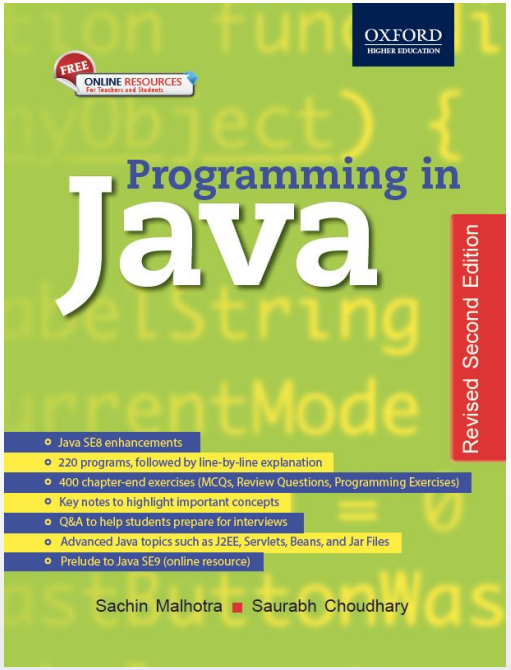
OXFORD
UNIVERSITY PRESS



Programming in Java

Revised 2nd Edition

Sachin Malhotra & Saurabh Choudhary

Chapter 5

Inheritance

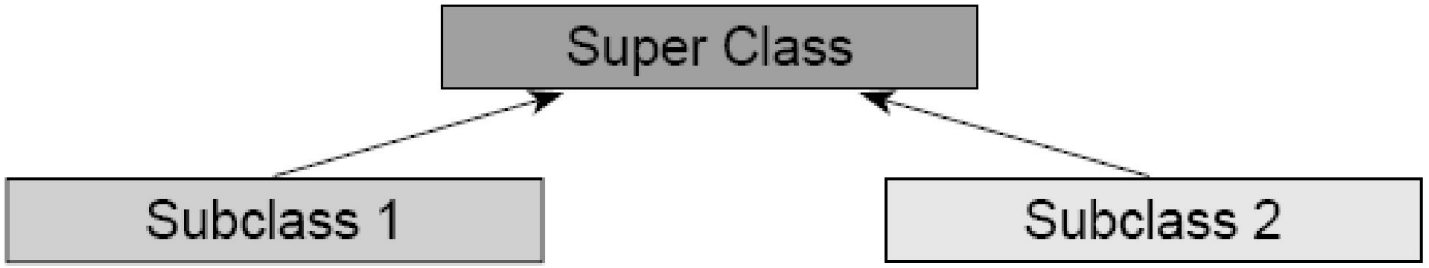
Objectives

- Know the difference between inheritance and aggregation
- Understand how inheritance is done in Java
- Learn polymorphism through method overriding
- Learn the keywords : super and final
- Understand the basics of abstract class

Inheritance

- Is the ability to derive something specific from something generic.
- Aids in the reuse of code.
- A class can inherit the features of another class and add its own modification.
- The parent class is the *super class* and the child class is known as the *subclass*.
- A subclass inherits all the properties and methods of the super class.

Inheritance



Aggregation

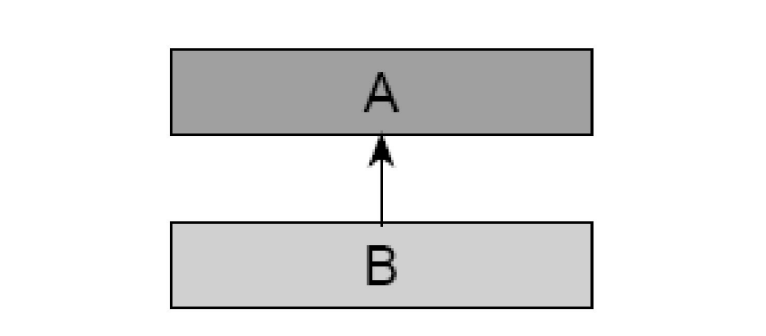
- We can make up objects out of other objects.
- The behaviour of the bigger object is defined by the behaviour of its component objects, separately and in conjunction with each other.
- Cars contain engine which in turn contains an ignition system and starter motor.
- A whole-part relationship exists in aggregation.
- Car being the whole and engine being its part.

