MODULE – 3

CLASSES & OBJECTS

- Everything in Java is associated with classes and objects.
- It is a basic unit of Object Oriented Programming.

OBJECTS

- **Definitions:**
  - An object is *a real-world entity*.
  - An object is *a runtime entity*.
  - The object is *an entity which has state and behaviour*.
  - The object is *an instance of a class*.

- **Characteristic of objects :**
  - **State(knows) :** represents the data (value) of an object.
  - **Behavior(does) :** represents the behavior (functionality)of an object.
  - **Identity:** A unique ID to identify an object. The value of the ID is not visible to the external user but used internally by the JVM to identify each object uniquely.

CLASSES:

- **Definitions:**
  - It is a template or blueprint of a class which objects are created.
  - It is a logical entity and cannot be physical.
  - A class is declared by use of the class keyword.
  - A class in java can contain

Fields

Methods

Constructors

Blocks

Nested class and interface.

- **Syntax of class definition**

```
class classname
{
     type instance-variable1;
    type  instance-variable2;
    type instance-variableN;

    type methodname1(parameter-list)
    {
        // body of method
    }
    type methodname2(parameter-list)
    {
         // body of method
    } // ...

    type methodnameN(parameter-list)
    {
        // body of method
    }
}
```

- **Example :**

```
class Box
{
    double width;
```

```
            double height;
            double depth;
    }
    class BoxDemo
    {
            public static void main(String args[])
            {
                     Box mybox = new
                     Box(); double vol;

                    mybox.width = 10;     // assign values to mybox's instance variables
                    mybox.height = 20;
                    mybox.depth = 15;
                    vol = mybox.width * mybox.height * mybox.depth;// compute volume of
                    box
                    System.out.println("Volume is " + vol);
            }
    }
```

- **Declaring a class:**

  Box mybox;

  mybox is just an object reference to Box Class. It will not create Object.

- **Creating an Object:**

  Object is created by using a new keyword followed by constructor.

  Example : Box mybox = new Box();

  - ❖ mybox will be an instance of Box.
  - ❖ It will have “physical” reality.

- ❖ Each time creating an instance of a class, will create an object that contains its own copy of each instance variable defined by the class. Thus, every Box object will contain its own copies of the instance variables width, height, and depth.
- ❖ To access these variables, will use the dot (.) operator. The dot operator links the name of the object with the name of an instance variable.
- ❖ any number of objects can be created for a particular class.

CONSTRUCTORS

- **Definition**
  - A constructor is a block of code and called when instance of class is created.
  - During constructor calling, memory for the object is allocated in the heap memory.
  - During Object creation, at least one constructor is called.
  - It call default constructor if no constructor available in the class.
  - Java compiler provides default constructor by default.
  - Every class in java should contain either customized constructor and default constructor.
- **Rules for creating java constructor**
  - Constructor name should be same as class name.
  - Return type not applicable to constructor.
  - The only applicable modifier is default, public protected and private. Other modifier such as native, strictfp, final, static, synchronized etc are not valid for constructor.

Note:

Case 1: if the constructor is declared as public then object can created from anywhere. Case 2: if the constructor is declared as default then object can be created

only

within the current package.

Case 3: if the constructor is declared as protected then object can be created only within the current child class package.
Case 4: if the constructor is declared as private then object can be created only

within the class. It is mainly used to implement Singleton class. Singleton class is a class which are used to create only one object for a class.

- **Types of constructor**
  - **Customised constructor**

    It is called customised constructor because these constructor are designed by the programmer/developer.

    It can be zero parameter constructor and parameterized constructor.

    Example

```
class Student
{
        int id; String
        name;

        Student(int i,String n) //creating a parameterized constructor
        {
                id = i;
                name = n;
        }
        void display() //method to display the values
                {
        System.out.println(id+" "+name);
        }

        public static void main(String args[])
        {
                Student s1 = new Student(111,"Karan");
                Student s2 = new Student(222,"Aryan");

                s1.display();    //calling method to display the values of object
                s2.display();
         }
}
```

- **Default Constructor**

- ❖ It is a no argument/zero parameterised constructor.
- ❖ It is generated by compiler not by JVM.
- ❖ The access modifier of default constructor is same as class modifier which is applicable only if class modifier is public and default. Because top level class cannot be private and protected.
- ❖ It includes one line of code i.e super() which is a no argument call to super class constructor.

- **super() & this() constructor**

  It is mainly used to call constructor of another class.

  - **super()**

    It is used to call base class constructor(Parent class).

