Exception Handling :-

(i) What is exception ?

Ans → An exception is an unwanted or unexpected event, which occur during the execution of a program. i.e at run time, that disturb the normal flow of the program.

* -: What is Exception Handling :-

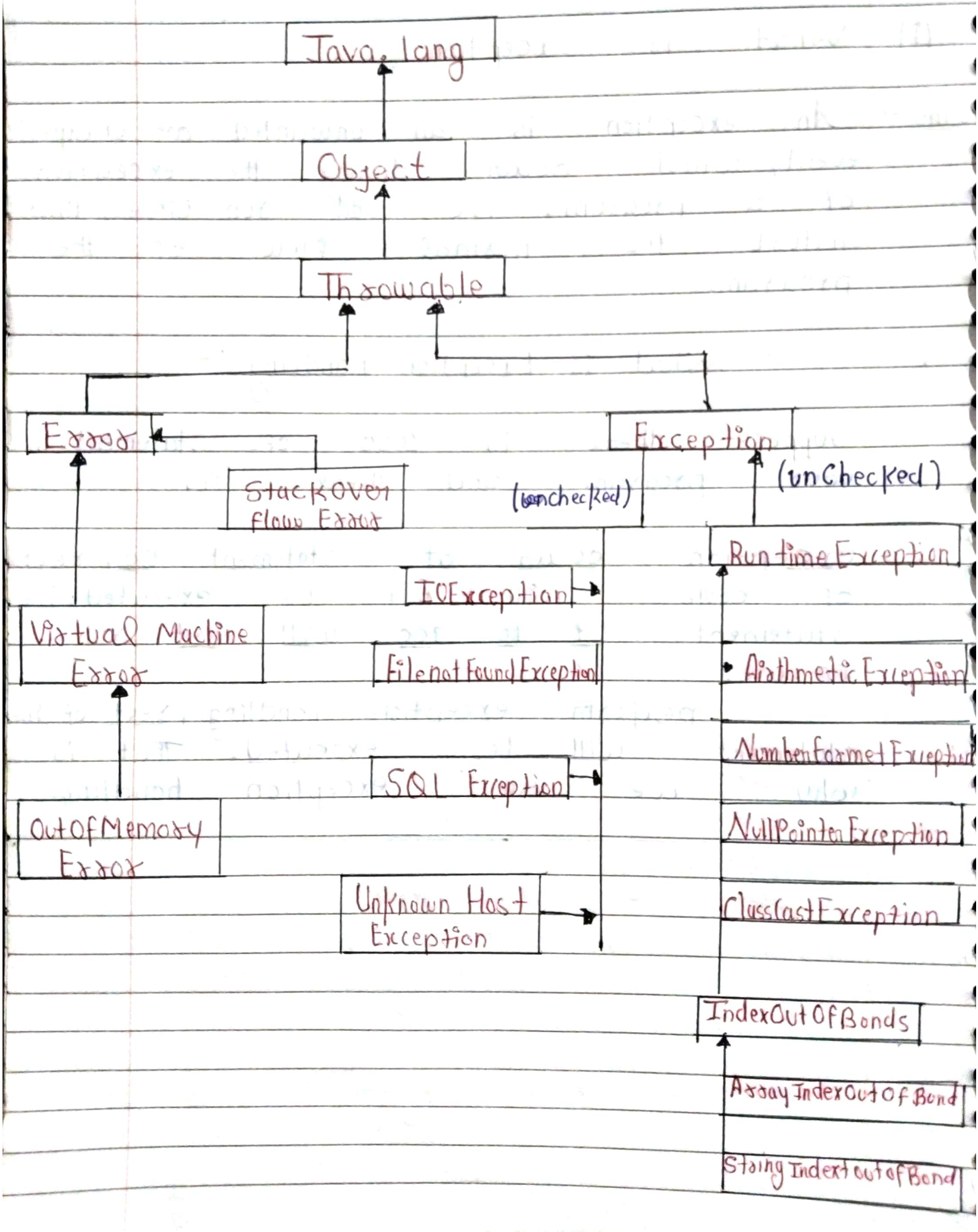
Suppose there is 1000 of statement in your program and there a,

exception occur at statement 50, rest of code will not be executed. i.e statement 51 to 100 will not run If we perform exception handling, rest of the statement will be executed. That is why we use exception handling.

