Declaration & Access Modifier :-

1. Java Source file structure
2. Class level modifier
3. Member level modifier
4. Interfaces.

1) Java Source file structure.

→ A java program contain any no. of classes, but at most one class can be declared as public.

→ If there is a public class, then the name of program & name of the public class must be matched otherwise we get compile time error.

Case 1

```
class A{
}
class B {
}
class C {
}
```

→ Any name to this program, no restriction.

→ A.java
→ B.java
→ C.java
→ Durga.java

Case 2

```
class A {
}
public class B{
}
class C{
}
```

→ Then compulsory, this program should give name as B.java

→ If class B is public, then name of program should be B.java.

Case - 3

```
class A {
}
public class B {
}
public class C {
}
```

Invalid.

→ If class B & C declared is public named of program is B.java, then we will get compile time error.

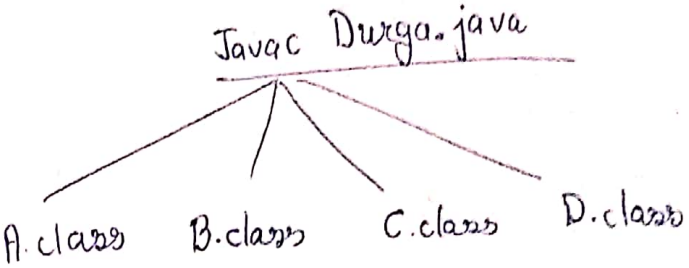
We compile a java file.

We run a java class.

→ There is no relation of main method & name of the java file.

```
class A {
    psvm(String[] args){
        Sout("A class main");
    }
}
class B {
    psvm(String[] args){
        Sout("B class main");
    }
}
class C {
    psvm(String[] args){
        Sout("C class main");
    }
}
class D{
}
```

Durga.java



Java A ↵

o/p: A class main

Java B ↵

o/p: B class main

Java C ↵

o/p: C class main

Java D ↵

o/p: NoSuchMethodError: main

Java Durga ↵

o/p: NoClassDefFoundError: Durga.

→ One Source file must contain only one class (Highly recommended)

→ com.micros.vi.common.dataElement.data

↳ Fully Qualified Name

import java.util.Arraylist;

↳ Best use of shortcut.

Case-1 types of import statement:

Explicit

i.e. import.java.util.Alist;

- Highly recommended.

Implicit

i.e. import.java.util.*;

- Readability
- Maintain-bility

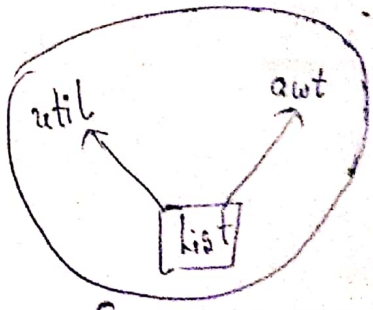
} Reduced.

Case 3 → Either use import or fully Qualified name.

Case 4 →

```
import java.util.*;
import java.sql.*;
class test
{
  psvm(String[] args)
  {
    Date d = new Date();
  }
}
```

Case 4.1



Same reference error.

→ Compile error → Reference to data is ambigious.

Thumb Rule

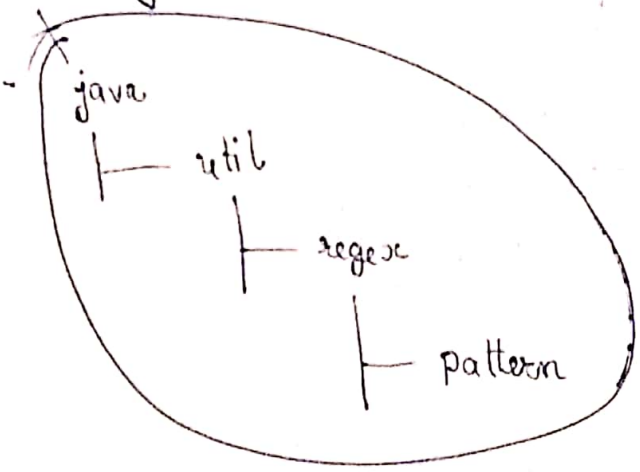
Case-5

Priority ↓

1. Explicit class import
2. Classes present in current working Directory (CWD).
3. Implicit class import

Case 6 → While importing the package, all interface & sub-class will be import, But not ~~subclasses~~ ;, package class

→ Compulsory we should write import package untill sub-class. package.



Untill sub-package class.

Case 7 →

1. java.lang
2. default pkg (CWD)

} Not require to import.

→ All classes & interfaces present in following packages are default available in every java program.

Not require to write import package.

Case - 6

Fully Qualified name	Short names import java.lang.* AL
① Compile time less	Compile time will be more
②	

Both program will take same time to run

Case - 7

C	Java
#include <stdio.h>	import java.io.*;
① Static Include	① Dynamic include ② Load on demand

printWriter

1.5 V

① for-each loop
② var-arg method
③ Autoboxing & Auto unboxing
④ Generics
⑤ Co-variant return types
⑥ Queue
⑦ Annotations
⑧ enum
⑨ Static import ⟹ flop version

Without Static import

```
class test
{
    psv main(String[] args)
    {
        Sout.println(Math.sqrt(4));
        Sop(Math.max(10,20));
        Sop(Math.random());
    }
}
```

With static import

```
import static java.lang.math.Sqrt;
import static java.lang.math.*;
class test
{
    psv main(String[] args)
    {
        Sop(Sqrt(4));
        Sop(max(10,20));
        Sop(random());
    }
}
```

Static import

Static import

Only import class

① Explicit class import
② Current Working directory
③ Implicit class import

Only import static class

① Current class static import
② Explicit static import
③ Implicit static import

Only 2 packages contain class or interface with the same name.

