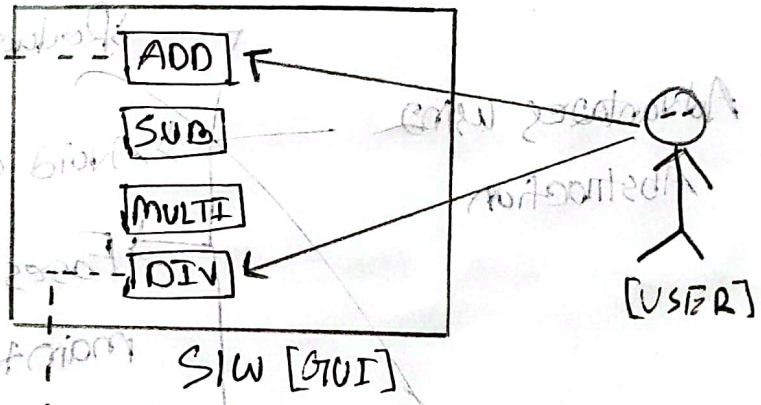
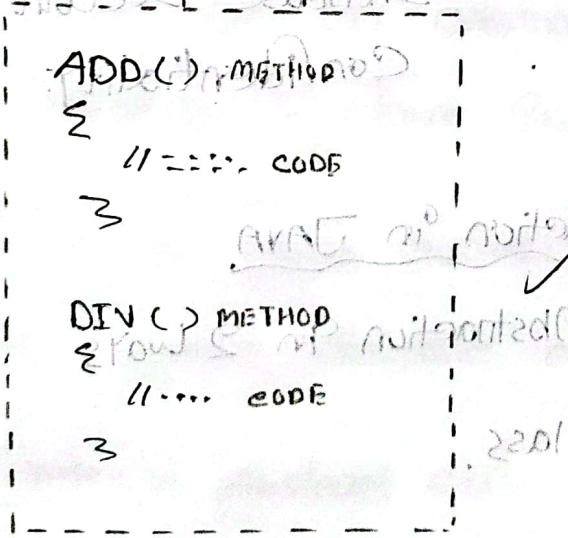
Abstraction

What is Abstraction ?

Abstraction is a Process of hiding the implementation details from the user, only the highlighted set of services Provided to the user.



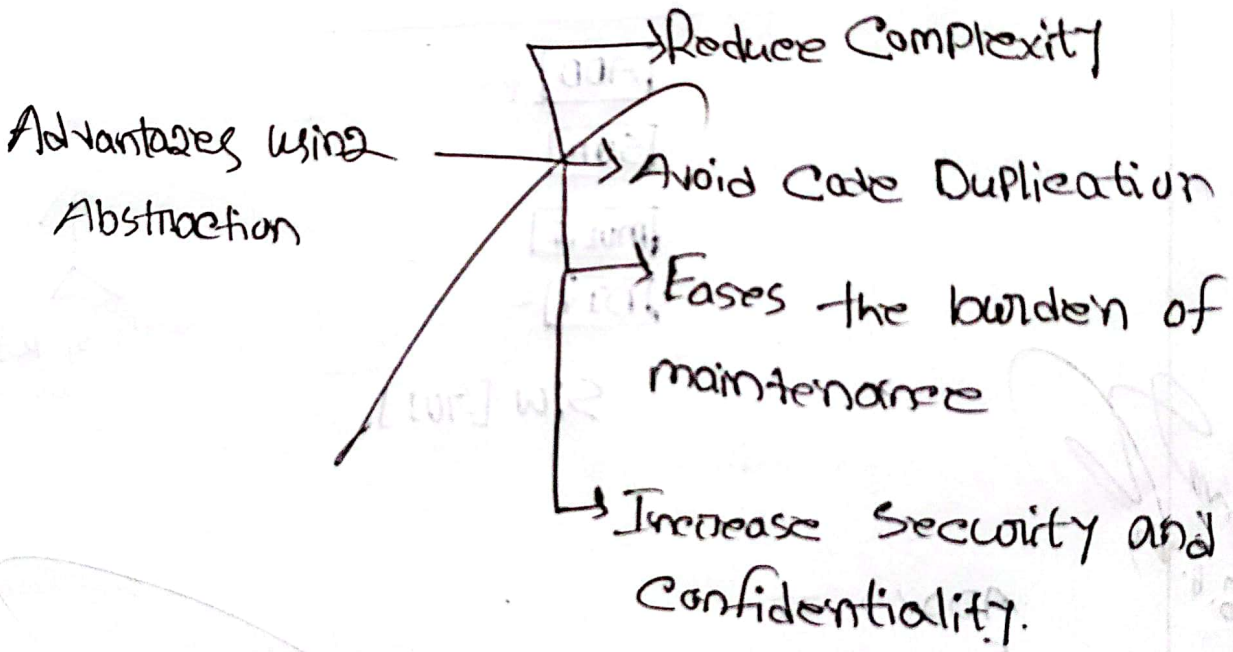
Pavan 6.6.23



By the using is Abstraction concept, The back-end will be hide from user. As a result user can't see or don't

know about backend code whether written in Java or others Programming languages or not.

Advantages using Abstraction in JAVA



Implementing Abstraction in JAVA

We can implement Abstraction in 2 ways

i) abstract class.

ii) interface.

Abstract class in JAVA

What is Abstract class ?

A class which contains the abstract keyword in its declaration is called abstract class.

NOTE: i) We can't create object for abstract class.

ii) It may or may not contain abstract methods.

iii) It can have abstract and non-abstract methods.

iv) To use an abstract class, you have to inherite it from Sub classes.

v) If a class contain partial implementation then we should declare a class as abstract.

