

# A Closer Look at Methods and Classes

Module 2

# A Closer Look at Methods and Classes:

## Controlling Access to Class Members

Java provides a number of access modifiers to set access levels for classes, variables, methods, and constructors. The four access levels are -

- Visible to the package, the default. No modifiers are Needed.
- Visible to the class only (private).
- Visible to the world (public).
- Visible to the package and all subclasses (protected)

# Default Access Modifier - No Keyword

Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.

A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public.

## Example

Variables and methods can be declared without any modifiers, as in the following examples -

```
String version="1.5.1";
```

```
boolean processOrder()  
{  
    return true;  
}
```

# Private Access Modifier - Private

Methods, variables, and constructors that are declared private can only be accessed within the declared class itself.

Private access modifier is the most restrictive access level. Class and interfaces cannot be private.



Variables that are declared private can be accessed outside the class, if public getter methods are present in the class.

Using the private modifier is the main way that an object encapsulates itself and hides data from the outside world.

## Example

The following class uses private access control –

```
public class Logger {  
    private String format;  
    public String getFormat() {  
        return this. format;  
    }  
    public void setFormat(String format) {  
        this. format - format;  
    }  
}
```



Here, the format variable of the Logger class is private, so there's no way for other classes to retrieve or set its value directly.

So, to make this variable available to the outside world, we defined two public methods:

`getFormat()`, which returns the value of format,

and `setFormat(String)`, which sets its value.



# Public Access Modifier - Public

A class, method, constructor, interface, etc. declared public can be accessed from any other class. Therefore, fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.

However, if the public class we are trying to access is in a different package, then the public class still needs to be imported. Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.

## Example

The following function uses public access control -

```
public static void main(String[] arguments) {
```

```
    //...
```


```
}
```

The main() method of an application has to be public.

Otherwise, it could not be called by a Java interpreter (such as java) to run the class.

# Protected Access Modifier - Protected

Variables, methods, and constructors, which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.



The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.

Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.

Example :

The following parent class uses protected access control, to allow its child class override openSpeaker() method –

```
class AudioPlayer {  
    protected boolean openSpeaker(Speaker sp) {  
        // implementation details  
    }  
}
```

```
class StreamingAudioPlayer extends AudioPlayer {  
    boolean openSpeaker(Speaker sp) {  
        // implementation details  
    }  
}
```

Here, if we define openSpeaker() method as private, then it would not be accessible from any other class other than Audio Player. If we define it as public, then it would become accessible to all the outside world. But our intention is to expose this method to its subclass only, that's why we have used protected modifier


# Access Control and Inheritance

The following rules for inherited methods are enforced

Methods declared public in a superclass also must be public in all subclasses.

Methods declared protected in a superclass must either be protected or public in subclasses; they cannot be private.

Methods declared private are not inherited at all, so there is no rule for them.



Modifiers are keywords that you add to those definitions to change their meanings. Java language has a wide variety of modifiers, including the following –

Java Access Modifiers

Non Access Modifiers

To use a modifier, you include its keyword in the definition of a class, method, or variable. The modifier precedes the rest of the statement, as in the following example.

# Access Control Modifiers

Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors.

The four access levels are –

Visible to the package, the default. No modifiers are needed.

Visible to the class only (private).

Visible to the world (public).

Visible to the package and all subclasses (protected).



# Non-Access Modifiers

Java provides a number of non-access modifiers to achieve many other functionality.

The *static* modifier for creating class methods and variables.


The *final* modifier for finalizing the implementations of classes, methods, and variables.

The *abstract* modifier for creating abstract classes and methods.

The *synchronized* and *volatile* modifiers, which are used for threads.

## Access Levels

Modifier	Class	Package	Subclass	World
<code>public</code>	Y	Y	Y	Y
<code>protected</code>	Y	Y	Y	N
<i>no modifier</i>	Y	Y	N	N
<code>private</code>	Y	N	N	N



The first data column indicates whether the class itself has access to the member defined by the access level. As you can see, a class always has access to its own members.

The second column indicates whether classes in the same package as the class (regardless of their parentage) have access to the member.

The third column indicates whether subclasses of the class declared outside this package have access to the member. The fourth column indicates whether all classes have access to the member.

# Pass Objects to Methods

Object references can be parameters. Call by value is used, but now the value is an object reference.

