

---

# OBJECT ORIENTED PROGRAMMING USING JAVA



# OUTLINE

- Non-Access Modifier in Java
- Final
- Abstract

---

# JAVA MODIFIERS

- A modifier is a keyword placed in a class, method or variable declaration that changes how it operates.
- There are two types of java modifiers, they are

- **Java Access Modifiers**

- ☐ public
- ☐ private
- ☐ protected
- ☐ Default (No modifiers)

- **Non-Access Modifiers**

- |                                       |                                    |
|---------------------------------------|------------------------------------|
| <input type="checkbox"/> final        | <input type="checkbox"/> strictfp  |
| <input type="checkbox"/> static       | <input type="checkbox"/> native    |
| <input type="checkbox"/> abstract     | <input type="checkbox"/> transient |
| <input type="checkbox"/> synchronized | <input type="checkbox"/> volatile  |

---

# FINAL

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

- Whatever the methods parent has by default available to the child.

**Methods** • If the child is not allowed to override any method, that method we have to declare with final in parent class. That is final methods cannot overridden.

## Class

- If a class declared as the final then we can't create the child class that is inheritance concept is not applicable for final classes.

## Variables

- If a variable declared as the final then we need to look for the type of variables and then correspondingly actions need to be taken (discussed later in the slides).

## EXAMPLE

```
final class Test
{
    void methodOne( )
    {
        System.out.println("Protected Member Access Modifier");
    }
}
```

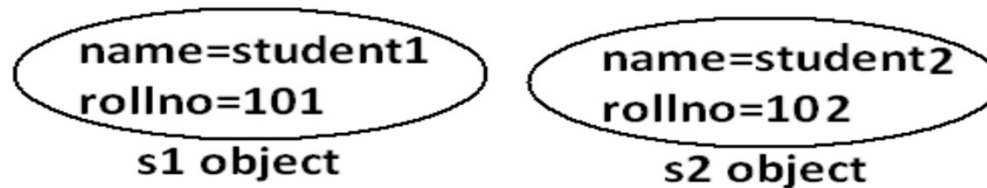
Every method present inside a final class is always final by default whether we are declaring or not. But every variable present inside a final class need not be final.

```
class TestI extends Test
{
    public static void main(String args[])
    {
        Test t=new Test();
        t.methodOne();
        TestI b = new TestI ();
        b.methodOne();
        Test c = new TestI ();
        c.methodOne();
    }
}
```

---

## FINAL VARIABLES IN INSTANCE VARIABLES

- If the value of a variable is varied from object to object such type of variables are called instance variables.
- For every object a separate copy of instance variables will be created.



- For the instance variables it is not required to perform initialization explicitly jvm will always provide default values.

---

## FINAL VARIABLES IN INSTANCE VARIABLES

- If the instance variable declared as the final compulsory we should perform initialization explicitly and JVM won't provide any default values. whether we are using or not otherwise we will get compile time error.

```
class Test
{
    int i;
}
```

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

- For the final instance variables we should perform initialization before constructor completion. That is the following are various possible places for this.
- Initialization while declaration, instance initialization block, and constructor.

## FINAL VARIABLES IN STATIC VARIABLES

- If the static variable declare as final then compulsory we should perform initialization explicitly whether we are using or not otherwise we will get compile time error.(The JVM won't provide any default values)

<pre>class Test {   static int i; }</pre>	valid	<pre>class Test {   final static int i; }</pre>	invalid
---	-------	---	---------

- For the final static variables we should perform initialization before class loading completion otherwise we will get compile time error. That is the following are possible places.
- Initialization while declaration and inside static block.



---

## FINAL VARIABLES IN LOCAL VARIABLES

- To meet temporary requirement of the Programmer sometime we can declare the variable inside a method or block or constructor such type of variables are called local variables.
- For the local variables jvm won't provide any default value compulsory we should perform initialization explicitly before using that variable.
- The only applicable modifier for local variables is final if we are using any other modifier we will get compile time error.

```
class Test
{
    public static void main(String [] args) {
        final int i;
        System.out.println("Bennett");
    }
}
```

---

## FINAL VARIABLES IN LOCAL VARIABLES

```
class Test
{
    public static void main(String args[])
    {
        private int x=10; _____(invalid)
        public int x=10; _____(invalid)
        volatile int x=10; _____(invalid)
        transient int x=10; _____(invalid)
        final int x=10; _____(valid)
    }
}
```

---

## ABSTRACTION IN JAVA

- **Abstraction** is a process of **hiding the implementation details** and **showing only functionality** to the user.
- The extent to which a **module hides its internal data and other implementation details** from the other modules is the most important factor that distinguishes a well-designed Object-Oriented module from other modules.
- It **shows only important things** to the user and **hides the internal details**.
- This is basically **called Abstraction** in which the **complex details are being hidden** from the users.

---

## EXAMPLE:

- Consider the example of **an email**, the user **does not know about the complex details** such as what happens just after sending an email, **which protocol is used by the server** to send the message. Therefore, we just need to mention the address of the receiver, type the content and click the send button.
- For example **sending sms**, you just **type the text and send the message**. You don't know the internal processing about the message delivery.