Syntex of Abstract class

```
abstract class A
{
    -----
    -----
}
```

We can't create Object for abstract class -

Example:

```
class animal
{
}
class Demo
{
    public static void main(String[] args)
    {
        animal obj = new animal();
    }
}
```

Output

ERROR.

```
abstract class animal
{
    animal() // Non-abstract method
    { System.out.Println("All animals ....!");
    }

    Public abstract void Sound(); // abstract method
}                                 + Common Properties

Class Dog extends animal
{
    Dog() // constructor
    {
        super(); // To access Non-abstract method.
    }
    @Override
    Public void Sound() // Method fun access Sound() abstract method
    {
        System.out.Println("Dog is Barking");
    }
}

Class Lion extends animal
{
    Lion() // constructor
    { super(); // To access Non-abstract method.
    }
```

```
@Override.
Public void Sound ()// method for access Sound() abstract
                                                    method.
{
    System.out.Print ln("Lion is Roar");
}

}

Class Test
{
    Public Static void main(String[] args)
    {
        Dog d=new Dog();
        Lion l=new Lion();

        d.Sound();
        l.sound();
    }

}
```

Output:

```
All animals....!
All Animals....!
Dog is Barking
Lion is Roar.
```

Abstract Method

What is abstract method?

A method which contain abstract modifier at the time of declaration is called abstract method.

Syntax:

```
abstract class A
{
    abstract void m1(); //Abstract method
}
```

NOTE:

i) It can only be used in abstract class

ii) It doesn't contain any body → '{ }' and always ends with ";".

iii) Abstract method must be overriden in sub-classes otherwise it will also become a abstract class.

iv) Whenever the action is common but implementation are different then we should use abstract method.

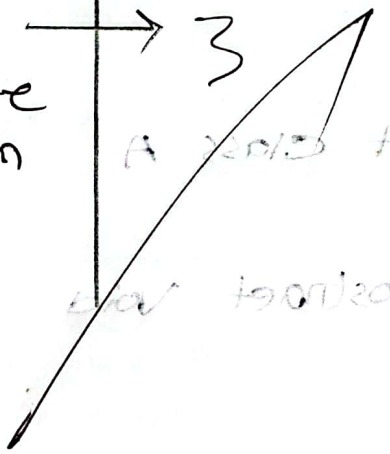
OR;

When we have incomplete information in the method then we turn ~~into~~ incomplete information method into abstract.

Example:

```
Class fruits
{
Public void taste ()
  {
    // Incomplete information
  }
}
```

```
abstract class fruits
{
Public abstract void taste();
//abstract method.
}
```



```
abstract class Programming // Abstract class
{
    Public abstract void Developer();
    Public abstract void Rank();
}

Class HTML extends Programming
{
    @Override
    Public void Developer()
    {
        System.out.Println("Tim Berners Lee");
    }

    @Override
    Public void Rank()
    {
        System.out.Println("3rd Rank");
    }
}

Class Java extends Programming
{
    @Override
    Public void Developer()
    {
        System.out.Println("James Gosling");
    }
```

Abstract Method

ELITE X2

```
@Override
Public void Rank()
{
System.out.Println(" 1st Rank ");
}

}

Class Main
{
   Public Static void main(String[] args)
   {

      HTML h=new HTML();
      Java J=new Java();

      h.Developer();
      h.Rank();

      J.Developer();
      J.Rank();

   }
```

CamScanner

Interface

What is °interface?

Interface is just like a class, which contains only abstract methods.

To achieve °interface Java provides a keyword called "°implements".

Syntax:

```
°interface Client
{
    Void m1();
}
```

NOTE:

i) Interface methods are by default Public and abstract.

ii) Interface variables are by default Public + static + final.

iii) Interface method must be overriden °inside the °implementing classes.

iv) Interface nothing but deals between client and developer.

Example - 01

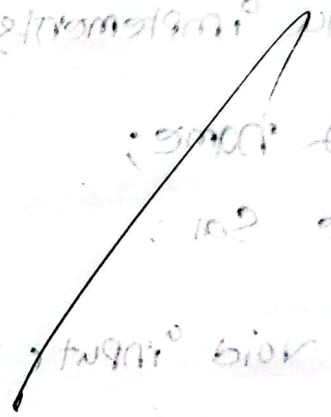
```
interface Animal
{
    Public void animalSound();
    Public void Sleep();
}

Class cat implements Animal
{
    @Override
    Public void animalsound()
    {
        System.out.Println("The cat says: meaww meaw");
    }

    @Override
    Public void Sleep()
    {
        System.out.Println("Zzzz");
    }
}
Class Test
{
    Public static void main(String[] args)
    {
        cat myCat = new cat();
        myCat.animal Sound();
        myCat.Sleep();
    }
```

Example - 02

```
import java.util.Scanner;

interface client
{
    void input(); // Public + abstract
    void output(); // Public + abstract
}

class Raju implements client
{
    String name;
    double sal;
    @Override
    Public void input()
    {
        Scanner n=new Scanner(System.in);
        System.out.Println("Enter Username = ");
        name = n.nextLine();

        System.out.Println("Enter Salary = ");
        sal = n.nextDouble();
    }
    @Override
    Public void output()
    {
        System.out.Println(name+ " " + sal);
    }
}
```

Interface method

```
class test
{
    Public static void main(String[] args)
    {
        Client c=new Raju();
        c.inPut();
        c.outPut();
    }
}
```



Why Interface variables are Static + Public + Final ?
Explain with CODE example.

```
interface customerSaim
{
    int amt = 5; // Public + Static + Final
    void Purchase(); // Public + abstract
}

class sellerZamil implements customerSaim
{
    @Override
    Public void Purchase()
    {
        // amt=7; is not possible because amt=5 is [final]
        // amt = 5 final
        System.out.Println("Saim needs" + amt + "kg rice");
    }
}

Class Test
{
    Public Static void main(String[] args)
    {
        customerSaim c=new sellerZamil();
        c.Purchase();
        System.out.Println(customerSaim.amt); // Interface variable are Static.
    }
}
```

Output:

Saim needs 5 kg Dice.

5

Here,

- Interface variable are Public because we can use this variable in many where / anywhere.
- Interface variable are final because once we initialize / declare the value of variable we ~~can't~~ in the interface block we can't ~~use~~ re-modify else where.
- Interface variable are Static because variable int amt=5; are not dependent on an object.