Types of Inheritance

- Single
- Multiple
- Multilevel
- Hierarchical
- Hybrid

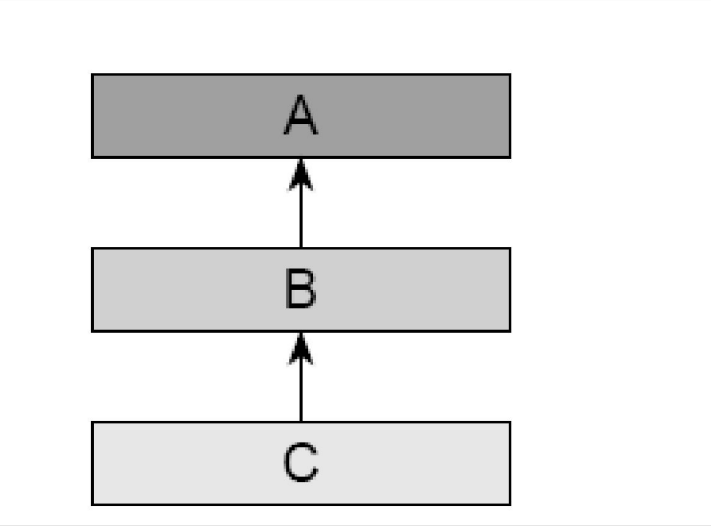
Single Level Inheritance

Classes have only base class



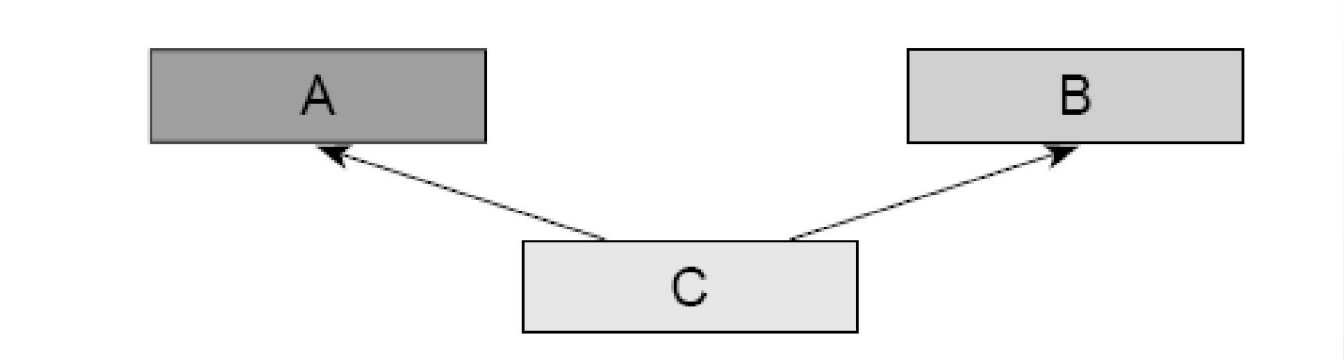
Multi Level Inheritance

There is no limit to this chain of inheritance (as shown below) but getting down deeper to four or five levels makes code excessively complex.



