Exception

↳ Unwanted interruption & unexpected interruption cause in normal flow. called Exception.

→ Exception handling

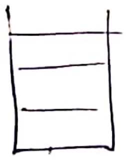
↳ If something goes wrong, then also your workflow need to continue as follows.

↳ Gracfull termination of program is the main objective

↳ To define the alternative way to continue rest of the program.

→ Runtime Stack mechanism

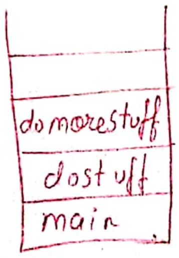
For every thread - JVM will create run time stack.



Each & every ~~thread~~ operation performed by a thread is put in stack.

Once all operation successfully complete JVM will destroy stack. Just Before termination.

→ Default Exception Handling



```
domorestuff(){
  x = 10/0;
}
```

→ If any exception occur then that calling method is responsible for Create an object with all the details, location, type, method stack trace.

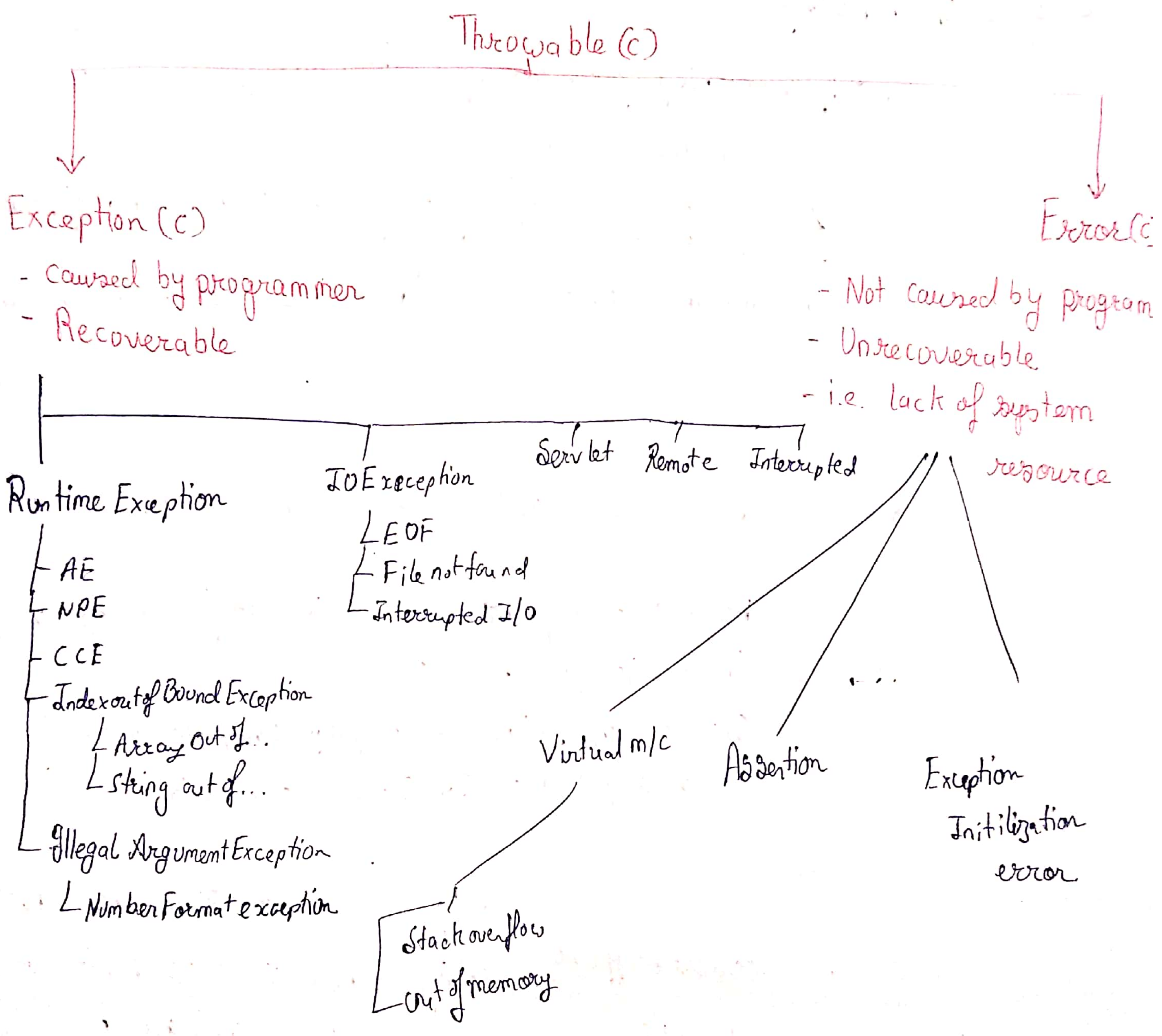
→ JVM will interrupt all its further execution of that method.