Interface Method

```
interface Client
{
    void webdesign();
    void webdevelope();
}
```

Interface method

```
class RajTech implements client
{
    @Override
    void webdesign()
    {
        System.out.println("Green, top menu, three dot button");
    }
}

class Test
{
    public static void main(String[] args)
    {
        RajTech r = new RajTech();
        r.webdesign();
    }
}
```

Output:

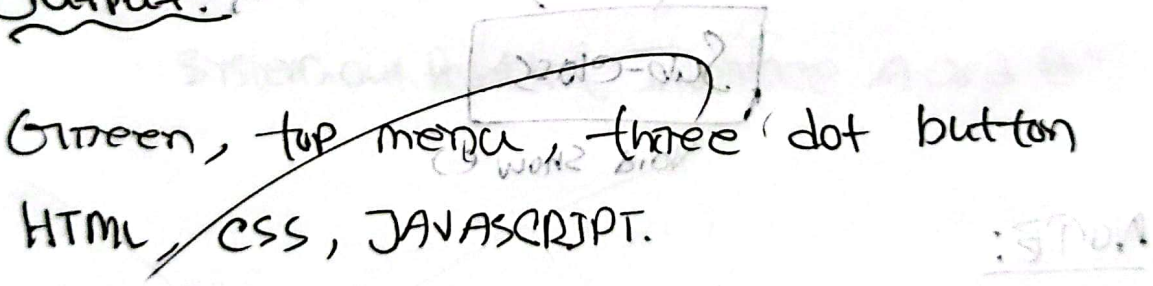
ERROR

Because, RajTech is not abstract and does not override abstract method webdevelope() in client class.

```
Interface Client
{
    void webdesign();
    void webdevelope();
}

Abstract class RajTech implements Client
{
    @Override
    Public void webdesign()
    {
        System.out.Println("Screen, top menue, three dot Button.");
    }
}

class RahulTech extends RajTech
{
    @Override
    Public void webdevelope()
    {
        System.out.Println("HTML, CSS, JAVASCRIPT");
    }
}
```

```
class Test
{
  Public Static void main(String[] args)
  {
    RahulTech T=new RahulTech();
    T.webdesign();
    T.Webdevelope();
  }
}
```

Output:



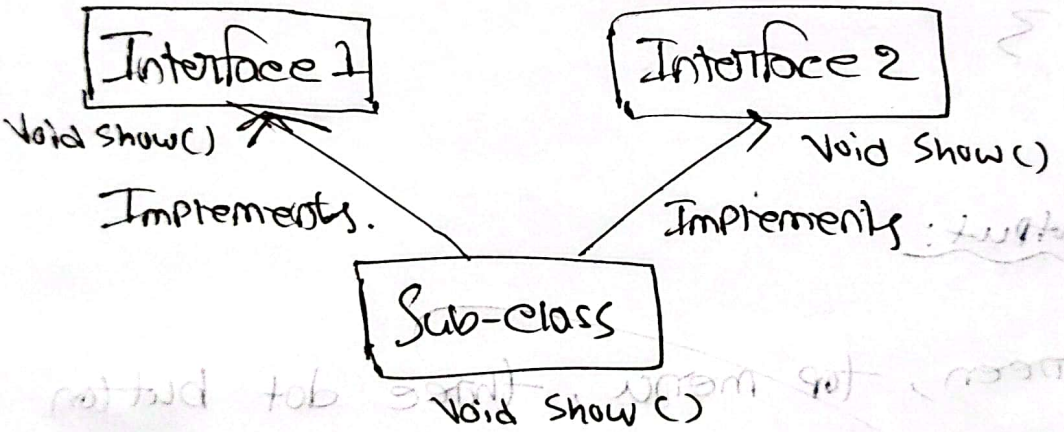
Green, top menu, three dot button

HTML, CSS, JAVASCRIPT.

Multiple inheritance using interface in JAVA.

#What is multiple inheritance using interface?

We can achieve multiple inheritance through interfaces because interface contains only abstract method, which implementation is provided by the Sub classes.

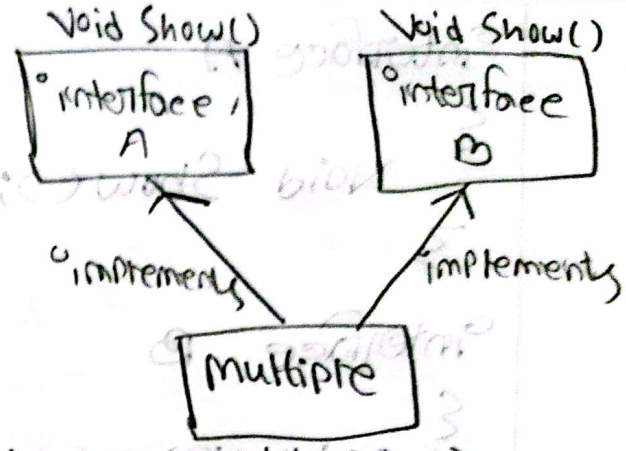


NOTE:

Class C extends A,B ⊗ (Wrong Syntax)

Class C implements A,B ✓ (correct Syntax)

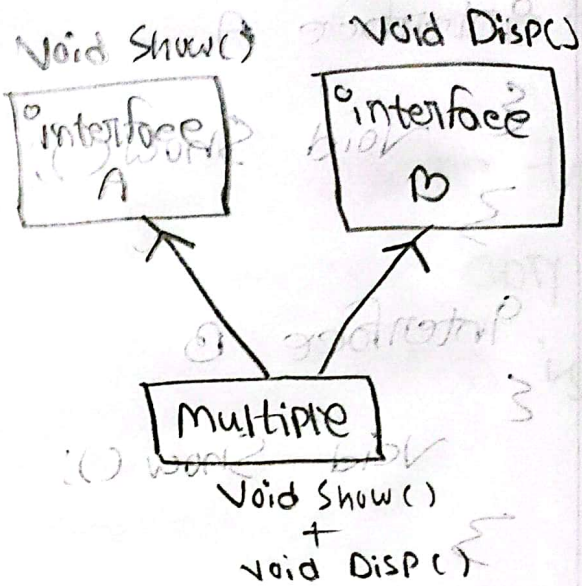
Example - 01




```
interface A
{  void show();
}
interface B
{
   void show();
}
Class Multiple implements A,B
{
   Public void show()
   {
      System.out.Println("Interface A and B");
   }
}
Class Test
{
   Public static void main(String[] args)
   {
      Multiple m=new Multiple();
      m.show();
   }
}
```