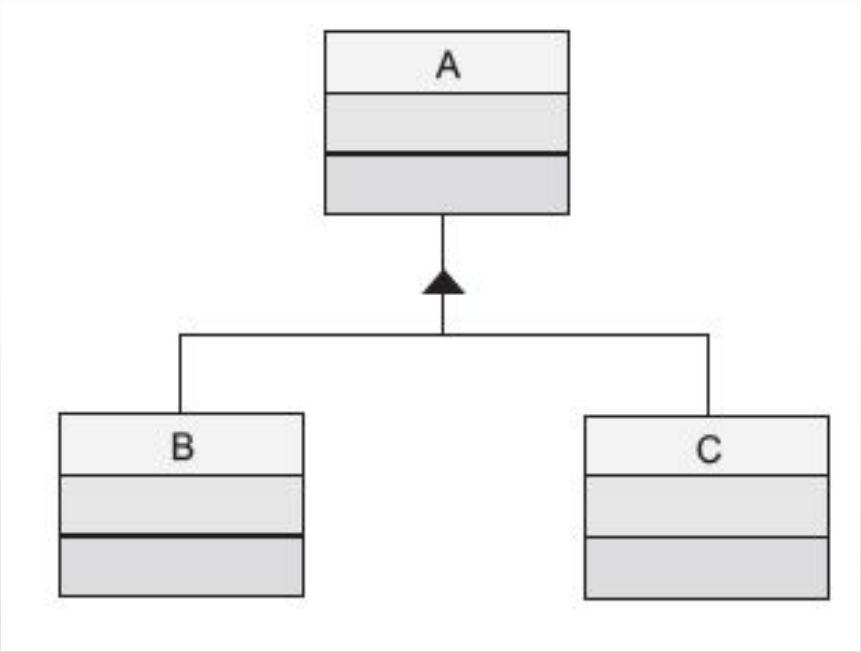
Multiple Inheritance

A class can inherit from more than one unrelated class.



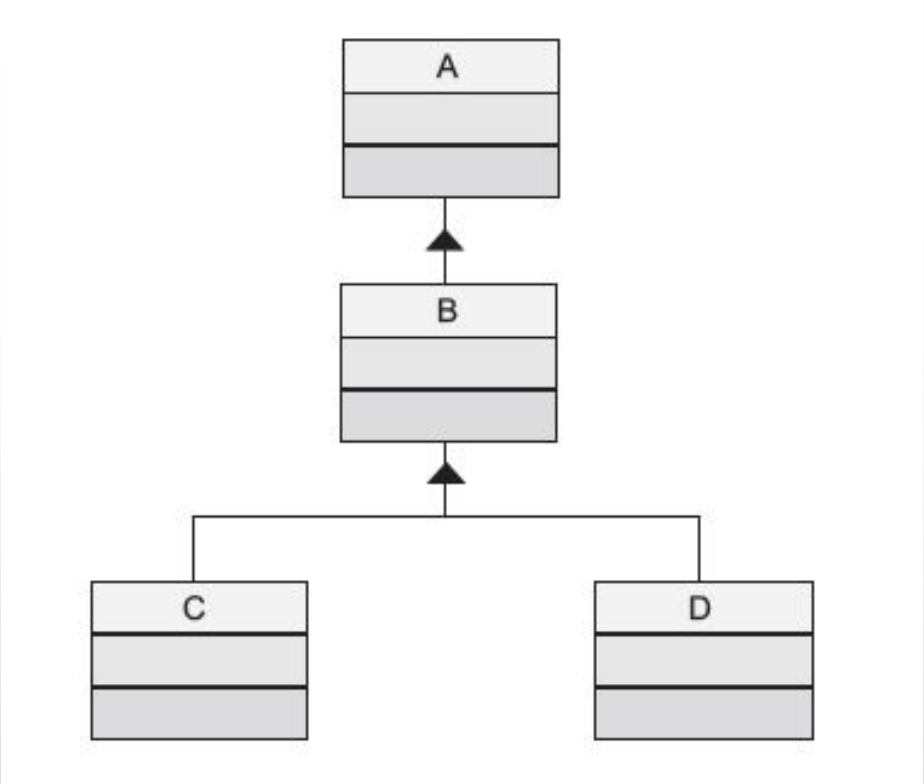
Hierarchical Inheritance

- In hierarchical inheritance, more than one class can inherit from a single class. Class C inherits from both A and B.