    Example:

```
class Parent
 {
  Parent()
   {
     System.out.println("Parent class's No " + " arg constructor");
   }
}

class Child extends Parent
 {
   Child()
   {
     super();
     System.out.println("Flow comes back from " + "Parent class no arg const");
   }
   public static void main(String[] args)
   {
     new Child();
```

```
        System.out.println("Inside Main");
    }
}
```

Output:

```
Parent class's No arg constructor
Flow comes back from Parent class no arg const
 Inside Main
```

Explanation

- ❖ In main, a statement **new Child()**, calls the no argument constructor of Child class.
- ❖ Inside that it has **super()** which calls the no argument of Parent class. Parent class constructor has SOP statement and hence it prints *Parent class's No arg constructor*.
- ❖ Now flow comes back to the no argument of the Child class and has SOP statement and hence it prints *Flow comes back from Parent class no arg const*.
- ❖ Flow now came back again to main and executes remaining statements and prints *Inside Main*.

- **this()**

It is used to call current class constructor (Parent class).

Example :

```
class RR
{
    RR()
    {
        this(10);
        System.out.println("Flow comes back from " + "RR class's 1 arg const");
```

```
}

RR(int a)
{
   System.out.println("RR class's 1 arg const");
}
public static void main(String[] args)
{
```

n
e
w
R
R
(
)
;
S
y
s
t
e
m
.
o
u
t
.
p
r
i
n
t

}

Output:

```
l                    e
n                    M
(                    a
"                    i
I                    n
n                    "
s                    )
i                    ;
d                   }
```

```
RR class's 1 arg const
Flow comes back from RR class's 1 arg
 const Inside Main
```

Explanations

- In main a statement **new Child()** calls the no argument constructor of Child class, inside that it calls **this(10)** which is 1 argument of current class(i.e, RR class).
- 1 argument constructor of RR class has SOP statement and hence it prints *RR class's 1 arg const*.
- Now flow comes back to the no argument of the RR class and it has SOP statement and hence it prints *Flow comes back from RR class's 1 arg const*.
- Flow now came back again to main and executes remaining statements and prints *Inside Main.*

✔ **Various cases of super() & this()**

- Case 1: call to super() and this() must be the first statement in constructor.

- Case 2 : call to super() & this() must be in first statement in constructor and cannot be used simultaneously.
- Case 3 :super() & this() should be used inside the constructor.

- **Super and this keywords**
   - If parent and current class contain instance variable in the same name then super and this keyword are used to access the super and current class instance variable respectively.
   - super is used to refer parent class instance variable.
   - this is used to refer current class instance variable.
   - Super and this can be used anywhere except in static area.
   - Example :

```
Class P

{
     String s = “parent variable”;
}
Class C extends P
{
     String s = “child variable”;
     public void m()
     {
          System.out.println(s);       //child variable
          System.out.println (this.s); //child variable
          System.out.println (super.s); //parent variable
     }
}
Class Main
{
     psvm(String[] args)
     {
             C c = new C();
             c.m1();
     }
}
```

- **Constructor Overloading**
   - It is a technique of having more than one constructor with different parameter lists.
   - They are differentiated by the compiler by the number of parameters in the list and their types.

- Example :

```
Class Box
{
     double width, height, depth;
     Box(double w, double h, double d)
     {
                                        width=w;height=h;depth=d;
     }
     Box()
     {

     }
                                        width = height = depth = 0;
     Box(double len)
     {
             width = height = depth = len;
     }
     double volume()
     {
             return width * height * depth;
     }
}
```

```
public class Test
{
    public static void main(String args[])

      Box mybox1 = new Box(10, 20, 15);
      Box mybox2 = new Box();
      Box mycube = new Box(7);

      double vol;

      // get volume of first box
      vol = mybox1.volume();
      System.out.println(" Volume of mybox1 is " + vol);

      // get volume of second
      box vol =
      mybox2.volume();
      System.out.println(" Volume of mybox2 is " + vol);

      // get volume of cube
      vol =
      mycube.volume();
     System.out.println(" Volume of mycube is " + vol);
```

```
        }
    }
```

Output:

```
Volume of mybox1 is 3000.0

Volume of mybox2 is 0.0

Volume of mycube is 343.0
```

Exception Handling

What is the difference between Error and Exception?