Example-02

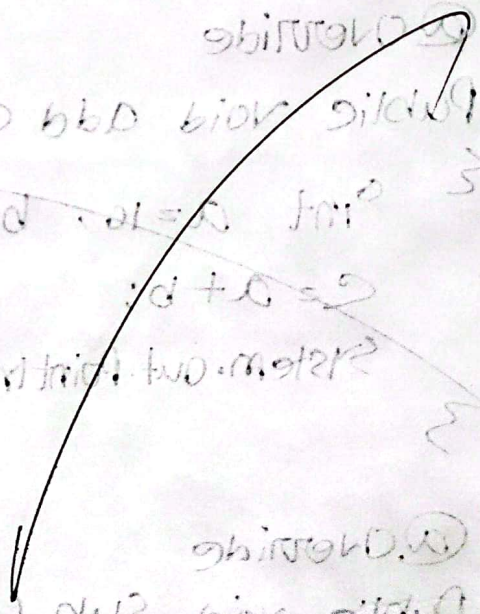
```
interface A
{
    void Show();
}

interface B
{
    void Disp();
}

Class Multiple implements A,B
{
    Public void Show()
    {
        System.out.Println("Interface A");
    }

    Public void Disp()
    {
        System.out.Println("Interface B");
    }
}

class Test
{
    Public static void main(String[] args)
    {
        Multiple m=new Multiple();
        m.Show();
        m.Disp();
    }
```



Extending Interfaces

```
interface Grill
{
    void add();
}
interface Raj extends Grill
{
    void sub();
}

class Ankit implements Raj
{
    @Override
    Public void add()
    {
        int a=10, b=20, c;
        c=a+b;
        System.out.Println("Addition "+c);
    }

    @Override
    Public void sub()
    {
        int a=20, b=10, c;
        c=a-b;
        System.out.Println("Subtraction "+c);
    }
}
```

```
Class Test
{
   Public Static void main(String2[] args)
   {
      Ankit Obj=new Ankit();

      obj.add();
      obj.sub();
   }
}
```



Output:

Addition 30

Subtraction 10

Interface JDK 1.8 version

Before JDK 1.8, Interface can only have abstract methods and all the abstract methods of interfaces must be overriden in implementing class as well as methods are public + abstract by default.

Example:

```
interface A
{
    void a1(); // public + abstract
    void a2(); // public + abstract
}
```

→ From JDK onwards - interface can have default and static methods

Example:

```
interface A
{
    void fun();
    void disp();
        +
    default void show(){}
    static void out(){}
}
```

Before 1.8 version

from 1.8 v.

Interface Default method (JDK 1.8)

```
interface A
{
    void a1();
    void a2();

    default void a3()
    {
        System.out.println("may or may not override in implementing classes");
    }
}

class B implements A:
{
    Public void a1()
    {
        System.out.println("class B a1()");
    }

    Public void a2()
    {
        System.out.println("class B a2()");
    }

    Public void a3()
    {
        System.out.println("Override in implementing class B");
    }
}
```

Interface Default method

```
class C implements A
{
    Public void a1()
    {
      System.out.Println("class C a1()");
    }

    Public void a2()
    {
      System.out.Println("class C a2()");
    }
}

class D implements A
{
    Public void a1()
    {
      System.out.Println("class D a1()");
    }

    Public void a2()
    {
      System.out.Println("class C a2()");
    }
}
```

```
class Test
{
    Public Static void main(String[] args)
    {
        B b=new B();
        b.a1(); b.a2(); b.a3(); // access def
                                   method in cl
        C c=new C();
        c.a1(); c.a2(); c.a3();

        D d=new D();
        d.a1(); d.a2(); d.a3();
    }
}
```

Output

class B a1()
class B a2()
Override in implementing class B
class C a1()
class C a2()
may or may not override in implementing classes
class D a1()
class D a2()
in implementation classes.

Static method in Interface

Rules:

1) Interface এর ভিতরে যে সকল Static method থাকে সেই সকল Static method কে কেহ Override করতে পারবে না।

2) Static method main class এর object এর উপর depend করে না। সুতরাং object create না করে সরাসরি Interface এ থাকা Static method কে main class এ access করতে পারবে।

SCENARIO - 1

```
interface A
{
    Public Static void Show()
    {
        System.out.Println("Can't Override interface
                            Static method");
    }
}
class Test
{
    Public Static void main(String[] args)
    {
        A.Show(); // No object create
    }
}
```

SCENARIO - 2

```
interface Test
{
  Public static void main (String[] args)
  {
    System.out.Println ("Learn coding");
  }
}
```

Exception Handling

What is Exception ?

An exception is un-expected / unwanted / abnormal situation that occured at runtime called exception.

Major reasons why an exception occurs:

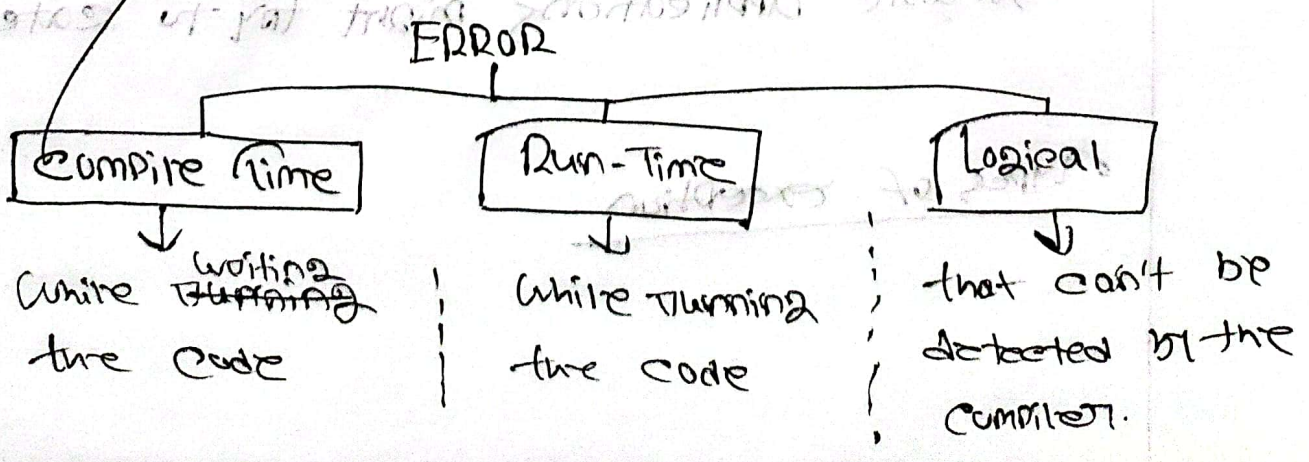
- Invalid user input.
- Device failure.
- Loss of network connection
- Code errors.