Hybrid Inheritance

- Is any combination of the above defined inheritances



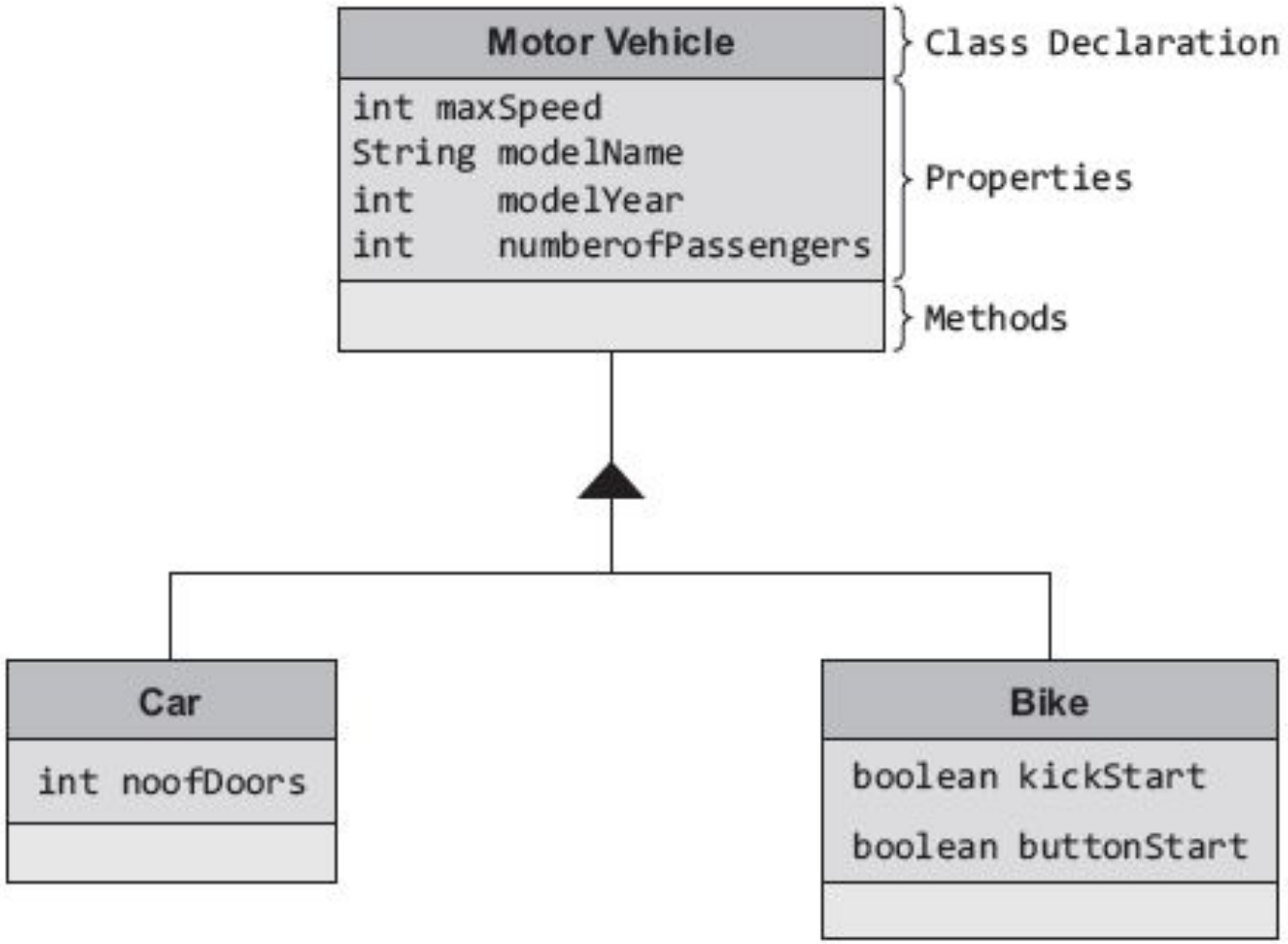
Deriving Classes

- Classes are inherited from other class by declaring them as a part of its definition
- For example

  class MySubClass **extends** MySuperClass

- *extends* keyword declares that *MySubClass* inherits the parent class *MySuperClass*.

