Declarations and Access Modifiers

- **Java source file structure**
-
- **Class modifiers**
-
- **Member modifiers**
-
- **Interfaces**

Java Source File Structure

- A java program can contain any no of classes but at most one class can be declared as the public if there is a public class the name of the program and name of public class must be matched otherwise we will get compile time error.
- If there is no public class then we can use any name as java source file name,There are no restrictions.

Example 1:

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

Save sri.java

Case1:

If there is no public class then we can use only name as java source file name.

Example 1:

A.java

B.java

C.java

Durga .java

Case2:

If class B declared as public and the program name is A.java Then we will get compile time error saying

“Class B is public should be declared in a file named B.java”

Case3:

If we declare both Aand B classes as public and name of the program is B.java then we will get Compile time error saying

“Class A is public should be declared in a file named A.java”

Example 1:

```
class A
{
public static void main(String[] args)
{
    System.out.println(“A class main method”);
}
}
class B
{
public static void main(String[] args)
{
    System.out.println(“B class main method”);
}
}
```

```
class C
{
public static void main(String[] args)
{
    System.out.println(“C class main method”);
}
}

class D

{

}
```

Save –Durga.java

$$\begin{tikzcd}
 & \text{Javac Durga.java} \arrow[dl] \arrow[d] \arrow[dr] \arrow[drr] & & \\
\text{A.class} & \text{B.class} & \text{C.class} & \text{D.class}
\end{tikzcd}$$

1) java A ↵

   Aclass main method

2) java B ↵

   B class main method

3) java C ↵

   C class main method

4) java D ↵

   Run time error-NoSuchMethodError:main

5) java Durga ↵

   Run time error-NoClassDefFindError:Durga

Note:

-It is highly recommended to take only one class per source file and name of the file and that classname must be matched.This approach improves readability of the code.

Import statement:-

Example 1:

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

Output=

```
Compile time error-Can not find Symbol.
                    Symbol:class ArrayList
                    Location:class Test
```

-We can resolve this problem by using fully qualified name.

Java.util

-The problem with usage of fully Qualified name every time increases length of the code and reduces readability.

-We can resolve the problem by using import statement.

Example 1:

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

-Whenever we are using import statement it is not required to use fully QualifiedName hence it reduces improves readability and reduces length of code.

Case1:

Types of import statement:-

-There are 2 types of import statements.

1)Explicit class import

2)Implicit class import

$$\begin{tikzcd} & \text{Import statement} \arrow[dl] \arrow[dr] & \\ \text{Explicit class import} & & \text{Explicit class import} \end{tikzcd}$$

Explicit class import

Example:

import java.util.ArrayList;

-This type of import is highly Recommended to use.Because It improves readability of the code.

Explicit class import

Example:

import java.util.*;

-It is never recommended to use this type of import because it reduce readability of the code.

Case2:

Difference between #include and import Statement:-

-In c language #include all the specified header files will be loaded at the time of include statement only irrespective of weather we are using those header files are not.Hence this is static loading.

-But in the case of java language import statement no class file will loaded at the time of import statement,in the next line of code whenever we are loading a class at the time only the corresponding .class file will be loaded,This type of loading is called dynamic loading or bad on demand or load on fly.

Case 3:

Which of the following import statement are valid

```
1)import java.util;                //Invalid
2)import java.util.ArrayList.*;    //Invalid
3)import java.util.*;              //valid
4)import java.util.ArrayList;      //valid
```

Case 4:

-Consider the code

```
  class MyRemoteObject extends Java.rmi.UniCastRemoteObject
  {
   }
```

-The code compile even though we are not using import statement because we use FullyQualifiedName.

Example 1:

```
import  java.util.*;
import  java.sql.*;
 class Test
{
public static void main(String[] args)
{

Date d=new Date();

//Compile time error-Reference to Date is ambiguous

}
}
```

Note:

Even in list case we will get the same ambiguity problem because it is available in both util and sql packages.

Case 5:

Example 1:

```
import  java.util.Date;
import  java.sql.*;
 class Test
{
public static void main(String[] args)
{

Date d=new Date();

 }

}
```

Conclusion:

While resolving class names compiler will always gives in the following order.

Case6:

-Whenever we are importing a package all classes and interfaces present in that package are available ,but not subpackage classes.

Example 1:

$$\begin{tikzcd}
\text{Java} \arrow[d] & & \\
\text{util package} \arrow[dr] & & \\
& \text{regex Sub packages} \arrow[d] & \\
& \text{pattern} &
\end{tikzcd}$$

-To use pattern class which of the following import is required.

1)import java.*; //Invalid

2)import java.util.*; //Invalid

3)import java.util.regex.*; //valid

4)import java.regex.pattern; //valid

Case 7:

-The following two packages are not required to import because all classes and interfaces present in these two packages are available by default to every java program

1)java.lang package.