31, Oct, 2022

```
class Aman
{
psvm (String ar[])
{
S.o.p ("Bhanu 1");

int x = Integer.parseInt(ar[0]);

S.o.p ("Bhanu 2");

int y = Integer.parseInt(ar[1]);

S.o.p ("Bhanu 3");
S.o.p (x/y);
S.o.p ("Bhanu 4");
}
}
```

Input	Output	Input	Output
10 2	Bhanu 1 Bhanu 2 Bhanu 3 5 Bhanu 4	10 abhi	Bhanu 1 Bhanu 2 NumberformetException
		10	Bhanu 1 Bhanu 2 Array Index Out of Bonds Exception -
0 2	Bhanu 1 Bhanu 2 Bhanu 3 Aisthmetic exception /0	abhi 0	Bhanu 1 NumberformetException


Java.lang
Object
Throwable
Error
Exception
StackOverFlow Error
(unchecked)
(unChecked)
Runtime Exception
IOException
Virtual Machine Error
FileNotFoundException
ArithmeticException
NumberFormetException
SQL Exception
OutOfMemory Error
NullPointerException
UnknownHost Exception
ClassCastException
IndexOutOfBonds
ArrayIndexOutOfBond
StringIndexOutOfBond

-: Types Of Exception :-

There are two types of exception :-

(i) Checked Exception

(ii) Unchecked Exception

The main difference between checked & unchecked exception is that unchecked exception are checked at runtime.

Runtime exception and its subclass are unchecked exceptions.

* What is the super class for all exception and error ?

Ans* The super or base class for all exception and error is Throwable

* What is the super class for all exceptions ?

Ans* Exception class.

Exception Handling Keywords.

(i) Try
(ii) Catch
(iii) finally
(iv) Throw
(v) Throws

-: Try Block :-

```
Class A
{
psvm (String ar[])
{
  S.o.p ("Softwaves_1");
int  x =  Integer.parseInt (ar[0]);
int  y =  Integer.parseInt (ar[1]);

try
{
   S.o.p (x/y);
}
  S.o.p ("Softwaves_4");
}
```

Error :- try without 'catch', finally or resource declaration.

Programmer when observe that there is a chance of coming exception in the code then that code is written insid the try Block.

We can not use try block Single we must use try block with catch, finally or with resource declaration.

otherwise error will occur.

1, Nov, 2022

```
class Neha
{
psvm (String ar[])
{
 S.o.p ("Softwaves_1");
int x = Integer.parseInt (ar[0]);
int y = Integer.parseInt (ar[1]);

try
{
 S.o.p (x/y);
}
 catch (ArithmeticException e)
{
 S.o.p ("Arithmetic");
}
 S.o.p ("Softwaves_4");
}
}
```

Input	output	Input	Output
10 2	Softwaves_1 5 Softwaves_4	10 0	Softwaves_1 Arithmetic... Softwaves_4
10 abc	Softwaves_1 NumberformetException	10	Softwaves_1 ArrayIndexOutOfBonds.

-: Catch Block :-

The catch block will always came after the try block no statement or block can came between try and catch block.

```
try
{
    S.o.p ("S.S.S");
}
 catch (Exception_Name e1)
{
 }
```

when exception is generated in try block then catch block is executed. we can use multiple catch block with one try block.

The type of exception is generated in try block, that type of catch block is executed.

```
Class Aman
{
 psvm (String ar[])
{
  S.o.p ("Softwaves_1");
```

```
try
{
 int   x =   Integer.parseInt (ar[0]);
int    y =   Integer.parseInt (ar[1]);

 S.O.p   (x/y);
}
catch  (Arithmetic_Exception e)
{
   S.O.p ("Arithmetic.....");
}
catch (NumberformetException e)
{
   S.O.p ("Number");
}
~~catch~~ {   S.O.p ("Softwaves 4");
}
}
```

Input	Output	Input	Output
10 5	Softwaves 1 5 Softwaves 4	10 0	Softwaves 1 Arithmetic Softwaves 4
10 tanu	Softwaves 1 Number Softwaves 4	ram	Softwaves 1 Number Softwaves 4
10	Softwaves 1 Array Index Out of Bond		

At a time only one exception is generated because the program is terminated from the point and if its corresponding catch block is available then it is executed.