Inheritance



Method Overriding

- A method in a subclass has the same name and type signature as a method in its superclass, then the method in the subclass is said to *override* the method in the superclass.
- It is a feature that supports polymorphism.
- When an overridden method is called through the subclass object, it will always refer to the version of the method defined by the subclass.
- The superclass version of the method is hidden.

Method Overriding Example

```
class A {
int i = 0;
void doOverride (int k) {
i = k;  } }
class B extends A {
void doOverride(int k) {
i = 2 * k;
System.out.println("The value of i is: "+i);  }
public static void main (String args[])
{  B b = new B();
   b.doOverride(12);
}}
```

The Output

The value of i is: 24

Late binding vs Early binding

- Binding is connecting a method call to a method body.
- When binding is performed before the program is executed, it is called ***early binding***.
- If binding is delayed till runtime it is late binding.
- Also known as *dynamic binding* or *runtime binding*.
- All the methods in Java use ***late binding (except static and final)***.

Superclass can Refer to a Subclass Object

```
class SuperClass {
int instanceVariable = 10;
static int classVariable = 20;}
class SubClass extends SuperClass{
int instanceVariable = 12;
static int classVariable = 25;
public static void main(String args[]) {
SuperClass s=new SubClass();
System.out.println("Superclass Instance variable: "+s.instanceVariable);
System.out.println("Superclass static variable: "+s.classVariable);
SubClass st=new SubClass();
System.out.println("Subclass Instance variable: "+st.instanceVariable);
System.out.println("Subclass static variable: "+st.classVariable); } }
```

The Output

Superclass Instance variable: 10

Superclass static variable: 20

Subclass Instance variable: 12

Subclass static variable: 25

super Keyword

- For invoking the methods of the super class.
- For accessing the member variables of the super class.
- For invoking the constructors of the super class.

super Keyword

```
class A {
void show()
{  System.out.println("Super Class show method");
} }
class B extends A
{
void show()
{  System.out.println("Subclass show method");    }
        public static void main (String args[ ]) {
   A s1=new A();
   s1.show();
   B s2=new B ();
   s2.show();
}}
```

Problem and Solution

- Two methods being called by two different objects (inherited), instead the job can be done by one object only, i.e. using super keyword.

Case 1: Invoking Methods using Super

```
class ANew {
void show()
{  System.out.println("Super Class show method");      } }
class BNew extends ANew {
void show()
{
   super.show();
   System.out.println("Subclass show method");
}
public static void main (String args[ ]) {
   BNew s2=new BNew ();
   s2.show(); }}
```

Case 2: Accessing Variables using Super

```
class Super_Variable {
int b=30;       }
class SuperClass extends Super_Variable {
int b=12;
void show()  {
System.out.println("subclass class variable: "+ b);
System.out.println("superclass instance variable: "+ super.b);
}
public static void main (String args[]) {
SuperClass s=new SubClass();
s.show(); // call to show method of Sub Class B
}}
```

The Output

- subclass class variable: 12
- superclass instance variable: 30

Case 3: Constructors of the Super Class

```
class Constructor_A {
Constructor_A() {
System.out.println(“Constructor A”);    }}
class Constructor_B extends Constructor_A {
Constructor_B() {
System.out.println(“Constructor B”);    }}
class Constructor_C extends Constructor_B {
Constructor_C() {
System.out.println(“Constructor C”);    }
public static void main (String args[]) {
Constructor_C a=new Constructor_C();
}}
```