2)default package(current working directory)

Case 8:

Static import:

- This concept introduced in 1.5 version.
- According to SUN Static import improves readability of the code,But according to worldwide programming experts (Like as)static imports reduces.The readability of the code and confusion.it is not recommended to use static import if there is no specific requirement.
- Usally we can access static members by using class names ,but whenever we are using static import,it is not required to use class name and we can access static members directly.

Example 1:

Wihout static import

```
class Test
{
public static void main(String[] args)
{
  System.out.println(Math.sqrt(4));
  System.out.println(Math.random());
  System.out.println(Math.max(10,20));
  }
}
```

Example 2:

With static import

```
        import static java.lang.Math.sqrt;
        import static java.lang.Math.*;
```

```
public class staticimport {
   public static void main(String ar[])
{
  System.out.println(sqrt(4));
  System.out.println(random());
  System.out.println(max(10,20));
  }
}
```

Output:

```
2.0
0.10269078484273308
20
```

- **Explain about System.out.println():**

Example:

```
public  class sop {
    static  String name="xyz";

  int a=sop.name.length();

}
```

sop:I t is class name.

name:static variable present in test class of the type string.

length:It is method present in string class.

```
public class system {

static PrintStream out;

   }

 System.out.println();
```

System:It is class present in java.lang package.

out:it is static variable of type printstream present in system class.

println:It is method presenr in printstream class.

Explaination:

- Out is static variable present in system class hwnce we can access by using classname.

- But whenever we are using static import it is not required to use class name we can access out variable directly.
  Example:

```
*/
import static java.lang.System.out;
public class importstatic {
   public static void main(String ar[])

   {
   out.println("Hello");
   out.println("Hi");
   }
}
```

Output:

```
Hello
Hi
```

  - **-ve perspective(Ambiuity):**

```
import static java.lang.Integer.*;
import static java.lang.Byte.*;
public class Ambiguity {
   public static void main(String ar[])
   {
   System.out.println(MAX_VALUE);
   }

}
C.E.:Reference to MAX-VALUE in ambiguity
```

Note:

Two classes contains a variable or method with same name is very common hence ambiguity problem is also very common in static import.

- While resolving static members compiler will always gives precedence in the following oreder.
  1)Current class static member
  2)Explicite static member
  3)Implicit static member.

  Example:

```
import static java.lang.Integer.MAX_VALUE;  ——————————  (2)
import static java.lang.Byte.*;  ——————————(3)

public class prioryty {
   static  int MAX_VALUE=999;  ——————————   (1)
   public static void main(String ar[])
   {
   System.out.println(MAX_VALUE);
   }

}
```

  Output:

```
999
```

  If we are commenting line (1) then explicit static import will get priority hence we will gwt Integer class MAX-VALUE is output.

  Example:

```
import static java.lang.Integer.MAX_VALUE;  ——————————(2)
import static java.lang.Byte.*;  ——————————     (3)

public class prioryty {
  //static  int MAX_VALUE=999;  ——————————  (1)
   public static void main(String ar[])
   {
   System.out.println(MAX_VALUE);
   }
}
Output:
 2147483647
```

If We are commenting line 1and 3 then Byte class MAX-VALUE will be considered and we will get 127 as output.

Example:

```
//import static java.lang.Integer.MAX_VALUE;  ————————(2)
import static java.lang.Byte.*;   ———————— (3)

public class prioryty {
   //static  int MAX_VALUE=999;    ————————   (1)
   public static void main(String ar[])
   {
   System.out.println(MAX_VALUE);
   }

}
```

Note:

Strictly speaking usage of class name to access static variables and methods improves readability of the code .Hence it is not recommended to use static imports.

Qu)Which of the following import statements are valid

```
import java.lang.math.*;          //We should not use * after the class

import java.lang.math.sqrt.*;     //We should not use * after the method
import static java.lang.math;            //Invalid

import static  java.lang .math.sqrt();   //Invalid

import static  java.lang .math.sqrt;      //Valid
```

- **normal static Vs.static import:**
  1)WE can use normal import to import classes and interfaces of a package.Whenever we are using general import it is not required to use fullyQualifiedNmae and we can use short names directly.
  2)We can use static import to import static variables and methods of a class.

PACKAGES

Defination:

It is an encapsulation mechanism to group related classes and interfaces into a single module.The main purpose of package are

1)To resolve naming conflict.

2)To provide security to the classes and interfaces.So that outside person can’t access directly.

3)It improves modularity of the application.