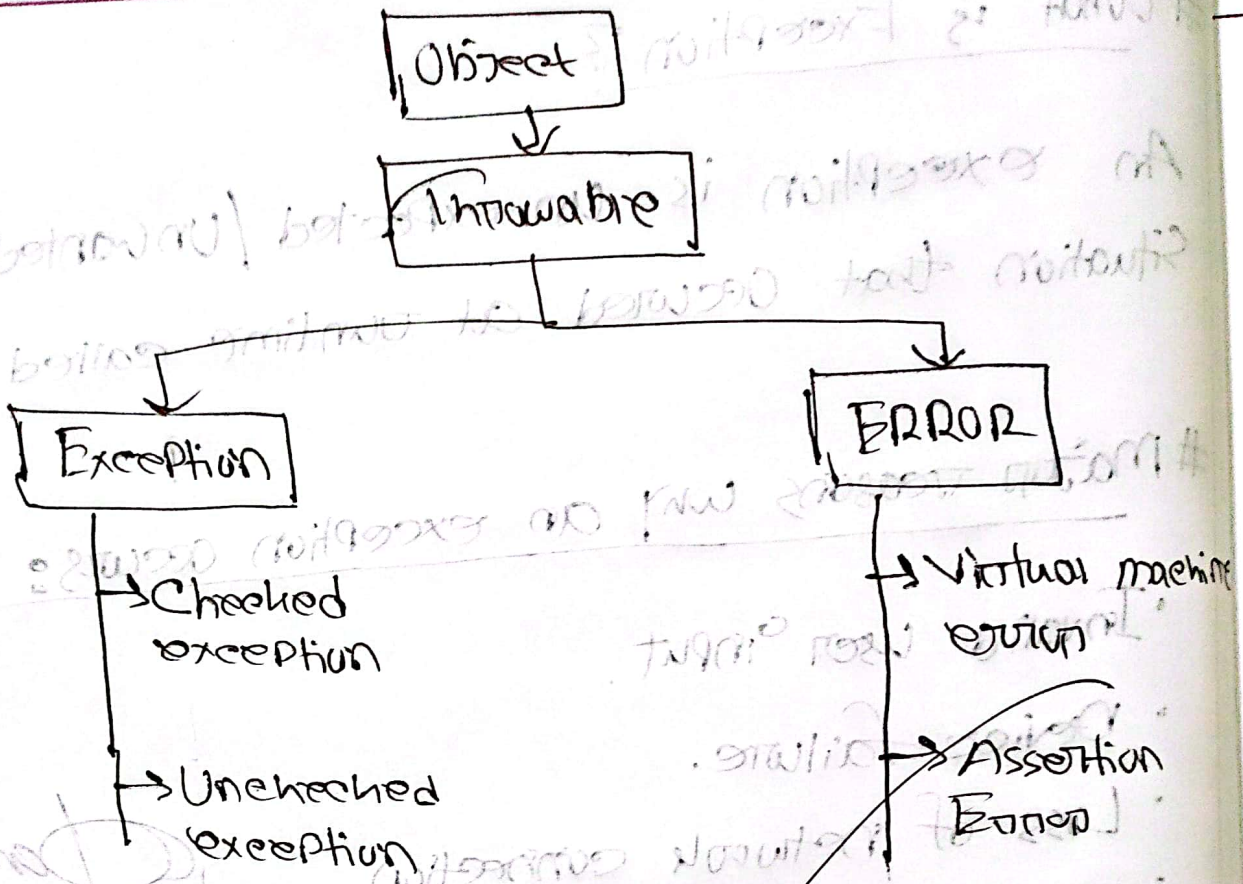
6.6.23

ERROR:

An error indicates a serious problem that a reasonable application should not try to catch

→ There are 3 Types of ERROR:

