4.

* Exception Handling (70 - 79)

* Regular Expressions (133 - 135)

# #10. Exception Handling

17-Apr-23

1. Introduction
2. Runtime stack mechanism
3. Default exception handling in java.
4. Exception Hierarchy
5. Customized exception handling by using try-catch
6. Control flow in try-catch
7. Methods to print exception information.
8. try with multiple catch blocks.
9. finally block.
10. Difference between final, finally, finalize.
11. control flow in try-catch-finally
12. control flow in nested try-catch-finally.
13. various possible combinations of try-catch-finally
14. throw keyword
15. throws keyword
16. Exception handling keywords summary.
17. Various possible compile time errors in excp. hand
18. Customized (or) user defined exceptions.
19. Top-10 exceptions
20. 1.7 version enhancements
    1. try with resources
    2. multi catch block.

# * Introduction :~

→ An unexpected, un-wanted event that disturbes normal flow of program is called exception.

Eg :- TyrePunchedException, SleepingException.
FileNotFoundException.

> open db connection
SqlExcept Read data
close database

→ It is highly recommended to handle Exceptions.

→ The main objective of Exception handling is graceful termination of the program.

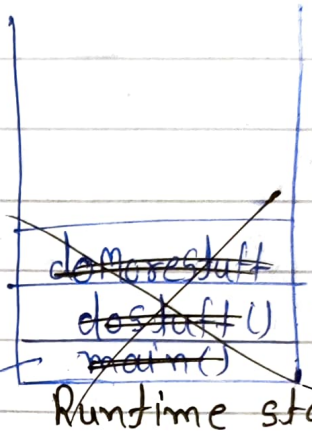
```
> try {
    read data from remote file location at londan
  }
  Catch (FileNotFoundException e) {
      use local file & continue rest of
      the program normally
  }
```

→ Exception handeling doesn't mean repearing an exception. we have to provide alternative way to continue rest of the program normally, is the concept of exception handeling.

→ Purpose :- Graceful termination of the program.

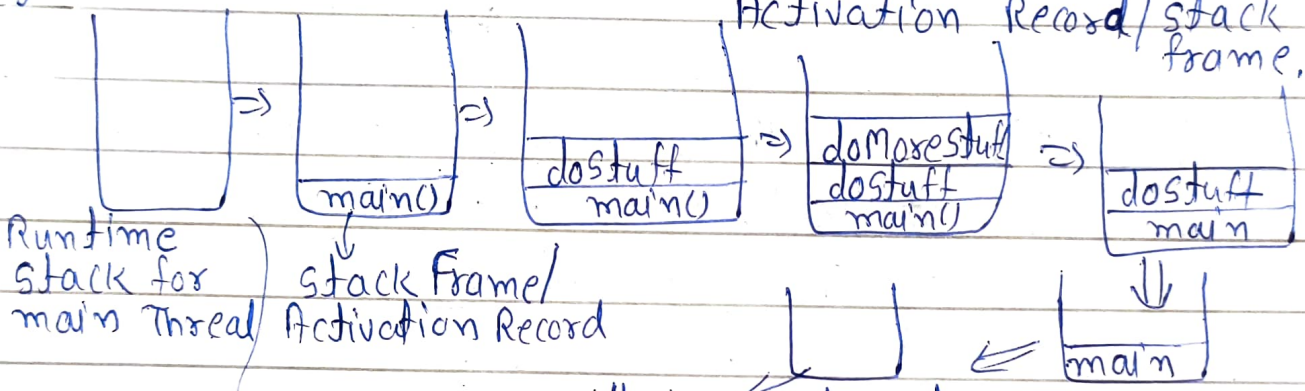
# * Runtime Stack Mechanism :-

```
> class Test {
      p s v m (S [] args) {
          dostuff();      }
      p s v dostuff() {
          doMorestuff(); }
      p s v doMoreStuff() {
          Sopln("Hello"); }
  }
```




O/p: Hello

Activation Record / stack frame.




will be distroyed by JVM.

→ For every thread JVM will create a runtime stack.

→ Each & every method call performed by that thread will be stored in corresponding stack.

→ Each entery in stack is called stack Frame.

→ After completing every method call the corresponding entery from the stack will be removed.

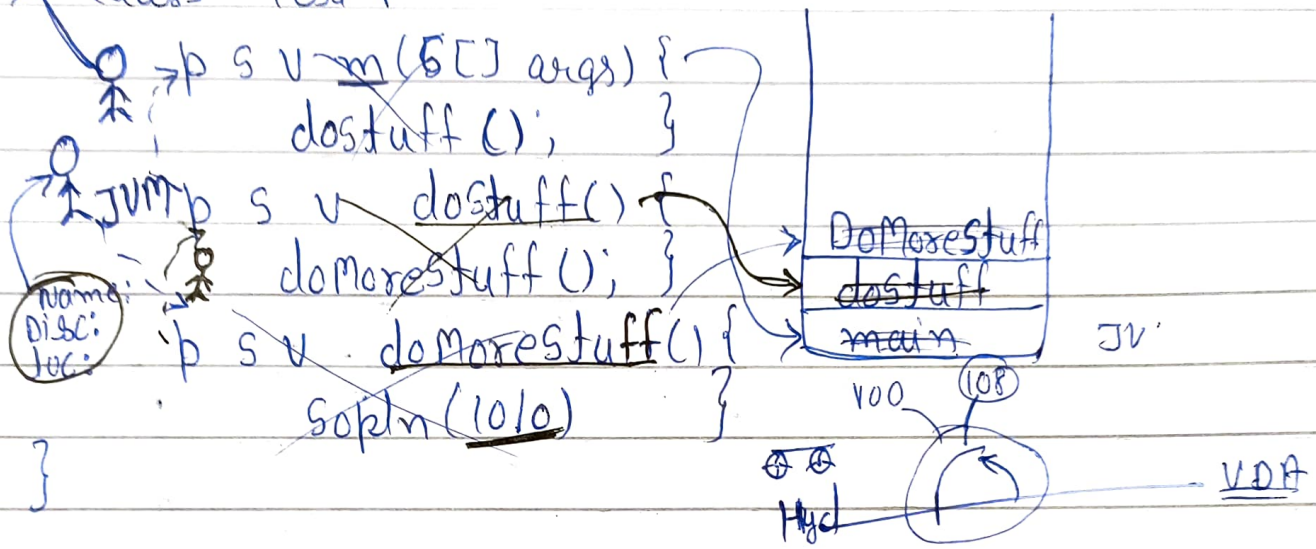
→ After completing all method calls the stack will become empty. & it will be distroyed by JVM just before termenating the thread.

