**Java Core**

**Day-1:25/7/24**

**Java:**

**C++**

**C**

Adv

MS Office (VC++)

IRCTC(JAVA)

Languages

Apks

Application S/w

System s/w

**Software**

**Applications and Languages**

**Java**

**Web**

**Internet**

`

Low level

Middle level

High level (Java, c, c++, vc++, python, dotNet)

**Why java is More Popular**

* **Platform Independent**

Windows

jdk

Source code

(.java)

.class file

Byte code

Linux

Mac

Java code JVM

* **Oops Concept**
* **Exception Handling**
* **Multi-Threading**
* **Support Standalone, web apks**

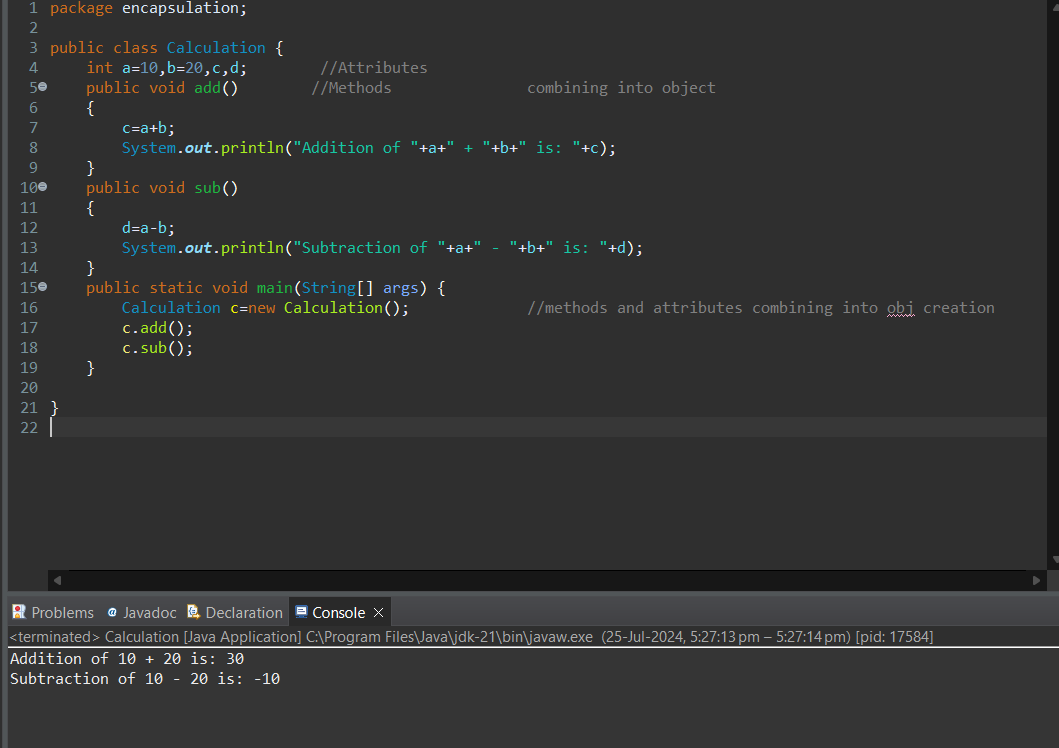
**OOPS Concept:**

* **Encapsulation**
* **Inheritance**
* **Polymorphism**
* **Abstraction**

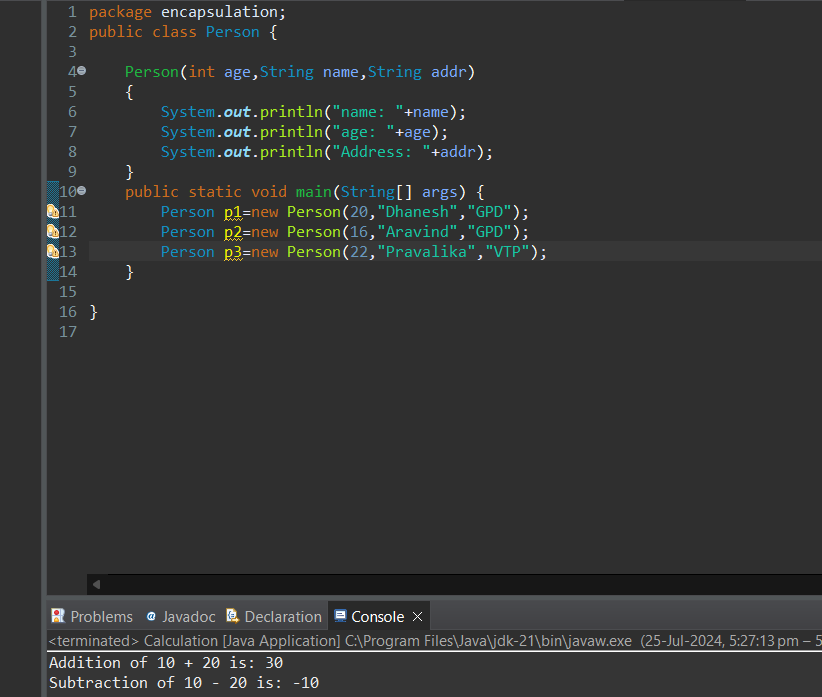
**Encapsulation:**

Encapsulation is a mechanism binding Attributes and methods together in side a class and object creation.

* Class is a protector

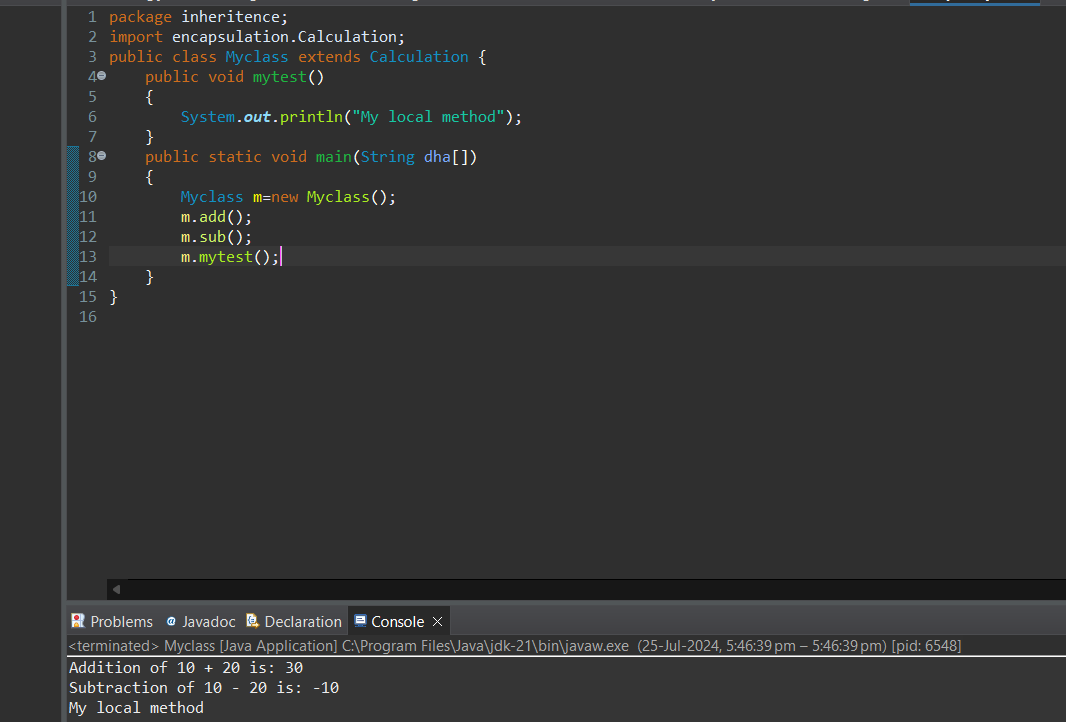


Person Program example for Encapsulation



**Inheritance:**

* Reusability of existing functionality from super class to sub class.
* Support Multi level Inheritance
* Doesn’t support Multiple Inheritance at class level due to ambiguity.
* Support multiple inheritance through interface.
* We can access Super class to sub class through “extends” key word.

****

**Polymorphism:**

**Compile Time Run Time**

Method Overloading Method Overriding

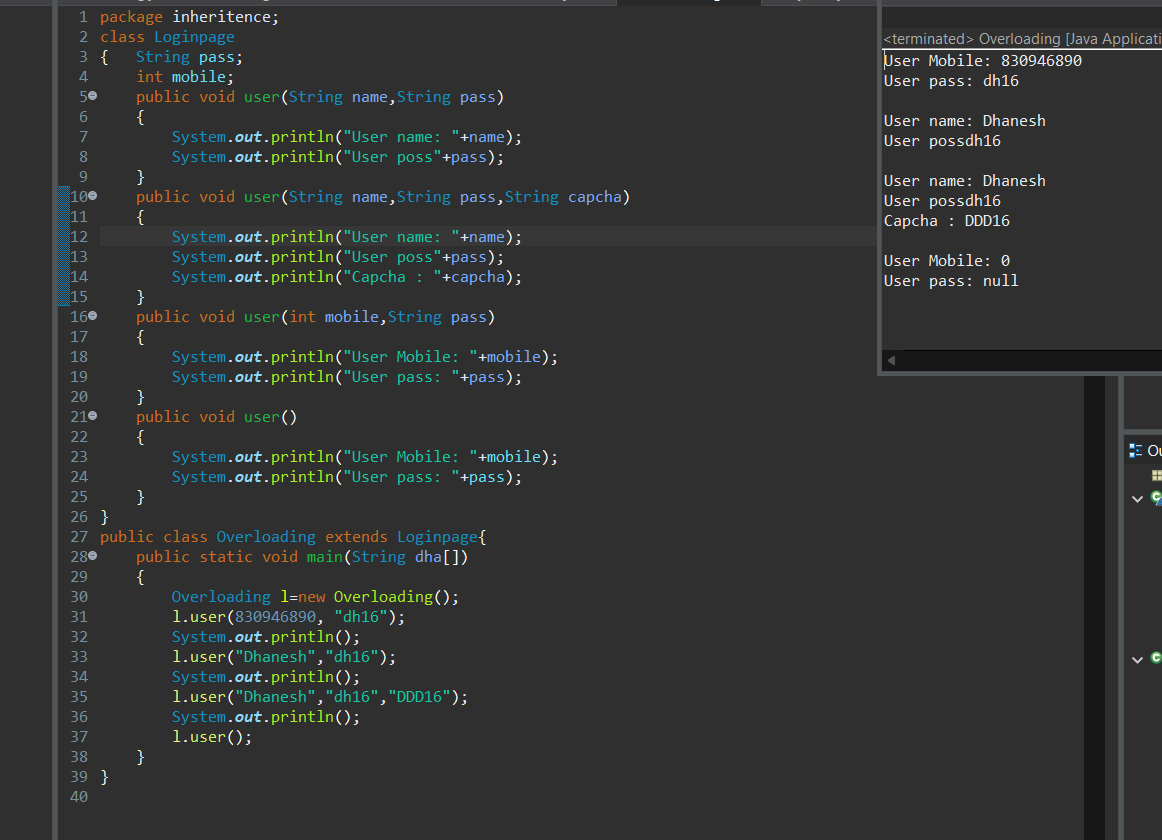
Method name is same, Parameters are different, return time type may or may not be same, will happen in same class or super class

Method name is same, Parameters are same, return time type same, will happen in sub class