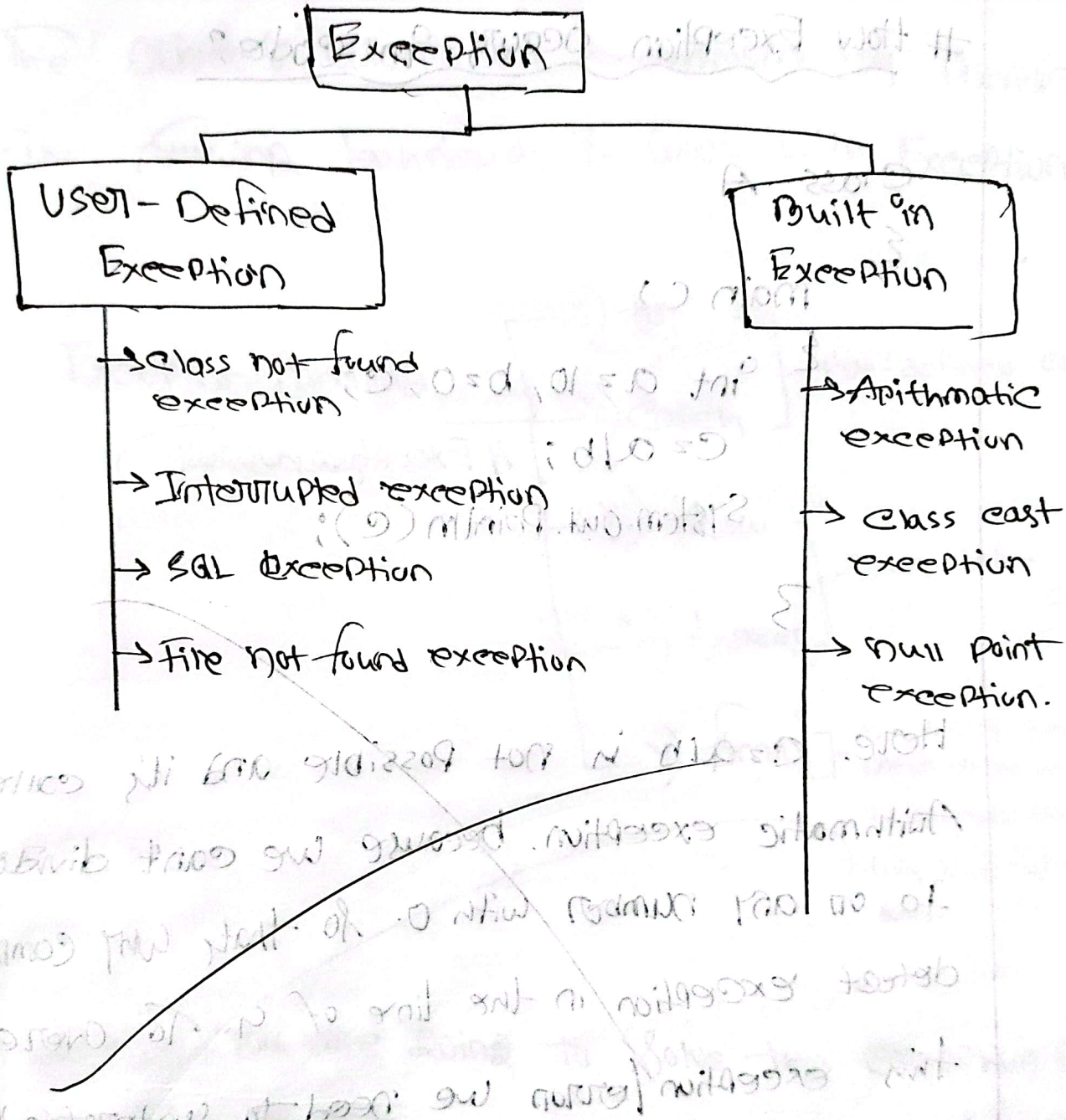




→ Exceptions:

Exception indicates conditions that a reasonable applications might try to catch

Types of exception


Exception
User - Defined Exception
Built in Exception
Class not found exception
Interrupted exception
SQL Exception
File not found exception
Arithmatic exception
Class cast exception
Null point exception.

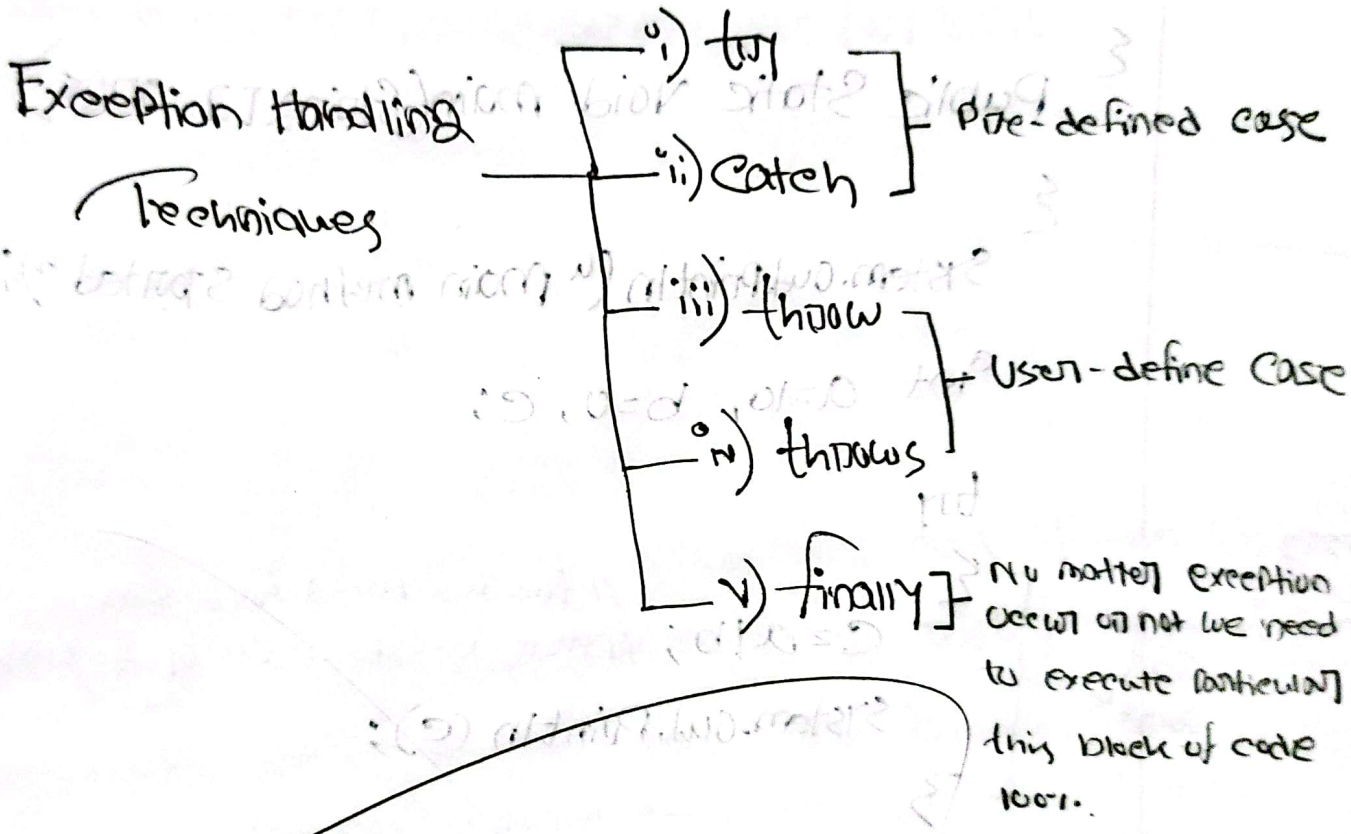
How Exception occurs in code?

```
class A
{
    main()
    { int a=10, b=0, c;
      c=a/b; // Exception occur
      System.out.Println(c);
    }
```

Here, c=a/b is not possible and it's called Arithmatic exception. because we can't divide to or any number with 0. So that why compiler detect exception in the line of 4. To overcome this exception/error we need to understand the concept of "Exception Handling".