Error	**Exception**
Error is a problem at runtime, forwhich we are unable to providesolutions programmatically.	Exception is a problem, for which, weare able to provide solution programmatically.
Ex: JVM internal Problem StackOverFlowError InSufficientMainMemory	Ex:ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException

Definition

- *Dictionary meaning of the exception is abnormal termination.*
- **An exception is an event that occurs during execution of the program that disturbs normal flow of the program instructions.**
- If the application contains exception then the program terminated abnormally then the rest of the application is not executed. To overcome above limitation in order to execute the rest of the application & to get normal termination of the application must handle the exception.

Reasons for exceptions

- *opening a non-existing file.*
- *Network connection problems.*
- *Values are out of range values .*
- *End user input mistakes…….etc*

Exception Handling

- **The main objective of exception handling is to get normal termination of the application in order to execute rest of the application code.**
- Exception handling means just we are providing alternate code to continue the execution of remaining code & to get normal termination of the application.
- There are two ways to handle the exceptions in java.
  1) By using try-catch block.
  2) By using throws keyword.

Types of Exceptions

As per the sun micro systems standards The Exceptions are divided into three types

1) Checked Exception 2) Unchecked Exception 3) Error

Checked Exception

- The Exceptions which are checked by the compiler at the time of compilation are called Checked Exceptions.
- Examples:-
  IOException,SQLException,InterruptedException,ClassNotFoundException etc
- If the application contains checked Exception the compiler is able to check it and it will give intimation to developer regarding Exception in the form of compilation error.
- Handle the checked Exception in two ways
  - ✔ using try-catch block.
  - ✔ using throws keyword.

Checked Exception scenarios

✔ ***java.lang.InterruptedException***

- ❖ *Thread.sleep(2000); is executed, thread enters into sleeping mode then other threads are able to interrupt the program is terminated abnormally & rest of the application is not executed.*
- ❖ *To overcome above problem compile time compiler is checking that exception & displaying exception information in the form of compilation error.*
- ❖ *Based on compiler generated error message will handle the exception using the try-catch blocks or throws , if runtime any exception raised the try-catch or throws keyword executed program is terminated normally.*

✔ ***Java.io.FileNotFoundException***

- ❖ *Reading the file from local disk but at runtime if the file is not available program is terminated abnormally rest of the application is not executed.*
- ❖ *To overcome above problem compile time compiler is checking that exception & displaying exception information in the form of compilation error.*
- ❖ *Based on compiler generated error message will handle the exception using the try-catch blocks or throws , if runtime any exception raised the try-catch or throws keyword executed program is terminated normally.*

✔ ***Java.sql.SQLException***

- ❖ *Connecting to data base but at runtime data base is not available program is terminated abnormally rest of the application is not executed. Note: In above scenarios compile time compiler is display just exception information but exception raised at runtime.*