# #71. Exception Handling

17/04/23

## * Default Exception Handling

```
class Test {
    p s v m(S[] args) {
        dostuff();
    }
    p s v dostuff() {
        domorestuff();
    }
    p s v domorestuff() {
        Sop(10/0)
    }
}
```




Exception in thread "main" : J.l.A E: / by zero
at Test. doMoreStuff (:13)
at Test. dostuff( :9)
at Test. main ( :5)

→ Inside a method if any exception occurs the method in which it raised is responsible to create exception object by including the following info.
① Name of Exception.
② Description of Exception.
③ Location at which exception occurs [Stack Trace]

→ After creating exception object method hands over object to the JVM.

→ JVM will check wether the method contains any method handeling code or not.

→ If the method doesn't contain any exception handeling code the JVM terminates that method abnormally & removes corresponding entery from stack.

→ Then JVM identifies caller methods & checks wether caller method contains any handeling code or not.

→ If the caller method doesn't contain handling code then JVM terminates that caller method also abnormally & remove corresponding enteries from stack.

→ This process will be continued until main method & if the main method also doesn't contain handling code then JVM terminates main method abnormally & removes entry for stack.

→ Then JVM handovers responsibility of exception handeling to default exception handler which is the part of JVM.

→ Default exception handler prints exception information in following format & terminates program abnormally.

> Exception in thread "xxx" Name of Exception: Description
>
> Stack Trace.

→ In a if atleast one method terminates abnormally then the programe termination is abnormal term.

## * Exception Hierarchy :-

Throwable

→ Throwable class acts as root for java exception Hierarchy.

→ Throwable defines two child classes
1) Exception 2) Error.

## * Exception :-

→ Most of the times Exceptions are cause by our program & these are recoverable.

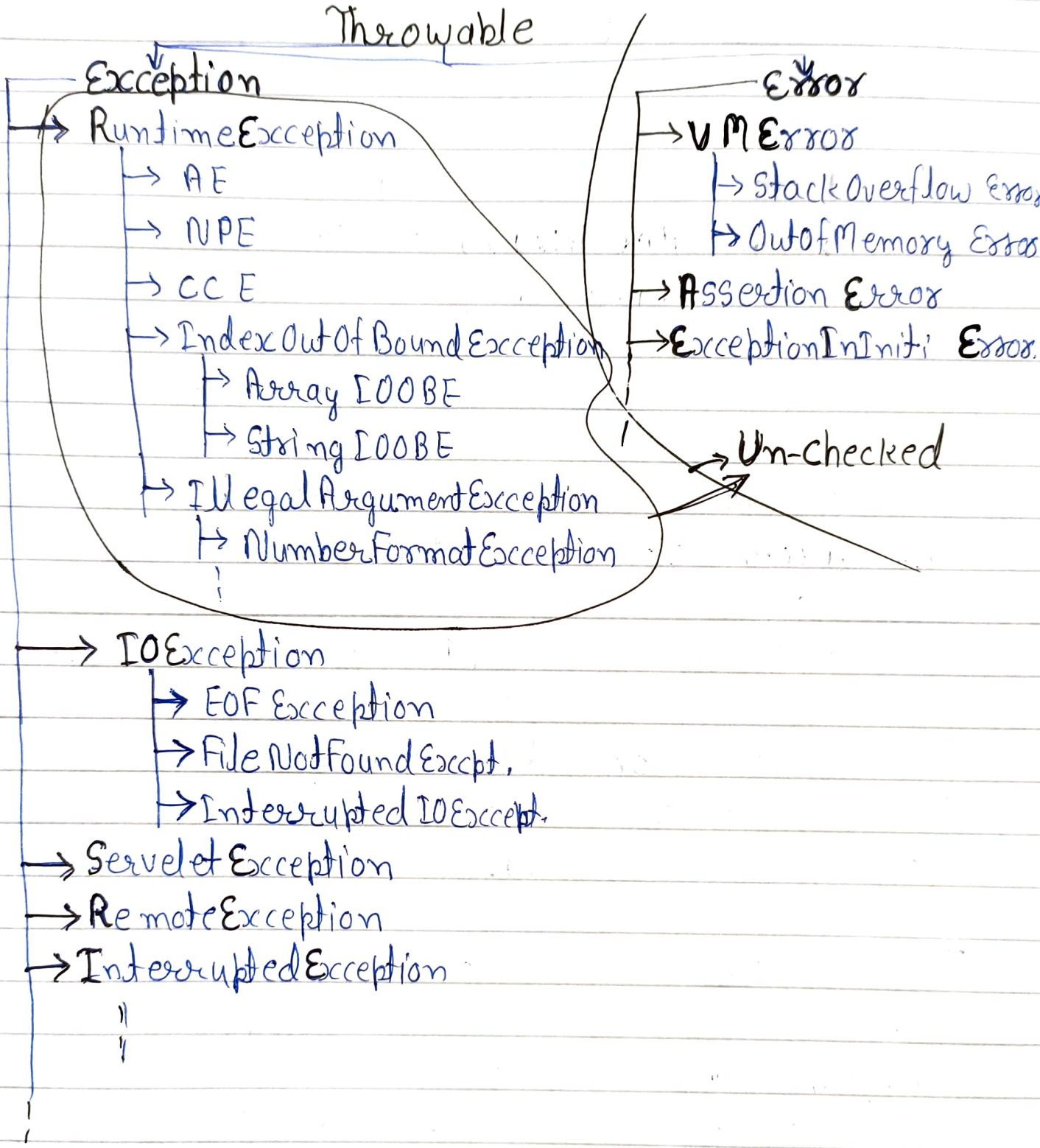
eg:- If our programing requirement is to read data from remote file locating at londan at runtime if remote file is not available the we will get Runtime exception : FileNotFound Ex. If this occurs we can provide local file & continue rest of the program normally.

## * Error :-

→ Most of the time errors are not cause by our program & these are non-re-coverable.

→ These are due to lack of system resources.

→ If OutOfMemory error occures we can't do any thing. System admin is responsible to increase heap memory.


Throwable
Exception
RuntimeException
AE
NPE
CCE
IndexOutOfBoundException
ArrayIOOBE
StringIOOBE
IllegalArgumentException
NumberFormatException
IOException
EOFException
FileNotFoundExcept.
InterruptedIOExcept.
ServletException
RemoteException
InterruptedException
Error
VMError
StackOverflowErro
OutOfMemoryErro
AssertionError
ExceptionInIniti Error
Un-checked

# #72. E H Checked, UnChecked Exception

17-Apr-23

****

## * Checked .io.*; Vs UnChecked Exception.

> class Test {

```
p s v m(S[] args) {
    PrintWriter pw = new PritWriter ("abc.txt");
    pw.println("Hello"); }
}
```

CE: unreported exception java.io.FileNotFound E, must be caught (or) declared to be thrown.