Anonymous Array

```
int x [] = new int [5] {10,20,30,40,50};
for (int i : x)
{
  S.o.p (i);
}
```

new int [] {10,20,30,40,50}; → Run

error :- } expected

```
class A
{
void show (int x [])
{
 for (int i : x)
{
   S.o.p (i);
}
}
Class Demo
{
 psvm (String ar [])
{
 A a = new A();
 a. show (new int [] {10,20,30,40,50});
}
}
```

10
20
30
40
50

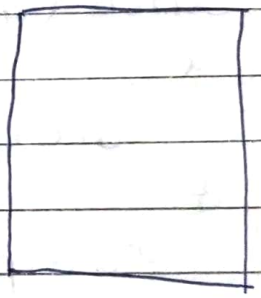
Lec-73

Finally (E.H)

It is always executable block. catch block is executed when exception is generated in try block but finally block is always executed irrespective of whether the exception is generated or not

Use of finally :-

(1) Connection open



```
try
{

}
catch ( )
{

}
```

(2) Connection close (

```
Class Demo
{
psvm (String ar[])
{
    S.o.p ("Connection open");
    S.o.p ("100 lines code");
try
{
int x = Integer.parseInt (ar[0]);
int y = Integer.parseInt (ar[1]);
 S.o.p (x/y);
}
catch (Arithmetic_Exception e)
{
    S.o.p ("Bhanu");
}
   S.o.p ("700 line code");
    S.o.p ("Connection close");
}
}
```

Input	Output	Input	Output
10 2	Connection Op.. 100 lines code 5 Connection close	10 0	Connection Open 100 Arithmetic Connection close
10 abhi	Connection open 100 Numberformet Exception for input Stream abhi.		

The code which we want to execute irrespective ar whether the exception is generated or not then we write that code in finally block. ex. file closing, connection closing etc.

Syntax (1)

```
try
{
}
finally
{
}
```

(2)

```
try
{
}
catch (Exception_name e)
{
}
finally
{
}
```

we can use finally after the try block or after the catch block.

```
class Demo 8
{
 psvm (String ar [])
{
   S.o.p ("Connection open");
   S.o.p (" 100 lines code");
try
{
 int x = Integer.parseInt (ar[0]);
 int y = Integer.parseInt (ar[1]);
 S.o.p (x/y);
}
    finally
{
   S.o.p ("Connection close");
}
```

Input	Output	Input	Output
10, 2	Connection Open 100 Lines Code 5 Connection close	10 0	Connection Open 100 lines code Connection close Arithmetic exception / by 0
10 abhi	Connection Open 100 lines code Connection close Numberformetexception for Input Stream abhi		

```
class Demo
{
psvm (String ar [])
{
  S.o.p ("Connection open");
  S.o.p ("100 lines code");
try
{
  S.o.p ("Softwaves_1");
int  x = Integer.parseInt (ar[0]);
S.o.p ("Softwaves_2");
int  y = Integer.parseInt (ar[1]);
  S.o.p (x/y);
S.o.p ("Softwaves_3");
}
catch (ArithmeticException e)
{
  S.o.p ("Arithmetic...");
}
  finally
{
  S.o.p ("Connection close");
}
  S.o.p ("Softwaves_4");
}
}
```

for Array Index (1more catch block)

if there is no any exception in prog.

(i) try block above code
(ii) try block code
(iii) Catch block (not executed)
(iv) finally block code
(v) below try, catch finally code

10 2	Connection open 100 Lines Code Softwaves 1 Softwaves 2 5 Softwaves 3 connection close Softwaves 4

Case 2: If exception is generated but that exception is handled.

(i) Code above the try block
(ii) try block code till exception
(iii) catch block
(iv) finally block code
(v) Code below try, catch & finally

10 0	Connection open
	100 Lines code
	Softwaves_1
	Softwaves_2
	Arithmetic
	Connection close
	Softwaves_4

Case 3: If exception is generated but not handled.

(1) Code above try block
(2) try block code executed till the exception
(3) catch block code (X)
(4) finally code executed
(5) Code below try, catch finally (not executed)
(6) Exception message.