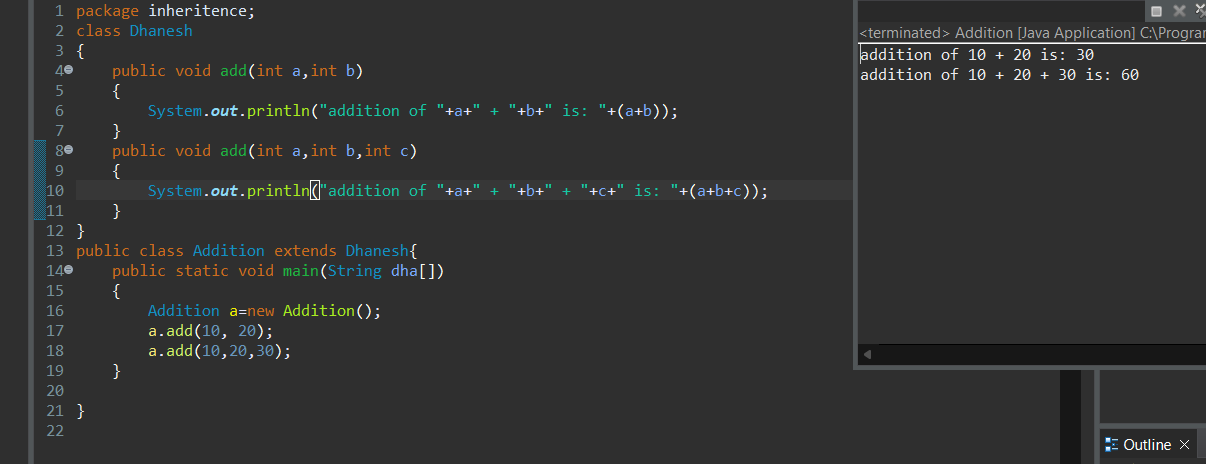
* Interface overriding is best example

**Method Overloading:**

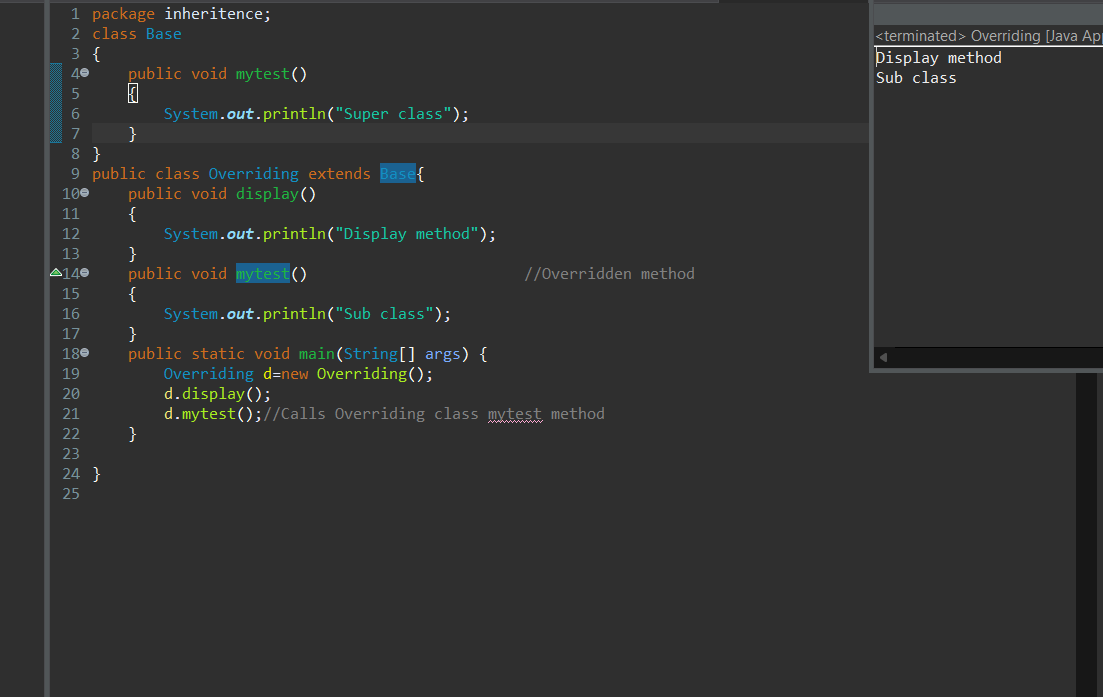
**Login page Example**

****

Addition using Inheritance example



**Method Overriding:**

****

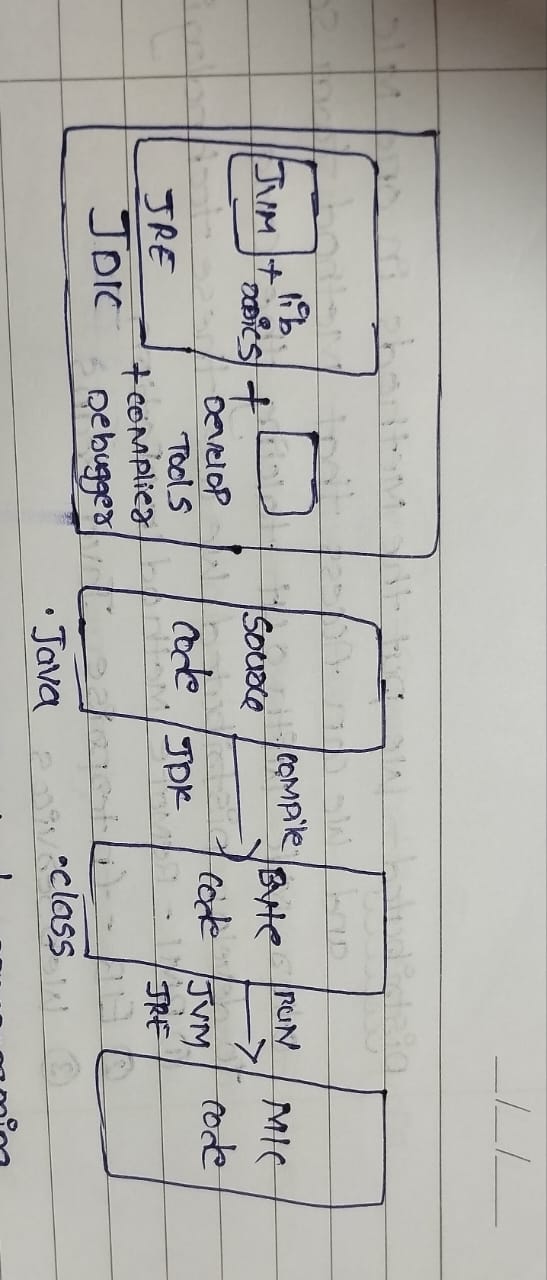
**Abstraction:**

Hiding the un necessary data and show necessary data.

# Day-2: 26/7/2024

**Difference b/w JDK, JRE, RVM.**

* Java Developers need JDK.
* Clients need JRE.
* JVM is a part of JRE.
* JVM converts Byte code into Machine code (Binary Code).

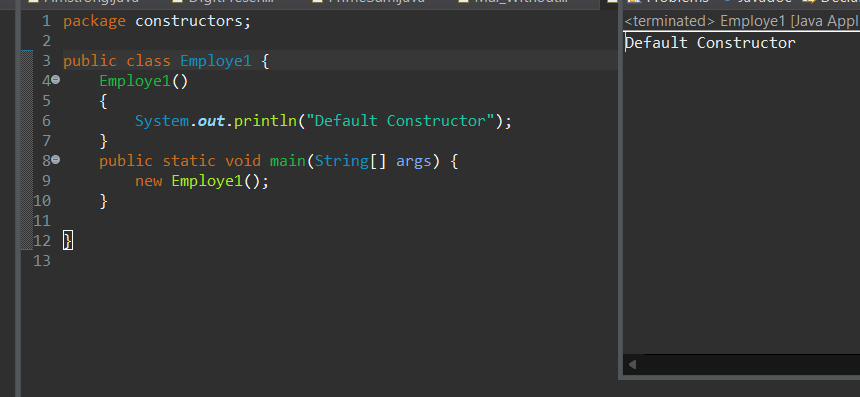


**Packages:**

* Predefined Packages
  + java.lang (default package) No need to import.
  + Java.util
* User Defined Packages
  + package package Name

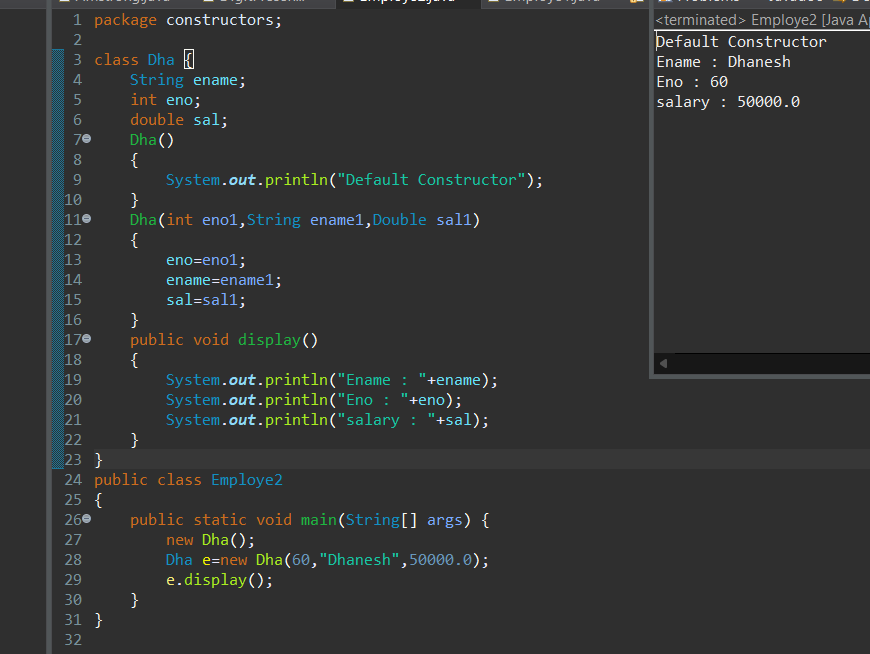
**5. Constructors:**

* + Constructors are mainly for initialization.
  + Class name and constructor name should be same.
  + Constructor will call while creating object.
  + No need of object reference to call default constructors/parameter constructor.
    - new classname();

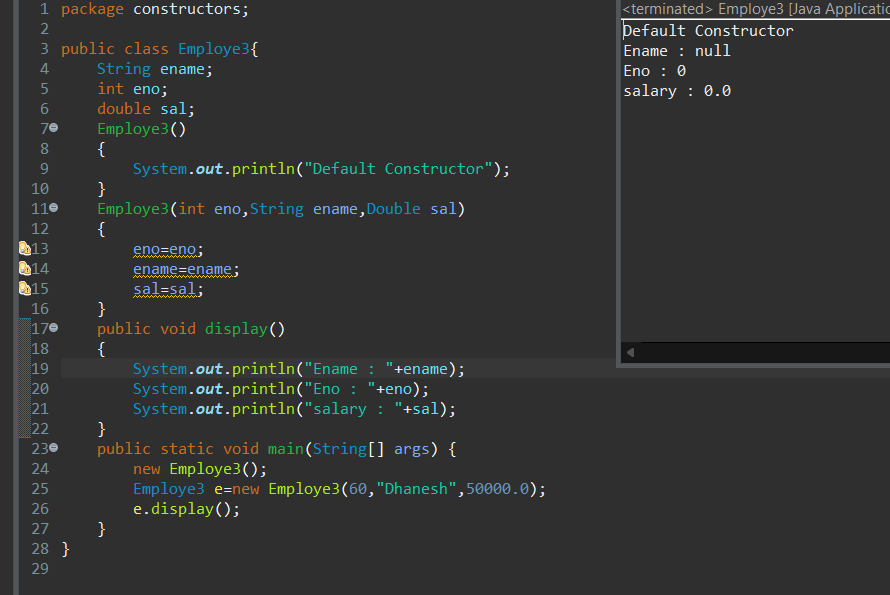


**How many types of Constructors.**

* + There are 2 types of Constructors
    - Default Constructor
    - Parameter Constructor.

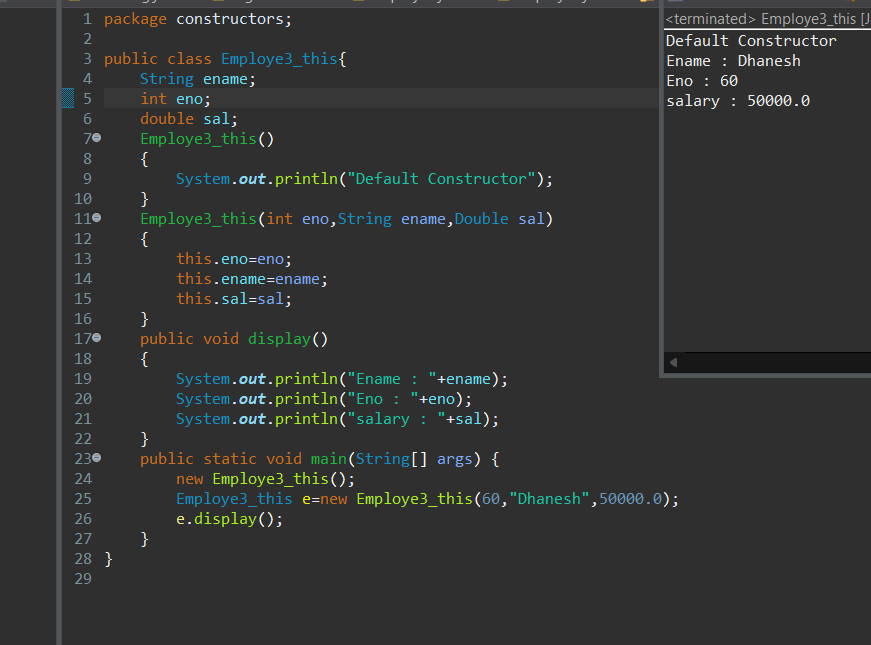


**Without this Keyword**



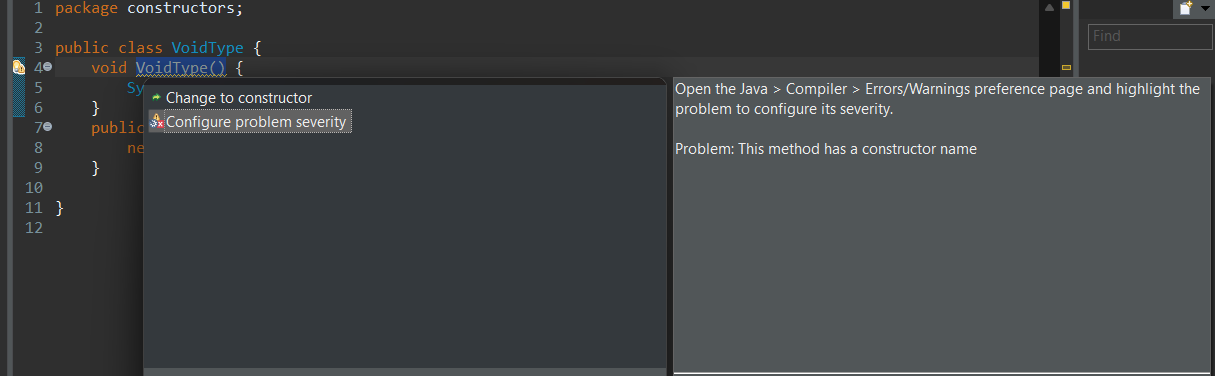
**“this” keyword is pointing to the instance variables.**

