**Java Access Modifiers – Public, Private, Protected & Default**

Access Modifiers is the way of specifying the accessibility of a class and its members with respective to other classes and members.

**Access Modifiers for Top-level Classes & Interfaces**: public, default, abstract, final  
**Access Modifiers for Members**: public Members, protected Members, default Members, private Members, static Members, final Members, abstract Methods, synchronized Methods, native Methods, transient Fields, volatile Fields.  
**Access Modifiers for Nested Classes & Interfaces:** Nested Interfaces, Nested Classes, Static member classes, Non-Static member classes, Local classes, Anonymous classes.

**Access Modifiers Types**

The basic Accessibility Modifiers are of 4 types in Java. They are

1. public
2. protected
3. package/default
4. private

There are other Modifiers in Java. They are

1. static
2. abstract
3. final
4. synchronized
5. transient
6. native
7. volatile

**Access Modifiers are used at 2 access levels in Java. They are**

1. Top-level for Classes & Interfaces
2. Member-level for Classes & Interfaces

1) Access Modifiers for Top-level Classes & Interfaces

Only 2 basic access modifiers are applicable for Top-level Classes & Interfaces. They are Public and Package/Default modifiers.

* Public:  If top level class or interface within a package is declared as Public, then it is accessible both inside and outside of the package.
* Default**:** If no access modifier is specified in the declaration of the top level class or interface, then it is accessible only within package level. It is not accessible in other packages or sub packages.

**Example**: In the below examples I have used few notations to make you understand better. for example I have named default class as CDef where Crepresents class and Def is for “default” like wise for default interface I usedIDef, I for interface and Def for “default”. Similarly IPub for public interface and cPubObj for public class reference.

// top-level interface declaration with public modifier

public interface IPub {…}

package pack1;

// top-level class declaration with default modifier

class CDef {….}

// top-level interface declaration with default modifier

interface CDef {…}

package pack1;

// another class in same package Pack1

class A1 {

// public Class is accessible within the package

CPub cPubObj;

// default Class is accessible within the package

CDef cDefObj;

}

package pack1;

// default Interface is accessible within the package

class B1 implements IDef {…}

package pack1;

// public Interface is accessible within the package

class C1 implements IPub {…}

package Pack2;

// Class in other package Pack1

class A2 {

// public Class is accessible in other package

CPub cPubObj;

// default Class is NOT accessible in other package

CDef cDefObj;

}

package pack2;

// default Interface is NOT accessible outside the package

class B2 implements IDef {…}

package pack2;

// public Interface is accessible outside the package

class C2 implements IPub {…}

2) Accessibility Modifiers for Members:

public Members: If members are declared as public inside a class then such members are accessible to the classes which are inside and outside of the package where this class is visible. This is the least restrictive of all the accessibility modifiers.

protected Members:If members are declared as protected then these are accessible to all classes in the package and to all subclasses of its class in any package where this class is visible.

Default Members: When no accessibility modifier is specified for the member then implicitly it is declared as Default. These are accessible only to the other classes in the class’s package.

private Members:This is the most restrictive of all accessibility modifiers. These members are accessible only with in the same class. These are not accessible from any other class within a class’s package also.

Example:

package pack2;

import pack1.cPub;

public Class subcPub2 extends cPub {

vPub = 10; // public variable is accessible in subclass of another package

vPro = 10; // protected variable is accessible in subclass of another package

vDef = 10; // default variable is not accessible in subclass of another package

vPri = 10; // private variable is not accessible in subclass of another package

fPub(); // public method is accessible in subclass of another package

fPro(); // protected method is accessible in subclass of another package

fDef(); // default method is not accessible in subclass of another package

fPri(); // private method is not accessible in subclass of another package

}

package pack2;

import pack1.cPub;

public Class cPubpack2 {

cPub subcPubObj = new cPub();

subcPubObj.vPub = 10; // public variable is accessible in another class of another package

subcPubObj.vPro = 10; // protected variable is not accessible in another class of another package

subcPubObj.vDef = 10; // default variable is not accessible in another class of another package

subcPubObj.vPri = 10; // private variable is not accessible in another class of another package

subcPubObj.fPub(); // public method is accessible in another class of another package

subcPubObj.fPro(); // protected method is not accessible in another class of another package

subcPubObj.fDef(); // default method is not accessible in another class of another package

subcPubObj.fPri(); // private method is not accessible in another class of another package