Eg. Date ← Sql / util

List ← util / Awt

Packages - statement

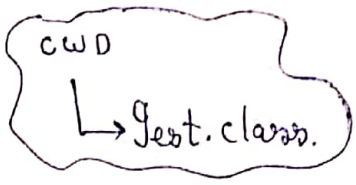
↳ A group of related thing. into a single unit.
- → Related classes
- → Related interface
- i.e. → Sql package
  → io package

Purpose
- → To resolve naming convention
- → Modularity increase.
- → Maintainability of module increased.
- → Security → outside can not access components.

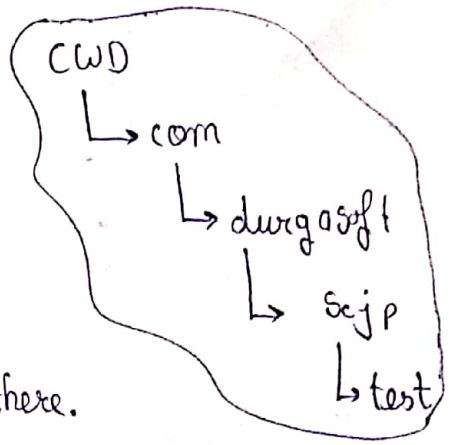
Case-1

```
package com.durgasoft.scjp;
public class Test{
    PSV main(String[] args)
    {
      Sop("pkg demo");
    }
}
```

(i) Javac Test.java



(ii) Javac -d . Test.java



① In any file at most one package will be there.

② ~~import java.util.AL;~~ ~~package ...~~ ✗

package ...
import java.util.AL; ✓

Atmost one → Package Statement
Any number → imports Statement
Any number. → class | interface | enum | declarations.

✓ Order is important.

class level modifiers

↳ to info. about class to JVM.

→ Public - It can access from anywhere

→ final → It can not access/ create child class

→ Abstract → Wheather obj creation is possible or not.

```
class test {
    .
    .
    .
    .
}
```

---

12 modifiers

Public	final	native
Private	abstract	strictfp
<default>	static	transient
Protected	Synchronized	volatile

→ Only modifier for top level classes are:

5 modifier

- public
- default
- final
- abstract
- strictfp

For Inner level classes are (8) -> modifier

- Public
- private
- <default>
- protected
- abstract
- final
- static
- strictfp

Top class level

①
```
private class test {
}
```

C.E. → modifier private not allowed here.

→ Access specifier Vs Access modifiers.

Public
Protected
<default>
private

All 12 are modifier

No such specifier are allowed.

1] Public class → If the classes declared as public then we can access that class from anywhere.

2] default class → If the class declared as default.
// package level access,
then we can access that only within the current package.
→ Outside package we can't access

3] final modifier → [classes, method, variable]

Uniqueness
Security

3.1 Final method (No change)

↳ It stops overriding.

3.2 Final class

↳ We cannot extend the class

↳ Child class can not implement from parent class.
Cannot allowed to create child class.

Case 3.3

```
final class P
{
   m1();
   m2();
    .
    .
   m1000();
}
```

→ Every method in final class are by default final.

→ It implicitly final

→ Every variable present in final class are need not to be final.
It can be re-assign.

final → loss

missing ↳ Inheritance (class) → ×

↳ polymorphism (method overiding) → ×

4] Abstract Modifier → [Classes, Method] Not for variable

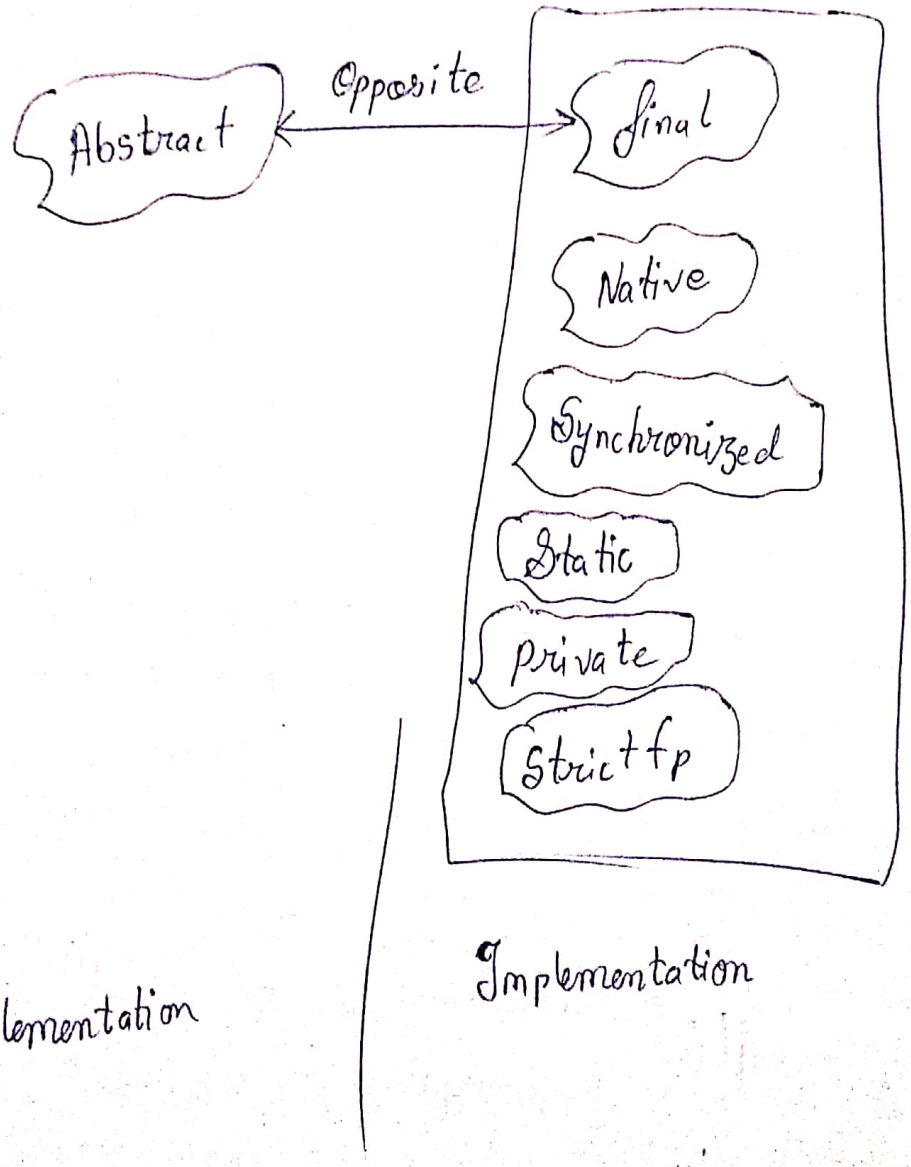
4.1] Abstract method

→ It does not know about implementation, we can declare modifier for abstract method.