There is one universally accepted convention to name packages i.e. to use internet domain name in reverse.

$$\begin{tikzcd}
\underline{\text{com.icic.bank}}.\arrow[d] & \underline{\text{loan}}.\arrow[d] & \underline{\text{housingloan}}.\arrow[d] & \underline{\text{Account}}\arrow[d] \\
\text{domain name in reverse} & \text{module name} & \text{submodule name} & \text{classname}
\end{tikzcd}$$

Example:

package com.cstjobs.itjobs;

```
public class cst

{

Public static void main(String ar[])

{

System out.println("Getting jobs");

}

}
```

Output:

```
Getting jobs
```

- Generated class file will be placed into corresponding package Structure:

$$\begin{tikzcd}
\text{cwd} \arrow[dr] & & & & \\
& \text{com} \arrow[dr] & & & \\
& & \text{cstjobs} \arrow[dr] & & \\
& & & \text{itjobs} \arrow[dr] & \\
& & & & \text{cst.class}
\end{tikzcd}$$

Conclusion:

1)In any java program there should be any at most 1 package statement.If we are taking more than one package statement we will get compile time error.

Example:

```
package pack1;
package pack2;                    //C.E.class,interfaceor enum expected

class A{

}
```

2)In any java program the first non comment statement should be package statement(if it is available).

Example:

```
import java.util.*;

package pack1;

                    //C.E.class,interface or enum expected

class A

{

}
```

…….

Class Modifiers

- Whenever we are writing our own java class compulsory we have to provide some information about our class to the JVM like,
    1) Whether our class can be accessible from anywhere or not.
    2) Whether child class creation is possible for our class or not.
    3) Whether instantiation is possible or not etc.
- We can specify this information by declaring with appropriate modifier.
- The only applicable modifiers for top-level classes are
    1) public
    2) <default>
    3) final
    4) abstract
    5) strictfp
- If we are using any other modifier we will get compile-time error saying that modifier not allowed.

  Example:

```
private class test
  {
    Public static void main (String args[])
     {
      int x = 0;
      for (int y=0;y<3;y++)
      {
       x=x+y;    // C.E. – modifier private not allowed here
      }
       System.out.println(x);
     }
}
```

- But for the inner classes the following modifiers are allowed
    1) public
    2) <default>
    3) final
    4) abstract
    5) strictfp
    6) private
    7) protected
    8) static

➢ Access Specifiers V/s Access Modifiers:

- In old languages like C and C++ public, private, protected and default are considered as access specifiers and all the remaining like final, static are considered as access modifiers.
- But in java there is no such type of division all are considered as access modifiers.

➢ public classes:

- If a class declared as the public then we can access that class from anywhere.
  Example:
    1)
```
package pack1;
public class A
{
 public void m1()
{
System.out.println("Hello");
}
}
```

```
        2) package pack2;
           Import pack1.A;
           class B
            {
             Public static void main(String args[])
             {
      A a = new A();
        a.m1();
       }
    }
Output: Hello
```

- If we are not declaring class A as public, then we will get compile time error while compiling B class, saying “pack1.A is not public in pack1, can’t be accessed from outside package”.

➢ **default classes:**

- If a class declared as default then we can access that class only within that current package. i.e. from outside of the package. We can’t access.

➢ **Final Modifier:**

- Final is the modifier applicable for classes, method and variables.
- If a method declared as the final then we are not allowed to override that method in the child class.
  Example:

```
  class p
 {
  public void property()
  {
 System.out.println(“Money+Gold+Land”);
   }
 public final void marry()
```

```
  {
System.out.println("Jayshree");
 }
}
class c extends p
{
public void marry()  // C.E. – marry in c cannot override marry()
                                                in p; override method is final.
 {
System.out.println("Kajal|35ba|tara");
 }
}
```

- If a class declared as the final then we can't create child class.
  Example:-

```
final class p
{

}
class c extend p
{
        // Invalid
}
```

- If we declare as final then this class accessible only the within the pack.
- If we declare class as public final or final public, this class can be accessible in another package.
- Every method present inside a final class is always final by default but every variable present in final class need not be final.
- The main advantage of final keyword is we are missing key benefits of Oop's inheritance and polymorphism (overriding). Hence, if there is no specific requirement never recommended to use final keyword.

➢ **Abstract Modifier:**

- Abstract is the modifier applicable for classes and methods but not variables.

➢ Abstract Method:

- Even though we don't know about implementation still we can declare a method with abstract modifier. i.e. abstract method can have only declaration but not implementation. Hence, every abstract method declaration should compulsory ends with " ; ".

  Example:

  1) public abstract void m1(){ }   // Invalid
  2) public abstract void m2();   //Valid

- Child classes are responsible to provide implementation for parent class abstract methods.

