1/4/2024

```
interface I1{
    int i=0;                          ✓
    final int a=5;                    ✓
    Static final int x=10;            ✓
    Public static final int y=15;     ✓
    Private static final int P=20;    ✗
}
```

{ Before JDK-8 , definition/Implementation of method was not allowed inside Interface, It was used as a prototype. But from JDK-8 onwards we can provide a default implementation inside Interface }

```
interface I1{
    Void m0();
    default Void m1(){
        S.o.P. {"Default implementation");
    }
}

Class A implements I1{
    Void m0(){
    }
}
```

← we do not need to implement m1, if we don't provide implementation then it can call it's default .m1().

{ from JDK-8 onwards Interfaces can have static methods as well }

```
interface I1{
    Static void m1(){
    }
}
```

```
Class A implements I1{
    I1.A();
}
```

{ from JDK-9 we can have Private methods {have to have implementation} }

```
interface I1{
    Private void m1(){
    }
    default void m2(){
        m1();
    }
}
```

default + keyword

default Public v.m1() ✓

```
Class A implements I1{
    Void m3(){
        m2();
    }
}
```

default + Public ✓, Private ✗, Protected ✗

{ If we do not provide method body in case of default / Private methods in interface then we would get an error: "Missing method body, or declare abstract" }

{ In case if we make default / Private abstract we would get error of illegal modifier combination. }

Modifiers ⇒

(1) final ⇒ Var → Local variable
Var → Instance var
Var → Class var

methods ⟶ Instance method

Class ⟶ final class

Final Class ⇒ ⇒ final class cannot be extended.

⇒ Execution of reference of final class is bit faster because subclass of final class is not possible.

```
Class A {

}

A a1;
```

{Binding happens at run time.}

```
final
Class B {

}

B ab1;
```

{final class binding is done at compile time}

Final Local Variable ⇒

```
Class A {

  Void m1(){
    final int a = 10;
     a = 20; //error
  }

  Void m2() {
     final int i; // Blank final variables
      i = 10; ✓
  }

  Void m3() {
     final int i;
      i = 10;
      i = 20; // error
  }
}
```

Final Instance Variable ⇒

```
class A {
final int i;
}
```

```
class B {
  A a1 = new A();
  a1.i = 10; ✗ error
}
```

{We cannot assign value to final instance variable from outside of the class.}

```
class A {
final int i;
 A() {
   sop(i); //error
 }
}
```

{No default initialization for instance final variable}

instance final variable can be initialised at :->

① Declaration

② Constructor

③ Initialiser.

---

①②

```
Class A{
final int i=5; ✓
final int j;

   A(){
      j=5; ✓   i=10; ✗
   }
}
```

```
Class A{
final int i;

   A(int j){
      i=j;
   }
}

A a1 = new A(5); ✓
```

```
Class A{
final int i;

   {
      i=2;
   }

   A(){
   }
}
```

Initialisation Block → A Class can have any no. of initialisation blocks.

These initialisation blocks will be executed in the sequence of their declaration and right before the execution of any Constructor

02/04/2024

Static final Variable ⇒ Can be assigned value at → either at declaration / Static initialisation Block.

```
class A{
   static final int i=1;
   static final int j;
   static {
      j=2;
   }
}
```

→ will execute only once when class gets loaded to the memory.

Final Method ⇒

⇒ The method cannot be overridden.

⇒ Binding of final methods is done at Compile time — Static Binding.

⇒ for final methods, the call of method can be replaced by its code, which is called inlining but this is completely dependent on your JVM.

Local Variable can only have one modifier - Final

(2) STATIC

⇒ Variable

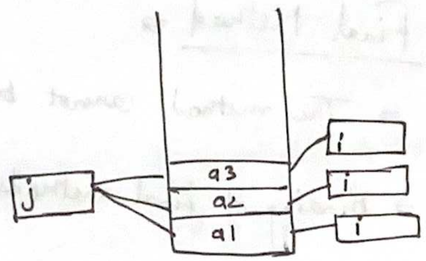
⇒ Method

⇒ Code block.

→ Static Variable ⇒ Class level variable

→ one for a class, all the objects share same variable.

```
class A {
  int i;
  static int j;
}
A a1 = new A();
A a2 = new A();
A a3 = new A();
```



{a1.j = a2.j = a3.j = A.j}

→ Static Method ⇒

⇒ A static method can't use non-static method of the same class directly.

Static Block ⇒

```
Static {
  ____________
}
```

← Things we want to do at class level.

(3) Abstract ⇒

→ Class → which cannot be instantiated

→ Method → for which implementation is not provided.

(4) Synchronised ⇒ {Multi-threading chapter}

(5) transient ⇒ {I/O Chapter}

(6) native → native code {for ex. C .lib} can be accessed by using native modifier.

(7) Volatile ⇒ for no optimization.