10 abhi	Connection Open
	100 line code
	Softwaves_1
	Softwaves_2
	Connection close
	NumberformetException.

Lec-44

Throw Keyword

The throw Keyword in java is used to manually throw an exception from a method or any block of code. We can throw either checked or unchecked exception.
The throw Keyword is mainly used to throw custom exceptions.

```
class Ram
{
 psvm (String ar []);
{
  int age = Integer.parseInt (ar [0]);

if (age < 18)
{
  S.o.p ("Invalid Age");
}
 else
{
  S.o.p ("Welcome");
}
  S.o.p ("Bhanv");
}
}
```

```
javac A.java
  java A 15
Invalid Age
Bhanv
```

```
javac A.java
java A 26
Welcome
Bhanv
```

Generating Custom Exception.

```
Class Demo
{
psvm (String ar[])
{

Int age = Integer.parseInt(ar[0]);

if (age < 18)
{
    ArithmeticException ae = new ArithmeticException();
     throw ae;
}
 else
{
   S.o.p ("welcome");
}
   S.o.p (" Softwaves_1");
}
}
```

javac A.java java A 26 welcome Softwaves_1	javac A.java java A 15 ArithmeticException for % by 0.

ArithmeticException ae = new ArithmeticException ("In valid age ="+age);

output => Arithmetic Exception Invalid Age is = 15

Lec-75

:- User Defined Exception :-

In java we can create our own exception class and throw that exception using throw keyword.

These exceptions are known as user-defined or custom exceptions. Java provides us facility to create our own exception which are basically derived classes of Exception:

---

make a file (same fold

```
class InvalidAgeException extends RuntimeException
{
  InvalidAgeException ()
  {
  }
  InvalidAgeException (String s)
  {
     super (s);
  }
}
```

```
class Demo
{
 psvm (String ar[])
{
 int age = Integer.parseInt (ar[0]);
if (age <18)
{
```

```
InvalidAgeException ae = InvalidAgeException ("Invalid
                         age is =" + age);
    throw ae;
}
 else
{
   S.o.p ("welcome");
}
   S.o.p ("Softwave_1");
}
}
```

javac A.java java A 25 welcome Softwaves 1	javac A.java java A 15 InvalidAgeException : Invalid Age is = 15

we can also use

```
throw new InvalidAgeException ("Age is =" + age);
}
```

-: Throws Keyword :-

If any unchecked exception is generated in our program then compiler has no problem with that but, if any checked exception generated in our program then we have to handle it with try catch or we have to throw it out from the method using throws keyword.

```
import java.io.*;
class Demo15
{
psvm (String ar[])
{
  S.o.p ("Softwaves_1");

PrintWriter pw = new PrintWriter ("xyz.txt");
  pw. print ("SoftWaves ");
 pw. close ();
S.o.p ("Softwaves_2...");
}
}
```

Error :- unreported exception FileNotFoundException must be caught or declared to be thrown.

```
  S.o.p ("Softwaves 1");
try
{
PrintWriter pw = new PrintWriter ("xyz.txt");
  pw. print ("Softwaves_1");
pw.close ();
}
Catch (FileNotFoundException e)
{
}
   System.out.println ("Softwaves_2");
}
}
```

Softwaves 1
Softwaves 2

```
psvm (String ar[]) throws FileNotFoundException.
```

Softwaves 1

Lec-6

(i) Campile time pr kabhi exception nhi aati hai. Exception hamesha run time pr hi aata hai.

(ii) Campile time pr jo exception show hoti hai vo exception nhi as a warning show hoti like aapka program is wayh se excecute nhi hoga and 1000 line ka code nhi chalega

Ye sirf campile time pr warning show hoti hai taki program safe rahe.

Like Bus me hame warning di jati hai " Apne Saman ki suraksha swayam kre" iska ye mtlb nhi hai ki hamara saman chori ho jayega bss chance hai isliye warning dete hai.

Same yha jo campile time pr exception show ho rhi hai wo as a warning hai ki exception aane ke chance hai to use handle krr le.