This reference can be used to access the object and possibly change it. Here is an example program:

# Parameters and Arguments

Information can be passed to methods as parameter.  
Parameters act as variables inside the method.

Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

The following example has a method that takes a String called fname as parameter. When the method is called, we pass along a first name, which is used inside the method to print the full name:

## Example

```
public class Main {  
    static void myMethod (String fname) {  
        System.out.println(fname + " Refsnes");  
    }  
    public static void main(String[] args) {  
        myMethod("Liam");  
        myMethod("Jenny");  
        myMethod("Anja");  
    }  
}  
  
// Liam Refsnes  
// Jenny Refsnes  
// Anja Refsnes
```

# Multiple Parameter

Example

```
public class Main {  
    static void myMethod (String fname, int age) {  
        System.out.println(fname + " Refsnes" + age);  
    }  
    public static void main(String[] args) {  
        myMethod("Liam",5);  
        myMethod("Jenny",8);  
        myMethod("Anja",31);  
    }  
}  
  
// Liam Refsnes 5  
// Jenny Refsnes 8  
// Anja Refsnes 31
```

# Returning Objects from Methods

Like any other datatype, a method can return object.

For example, in the following program, the `makeTwice()` method returns an object in which the value of instance variable is two times than it is in the invoking object.



This program demonstrates how a method can return a reference to an object.

```
public class Sample
{
    private int value;
    public Sample(int i)
    {
        Value = i;
    }
    /*The makeTwice method returns a Sample
       object
    * containing the value twice the passed to it.
    */
    public Sample makeTwice()
    {
        Sample temp=new Sample(value * 2);
        return temp;
    }
}
```

```
public void show()
{
    System.out.println("Value :"+ value);
}

public class ReturnObjectDemo
{
    public static void main(String[] args)
    {
        Sample obj1= new Sample(10);
        Sample obj2;

        // The makeTwice method returns a reference
        obj2 =obj1.makeTwice();
        obj2.show();
    }
}
```

Output:  
Value: 20

# Method Overloading

With method overloading, multiple methods can have the same name with different parameters:

Example

```
int myMethod(int x)
```

```
float myMethod(float x)
```

```
double myMethod(double x, double y)
```

# Constructor Overloading

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

Constructor Overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

//Java program to overload constructors

```
class StudentS {  
    int id;  
    String name;  
    int age;  
    //creating two arg constructor  
    StudentS(int I, String n){  
        id = I;  
        Name=n;  
    }  
    //creating three arg constructor  
    StudentS(int I, String n, int a){  
        id = I;  
        name = n;  
        Age=a;  
    }  
    void display(){
```

```
        System.out.println(id+""+name+""+a  
            ge);}
```

```
    public static void main(String  
        args[]){  
        StudentS s1= new StudentS(111, Kara  
        StudentS s2= new  
            StudentS(222,"Aryan"25);  
        s1.display:  
        s2.display:  
    }  
}
```

```
111 Karan o  
222 Aryan 25
```

There are many differences between constructors and methods. They are given below.

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.

# Recursion

Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method.

It makes the code compact but complex to understand.

## **Syntax:**

```
return type methodname(){  
    //code to be executed  
    methodname();//calling same method  
}
```

## Java Recursion Example 1: Infinite times

```
public class RecursionExample1 {  
    static void p(){  
        System.out.println("hello");  
        P();  
    }  
    public static void main(String[] args) {  
        P();  
    }  
}
```

```
public class RecursionExample4 {  
    static int n1=0,n2=1,n3=0;  
    static void printFibo(int count) {  
        if(count>0){  
            n3 = n1 + n2;  
            n1 = n2;  
            n2 = n3;  
            System.out.print(" "+n3);  
            printFibo(count-1);  
        }  
    }  
    public static void main(String[] args) {  
        int count=15;  
        System.out.print(n1+" "+n2);//printing 0 and 1  
        printFibo(count-2);//n-2 because 2 numbers are already printed
```



# Understanding Static

The static keyword in Java is used for memory management mainly. We can apply static keyword with variable, methods, blocks and nested classes. The static keyword belongs to the class than an instance of the class.

The static can be:

- Variable (also known as a class variable)
- Method (also known as a class method)
- Block
- Nested class

# Java static variable

If you declare any variable as static, it is known as a static variable.

- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.

# Java static method

'If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than the object of a class. A static method can be invoked without the need for creating an instance of a class. A static method can access static data member and can change the value of it.