→ It implementation done in child class.

Note :- If a class method contain abstract method, then that class must represent with abstract modifier

→ Only declaration method no implementation.



4.2] Abstract class

- Atleast one abstract method → [Zero or more]
- In java class, there is no need/want to create object.
- partial implementation.
- Instantiation is not possible.

i.e.

HttpServlet → Is abstract but it does not contain any abstract method.

Abstract	final
① Method : Need to override	① Method - No overriding.
② class - Need to extend child class.	② class - Can't extend child class
③ Inside abstract → final method.	③ final class → Abstract method inside

```
abstract class Test {
    public final void m1(){
    }
}
```

Valid.

promotes oops features

```
final class Test {
    abstract public void m1(){
    }
}
```

Not valid.
Not able to overload

Suppress oops features.

5] Strictfp : [strict floating point] 1.2 version

- Classes
- Method

Not for variable

- Need platform independent result.

5.1] Method:

If a method declared as strictfp
all floating point calculation has to follow IEEE 754 standard.
So we will get platform independent result.

```
strictfp void m1(){
    Sop (10/3);
}
```

5.2] Classes:

If a class declared as strictfp.
then inside all concreate method has to follow IEEE 754 standards.

```
abstract strictfp class test {
    m1();
    m2();
    m3();
    m11();
    m12();
    m13();
}
```

Concrete method

abstract method

Note:

- Abstract strictfp combination at class level is legal
- But illegal for method.

Member level modifiers

Variable level modifiers

1] public - members

- It can be accessed from anywhere.
- But corresponding class should be visible.

2] Default members

- Members declared as default then we can access that member only with the current package.
- package level access.

3] private member

- It can accessed that member only within the class.

Note :- Abstract method should be available to child classes. To provide implementation, where as private method are not available to the child class provide implementation.

4] protected members (The most misunderstood modifier)

- Only in current package
- Outside only in child classes

protected = <default> + kids

protected member Access

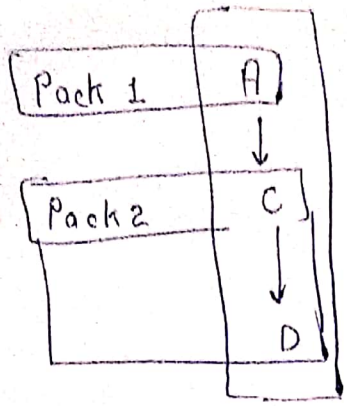
```
package pack1;
Public class A
{
   Protected void m1() {
      Sop("Hai");
   }
}

class B extends A
{
   PSV main(String[] args)
   {
①    A a = new A();
       a.m1();

②    B b = new B();
       b.m1();

③    A a1 = new B();
       a1.m1();
   }
}
```

```
Package Pack2;
import Pack1.A;
class C extends A
{
   psv main(String[] args)
   {
①    A a = new A();
×      a.m1();

②    C c = new C();
       c.m1();

③    A a1 = new C();
×      a1.m1();
   }
}
```

Note: We can access protected member within current package anywhere either by using parent reference or using child reference.

But we can access protected member in outside package only in child classes and we should use child reference only.

that is parent reference cannot be used to access protected member from outside package.



$\Rightarrow$ D d = new D()
d.m1();

We can access protected member from outside package and we should use that child class reference only.

Private < default < protected < public

recommended data member - private
" method - public

variables
- Instance variable (object level variable)
- Static variable
- local variable.

① Final Instance variables :

- If instance variables declared as final must need to initialized. whether we are using or not.
  Jvm not provide default value.
- Perform initialization before constructor completion.
  - ① At the time of declaration.

```
class test
{
  final int x = 10
}
```

  - ② Inside instance block (before constructor)

```
class test
{
  final int x;
  {
    x = 10;
  }
}
```

  - ③ Inside constructor.

```
class test
{ final int x;
  Test()
  {
    x = 10;
  }
}
```

② Final static variable. (Only one copy is same for every object)

↳ for static variable - not require to perform initialization.
JVM will provide default value. (0).

final: need to initialize explicitly

↳ Before class loading, perform initialization.

① At the time of declaration

```
- class test {
      final static int x = 10;
  }
```

② Inside static block

```
- class test {
      final static int x;
      static
      {
        x = 10;
      }
  }
```