}

**Static Members:**

* Static members are the members with Static keyword in their declarations.
* Class variables are called as Static variables. These members belong to the class not to the object i.e. they are not instantiated when the class instance is created.
* The values of these variables are not part of the object state.
* The static variables are initialized to their default values (if explicit initialization is not specified) at the time of class loading.
* The Static methods are called as class methods. A static method can directly access other static members in the class. It cannot access instance (non-static) members of the class. But it can always use a reference of the class type to access its members both static and non-static.

**Final Members:**

* Final variable is a constant; its value cannot be changed after its initialization.
* This applies to instance, static and local variables including parameters that are declared as final.
* A final variable of primitive data type cannot change its value once it has been initialized. A final variable of a reference type cannot change its reference value once it has been initialized, but the state of the object it denotes can still be changed.

**Note:**

* Variables defined in Interfaces are implicitly Final.
* Final variables must be initialized before it is used.
* Final methods in a class are complete i.e. these methods has implementations and hence cannot be overridden in the subclasses.

class cDef { // normal class declaration

int vInt; // instance variable declaration

static int vIntS; // static variable declaration

final int vIntF=10; // final instance variable declaration

final static int vIntFS = 10; // final static variable declaration

public static void fPubStatic() { // static method declaration

vIntS= 10; // static variable is accessible

vInt =10; // non-static variable or instance variable is not accessible

}

final public void fPubFinal() { // final method declaration

vInt = 10; // instance variable is accessible

}

public void fPub() { // normal instance method declaration

final int vIntFLoc = 10; // final local variable declaration

vIntF = 20; // modifying the final instance variable value is not possible

vIntS= 20; // static variable is not accessible in non-static method

vIntFS = 20; // modifying the final static variable value is not possible

vIntFLoc =20; // modifying the final local variable value is not possible

}

}

public class csubDef extends cDef{ //subclass declaration

public void fPubFinal() {…} // overriding the final method in subclass is not possible

}

public class cMain { // Main class declaration

public static void main (Strings[] args) { // main method declaration

cDef.fPubStatic(); // static method access using Class name

System.out.println(“static variable using class name” + cDef.vIntS);

// static variable access using Class name

cDef cDefObj =new cDef(); // creation of class instance

cDefObj.fPubStatic(); // static method access using reference

System.out.println(“static variable using reference ” + cDef.vIntS);// static variable access using reference

final cDefFinObj =new cDef(); // A final reference variable declaration

cDefFinObj = new cDef(); // modifying the final reference variable value is not possible

}

}

**Abstract Methods:**

If method has a keyword *abstract* in its declaration, then such method/function is called Abstract method. Abstract methods does not have an implementation i.e. method body is not defined; only method prototype is specified in the class definition.

**Note:**

* Only instance methods can be declared as *abstract*.
* Since Static methods cannot be overridden declaring abstract static method would of no use.
* A Final method cannot be abstract and vice versa.
* Methods specified in an Interface are implicitly *abstract*.

**Synchronized Methods:**

Multiple threads can be executing in a program and at times they might try to execute several methods on the same object simultaneously.

If there is a requirement that only one thread at a time should execute a method in the object, then such methods can be declared as *Synchronized*. Their execution will be mutually exclusive among all threads. At any given time, at the most one thread will be executing a Synchronized method on an object.

**Note:** Synchronized methods are also applicable to Static methods of a class.

**Native Methods:**

Native Methods are also called as Foreign methods. Such methods implementation is not defined in Java but in another programming language.