Then JVM called default exception handling it will print exception error(e) and program terminated.

Exception in thread "xxx" Name of exception : Description

Stack Trace

Exception Hierarchy



→ Checked & Unchecked Exception

Runtime only

→ The exception which are checked by the compiler, for smooth execution at runtime - checked exception

10th Std

mom query

→ possibility of Filenotfoundexception

→ unreported exception java.io.FilenotFoundException must be caught or declared to be thrown.

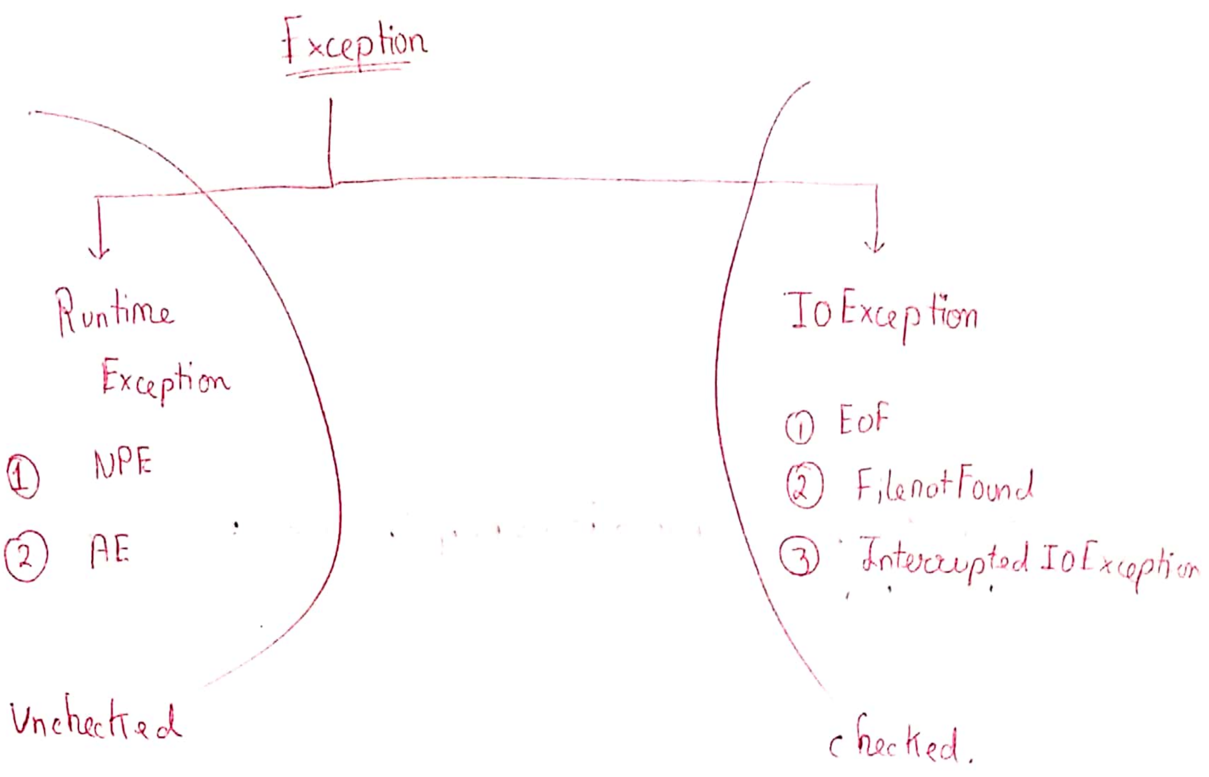
→ The exception which are not checked by the compiler, wheather programmer handling or not - unchecked exception

