



# 19CSE204

## Object Oriented Paradigm

### 2-0-3-3

Amrita Vishwa Vidyapeetham  
Amritapuri Campus





## **Inheritance –**

- **Abstract Class**
- **Final Keyword**
- **Order of constructor invocation**

# **Abstract Class**

# Abstraction in Java

- **Abstraction** is the quality of dealing with ideas rather than events.
  - For example, consider the case of e-mail, complex details such as what happens as soon as you send an e-mail, the protocol your e-mail server uses are hidden from the user.
  - To send an e-mail you just need to type the content, mention the address of the receiver, and click send.
- In Object-oriented programming, **abstraction is a process of hiding the implementation details** from the user,
  - only the functionality will be provided to the user.
  - In other words, the user will have the information on what the object does instead of how it does it.
- .

**In Java, abstraction is achieved using Abstract classes and interfaces**

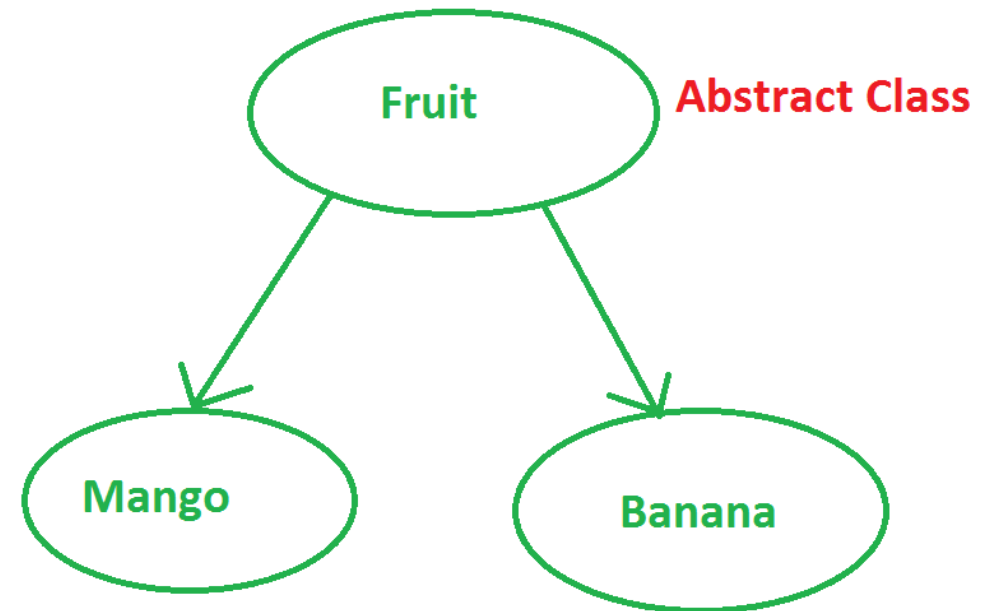
# Abstract Class

- Abstract Class : **Place holder in a class hierarchy**
- A **superclass that only defines a generalized form** (structure of a given abstraction without providing a complete implementation)

To declare an abstract method, use this general form:

***abstract type name(parameter-list);***

As you can see, no method body is present.





# Abstract class properties

- A class which contains the **abstract** keyword in its declaration is known as abstract class.
- Abstract classes can have **abstract** and **non-abstract** methods.
- 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.
- 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- a simple example

```
1 package abstractclass;
2
3 //A Simple demonstration of abstract.
4 abstract class A {
5     abstract void callme();
6     //concrete methods are still allowed in abstract classes
7     void callmetoo() {
8         System.out.println("This is a concrete method.");
9     }
10 }
11 class B extends A {
12     void callme() {
13         System.out.println("B's implementation of callme.");
14     }
15 }
16
17
18 public class Abstract {
19
20     public static void main(String[] args) {
21         B b = new B();
22         b.callme();
23         b.callmetoo();
24     }
25 }
26
27 }
```

- Notice that no objects of class **A** are declared in the program
- It is not possible to instantiate an abstract class.
- Class **A** implements a concrete method called **callmetoo()**. This is perfectly acceptable.
- All Abstract methods to be implemented in the extended class

## Output

B's implementation of callme  
This is a concrete method

# Abstract Class example

```
1 package abstractclass;
2
3 abstract class Shape{
4     abstract void draw();
5 }
6 //In real scenario, implementation is provided by others i.e. unknown by end user
7 class Rectangle extends Shape{
8     void draw(){System.out.println("drawing rectangle");}
9 }
10 class Circle1 extends Shape{
11     void draw(){System.out.println("drawing circle");}
12 }
13 //In real scenario, method is called by programmer or user
14 class Abstract{
15     public static void main(String args[]){
16         Shape s=new Circle1();//In a real scenario, object is provided through method, e.g.,
17
18         s.draw();
19     }
20 }
```