**With this Keyword:**

****

**Constructor has Return Type.**

* **Constructor with void Type-----Error**

****

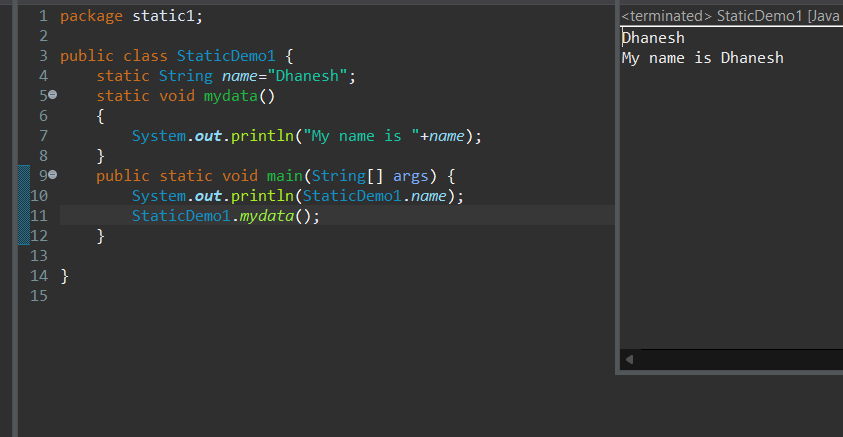
**Questions on Constructors:**

**What is main purpose of Constructor.**

**Can you tell me difference b/w Constructor and Method.**

**Static:**

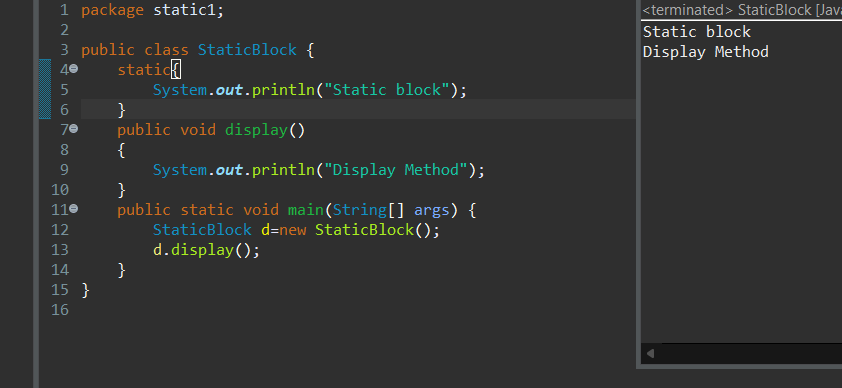
* Static is a keyword
* We can declare variables and as static. Static variables and methods are accessed with object or class in the same class.
* We can access static variables and methods using class name (classname.var\_name, classname.method\_name).



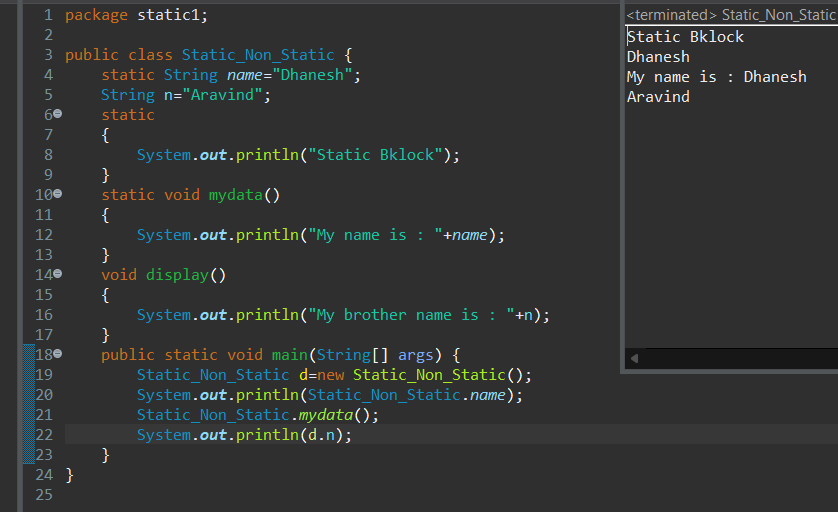
* Static block: will be called while class is loaded first into JVM.

**What is Static Block:**

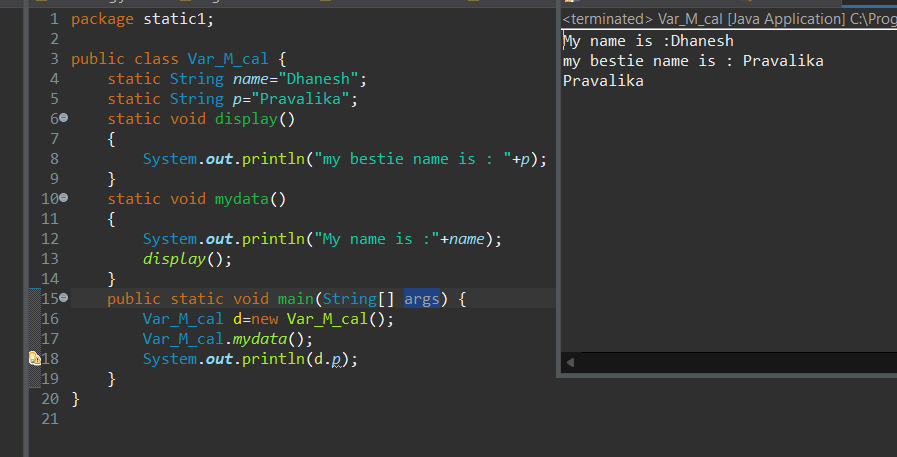
* Whenever the class is loaded into the JVM at that time static block is initialized first only.
* The static block is always first loaded and executed.
* When we are created object, it consumes more memory .so we use static it changes the original memory.



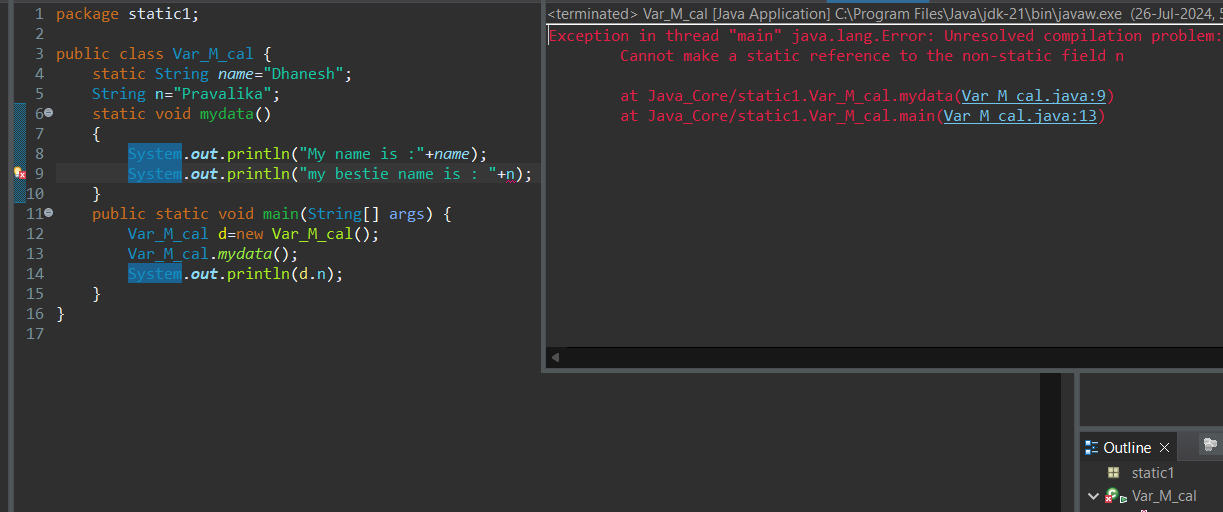
**Static Block and Non Static**



* Static Methods can call only static variables and static methods only.



* Static Methods cannot call non static variables and non\_static methods only.

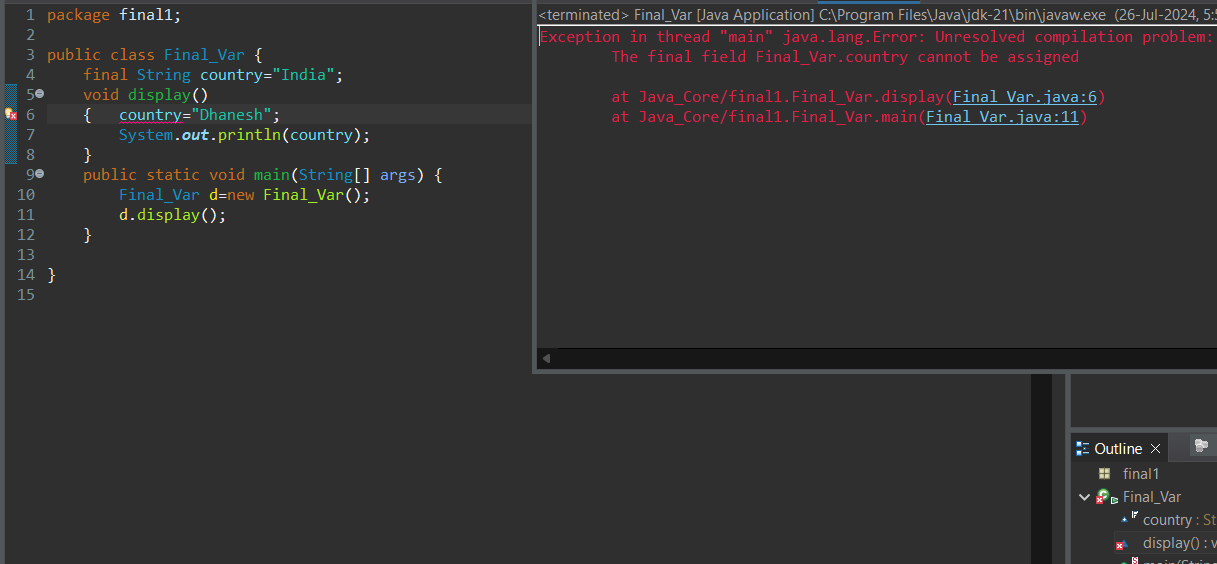


* Non Static Methods can call static variables and static methods.