→ Both checked & unchecked exception occur only run time

→ Runtime Exception and its child classes } Unchecked Exception
Error and its child classes

→ Remaining exception classes } checked Exception

Fully checked Vs partially checked



If the class exception is fully checked iff its child classes are checked exception i.e. IoException

If the class exception is ~~not~~ partially checked iff its child class are not checked | unchecked exception i.e. [Exception. throwable] only 2

Customized Exception handling (by using try/Catch)

→ If we don't want to terminate the abnormally.

Without try-catch

```
psvm(String[] args)
{
    Sout("Stmt 1");
    Sout (10/0);
    Sout ("Stmt 3");
}
```

```
O/p : Stmt1
      R.E.  1 by zero.
```

Abnormal Termination

With try-catch

```
psvm(String[] args)
{
  Sout (" Stmt 1");
  try
  {
     Sout (10/0);
  }
  ~~Sout ("Stmt 3");~~
  }
  catch (Arithmetic Exception e)
  {
     Sout(10/2);
  }
  Sout("Stmt 3");
}
```

```
O/p:  Stmt 1
       5
      Stmt 3
```

```
try
{
 Stmt 1
 Stmt 2  ←      O/p 1, 4, 5
 Stmt 3
}
catch {
  Stmt 4
}
 Stmt
```

→ If ~~anye~~ any exception occur outside try block it will terminate abnormally.

Methods to print exception information

i.e.

- e. PrintStacktrace → Name of exception | description | Complete stacktrace.
- e
- e. tostring() → Name of exception | description |
- e. getMessage → description

Print Stacktrace → Internally handle default exception

Try with multiple catch

Good programming practise

```
try
{
    ________
}
catch (AE)
{
    ________
}
catch (EOF)
{
    ________
}
catch (FNFound)
{
    ________
}
catch(exception e)
{
    default-________
}
```

→ **Final**
- class — No inheritance
- method - override
- variable - constant

→ **Finally** — Is a associated with block [try-catch] to maintain cleanup code.

```
try
{
    Risky Code
}
Catch (Exception e )
{
    Handling Code
}
~~Catch~~
finally
{
    cleanup Code
}
```

→ To cleanup activity associated with try block.

→ **Finalize()** → It is called by garbage Collector.

When → Just before destroying an object

Garbage collector always finalize method to cleanup activity

Once the finalize method completes automatically G.C. will destroy the object.

→ To cleanup activity associated with object resources.

→ Various possible combination of try, Catch, finally

try	try	try.	catch	finally
catch	finally	try without catch or finally.		
✓	✓	✗	✗	✗

try	try
finally	sop
catch	catch
✗	✗

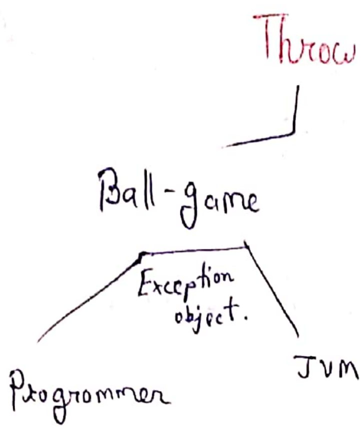
```
try
{
    try
    {
    }
    catch(x e)
    {
    }
catch (x e)
{
}   ✓
```

Try → catch → finally

⇓

Order is important

Throw & Throws
→ Dangerous.



Case-1

```
class Test{
    psvm(String[] args)
    {
        Sout(10/0);
    }
}
```

Exception handling done by JVM Internally

Hand over Internally

Case-2

```
class Test{
    psvm(String[] args)
    {
        throw new AE("/by zero");
    }
}
```

throw → Hand over our created exception object to the JVM manually

new AE("/by zero") → Creation of Arithmetic Exception object explicitly

Customized ~~Algorithm~~ exception

We can throw only exception & error not a normal java class.

```
class Test extends RuntimeException
{
  psvm(String[] args)
  {
    throw new Test();
  }
}
```

Exception in thread main: Test
at Test.main().