Unchecked Exception

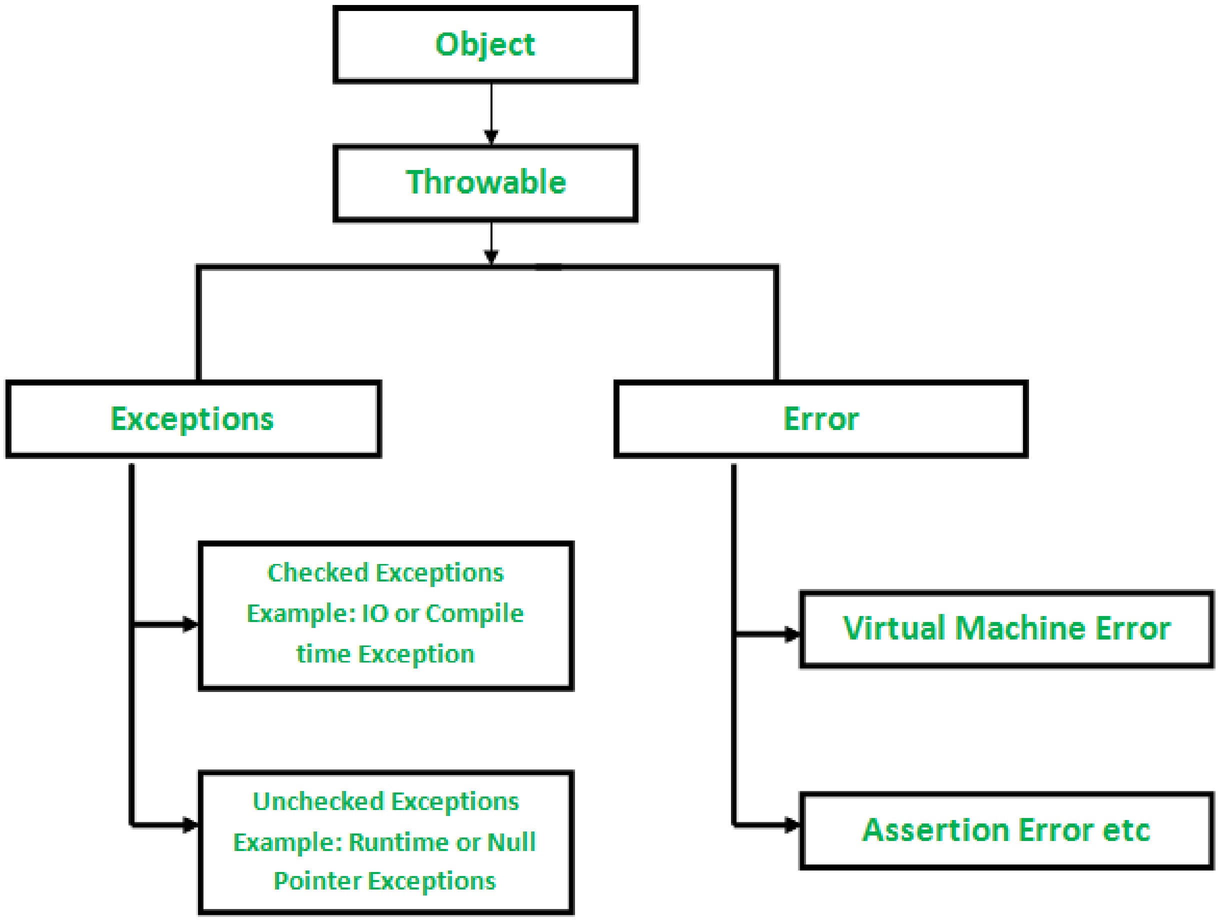
- The exceptions which are not checked by the compiler at the time of compilation are called unchecked Exception .
- Example:
  - ArithmeticException,
  - ArrayIndexOutOfBoundsException,
  - NumberFormatException….etc
- If the application contains un-checked Exception code is compiled but at runtime JVM (Default Exception handler) display exception message then program terminated abnormally.
- To overcome runtime problem must handle the exception in two ways.
  - ❖ using try-catch blocks.
  - ❖ using throws keyword.

```
class Test
{
        public static void main(String[] args)
        {
                int[] a ={10,20,30};
                System.out.println(a[6]);
         }
}
```

```
D:\>java Test Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 6
```

Note-1:- *Whether it is a checked Exception or unchecked exception exceptions are raised at runtime but not compile time.*
Note 2:- *In java whether it is a checked Exception or unchecked Exception must handle the Exception by using try-catch blocks or throws keyword to get normal termination of application & to execute rest of the application.*

Exception Handling Tree Structure


Object
Throwable
Exceptions
Error
Checked Exceptions Example: IO or Compile time Exception
Unchecked Exceptions Example: Runtime or Null Pointer Exceptions
Virtual Machine Error
Assertion Error etc

Exception handling key words

1) try
2) catch
3) finally
4) throw
5) throws

There are two ways to handle the exceptions in java.

1) By using try-catch block.
2) By using throws keyword.

Exception handling by using Try –catch blocks

Syntax:-

```
try
{
        exceptional code;
 }
 catch (ExceptionName reference_variable)
{
```

```
                Code to run if an exception is raised (alternate code);
        }
```

Example-1 :- Application without try-catch blocks

```
class Test
{
        public static void main(String[] args)
        {
                System.out.println("start");
                System.out.println(10/0);
                System.out.println("rest of the application");
        }
}
```

```
E:\>java Test
    start
    Exception in Thread "main" java.lang.ArithmeticException: / by zero
    Handled by JVM            type of the Exception
    description
```

In above example exception raised program is terminated abnormally & rest of the application is not executed

Whenever the exception raised the default exception handler is responsible to handle the exception & it is component of the JVM.

Application with try-catch blocks

Whenever the exception is raised in the try block JVM won't terminate the program immediately it will search corresponding catch block.

a. If the catch block is matched then that block will be executed & rest of the application executed & program is terminated normally.

b. If the catch block is not matched program is terminated abnormally.

```
class Test
{
        public static void main(String[] args)
        {
                System.out.println("start");
                try
                {
                        System.out.println(10/0);
                }
                catch (ArithmeticException ae)
```

```
{
        System.out.println(“cannot divide by zero”);
```

```
                    }
                    System.out.println("rest of the application");
            }
        }
E:\>java Test
 start
cannot divide by zero
rest of the application
```

In above example we are handling exception by using try-catch block hence the program is terminated normally & rest of the application is executed.

