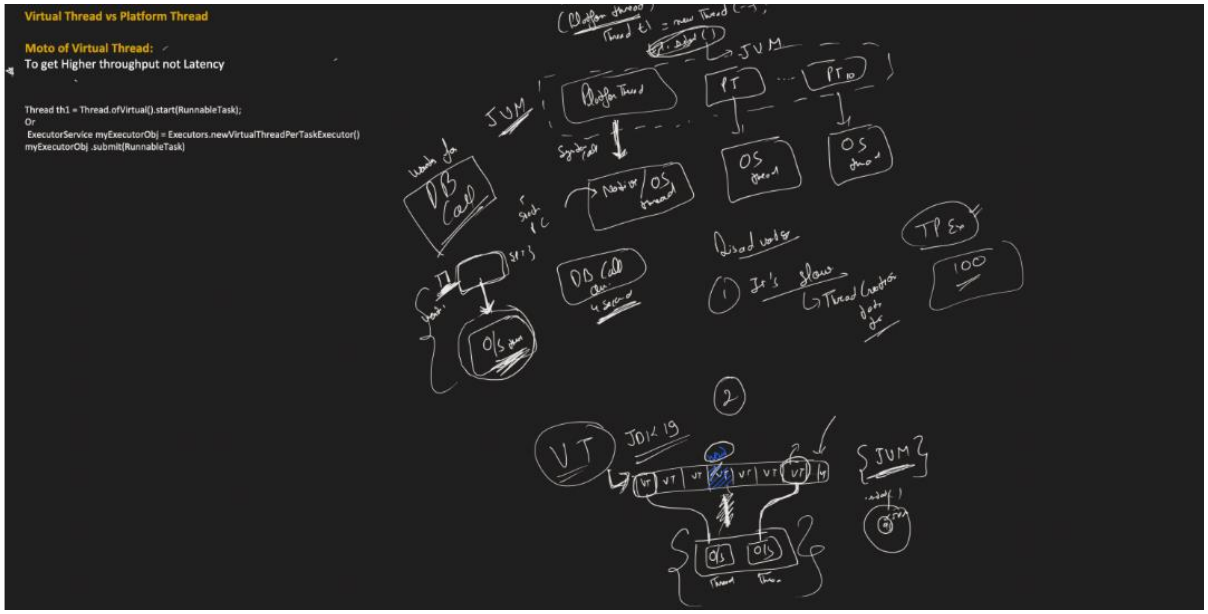
- Takes times to create Thread so we use threadPool but still inside it we need to create threads
- Also other one is once task linked to the os then if it has some db operations and goes to wait() then os will be unused .

## Virtual Threads

Focus on throuput means in 1sec how many process got executed

So here we can create any number of virtual threads and those were handled by JVM

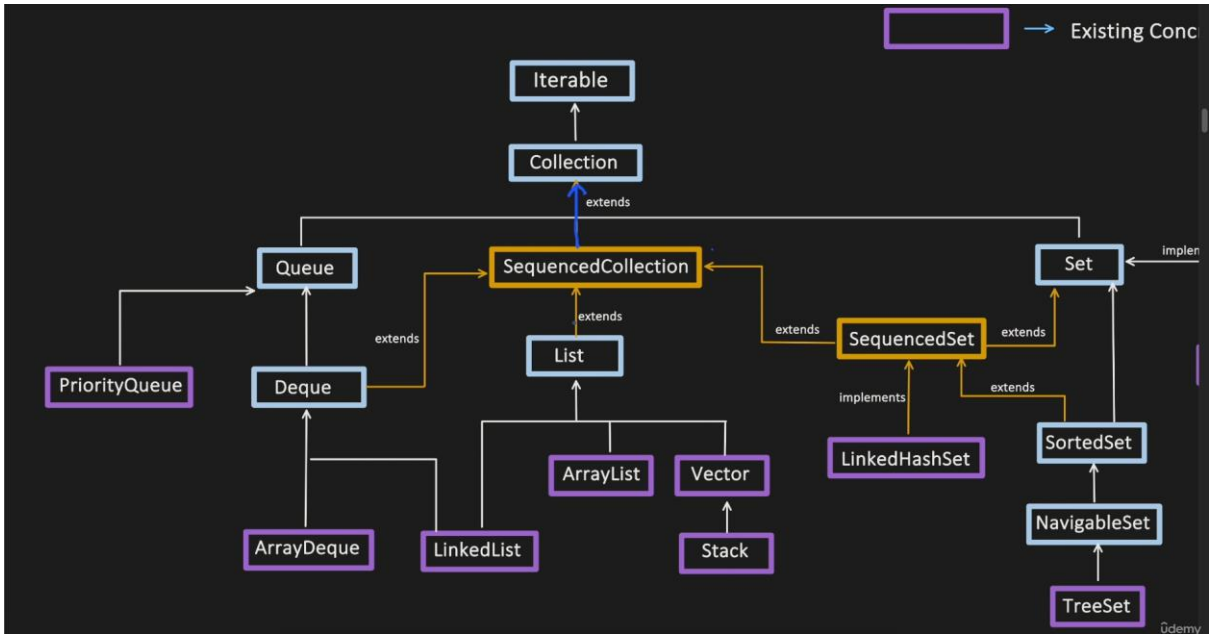
If at any point that thread goes to waiting it will unlink from os and other thread uses the OS.

