**Access Modifiers in Java**

What is Modifier in Java?

A modifier in java is a keyword that we add to those definitions that we need to change their meaning.

In other words, a modifier limits the visibility of classes, fields, constructors, or methods in the Java program.

The functionality of members of a class or a class itself can be protected from other parts of the program by the presence or absence of modifiers.

Java language provides a total of 12 modifiers. They are public, private, protected, default, final, synchronized, abstract, native, strictfp, transient, and volatile.

Twelve modifiers in java can be divided into two categories:

**Access modifiers**

**Non-access modifiers**

Modifiers in Java

Access Modifiers in Java

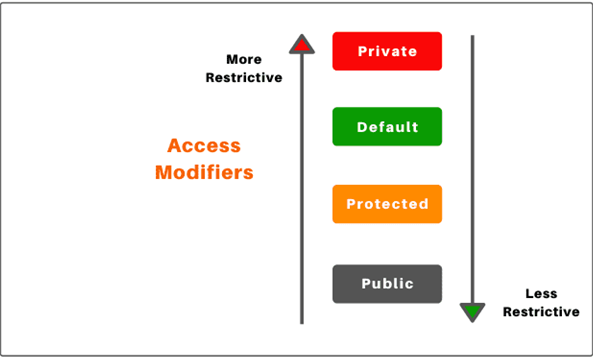
**Public, private, protected and Default**

Access modifiers/specifiers in java define the boundary for accessing members of a class and a class itself.

In other words, access modifiers are those modifiers that are used to restrict the visibility (accessibility) of classes, fields, methods, or constructors.

In short, the accessibility/visibility of data depends on the access modifiers. The access modifiers are also known as visibility modifiers.

Java provides four explicit access modifiers in object-oriented programming languages. They are private, default, protected, and public



Private Access Modifier in Java

There are the following points about private access modifiers that need to keep in mind.

1. Private access modifier in java can apply to a variable, method, constructor, inner class but not the outer class that is class itself.

2. The instance variable can be private but a local variable cannot be private.

3. Private members (field, method, or constructor) of a class cannot be accessed from outside the class. They are accessible only within the class.

4. Private members of a superclass cannot be inherited to the subclass. Therefore, they are not accessible in subclasses.

5. If we make any constructor as private, we cannot create an object of that class from another class and also cannot create the subclass of that class.

6. A class cannot be private except for inner classes. Inner classes are members of the outer class. So, members of the class can be private.

7. If we declare a method as private, it behaves as a method declared as final. We cannot call the private method from outside the class.

Let us create a program where we will create two classes Class A and Class B. In class A, we will declare field and method as private. When we will access this private field and private method from outside the class B, then they will give a compile-time error. Look at the source code to understand better.

Program source code 1:

package modifiersProgram;

class A // Here, Class is default.

{

// Declaration of Instance Variable.

private int data = 30; // Here, instance variable is private.

// Declaration of instance method.

private void msg() // Here, method is also private.

{

System.out.println("Hello Java, this is my first java program");

}

}

Now, create another class B and call private field and method of class A.

class B

{

// Main method.

public static void main(String[] args)

{

// Create an object of class A and call members of class A using reference variable 'a'.

A a = new A(); // a is a reference variable of class A and pointing to the objects of class A.

System.out.println(a.data): // Compile time error will occur because we cannot call private members of any class from outside the class.

a.msg(); // Compile time error. So we cannot call the private method of any class because this is accessible within the class only.

}

}

Role of Private Constructor

If we declare any constructor of a class as private then we cannot create an object of that class from outside the class. In other words, we cannot create the subclass of that class.

Let’s take an example program where we will declare a constructor of class as private and try to create an object of that class. But, we will get a compile-time error.Program source code 2:

package modifiersProgram;

class A

{

// Declaration of Constructor.

private A() // Here, Constructor is declared as private.

{

System.out.println("Constructor is private");

}

// Declaration of instance method.

private void msg() // Method is private.

{

System.out.println("Method is private");

}

}

Now, create another class B and call class A from class B. As you will call class A, it will give compile-time error.

class B

{

// Main method.

public static void main(String args[])

{

// Create the object of class A.

A a = new A(); // Compile time error because the constructor is private and we cannot create the object of class A from outside the class.

}

}

Default Access Modifier in Java

1. When access modifier is not specified to members of a class or a class itself, it is called default access modifier.

2. The default can apply to the instance variable, local variable, constructor, methods, inner class, or outer class.

3. Default members of a class are visible inside of the class and everywhere within classes in the same package or folder only. Therefore, they can be accessed from outside the classes in the same package but can not be accessed outside the package.

4. Default members can be inherited to the subclass within the same package only. It cannot be inherited from outside the package.

Let’s understand the default access modifier or specifier better with the help of an example program.

In this example program, we have created two packages pack1 and pack2. We are accessing class A from outside its package since class A is a default, not public. Therefore, it cannot be accessed from outside the package.

Program source code 3:

package pack1;

class Student

{

// Declaration of Instance variable.

int roll = 12; // Here, instance variable is default.