→ The exceptions which are checked by comiler for smooth execution of the program are called checked exception.
eg:- HallTicketMissing Excpt$^n$, PenNotWorking Excpt$^n$, FileNot

→ In our program if there a chance of raising checked exception then compulsary we should handle that checked exception (either by try-catch (or) throws key-word. Otherwis we will get CE.

→

⇒ The exception which is not checked by compiler wether programmer handeling or not. Such type of exceptions are called un-checked Exceeptions.
eg:- Arithmetic Exception, BomBlast Exception etc.

Note:- Wether it is checked or Un-checked every exception occurs at Runtime only. there is no-chance of occuring any Exception at compile time.

→ RuntimeException & it's child classes, Error & it's child classes are un-checked. except these remaining are checked.

* Fully checked Vs Partially checked :-

→ A checked exception is said to be fully checked iff all it's child class is also checked.
eg :- IOException, InterruptedException etc.

→ A checked exception is said to be partially checked iff some of it's child classess are un-checked.
eg :- Exception, Throwable.

Note :- The only possible partially-checked exceptions are ① Exception ② Throwable.

> IOException - Checked (Fully)
> RuntimeException - Un-checked
> InterruptedException - Checked (Fully)
> Error - Un-checked
> Throwable - Checked (Partially)
> ArithmeticException - Un-checked
> NullPointerException - Un-checked
> Exception - Checked & Partially)
> FileNotFoundException - Checked (Fully)

# #73. EH. Customized Exception Handeling

17-09-23

* Customized Exception Handeling using try-catch

* without try-catch

```
> class Test {
      p s v m(S[] args) {
          Soln("start 1");
          Soln(10/0);
          Soln("st-3"); }
  }
```

O/p: start 1
RE: AE: / by zero
Abnormal Termination

* With try-catch

```
> class Test {
      p s v m(S[] args) {
          Soln("stat-1");
          try {
              Soln(10/0); }
          Catch (Arithmetic Exception e) {
              Soln(10/2); }
          Soln("stat-3"); }
  }
```

O/p:
stat 1
5
stat 3

```
> try {
      Risky code }
  Catch (Exception e) {
      Handling code
  }
```

→ It is highly recommended to handle exceptions the code which may raise an exception is called Risky code & we have to define that code inside try-block & corresponding handeling code ~~in corresponding~~ inside catch block.

## * Control flow in try-catch :-

```
try {
    stat 1 ;
    stat 2 ;
    stat 3 ; }
catch ( X e ) {
    stat 4 ; }
stat 5 ;
```

C-I :- If no exception
1, 2, 3, 5, NT

C-II :- If exception S-2 & match
1, 4, 5, NT

C-III :- If Excpt$^n$ S-2 & not-match
1, AT

C-IV : If Excpt$^n$ S-4
AT

→ If exception occurs at stat-4 & 5 always A.T.

Note :- Within try block if anywher exception raised then rest of the try block won't be executed. eventhough we handled that exception. hence within the try-block we have to take only risky take, & length of try block should be as less as possible.

→ In addition to try block there may be chance of raising an exception inside catch & finally blocks.

→ If any statement which are not part of try block & raises an exception then it is always abnormal termination.

* Methods to print Exception information.

```
> class Test {
      p s v m(S[] args) {
        try {
          Sopln(10/0); }
        Catch (ArithmeticException e) {
          e.printStackTrace();
          Sopln(e); (or) Sopln(e.toString());
          Sopln(e.getMessage());
        }
      }
}
```

/ by zero — Description

Java.lang.AE : / by zero — Name : Description.

Java.lang.AE : / by zero
at Test.main() — Name : Desc. Trace.

→ Default Exceptional handler will use printStackTrace() method.

* try with multiple catch block :-

```
> try {
      Risky code }
  Catch (Arithmatic Exc. e){
      perform alternative arth. operation }
  Catch { SQLException e) {
      use MySQL db insted of oracle db }
  Catch ( Exception e'){
      // default exception - handeling
  }
```

Best prog. practice

→ The way of handeling an exception is varied from exception to exception hence for every exception type it is heighly recommended to take seperate catch block. i.e try with multiple catch block is always possible & it is recommended to use.

```
try {
    Risky code }
Catch { exception e) {
  }
```

worst prog. practice.

```
> try {
    Risky code }
  catch (Exception e) { }
  catch (ArithmExc. e){
  }
```

CE: already caught

```
> try {
      Risky code }
  Catch (ArthiEx. e) { }
  Catch (Exception e) { }
```

# #74. EH finaly block

18-Apr-23

→ If try with multible catch block present then the order of catch block is very important. We have to take child first then parent other -wise we will get CE:.

> try {
Risky code }
Catch (AE e){ }
Catch ( AE e) { } CE: already been caught.

→ We can't declare two catch block for same exception otherwise we will get CE:~

* finally block :~

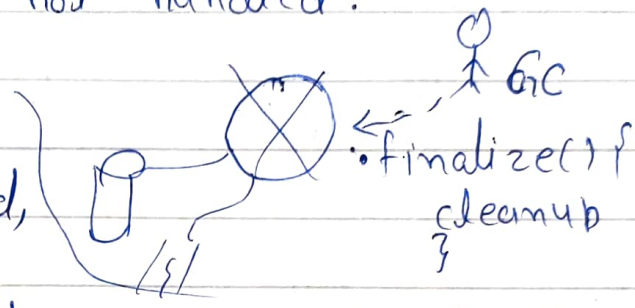
① final :
final is a modifer, applicable for classes method & variables. If class declared as final then we can't extend that class (Inheritance not p
→ If method is final, we can't override that method
→ If variable is final, then we can't re-assignment for that variable.

② finally :
finally is a block, always associated with try-catch to maintain clean-up code.

```
try {
    Risky code }
catch (Exception e){
    Handling code }
finally {
    Cleanup code }
```

→ It will be executed always irrespective of wether exception is raised or not raised or wether handled or not handled.

③ finalize() :-




finalize is a method, always invoked by GC just before distroying an object to perform clean-up activity.

→ Once finalize method completes immediately GC

Note: Finally block is responsible to perform clean-up activity related to try block i.e. whatever resources we opened as try block will be closed inside finally block.