Various cases in Exception Handling

Case 1:

Example 1

If there is no exception in try block the corresponding catch blocks are not checked.

```
class Test
{
        public static void main(String[] args)
        {
                try
                {
                                        Sy
                                        st
                                        e
                }                       m.
                                        ou
                                        t.p
                                        ri
                                        ntl
                                        n(
                                        "s
                                        tar
                                        t")
;                                                    ri
Sy                                                   ntl
st                                                   n(
e                                                    10
m.                                                   /5
ou                                                   );
t.p
                    catch(ArthimeticException e)
                    {
                          System.out.println("arthimetic Exception");
                    }
                     System.out.println("rest of the app");
              }
       }
      Output:
```

```
s
t
a
r
t
5
rest of the app
```

Case 2

Example 2

In Exception handling independent try blocks declaration are not allowed must declare try-catch or try- finally or try-catch-finally.

```
class Test
{
public static void main(String[] args)
{
        try
        {
                System.o
                ut.println
                ("start");
                System.o
                ut.println
                ("hello
                world");
        }
        System.out.println("rest of the app");
}
}
```

Output:

```
javac Test.java
Test.java:: 'try' without 'catch' or 'finally'
```

Case 3

- ✔ In between try-catch blocks it is not possible to declare any statements, if we are declaring statements compiler will generate error message.
- ✔ In exception handling must declare try with immediate catch block.

- ✔ Example 3

```
class Test
{
        public static void main(String[] args)
        {
                try
                {                                   System.out.println(10/0);

                }

                 System.out.println("Hello
                world");
                catch(ArithmeticException e)
```

```
                    {
                              System.out.println(10/2);
                     }
                     System.out.println("rest of the app");
            }
    }
```

Case 4

If the exception raised in try block jvm will search corresponding catch block but if the exception raised other than try-catch blocks it is always abnormal termination.

In below example exception raised in catch block hence program is terminated abnormally.

Example 4

```
class Test
 {
        public static void main(String[] args)
         {
        try
         {
                 System.out.println("start");
                 }
                 catch(ArithmeticException e)
                {
                         System.out.println(10/0);
                 }
                 System.out.println("rest of the app");
        }
 }
```

Case 5

If the exception raised in try block the remaining code of try block is not executed.

Once the control is out of the try block the control never entered into try block once again.

Don’t take normal code inside try block because no guarantee all statements in try-block

will be executed or not.

Example 5

```
class Test
 {
        public static void main(String[] args)
         {
                 try
                 {
                                                System.out.println(10/0);
                                                System.out.println("srtart");
                                                System.out.println("stop");
                 }
                catch(ArithmeticException e)
                {
                          System.out.println("Arthimetic Exception");
                 }
                 System.out.println("rest of the app");
        }
}
Output :java Test
Arthimetic Exception
start
 rest of the app
```

Case 6 : Multiple catch block

The way of handling the exception is varied from exception to the exception hence it is recommended to provide try with multiple number of catch blocks.

Example 1

```
public class MultipleCatchBlock1
{

        public static void main(String[] args)
        {
                try
                {                                    int a[]=new int[5];a[5]=30/0;


                }
                catch(ArithmeticException e)
                {
                        System.out.println("Arithmetic Exception occurs");
                }
                catch(ArrayIndexOutOfBoundsException e)
                {
                    System.out.println("ArrayIndexOutOfBounds Exception occurs");
                }
                catch(Exception e)
                {
```

```
                System.out.println("Parent Exception occurs");
        }
                                        System.out.println("rest of the code");
            }
      }
```

Output:

```
Arithmetic Exception
occurs rest of the code
```

Example 2

```
public class MultipleCatchBlock2
{
        public static void main(String[] args)
      {
            try
            {                                   int a[]=new int[5];
                                                System.out.println
                                                (a[10]);
            }

            catch(ArithmeticException e)
            {
                    System.out.println("Arithmetic Exception occurs");
            }
            catch(ArrayIndexOutOfBoundsException e)
            {
                  System.out.println("ArrayIndexOutOfBounds Exception occurs");
             }
            catch(Exception e)
            {
                    System.out.println("Parent Exception occurs");
            }
            System.out.println("rest of the code");
      }
}
```

Case 7

When we declare multiple catch blocks then the catch block order must be child-parent but if we are declaring parent to child compiler will generate error message.

