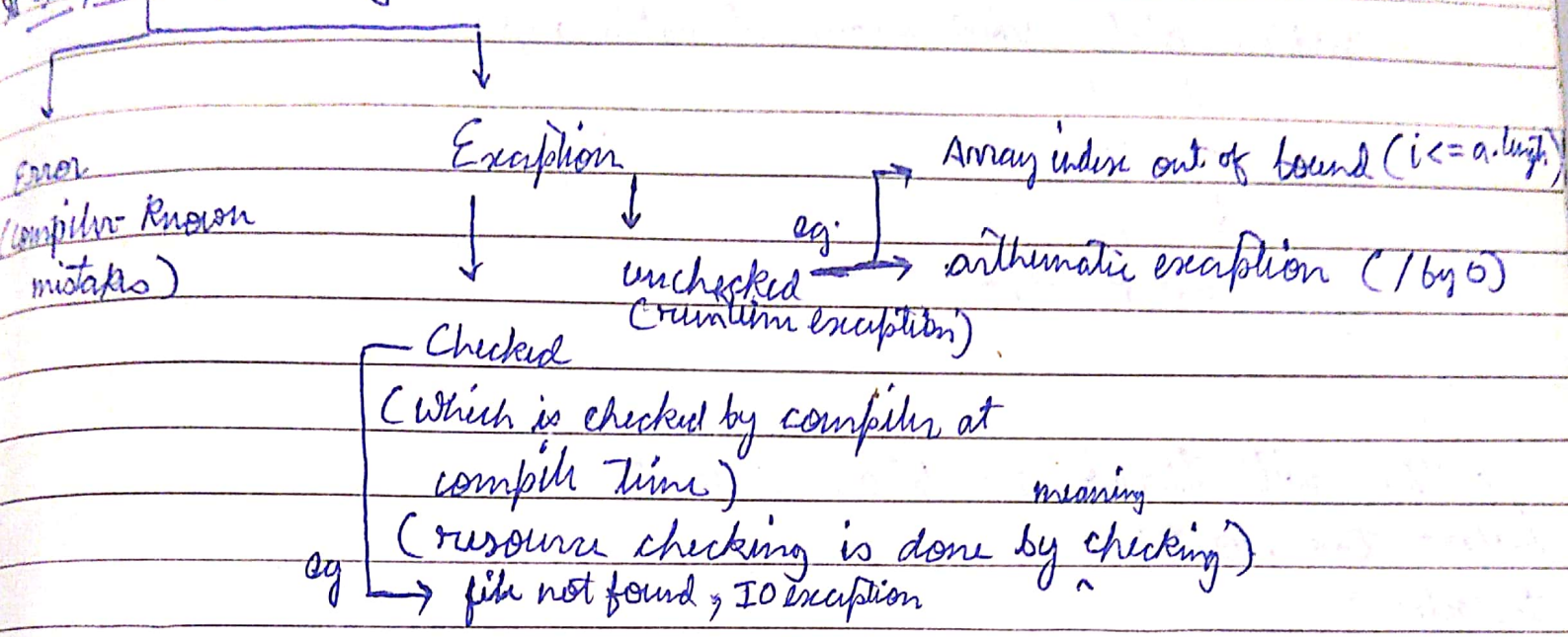
methods who throw exception must be declared in try catch block.
eg. = new FileInputStream("my.txt");
if throw file not found exception so declare it in try catch block.

Exception Handling:




```
try {
    // code in which exception may occur.
}
catch (Exception) {
      name          // multiple catch can be there for a single try
   // code to run when exception come.
   // catch can't be there without try
}
finally {   // only 1 finally with 1 try and finally not necessary
   // code to run exception occur or not.
   // finally can't be there without try.
}
```

* see ppt for all exception list.