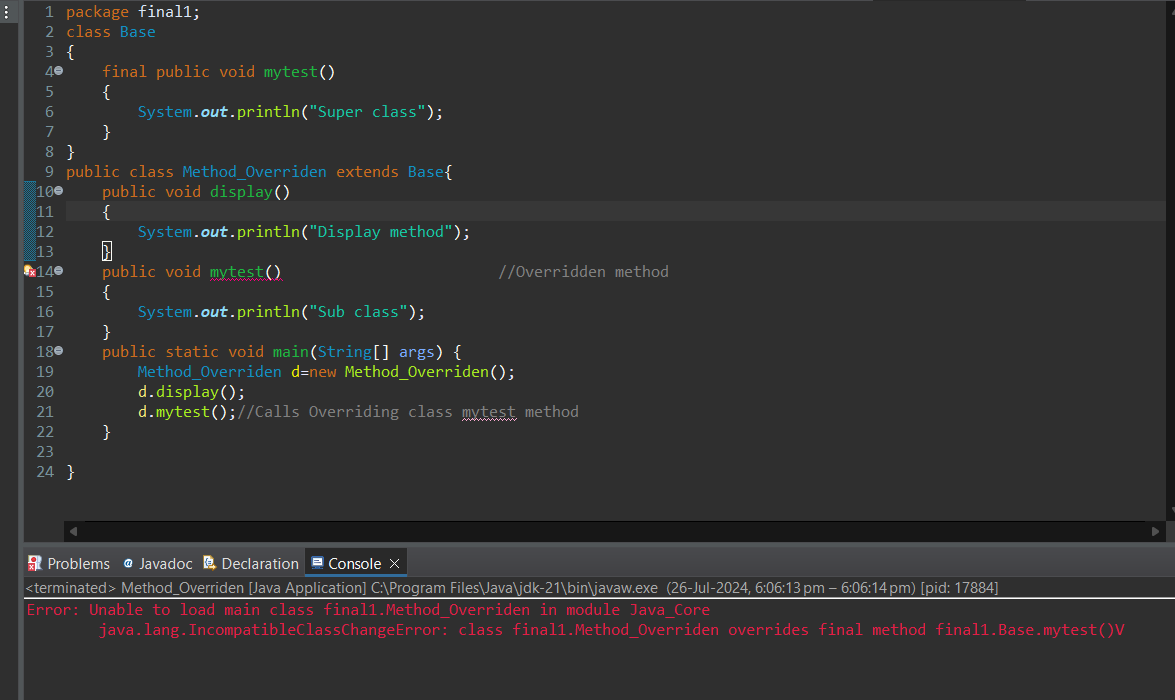


**final**

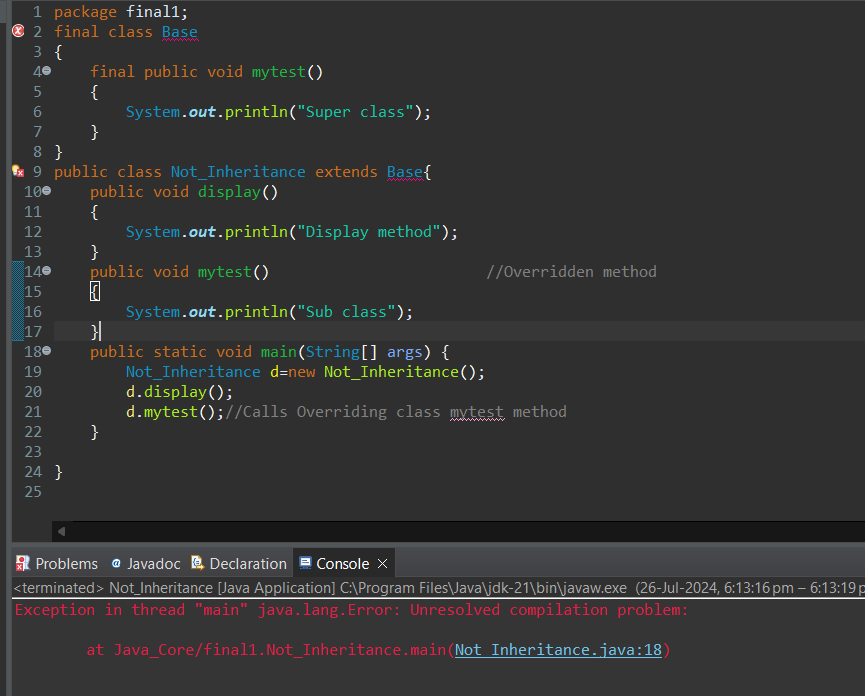
* final is a keyword.
* We can declare variables, methods, classes as final.
* **Final variables cannot be modified.**
* Final class methods are accessed throughobject reference.



* **Final methods can not be Overridden.**

****

* **Final class cannot be inherited**
* **We cannot override methods of a final class**

****

**Object class**

* Object is the super class of all classes.
* Object class is a top most class.
* Any class in java is a sub class of Object Class.
* Object class is also called as Cosmic super Class.

**Methods in Object class:**

* **clone ()**: used to create a copy of object

Eg: Emp e = new Emp ();

Emp e1 = e.clone();

* **equals()** :used to compare hashcode of two objects

eg: sop(e.equals(e1));

* **finalize():** used for cleaning operation like closing the file and database connections.

-it is called just prior to the garbage collection

Eg:

Public void finalize(){

Out.close();

Con.close();

}

* **getClass()** : used to return the class name of an object

eg: sop(e.getclass());//class name

* **hashcode():** used to return the hash code of an object

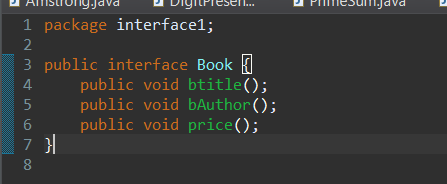
eg: sop(e.hashCode());

* **toString()**: return the class name of object along with hashcode;
* wait()
* wait(long)
* wait(long,int)
* notify()
* notifyAll()

**Day-3:27/7/2024:**

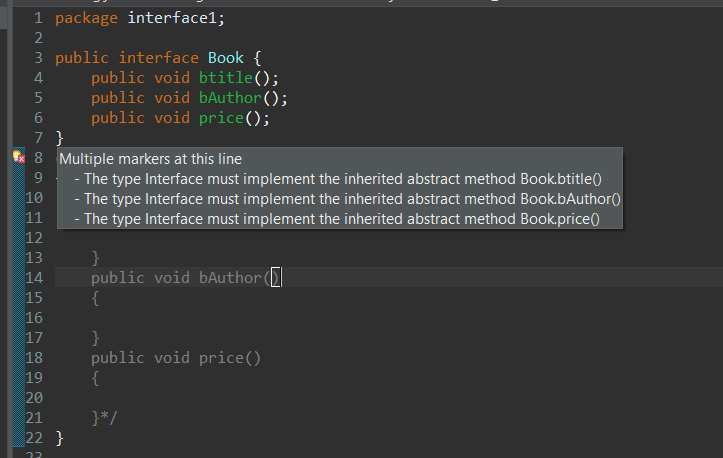
**Interface:**

* Interface is a keyword.
* Interface having only method signature only. there is no implementation or body.

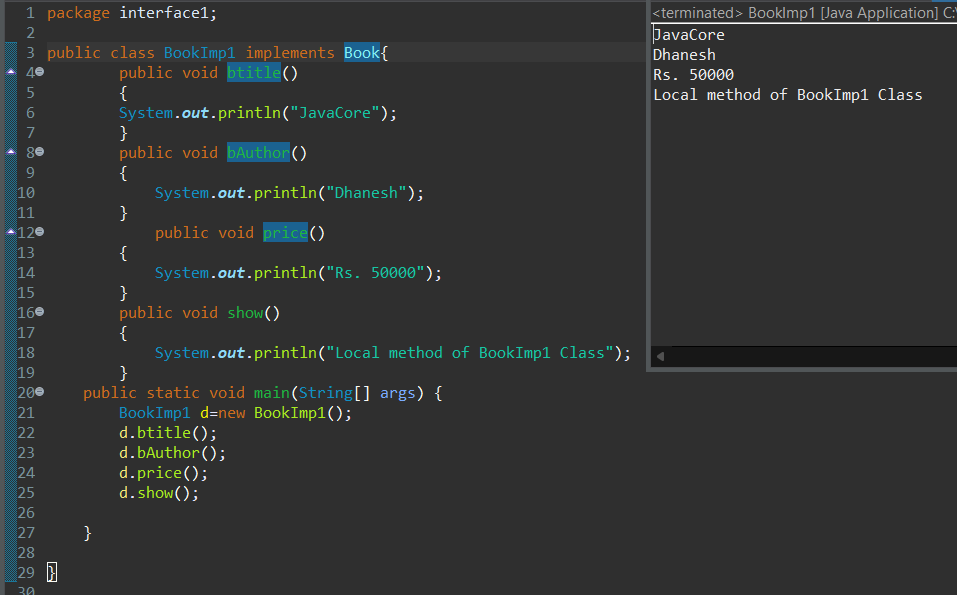


* If any class implements the interface that class should be override all the interface methods otherwise the class will showing compile time errors.

**Without overridden methods shows errors**

****

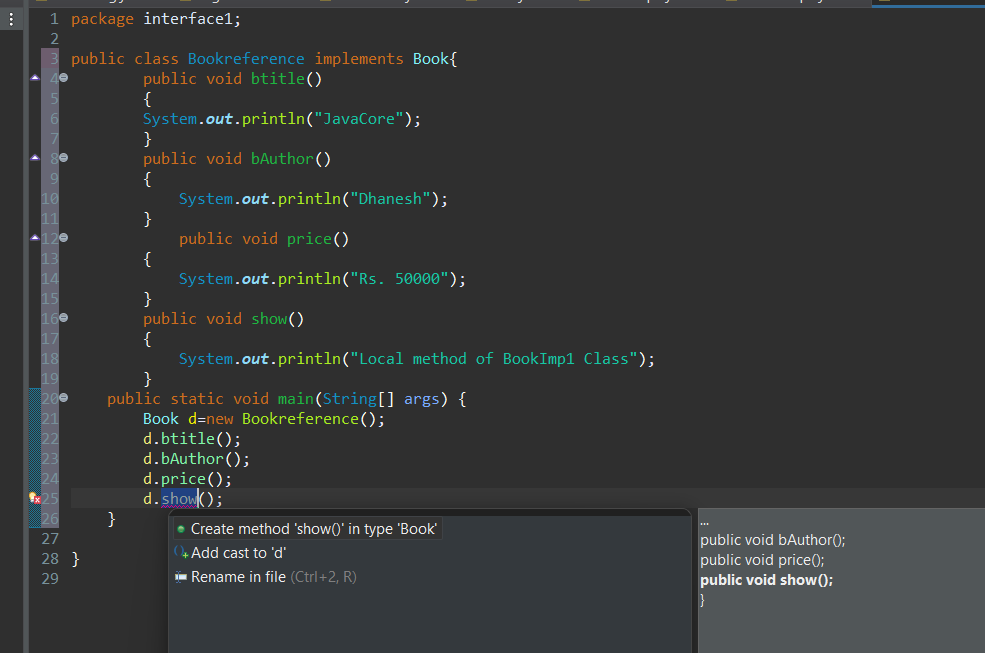
**With Overridden Methods No Errors**

****

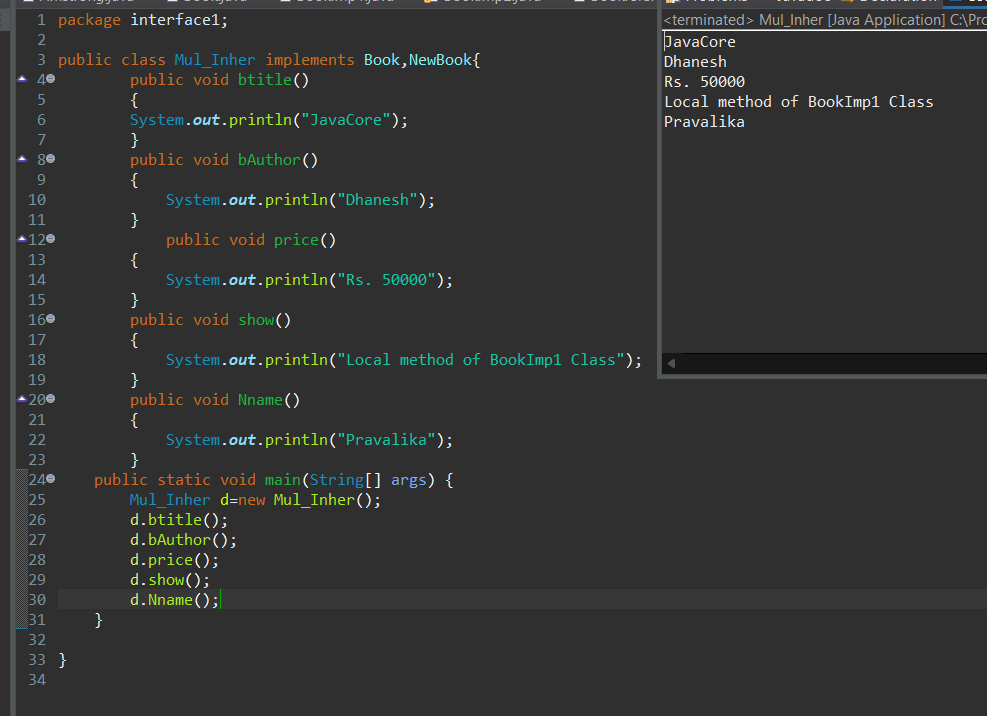
* We can not create object for interface classes but we can create object reference for interfaces. (reference only access Interface methods)