Like Crowd Area me waring hoti hai ki "chor se savdhan rhe"

Lets see it some examples =>

```
import java.io.*;
class Demo12
{
 psvm (String ar[])
{
   S.O.p ("Softwaves_1");
try
{
   PrintWriter pw = new PrintWriter ("xyz.txt");
}
Catch (FileNotFoundException e)
{
  S.O.p ("abhi");
}
  S.O.p ("Softwaves_2");
}
}
```

output →

```
Softwaves_1
Softwaves_2
```

Agr Campile time pr phle jo exception show ho rhi thi vo exception hoti to catch block execute hota or abhi message show hota pr yha catch block execute nhi hua hai Toh,

its a proof ki voh exception Nahi hai sirf Campile time pr warning thi.

Lec -47

```
class Surbhi
{
psvm (String ar[])
{
  S.o.p("Softwaves 1");
  S.o.p (10/0);
S.o.p ("Softwaves 2");
}
}
```

```
Softwaves 1
ArithmeticException / by zero.
```

```
Class Friends
{
psvm (String ar[]) throws ArithmeticException
{
  S.o.p ("Bhanu");
S.o.p (10/0);
S.o.p ("Shailu");
}
}
```

```
Softwaves 1
ArithmeticException / by zero
```

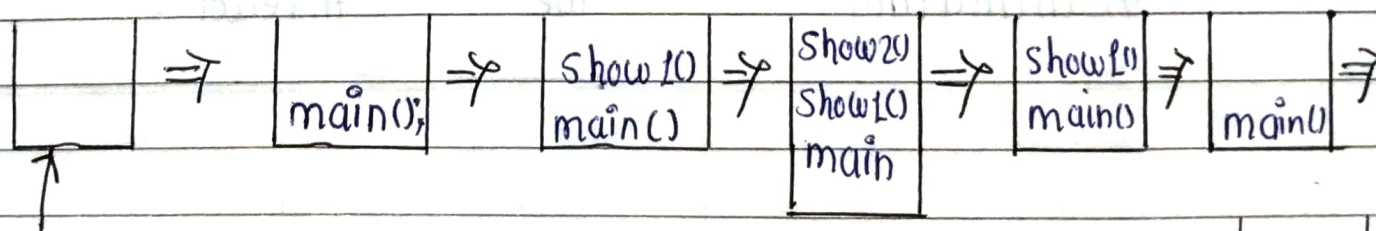
Throws Keyword sirf checked exceptions ko hi method se throw krta hai.

throw Keyword se ham kabhi bhi unchecked exceptions ko remove nahi kr skte hai.

Runtime Stack Mechanism

```
class Ram
{
 static void show2()
{
  S.o.p ("Softwaves_5");
}
static void show1()
{
  S.o.p ("Softwaves_2");
show2();
S.o.p ("Softwaves_3");
}
 psvm (String ar[])
{
  S.o.p ("Softwaves_1");
show1();
S.o.p ("Softwaves_4");
 }
}
```

```
Softwaves_1
Softwaves_2
Softwaves_5
Softwaves_3
Softwaves_4
```



Stack created by JVM

Stack Destroyed By JVM.

Run-time machanism in Java.

for every thread JVM (Java Virtual machine) created a run time stack.

After Completing

JVM stores every methods calls performed by that Thread in the stacks.

After completing every method call by the thread is removed from the Corresponding entry of the stack.

After Completing all the methods, the stack will be empty and that run-time stack will be destroyed by the JVM before the terminating the thread.

Lec-48

```
class Ram
{
 static void show2()
{
   S.o.p ("Softwaves_5");
    S.o.p (10/0);
  S.o.p ("Softwaves_6");
}
 static void show1()
{
  S.o.p ("Softwaves_2");
 show2();
 S.o.p ("Softwaves_3");
}
  psvm (String ar[])
{
 S.o.p ("Softwaves_1");
 show1();
 S.o.p ("Softwaves_4");
}
}
```

```
Softwaves_1
Softwaves_2
Softwaves_5
Arithmetic Exception /by zero
 at Ram.show2()
at Ram.show1()
 at Ram.main()
```

* When an exception is generated in a method then the method handsover it to the JVM then jvm will check whether the exception handling code is in the method or not.
* If the method cantain exception handling code then the exception is handled.
* If the method doesn't cantain exception handling code the jvm will terminate the method abnormally & remove its entry from runtime stack

* Now the JVM will check whether the calling method has handled the exception or not.