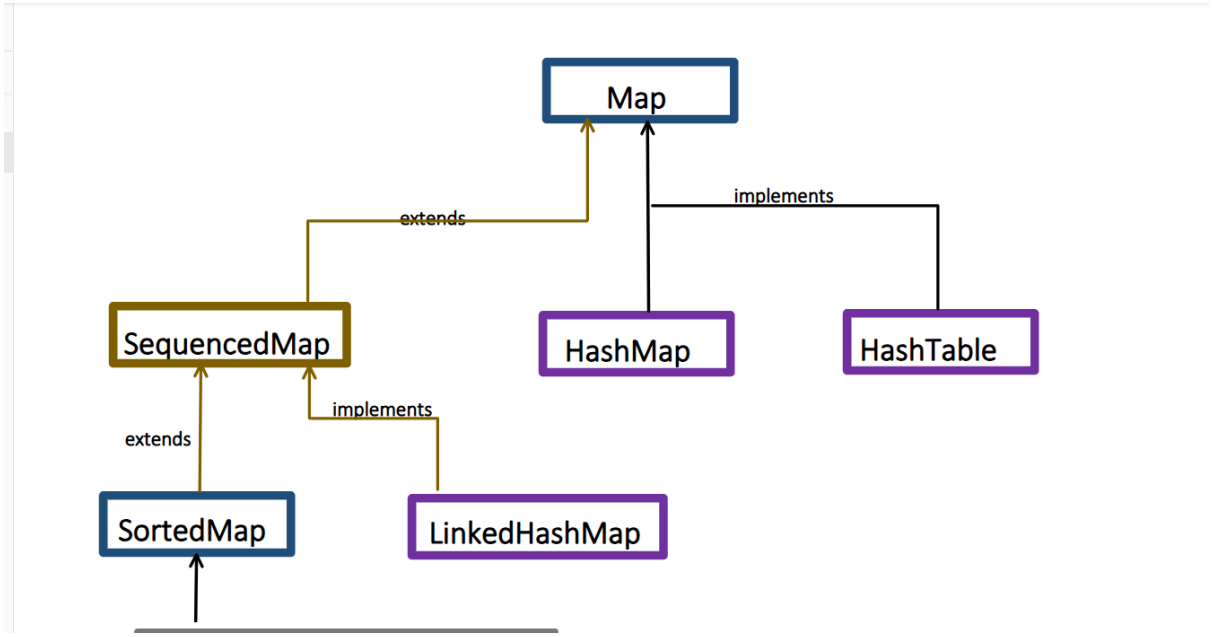



# ThreadLocal

Every thread have this function Thread local we can set this and can see


SUBCLASS MEANS EXTENDS.




SUBCLASS MEANS EXTENDS.

Any Collection which follows below conditions, can be termed as Sequenced

- Collection should follow Predictable Iteration :
  Means elements are returned in consistent and well defined order every time we iterate over the collection.
  So if a collection maintains elements in:
    - Insertion order, or
    - Sorted order (e.g., ascending or descending)

  Then we can say, it follows Predictable iteration.

- Collection provide support for Access or Manipulate the First and Last element
- Collection supports Reversible View

Like List Follows so it can be extended by sequenceCollection but priority queue does not follow sseq so not extended .similarly in queue can add aend so not extended.

Similalrly for Map and set also.

Why using :Using Common methods for all the sequenced collection.

SUBCLASS MEANS EXTENDS.