**We can not access non Interface methods through reference**

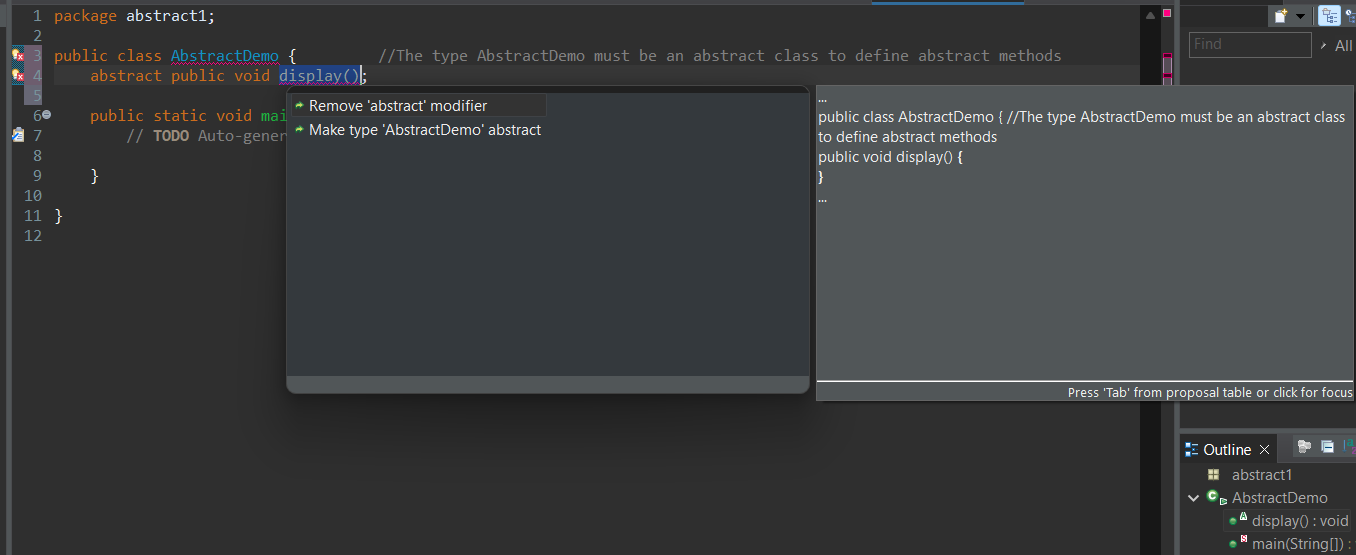
****

* All interface methods are by default **public and abstract.**
* Only we need Interface methods then we go for reference of interface.
* Java will support multiple inheritance through interfaces.

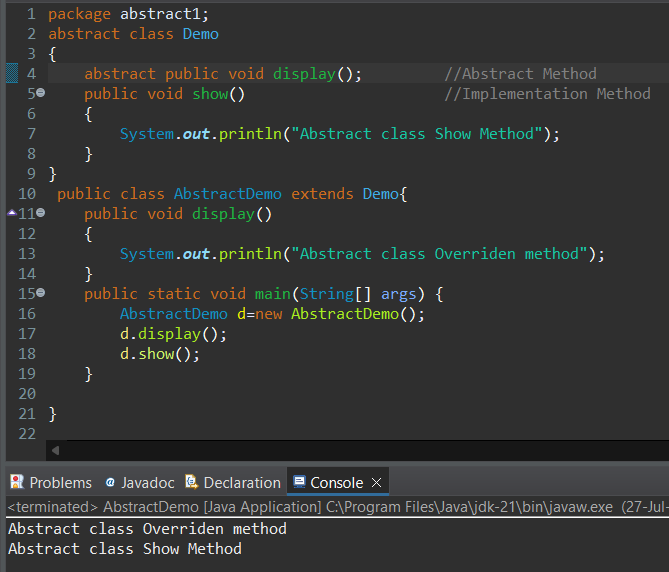


**Abstract class:**

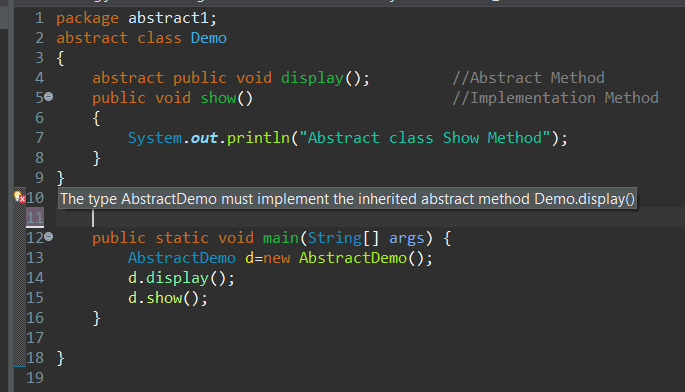
* Abstract is a key word
* If any class having one abstract method that class should be declared as **abstract class.**



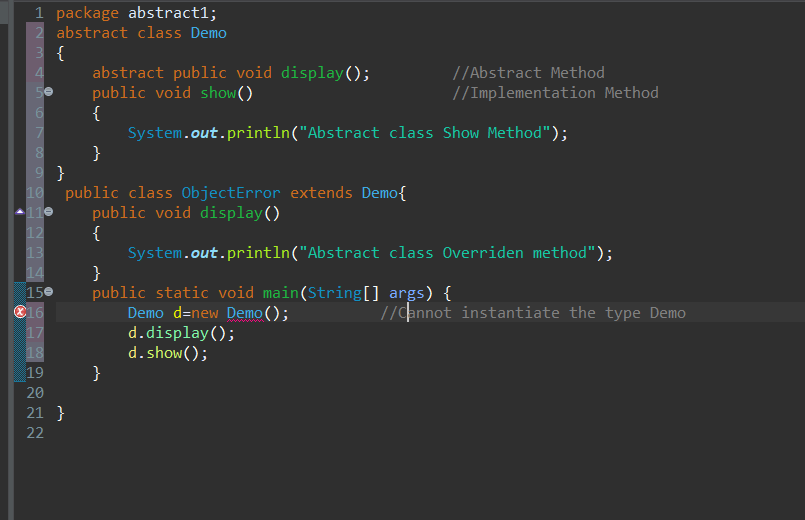
* Abstract class having abstract methods and implemented Methods.



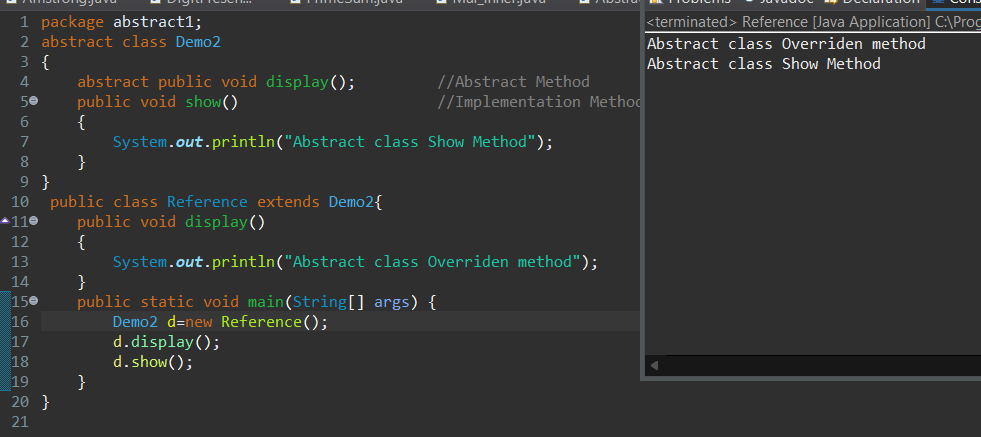
* if any class extends abstract class that class should be override all abstract methods otherwise it shows Compile time error.



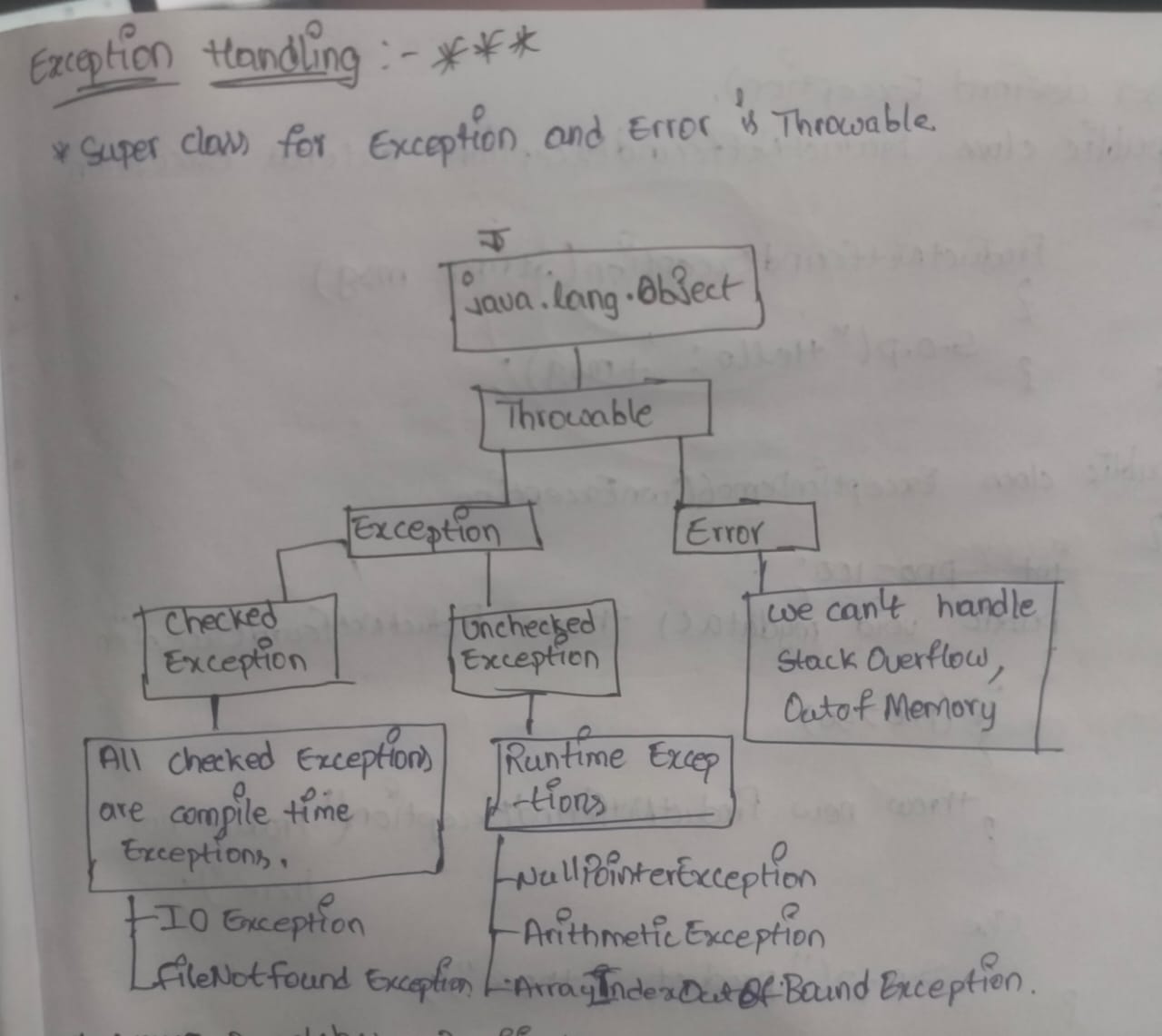
* Can we create object to abstract class?
  + No, because it is not fully implemented.



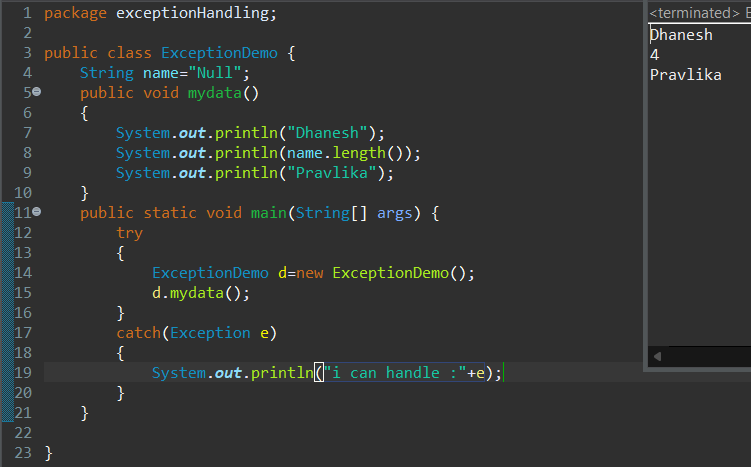
* But we create reference to abstract class.



