#Exception handling

⊙ Exceptions are runtime errors and with help of exception handling we can handle runtime errors

⇒ Runtime errors are one which get invoked when loading of .class or .exe file is taking place in memory

⇒

#Types of error in programming

[A) Syntax error] → Compile time errors (syntax of lang. is not followed)

B) logical errors
C) runtime errors (Exceptions)] → runtime error

⇒ Errors which we get while compiling java file those type of errors are compile time errors

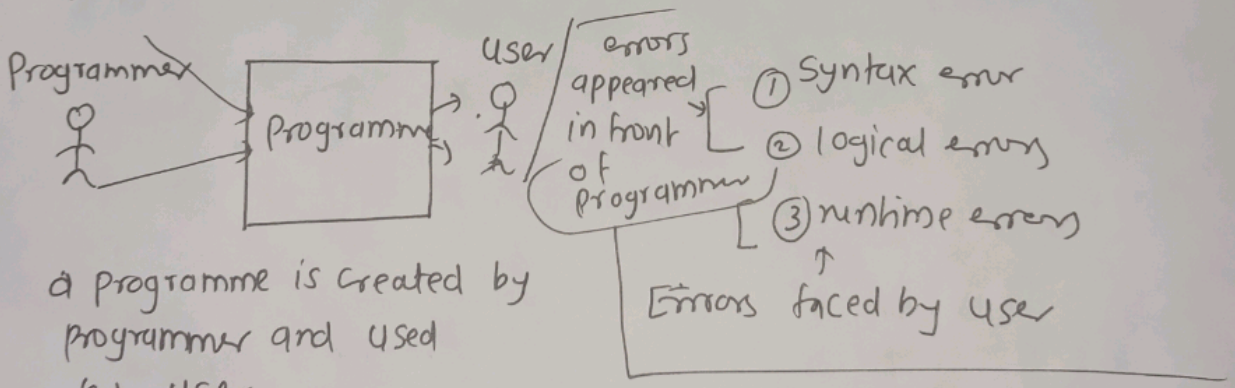
It mostly contains syntax errors or semantic errors

⇒ compiler is not enabled to find out runtime & logical errors

Ⓑ logical errors

this types of errors one in which we not get expected results which we want to get

(Programme runs but not give expected results)



a programme is created by programmer and used by user

⇒ logical errors takes lot of time to resolve & are dangerous.

⇒ Debugger can be helpfull to solve logical errors and you can visualize working of programme

Ⓒ Runtime errors

such errors are faced by user by not handing programmer.

in correct manner

⇒ user can't be able to solve errors so programmer need to solve it.

Ex:→ ① Programme ask for age and user provide -10

② Programme need a file to execute but user deleted it

③ some programme need internet connection but internet is not available in user's end

④ Calculator programmer is there & while division user give 0 as denominator

⑤ you can find $\log n$ and user send $n=0$

① Bad input
② Resources unavailable

Such errors can be there in which ~~user~~ due to user's missuse error will occurs

⇒ such errors are called as runtime-errors or exception

⊕ Examples of · ~~compiler~~ syntax and ~~semantic~~ logical errors

Ⓐ int a,b, → missing character

Ⓑ x=10 → missing semicolon

Ⓒ int z = x + y;
↑ ↑
uninitialised variable provided

ⓓ Print("Hello');

⇒ " expected at end

Such are ~~(also)~~ syntax error mainly occurs on compile time

malfunction of code occurs in logical errors

$r = \frac{-b}{2a}$ $\quad r = (1 \times b) / 2 * a$
error

this will solve but not give expected value

```
int A[ ] = new int[10];
for (int z = 1; z < A.length; z++){
    S.O.P(A[z]);
}
```

• Such like that we can be able to find out the errors inside java code

⊖ Back to runtime errors

In order to solve runtime errors (problems due to user) we ~~(also)~~ don't have to do modification in code for that we can use exception handling

→ Runtime errors will result in crash of code

→ that is we can guide user through ~~(also)~~ providing information to them about what happened

⇒ For exception handling we use it to handle runtime error this ~~exception~~ exception handling can avoid exceptional cases

Exception handling

```
Class Test {
    P.S.V.m(){
    int a,b,c;
    a=19;
     b=0; → Intention
    c=a/b  ⊟ Breaking point
S.O.p("Division:"c");
So.p("Programme ends.\n Bye!");
    }
```

as denominator is 0 code will break warning ~~or~~ will be printed

⇒ following lines will not going to execute

⇒ In code to do this we use try catch block where in try block we can ~~can~~ write code which can go lead to exception and in catch block it take type of exception and can also print or do something in catch block