Exception Handling Mechanism

The Object Orientation mechanism has provide the following techniques to work with Exception :-



Q How you are going to solve the exception occur Don't through Exception Handling of the following code:?

→ From the following code, it's a Re-defined case so we need to use try catch-block for exception

handling.. Here is the example given below-

```
Class Test
{
  Public Static void main(String[] args)
  {
    System.out.Println("main method started");
    int a=10, b=0, c;
    try
    {                    // this is a Pre-defined case.
      c=a/b; // Here exception occur. So that's why we insert c into try block.
      System.out.Println(c);
    }
    catch(Exception e)
    {
      System.out.Println(e); // If you want to know what is the name of Exception
    }
    System.out.Println("main method ended");
  }
```

Try Block

Super class - Exception (Throwable)

Referenced variable (anything you want)

catch block

Output:

Main method started

Java.lang.ArithmaticException: / by zero

Main method ended.

---

try and catch block

What is try block?

Whenever we write a statement and if the statement is error suspecting statement or risky code then put that code inside the try block.

```
try
{
}
```

What is catch block

The main purpose of catch block is to handle the exception which are throws by try{} block.

NOTE: Catch block will not be executed if there is no exception inside the try{} block.

Exception Hierarchy

What is exception Hierarchy?

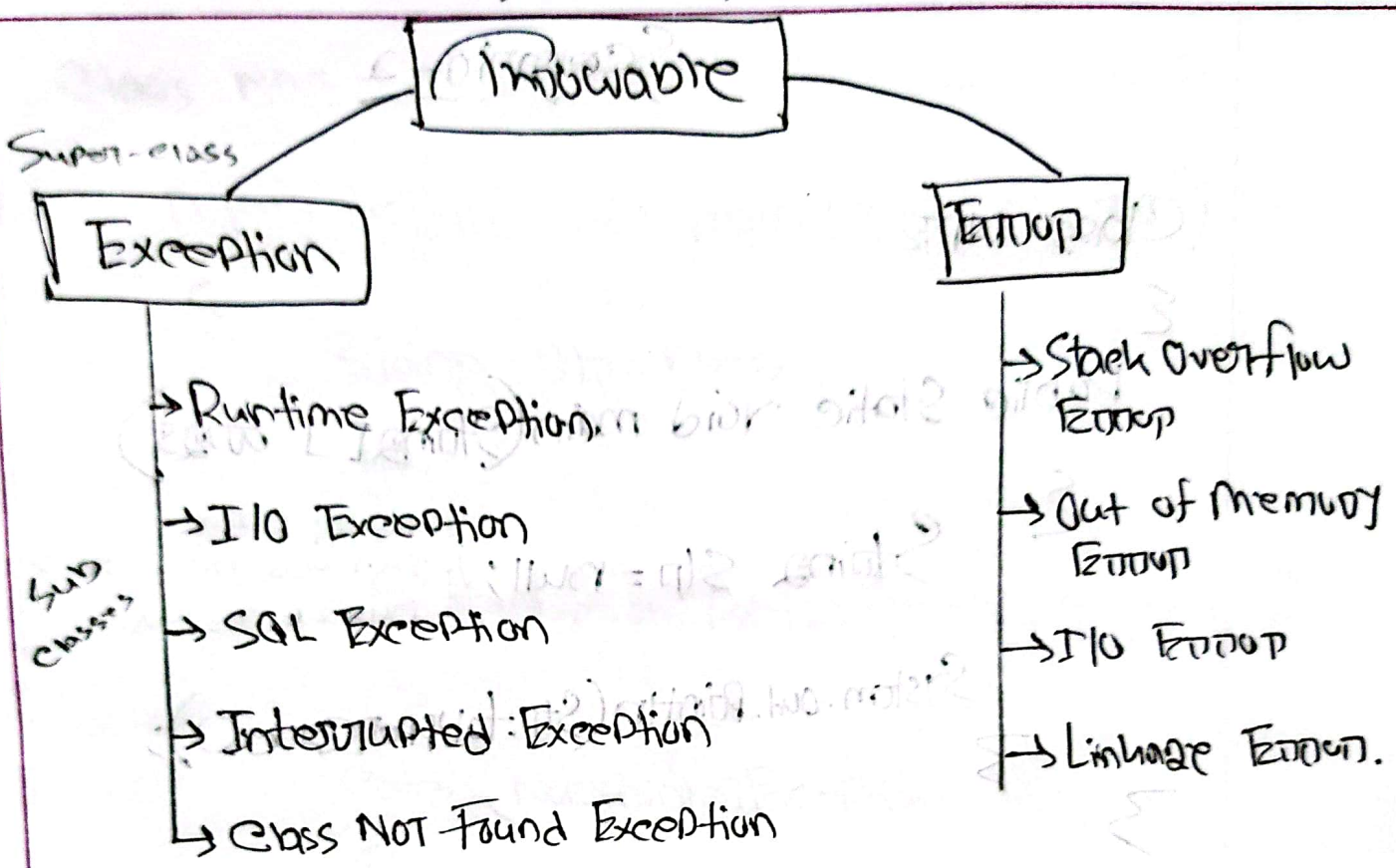
Throwable class is the super / root class of Java exception hierarchy. which, containing 2 sub classes that is -

i) Exception
ii) Error.

Difference between Exception and Error.

Exceptions	Error
i) Exceptions are recoverable	i) Error's are non-recoverable
ii) most of the cases exceptions raised due to our program code	ii) most of the cases, errors are due to lack of system resources but not due to our program
iii) For example :- ~~Memory~~ • Arithmatic Exception	iii) For example :- • memory error • JVM error • hardware error

Super-class/Root



⇒ Runtime Exceptions

- Arithmatic Exceptions (DONE)
- Null Pointer Exceptions (DONE)
- Number format Exceptions (DONE)
- Index Out of Bound Exceptions
  - Array out of Bound exception
  - String out of Bound exception.

