we mainly use 2-ways in debug-error:-

→ when we don't know how to the code executing.

→

→ We can create static method in interface.

→ implements → giving body to the method. we can't implement one interface in other interface.

→ In extern and implement we will give first priority to extern.

→ public static void main(String[] args) {

System.out.println("Hi");

try {

int res = 10/0;

S.O.P(res);

S.O.P("Hi"); → This will not execute

} Catch(Exception name any variable e) {

Arithematicexception ae) {

S.O.P("we can't divide any number with zero") }

}

→ o/p:- Hi

we can't divide any number with zero

→ If exception occurs in try, and exception name does not match with catch block, the JVM terminates prog. abruptly. And remaining lines present after catch block will not be executed further.

public static void main(String[] arg) {

S.O.P("Hi")

try {

int[] arr = {10,20};

S.O.P(

JVM will not enter into catch block.
→ If exception not occurs in try, then JVM will not enter into catch block.
→ And remaining lines after catch block will execute.

Properties:
→ If one catch block will execute other catch blocks will not execute.
→ If one exception occurs, or it came once out of try then other try block it will return not return into the try block.

```
try s.o.p("Hi");
    int a=10/0;        → Here this exception occur, then it will enter into catch block, other
    s.o.p(a);  (error)
    int[] a1={5,6,7,8}  → This statements not consider.
    s.o.p(a1[7]); (error)
    }
catch(Arithematic exception e) { don't have any
  s.o.p("divide by zero not possible"); } exceptions,
catch(ArrayIndexoutof bound e1) {   then only it
  s.o.p("Not possible");            occurs. or this
  }                                 exception considered
```

If we want execute this error, then, before that we

```
try {
s.o.p("Hi")
int a=10/5;
s.o.p(a);
  }
catch(AE e) {
  s.o.p("Enter the correct one");
  }
try{
 int[] a1={5,6,7,6};
s.o.p(a1[7]);
catch(AIOBE e) {
  s.o.p("");
  }
```

→ Here every try block will execute
→ we are allowed to write one or final block, to allowed try.
→ 1 or more catch blocks for given try.

Nested try

try
{
}
try {
}
catch(Exception ex) {
}
}
catch(Exception ex) {
}
finally {
}

→ we can place try-catch inside a try.
→ this is known as nested try.
→ mostly it is not required in very rare situations only we use.
→ Ily we can place try-catch inside catch also.

Case-I:-

```
try {
s.o.p("Hello");
try {
int a=5/0;
s.o.println(a); }
catch(ArithmaticException e) {
s.o.p("don't divide by 0"); }
int[] arr={10,20};
s.o.p(arr[3]);
}
catch(AIOBE e) {
s.o.p("Invalid Index");}
```

O/P:-
Hello
don't divide by zero
Invalid Index

②
```
try {
s.o.p("Hello");
try {
int a=5/0;
s.o.println(a); }
catch(AIOBE e) {
s.o.p("Index out of bond"); }
int[] arr={10,20};
s.o.p(arr[3]);
}
catch(AE e) {
s.o.p("try 0"); }
s.o.p("Bye");
```

O/P:- hello
try 0
Bye

→ Here if inner catch, not match with inner try exception, then it will go to outer catch it will match, then it will print.

```
try {
S.O.P("Hello");
 try {
   int a = 5/0;
   S.O.println(a);
 }
catch(AIOBE e) {
    S.O.P("Index out of bound");
 }
int arr = {10,20};
S.O.println(arr[3]); }
catch(IoException e) {
    S.O.P("File corrupted"); }
    S.O.P("Bye")
```

o/p:- Hello
Exception message
[error]

```
→ try {
  int a=5/0;
  S.O.P(a);
 try {
  int[] arr= {10,20};
  S.O.P(arr[2]); }
catch(ArithematicException e) {
    S.O.P("/ by 0"); }
 }
catch(AIOBE e) {
    S.O.P("Invalid Index"); }
      S.O.P("Bye")
```

o/p:- Exception message.

