20-3-2020

20 Friday — March — 12th week 079-286

Exception Handling →

An exception is a problem that arises during the execution of a program.

eg — User can enter any invalid data during run-time.

— File Not found during run-time.

Exception handling is a task to maintain normal flow of the program. It does not mean repairing an exception, but it rather defines alternative way to continue rest of the program normally.

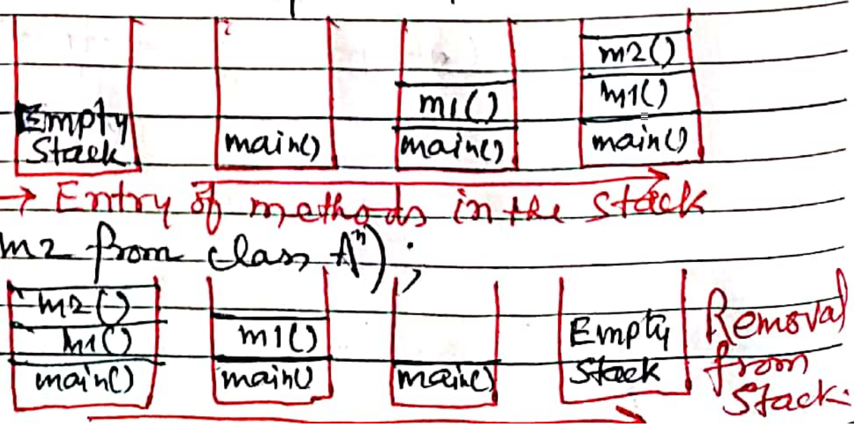
How Exception is Handled in Java.

Runtime Stack Mechanism is maintained by java virtual machine for every running thread or process of a program which is running.

```
class A
{
public static void main(String args[])
 {
   m1();
 }
static void m1()
 {
   m2();
 }
static void m2(){
 System.out.println("m2 from class A");
 }
}
```

Construction of Run-time Stack

1. main() is first called by the main thread.
2. After that main() is calling m1().
3. In the m1() m2() is called.
4. Finally "m2 from class A" printed.



Distruction - LIFO

Example:- exceptest.java

```
class exceptest
{
public static void main(String args[])
  {
    int i=10;
    i=i/0;
    System.out.println("Value of i is:"+i);
  }
}
```

If you compile and run it then you will see the ArithmeticException.

Modify this example as:-

```
class exceptest
{
public static void main(String args[])
  {
    int i=10;
    i=i/Integer.parseInt(args[0]);
    System.out.println("Value of i is:"+i);
  }
}
```

22 Sunday

If you compile and run it as follows then we see the ArithmeticException

ArrayIndexOutOfBoundsException

```
javac exceptest.java
java exceptest 2
```

output:
Value of i is:5

Notes

① Use of try, catch blocks →

13th week

23 Monday March

We have to put risky code within the try block and corresponding handling code inside the catch block such as :—

```
class Excep1
{
    public static void main(String args[])
    {
        try{ int i = 100/0; //protected code
        }
        catch(ArithmeticException e)
        {
System.out.println(e);
        }
System.out.println("After try-catch block");
    }
}
```

Note:- Within the try block if anywhere an exception occurs, then rest of the try block would not be executed even though we handled that exception.

Methods to Print Exception information

- printStackTrace()
- toString()
- getMessage()

e.printStackTrace();; — Prints Name of Exception: description: Stack Trace.

e.toString() :- Prints Name : description

e.getMessage() :- Prints only description

Default Exception Handler uses - printStackTrace().

Uses of these above three methods :-

```
try { int i = 100/0;
    }
catch (ArithmeticException e)
  { System.out.println(e.toString());
    }
```

or

```
try{  int i=100/0;
    }
catch(ArithmeticException e)
  { e.printStackTrace();
    }
```

or

```
try { int i=100/0;
    }
catch(ArithmeticException e)
  {
System.out.println(e.getMessage());
    }
```

Notes

Now, we can give more than one catch blocks with one try block.

13th week
084-281

25 March
Wednesday

But the order of catch block is more important. The base class exception object handles the exception object in last. The child classes of exception object is used to handle the exception object firstly.

If super class of exception class handles first then the next catch block will be unreachable by the compiler.

Example :-

```
class exception1
{
public static void main(String args[])
{
  int i=10;
    try{ i=i/Integer.parseInt(args[0]);
       }
      catch(ArithmeticException e)
       { System.out.println(e.toString());
       }
      catch(Exception e1)
       {
    System.out.println(e1.toString());
       }
  System.out.println("value of i is:"+i);
}
}
```

Notes

Hence, the order of catch block's argument will be from child to ~~parent~~ class order.