// Declare the method.

void name() // Here, method has been defined with default access modifier.

{

System.out.println(" Hello Java");

}

}

package pack2;

import pack1.\*;

class College

{

// Main method.

public static void main(String[] args)

{

// Creating an object of class Student from package pack2.

Student obj = new Student(); // Compile time error because class Student has been defined with default access modifier and cannot be instantiated from outside the package. But, if it is declared as public then we can instantiate from outside the package.

obj.name(); // Compile time error because of the default access modifier.

}

}

Protected Access Modifier in Java

1. Protected access modifier can be applied to instance variables, local variables, constructors, methods, inner classes but not the outer class.

2. Protected members are accessible inside the class and everywhere within classes in the same package and outside the package but through inheritance only.

3. Protected members can be inherited to the subclass.

4. If we make constructor as protected then we can create the subclass of that class within the same package but not outside the package

Let’s take an example program to understand the concept of default access modifier.

In this example, we have created two packages pack1 and pack2. The Student class of pack1 package is public. So, it can be accessed from outside the package.

But the name() method of this package is declared as protected. So, it can be accessed from outside the class only through inheritance.

Program source code 4:

package pack1;

public class Student

{

// Declaration of Instance variable.

int roll = 12; // here instance variable is default.

protected void name() // Here method has been defined with protected access modifer.

{

System.out.println("My roll no. is 12");

}

}

package pack2;

import pack1.\*;

class College extends Student

{

public static void main(String args[])

{

// Creating the object of class Student from pack2.

Student obj = new Student(); // Calling the method using reference variable obj.

obj.name();

}

}

Output:

My roll no. is 12.

Public Access Modifier in Java

1. Public access modifier can apply to instance variables, constructors, inner classes, outer class, methods but not with local variables.

2. Public members of a class can be used anywhere.

3. Public members of a class can be inherited to any subclass.

Let’s understand public access modifier with the help of an example program to understand better.

Program source code 4:

package pack1;

public class Student

{

int roll = 12; // Here, instance variable is default.

public void name() // Here, method has been defined with protected access modifier.

{

System.out.println(" My roll no. is 12");

}

}

package pack2;

import pack1.\*;

class College extends Student

{

public static void main(String args[])

{

// Creating the object of class Student from pack2.

Student obj = new Student();

obj.name();

}

}

///////////////////////////////////

Except for these modifiers, all the remaining modifiers available in java are called non-access modifiers. They are abstract, final, native, static.

**Abstract Modifier in Java**

1. Abstract is a keyword that can be applied with outer class, [inner class](https://www.scientecheasy.com/2020/06/inner-class-in-java.html/), method, outer interface, and inner interface. It cannot be applied with variable, constructor, block, and enum.

2. When a class is declared with a keyword abstract, it is called abstract class in java.

3. An abstract keyword cannot be simultaneously declared with the final keyword.

. When a method is declared with an abstract keyword, it is known as abstract method. It contains only a signature and no body. If you declare a method as abstract in a class, the class must be declared as an abstract class.

6. Abstract keyword cannot be declared simultaneously with final, private, native, static, or synchronized.

**Abstraction in Java** is another **OOPs** principle that manages complexity. It is a process of hiding complex internal implementation details from the user and providing only necessary functionality to the users.

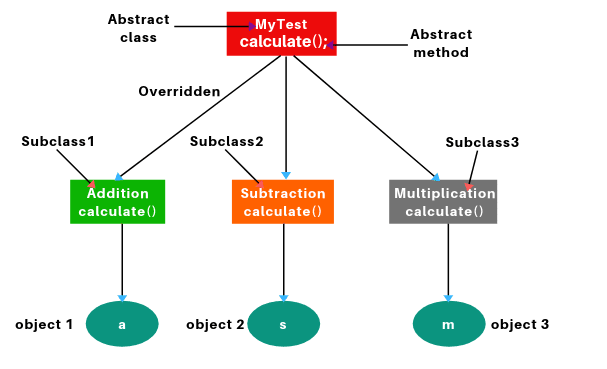
In other words, abstraction in Java is a technique by which we can hide the data that is not required to a user.

**How to achieve Abstraction in Java?**

There are two ways to achieve or implement abstraction in java program. They are as follows:

1. Abstract class (0 to 100%)
2. Interface (100%)

## When to use Abstract method in Java?



Program source code 1:

package com.abstraction;

public abstract class MyTest

{

abstract void calculate(int a, int b); // No body.

}

public class Addition extends MyTest

{

void calculate(int a, int b)

{

int x = a + b;

System.out.println(“Sum: ” +x);

}

}

public class Subtraction extends MyTest

{

void calculate(int a, int b)

{

int y = a - b;

System.out.println(“Subtract: ” +y);

}

}

public class Multiplication extends MyTest

{

void calculate(int a, int b)

{

int z = a \* b;

System.out.println(“Multiply: ” +z);

}

}

public class MyClass

{

public static void main(String[] args)

{

Addition a = new Addition();

Subtraction s = new Subtraction();

Multiplication m = new Multiplication();

a.calculate(20, 30);

s.calculate(10, 5);

m.calculate(10, 20);

}

}

**Features of Abstract class in Java**

There are following important features of abstract class in Java that should be kept in mind. They are as follows:

1. Abstract class is not a pure abstraction in java.

2. In Java, object creation is not possible for an abstract class because it is a partially implemented class, not fully implemented class.