If the exception is not handled by the calling function then jvm will terminate that method abnormally and remove its entry from the runtime- stack.

The process is continve untill it reach upto the main method.

* Now JVM will check that the main method contain the code for exception handling or not. If the main method does not handle the exception then the main method is terminated abnormally.

But just before the program terminate the jvm forward the exception to the default exception handler and the default exception hander prints the exception in the form of stack.

Syntax :-

Name of Exception : Description
Stack Trace

```
Class Demo23
{
 static void show2()
{
  S.o.p ("Softwaves_5");
  S.o.p (10/0);
  S.o.p ("Softwaves_6");
}
  static void show1()
{
  S.o.p ("Softwaves_2");
try
{
   show2();
}
  catch (ArithmeticException e)
{
   S.o.p ("Abhi");
}
   S.o.p ("Softwaves_3");
}
   psvm (String arr[])
{
  S.o.p ("Softwaves_1");
 show1();
S.o.p ("Softwaves_4");
}
}
```

```
Softwaves_1
Softwaves_2
Softwaves_5
Abhi
Softwaves 3
Softwaves 4
```

```
Class A
{
 static void show2()
{
  S.o.p ("Softwaves_5");
  S.o.p (1010);
  S.o.p ("Softwaves_6");
}
}
Class B
{
  Static void show1()
{
  S.o.p ("Softwaves_2");
  A.show2();
  S.o.p ("Softwaves_3");
  }
}
Class Demo24
{
 psvm (String ar[])
{
  S.o.p ("Softwaves_1");
  B.show1();
 S.o.p ("Softwaves_4");
 }
}
```

```
Softwaves_1
Softwaves_2
Softwaves_5
ArithmeticException / by zero
at A.show2(Demo.24.java)
at B.show1(Demo24.java)
at Demo24.main(Demo24.java)
```

Lec-79

Exception Propagation

```
impost java.io.*;
class Demo25
{
  static void show1()
{
 PrintWriter pw = new PrintWriter("xyz.txt");
}
 psvm(string ar[])
{
   show1();
    S.o.p("Softwaves4");
}
} Unreported Exception : FileNotFoundException
```

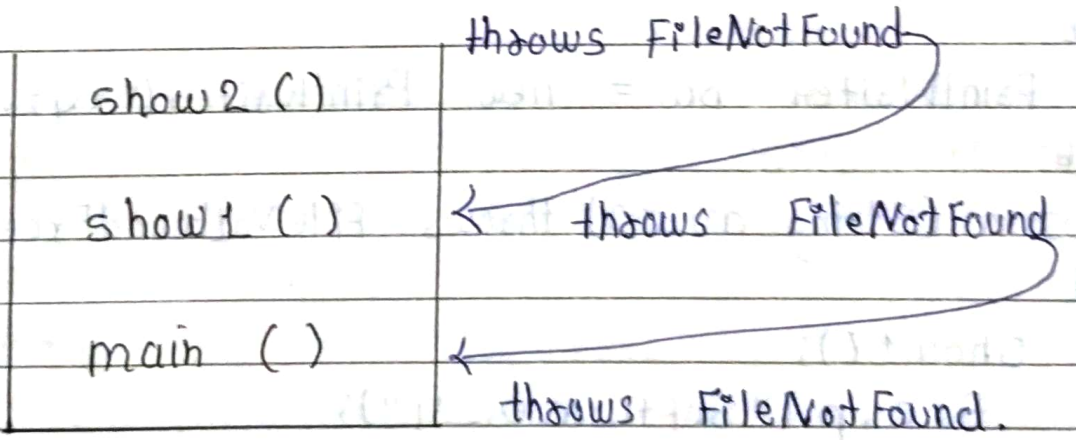
```
class Demo25
{
 static void show1() throws FileNotFoundException
{
  PrintWriter pw = new PrintWriter("xyz.txt");
}
psvm(String ar[]) throws FileNotFoundException
{
   show1();
    S.o.p("Softwaves_4");
  }
}
```

Softwaves—4

If we throws the exception from show1() method then the exception will be forward to the calling method & exception will occur thats why we throws exception from both method

```
import java.io.*;
class Demo26
{
 static void show2() throws FileNotFoundException
{
PrintWriter pw = new PrintWriter ("xyz.txt");
}
static void show1() throws FileNotFoundException
{
  show2();
}
psvm (String ar[]) throws FileNotFoundException
{
  show1();
  S.o.p ("Softwaves 4");
}
}
```

Softwaves 4



The exception will be forwarded to the calling method untill the main or calling method.