MARCH

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

APRIL

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

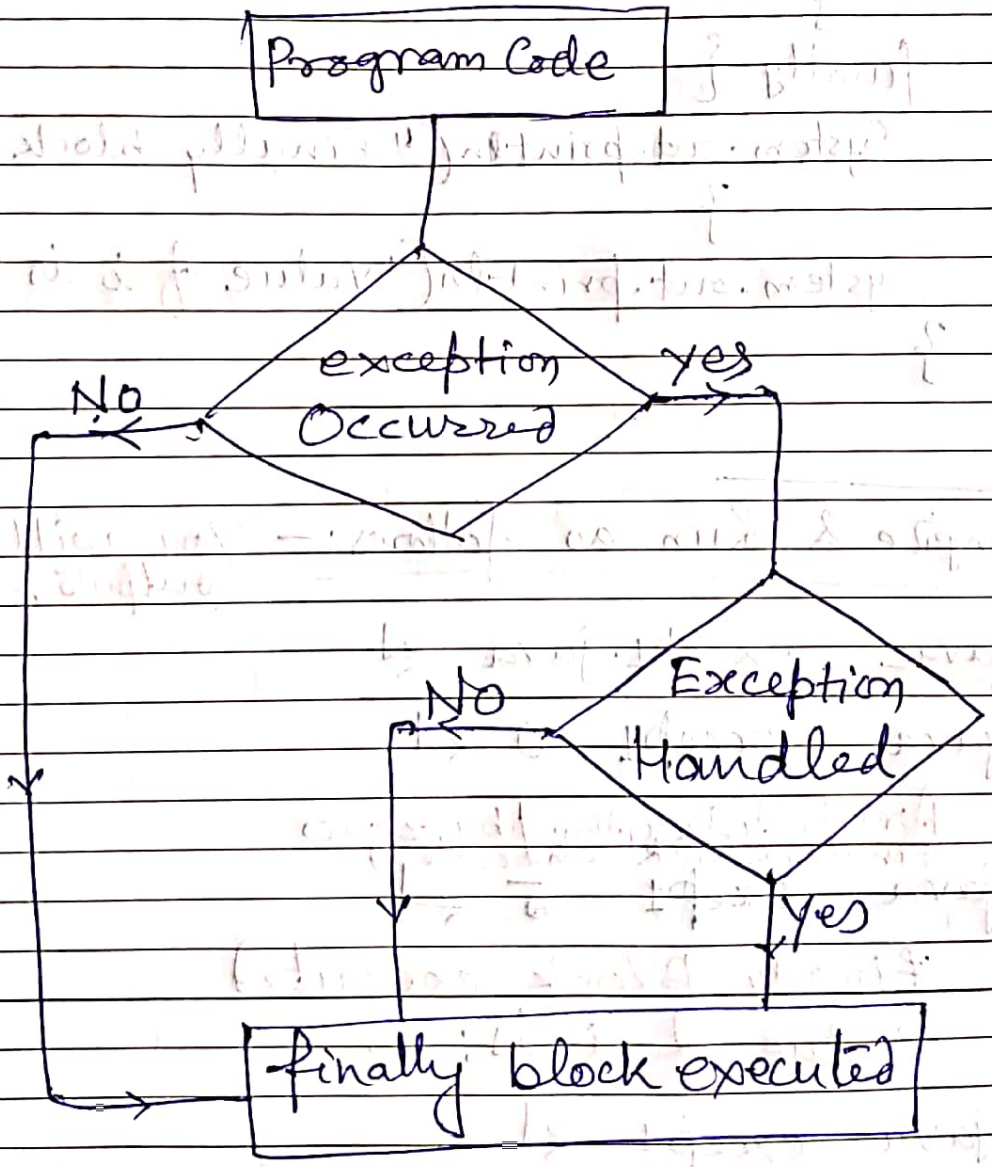
14th week

26
Thursday

finally block: — code in finally block always executes, whether or not exception has occurred.

finally block allows us to run any cleanup-type statements that we want to execute.

On flow-chart we can see this as:—



Notes

27 March
Friday

Example :- with finally block.

```
class except1
{
public static void main(String args[])
{
    int i=10;
    try{ i= i / Integer.parseInt(args[0]);
    }
    catch(ArithmeticException e)
    { System.out.println(e.toString());
    }
    finally {
    System.out.println("Finally block executed");
    }
System.out.println("Value of i is :"+i);
}
}
```

Compile & Run as follows :- You will see different outputs :-

```
javac except1.java ↵
java except1 0 ↵
```

ArithmeticException: / by zero
Finally Block executed

```
java except1 5 ↵
```

Finally Block executed
Value of i is : 2

```
java except1 ↵
```

ArrayIndexOutOfBoundsException:

Notes