Throws

- → Unexpected Checked exception must be caught or declared to be thrown.
- → Throws Keyword is used to deliver the exception handle to the caller.

  caller.
  - JVM
  - Method

```
class Test
{
   psvm(String[] args)
       throws InterruptedException
   {
     thread.sleep(100);
   }
}
```

- → It is required only to conveince compiler and usage of throws does not prevent abnormally termination of program.

```
class Test Extends Runtime Exception
{
   psvm(String[] args) throws Test
   {

   }
}
```
✓

→
```
class Test
{
  PSV Main(String[] args)
  {
    throw new Exception();
              └ checked
  }
}
```

C.E. → Unreported Exception: Exception must be caught or declared to be thrown.

→
```
class Test
{
  PSV Main(String[] args)
  {
    throw new Error();
  }           └→ Unchecked
}
```

R.E. → Exception in thread 'main' Error at test.main()

```
try
{
  Sout("Hello");
}
catch (IOException e)  ————————→  catch (Interrupted e)
{                                  {
}                                  }
```

Compiler time error
Exception java.io is never thrown corresponding try Statement

Only applicable for fully checked exception

Keyword Summary

1. try - to maintain risky code
2. catch - to maintain exception handling code
3. finally - To maintain cleanup code
4. throw - To handover our created exception object to the JVM manually.
5. throws - To delegate responsibility of creation handling to the caller.

Compiler time error in exception handling.

1. Unreported exception xxx ; must be caught or declared to be thrown.
2. Exception xxx has already been caught.
3. Exception xxx is never thrown in body of corresponding try statement.
4. Unreachable statement
5. ~~in~~ incompatible types
   found : Test
   required : java. lang. throwable
6. try without catch or finally
7. catch without try
8. finally without try.

[illegible]

Customized - User defined Exception

```
class TooyoungExceptions Extends RuntimeExceptions
{
     TooyoungExceptions (String s)
     {
      Super(S);
     }
}
```

→ Throw keyword is best suitable for user defined or customized Exceptions. But not for pre-defined exceptions (Unchecked)

① ArrayIndexoutof Bound Exception

└> Runtime Exception - Unchecked

```
int[] x = new int[4];
x[5];   ✓ Exceptions
```

② Nullpointer Exception

```
String S = null;
S.length();
```

③ ClassCast Exception

└ Runtime

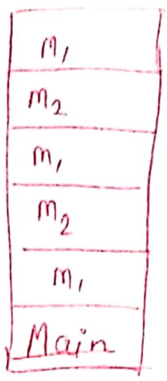
└ Unchecked

```
String S = new String("Durga");
object o = object(s)
```

```
String S = String (O)
```

$\boxed{\text{Exception;}}$

(4) StackOverflow Error — Unchecked
↳ Runtime



⑤ NoClassdefFoundError - child class of Error - Unchecked.
↳ If JVM not able to found the class.

⑥ ExceptionInInitializerError. - child class of error - Unchecked

```
class Test
{
    static int i = 10/0
}

static
{
    String s = null;
    Sop ( s.length());
}
```

7) Illegal ArgumentException
   ↳ Runtime
      ↳ Unchecked.

```
Thread t = new Thread();
t.setpriority (7) ✓
t.setpriority (15) ×
```

⑧ Number Format Exception

```
int i = Integer.parseInt ("10");
int i = Integer. parseInt ( ten);
```

⑨ Illegal State Exception

```
t.start()
 .
 .
 :
t.start(); × R.E = Illegal State Exception.
```

⑩ Assertion Error

```
assert (x > 10);
```

↳ ~~Runtime Exception~~tion Error
   ↳ Assertion error

→
```
try
{

    ______
}
catch (AE | NPe )
{

    ________
}
```

→ Re-throwing Exception

```
try
{

}
catch (Arithmetic Exception e)
{
  throw new Nullpointer Exception();
}
```