  Example:

```
abstract class Vehicle
   {
      public abstract int getNoOfWheels();
   }
class Bus extends Vehicle
   {
      public int getNoOfWheels()
   {
      return 6;
   }
   }
class Auto extends Vehicle
   {
      public int getNoOfWheels()
   {
      return 3;
   }
}
```

- By declaring abstract methods in parent class we can define guidelines to the child classes which describes the methods those we to be compulsory implemented by child class.
- Abstract modifier never talks about implementation, if any modifier talk about implementation than it is always illegal combination with abstract.
- The following are various illegal combinations of modifiers for methods.
- abstract (method)- final
  static
  synchronized
  native
  strictfp
  private

➢ **Abstract class:**

- For any java class if we don't want instantiation. Then we have to declare that class as abstract. i.e. for abstract classes instantiation (creation of object) is not possible.

  Example:

```
 abstract class Test
{

}
Test t = new Test();//Output= C.E. Test is abstract; cannot be
                                        Instantiated
```

Abstract class Vs abstract method

If a class contains atleast one abstract method then compulsory that class should be declared as abstract otherwise we will get compile time error. Because, The implementation ius not compile & hence we can't create an object.

Eventhough class does not contain any abstract method still we can declare the class as abstract.i.e.abstract class can contain zero ‘0’ no. of abstract method.

Example:HttpServelet ,This class doesn’t contain any abstract method but still it is declared as abstract.

Ex-

```
1)class Test
  {
   public void m1();
   }
2) class Test
  {
   public abstract void m1();
   {
   }
   }  C.E.-abstract method can’t have as body
 3)class Test
  {
   public abstract void m1();
   }
```

C.E-Test is not abstract and dosen’t override abstract method m1() in Test.

Example-

```
          abstract class Test
  {
```

```
 public abstract void m1();
public abstract void m2();
}
class subtest extends Test
 {
  public abstract void m1()
 {
 }
}
```

C.E.-SubTest is not abstract & does not override abstract method m2() in Test.

We can handle these compile time error either by declaring class subtest as abstract or by providing implementation for m2().

Note-

The usage of abstract methods ,abstract class & interfaces are recommended & it is always good programming practice.

Abstract Vs final:

Abstract method we have to override in child classes to provide implementation.where as final method can't be overrider.Hence abstract final combination is illegal combination method.

For abstract classes we should create child classes to provide proper implementation but for final classes we can't create child class.Hence abstract final combination is illegal for classes.

Final class can't have abstract methods where as abstract class canb contain final method.

Example:

```
1) final class A
 {
   abstract void m1();
   }                                  //Invalid


2) abstract class A
   {
    public final void m1();
    {
    }
    }                                 //Valid
```

Strictfp(all loewer case)Modifier (Strict floating pointing):

Strictfp is the modifier applicable for method & classes but not for variables.

If a method declared as strictfp all floating point calculation in that method has to follow IEEE 754 standard, so that we will get platform independent results.
Strictfp method al;ways talks about implementation where as abstract method never talks about implementation.Hence strictfp abstract method combination is illegal combination for method.

If a Class declared as strictfp then every concreate method in that class has to follow IEEE 754 Standard So That we will get flatform independent results.

Abstract –strictfp combination is illegal for classes but illegal for methods.

Example:

```
abstract strictfb classTest
 {
 }
 public abstract Strictfp void m1();    //Invalid
```

Member (Variables & Methods)Modifiers:

1)public members :

If We declare a member as public then we can access that member from anywhere but corresponding class should be visible(public). i.e. Before checking member visibility we have to check visibility.

Example:

1) package pack1;

```
class A
{
 public void m1()
 {
  System.out.println("Hi");
 }
 }
```

2) package pack2;

```
import pack.A;
class B
{
```

```
public static void main(String args[])
{
 A a=new A();
 a.m1();
}
}                        //Invalid
```

2)default members :

If a declare a member as default then we can access that member only with in the current package & we can't access from outside of the package. Hence ,default access is also package.level access.

3)private members :

If a declare a member as private then we can access that member only wihin current class.

Abstract method should be visible in child classes to provide implementation.Where as private method are not visible in child classes.Hence private abstract combination is illegal for methods.

4)protected members (The most misunderstood modifier in java):

If a declare a member as protected then we can access that member within .The current package anywhere but outside package only in child classes.

protected=<default>+kids of an another package(only child reference)

Within the current package we can access protected member either by parent reference or by child reference.

But from outside package we can access protected member only by using child reference.if we are trying to use parent reference we will get C.E.