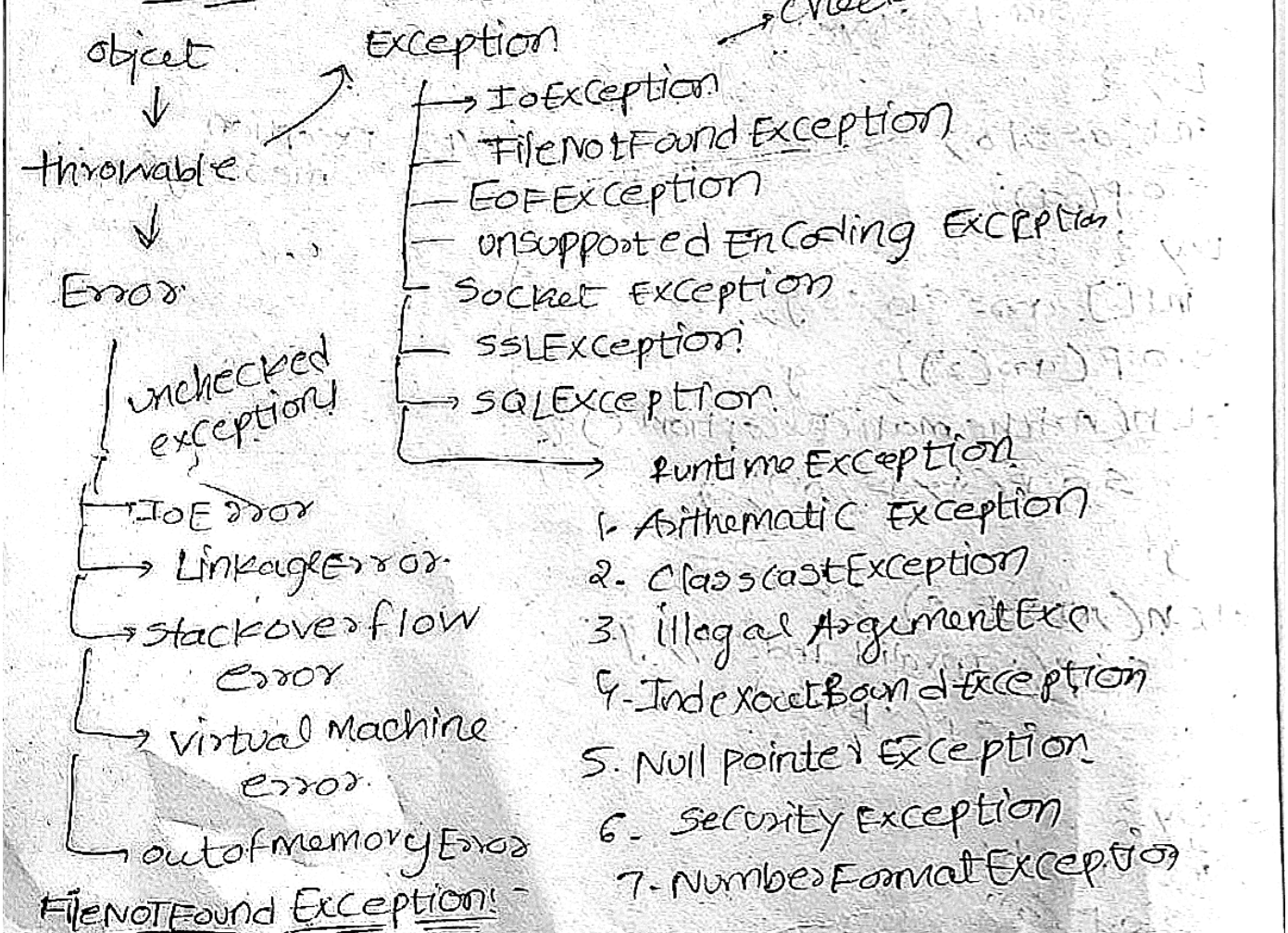
```
③ try {
  int a=5/0;
  S.O.P(a)
  try {
   int[] arr = {10,20};
   S.O.P(arr[2]);
  }
catch(ArithematicException e) {
    S.O.P("Invalid"); }
      S.O.P("Bye")
```

types of exceptions:

checked exception
↓
compile will throw error.
filenotfound exception
ioexception
Sql exception.
FilenotFoundexception
iostream

unchecked exceptions
↓
compile will not throw error.
Ex:- [2/0];
Indexoutofbound
nullpoint
outofmemoryerror.

Exception class hierarchy:



FileNotFound Exception:-

It comes under the ioexceptions and is a part of checked Exceptions
→ this means we have to handle these types of exceptions
→ if the file doesn't exist in the given path" then filenotfoundexception occurs.
→ to handle the checked exceptions we are having two ways, one is by using try-catch block and the other is by using throws clause

parameter
→ Java's ability to run on different platforms:-
→ Write once, Run Anywhere
→ Java virtual machine.
→ All input and output devices are connected to RAM.
→ programming involves giving instructions to computers to perform specific tasks, making it the process of communicating with computers.
→ component of the JVM responsible for executing bytecode is the JVM execution engine.
→ JVM execution egine engine uses both interpretation and compilation for executing bytecode.
→ The part of the JVM that directly executes Java bytecode is the interpreter.