Syntax

```
try {
Code which can lead
 to exception;
}
catch (TypeofExcephion (e)variable ){
    Code to execute if we
    tackle with some runtime
    errors }
}
```

e: It is object of excephion mean e has value of object which caused excephion

multiple catch cases or nested try & catch

so you are in condition where you have chances of code going throught multiple exceptions so you can able to use such statement where you have single try block & for multiple catch block each having different exception

⇒ you also can use ~~fast~~ nested try & catch in order to deal with multiple exceptions

⊕ Example

Code is ⇒

```
int A[] = {10, 0, 8, 9, 7}
int r;
r = A[0] / A[i];
```

→ Here arithematic Exception divide by zero can be possible

```
S.O.P.(r);
S.O.P(A[10]);
```

Here Array index out of bound exception is possible ⇒

So here two types of exceptions can be possible

→ you can deal with above with nested try catch or multiple catch block

```
try {
int A[] = {10, 0, 8, 9, 7};
int r;
r = A[0] / A[1];
S.O.P(r);
S.O.P(A[10]);
}
catch (ArithematicException e){
    S.O.P(e);
}
catch(ArrayIndexOutOfBound e){
    S.O.P(e)
}
```

Code with multiple catch and each catch deals with diffrent exception/ runtime error

```
try {
    int A[] = {10, 0, 8, 9, 7};
    int r;
    try {
        r = A[0] / A[1];
    }
    catch (ArithematicException e){
        S.O.P(e);
    }
    S.O.P(A[10]);
}
catch(ArrayIndexOutOfBound e){
    S.O.P(e)
}
```

nested can be more useful when you know when exception can happen.

Both the nested & multiple catch block are useful

Note
⇒ If you have multiple catch block with multiple possible exceptions so at such moments must write exception in ascending order i.e. subclass exception catch block first & then superclass/Parent class catch block

⇒ this note because parent can shadow subclass if written in up the order

try { } catch(1){} catch(2){} catch(3){} Ex:

so if try got exception, it first check in first catch block, then second and then so on

finally block

```
try {
} catch() {
  ____
} catch() {
  ____
}
finally { -> finally block is used for clean up process
  -11-
}
```

⇒ finally block is written at last and it is optional to write

⇒ the code inside finally will run always even if exception is there in code or not.

⇒ main goal of exception handling is that except exception code other code should run intact

⇒ nested try catch are more functionally useful than that of multiple catch block

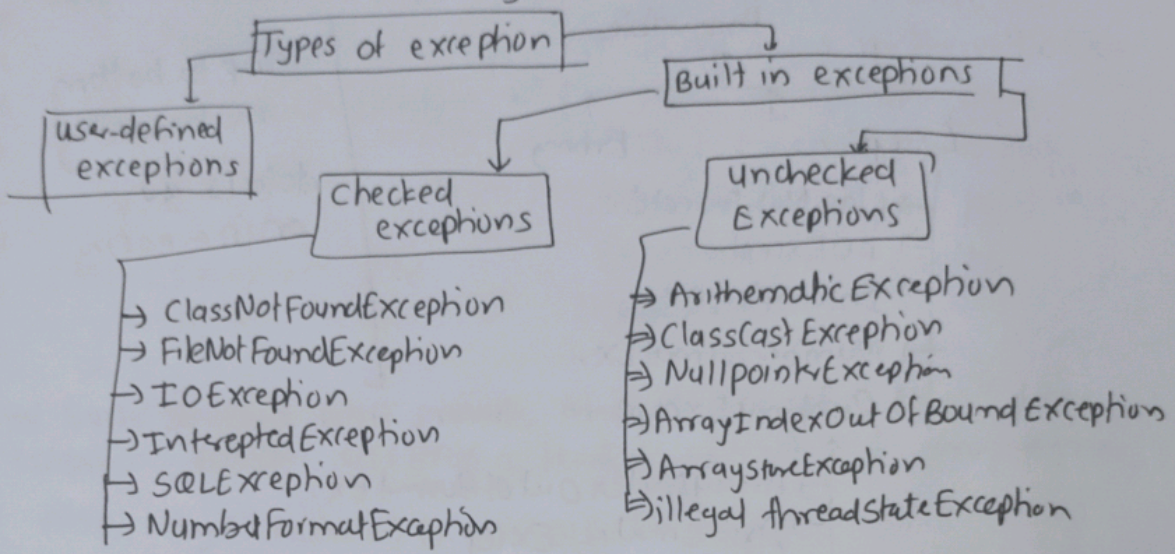
⇒ o in try block code upto exception will run code after that will not executed but with nested try catch we can tackle it.

⇒ Exception handling is trying to avoid crashing of code

⇒ only write problematic code inside try block

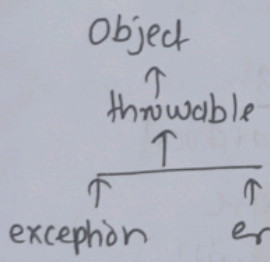
Built-in Exception classes In java

In java you can able to create user defined exceptions along with it java also provide some built in exceptions ten lets learn them with diagram



⇒ Checked exceptions are one for which you have to write code of try catch block (if possibility is there you must handle it).