# Restrictions for the static method

There are two main restrictions for the static method.

They are:

- The static method can not use non static data, member or call non-static method directly.
- this and super cannot be used in static context.

# Java static block

- Is used to initialize the static data member • It is executed before the main method at the time of classloading.
- ## Types of Nested classes

There are two types of nested classes non-static and static nested classes. The non-static nested classes are also known as inner classes,

Non-static nested class (inner class) • Member inner class • Anonymous inner class • Local inner class Static nested class

# Java Inner Classes

- Java inner class or nested class is a class which is declared inside the class or interface.
- We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.
- Additionally, it can access all the members of outer class including private data members and methods.

# Advantage of java inner classes

There are basically three advantages of inner classes in java. They are as follows:

- 1) Nested classes represent a special type of relationship that is it can access all the members (data members and methods) of outer class including private.
- 2) Nested classes are used to develop more readable and maintainable code because it logically group classes and interfaces in one place only.
- 3) Code Optimization: It requires less code to write.

# Varargs: Variable-Length Arguments.

A method with variable length arguments (Varargs) in Java can have zero or multiple arguments. Variable length arguments are most useful when the number of arguments to be passed to the method is not known beforehand. They also reduce the code as overloaded methods are not required.



# Inheritance

## **Why use inheritance in java**

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

# Inheritance

## Terms used in Inheritance

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called derived class, extended class, or child class.

# Super Class/Parent Class

**Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

**Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

In Java, inheritance is achieved using the keyword extends.  
The syntax is given below

```
class A                //super class
{
//members of class A
}
class B extends A      //sub class
{
//members of B
}
```

## Program-1

```
class Employee{  
float salary-40000;  
}  
class Programmer extends Employee{  
int bonus-10000;  
public static void main(String args[]){  
Programmer p=new Programmer();  
System.out.println("Programmer salary is: "+p. salary);  
System.out.println("Bonus of Programmer is: "+p.bonus);  
}}
```

# Type of Inheritance

Single Inheritance: If a class is inherited from one parent class, then it is known as single inheritance. This will be of the form as shown below

Superclass → Subclass

The previous program is an example of single inheritance.

# Multilevel Inheritance

If several classes are inherited one after the other in a hierarchical manner, it is known as multilevel inheritance, as shown below –

**A → B → C → D**

# Why multiple inheritance is not supported in java?

Consider a scenario where A, B, and C are three classes.

The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.



```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){ System.out.println("Welcome");}
}
class C extends A,B
{//suppose if it were
public static void main(String args[]){
C obj=new C();
obj.msg();//Now which msg() method would be invoked?
}
}
```



# Member Access and Inheritance

Inheriting a class does not overrule the private access restriction.

Thus, even though a subclass includes all of the members of its superclass, it cannot access those members of the superclass that have been declared private

# Constructors and Inheritance:

**Constructors** of sub class is invoked when we create the object of subclass, it by default invokes the default constructor of super class. Hence, in inheritance the objects are constructed top-down.

The superclass constructor can be called explicitly using the **super keyword**, but it should be first statement in a constructor.

# Using super to Call Super class constructors:

The super keyword in java is a reference variable that is used to refer parent class objects.

The keyword "super" came into the picture with the concept of Inheritance.

It is majorly used in the following contexts

# Creating a Multilevel Hierarchy:

When a class extends a class, which extends another class then this is called **multilevel inheritance**.

For example class C extends class B and class B extends class A then this type of inheritance is known as multilevel inheritance.

# When are Constructors Executed

Constructor of any class gets any executed only when an object of that class is being created either dynamically using new or locally.

The object memory is allocated, the field variables with initial values are initialized, and then the constructor is called, but its code is executed after the **constructor** code of the object super class. At the byte code level. An object is created but not initialised.

# Order of execution of Initialization blocks and constructor in Java

Static initialization blocks will run whenever the class is loaded first time in JVM

**Initialization blocks** run in the same order in which they appear in the program

**Instance Initialized Blocks** are executed whenever the class is initialized and before constructors are invoked. They are typically placed above the constructors within braces.

# Superclass References and Subclass Objects:

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.



# Usage of Java super Keyword

- super can be used to refer immediate parent class instance variable.
- super can be used to invoke immediate parent class method.
- super() can be used to invoke immediate parent class constructor.