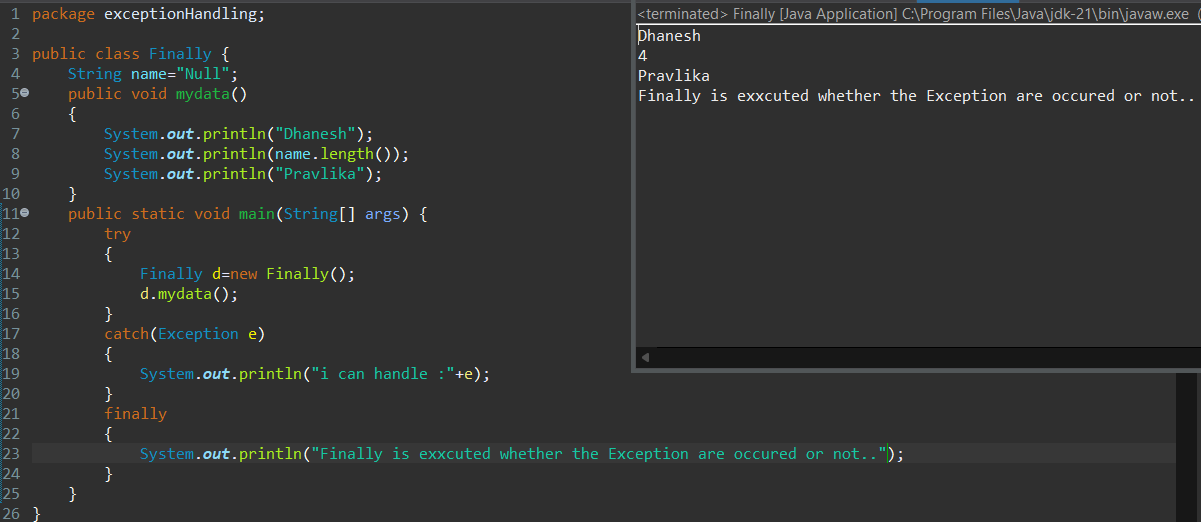
**Exception Handling:(\*\*\*-Written Test, Interview, DevelopMent)**



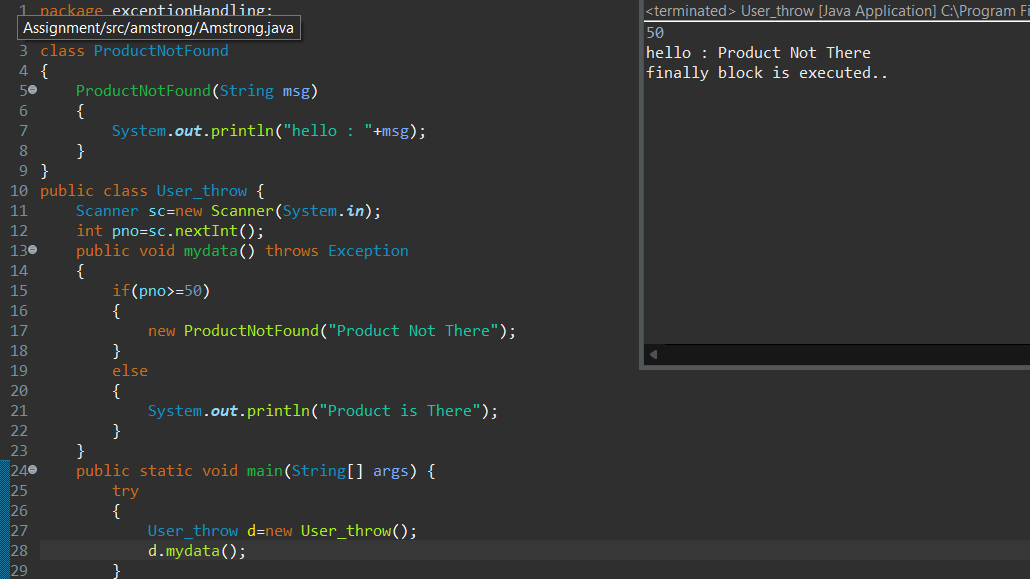
* There are mainly 5 key Words
  + try
  + catch ()
  + finally
  + throws
  + throw
* Normal flow will not execute any catch block. Abnormal flow only catch block will be executed.



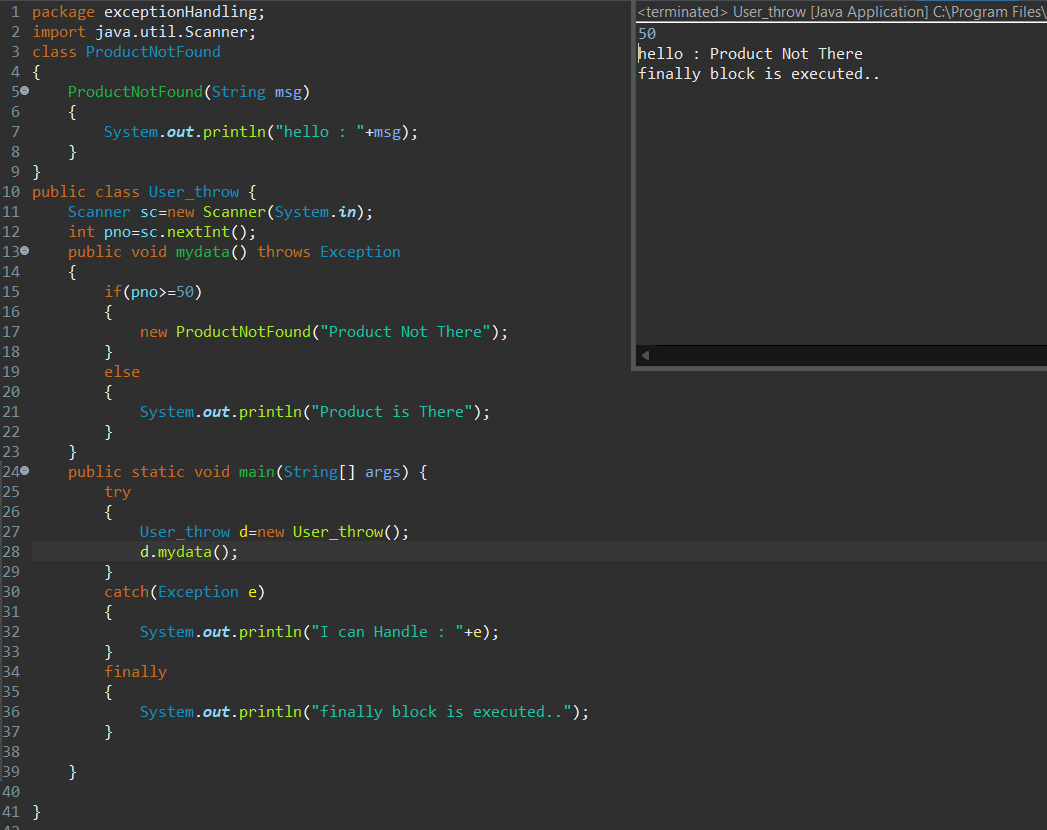
* Try block should be followed catch block or finally block.
* Finally block will be executed whether Exception occurred or not.



* Throws is a key word. Throw will be executed layer by layer (or) Method by Method.
  + Throws is a checked Exception-Compile Time.



* Throw is a key word. Throw will be executed for Runtime and User Defined Exceptions.
  + Throw is a Unchecked Exception-Runtime.



**RUNTIME EXCEPTIONS:**

* 1)Arithmetic exception -divided by zero
* 2)ArrayindexoutOfBoundException -access element out of index
* 3)StringIndexOutOfBoundException -access when element out of index
* 4)NegativeArraySizeException -when we initialize with negative index
* 5)NumberFormatException -when we try to convert a string to a numeric which is not possible.
* 6)NullPointerException -any obect pointing to null by using that object if you access method of String class then is throw exception

**String:**

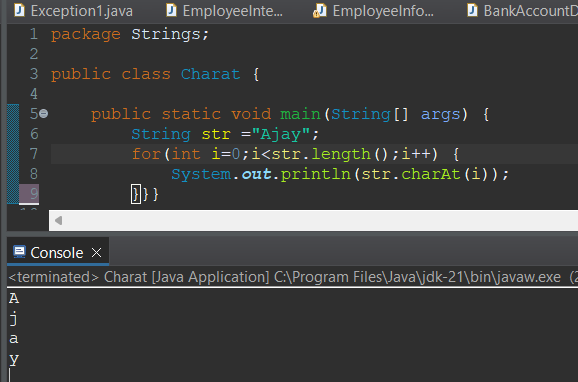
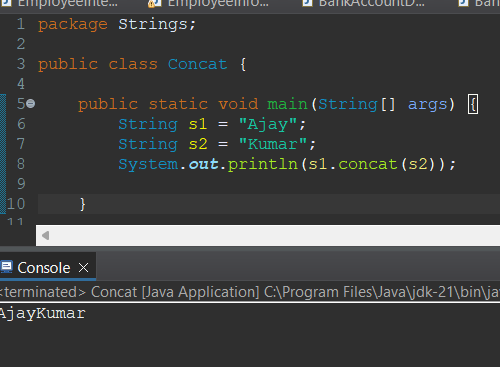
* A string is a final class it is present in java.lang.String package.
* String is a group of characters
* String is immutable
* Ways to create a string:
* String str = new String(“Ajay”);
* String str = “Ajay”;

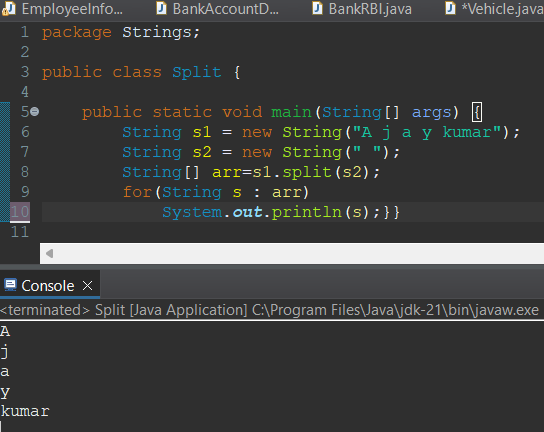
**Note:**

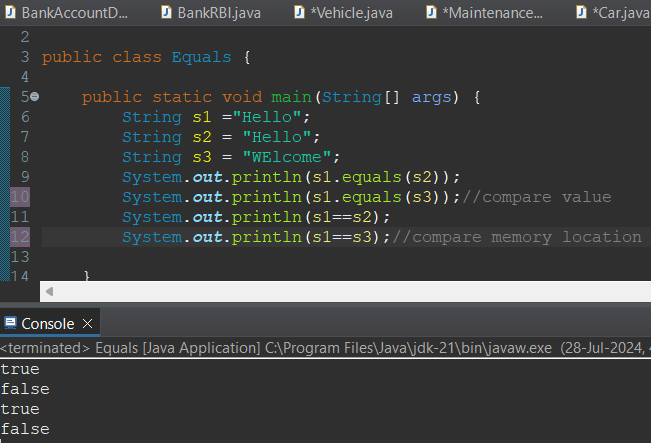
== -> Compares the memory location of two objects.

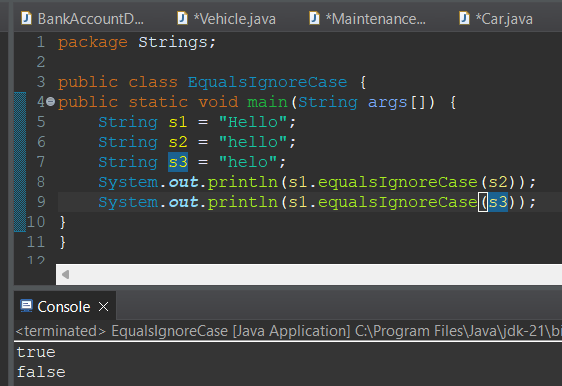
equals -> compares the content of two objects

**Methods of String Class:**

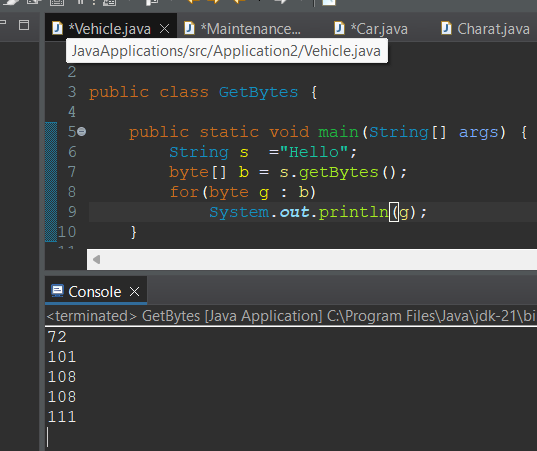
* char charAt(index) – used to access each character in a **String**
* **String concat(String):** combine two Strings
* **split(String):**

By using split() method we can divide the string in to tokens.