→ whereas finalize method is responsible to perform clean-up activity related to object i.e whatever resources associated with object will be de-allocated before distroying an object by using finalize method.

* Various combination of try-catch-finally.

① → try { }
catch(xe)

② → try { }
Catch(xe){}
catch(ye){}

③ → try { }
catch(xe){}
catch(xe){}
CE X

④
```
try { }
catch(x e) { }
finally { }
```

⑤
```
try { }
finally { }
```

⑥
```
try { }
catch(x e) { }
try { }
catch(y e) { }
```

⑦
```
try { }
catch(x e) { }
try { }
finally { }
```

⑧
```
try { }
```
CE: ✗

⑨
```
catch(x e) { }
```
CE: ✗

⑩
```
finally { }
```
CE: ✗

⑪
```
try { }
finally { }
catch(x e) { }
```
✗ CE:

⑫
```
try { }
Sop(n("Hello");
catch(x e) { }
```
CE: 1 | CE: 2 ✗

⑬
```
try { }
catch(x e) { }
Sop(n("Hello")
catch(y e) { }
```
✗ CE:

⑭
```
try { }
catch(x e) { }
Sop(n("Hi");
finally { }
```
✗ CE:

⑮
```
try {
    try { }
    catch(x e) { }
} catch(x e) { }
```

⑯
```
try {
    try { }
} catch(x e) { }
```
CE: ✗

⑰
```
try {
    try { }
    finally { }
} catch(x e) { }
```

⑱
```
try { }
catch(x e) {
    try { }
    finally { }
}
```

⑲
```
try { }
catch(x e) {
    finally { }
}
```
✗ CE:

⑳
```
try { }
catch(x e) { }
finally {
    try { }
    catch(x e) { }
}
```

㉑
```
try { }
catch(x e) { }
finally {
    finally { }
}
```

㉒
```
try { }
catch(x e) { }
finally { }
finally { }
```
✗ CE:

㉓
```
try
    Sop(n("try");
catch(x e) {
    Sop(n("catch");
} finally { }
```
✗

(24)

```
try { }
Catch (x e)
    Sop1n ("catch");
finally { }
```

✗

(25)

```
try { }
catch (x e) { }
finally
    Sop1n ("finally");
```

✗

→ In try-catch-finally order is important.

→ whenever we are writing try compulsary we should write either catch or finally. i.e try whithout catch & finally is invalid.

→ whenever we are writing catch block compulsary try block must be required. i.e catch without try is invalid.

→ whenever we are writing finally block compulsary we should write try block i.e finally without try is invalid.

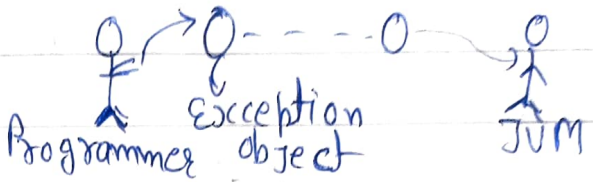
→ Inside try, catch & finally blocks we can declare try-catch & finally blocks i.e nesting of try-catch-finally is allowed.

→ for try-catch & finally blocks { } are mandatory.

# #75. EH throw & throws

19/Apr/23

## * throw keyword :-




```
> class Test {
    p s v m(S[]args){
        Sop(ln(10/0);}
  }
```

```
>class Test {
    p s v m(S[] args)
    throw new AE("/by zero");
  }
```

hand-over our created exception object to the JVM manually

creation of AE object explicitly

→ Sometime we can create exception object explicitly, we can handover to JVM manually for this we have to use throw key-word.

→ The main objective of throw key-word is to hand-over our created exception to the JVM. manually.

→ The result of above 2-program is exactly same

P-① here main method is responsible to create exception object & handover to JVM.

P-② in this case programmer creating exception object expli. & handover to JVM manually

```
> withdraw(double amount){
      if (amount > balance)
          throw new InsufficientFundsException();
  }
```

→ Best use of throw key-word is for user defined exceptions or customized exception.

C-I :-

```
> class Test {
      static AE e = new AE();
          p s v m(S[] args) {
              throw e;          }
  }
```

RE: AE

```
> class Test {
      Static AE e;
      p s v m(S[] args) {
          throw e;          }
  }
```

RE: : NPE

> throw e; | If e refers null then we will get Null Pointer Exception

C-2

```
> class Test{
      p s v m (S[] args){
          Sopln(10/0);
          Sopln ("Hello"); }
  }
```

RE: AE : / by zero

```
> class Test {
      p s v m (S [] args){
          throw new AE("/ by zero");
          Sopln ("Hello");          }
  }
```

CE: Unreachable statement.

→ After throw statement we are not allowed to adwrite any statement directly otherwise we will get CE: unreachable statement.

C-III

> class Test {
    p s v m(S [] args) {
        throw new Test();}
}
CE: incompatible.

> class Test extends RuntimeException {
    p s v m(S [] args ) {
        throw new Test(); }
} RE:

→ We can use throw keyword only for throwable types. if we are trying to use for normal java objects we will get CE: incompatible types.

# #76. EH throws keyword

19/Apr/23

* throws keyword :-

eg:-1

```
> import java.io.*;
> class Test{
      p s v m(S[] args){
          PrintWriter pw = new PrintWriter("abc.txt");
          pw.println("Hello"); }
  }
```

CE: unreported exception ....

→ In our program, if there is possibility of raising checked exception then compulsary we should handle that checked exception otherwise we will get compiletime error saying : unreported exception

eg:-

```
> class Test{
      p s v m(S[] args){
          Thread.sleep(10000); }
  }
```

CE: unreported exception InterruptedException.

→ We can handle this compile time error by using the following two ways.

① By using try-catch.

```
main(S[] a){
    try{
        Thread.sleep(10000);
    }catch(InterruptedException e){
    }
}
```

② By using throws :-

```
p s v m (S [] args) throws IE
    Thread.Sleep(10000);
}
```

→ we can use throws keyword to delegate responsi-bility of exception handling to caller (it may be another method or JVM) then caller method is responsible to handle that exception.

→ Throws keyword is required only for checked exception & usage of it for unchecked exception, no use or impact.

→ ~~Throw~~ It is required only to convence compiler & usages of throws keyword doesn't prevent abnormal termination of program.

> class Test {

```
p s v m(S[] args){ throws IE {
    doStuff();      }
p s v doStuff() { throws IE{
    doMoreStuff(); }
p s v doMoreStuff() { throws IE {
    Thread.Sleep(10000); }
}
```

CE:

→ If in above prog. if we remove atleast one throws statement the code won't compile.

