- Decloration and Access Modifiers

(1) Java Source file structure.
(2) class level modifiers
(3) Member level modifiers
(4) Interfaces.

- Java Source file structure :

- Java contains any no. of classes. but atmost one class - Can be declared as "public".

- if there is a public class then name of the program & name of the public class must be matched. otherwise
- We will get compile time Error.

```
Eg. Class A
    {
      ==
    }
    class B
    {
      ==
    }
    class C
    {
      ==
    }
```

Case I: If there is no public class then we can use any name and there are no restricth.

eg. A.java
B.java
C.java
cs.java

Case II: If class B is public then name of the program should be B.java. otherwise we will get Compile-time Error saying "class B is public, should be declared in a file named B.java"

case III: If class B and class c declared as public and name of the program is B.java then we will get compile time error saying "class c is public, should be declared in a file C.java"

Eg.②
```
class A
{
  Public static void main (String args[])
  {
    System.out.println(" A class main");
  }
}
class B
{
  public static void main (String args[])
  {
    System.out.println("B class main");
  }
}
class C
{
  public static void main (String args[])
  {
    System.out.println ("c class main");
  }
}
class D
{

}
```

→ No any public class, so we can use any name.

$$\begin{tikzcd} & & \text{javac first.java} \arrow[dll] \arrow[dl] \arrow[d] \arrow[dr] & \\ \text{A.class} & \text{B.class} & \text{C.class} & \text{D.class} \end{tikzcd}$$

→ java A ↵

O/P: A class main.

→ java B ↵

O/P : B class main

→ java C ↵

O/P : C class main

→ <u>D.class</u> : No main method

∴ java D ↵

RuntimeError: Nosuch Method found.

→ java first ↵

R.T : NoClassDefFound : first

Conclusions

- Whenever we are compiling a java program for every class present in that program a seprate .class file will be generated.
  [In our eg. 4 .class file generated as we have 4 classes]

- We can compile a java program (java source file) but we can run a java .class (.class file).

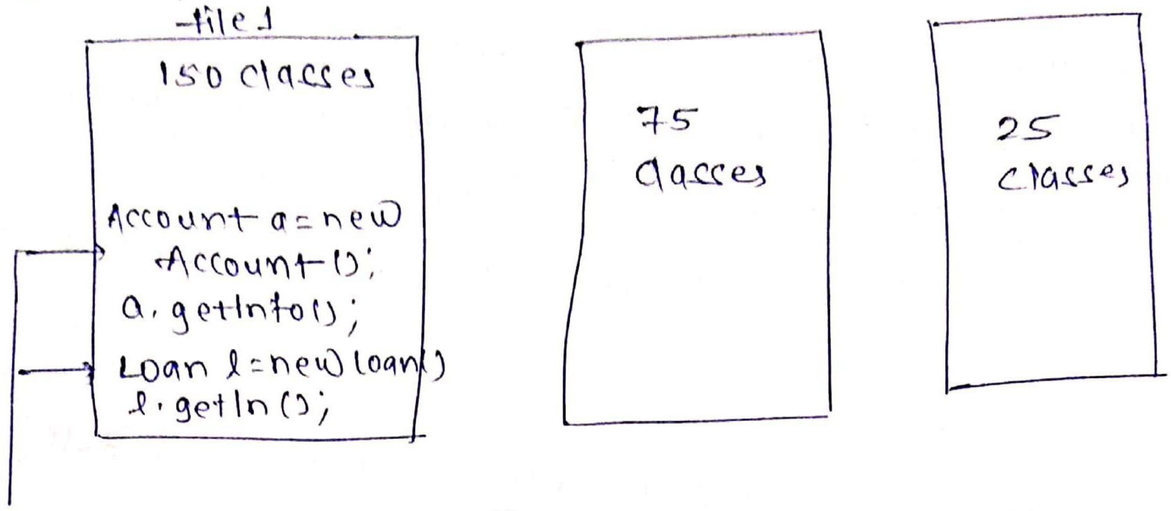
- Whenever we are executing a java class corresponding class main method will be executed.

- If that class doesnot contain main method then we will get runtime exeeption saying "NosuchmethodError: main".

- If the corrosponding .class file not available then we will get Runtime Exception saying "noclassDefFound-Error: first".

- It is not recommended to declare multiple classes in a single source file, It is highly recommended to declare only one class per source file and name of the program we have to keep same as class name. The main advantage of this approach is readability and maintainability of the code will be improved.

Eg. suppose there are 250 classes.