**Shape is the abstract class, and its implementation is provided by the Rectangle and Circle classes.**



# Abstract Class example

```
1 package abstractclassdemo;
2
3 //Using abstract methods and classes.
4 abstract class Figure {
5     double dim1;
6     double dim2;
7     Figure(double a, double b) {
8         dim1 = a;
9         dim2 = b;
10    }
11    // area is now an abstract method
12    abstract double area();
13 }
14
15 class Rectangle extends Figure {
16     Rectangle(double a, double b) {
17         super(a, b);
18     }
19     // override area for rectangle
20     double area() {
21         System.out.println("Inside Area for Rectangle.");
22         return dim1 * dim2;
23     }
24 }
25 class Triangle extends Figure {
26     Triangle(double a, double b) {
27         super(a, b);
28     }
29     // override area for right triangle
30     double area() {
31         System.out.println("Inside Area for Triangle.");
32         return dim1 * dim2 / 2;
33     }
34 }
```

Figure.java

```
1 package abstractclassdemo;
2
3 public class area {
4
5     public static void main(String[] args) {
6         // Figure f = new Figure(10, 10); // illegal now
7         Rectangle r = new Rectangle(9, 5);
8         Triangle t = new Triangle(10, 8);
9         Figure figref; // this is OK, no object is created
10        figref = r;
11        System.out.println("Area is " + figref.area());
12        figref = t;
13        System.out.println("Area is " + figref.area());
14    }
15 }
16
17 }
```

Area.java ( driver class)

Inside Area for Rectangle.  
Area is 45.0  
Inside Area for Triangle.  
Area is 40.0

**Final Keyword**

# 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
- **Java final variable**
  - If you make any variable as final, you cannot change the value of final variable(It will be constant).
- **Java final method**
  - If you make any method as final, you cannot override it.
- **Java final class**
  - If you make any class as final, you cannot extend it.

# Java final variable- example

```
class Bike9{
    final int speedlimit=90;//final variable
    void run(){
        speedlimit=400; }
    public static void main(String args[]){
        Bike9 obj=new Bike9();
        obj.run();
    }
} //end of class
```

Output:  
Compile time Error

There is a final variable **speedlimit**, 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.

# Final Method :Using final to Prevent Overriding

- To disallow a method from being overridden, specify **final** as a modifier at the start of its declaration. Methods declared as **final** cannot be overridden.

```
51 class A {  
52     final void meth() {  
53         System.out.println("This is a final method.");  
54     }  
55 }  
56 class B extends A {  
57     void meth() { // ERROR! Can't override.  
58         System.out.println("Illegal!");  
59     }  
60 }
```

Because **meth( )** is declared as **final**, it cannot be overridden in **B**. If you attempt to do so, a compile-time error will result

Normally, Java resolves calls to methods dynamically, at run time. This is called **late binding**.

However, since **final** methods cannot be overridden, a call to one can be resolved at compile time. This is called **early binding**

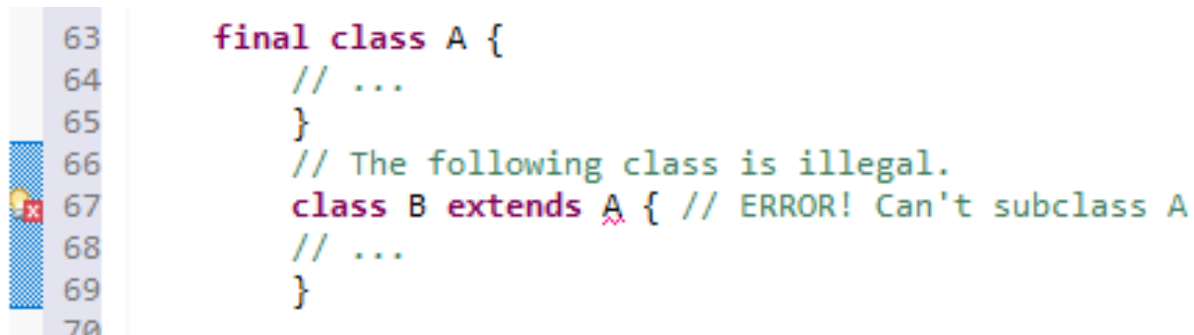
# Final Class: Using final to Prevent Inheritance

To prevent a class from being inherited, precede the class declaration with **final**

Declaring a class as **final** implicitly declares all of its methods as **final**, too.