Exception Handling

⇒ Errors at run time

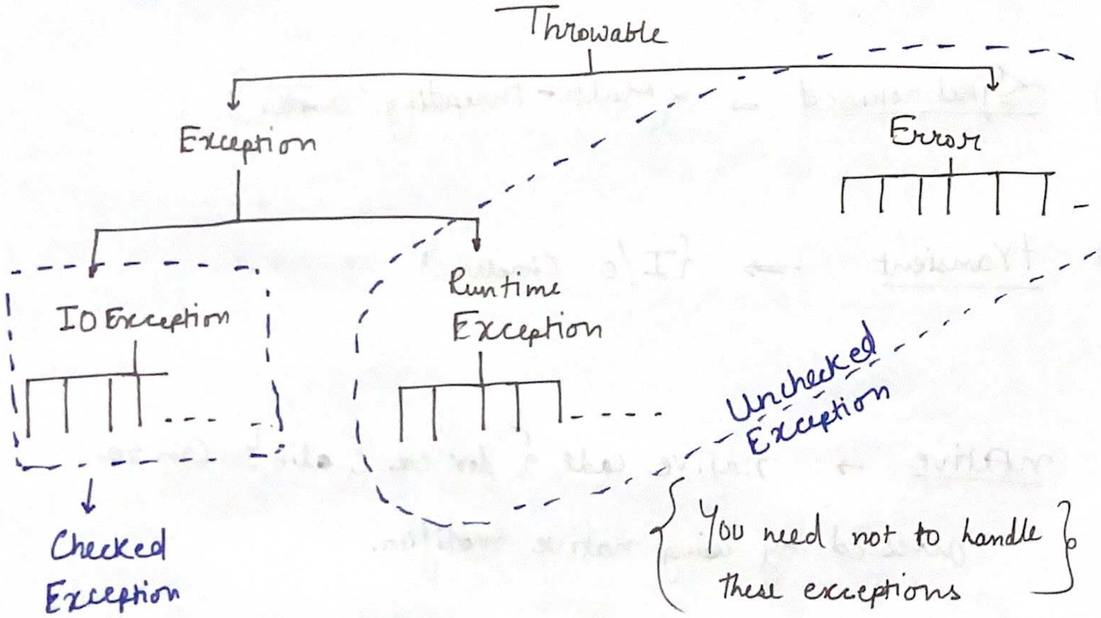
```
int i, j;
Read i, j;
i/j;
```

values assigned at run time

10/0 // divide by 0 exception

Errors
- Compile – Syntax errors
- Runtime – Semantic errors

Exception Hierarchy.



3/4/2024

Throwable :→

String getMessage() → Message string

String toString() → String representation of exception object.

Void PrintStackTrace() → Whole hierarchy of movement of exception.

```
A a1 = new A();
a1.i = 5; a1.j = 10;
SoP(a1);
```

This calls toString() of that Object.

Output :→ A@23ab12
- A → Class
- 23ab12 → hash code

Let's by SoP(a1) we want it display i & j values, then.

```
Class A{
    int i,j;
    Public String toString(){
        return "i=" + i + "j=" + j;
    }
}
```

Class A{
int i,j;

Output of SoP(a1) :→ {i=5 j=10}

```
Class A{
  PSVM(_){
    int a;
    a = 10/0;
    SOP("After a");
  }
}
```

Compile

:→ java A

Exception in thread Main

java.lang.ArithmeticException

DivisibleByZero.

```
Class A{
  PSVM(_){
    SOP("Before M1");
    m1();
    SOP("After M2");
  }
  pbv m1(){
    int a;
→   a=10/0
  }
}
```

Output ⇒

- Before M1
- Exception

java.lang.Arith - - - -

At A.m1 : 10

At A.main : 5

```
try {
  - Statement
}
Catch( Exp1   e1) { }
Catch (Exp2   e2) { }
Catch (Exp3   e3) { }
finally {
  - always
}
```

If exp of type Exp1 → Catch (Exp1); exp2 type → Catch (Exp2); exp3 type exception → Catch (Exp3)

If it is not of {Exp1, Exp2, Exp3} type. → finally

finally: at most 1

try can have min one of these statements {finally or Catch}

0 Catch	1 finally
1 Catch	0 finally
1 Catch	1 finally
2 Catch	1 finally
5 Catch	1 finally

finally will run

- even if exception is there, catch ~~did~~ wasn't there.
- if exception is not there.
- if try has return

```
class A {
  PSVM(_)
  {
    int a;
    try{
      SOP("Before Division");
      a=10/0;
      SOP("After Div");
    }catch( ArithmeticException e)
    {
      Sop(" Some Exception has Occured: " + e);
    }
    SOP("After Catch");
  }
}
```

Output :→ Before Division

Some Exception has Occurred - - .