* Throws clause

1) We can use to delegate responsibily of EH to the caller.
2) It is required only for checked exceptions.
3) It is required only to convience compiler & usage of throws does not prevent abnormal termination of program.

→ It is recommended to use try catch over throws keyword.

Case-I :-

```
class Test throws Exception {
    Test() throws Exception {
    }
    public void m() throws Exception {
    }
}
```

→ We can use throws keyword for methods & constructors but not for classes.

Case-II :-

```
class Test {
    public void m() throws Test {
    }
}
```

↳ CE:

```
class Test extends RuntimeException {
    public void m() throws Test {
    }
}
```

→ We can use throws keyword only for throwable types. normal java classes ⇒ CE: incompatible type

Case-III

```
class Test {
    p s v m(S[] args){
        throw new Exception();}
}
```
↳ checked CE:

```
class Test {
    p s v m (S[] args){
        throw new Error();}
}
```
↳ unchecked

RE:

Case-IV

```
class Test {
  p s v m(S[] args){
    try {
      Sopln("Hello");}
    Catch(AE e) { } }
}
```
I/O: Hello → Unchecked

```
class Test {
  p s v m(S[] args){
    try {
      Sopln("Hi");}
    Catch(Excp e) { }
}
```
O/p: Hi → partially checked.

```
class Test {
  p s v m(S[] a){
    try {
      Sopln("Hi"); }
    catch(IOE e) { }
  }
}
```
↳ CE: ↳ fully checked

```
class Test {
  p s v m (S[]a){
    try {
      Sopln("Hi"); }
    catch(Int E e) { }
  }
}
```
↳ fully checked

```
class Test {
    p s v m(S[] a) {
      try {
          Sop(n("Hi"); }
      catch (Error e)
        { }
}
```

→ Unchecked

o/p: Hi

→ In our program, within the try block if there is no chance of raising a exception then we can't write catch block for that exception. but this rule is applicable only for fully-checked exceptions.

* Summary :-

① try → To maintain Risky code
② Catch → To maintain exception handling code
③ finally → To maintain cleanup code
④ throw → To hand-over our created exception object to the JVM manually
⑤ throws → To delegate responsibility of exception handling to the caller.

# #77 EH Customized Exceptions.

19/Apr/23

*various possible CE 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) incompatible types
   found: Test
   required: Java.lang.Throwable
6) try without catch or finally
7) Catch without try
8) finally without try

*Customized (or) User-defined Exceptions :-

→ Sometimes to meet programming requirements we can define our own exceptions, such type of exceptions are called customize (or) user defined exceptions.
eg :- TooYoungException, TooOldException etc.

→
```
class TooYoungException extends RuntimeException
{   TooYoungException (String s) {
        super(s);
    }
}
```
→ To make description available to Default Excp^n. handler.

Defining customized exception.

```
> class TooOldException custExceptDemo {
      p s v m (S[] args) {
          int age = Integer.parseInt(args[0]);
          if (age>60) {
              throw new TooYoungException("Plz wait
          }       Some more time, you will get best match")
          else if (age<18) {
              throw new TooOldException("Your age crossed")
          }else {
              Sop("You will get match");
          }
```

Note:--> Throw keyword is best suitable for userdefine or customized exception but not for pre-defined exceptions.

→ It is highly recommended to define customized exception as unchecked i.e. we have to extends runtime exception but not exception.

→ super(s) ⇒ To make description available to Default Exception Handler.

# 78 EH Top10 Exception

19/Apr/23

* Top-10 Exceptions :-

→ Based on the person who is raising an exception, all Exception are divided to 2 cat

JVM ← Exception → Programitic

* JVM Exceptions :-

The exceptions which are raised automatically by JVM whenever that particular event occurs.

eg - AE, NPE etc

* Programatic exceptions :-

The exceptions which are raised explicitly either by programmer or API developer to indicate that something goes wrong.

eg: TooOldException, IllegalArgumentException.

① ArrayIndexOutOfBondException

→ It the child class of RuntimeE. & it is un-checked

→ Raised automatically by JVM. whenever we are trying to Access Array element with out of range @index.

eg int [] x = new int [4];
Sop(x[10]); RE: AIOOBE

② NullPointerException (NPE)

→ It is the child class of RuntimeE. & it is un-checked

→ Raised automatically by JVM, whenever we are trying to perform any operation of null.

→ String s = null;
Sop(s.length()); RE: NPE

③ ClassCastException (CCE)

→ It is child of RuntimeE & it is un-checked.

→ Raised automatically by JVM, whenever we are trying to typecast Parent object to child typ

eg> String s = new String("durga");
Object o = (object)s;

> object o = new Object();
String s = (String) o; RE:CCE

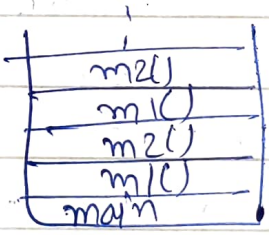
> object o = new String("durga");
String s = (String) o;

④ StackOverFlowError

→ It is child of Error & it is un-checked

→ Raised by JVM, whenever we are trying to perform recursive method call.

> class Test {
p s v m1(){
m2(); }
p s v m2(){
m1(); }
p s v m(s[]args){
m1(); }
}




RE:SOFE

⑤ NoClassDefFoundError

→ It is child of Error & it is un-checked.

→ Raised by JVM, whenever ~~we~~ ~~are~~ JVM unable to find required .class file.

Java TestTest ↵ RE: NCDFE

⑥ NumberFormatException

→ It is child of ~~RuntimeE~~ IAE & it is un-checked.

→ It is raised by programmer @ API developer to indicate that we are trying to convert string to number & string is not properly formatted.

> ~~int~~ i = Integer.parseInt("10");

X int i = Integer.parseInt("ten"); RE: NFE

⑨ IllegalStateException

→ It is child of RuntimeE & it is un-checked.

→ It is raised by programmer @ API developer to indicate that a method has been invoked at wrong time.

→ After starting of a thread we are not allow to re-start the same thread once again.

> Thread t = new Thread();

t.start(); ✓

t.start(); X RE: ISE

⑩ AssertionError

→ It is child of Error & it is un-checked.

→ It is raised by programmer @ API Developer to indicate that assert ~~or~~statement fails.

> assert (x>10); RE: AE

→ if x is not greater than 10 then we will get RE: AssertionError.

# #79 EH try & multi-catch block

19|Apr|23

⑥ ExceptionInInitializerError

→ It is child of Error & it is un-checked.