It is responsible for directly executing java bytecode by reading and interpreting the instructions sequentially which is crucial for running Java applications.
→ Java class Libraries offer a wide range of functionalities to java applications, including everything from basic language support to complex utilites for data handling, network communication and graphical user interface creation.
→ Java uses camelcase for variables.
→ var can infer the type of a variable based on the assigned value,

```
public class Demo {
    public static void main (String[] args) {
    var a=10;
    var b;
    b=a++
    S.o.P(a);
    S.o.P(b);
    }
}
```

=> 11, 10

b=a++ assigns the current value of a (10) to b, then increments to 11.

```
var a=10;
var b;
b=a--;
S.o.P(a);
```

→ 9, 10,

b=a-- assigns the current value of a(b) to b, then

throws:-

when programmer don't know how to handle the exception then, he can throw the exception to the parent method (i.e calling method).

→ Always use after the method declaration.

→
```
class test {
    public void readTestFile(String path) throws
                    FileNotFoundException {
        FileReader fr=new
```

example for filenotfound exception:-

```
package com.techpalle;
import java.io.FileNotFoundException;
import java.io.FileReader;
public class EgForFileNotFound {
    public static void main(String[] args) {
        String path="C:\\java\\abc.txt";
        try {
            FileReader f=new FileReader(path); }
        Catch(FileNotFoundException e) {
            S.o.println("File does not exist");
        } }
```

Here it will enter catch block if the file doesn't exit.

→ Here exception is catching successfully.

example for throws clause:-

```
class CheckExep {
    public void readMe(String path) throws
                    FileNotFoundException {
        FileReader Fr = new FileReader(path); } }
public static void main(String[] args) {
    CheckExep e=new CheckExep();
    try {
        e.readMe("C:/Users/Admin/Desktop
                /read.txt");
    }
    Catch(FileNotFoundException e) {
        System.out.println("File Doesn't Exist");
        e.printStackTrace();
    }
}
```

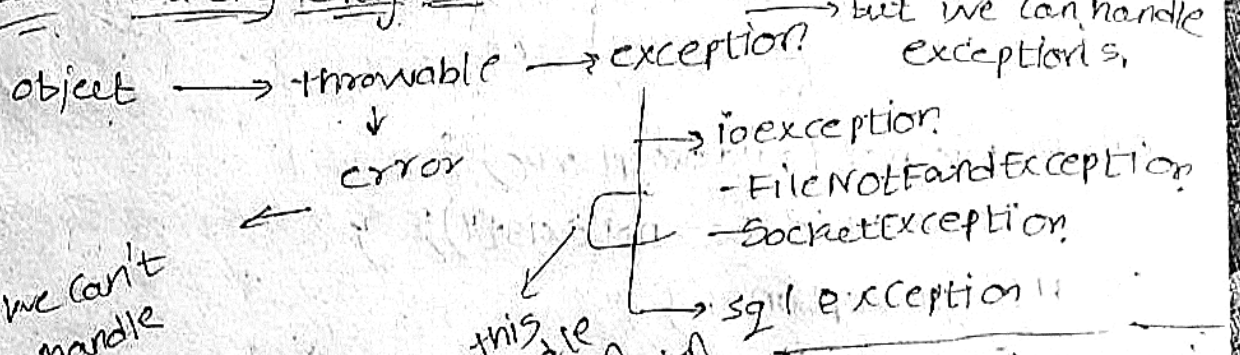
If you use throws clause in main(), your program will crash.

signature of the function, before starting the body after that we have to write that exception name.
→ if we are throwing multiple exceptions then we can specify them by comma.
→ in parent function we have to use try-catch block in the corresponding lines, i.e the lines that causing exception.

order of exceptions to handle:-

→ It always handle most specific exceptions first and generic exceptions last.
most child class first and most parent class last.

class hierarchy diagram:-



If we give generic exception, it will print same for other exceptions also.
If we use specific, it will mention specific

```
public class sample {
    public static void main(String[] args) {
        try {
            FileReader f = new FileReader("D://FavallException.txt");
            f.read();
        }
        catch (FileNotFoundException e) {
            S.O.P("File does not exist");
        }
        catch (IOException e) {
            S.O.P("File may be corrupted");
        }
    }
}
```

Finally block:- It is used to clean (or) close important system resource like.
* a. file connections

Properties of finally:

→ placed only after all catch blocks.
→ finally block will be definetely called by jvm whether an exception occur or doesn't occur.
→ try-finally also possible without catch block.
→ finally block will be executed even if programmer uses break/continue/return statements.
→ resolved:- [error]
↓ when we don't create a variable we will get this error.

```
try {
    f=new FileReader("D:\Java\Java\Exception.txt");
    f.read();
  }
catch(FileNotFoundException e) {
    S.O.P("File does not exist"); }
catch(IOException e) {
    S.O.P("File may be corrupted"); }
finally {
    try {
        f.close(); }
    catch(IOException e) {
        e.printStackTrace();
    }
}
```

Closing the File in try-block or catch block:-

```
try {
    f=new FileReader("D:\Java\Exception.txt");
    f.read();
    f.close();
  }
catch(File Not FoundException e) {
    S.O.P("File does not exist"); }
catch(IOException e) {
    S.O.P("File may be corrupted");
```