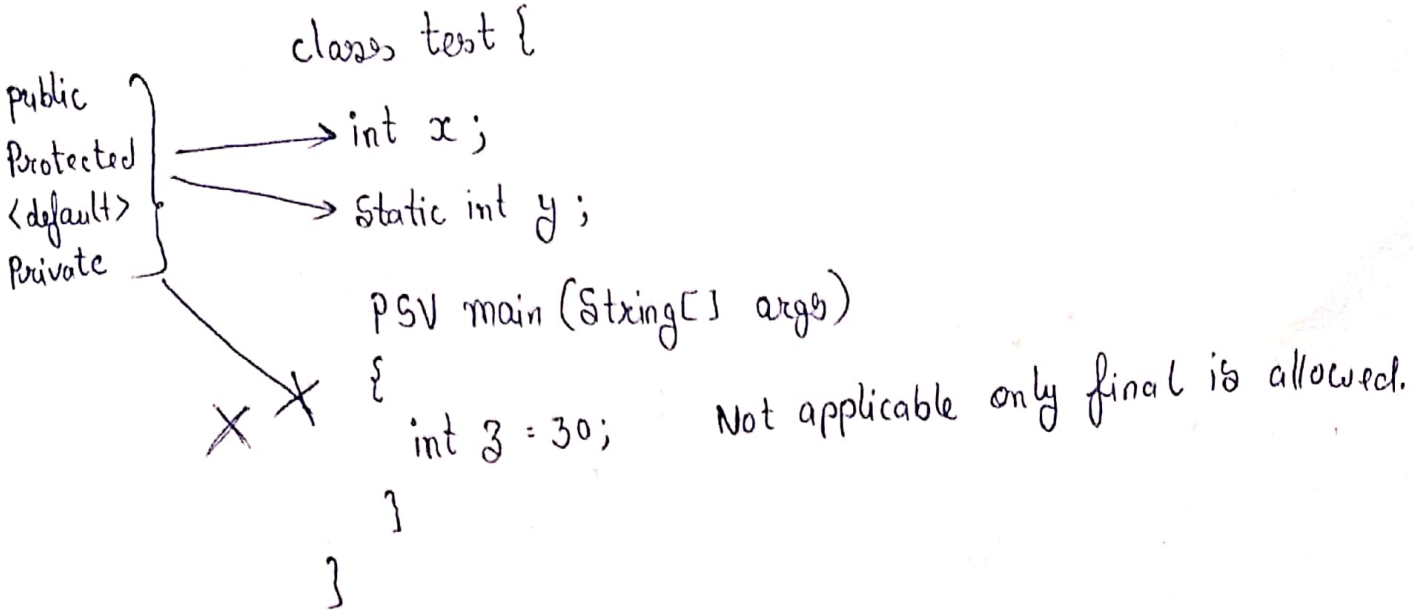
Final local variable - declared in method, block, constructor.

↓
temporary
↓
Automatic
↓
Stack

↳ Before using that local variable, perform initialization.

Final local variable.

- Before using local variable.
- The only modifier applicable to local variable - final.



Static modifier - method
Variable
But not for top level class.
But for inner class is allowed.
(Static nested classes)

Case 1) Static variable

```
class Test {
    int x = 10;
    static int y = 20;
    public void m1()
    {
①       Sop(x);   ✓
②       Sop(y);   ✓
    }
    public static void m2()
    {
→③      Sop(x);   ✗  Ck can not access instance member from static area.
④       Sop(y);   ✓
    }
```

Case 2)

```
class Test {
    public static void main(String[] args)
    {
        Sop(String[]);
    }
    public static void main(int[] args)  ⟹ Need to call explicitly
    {
        Sop(int[]);
    }
}
```

Method Overloading

Static method → possible overloading.

(case 3)

```
class P {
  psv main (String[] args)
  {
    sop("parent main");
  }
}
class C extends P
{

}
```

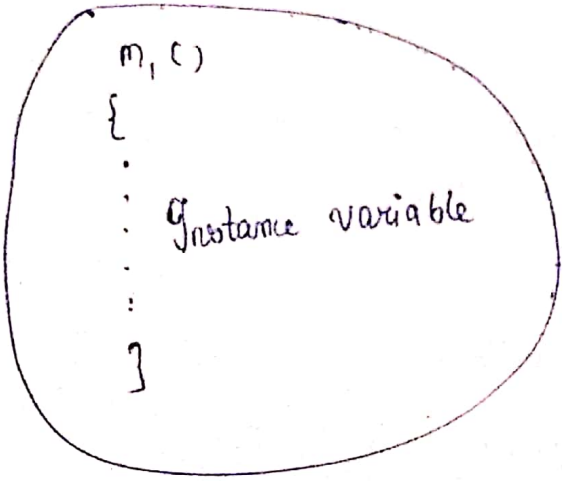
① Javac P.java
o/p parent main

② Javac C.java
o/p parent main

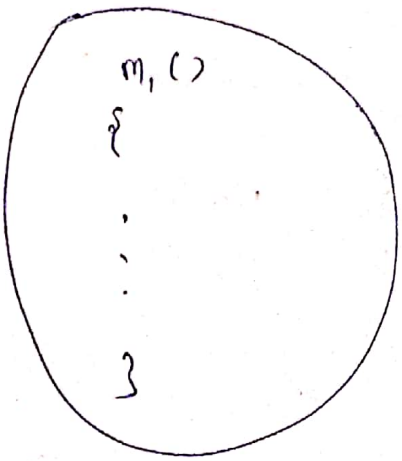
Static → is also applicable for inheritance.

→ For static → Overiding is not possible
instead method hiding is possible.

Instance method

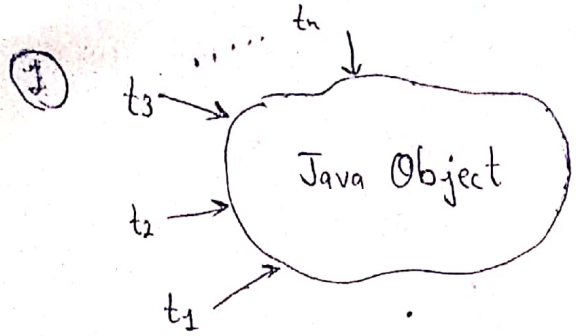


Static method



No instance variable.

Synchronized Modifier → Method / Blocks / But not for classes & variables.



Data Inconsistency problem.

At a time one thread is allowed.

Performance degrade.

Native modifier → Method → We can't apply anywhere else.

The method which are implement in non-java.

Area — performance - to improve performance of system.
└→ programming friendly languag → device driver
OS
H/w connection.

Eg [public native int hashcode()]

To use legacy non-java code.

```
class Native{
    static{
        System.loadLibrary("path");
    }
    public native void m1();
}
```

// Compulsory native library must be loaded before class loading

① load Native library

② Declare a native method

③ Invoke a native method

Internally JNI require dll. file mapped.