→ Raised by JVM, if any exception occurs while executing static var. assignment (or) static blocks.

> class Test {
    static int x = 10/0; RE: EIIE
  }

> class Test {
    static {
      String s = null; RE: EIIE
      Sop(s.length()); }
  }

⑦ IllegalArgumentException

→ It is child of RuntimeE & it is un-checked.

→ It is raised by programmer (or) API developer. to indicate that a method has been invoked with illegal argument.

> eg:- The valid range of thread priorities is 1-to-10 if we try to set the priority with any other value then we will ge RE: IAE

> Thread t = new Thread();
  t.setPriority(7); ✓
  t.setPriority(15); RE: IAE

⑧ Number Format Exception

→ It is child of ~~Run~~ IAE & it is un-checked.

→ It is raised by programmer @ API developer to indicate that we are trying to convert string to number & string is not properly formatted.

> ~~Int~~ i = Integer.parseInt("10");

X int i = Integer.parseInt("ten"); RE: NFE

⑨ Illegal State Exception

→ It is child of RuntimeE & it is un-checked.

→ It is raised by programmer @ API developer to indicate that a method has been invoked at wrong time.

→ After starting of a thread we are not allow to re-start the same thread once again.

> Thread t = new Thred();

t.start(); ✓

t.start(); X RE: ISE

⑩ Assertion Error

→ It is child of Error & it is un-checked.

→ It is raised by programmer @ API Developer t indicate that assert ~~no~~statement fails.

> assert (x>10); RE: AE

→ if x is not greater than 10 the we will get RE: Assertion Error.

## * 1.7 V Enhancements w.r.t. Exception Handling.

① try with resources
② multi-catch block

1.6V

```
BR br = null;
try {
    br = new BR(new FR("input.txt"));
    // use br based on our req.   }
catch(IOE e){
    // Handling code }
finally {
    if(br != null){
        br.close(); }
}
```

1.7V

```
try(BR br = new BR(new FR("input.txt"))){
    // use br based on our req.   } → Resources.
catch(IOE e){
    // Handling code
}
```

→ br will be closed automatically once control reaches end of try block either, normally or abnormally & we are not responsible to close explicitly.

⇒ Conclusion :-

→ We can declare multiple resources but these resources should be seperated with ";".

> try (R1; R2; R3) {
=
}

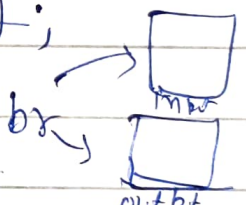
→ All resources should be auto-closable resources.

→ A resource said to be auto-closable iff corresponding class implement AutoCloseable interface.

→ All IO related, DataBase, and network related resources are already implemented AutoCloseable

→ AutoCloseable interface came in 1.7v & it only contain close() method.

> try (BR br = new BR(new FR("input.txt"))) {
final
br = new BR(new FR("output.txt"));
FileReader("output.txt"));
}




→ All resource reference variables are implicitly final & hence within the try block we can't perform re-assignment.

> try { } finally { }
try (R) { = }

→ Untill 1.6v, try should be associated with either catch or finally. but 1.7v onwards we can only try with resoure without catch & finally.

→ The main adv. of try with resourse is we not reaire to write finally block explisitly

→ finally become Hero to zero.

## * Multi-catch block :-

```
> try { }
  Catch (AE e) {
      e.printStackTrace(); }
  Catch (IOE e) {
      e.printStackTrace();
  }Catch (NPE e) {
      Sop ln (e.getMessage());
  }
```

```
> try { }
  Catch (AE | IOE e) {
      e.printStackTrace();
  }catch (NPE | IE e) {
      Sop ln (e.getMessage());
  }
```

→ untill 1.6 V, eventhough multiple different exception having same handling for every exception type we have to write a seperate catch block it increases length of the code & reduces readibility.

⇒ Alc to multi-catch block, we can write a single catch block that can handle multiple different type of exceptions.

→ Adv., length of code is reduced & readibility improv

> try { }
Catch (AE | Exception e) {
e.printStackTrace(); CE:
}

→ In multi catch block there should not be any relation between exception types (either Child to parent or parent to child or same type).

* Exception propagation :-

> m1() {
m2();
}

m2() {
Sop(10/0);
}

→ Inside a method, if a exception raised & if we are not handling that exception then exception object will be propagated to caller then caller method is responsible to handle exception.

> ~~try~~ {
Sop(10/0);

* re-throwing Exception :-

→ we can use this approach to convert one exception to another exception type.

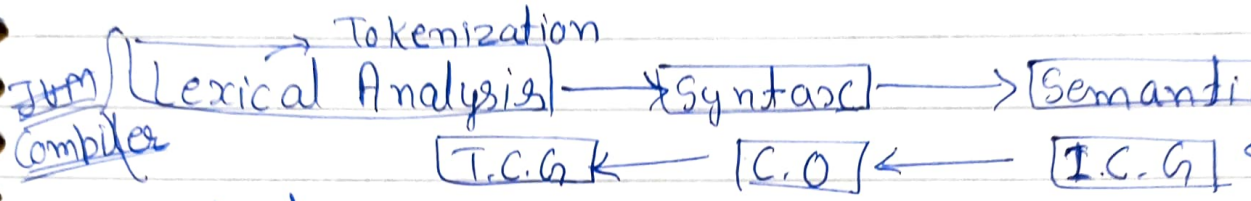
> try {
Sop(10/0); }
Catch(AE e) {
throw new NPE();
}

# #133 Regular Expressions

29/Apr/23

## * Regular Expression

→ If we want to represent a group of strings according to a particular pattern, then we should go for Regular Expression.
eg :- To represent mobile no, emailIds etc.




→ Application area.
1) To develope validations framework.
2) To develope pattern matching applications (Ctrl F)
3) To develope translators {Assemblers, compiler, Interpr
4) To develope digital circuit
5) To develope communication protocals (TCP/IP)(UDP)

```
> import java.util.regex.*;
> class RegExDemo {
      p s v m(S[] args){
          Pattern p = Pattern.compile("ab");
          Matcher m = p.matcher("abbabbba);
          while (m.find()) {
              Count++;
              Sopln(m.start()); {m.end(), m.group(
          Sopln("Total occ" + Count);  }
  }
```

(ab)
a bb a bbb a
↑ 0  ↑ 3

→ A Pattern object is compiled version of regu
-ar expression.

# #133 Regular Expression

29/Apr/23

→ We can use create pattern obj. by using compile() method of pattern class.

> public static Pattern compile(String re)

→ We use Matcher object to check given pattern in the target string.