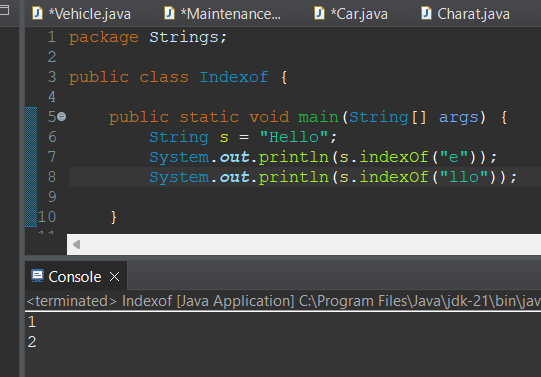
* **Boolean equals(String):** checks if two strings are equal or not.
* **Boolean equalsIgnoreCase(String):** Checks If two Strings are equal ignoring case differences.

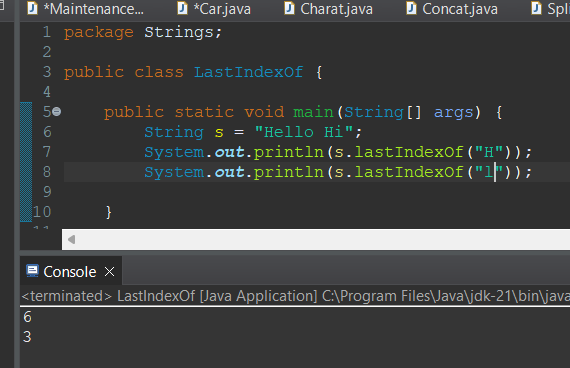
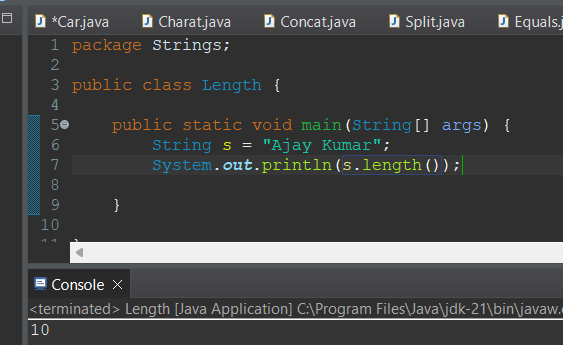
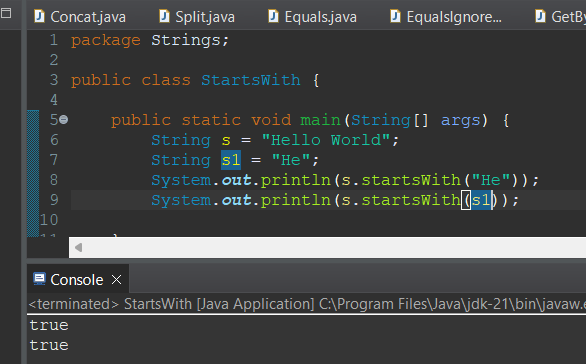
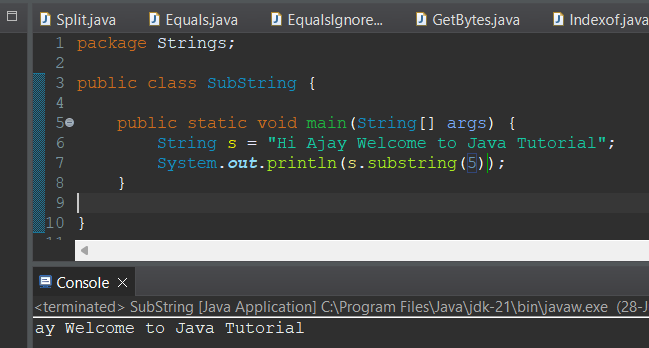
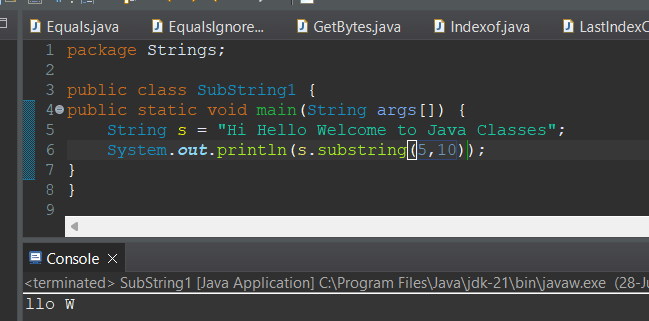


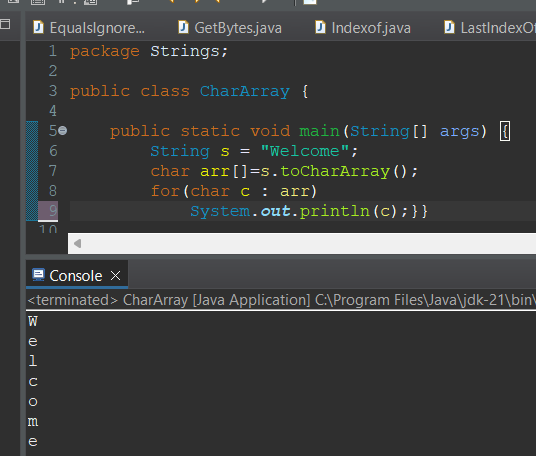
* **getBytes():** Converts a String to a byte array



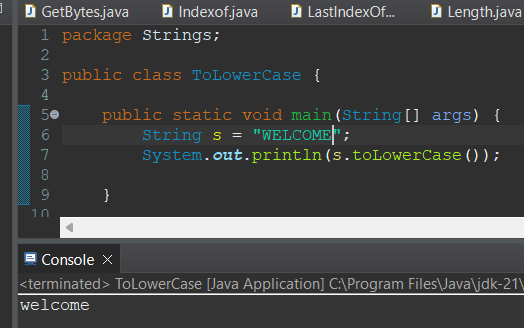
* **int indexOf(string):** Returns the index within the string of the first occurrence of the specified substring.



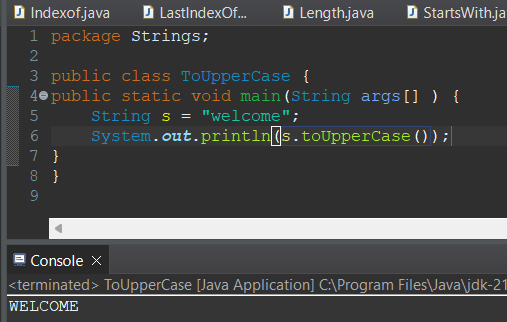
* **int LastIndexOf(string):** Return the index within the string of the last occurrence of the specified substring.
* **int length():** Returns the Length of the string
* **boolean startsWith(string):** checks if the string starts with the specified prefix.
* **String substring(int):** Returns a new string that is a substring starting from the specified index to the end
* **String substring(int,int):** Returns a new String that is a substring starting from the specified start index to the end index.
* **Char[] toCharArray():** Converts the string to a character array.

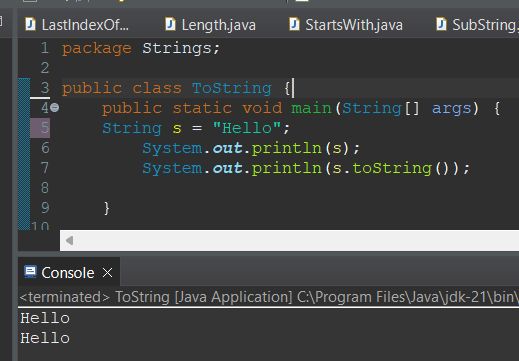
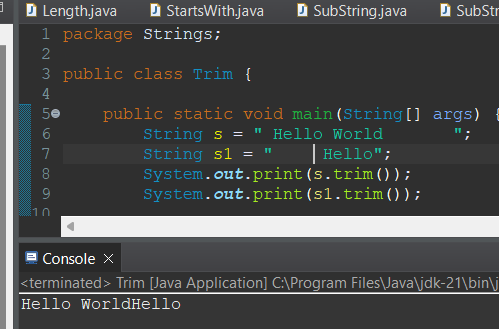
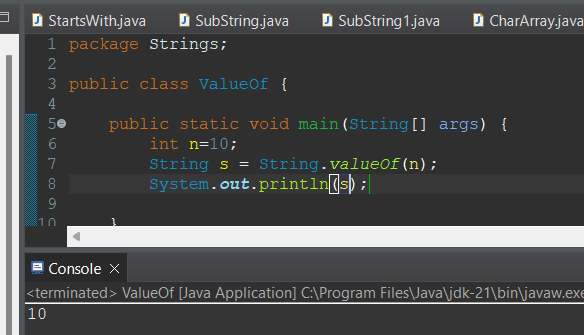


* **String toLowerCase():** converts all characters in the string to lowercase.

****

* **String toUpperCase():** Converts all characters in the string to Upper Case



* **String toString():** Returns the String itself
* **String trim():** removes Beginning and Ending white spaces.
* **String valueOf(int):** Converts the specified value to a string.

**Why Strings are immutable:**

-When we create a string object it is constant, if we are trying to modify existing string it will create another memory location.

-existing object is eligible for garbage collector.

**Where it will store:**

-When we create with new it store in heap

-No new keyword store in constant pool

Interview Questions:

1)What is String?

String is a final class it is immutable.

2)Why string is immutable?

3)Difference between String and StringBuffer , StringBuilder?

**String:**

-immutable

-All methods are not synchronized

**StringBuffer:**

-Mutable

-All Methods are synchronized

**StringBuilder:**

-Mutable

-All Mwthods are noy Synchronized

**Multi-Tasking:**

**IRCTC**

**APP**

**OS**

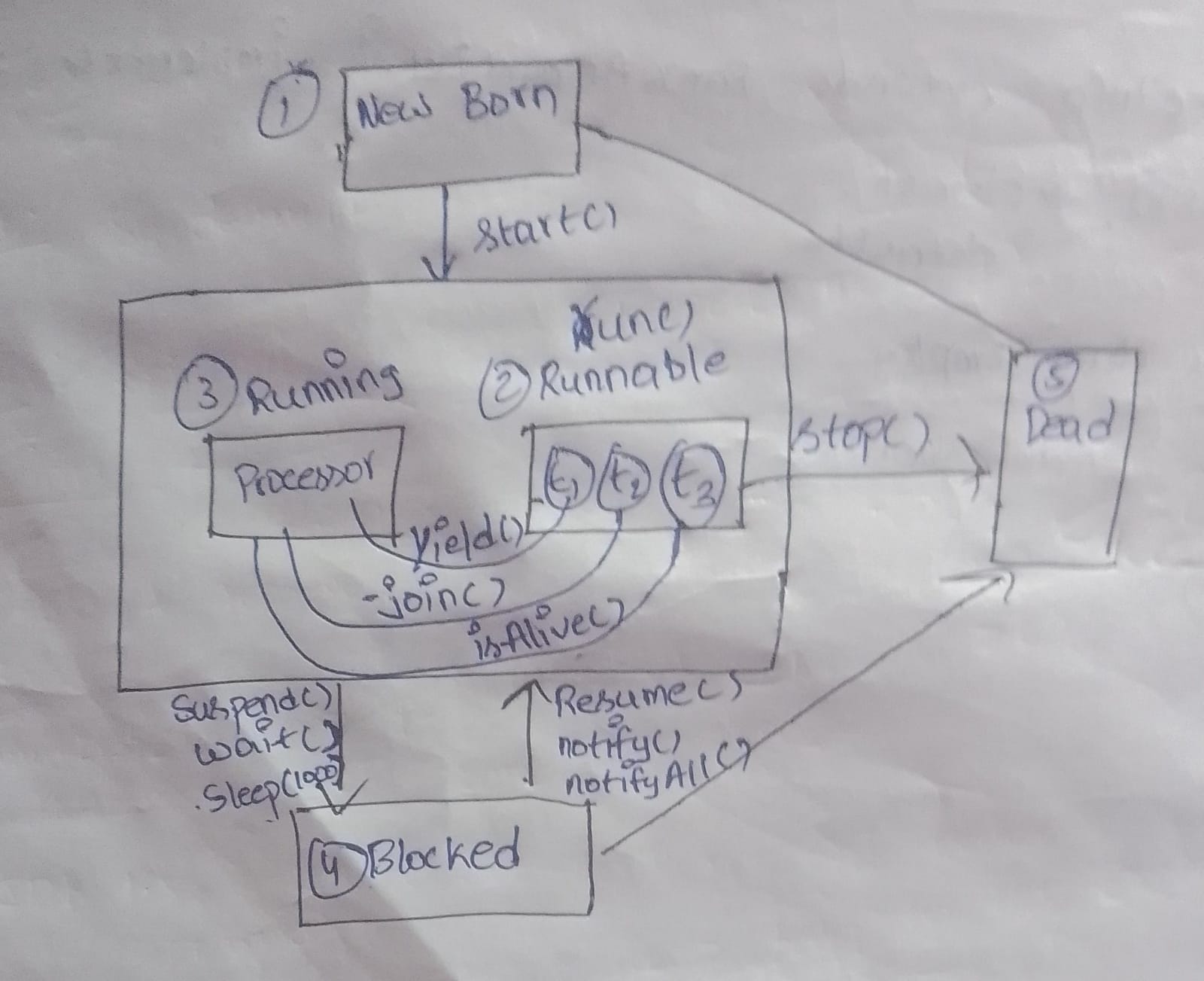
**Day-4:**

**Multithreading:**

-Multiple Threads Are Running Simultaneously Reducing CPU Ideal Time.