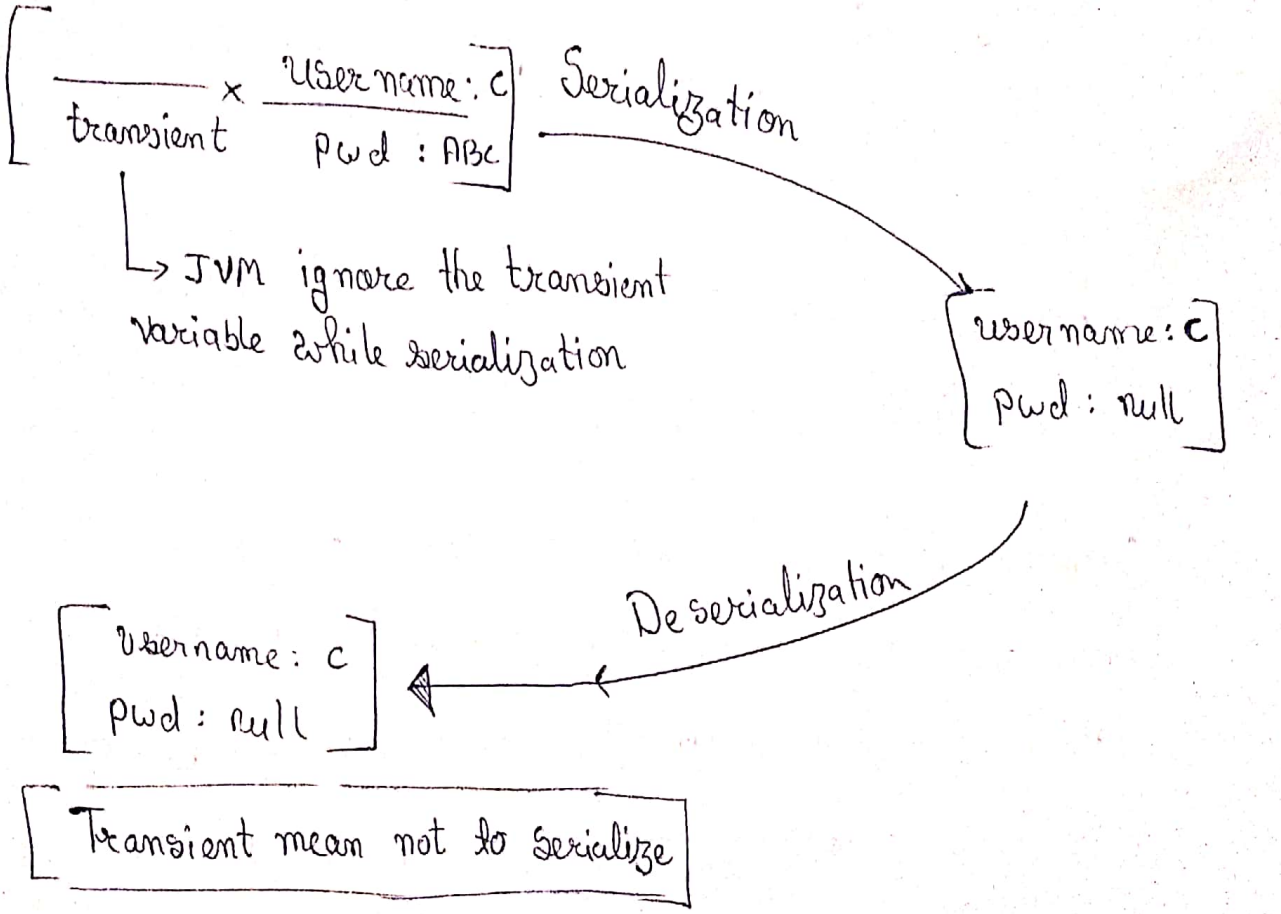
For abstract, implementation is not available

For native, implementation is already available in non-java.

→ We can't declare strictfp, native beoz other might be not support Non-java language.

→ Transient Keyword: (Variables)

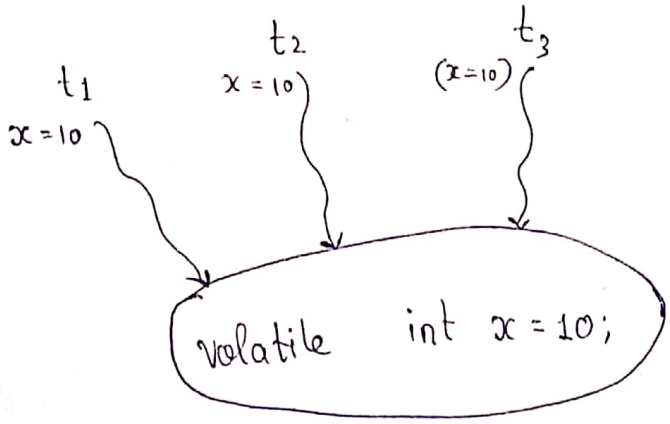
- Serialization context
  - State of an object → [Save, transfer,]
- Deserialization context → [Read state of an object]



[Transient mean not to Serialize]

volatile (variables) (child - ball) - problem.

↳ RAM



→ value of variable x keep on changing multiple threads there is a chance of data inconsistency problem.

→ Then for every thread JVM will create a separate local copy. Every modification perform by thread will take place in local. No effect on remaining thread.

→ Creating & maintaing every local copy for thread memory issue & complexity increase.