→ we can create a matcher obj. by using matcher() method of Pattern class.

> public Matcher matcher(String target)

* Imp. methods.

1) boolean find() ⇒ To find next match
2) int start() ⇒ return start index of match
3) int end() ⇒ return end+1 index of match.
4) String group() ⇒ return matched pattern.

Note :- Pattern & Matcher classes ⇒ 1.4V

* character matching

[abc] ⇒ either 'a' or 'b' or 'c'
[^abc] ⇒ Except 'a' or 'b' or 'c'
[a-z] ⇒ Any Lower case alpabet symbol from a to z
[A-Z] ⇒ Any Upper case alpa. symbol from A to Z.
[a-z A-Z] ⇒ any alphabet symbol
[0-9] ⇒ Any digit from 0 to 9
[0-9 a-z A-Z] ⇒ Any alphanumeric symbol.
[^0-9 a-z A-Z] ⇒ Except alphanumeric symbol
i.e. special symbol

> Patter p = Pattern compile("x");
> Matcher m = p.matcher("a3b#k@9z");
while(m.find()){
SoPln(m.start() +"---" + m.group());

| x=[abc] | x=[^abc] | x=[a-z] | x=[0-9] | x=[a-z A-Z 0-9] | x=[^a-z A-Z 0-9] |
|---|---|---|---|---|---|
| 0 -- a | 1 -- 3 | 0 - a | 1 - 3 | a - a | 3 - # |
| 2 -- b | 3 -- # | 2 - b | 6 - 9 | 1 - 3 | 5 - @ |
| | 4 -- k | 4 - k | | 2 - b | |
| | 5 -- @ | 7 - z | | 4 - k | |
| | 6 -- 9 | | | 6 - 9 | |
| | 7 -- z | | | 7 - z | |

pre-defined
* ~~Space~~ character classes
\s ⇒ space character
\S ⇒ Except space character
\d ⇒ Any digit from 0 to 9 [0-9]
\D ⇒ Except digit, any character
\w ⇒ Any word character [0-9a-z A-Z]
\W ⇒ Except word character [special char]
. ⇒ Any character

>
> Target S = "a7b k@9z";

| x=\s | x=\S | x=\d | x=\D | x=\w | x=\W | x=. |
|---|---|---|---|---|---|---|
| 3 --- | 0 - a | 1 - 7 | 0 - 9 | 0 - 9 | 3 - | 0 - 9 |
| | 1 - 7 | 6 - 9 | 2 - b | 1 - 7 | 5 - @ | 1 - 7 |
| | 2 - b | | 3 - | 2 - b | | 2 - b |
| | 4 - k | | 4 - k | 4 - k | | 3 - |
| | 5 - @ | | 5 - @ | 6 - 9 | | 4 - k |
| | 6 - 9 | | 7 - z | 7 - z | | 5 - @ |
| | 7 z | | | | | 6 - 9 |
| | | | | | | 7 - z |

# String class split()

```
String s = "Durga Soft Sol";
String[] s1 = s.split("18");
for (String s2 : s1) {
    Sop1n(s2);
}
```

o/p: Durga

String class also contain split method to split the target string A/c to a particular pattern.

Note:- Pattern class split() method can take target string as argument whereas String class split() method can take pattern as argument.

## String tokenizer

- It is designed for tokenization activity
- present in java.util pkg.
- StringTokenizer st = new StringTokenizer("Durga Soft Sol");
- while (st.hasMoreTokens()) {
      Sop1n(st.nextToken());
  }

o/p: Durga
Soft
Sol.

→ Default regex for it is space.

eg:- StringTokenizer st = new StringTokenizer("19-09-2014", "-");

target string / RE/Pattern Delimeter

## * Quantifiers

a ⇒ Exactly one 'a'
a+ ⇒ Atleast one 'a'
a* ⇒ Any no of a's including zero numbe
a? ⇒ Atmost one 'a'

→ we use quantifier to specify no. of occurances to match.

> target = "abaabaaab";

| x = a | x = a+ | x = a* | x = a? |
|---|---|---|---|
| 0 - a | 0 - a | 0 - a | 0 - a |
| 2 - a | 2 - aa | 1 - | 1 - |
| 3 - a | 5 - aaa | 2 - aa | 2 - a |
| 5 = a | | 4 - | 3 - a |
| 6 = a | | 5 = aaa | 4 - |
| 7 ~~a~~ | | 8 - | 5 - a |
| | | 9 - | 6 - a |
| | | | 7 - a |
| | | | 8 - |
| | | | 9 - |

# 134 Regular Expression 29/04/23

## * pattern class Split()

> Pattern p = Pattern.compile("118");
String[] s = p.split("Durga Soft Sol");
> for(String s1 : s){
Sopln(s1);
}

Durga
Soft
Sol

→ We can use pattern class split() method to split the target string A/c to a particular pattern.

p = pattern.compile("11."); ⓐ [.]

## * String class split()

```
String s = "Durga Soft Sol";
String[] s1 = s.split("118");
for (String s2 : s1) {
    Sopln(s2);
}
```

o/p: Durga

→ String class also contain split method to split the target string A/c to a particular pattern.

Note:- Pattern class split() method can take target string as argument whereas

→ String class split() method can take pattern as argument.

## * String tokenizer

→ It is designed for tokenization activity

→ present in java.util pkg.