Need to search in all 3 files.
∴ Instead create seprate file for each class.
Account.java
Loan.java & so on.

import statement:

Eg.
```
class Test
{
    public static void main (String args[])
    {
        ArrayList l = new ArrayList();
    }
}
```

compileTimeError:
Cannot find symbol: class: ArrayList
location: class Test.

- We can solve this problem by using fully Qualified name.

```
java.util.ArrayList l = new java.util.ArrayList();
```
$\uparrow$
fully Qualified name.

The problem of fully Qualified name is everytime
+ it increases length of code and reduces readability.

- We can solve this problem by using import statement
- Whenever we are writting import statement it is not requred to use fully qualified name everytime, we can use short name directly.

Eg.
```
import java.util.ArrayList;
class Test
{
    public static void main(String args[])
    {
        ArrayList l= new ArrayList();
    }
}
```

- There are two types of import statements
  (1) Explicit class import
  (2) Implicit class import.

(1) Explicit class import

Eg. import java.util.ArrayList;

- It is highly recommended to use explicit class import bcoz it improves readability of the code.

- Best suitable for applicatn where readability imp.

(2) Implicit class import

Eg
```
import java.util.*;
```

- Not recommended to use bcoz it reduces readability of the code.

Eg.

```
import com.hdfc.*;
import com.icici.*;

==

Account a= new Account();
a.getInfo();
==

Loan l= new Loan();
l.getInterestRate();

==
```

→ To search loan & Account class, we need to spend time to search in com.hdfc & com.icici. (i.e Implicit) to overcome this use explicit i.e

```
import com.hdfa.Account;
import com.icici.Loan;
```

Case II: Which of the following import stmts are meaningful.

✓(a) import java.util.ArrayList;
×(b) import java.util.ArrayList.*;
✓(c) import java.util.*;
×(d) import java.util;

Case III: Consider the following code

```
class myobject extends java.rmi.UnicastRemoteObject
{

}
```

$\underbrace{\text{java.rmi.}}_{\text{fully Qualified}}$

The code compiles fine eventhough we are not writting import statement because we used fully Qualified name.

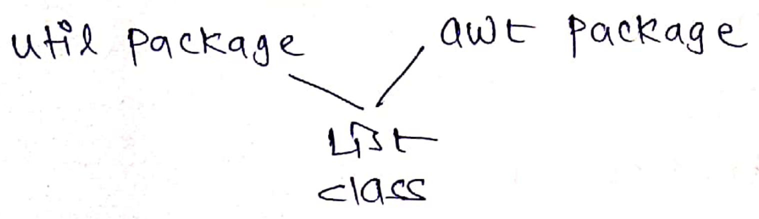
NOTE: Whenever we are using fully qualified name, it is not required to write import statement. similarly

Whenever we are writing import stmt it is not requred to use feilly qualiffied name.

Case IV :

```
import java.util.*;
import java.sql.*;
class Test
{
 public static void main(——)
 {
      Date d= new Date();
  }
}
```

CT : reference to Date is ambiguous.

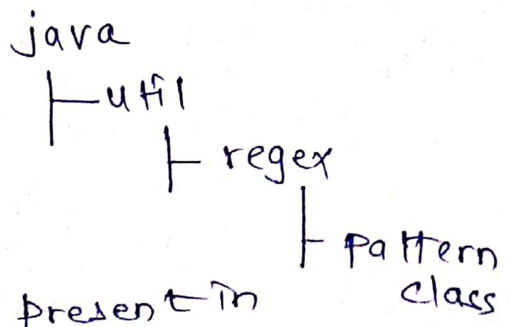
→ 

→ import java.util.Date;
import java.sql.*;
} compiler will give priority to explicit class import next to implicit class import.

Case V :
Whenever we are importing a java packages, all classes & interfaces present in that packages by default available. but not subpackage classes. If we want to use sub-package class compulsory we should write import stmt until subpackage level.

Eg.

import java.util.regex.*;



Case VI : All classes and interfaces present in the following packages or by default available to every java program hence we are not required to write

(a) java.lang
(b) default package (current working directory)

→

```
class Test
{
 static String s = "java";
 }
```

find length of s ?

⇒

Test. s.
↳ static variable can be access through class name.

∴ Test . s. length ();
↳ method of string class.

Test ↓ A class name

s ↳ Static variable present in Test class of type java.lang.string

```
class System
{
   static printstream out ;

}
```

System.out. println();
↳ printstream class have println() method.

System ↓ A class present in java.lang Package.

out ↓ static variable present in System class of type printstream