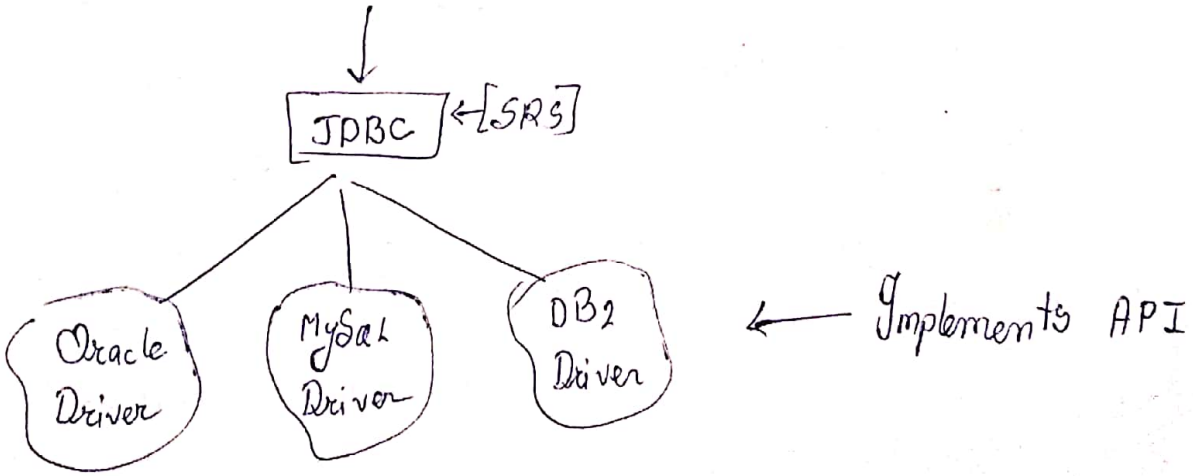
→ [Depricated]

** → volatile, final → Illegal access of modifier combination.

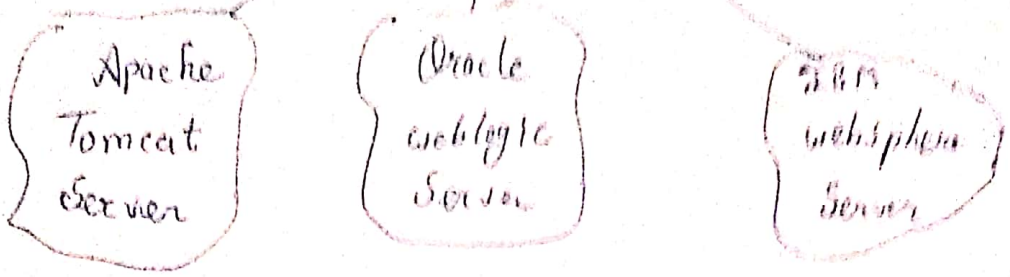
Interfaces

1. Introduction
2. Declaration & implementation
3. Extends vs implements
4. Interface methods
5. Interface variables
6. Interface naming Conflicts.
   - (i) Method naming conflicts
   - (ii) Variable naming conflicts
7. Marker interface
8. Adapter interface
9. interface vs Abstract class vs Concrete class
10. Difference b/w interface, Abstract class.
11. Conclusion.

Interface
- Requirement Specification.
- Service requirement specification. (SRS)



[Servlet API] — API's to develop web server



Interface

→ Every method is always abstract whether we are declaring or not.

→ 100% pure abstract class.

② → Declaration & Implementation

↳ (2 mistake)

```
Interface interf
{
    void m1();
    void m2();
}
```

```
Abstract class S implements interf
{
    public void m1(){
    }
}

class SS extends S
{
    public void m2(){
    }
}
```

① In S class, m_2 is not implemented. So write as Abstract.

② In interface method $m_1()$ in implemented class its scope must increase. public

→ Extends , Implements.

Class – extends – one class

Class – implements – many interface.

Interface – extends – many interface.

class – extends – one class – implements – many interfaces

⇒ Interface method

↳ Every method in interface public, Abstract always.

Why →

```
interface Interf
{
    void m1();
}
```

→ public : To make this method available to every implementation class.

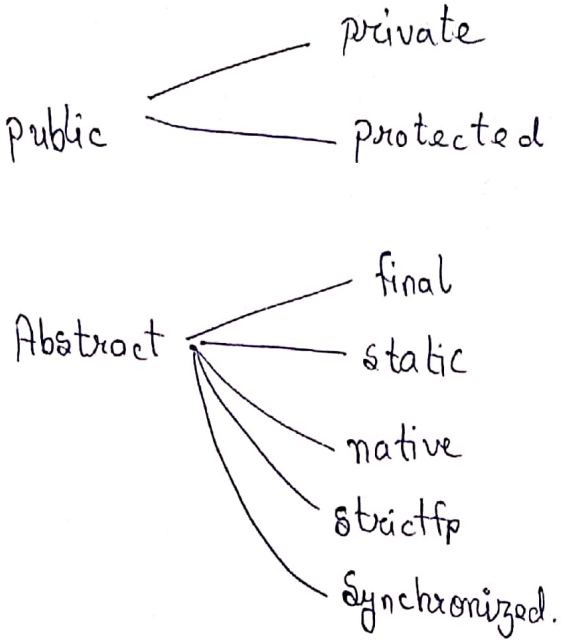
→ Abstract : Implementation class is responsible to provide implementation.

Inside interface - the method declaration are equal.

Equal

```
void m1();
Public void m1();
abstract void m1();
Public abstract void m1();
```

Every interface method follows:
is always public, Abstract.



Illegal combination of access modifier.

Interface variable : requirement level constant.