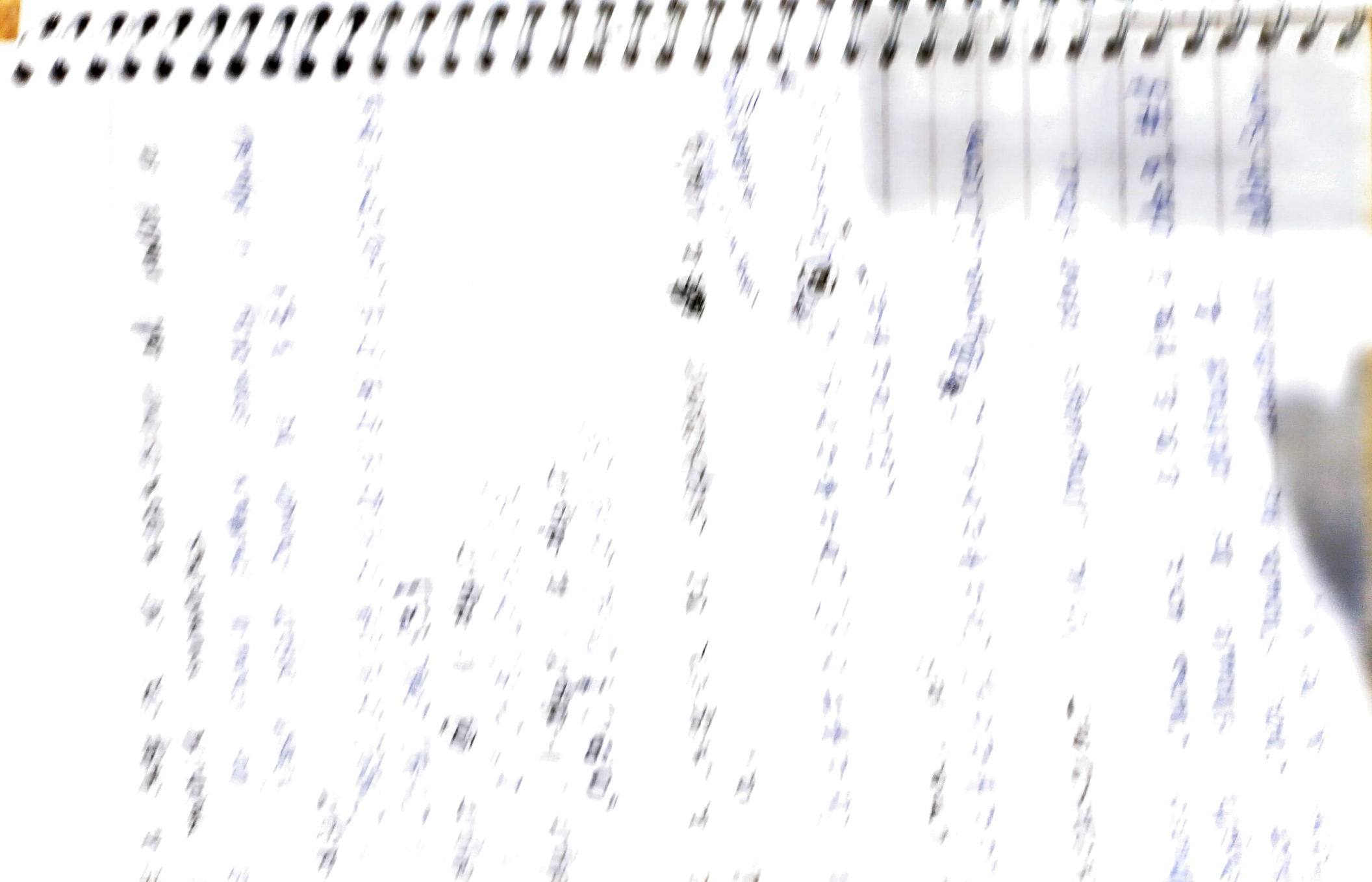
```
StringTokenizer st = new StringTokenizer("Durga Soft Sol");
while (st.hasMoreTokens()) {
    Sopln(st.nextToken());
}
```

o/p: Durga
Soft
Sol.

→ Default regex for it is space.

eg:- `StringTokenizer st = new StringTokenizer("19-09-2017", "-");`

target string / RE / Pattern / Delimeter



Q Write a reg-ex to represent all valid 10 digit mobile numbers.

Ans:- Rules :- no. should contain exactly 10 digits
first digit should be 7, 8, 9.

⇒ 10-digit

> S = [789] [0-9] [0-9] [0-9] [0-9] [0-9] [0-9] [0-9] [0-9] [0-9]

> S = [7-9] [0-9] {9}

⇒ 10-digit (or) 11-digit

> S = 0? [7-9] [0-9] {9}

⇒ 10-digit | 11-digit | 12-digit

> S = (0/91)? [7-9] [0-9] {9}

Q Write a reg-ex to represent a valid email id:

Ans:-

any no. of times at least once

S = [a-z A-Z 0-9] [a-z A-Z 0-9 _ .]* @ [a-z A-Z 0-9]+ ([.][a-z A-Z]+)+

only gmail id's

S = [a-z A-Z 0-9] [a-z A-Z 0-9 _ .]* @gmail[.]com

Q:- To represent Java language identifiers:

Rules:

1) allowed chars are a to z, A to Z, 0 to 9, #, $
2) length should be atleast 2.
3) first char. should be lowercase alph-sym from a to k.

4) 2nd char. should be a digit divisible by 3. (0, 3, 6, 9)

> S = [a-k][0369][a-z A-Z 0-9 #$]*

135

# Regular Expresions

30-4-23

* How to use reg-ex.

Q:- W.A.P to check wether the given no. is valid mobile no. or not.

Ans:-

```
> class RegExDemo {
      p s v m (S[] args) {
        Pattern p = Pattern.compile("(0|91)[7-9][0-9]{9}");
        Matcher m = p.matcher(args[0]);
        if (m.find() && m.group().equals(args[0]))
            Sop("valid");
        }else{
            Sop("invalid");
        }
```

check total thing also.

~~9848022308~~

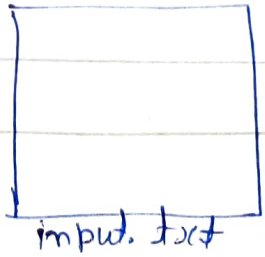
~~9809844898480~~

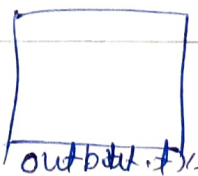
X 9299929292929292

~~9193938 7325~~

=> for mail validation just chang reg-ex.

Q W.A.P to read all mobile no.s present in given input file. where mobile nos are mixed with text data.



```
> PrintWriter pw = new PW("Output.txt");
  Pattern p = Pattern.compile("(0|91)?[7-9][0-9]{9}");
  BufferReader br=new BR(new FR("input.txt");
  String line = br.readline();
  while(line != null){
      Matcher m = p.matcher(line);
      while(m.find()){
          pw.println(m.group());
      }
      line = br.readline();
  }
  pw.flush();
}
  close()
```

Q:- W.A.P. to extract all mail Id. for input file.

Ans:- change reg-ex of above program.

Q:- W.A.p to display all .txt file name present C:\\ durga_classes.

Ans:-

```
> regEx = [a-zA-Z0-9_.$]*[.]txt
> pattern p = Pattern.compile(regEx);
  File f = new File("C:\\durga_classes");
  String[] s = f.list();
  for(String s1:s){
      Matcher m = p.matcher(s1);
```

abc.txt ✓

123abc.txt ✓

abc.txt.back ✗

```
if (m.find() && m.group().equals(s1)) {
        count++;
        System.out.println(s1);
    }
}
System.out.println("Total files:" + count);
```