Null Pointer Exception

Scenario-1

```
Class NPE
{
  Public Static void main(String[] args)
  {
    String str = null; // null is a reserve keyword in Pascal case
                       // null==0, no cast.
    System.out.println(str.toUpperCase());
  }
}
```

Output

ERROR: because,

Exception in thread "main" java.lang.NullPointerException: can not invoke "String.toUpperCase()" because "<local1>" is null.

Scenario-2 (correct form)

```
class NPE
{
    public static void main(String[] args)
    {
        String str = null;
        try
        {
            System.out.println(str.toUpperCase());
        }
        catch(NullPointerException n)
        {
            System.out.println("null can't be casted");
        }
    }
}
```

Output :-

null can't be casted.

Scenario - 02 (without null)

```
Class NPE
{
Public Static void main(String[] args)
  {
    String Str = "Saim"

    try
    {
      System.out.Println(Str.toUpperCase());
    }
    catch (NullPointerException e)
    {
      System.out.Println("null can't be casted");
    }
  }
}
```

Output:

SAIM

NumberFormatException

Scenario - 01

```
Class NFE
{
  Public Static Void main(String[] args)
  {
    String str="ankush";
    int a=Integer.ParseInt(str);
    System.out.Println(a);
    System.out.Println("String2 NumberFormatException");
  }
}
```

//Integer.ParseInt() is a method in JAVA that Parses the String argument as a Signed decimal Integer.

Output:

ERROR

Exception in thread "main" Java.lang.NumberFormat Exception: For input String "ankush".

Scenario-02

```
class NFE
{
  Public Static void main(String [] args)
  {
    String Str = "ankush";
    try
    {
      int a = Integer.ParseInt(Str)
      System.out.Println(a)
    }
    catch (NumberFormetException n)
    {
      System.out.Println("String "+str+" can't be
                          converted");
    }
  }
}
```

Output:

String ankush can't be converted.

Try Catch finally block in JAVA

```
class Test
{
   Public static void main(String[] args)
   {
      try
      {
         System.out.Println("Learn coding");
         int a=20, b=0, c;
          c= a/b;
          System.out.Println(c);
          System.out.Println("like share");
      }
    Catch(Arithmatic Exception a)
    {
       System.out.Println("can't devide by zero");
    }
  finally
    {
       System.out.Println("Subscribe");
    }
    System.out.Println("main method ended");
   }
}
```

Output:

Learn coding

Can't devide by zero

Subscribe

Main method ended.

Threads

Threads in Java provides a way to achieve multitasking and concurrent execution of task within a program.

(একটি program এর ভিতর বা একাধিক program এর ভিতর কাজ চালানো করতে দেয়।)

→ There are 2 ways to create a thread:-

1. By extending Thread class.
2. By implementing Runnable interface.

Thread class:

Thread class provide constructors and methods to create and perform operations on a thread.

Thread class extends Object class and implements Runnable interface.

6.6.23

Constructors of Thread class

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r, String name)

Methods of thread class:

* Public void run(): is used to perform operation for a thread.

* Public void start(): Starts the execution of the thread. JVM calls the run() method on the thread

* Public void sleep(): causes the currently executing thread to sleep for the specified number of milliseconds

* Public void Join(): waits for a thread to die.

* Public int getPriority(): Return the Priority of the thread.

* Public int setPriority(): Change the Priority of the thread.

* Public Thread currentThread: returns the reference of currently executing thread.

* Public boolean isAlive: tests if the thread is alive.

* Public void Suspend(): is used to Suspend the thread.

* Public void Stop(); is used to Stop the thread.

Runnable interface:

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

Public void run(): is used to perform action for a thread.

Starting a thread:

The start method of thread class is used to start a newly created thread. It performs the following tasks:-

⇒ A new thread starts (with new call stack)

⇒ The thread moves from new state to the runnable state.

⇒ When the thread gets a chance to execute its target run() method will run.

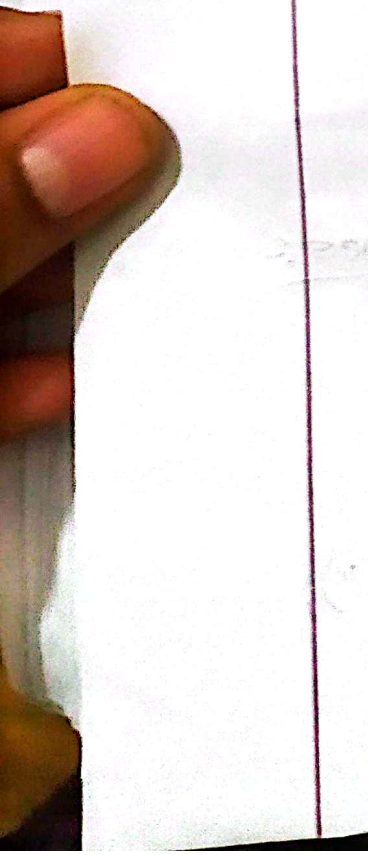
* Example by extending Thread class:

```
Class multi extends Thread
{
    Public void run()
    {
        System.out.Println("thread is running");
    }

Public Static void main (String [] args)
    {
        multi t1 = new multi();
        t1.start();
    }
}
```

* Example by implementing Runnable interface:

```
Class multi implements Runnable
{
    Public void run()
    {
        System.out.println("thread is running");
    }
```

```
Public Static Void main (String [] args)
{
    multi m1= new multi();
    Thread t1= new Thread(m1) //using constr
    t1.start();
}
}
```



* Using the Thread class: Thread(String name):

```
Public class MyThread1
{
    Public Static void main(String[] args)
    {
        //create object of the thread class using
        the constructor Thread(String name)
        Thread t=new Thread("My first thread");
        t.start();
        String str=t.getName();
        System.out.println(str)
    }
}
```

Output:

My first thread

* Thread (Runnable r, String name)

```
Public class MyThread2 implements Runnable
{
   Public void run ()
   {
      System.out.println("Now the thread is closed");
   }
   // main method
   Public static void main (String[] args)
   {
      // creating object
      Runnable r1= new MyThread2();
      Thread th1= new Thread(r1, "my new thread");
      th1.start();
      String str= th1.getName();
      System.out.println(str)
   }
}
```

Output: → my new thread
Now thread is closed.