Example 1:

```
 package pack1;
public class A
{
 protected  void m1()
 {
  System.out.println(“The most misunderstood modifier in java”);
 }
 }
  claas B extends A
  {
  public static void main(String args[])
   {
    A a=new A()              //The most restricted modifier is ‘private’
   a.m1();
    B b=new B()              // The most accessible modifier is ‘public’
    b.m1();
    A a1=new B()             //private <default<protected <public
    a.m1();
```

Example 2:

```
 package pack1;
import pack1.A;
 public class C extends A
 {
 public static void main(String args[])
  {
  A a=new A()
  a.m1();                      // Invalid
  C c=new C()
   c.m1();                     //Valid
   A a1=new C()
   a1 .m1();                        //Invalid
   }
   }
```

Visibility	private	<default>	protected	public
1)within the same class	accessible	accessible	accessible	accessible
2)From chlild class of some package	Not accessible	accessible	accessible	accessible
3)From non child class of same package	Not accessible	accessible	accessible	accessible
4)from child class of outside package	Not accessible	Not accessible	accessible	accessible
5) From non child class of outside package	Not accessible	Not accessible	Not accessible	accessible

final Variables:

In general for instance & static variables it is not required to perform initialization explicitly Jvm will always provide default values.

But for the local variable jvm wan't to provide any default values compulsory we should provide initialization before using that variable.

final instance Variables:

For the normal instance variable it is not require perform initialization explicitly Jvm will always provide default values.

If the instance variable declared as the final then compulsory we should perform initialization whether we are using or not otherwise.

We will get compile time error

Example 1:

```
class Test
{
 int x;
 }
```

Example 2:

```
class Test
{
 final int x;
 }
```

C.E.-variable x might not have been initialized.

Role:

For the final instance variables we should perform initialization before constructor compilation.

i.e.-The following are various places for this ,

1)At the time of declaration

```
Ex.   class Test
         {
          final int x=10;
         }
```

2)inside instance block

```
Ex.   class Test
         {
          final int x;
         {
          x=10;        //instance block
         }
          }
```

3)inside constructor

```
Ex.   class Test
         {
          final int x;
          Test()


         {
           x=10;
          }
           }
```

Other than these if we are perform initialization any where else we get compile time error.

Example:

```
          class Test
           {
            final int x;
             public void m1()
             {
              x=0;
               }
               }
```

C.E.-can not assign a value to final variable x.

final static variables:

For the normal static variable it is not required to perform initialization explicitely ,jvm will always provide default values.

But for final static variable we should perform initialization explicitly otherwise we will get C.E.

Example 1:

```
class Test
 {
   Static int x;
    }              //Valid
```

Example 1:

```
class Test
 {
   final Static int x;
    }               //Not valid
```

C.E-Variable x might not have been initialized

Role:

For the final static variables we should perform initialization before constructor compilation.

i.e The following are various places to perform this.

1)At the time of declaration

Example 1:

```
class Test
 {
```

```
          final Static int x=10;
           }
```

2)Inside static block

Example 1:

```
        class Test
         {
          final Static int x;
           static
            {
             x=10;
              }
           }
```

If we are performing initialization any where else we will yet compile time error.

Example 1:

```
        class Test
         {
          final Static int x;
           public void m1()
            {
             x=10;
              }
              }
```

C.E-Can’t assign value to final variable x.

3)Final local variable:

For the local variable jvm wan’t to provide any default values compulsory we should perform initialization before using that variable.

Example 1:

```
class Test
 {
   public static  void main()
    {
     int x;
      System.out.println(“Hello”)
      }
      }
```

O/P-Hello

Example 2:

```
class Test
 {
   public static  void main()
    {
     int x;
      System.out.println(x)
```

```
            }
            }
```

C.E-variable x might not have been initialized.

Eventhough local variable declared as the final it is not required to perform initialization if we are not using that variable.

Example 1:

```
        class Test
         {
           public static  void main()
            {
              final int x;
              System.out.println(“Hello”)
              }
              }
```

O/P-Hello

The only applicable modifier for local variable is final.if we are using any other modifier we will get compile time error.

Example 1:

```
        class Test
         {
           public static  void main()
            {
             public int x=10;            //Not valid
            private int x=20;         // Not valid

            static int x=60;          // Not valid

            protected int x=30;     // Not valid

            final int x=40;          // valid

              }

              }
```

Formal parameter of a method simply access as local variable of that method.hence a formal parameter can declared as final.

If we declare a formal parameter as final within the method we can't change its value otherwise we will get compile time error.

Example 1:

```
        class Test

         {

           public static  void main(String args[])

            {

            m1(10,20);      //actual parameter

           }

           public static  void m1(final int x,int y)   //formal parameter

           {

           x=1000;

           y=2000;

          System.out.println(x+"……"+y);

              }
```

}

Static modifier:

Static is the modifier applicable for variable & methods but not for classes (but inner class can be declared as static).

If the value of a variable is varied from object to object. Then we should go for instance variable.In the case of instance value for every object a separate copy will be created.