# super example: real use

Let's see the real use of **super** keyword.

Here, Emp class inherits Person class so all the properties of Person will be inherited to Emp by default.

To initialize all the property, we are using parent class constructor from child class.

In such way, we are reusing the parent class constructor.

# Method Overriding

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding

# Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

# Rules for Java Method Overriding

- The method must have the same name as in the parent class.
- The method must have the same parameter as in the parent class.
- There must be an IS-A relationship (inheritance).

# Example of method overriding

In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation.

The name and parameter of the method are the same, and there is IS-A relationship between the classes, so there is method overriding.

# A real example of Java Method Overriding

Consider a scenario where Bank is a class that provides functionality to get the rate of interest.

However, the rate of interest varies according to banks.

For example, SBI, ICICI and AXIS banks could provide 8%, 7%, and 9% rate of interest.

# OVERRIDDEN METHODS SUPPORT POLYMORPHISM


Polymorphism in Java is a concept by which we can perform a single action in different ways.

Polymorphism is derived from 2 Greek words:  
poly and morphs.

The word "poly" means many and "morphs" means forms.

So polymorphism means many forms.





There are two types of polymorphism in Java:  
compile-time polymorphism and runtime  
polymorphism.

We can perform polymorphism in java by method  
overloading and method overriding.

If you overload a static method in Java, it is the example  
of compile time polymorphism. Here, we will focus on  
runtime polymorphism in java.

```
class A{}  
class B extends A{}
```

```
A a=new B();//upcasting
```

For upcasting, we can use the reference variable of class type or an interface type. For Example:

```
interface I{  
class A{  
class B extends A implements I{}
```

Here, the relationship of B class would be:

B IS-A A

B IS-A I

B IS-A Object

# Abstraction in Java

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the object does instead of how it does it.

## Ways to achieve Abstraction

There are two ways to achieve abstraction in java

Abstract class (0 to 100%)

Interface (100%)

# Using Abstract Classes

A class which is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

## Points to Remember

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods,
- It cannot be instantiated
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method

```
abstract class Bike{  
    abstract void run();  
}  
class Honda4 extends Bike{  
    void run(){  
        System.out.println("running safely");  
    }  
    public static void main(String args[]){  
        Bike obj = new Honda4();  
        obj.run();  
    }  
}
```

# Abstract class having constructor, data member and methods

An abstract class can have a data member, abstract method, method body (non-abstract method), constructor, and even main() method.

# Using final

Sometimes you will want to prevent a class from being inherited. To do this, precede the class declaration with `final`. Declaring a class as `final` implicitly declares all of its methods as `final`, too.

As you might expect, it is illegal to declare a class as both `abstract` and `final` since an abstract class is incomplete by itself and relies upon its subclasses to provide complete implementations:

# Final Keyword In Java

The **final** keyword in java is used to restrict the user. The java final keyword can be used in many context.

Final can be:

- variable
- method
- class





The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable.

It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only.

We will have detailed learning of these. Let's first learn the basics of final keyword.

# Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

## Example of final variable

There is a final variable speed limit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

# Java Final Keyword

- Stop Value Change
- Stop Method Overriding
- Stop Inheritance

# Is final method inherited?

➔ Yes, final method is inherited but you cannot override it.

For Example:

```
class Bike
{
    final void run()
    {System.out.println("running...");}
}
class Honda2 extends Bike{
    public static void main(String args)
    {
        new Honda2().run();
    }
}
```

Output:running...

# The Object Class.


1. `toString()`: `toString()` provides String representation of an Object and used to convert an object to String. The default `toString()` method for class Object returns a string consisting of the name of the class of which the object is an instance
2. `hashCode()`: For every object, JVM generates a unique number which is hashCode. It returns distinct integers for distinct objects. A common misconception about this method is that `hashCode()` method returns the address of object, which is not correct. It convert the internal address of object to an integer by using an algorithm.



`getClass()`: Returns the class object of "this" object and used to get actual runtime class of the object

`finalize()` method: This method is called just before an object is garbage collected.

It is called by the Garbage collector on an object when garbage collector determines that there are no more references to the object. We should override `finalize()` method to dispose system resources, perform clean-up activities and minimize memory leaks.

- 
- `clone()`: It returns a new object that is exactly the same as this object
  - `void notify()`: resumes execution of thread waiting on the invoking object