---

## HOW TO ACHIEVE ABSTRACTION IN JAVA?

There are **two ways** to achieve abstraction in java:

- **Abstract class (0 to 100%). (partial to complete abstraction)**

--abstract classes can contain concrete methods that have the implementation which results in a partial abstraction.

- **Interface (100% (complete abstraction)).**

--Interfaces allow you to abstract the implementation completely.

---

## ABSTRACT KEYWORD

- **Abstract** is a keyword which means **not complete / partial implementation** (having only declaration not definition)
- It is applicable for **Method** , **class** but **not on variable**.

---

## ABSTRACT METHOD

- Abstract methods are methods with **no implementation** and **without a method body**. They do not contain any method statement.
- An abstract method is declared with an **abstract keyword**.
- The declaration of an abstract method **must end with a semicolon ;**
- **The child classes which inherit the abstract class must provide the implementation of these inherited abstract methods.**

```
access-specifier abstract return-type method-name();  
For ex: public abstract int myMethod(int n1, int n2); // no body
```

## NEED OF ABSTRACT METHOD

```
Public class covid19  
{  
Public void vaccine ()  
{  
  
}  
}
```

**Don't know  
implementa  
tion**



**Know  
only  
declaration**

```
Public abstract class covid19  
{  
Public abstract void vaccine ();  
} // ends with ;  
// no cur
```

**Abstract  
Methods are  
implemented in  
the child class**



---

## WHICH ONE IS VALID?

- **public abstract void m1(){}**
- **public void m1();**
- **public abstract void m1();**
- **public void m1(){}**

---

## WHICH ONE IS VALID?

- `public abstract void m1(){} // compilation error: abstract method cannot have a body`
- `public void m1(); // compilation error: missing method body or abstract method`
- `public abstract void m1();`
- `public void m1(){}`

---

## ABSTRACT CLASS

- A class which contains the **abstract** keyword in its declaration is known as abstract class.
- an **abstract class is like a guideline or a template** that has to be extended by its child classes for the implementation.
- **Need of Abstract class:** Sometimes **implementation** of a class is **not complete** or having **partial implementation**.
- **Abstract classes may or may not contain *abstract methods***, i.e., methods without body  
( **public void get();** )
- But, if a class has **at least one abstract method**, then the **class must be declared abstract**.
- **If a class is declared abstract, it cannot be instantiated.**
- To use an abstract class, you have to **inherit** it from another class, provide implementations to the abstract methods in it.
- If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.

## CASE 1:

## CASE 1 & 2

```
public class Main
{
    abstract void wheel();
    public static void main(String[] args) {
        System.out.println("Hello World");
        Main m = new Main();
    }
}
```

## CASE 2:

```
abstract public class Main
{
    abstract void wheel();
    public static void main(String[] args) {
        System.out.println("Hello World");
        Main m = new Main();
    }
}
```

## CASE 1:

## CASE 1 & 2

```
public class Main
{
    abstract void wheel();
    public static void main(String[] args) {
        System.out.println("Hello World");
        Main m = new Main();
    }
}
```

### Output

Main.java:14: error: Main is abstract; cannot be instantiated

### Output

Main.java:9: error: Main is not abstract and does not override abstract method wheel() in Main

## CASE 2:

```
abstract public class Main
{
    abstract void wheel();
    public static void main(String[] args) {
        System.out.println("Hello World");
        Main m = new Main();
    }
}
```

---

## CONSIDER A SCENARIO THAT YOU HAVE TO BUILD YOUR FINAL YEAR PROJECT

- You want to make a new mobile app with some different features.
- You want to add features like
  - Calling
  - Moving
  - Video Calling

---

## PROGRAM

```
class FirstYear
{
    public void calling()
    {
        System.out.println("Calling");
    }
    public void moving();
    public void videocaling();
}
```

```
public class ABC {
    public static void main(String args[])
    {
        FirstYear b = new FirstYear();
        b.calling();
    }
}
```

Response	Percentage
Yes	55%
No	35%
Don't know	10%

```
abstract class FirstYear
{
    public void calling()
    {
        System.out.println("Calling");
    }

    public abstract void moving();
    public abstract void videocaling();
}
```

```
public class ABC {  
    public static void main(String args[])  
    {  
        //FirstYear b = new FirstYear();  
        //b.calling(); { we can't create  
the object of abstract class}  
    }  
}
```



---

## PROGRAM

```
abstract class FirstYear
{
    public void calling()
{
    System.out.println("Calling");
}

    public abstract void moving();
    public abstract void videocaling();
}
```

```
Class SecondYear extends FirstYear{
    public void moving()
{System.out.println("Calling");}}
```

```
public class ABC {
    public static void main(String args[]) {
        SecondYear b = new SecondYear();
        b.calling();
    }
}
```