If the value of a variable is same for all objects.Then we should go for static variables.In the case of static variable only one copy will be created at class level & share.That copy for every object of that class.

native modifier :-

- native is the modifies applicable only for methods but not for variable and classes
- The native methods are implemented in some other languages like C & C++ hence native methods also known as “foreigen methods”.
- The main objectives of native keyword are
  1)To improve performance of the system.
  2)To use already existing lagecy non-java code.

- Psudo Code.
  To use native keyword:

Example:

```
public class NativeMethod {

  static
```

```
    {
    System.loadLibrary("natiive library");
    }
    public  native void m1();
    }
class child
{
    public static void main(String[] args) {
     NativeMethod n=new NativeMethod();
      n.m1();// TODO code application logic here
  }
}.
```

- For native methods implementation is already available in other language and we are not responsible to provide implementation.Hence native method declaration should compulsary end with “semicolon”(“;”).

- Example1:

```
class test
{
public native void m1()
{

}                              //native methods can’t have body.
}
```

Example2:

public native void m1();

Note:

1)For native methods implementation should be available in some others languages where as for abstract methods implementation should not be available hence abstract-native combination is illegal for methods.

2)native methods cannot be declared with strictfp modifier because there no guarantee that old language follows IEEE754 standard.Hence abstract-nativ combination is illegal for methods.

Disadvantage:

The main disadvantage of native keyword is it breaks platform independent nature of java because we are depending on result of platform dependemt languages.

- **synchronized modifier:**
- Synchronized is the modifier applicable for methods and blocks.
- We can't declare class and variable with this keyword.
- If method or block declared as synchronized then at a time only one thread is allowed to pperate on the given object.
- Main advantage of synchronized keyword is it can resolve data inconsistency.But the main disadvantage of synchronized keyword is it

increase waiting time of thread and effects performance of the system.Hence if there is no specific requirement it is never recommended to use synchronized keyword.

transient modifier:

- Transient is the modifier applicable only for variables and we can't apply for method and classes.
- At the time of serialization,if we don't want to save the value of a particular variable to meet security constraint,then we should go for transient keyword.
- At the time of serializationJVM ignores the original value of transient variable and default value will be serialization.

- **volatile modifier:**

- Transient is the modifier applicable only for variables but not for methods and classes.
- If the value of avariable keep on changing such type of variables we have to declare with volatile modifier.
- If variable declared as volatile then for every thread a separate local copy will be created.
- Every intermediate modification performed by that thread will takes place in local copy instead of master copy.
- Once the value got finalized just before terminating the thrad the master copy value will be updated with local state value.
- The main advantage of volatile keyword is we can resolve data inconsistency problems.
- But the main disadvantage of volatile keyword is that creating and maintaining a separate copy for every thread increases complexity of the programing and effects performance of the system.Hence ,if there is no specific requirement it is never recommended to use volatile keyword and it is almost outdated keyword.

Volatile variable means its value keep on changes where as final variable means value never changes.Hence final-volatile combination is illegal for variables.

Conclusion:
1)The only applicable modifier for local variable is final.
The modifier which are applicable only for local variable is variable ,but not for classes and methods are volatile and transient.
2)The modifier which are applicable only for methods but not for classes and variables native and transient.
3)The modifier which are applicable only for method but not for classes and variables native and synchronized.
4)The modifier ehich are applicable for top level classes,methods and variables are public default and final.

Interface:

- Any service requirement specification(srs) is considered as interface.
- From the client point of view an interface defines set of services what is expecting.
- From the programmer point of view interface define the set of services what is offering.
- Hence an interface considered as contract between client and service provider.
  Example:By using Bank Atm Gui Screen ,Bank people will highlight the set of services what they are offering same time.The same time screen describe the set of services what end user expected.Hence this gui screen acts as contract between client and service provider.
- With in the interface we can't write any implementation,because it has to highlight just the set of services what we are offering or what you are expecting.Hence every method present inside interface should be abstract.Due to this interface is considered as 100% pure abstract class.

  - What is an Interface:
- Any service requirement specification(SRS) (or) any contract between clientand service provider(or)100% pure abstract call is nothng but an interface.
- The main Advantage of interface are:
  1)we can achive security because we are nothighlighting our internal implementation.
  2)Enhancementwill become very easy,because without effecting outside person we can change our internal implementation.
  3)Two different system cen communicate via interface
  [A java application can tall with mainframe system through interface]

- Declaratoin amd Implementation of interface:
  - WE can declare an interface by using Interface keyword,We can implement an interface by using implements keyword.
  - Example:

    ```
    Interface inter
    {
    void m1();
    void m2();
    }
    Abstract class ServiceProvider implements Intrer
    {
    Public void m1();
    {
    }
    }
    ```

    If class implements an interface compulsory we should provide implementation for every mthods of that interface.Otherwise we have to declare clss as object.Violatin leads to compile_time Error.