⇒ Unchecked exceptions are one for which we don't have any mandatory need to ~~write~~ do exception handling or not



⇒ In java errors are handled by JVM architecture you can't do above error you must have to resolve them

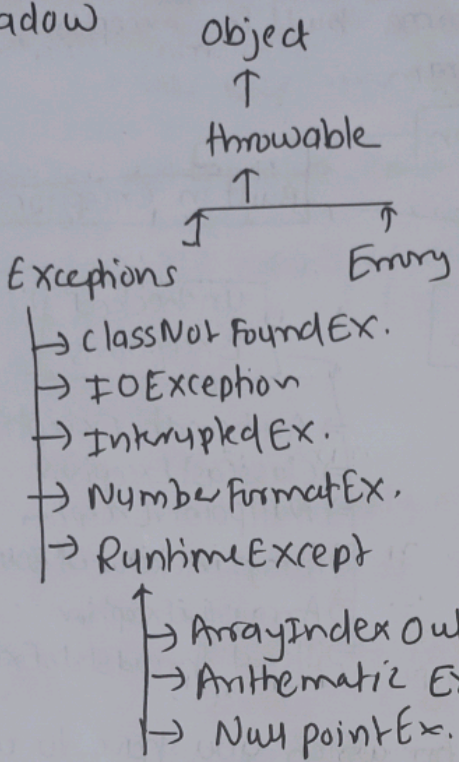
⇒ exception can divided as above, exception can be handled but we have Checked (manddatory EH) or unchecked (non-mandatory EH)

⊛ lets go back to multiple catch so in those case you always have to observation about are you following heiarchy or not

⇒ So the exception which is present in upper heiarchy can consist of all belowed exception type so ~~must~~ must write catch argument in ascending order

i.e ~~for~~ from lower level heirarchy to upper level

→ If you not do so one at upper level heirarchy will overshadow



Top to bottom in heirarchy details go on increasing

Ex

```
try {
   —— 11 ——
}
Catch (Exception  p) {
   —— 11 ——
{
catch (arithematic  g)
        Ex.
{  —— 11 ——
}
}
Catch ( NullPointer Ex  N) }
   —— 11 ——
{
Catch (ArrayIndexOutOfBoundEx A)
   }
   —— 11 ——
} }
```

Invalid

So now exception can handle all runtime errors

So even though lowe level heirarchical exception are there they will never be called.

So what can we do

this is also not valid so compiler throw error for this

```
try {
   --11--;
   }
catch (arithematicExp a) {
    - 11 --;
} catch (NullpointerException N){
    --11--;
   }
catch (Exception p){
    --11--
}
```

⇒ valid & accurate

⇒ heiarchy in exception

So now try check call catch in order they written so it will first check about arithematicExp. then Nullpointer and That exceptions are not happende upper level Exception can easily handle it

⇒ Such structure will provide more precise info about the exception which occured & lead to crash of programme

⇒ ~~Also~~ try{} catch(A){} catch B(){} catch C(){}

⇒ so in such structure A should be on same level with B or (B) is should be superclass of (A) & (A) is subclass of (B) Similarly (C) should be at same level with (B) or (C) is superclass of (B) or 'B' is subclass of (C)

Built in Exception class in java

It has few important functions and they are

(A) String getmessage();

this function will return message in string format for object of class Excephon

(B) String toString();

this also return string for exception msg

S.O.P (object.getMessage());

System.out.println(object); ← to print Exception Msg

⇒ toString() by default function inside sy.stem.out.println

⇒ SOP uses toString method

③ void printStackTrace();

⇒ this function will ~~return~~ print order in which functions are called and can be found helpful while handling Exception

→ tracing of method calls can be used.

object.printStackTrace(); → It will print by itself.

#How can you create own exception

In java you can create own exception with using a class and extending built in Exception class

→ If you not inherit exception then created class is not a exception

and to provide message in your language you can override previous function.

i.e toString(); getMessage() or printStackTrace();

```
class minimumBalanceException extends Exception {
    public String toString() {
        return "minimum balance should be 13000";
    }
}
```

} ⇒ Such like that you can use & create own user defined exception

⇒ for unchecked exception programme will not give an error

④ A file not found exception

⊙ import java.io.*; (a checked exception)

and FileInputStream a = new FileInputStream();

Such like that you can able to create a file not found exception

⇒ So here you have to remove it so that your programme will execute for sure

Users-defined exception in details

⊕ Exception propagation

```
Void meth2(int a, int b){
int c = a/b
return c;
}

Voidmeth1(){
int z = meth2(10,0);
S.o.print("meth1");
}

main(){
meth1();
}
```

See exceptPropagation.java code

So where we can handle exception so lets see function calls first

main() ⇒ meth1() ⇒ meth2();

So exception arise first at meth2(); so we can handle it here itself else it will propagate

Exception Propagation ⇒ meth2() → meth1() → main()

→ Exception propagates

⇒ So you can handle except in meth2 itself or can be in meth1() or at last in main() & if that doesn't happen code will throw err

⇒ such exception propagation can be seen with printStackTrace();

⇒ this also proves that except propagation exists and will take place in reverse order that of function call propagations.