These methods are specified in the class as method prototypes with prefix with keyword native, no method body is defined in the Java class.

class cNat() {

native void fNat(); // native method declaration

synchronized public void fSyn() {…}; // synchronized method declaration

}

class cmain {

public static void main (String[] args) {

cNat cNatObj =new cNat(); // class instance creation

cNatObj.fNAt(); // accessing native method

}

}

**Transient Fields:**

Objects can be stored using serialization. Serialization transforms objects into an output format which is helpful for storing objects. Objects can later be retrieved in the same state as when they were serialized, meaning that fields included in the serialization will have the same values at the time of serialization. Such objects are said to be Persistent.

The fields are declared with keyword Transient in their class declaration if its value should not be saved when objects of the class are written to persistent storage.

class sample implements Serializable{

transient str varStr; // transient field declaration

int varInt; // instance field declaration

}

**Volatile Fields:**

During execution, complied code might cache the values of fields for efficiency reasons. And as multiple threads will access the same field, caching is not allowed to cause inconsistencies when reading and writing the value in the field.

The *Volatile* modifier can be used to inform the compiler that it should not attempt to perform optimizations on the field which could cause unpredicted results when the field is accessed by multiple threads

class sample {

volatile int varInt; // volatile field declaration

}

3) Accessibility Modifiers for Nested Classes & Interfaces

**Nested Interfaces:**

Access modifiers can be used in Nested interfaces. An interface declared within another class or interface is called a Nested interface. A top-level interface is the one which is not nested.

Only one type of nested interface is available in Java based on declarative context, Static member interface. These are interfaces defined with keyword Static inside the top-level interface or class or in another Static member class or interface. It can be instantiated like a normal top-level interface or class, no enclosing instance is required for this interface instantiation.

**Nested Classes:**

Access modifiers are also used in Nested classes. A class declared within another class is called a Nested class. A top-level class is the one which is not nested.

They are 2 categories of Nested classes based on the declarative context. They are

* Static member classes
* Inner classes

Inner classes are defined in 3 different categories. They are

* Non-static member classes
* Local classes
* Anonymous classes

**Static member classes:**

Classes which are defined with Static modifier inside the top-level class or other Static member class are called Static member classes. It can be instantiated like a normal top-level class; no enclosing instance is required for this class instantiation.

All the 4 accessibility modifiers i.e. Public, Protected, Package & Private are applicable to Static member classes’ declaration.

**Non-Static member classes:**

Classes which are defined without Static modifier inside another class are called non-static member classes. An instance of a non-static member class always has an enclosing instance associated with it.

The accessibility modifiers i.e. Public, Protected, Package & Private,abstract, final are applicable to Non-Static member classes’ declaration.

**Local classes:**

These classes are defined in the context of a block as in a body of the method or normal local block, just as local variables defined in a method body or local block. An instance of a local class has an enclosing instance associated with it, if it is declared in non-static context.

No accessibility modifiers are applicable for Local classes.

**Anonymous classes:**

These are defined as expressions and instantiated on the fly. An instance of anonymous class has an enclosing instance associated with it, if it is declared in non-static context.

No accessibility modifiers are applicable for Anonymous classes.

Note: A Nested Class or Interface cannot have the same name as any of the enclosing classes or interfaces.

class A {                // Top-level Class

static class B {…}       // Static member class

static interface C {…}   // Static member interface

class D {….}             // Non-static inner class

void func () {

class E {…….}           // Local class in non-static context

static void func1 () {

class F {…}              // Local class in static context

}

B bx = new B () {...} // Anonymous class in non static context

static C cx= new C () {……} // Anonymous class in static context

}

The following table shows the access to members permitted by each modifier.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Levels** | | | | |
| **Modifier** | **Class** | **Package** | **Subclass** | **World** |
| public | Y | Y | Y | Y |
| protected | Y | Y | Y | N |
| *no modifier* | Y | Y | N | N |
| private | Y | N | N | N |