- extends Vs implements:

  1)A class can extend only one class at a time.

  2)A class can implement no. of interfaces ata time.

  3)A class can extend a class and implement interface simulteneously.

  4)An interface can extend any no. of interfaces at atime.

  Example 1:

  ```
   interface A
  {
  }
  interface B
  {
  }
  interface C extends A,B
  ```

```
{
}
```

Interface methods:

- Wheather we are declaring or not, every interface method is by default public and abstract.
  Example:

  ```
   interface interf
  {
   Void m1();            //method is public
  }
  ```

- public-
  To make this method availability for every implementation class.
- abstract-
  because interface methods specifies requirements but not implementation.

  Hemce the following method declarations are equal inside interface.
  1)void m1();
  2)public void m1();
  3)abstract void m1();
  4)public abstract void m1();

- As every interface method is by default public abstract.
- The following modifier are not applicable for interface methods:
  1)private 2)protected 3)<default> 4)final

5)static 6)strictfp 7)synchronized 8)native

Qu)Which of the following method declaration are valid inside interface?

1)public void m1() {} //invalid

2)public static void m1(); //invalid

3)public synchronized void m1(); //invalid

4)private abstract void m1(); //invalid

5)public abstract void m1(); //valid

Interface variable:

- An interface can contain variables.The main purpose of the variable is to specify

  - Constant at requirement level:
    Every interface variable is always public,static,final wheather we are declaring (or) not.

    ```
    interface inter
    {
    int x=10;
    }
    ```

    public:
    To make this variable available for every implementation available class.
    static:
    without existing object also implementation class can access this variable.
    final:

implementation class can access these variable but can,t modify.

- Hence inside interface the following declaration are valid and equal.
  1)int x=10;
  2)public int x=10;
  3)public static int x=10;
  4)public static final int x=10;
  5)final int x=10;
  6)public final int x=10;
  8)static final int x=10;

- As interface variable are public static and final we can,t declare with the following modifier:
  1) private2)default3)volatile4)protected 5)transient
- For the following variable compulsory you should perform initialization at the time of declaration only otherwise you will get compile time error.

```
interface intrf
{
 int x;          //C.E:initialsation expected
}
```

Qu)Which of the following variable declation are allowed inside interface.

1)int x=10;   //Valid

2)int x; //Invalid

3)private int x=10;   //Invalid

4)public int x=10;//Valid

5)transient int x=10;//Invalid

6)volatile int x=10;   //Invalid

7)public static final int x=10;   //Valid

- Inside implementation classes we can access interface variables but we can,t modify there values.

  Example:

  - Interface:

```
public interface interfacevariable
 {
    int x=10;
 }
```

  - Class implements interface:

```
public class interf implements interfacevariable

             {

  public static void main(String ar[])

 {

             int x=888;

            System.out.println(x);

           }

}
```

Output:888

Interface naming conflicts:

1)Method naming conflicts:

Case1:

If two interfaces contains a methods with same signature and same return type in the implementation class we can provide implementation for only one method.

Example:

Interface left:

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

Interface right:

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

Class classleftright:

```
public class classleftright implements left,right{
    public  void m1() {
        System.out.println("jai");
    }
    public static void main(String ar[])
    {
    classleftright obj=new classleftright();
    obj.m1();
    }   }
```

Output:

```
Jai
```

Case2:

If two interfaces contains method with same name but different args then,in the implementation class we have to provide implementation for both method and these methods are consider as overloaded methods.

Example:

- interface left:

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

- interface right:

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

- class classleftrightoverload:

```
public class classleftrightoverload  implements left,right
   public void m1()
{
      System.out.println("jai");
 }
 public void m1(int x)
  {
         System.out.println(x);
 }
  public static void main(String ar[])
  {
 classleftrightoverload obj=new classleftrightoverload();
  obj.m1();
  obj.m1(10);
       }
}
```

Output:

```
jai
10
```

Case3:

If two interfaces contains a method with same signature but different return type,then it is impossible to implement both interfaces at a time.

1)We can,t write any java class which implements both interfaces simultaneously.

Example:

interface left1:

```
public interface left1 {
    public void m1();
}
```

interface right1:

```
public interface right1
 {
    public int m1();
}
```

Class classleftrightreturntype:

```
public class classleftrightreturntype implements left1{
    public void m1() {
       System.out.println("cst");
    }
    public static void main(String ar[])
    {
      classleftrightreturntype obj=new classleftrightreturntype();
      obj.m1();
    }
}
```

Output:

cst

Qu)Is it possible a java class can implements any no,of interfaces simultaneously?