After Catch

```
Class A {
  PSVM(_){
    try{
      int a = args.length;
      int b= 42 / a;        // Case 1 :→
      int c[] = {1};
      c[42] = 10;           // case 2:→
    }
    Catch( ArithmeticException e){
      SOP(ae);
    }
    catch( ArrayIndexOutOfBound ae){
      SOP(ae);
    }
  }
```

Case 1 ⇒ java A (Value of e)

Case 2 ⇒ java A 10 (Value of ae)

4/4/2024

```
class Recovery{

    PSVM(String[]ar){

        int cost, qty, rate;
        Scanner sc = new Scanner(System.in);
        SOPC("Enter cost: ");
        cost = sc.nextInt();

        while(true){

            try{
                soPC"Enter Quantity: ");
                qty = sc.nextInt();
                rate = cost/qty;
                break;
            }
            Catch(ArithmeticException e){
                soPC"Quantity should be non-zero");
            }
        }
        soPC"Rate= "+rate);
    }
}
```

If 0 is entered then we would go to catch, then loop would iterate again until valid value is entered

```
Class

for(int i=0; i<4; i++){

    int k=0;

    try{

        switch(i){

            Case 0: int zero=0;
                    k=900/zero;
                    break;

            Case 1: int b[]=null;
                    k=b[0];
                    break;

            Case 2: int[]c=new int[2];
                    k=c[9];
                    break;

            Case 3: char ch="abc".charAt(3);
                    break;
        }
    }

    Catch(Exception c){
        soP(c);
    }
}
```

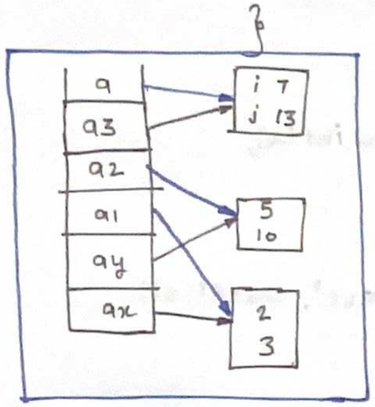
```
Catch(Exception e){
    soP(e);
}
Catch(ArithmeticException ae){
    soP(ae);
}
```

we can have subclass & superclasses of exception together but where would get error, because Exception is before ArithmeticException, and Arithmetic Exception block would never execute.

Object Passed as Parameter

```
Class A {
    int i, j;
    A Add(A a1, A a2){
        A a3 = new A();
        a3.i = a1.j + a2.i;
        a3.j = a1.j + a2.j;
        return a3;
    }
}
```

{ Object Reference gets passed as Value }




```
Class B {
    PSVM(-){
        A a;
        A ax = new A();
        A ay = new A();
        ax.i = 2; ax.j = 3;
        ay.i = 5; ay.j = 10;
        a = ax.add(ax, ay);
        SOP(a.i+" "+aj);
    }
}
```

5/4/2024

```
Class A
{
    PSVM(-){
        //File copy
    }
}
```

IO Exception
↓
FileNotFound Exception

{ Compilation error }

Because it is ~~catch~~ Checked exception

```
Class A{
    PSVM(-){
        try{
            //File Copy
        }catch(IOException e){
        }
    }
}
```

```
Class A{
    PSVM(-) throws Exception{
        //File Copy
    }
}
```

```
Class A{
    PSVM(_){
        m1()
    }
    PSV m1(){
        m2();
    }
    PSV m2(){
        m3();
    }
    PSV m3(){
        //file Copy
    }
}
```

Either we can have try catch here OR Can we use throws for which we have to use try-catch with caller.

```
Class A {
    PSVM(_) throws IOException{
        m1();
    }
    PSV m1() throws IOException{
        m2();
    }
    PSV m2() throws IOException{
        m3();
    }
    PSVm3() throws IOException{
        //File copy
    }
}
```

Sequence has to be maintained

PSVM(_) throws IOException, ArithmeticException, ArrayIndexOutOfBound ...

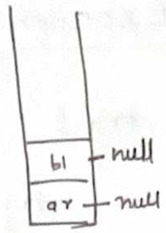
Class hierarchy of exceptions in above cases has to be maintained, Similar Meaning if m3() throws Exception (higher hierarchy) then m2() or m1() or main() shouldn't throw ArithmeticException (lower hierarchy). Vice-Versa is possible

Nested Exception

```
try{
    try{
        ___
    }catch(_){
        ___
    }
}catch(_){
    ___
}
```

```
Class A{
    PSVM(String[]ar){
        int a = ar.length;
        try{
            int d = 10/a;
            try{
                if(a==1){
                    a=a/(a-a);
                }
                if(a==2){
                    int b[]={1};
                    b[20]=50;
                }
            }catch(ArrayIndexOutOfBoundException e)
            {
                ___
            }
        }catch(ArithmeticException a){
            ___
        }
    }
}
```

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



Finally ⇒

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

```
SV ProcA(){
    try{
        SOP("No Exception");
    }finally{
        SOP("ProcA");
    }
}
int
SV 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.}