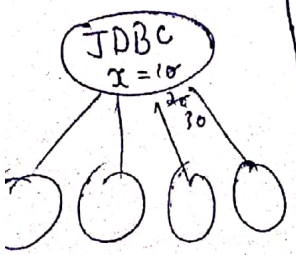
↳ Every interface variable is always
Public, static, final

```
Interface interf
{
    int x=10
}
```

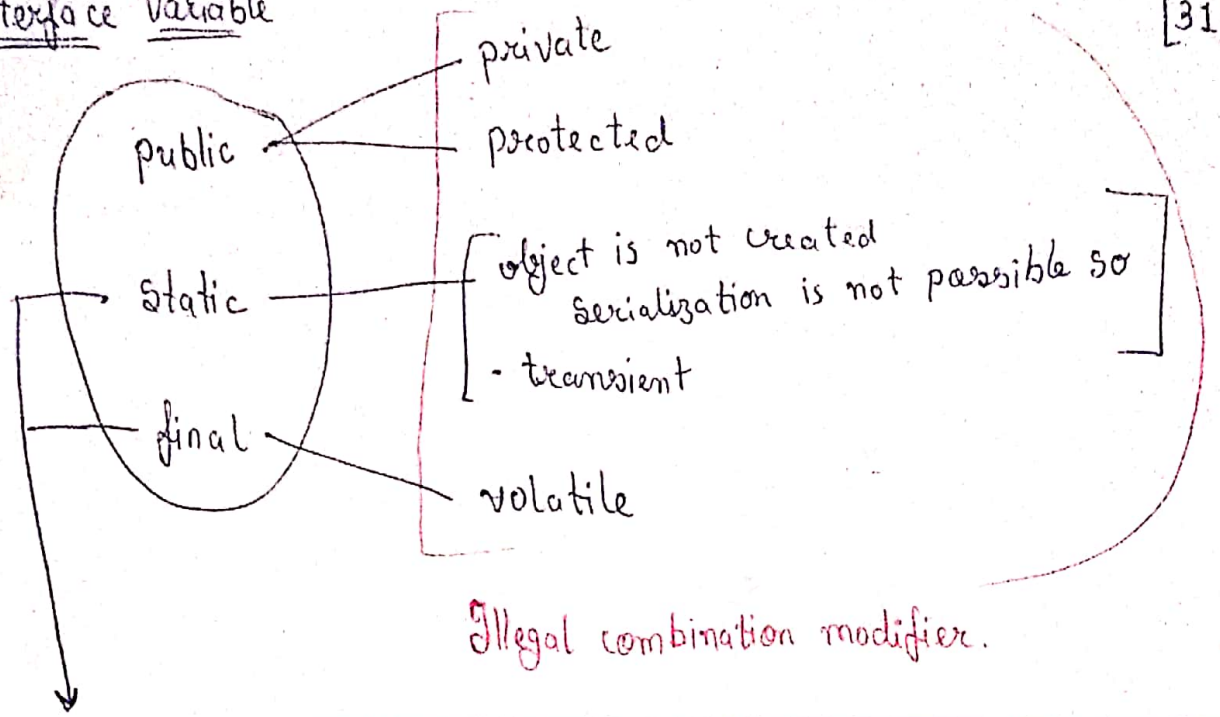
→ public - to make this variable available to every implementation class.

→ Static - Not possible to create object for interface.
- Without existing object, it can access this variable.

→ final - If one implementation class change its variable then remaining implem.. class affected.



Interface Variable



Note - final static variable - it should be intialize before class loading.

Interface variable - must need to ~~init~~ intidize, at time of declaration.

Method Naming Conflicts:-

Case-1

```
interface left
{
    public void m1();
}
```

```
interface Right
{
    public void m1();
}
```

```
Class test implements left, right
{
    public void m1()
    {

    }
}
```

No Conflict

Case-2

```
interface left
{
  public void m1();
}
```

```
interface right
{
  public void m1(int n);
}
```

```
class test implements left, right
{
    public void m1()
    {
    }
    public void m1 (int i)
    {
    }
}
```

Overload methods

Note Case-3

```
interface left
{
  public void m1();
}
```

```
interface right
{
  public int m1();
}
```

```
class test implements left, right
{
```

Not possible to implements both interface method m1 in same class bcoz of same signature

JVM confused throws error.

```
}
```

→ Same signature but different return types.

Marker interface

Ability → Tag

└→ An interface which does not contain any methods by implementing that interface our object get some ability.

1. Serializable (I)
2. Cloneable (I)
3. Random Access (I)
4. SingleThreadModel (I)

These are marked for some ability.

Adapter Class - trick approach to not to implement all interface method.

Note :- Abstract class contain zero abstract method.
If our class don't want to create object.

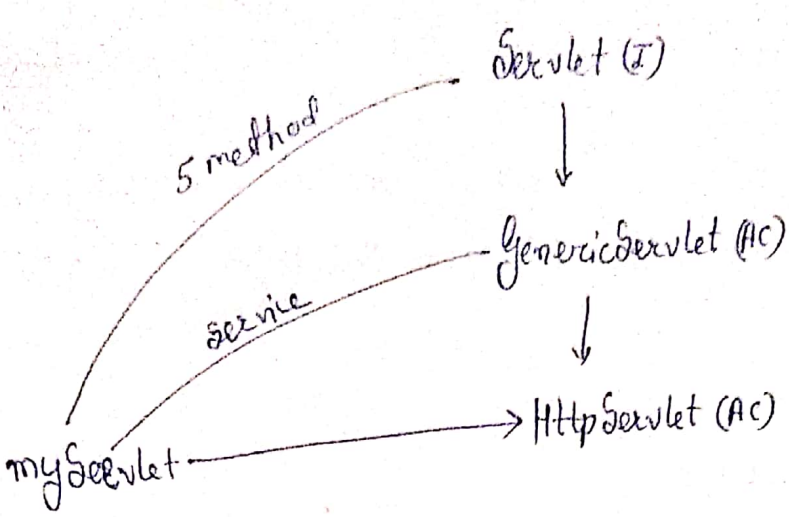
```
Interface interf
{
    m1();
    m2();
    m3();
    .
    .
    m100();
}
```

```
Abstract adpater implements interf
{
    m1(){ }
    m2(){ }
    m3(){ }
    m4(){ }
    m5(){ }
    .
    .
    m100{ }
}
```

```
class x extends adapter
{
    m1()
    {
    ..
    .
    .
    }
}
class y extends adapter
{
    m3()
    {
    .
    .
    }
}
```

