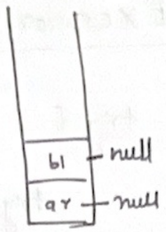
```
int ar[] = null;
ar[1] = 10;
Box b1 = null;
b1.i = 5;
```



Finally ⇒

```
class A {
    PSVM(_){
        ProcA();
        SOP(ProcB());
        try{
            ProcC();
        }catch(Ari..Ep. ae){
            SOP("In Main");
        }
    }
```

```
SV ProcA(){
    try{
        SOP("No Exception");
    }finally{
        SOP("ProcA");
    }
}
int ProcB(){
    try{
        int i;
        SOP("In Proc B");
        return 10;
    }
    finally{
        return 5;
    }
SV ProcC(){
    try{
        int a = 10/0;
    }finally{
        SOP("In ProcC");
    }
}
```

Output :⇒ ⇒ No Exception
⇒ Proc A
⇒ 5
⇒ In Proc C
⇒ In Main

{Only in abnormal termination Case is the one where finally might not execute}

{Here Exception is not handled that's why Catch of Main will execute.}

9/4/2024

User Defined Exception

→ Invalid Marks Exception → {0 – 100}

⇒ Class InvalidMarksException extends Exception {

```
    String msg;
    Public InvalidMarksException(String msg){
        this.msg = msg;
    }
    String toString(){
        return msg;
    }
}
```

⇒ Class A {

```
PSVm1() throws InvalidMarksException {
    int i;
    Scanner sc = new Scanner(System.in);
    i = sc.nextInt();
    if(i<0 || i>100)
        throw new InvalidMarksException("Marks must be
        in between 0 to 100");
}
PSVM(_){
    try{
        m1();
    }Catch(InvalidMarksException e){
        SOP(e);
    }
}
```

12/4/2024

Exception Handling ⇒

User defined exception →

Multi-Catch ⇒

```
try {
  ___
}
catch(ArithmeticException e){
  ___
}
catch(Array ---){
  ___
}
:
```

From JDK-7 onwards ⇒ We can use "|" (OR)

```
try {

}
catch(ArithmeticException |ArrayIndex
OutOfBoundException e)
{
  ___
}
```

eg ⇒

```
int arr[] = {5, 10, 15, 20}
try {
    int a = 20/d;
    a[50] = 50;
}
catch (ArithmeticException | ArrayIndexOutofBoundException e)
{
  ___
}
```

From JDK-5 we got Chained Exceptions ⇒ 2 methods were added

Throwable initCause(Throwable t)

⇒ Throwable getCause();

example ⇒

```
class A {
    static void demoProc(){
        NullPointerException ne = new NullPointerException();
        ne.initCause(new ArithmeticException());
        throw ne;
    }
    PSVM(—){
        try{
            demoProc();
        }catch(NullPointerException n){
            system.out.println(
        }
    }
}
```

This will not compile {logic is not correct, example was given to explain Chained Exception}

```
Class A{
    static void demoProc() throws Throwable
    {
        throw new NullPointerException().initCause(new ArithmeticException());
    }
    P SVM (){
        try{
            demoProc();
        } catch(Throwable e){
            SOP(e)
        }
    }
```

Output ⇒
① Java.lang.NullpointerException
If we do SOP(e.getCause());
① Java.lang.ArithmeticException
Method have to throws Throwable as both initCause(), getCause() wont work.

from JDK-9 we got ⇒ Try with resource

```
try {
    FileInputStream fin = new FileInputStream("a.txt");
    :
} catch(--) {
} finally {
    fin.close();
}
```

OR

```
try (FileInputStream fin = new FileInputStream("a.txt"))
{
}
catch(--) {
}
```

This will automatically close the file

① Close(). of fin will be closed automatically

② Resources which implements AutoCloseable Interface can be used in try.

③ All Stream classes (IO ones) implements Autocloseable which is in Java.lang.

Java.io.closeable
↓
Java.lang.AutoCloseable

from JDK 10 ⇒

```
try (var fin = new FileInputStream("a.txt"))
{
}
```

```
try (FileInputStream fin = new FileInputStream("a.txt");
     FileOutputStream fout = new FileOutputStream("b.txt"))
{
}
```