Ans:yes,Except if two interfaces contains a method with same signature but different return types.

2)Variable naming conflicts:

Example:

- interface left2:

```
public interface left2

{

   int x=88;

}
```

- Interface righ2:

```
public interface right2

{

   int x=888;

}
```

- class classleft2right2variable:

```
public class classleft2right2variable implements left2,right2{
    public static void main(String ar[])
        {
 System.out.println(x);        //C.E:reference to x is
                                 ambiguous
           }
}
```

Note:

There may be a chance of 2 interfaces contains available with same name and may rise variable naming conflicts but we can resolve these naming conflicts by using interfaces names.

```
System.out.println(left2.x);              //88
System.out.println(right2.x) ;            //888
```

Marker interface:

If an interface won,t contain any method and by implementing that interface if own object will yet ability such type of interfaces are called marker interface (or) Tag interface(or ability) interface.

Ex:serializable,clonable,RandomAccess,single Threadinterface

These interfaces are method from some sbility.

Example:

1)By implementing serializable interface we can send object across the n/w and we can save state of object to a file.This extra ability is provided through serializable interfaceour object..

2)By implementing clonnable interface our object will be in position to provide exactly duplicate object.

Qu 1)marker interface won't contain any method then how the object will get that spcialibility?

Ans:JVM is responsible to provide required ability in marker interface.

Qu 2)Why JVM is providing required ability in marker interface?

Ans:To reduce complexity of the programming.

Qu 3)Is it possible to create our own marker interface?

Ans:Yes,But customization of JVM is required.

Example:sleepable,Eatable,Jumpable,Lovable,Funnable

Adapter class

Adapter class is a simple java class that implements an interface,an interface only with empty implementation.

Example:

```
interface x

{

m1();

m1();


-

-

-

m1000();

}


Abstract class adapterx implements x

{

m1(){}

m2(){}

m3(){}

-

-

-
m1000{}

}
```

If we implement an interface directly compulasary we should implementation foer every method and that interface wheather we are interested or not and wheather it is required(or) not.If increases length of the code ,so that readability will be reduced.

Example:

```
interface interfacex

{

public void add();

    public void sub();

    public void mul();

    public void div();

    public void disp();

}
```

class abstract class adapterx:

```
public abstract class adapter implements interfacex{

int a=10,b=12;


    @Override

    public void add() {

       System.out.println("add:"+(a+b));
    }
    @Override
    public void sub() {
          System.out.println("sub:"+(a-b));
    }
    @Override
    public void mul() {
          System.out.println("mul:"+(a*b));
}
abstract public void div();
   abstract public void disp();


 }
```

class simple :

```
public  class simple extends adapterx{
  public void disp()
  {
  System.out.println(a);
  System.out.println(b);
  }
  public void div() {
          System.out.println("div:"+(a/b));
```

```
    }
public static void main(String ar[])
{
adapterx obj1=new adapterx();
obj1.disp();
obj1.mul();
}
}
```

Output:

```
10
12
mul:120
```

- If we extends adapter class instead of implementation of interface directly then we have to provide implementation of only for required method but not all this approach reduce lengthof the codeand improve readability.

concrete class Vs. abstract class Vs. interface:

- interface:We don,t know anything about implementation just we have requirement specification,then we should go for interface.
  Ex. Servlet
- Abstract class:We are talking about implementation but not completely (just partially implementation) then we should go for abstract class.
  Ex.Generic servlet,HTTP servlet.
- Concrete class:We are talking about implementation completely and ready to provide service,then we should go for concrete class.
  Ex.our own class

Difference between interface and abstract class

	Interface	Abstract class
1	1)If we don't know any thing about implementation just we have requirement specification,then we should go for interface.	1) If we are talking about about implementation but not completely(partial implementation)then we should go for abstract class.
	2)Every method present inside interface is by default public and abstract	2) Every method present inside iabstract class need not be public and abstract.We can take concrete methods also.
2	3)The following modifier are not allowed for interface methods:strictfp,protected,static,native,private,final,synchronized	3)There are no restrictions for abstract class method modifier i.e. we can use any modifier.
3.	4)Every variable present inside is public,static,final,bydefault wheather we are declare or not.	4)abstract class variable need not be public,final,static.
4.	5)for interface variables we can,t declare the following modifier private,protected,transient,volatile,	5) There are no restriction for abstract class variable modifier

5.	6)For the interface variable compulasry we should perform initialization at the time of declaration only.	6) For the abstract class variable there is no restriction like performing initialization at the time of declararation.
6.	7)Inside interface we can't take any constructor.	7) Inside interface we can't take any constructor.
7.	8)Inside interface we can't take instance and static blocks.	8)Inside abstract class we can take static and instance block