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.

Here is an example of a **final** class:



```
63     final class A {  
64         // ...  
65     }  
66     // The following class is illegal.  
67     class B extends A { // ERROR! Can't subclass A  
68         // ...  
69     }  
70
```

As the comments imply, it is illegal for **B** to inherit **A** since **A** is declared as **final**.



# **Order of invocation of constructors in subclass**

# When Constructors Are Called?

- When a class hierarchy is created, in what order are the constructors for the classes that make up the hierarchy called?
  - Given a subclass called **B** and a superclass called **A**, constructors are called in order of derivation, from superclass to subclass

```
1 package abstractclass;
2
3 //Demonstrate when constructors are called.
4 //Create a super class.
5 class A {
6     A() {
7         System.out.println("Inside A's constructor.");
8     }
9 }
10 //Create a subclass by extending class A.
11 class B extends A {
12     B() {
13         System.out.println("Inside B's constructor.");
14     }
15 }
16 //Create another subclass by extending B.
17 class C extends B {
18     C() {
19         System.out.println("Inside C's constructor.");
20     }
21 }
22
```

```
24 public class CallingConst {
25
26     public static void main(String[] args) {
27         C c = new C();
28     }
29
30 }
31
32 }
```

The output from this program is shown here:  
Inside A's constructor  
Inside B's constructor  
Inside C's constructor

The constructors are called in order of derivation

Because a superclass has no knowledge of any subclass, any initialization it needs to perform is separate from and possibly prerequisite to any initialization performed by the subclass. Therefore, it must be executed first.

# Predict the output

```
class Parentclass
{
    //no-arg constructor
    Parentclass(){
        System.out.println("no-arg constructor of parent
class");
    }
    //arg or parameterized constructor
    Parentclass(String str){
        System.out.println("parameterized constructor of
parent class");
    }
}
```

```
class Subclass extends Parentclass
{
    Subclass(){
        super("Hahaha");
        System.out.println("Constructor of child class");
    }
    void display(){
        System.out.println("Hello");
    }
    public static void main(String args[]){
        Subclass obj= new Subclass();
        obj.display();
    }
}
```

# Constructor invocation

```
class Subclass extends Parentclass
```

```
{
```

```
    Subclass(){
```

```
        /* super() must be added to the first statement of constructor
```

```
        * otherwise you will get a compilation error. Another important
```

```
        * point to note is that when we explicitly use super in constructor
```

```
        * the compiler doesn't invoke the parent constructor automatically.
```

```
        */
```

```
        super("Hahaha");
```

```
        System.out.println("Constructor of child class");
```

```
    }
```

```
    void display(){
```

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

```
    }
```

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

```
        Subclass obj= new Subclass();
```

```
        obj.display();
```

```
    }
```

```
}
```

Output:

parameterized constructor of parent class

Constructor of child class

Hello

- **There are few important points to note in this example:**

- super()(or parameterized super) must be the **first statement in constructor** otherwise you will get the **compilation error**: “Constructor call must be the first statement in a constructor”
- When **we explicitly placed super in the constructor**, the java compiler **didn't call the default** no-arg constructor of parent class

Namah Shivaya

# Abstract class example

```
1 package abstractclass;
2
3 //Using abstract methods and classes.
4 abstract class Figure {
5     double dim1;
6     double dim2;
7     Figure(double a, double b) {
8         dim1 = a;
9         dim2 = b;
10    }
11    // area is now an abstract method
12    abstract double area();
13 }
14 class Rectangle extends Figure {
15     Rectangle(double a, double b) {
16         super(a, b);
17     }
18     // override area for rectangle
19     double area() {
20         System.out.println("Inside Area for Rectangle.");
21         return dim1 * dim2;
22     }
23 }
24 class Triangle extends Figure {
25     Triangle(double a, double b) {
26         super(a, b);
27     }
```

```
25     Triangle(double a, double b) {
26         super(a, b);
27     }
28
29     // override area for right triangle
30     double area() {
31         System.out.println("Inside Area for Triangle.");
32         return dim1 * dim2 / 2;
33     }
34 }
35
36
37 class Abstract {
38     public static void main(String args[]) {
39         // Figure f = new Figure(10, 10); // illegal now
40         Rectangle r = new Rectangle(9, 5);
41         Triangle t = new Triangle(10, 8);
42         Figure figref; // this is OK, no object is created
43         figref = r;
44         System.out.println("Area is " + figref.area());
45         figref = t;
46         System.out.println("Area is " + figref.area());
47     }
48 }
```