---

## PROGRAM

```
abstract class FirstYear
```

```
{  
    public void calling()  
{  
    System.out.println("Calling");  
}  
    public abstract void moving();  
    public abstract void videocaling();  
}
```

```
abstract Class SecondYear extends FirstYear{  
    public void moving()  
{System.out.println("Calling");}}
```

```
public class ABC {  
    public static void main(String args[]) {  
        //SecondYear b = new SecondYear();  
        //b.calling(); { we can't create the  
        object of abstract class}  
    }  
}
```

## PROGRAM

```
abstract class FirstYear
{
    public void calling()
    {
        System.out.println("Calling");
    }
    public abstract void moving();
    public abstract void videocaling();
}

abstract Class SecondYear extends FirstYear{
    public void moving()
    {System.out.println("Calling");}}
```

```
Class ThirdYear extends SecondYear{
    Public void videocaling()
    {System.out.println("Video Call");
    }
    public class ABC {
        public static void main(String args[]) {
            ThirdYear b = new ThirdYear();
            b.calling();
            b.moving();
            b.videocaling();
        }
    }
}
```

---

## ABSTRACT CLASS VS ABSTRACT METHODS

- If a class contain at least one abstract method compulsory we have to declare that class as a abstract class.
- Even if class does not contain abstract method still we can declare a class as abstract class

## CASE 3

```
abstract public class Main
{
    abstract void wheel();
    abstract void engine();

    public static void main(String[] args) {
        System.out.println("Hello World");

        //      Main m =new Main();
    }
}

class main2 extends Main
{
}
```

Main.java:20: error: main2 is not abstract and does not override abstract method engine() in Main

## CASE 4

```
abstract public class Main
{
    abstract void wheel();
    abstract void engine();
    public static void main(String[] args) {
        System.out.println("Hello World");
        //Main m =new Main();
    }
}

abstract class main2 extends Main
{
}
```

### OUTPUT

Hello World

## CASE 5

```
abstract public class Main
{
    abstract void wheel();
    abstract void engine();
    public static void main(String[] args) {
        System.out.println("Hello World");
        //      Main m = new Main();
    }
}

class main2 extends Main
{
    void wheel(){}
}
```

### OUTPUT

Main.java:20: error: main2 is not abstract and does not override abstract method engine() in Main

---

## AN ABSTRACT CLASS CAN HAVE A CONSTRUCTOR BUT CAN NOT CREATE AN OBJECT.

```
public abstract class parent{  
    int i;  
    int j;  
  
}
```

```
public class child1 extends parent{  
    int m;  
    int n;  
    public child1 (int i, int j, int m, int n)  
    {  
        this.i=i;  
        this.j=j;  
        this.m=m;  
        this.n=n;  
    }  
    public static void main(String[] args) {  
        Child1 c=new child1(1,2,3,4);  
    }  
}
```



## AN ABSTRACT CLASS CAN HAVE A CONSTRUCTOR BUT CAN NOT CREATE AN OBJECT.

```
abstract class parent{  
    int i;  
    int j;  
  
}
```

```
public class child1 extends parent{  
    int m;  
    int n;  
    public child1 (int i, int j, int m, int n)  
    {  
        this.i=i;  
        this.j=j;  
        this.m=m;  
        this.n=n;  
    }  
    public static void main(String[] args) {  
        Child1 c=new child1(1,2,3,4);  
    }  
}
```

```
class child2 extends parent{  
    int x;  
    int y;  
    public child1 (int i, int j, int x, int y)  
    {  
        this.i=i;  
        this.j=j;  
        this.x=x;  
        this.y=y;  
    }  
}
```

- **Duplicate code**
- **Waste a lot of time**

## AN ABSTRACT CLASS CAN HAVE A CONSTRUCTOR BUT CAN NOT CREATE AN OBJECT. (SOLUTION)

```
public abstract class parent{  
    int i;  
    int j;  
    public parent (int i, int j)  
    {  
        this.i=i;  
        this.j=j  
    }  
}
```

```
public class child1 extends parent{  
    int m;  
    int n;  
    public child1 (int i, int j, int m, int n)  
    {  
        super(i,j);  
        this.m=m;  
        this.n=n;  
    }  
    public static void main(String[] args) {  
        Child1 c=new child1(1,2,3,4);  
    }  
}
```

```
public class child2 extends parent{  
    int x;  
    int y;  
    public child2 (int i, int j, int x, int y)  
    {  
        super(i,j);  
        this.x=x;  
        this.y=y;  
    }  
}
```

---


## MCQ

- Abstract class A has 4 virtual functions. Abstract class B defines only 2 of those member functions as it extends class A. Class C extends class B and implements the other two member functions of class A. Choose the correct option below.
  1. Program won't run as all the methods are not defined by B
  2. Program won't run as C is not inheriting A directly
  3. Program won't run as multiple inheritance is used
  4. Program runs correctly

---

## MCQ

- Abstract class A has 4 virtual functions. Abstract class B defines only 2 of those member functions as it extends class A. Class C extends class B and implements the other two member functions of class A. Choose the correct option below.
  1. Program won't run as all the methods are not defined by B
  2. Program won't run as C is not inheriting A directly
  3. Program won't run as multiple inheritance is used
  4. Program runs correctly



THANK YOU  
?