Exception is parent class ∴ can handle any exception but don't use it as we need to maintain log to find out why which exception occured.

```
class expt {
    public static void main(String[] args) {
        int a,b;
        a = 10;
        b = 0;
        s.o.p(a/b);
    }
}
```

* this will throw exception at line 8

```
import java.io.*;
class expt {
    public static void main(String[] args) {
        int a,b;
        a = 10;
        b = 0;
        try {
            s.o.p(a/b);
        }
        catch(IOException ioe) {
            ioe.printStackTrace();
        }
    }
}
```

→ This will still generate exception as exception was arithmatic exception

→ Tells detail of exception on output screen.

```
class expt {
    public static void main(String[] args) {
        int a,b;
        a = 10;
        b = 0;
```

best practice use Exception obj in outer catch as it should be able to catch all type of exception.

```
try {
    S.O.P(a/b);
}
catch ( ArithmaticException ae) {
    ae.printStackTrace();
}
finally {
    S.O.P(" Exception handled ");
}
}
}
```

Can also write only Exception as it is parent class of ArithmaticException.

☆ Nested try-catch :

```
try {
    code(s)
    try {
        code(s)
    }
    catch(Exceptionclass obj) {
    }
    code(s)
}
catch(Exceptionclass obj) {
}
```

first this catch try to solve. if not able to solve then exception comes in outer try

then this catch try to solve.

```
import java.util.Scanner;
class NestedTryCatch {
    public static void main(String[] args) {
        int a[] = {10,20,30,40,50};
        int b=10; c=0;
        try {
            try {
                Scanner sc = new Scanner(System.in);
                b = sc.nextInt();
                c = sc.nextInt();
                s.o.p(b/c);
            }
            catch(NullPointerException obj) {
                obj.printStackTrace();
            }
            finally {
                for(int i=0; i<=a.length; i++) {
                    s.o.p(a[i]);
                }
            }
        }
        catch(ArithmeticException ae) {
            ae.printStackTrace();
        }
        catch(ArrayIndexOutOfBoundException obj) {
            obj.printStackTrace();
        }
    }
}
```

This exception will not occur as arithmetic exception already occur and handled by outer catch ∴ this code is not executed.

for both the exception to occur use scanner take input of b & c in inner try or my solution:
use finally in inner try but
finally always used to close all the object used.

if exception handled terminates normally
if not handled - terminated abnormally by JVM.

Exception propagation between methods:

```
import java.util.Scanner;
class Am1 {
    void m1() {
        int a,b;
        S.o.p("Enter Two no");
        Scanner sc = new Scanner(System.in);
        a = sc.nextInt();
        b = sc.nextInt();
        S.o.p(a/b);
    }
}
class Bm2 {
    void m2() {
        Am1 obj = new Am1();
        obj.m1();
    }
}
class DemoABm {
    public static void main(String[] args) {
        try {
            Bm2 obj = new Bm2();
            obj.m2();
        }
        catch (Exception obj2) {
            obj2.printStackTrace();
        }
        S.o.p("code runs normally");
    }
}
```

Exception thrown

control transfer

Exception thrown

control transfers

handled here

// if not handled, handled by JVM and program terminate and, line is not executed then.

Throw फेंकता!
Throws फेंकता है! } khaas baat ye hai!

★Throw: it is used to throw an exception. with the help of throw we can throw our own exception

★ Throws: it is used to declare any method throwing some exception.

eg. `void m3() throws IOException, SQLException`

method m3() throws 2 exceptions which must be handled by the calling method

if

m3() → m2() → m1() → main()

```
try{
    m3();
}
catch(SQLException | IOException obj){
}
```

pipe operator used to handle multiple exception

★ Throwing my own Exception:

```
// we can create our own exception by extending Exception class
import Java.util.Scanner;
Class myClass extends Exception {
    myClass(String msg){
        super(msg);
    }
}
class testException{
    void m1() throws myClass {
```

```
        int a;
        Scanner sc = new Scanner(System.in);
        a = sc.nextInt();
        if(a < 18){
          throw new myclass("age should be less than 18 to go ahead");
        }
      }
    }
    class demoException {                    throws myclass (try doing this and then try catch)
      public static void main(String[] args){
        testException obj = new testException();
        try{
            obj.m1();
        }
        catch(myclass e){
          e.printStackTrace();
        }
      }
    }
```

★ Throws by overriding :

- if superclass don't throws exception child class can throw only unchecked exception.
- if superclass throws exception then child class can throw only :
  1. same exception
  2. subclass exception
  3. no exception

but can't throw parent class exception.

★ see codes in ppt.