Exception Propagation

* When a checked exception is generated in the top of the stack then we have to handle it either by try catch block or we have to throw it from the method using throws keyword.

* If we throw the checked exception using throws keyword then it is dropdown to the calling method / previous method then,

* If it is not handled in the calling method and thrown from the method using throws keyword then it is again drop down to the previous method and the process is continue unit it reach upto the main method.

This process is called as exception propagation.

Lec → 10

```
import java.io.*;
class Demo27
{
  static void show2()
{
   try {
     PrintWriter pw = new PrintWriter ("xyz.txt");
}
  catch (FileNotFoundException e) {
                                  }
}
 Static  void show1()                    Softwaves 4
{
   show2(); }
}
psvm (String ar[])
{
  Show1();
  S.o.p ("Softwaves_4"); }
}
```

Diff throws & try Catch

throws Keyword doesn't handle the exception it only forword the checked exception from that to the calling method but,
try & catch doesn't delegate the exception it can handle both checked & unchecked exception

Note :- Throws Keyword only forward the checked exception from the method to the calling method not unchecked exception.

Lec → 11

Diff. throw & throws

(1)

```
import java.io.*;
class Demo28
{
psvm (String ar[])
{
int age = Integer.parseInt(ar[0]);
if(age<18)
throw new ArithmeticException();
else
S.o.p("welcome");

S.o.p("Softwaves_2");
}
}
```

javac Dem.java
java Demo 15

ArithmeticException

```
import java.io.*;
class Demo29
{
psvm (String ar[]) throws FileNotFoundEx
{
S.o.p("Softwaves 1");

PrintWriter pw = new
        PrintWriter("xyz".txt

S.o.p("softwaves_2");
pw.close();
}
}
```

Softwaves – 1
Softwaves – 2

(2) If we want to generate checked, unchecked, predefined or user defined exception manvally then we have to use throw Keyword. throws Keyword is used to forward the checked exceptions from that to calling method.

(2) throw Keyword is always used inside the method. throws Keyword is always used with method signature.

(3) When we use throw Keyword then we have to create the object of exception we want to generate manvally. When we use throws Keyword then we use the exception class name.

(4) throw :- Generate one exception at a time.
throws :- Can throw multiple checked exception at a time.

Lec - 12

Using More than 1 Catch

```
Class Demo30
{
 psvm (String ar[])
{
 try
{
 int x = Integer.parseInt(ar[0]);
 int y = Integer.parseInt(ar[1]);
   S.O.p (x/y);
}
catch (ArithmethiException | NumberFormatException e)
{
    S.O.p (e);
}
   S.O.p ("Softwaves_22");
}
}
```

```
java Demo30 10 abhi
NumberFormatException
Softwaves_22
```

Note:- We can handle multiple exception with one exception catch block using a pipe (|):

```
import java.io.*;
class Demo31
{
psvm (String ar[])
{
```

Handle All Exception in Single catch

```
try
{
int x = Integer.parseInt(ar[0]);
int y = Integer.parseInt(ar[1]);
 S.o.p (x/y);
}
  catch (Exception e)
{
   S.o.p (e);
}
  S.o.p ("Softwaves_22");
 }
}
```

Input	Output
10 0	Arithmetic Exception : / by zero Softwaves_22
10 abhi	NumberFormetException : for input string "abhi" Softwaves_22
10	Array Index Out Of Bonds Exception Softwaves_22

We can handle all the exceptions in single catch block by using all exceptions Super class "Exception". It will automatically handle all the exceptions which was generated or occur.

```
Class Demo31
{
psvm (String ar[])
{
 try
{
int x = Integer.parseInt (ar[0]);
int y = Integer.parseInt (ar[1]);
 S.o.p (x/y);
}
 catch (ArithmeticException e)
{
   System.out.println ("Bhanu");
}
 Catch (Exception e)
{
   System.out.println ("Shailu");
}
 S.o.p ("Softwaves_22");
}
}
```