- Multithreading is used to increase the performance of application

-A Thread Is a Light Weight , These Threads are Running on a Separate Path



-Threading is inbuild Mechanism

-Object is Heavy weight

-Thread scheduler allocate the time for threads

-isAlive() : Thread is running or not (It returns Boolean Values).

-Yield(): If a Thread execution is completed or stopped within the time slot. The remaining time is allocated for next priority thread.

-join() : Communication Between Main And Child Class

-Suspend(),Resume() combine one.

Java.lang.Thread(is a class)

Java.lang.Runnable(interface)

**How Many ways to create a Thread:**

-We can create a Thread in two ways

1)extends Thread class

Ex:

eg:

class MyThread extends Thread

{

public void run()

{//logic goes here

}

}

MyThread t = new MyThread();

t.start();

2)implements Runnable interface

Runnable

---------

class MyThread implements Runnable

{

public void run()

{//logic goes here

}

}

**Methods of Thread class:**

------------------------e----------

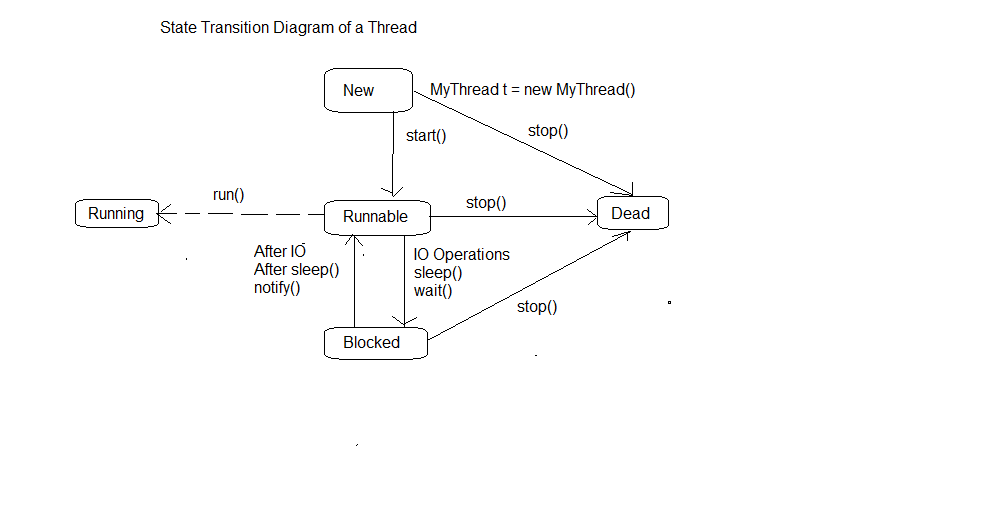
1) **Thread currentThread():** This static method returns a reference to the currently executing thread object.

Advantages of Runnable interface over Thread class

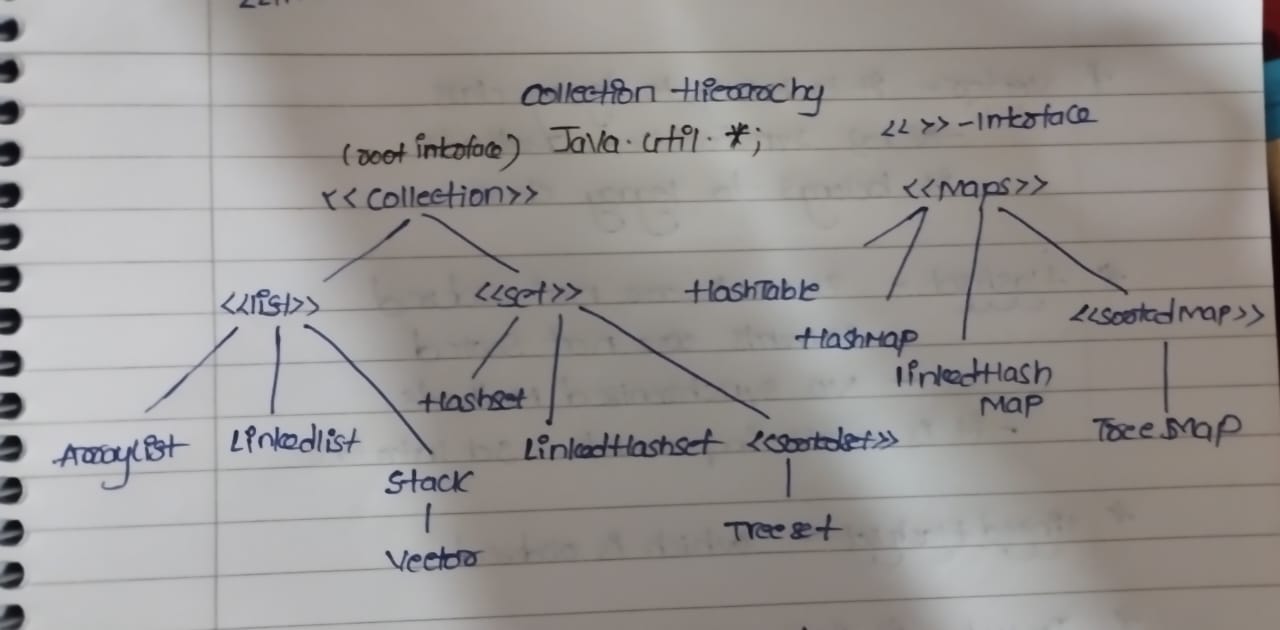
--------------------------------------------------

- If a class already extends another class, to make this class as thread, the class should implement Runnable interface as Java does not support multiple inheritance

- Ensures that run() is overridden

**Life Cycle of a Thread:**

**Collection Framework:**

****

**What is collection Framework:**

* + it is a predefined framework it will present in the java.util.Collection.
  + It will Allow different data type
  + Increase/Growing their memory location
  + Business Logic Methods Can be implement (Add,delete).

**Collection(interface):**

-it is an root interface

**List(interface):**

-It is a Interface which extends Collection interface.

-List allow duplicate elements

-List is Ordered

-The List size is growing and allow any type of data.

**Methods in List Interface:**

**-**boolean add(object)

-void add(int, object)

-void set(int,object)

-boolean remove(int)

-object get(int)

-int size()

**Arraylist(class):**

-ArrayList is a class which implements List Interface.

-It is a continuous Memory location

-There is no synchronized Methods

-if initial size is 10 after filling the 10 elements its size increase to 15. (Initial size+(size/2))

Dis adv: if we want to store only 11 elements the remaining memory is waste.

**Linked List(class):**

-Linked List is a class that implements the List Interface.

-the is no continuous memory location in LinkedList

-The Methods are not synchronized

-If we want to add & remove an element we can use the Linked List

LinkedList Method

Contains(SearchEl)

Retrieve

Adding

Remove

.add(ele) remove() get(int index)

.add(index,ele) remove(int index) getFirst()

.addFirst(ele) removeFirst() getLast()

.addLast(ele) removeLast() size()

* **add(E e)**: Appends the specified element to the end of the list.
* **add(int index, E element)**: Inserts the specified element at the specified position in the list.
* **addFirst(E e)**: Inserts the specified element at the beginning of the list.
* **addLast(E e)**: Appends the specified element to the end of the list.
* **get(int index)**: Returns the element at the specified position in the list.
* **getFirst()**: Returns the first element in the list.
* **getLast()**: Returns the last element in the list.
* **remove()**: Removes the first element from the list.
* **remove(int index)**: Removes the element at the specified position in the list.
* **removeFirst()**: Removes the first element from the list.
* **removeLast()**: Removes the last element from the list.
* **size()**: Returns the number of elements in the list.

**Vector:**

-Vector is a class implements list interface

-Vector Is a legacy class.

-The Methods Inside the Vector or Synchronized.

- if initial size is 10 after filling the 10 elements its size increase to 20. (Initial size\*2))

- Dis adv: if we want to store only 11 elements the remaining memory is waste.

**Set(interface):**

-Set is an interface which extends Collection Interface.

-Set will not allow duplicates.

-it is unordered.

-Size is growing allow any type of data.

**Hashset(class):**

-Hashset is a class which implements Set Interface.

-elements are not ordered.

-elements are not sorted.

-no Synchronization methods

**LinkedHashSet(class):**

-it is ordered.

-it has no synchronization methods.

**Treeset(class):**

-it is class which implements the Set interface

-it have synchronized methods

-it belong to legacy class

Note: in tree set all elements should be of same type.

**Maps(interface):**

-map is an interface which is present in java.util.Map

-Map is in form of key value pair.

**HashMap(class):(JDK 1.5)**

-HashMap is a class which implements the Map interface.

-it is present in java.util.HashMap.

-it is in the form of key value pair.

-it allow null key & nullvalue.

-All methods are not synchronized.

-it is not ordered.

**Hashtable(class):(JDK 1.O)**

-Hashtable is a class which implements map interface

-it is in the form of key value pair.

-it don’t allow null key and null value.

-all methods are synchronized and it is a legacy class

**Linked HashMap(class**)

-it is a class which implements the map interface .

-it is ordered based on keys.

**Treemap:**

-it is sorted

-all methods are synchronized.

**Iterators:**

------------

-iterators are used to iterate in collection hierarchy.

- iterator is used to retrieve the data from object class.

It is of three types

1)Iterator

2)ListIterator.

3)Enumeration.

**Iterator:**

-It is used to iterate in a collection hierarchy.

-it has remove method.

-it will move in forward direction.

Methods:

-boolean hasnext()

-object next().

**ListIterator:**

-it is used to iterate in a collection hierarchy.

-it has both add and remove methods.

-it will move in forward and backward direction.

-initially it points to center based on user preference it will move to start or end.

Methods:

-boolean hasNext()

-object next()

-boolean hasPrevious()

-object previous()

**Enumeration:**

-it is legacy class

-all the methods are synchronized.

-it will use only to retrive the data not to perform any action.

**Special Guest:**

**Generics:**

-if we want to store particular type of elements we use generics.

-from jdk 1.5 onwards they introduce generics

**Enhanced for loop:**

-it will used to loop over the collections if we have same type of data.

**-**we use enhanced for loop when we are using collections with generics.

**Arrays vs Collections**

----------------------

- In Arrays the size is fixed where as in Collections the size is not fixed

- In Arrays we can collect only similar type of elements where as in Collections we can collect different type of elements

- In Arrays we can store primitive type as well as reference type of elements where as in collections we can store only reference type of elements

**Collection vs Map**

-----------------

- Collection is a collection of elements where as Map is a collection of key-value pairs

**List vs Set**

-----------

- List is ordered where as Set is unordered

- List allows duplicate elements where as Set does not allow duplicate elements

**ArrayList vs LinkedList**

-----------------------

- In ArrayList the elements are stored in continuous memory locations where as in LinkedList the elements are stored in non-continuous memory locations

- The cost of insert and delete operations are more in ArrayList where as the cost of insert and delete operations are less in LinkedList

**Stack**

-----

- A Stack is collection of elements in the form of Last In First Out (LIFO) operations

**Operations on Stack**

-------------------

- push -> inserts an element into the stack at the top

- pop -> deletes an element from the stack at the top

- peek -> retrives the top element from the stack

**ArrayList vs Vector**

-------------------

- In ArrayList the methods are not synchronized where as in Vector the methods are synchronized

- ArrayList is not thread safe where as Vector is thread safe

**HashSet vs LinkedHashSet**

--------------------------

- In HashSet the elements are not ordered where as in LinkedHashSet the elements are ordered

**HashSet vs TreeSet**

------------------

- HashSet is not sorted where as TreeSet is sorted

**HashMap vs Hashtable**

--------------------

- In HashMap the methods are not synchronized where as in Hashtable the methods are synchronized

- HashMap is not thread safe where as Hashtable is thread safe

**HashMap vs LinkedHashMap**

------------------------

- In HashMap the elements are not ordered where as in LinkedHashMap the elements are ordered based on keys

**HashMap vs TreeMap**

------------------

- In HashMap the elements are not sorted where as in TreeMap the elements are sorted based on keys

**Additional LinkedList methods**

-----------------------------

- void addFirst(Object)

- void addLast(Object)

- Object removeFirst()//it is faster than index values

- Object removeLast()