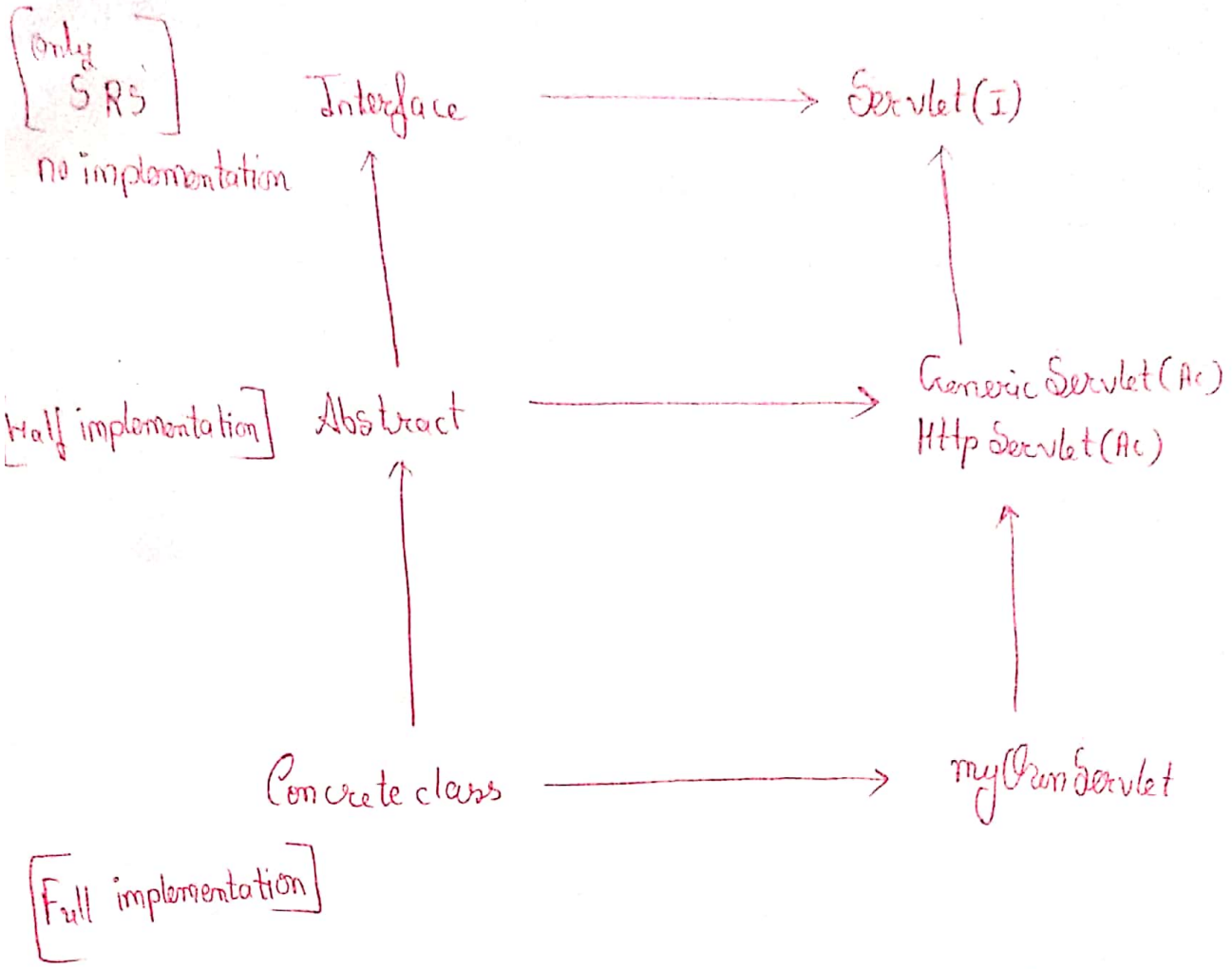
As such adapter class not given any kind of implementation.

need to give abstract in adapter class.



init ()
service()
destroy ()
getServletConfig()
getServletInfo()

Interface / Abstract / Concrete class.



Interface	Abstract
No implementation	Half implementation
method: public abstract don't pure abstract class	method: need to be public, static It can be concrete
No method contain private, protected, synchronized, native final, static	No restriction method modifier
Variable must be public, static, final	No restriction variable modifier.
No Variable contain private, protected, volatile, transient	No restriction on variable modifier.
Compulsory initialization of variable at the time of declaration.	No restriction.
No static block instance block	Yes
No constructor	Yes

→ We cannot create a object for abstract class but it contain Constructor

↳ Need : It will executed only when child class object created.

↳ To reduce code of lines.

The main purpose of constructor is to perform intialization of instance variables,

↳ Abstract class can contain instance variable which are required for child object to perform intialization of those instance variable, constructor is required for abstract class

↳ But every variable present inside interface is always public static final wheather we are declaring or not and there is no chance of existing instance variable inside interface. Hence constructor concept is not required for interface.

```
class P {
    P()
    {
     Sout(this.hashCode());
    }
}
class C extends P {
      C()
      {
       Sout(this.hashCode());
      }
}
```

```
class Test
{
 psvm(String[] args)
 {
    C c = new C();
    Sout(c.hashCode());
 }
}
```

All has a same hashCode();

→ Inside interface every method is always abstract

→ In abstract class only abstract method.

→ Difference between interface with abstract class.
We can replace but it is not good programming language.

abstract class x
{
 	
}
	interface x
	{
	
	}

class test extends X	class test implements x
① Inheritance available not	① Inheritance ~~is~~ available
② test t = new test(); more time to Create due to parent constructor	② test t = new test(); less time to create object ~~less~~ no constructor.

→ new - to create an object

constructor - to initialize an object

↳ When child class object is created then child & parent constructor executed.

↳ Either directly or Indirectly we can't create object for abstract class but Abstract class contain constructor.

↳ Whenever we are creating child object automatically abstract class constructor will be executed to perform initialization of child object for the ~~per~~ properties whether inheriting from Abstract class (Code Reusability)