Input	Output
10 0	Bhanu Softwaves_22
10 abhi	Shailu Softwaves_22
10	Shailu Softwaves 22

Lec-13

```
import java.io.*;
class Demo34
{
psvm (String args[])
{
try
{
int x = Integer.parseInt (ar[0]);
int y = Integer.parseInt (ar[1]);
System.out.println (x/y);
}
catch (ArithmeticException e)
{
   S.o.p ("Shailu");
}
catch (ArithmeticException e)
{
S.o.p ("Bhanu");
}

S.o.p ("Softwaves");
}
}
```

Error:- ArithmeticException has already catchght.

```
class Demo33
{
psvm (String ar[])
{
int x = Integer.parseInt (ar[0]);
int y = Integer.parseInt (ar[1]);
S.o.p (x/y);
}
catch (Exception e)
{
  S.o.p ("Aman");
}
catch (ArithmeticException e)
{
 S.o.p ("Ambu");
}
}
```

Error :- ArithmeticException is already caught.

Note :- When we use multiple catch block with one try blocks then we can use super class only in last catch block. Because if use it another place then it will give error.

Because super class can handle all the exceptions.

```
class Demo35
{
public static void main (String ar[])
{
try
{
int x = Integer.parseInt (ar[0]);
int y = Integer.parseInt (ar[1]);
S.o.p (x/y);
}

catch (ArithmeticException | Exception e)
{
   System.out.println ("Shailu");
}
    S.o.p ("Bhanu");
}
}
```

Error :- Alternatives in a multi-catch statement can not be related by subclassing.

Note :- When we handle multiple exception with one catch block using a pipe (|), then,

it should not contain any parent - child relationship between the exceptions.

Lec → 14

Different Ways to Display Exceptions

```
Class Demo36
{
psvm (String ar[])
{
try
{
 int x = Integer.parseInt(ar[0]);
 int y = Integer.parseInt(ar[1]);
S.o.p (x/y);
}
Catch (Exception e)
{
   // S.o.p (e);
  S.o.p (e.toString());
}
  S.o.p ("Softwaves_33");
}
}
```

Input	Output
10 0	Arithmetic Exception : / by zero Softwaves 33
10 abhi	NumberFormatException for input String abhi Softwaves 33

(1) **Note**:- toString() method :- By using this method, we will only get exception name and description of an exception.

(i) toString () => Excep_Name : Descrip.
(ii) getMessage () => Message
(iii) printStackTrace() => Excep_Name : Descrip.
at (Method name)

(2) getMessage():- By using this method, we will only get description of an exception.

(3) printStackTrace() :- By using this method, we will get exception name and discription of an exception separated by colon, and stack trace in the next line.

	Method	Exception showing
(1)	toString()	Arithmetic Exception : / by zero Softwaves 33
(2)	getMessage()	/ by zero Softwaves 33
(3)	printStackTrace()	NumberFormetException : for input string abh at java.lang.Integer.parseInt( ) at Demo36.main ( );

S.o.p (e.getMessage());
S.o.p (e.printStackTrace());
] in catch block

```
class A
{
 void show1()
{
   try
  {
     S.o.p (10/0);
  }
Catch (Exception e)
 {
     e.printStackTrace (); }
  }
}
class B
{
  void show2()
{
     A a= new A();
      a.show1();
   }
}
class Demo38
{
 psvm (String ar[])
{
   B b= new B();
b. show2();
S.o.p ("Softwaves 2");
}
}
```

```
ArithmeticException: / by zero
at A.show1();
at B.show2()
at Demo38. main ()
Softwaves 2
```

Note :- Default Exception handler use printStackTrace() method to print the exception message therefore it always prints the exception in the form of stack.

Lec - 15

(i) :- Fully checked Exception :-

The checked exception which have all checked subclasses is called a fully checked exception.
ex. IOException.

(ii) Partially Checked Exception :- The exception which have the both checked and unchecked subclasses is called a partially checked exception.
ex. Throwable, Exception.