How you can define and create own exceptions with help of throw and throws.

⊕ Example you are working on rectangle class so it is geometry & say length is not should not be negative so you have to create exception that length should not inputed as negative

So In function throwing exception = return value

You can only throw ⊕ Exception nothing else

⇒ So if you have a function which throw exception you must have something to handle it (try catch block) should be there

⇒ And function which throw exception should declared with throws Exception; ~~keyword~~;

In order to remind one that the function you want to call (throws) Exception so you should have mechanism to handle it.

⇒ which makes mandatory to handle exception in caller function

```
Ext
double area(int l, int b) throws Excepti
  if (l<=0 || b<=0)
      throw new
      Exception("negative
      ~~length~~
      distance
      provided");
  else {
  return l*b;
}
```

```
int method(a,b)
try {
double = area(a,b);
}
catch(Exception z){
s.o.p (z);
}
```

throws came with a function which throw Exception throw present inside body

method ⇒ caller function
area ⇒ called function

called function which throws exception and caller function has mandatory to handle it

Note ⇓ Useful

```
Exception z = new Exception("Here you can provide
                some message with exception object");
```

you can also use caller function as handle exception so it can't also able to throw exception to it's caller function if it has any

(*) If a function don't want to handle Exception it can able to throw it further

⇒ Also while throwing except you can throw user defined exception

```
// It is a user defined exception
class ghanshamExcephion extends Exception {
    public String getMessage() {
        return "ghansham Exception" + getmessage();
    }
    public string toString() {
        return "ghansham Exception from to string";
    }
}
```

```
main() {
try {
 meth1();
}
catch (ghanshamException z) {
println(z);
}
```

```
void meth1() throws ghanshamException {
    try {
    double area(-10,5)
    println(area)
    }
    catch (ghanshamExp. z)
    println(z);
}
```

```
double area(int a, int b)
throws ghanshamException {
if (a ≤ 0 or b ≤ 0)
    throw
    new ghansham Exp.();
else {
    return a*b;
}
```

→ Function calls ⇒ main() ⇒ meth1() → area()

area throw ghanshamException meth1() will not handle so it also contains throws exception in function signature & throw exception to main()

Such passing ~~throw~~ of Exception can acts like propagation of Exceptions inside java.

```
funchon() throws BlahBlahBlah;
```

this should be type of excephin your funchon Throwing

finally keyword.

this will written after try catch block and will execute even though exception occurs or not

```
try {
   code
}
catch (Except z):
  S.O.P(z)
}
finally {
S.O.p.("final msg")
}
```

Case 1) Exception occurs

message from z

output: final msg

Case 2) Exception not occurring

~~message from~~

output: final msg

⇒ you can only write try and finally block also and to handle catch is must.

⇒ finally is not compulsory

⇒ It can be used for ~~from~~ closing resources

So if you want print msg after thrown you can use finally

```
try {
    throw new Exception();
} finally {
    code
}
```

Try with Resources

Heap is not part object so it is resource

⇒ Java programme run inside stack session of memory all things other than it is a resource

Ex:- mouse, database, IO devices, CPU all are resources

⇒ All facility of program like taking input so it done with help of resources

→ So when programme need resources | take resource | when your work is done loose it or return it

⊛ As in java it has garbage manager which deletes all data from heap after his work is done

Ex:→ you need input scanner to input so take it.

Take resource) Scanner input = new Scanner(System.in);

Release the resource ⇒ input.close(); (one should not hold it unneccesarily)

⇒ If you still using mean you taken control & not returned it so programme thinks that user holding resources is still working & this is bad for system performance.

Use of finally

Ex: you get some resources and got some exception before closing it.

```
int meth() throw Exception {
    FileReader f;
    try throw
```

⇒ Due to try and error you forgot to handle and close resources at end so lines after Try will not execute so

it is good idea to put close blocks into finally

⇒ finally used to done cleanup

```
FileReader f;
try {
    f = new FileReader("my.txt");
    //use file
    ⇒ code which used
}
finally {
    f.close();
}
```

⇒ ~~Int~~ you can use try block itself for throwing exception and ~~witho~~ if you don't know exact line which can cause exception

⇒ finally help us keep a good programming practise to close resources after using