No compilation error (catch block order child to parent type)

Example 7

```
public class MultipleCatchBlock4
```

```
{
        public static void main(String[] args)
        {
                try
                {                                                        String s=null;
                                                                         System.out.println
                                                                         (s.length());
                }

                catch(ArithmeticException e)
                {
                        System.out.println("Arithmetic Exception occurs");
                }
                catch(ArrayIndexOutOfBoundsException e)
                {
                      System.out.println("ArrayIndexOutOfBounds Exception occurs");
                }
                catch(Exception e)
                {
                        System.out.println("Parent Exception occurs");
                }
                System.out.println("rest of the code");
        }
}
```

Case 8

- ❖ There are three methods to print Exception information
  - ● toString() : used to display name and description of exception.
  - ● getMessage() : used to display description of the exception.
  - ● printStackTrace() : used to display name, description and stack trace(location). In other words, it is used to display complete information of exception.
- ❖ Example 8

```
class Test
{
        void m1()
        {
                m2();
```

```
}
void m2()
{
```

```
                m3();
            }
            void m3()
            {
                    try
                    {                                   System.out.println(10/0);

                    }

                    catch(ArithmeticException ae)
                    {
                            System.out.println(ae.toString());
                            System.out.println(ae.getMessage());
                            ae.printStackTrace();
                    }
            }
            public static void main(String[] args)
            {
                    Test1 t = new Test1();
                    t.m1();
            }
    };
    java.lang.ArithmeticException: / by zero        //toString() method output
    / by zero                                       //getMessage() method
     output java.lang.ArithmeticException: / by zero    //printStackTrace()
     method
```

Note : internally JVM used printStackTrace() to print exception information.

Finally Block

- It is used to execute important code such as closing connection, stream etc.

- Java finally block is always executed whether exception is handled or not.

- Java finally block follows try or catch block.

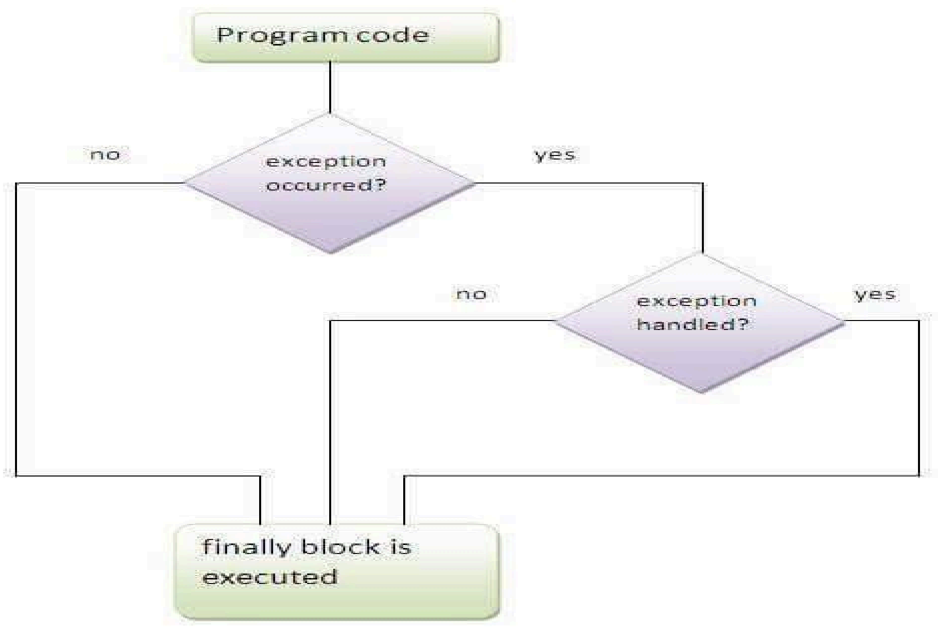
- **Syntax :**

```
try
 {
          critical code;
  }
catch (Exception obj)
 {
                    code to be run if the exception raised (handling code);
}
 finally
{
                    Clean-up code;(database connection closing , streams closing……etc)
}
```

- Example :

```
class TestFinallyBlock1
{
        public static void main(String args[])
       {
              try
              {
                     int data=25/0;
                     System.out.println(data);
              }
              catch(NullPointerException e)
              {
                     System.out.println(e);
              }
              finally
              {
                     System.out.println("finally block is always executed");
              }

              System.out.println("rest of the code...");
        }
```


Program code
no
exception occurred?
yes
no
exception handled?
yes
finally block is executed

}