3. It can be abstract even without any abstract method.

4. It can have one or more abstract methods or non-abstract methods (or concrete methods) or combination of both methods.

5. Abstract class allows to define private, final, static and concrete methods. Everything is possible to define in an abstract class as per application requirements.

6. It can have [constructors](https://www.scientecheasy.com/2020/06/constructor-in-java.html/).

7.  Abstract class does not support multiple [inheritance](https://www.scientecheasy.com/2020/07/inheritance-in-java.html/) in java but allows in interfaces.

8. It can implement one or more interfaces in java.

**Rules of Abstract class in Java**

There are the following rules to define an abstract class in Java program. They are as follows:

1. Class must be declared with abstract keyword to make an abstract class.

2. We cannot instantiate an abstract class but we can create object of subclass of the abstract class provided they must implement abstract method.

3. If any method is abstract in a class, the class must be declared as abstract.

4. To use methods declared in an abstract class, the abstract class must be extended by an ordinary class and must implement (override) all abstract methods in that ordinary class.

5. If a new abstract method is added in the abstract class, all non-abstract subclass which extends that abstract class, must implement the newly added abstract method. If it does not implement all the abstract method, the class must be declared as abstract.

6. If a new instance method is added in the abstract class, all non-abstract subclass which extends that abstract class, is not necessary to implement newly added instance method.

7. Inside the abstract class, we can create any number of constructors. If you do not create a constructor, the compiler will create a default constructor.

**Rules of Abstract method in Java**

The rules of abstract method to define in an abstract class are as follows:

1. Abstract method can only be declared in an abstract class.

2. A non-abstract class cannot have an abstract method whether it is inherited or declared in Java.

3. It must not provide a method body/implementation in the abstract class for which it is defined.

4. Method name and signature must be the same as in the abstract class.

5. The visibility of the method in the subclass cannot be reduced while overriding abstract method.

6. Abstract method cannot be static or final.

7. It cannot be private because the abstract method must be implemented in the subclass. If we declare it private, we cannot implement it from outside the class.

Let’s take some example programs based on these rules to understand the abstract class and abstract method concepts more clearly.

Program source code 3:

package com.abstraction;

public abstract class Hello

{

// Declaration of instance method.

public void msg1()

{

System.out.println("msg1-Hello");

}

abstract public void msg2();

}

public class Test extends Hello

{

// Overriding abstract method.

public void msg2()

{

System.out.println("msg2-Test");

}

public static void main(String[] args)

{

// Creating object of subclass Test.

Test obj = new Test();

obj.msg1();

obj.msg2();

}

}

Let’s make a program where an abstract class can have a data member, constructor, abstract, final, static, and instance method (non-abstract method).

Program source code 4:

package com.abstraction;

public abstract class AbstractClass

{

int x = 10; // Data member.

AbstractClass()

{

System.out.println("AbstractClass constructor");

}

final void m1()

{

System.out.println("Final method");

}

void m2()

{

System.out.println("Instance method");

}

static void m3()

{

System.out.println("Static method");

}

abstract void msg();

}

public class AbsTest extends AbstractClass

{

AbsTest()

{

System.out.println("AbsTest class constructor");

}

void msg()

{

System.out.println("Hello Java");

}

public static void main(String[] args)

{

AbsTest t = new AbsTest();

t.msg();

t.m1();

t.m2();

m3();

System.out.println("x = " +t.x);

}

}

fter object creation, the constructor of non-abstract subclass will be called immediately.

In the first line of constructor, internally super will call the constructor of an abstract class. The control of execution will be immediately transferred to the constructor of abstract class.

Therefore, the first output is “AbstractClass constructor”. After executing abstract class constructor, control of execution again comes back to execute subclass constructor. The second output is “AbsTest class constructor”.

**abstract class has constructor even though we cannot create object?**

We cannot create an object of abstract class but we can create an object of subclass of abstract class. When we create an object of subclass of an abstract class, it calls the constructor of subclass.

This subclass constructor has super in the first line that calls constructor of an abstract class. Thus, the constructors of an abstract class are used from constructor of its subclass.

If the abstract class doesn’t have a constructor, a class that extends that abstract class will not get compiled.

public abstract class Employee

{

private String name;

private int id;

public Employee(String name, int id)

{

this.name = name;

this.id = id;

}

// Declaration of concrete method.

void m1()

{

System.out.println("Name: " +name);

System.out.println("Id: " +id);

}

}

public class Engineer extends Employee

{

public Engineer(String name, int id)

{

super(name, id); // This statement is used to call super class constructor.

}

public static void main(String[] args)

{

// Creating an object of the subclass of abstract class.

Engineer e = new Engineer("Deep", 10202); e.m1();

}

}