The Output

Constructor A

Constructor B

Constructor C

Case 3: (contd.)

```
class Constructor_A_Revised {
Constructor_A_Revised()
{
    System.out.println("Constructor A Revised");
}}
class Constructor_B_Revised extends Constructor_A_Revised {
/*Constructor_B_Revised()
{
    System.out.println("Constructor B");
}*/
Constructor_B_Revised(int a)                {
    a++;
System.out.println("Constructor B Revised "+ a );          }}
class Constructor_C_Revised extends Constructor_B_Revised {
Constructor_C_Revised()  {
    super(11); // if omitted compile time error results
    System.out.println("Constructor C Revised");                    }
public static void main (String args[]) {
Constructor_C_Revised a=new Constructor_C_Revised();
}}
```

The Output

Constructor A Revised

Constructor B Revised 12

Constructor C Revised

final Keyword

- Declaring constants (used with variable and argument declaration)

```
final int MAX=100;
```

- Disallowing method overriding (used with method declaration)

```
final void show (final int x)
```

- Disallowing inheritance (used with class declaration)

```
final class Demo {}
```

Your Turn

- What is method overriding?
- Highlight the usage of super keyword?
- What are the usages of final keyword?
- What is late binding and early binding?

Abstract Class

- Abstract classes are classes with a generic concept, not related to a specific class.
- Abstract classes define partial behaviour and leave the rest for the subclasses to provide.
- Contains one or more abstract methods.
- Abstract method contains no implementation, i.e. no body.
- Abstract classes cannot be instantiated, but they can have reference variable.
- If the subclass does not override the abstract methods of the abstract class, then it is mandatory for the subclass to tag itself as *abstract.*

Why Create Abstract Methods?

- To force *same name* and *signature pattern* in all the subclasses.
- Subclasses should not use their own naming patterns.
- They should have the flexibility to code these methods with their own specific requirements.

Example of Abstract Class

```
abstract class Animal
{
   String name;
   String species;
   Animal(String n, String s)
   {
         name=n;
         species=s;
   }
   void eat(String fooditem)
   {
   System.out.println(species +" "+ name + "likes to have "+ fooditem);
   }
   abstract void sound();
}
```

Shadowing vs Overriding

- Shadowing of fields
  - occurs when variable names are same. It may occur when local variable and instance variable names collide within a class or variable names in superclass and subclass are same.
- In case of methods,
  - instance methods are overridden whereas static methods are shadowed. The difference between the two is important because shadowed methods are bound early whereas instance methods are dynamically (late) bound.

Shadowing vs Overriding

```
class Shadowing{
static void display(){
System.out.println("In Static Method of Superclass");
}
void instanceMethod() {
System.out.println("In instance Method of Super Class");
}}
```

Shadowing vs Overriding

```
class ShadowingTest extends Shadowing {
static void display(){
System.out.println("In Static Method of Sub Class");
}
void instanceMethod() {
System.out.println("The Overridden instance Method in Sub Class");
}
```

Shadowing vs Overriding

```
public static void main(String args[]){
Shadowing s=new ShadowingTest();
s.display();
s.instanceMethod();
ShadowingTest st=new ShadowingTest();
st.display();
st.instanceMethod();
} }
```

The Output

In Static Method of Superclass

The Overridden instance Method in Subclass

In Static Method of Subclass

The Overridden instance Method in Subclass

Practical Problem: Circle Class

```
class Circle {
float radius;
final fl oat PI = 3.141f; //value of pi is fi xed
Circle(){
radius = 1.0f;}
Circle(fl oat radius) {
this.radius = radius;}
float getArea() {
return PI * radius * radius; }}
```

Practical Problem: Cylinder Class

```
class Cylinder extends Circle {
float height;
Cylinder(fl oat radius, fl oat height){
super(radius);
this.height = height;
}
float getArea() {
return 2 * super.getArea() + 2 * PI * radius * height;      }
```

Practical Problem

```
public static void main(String args[]) {
Circle c = new Circle(1.5f);
System.out.println("The Area of Circle is: " + c.getArea());
Cylinder cy = new Cylinder(1.5f,3.5f);
System.out.println("The Surface Area of Cylinder is: " + cy.getArea());
}}
```

Summary

- The concept of inheritance is derived from real life.
- The properties and methods of a parent class are inherited by the children or subclasses.
- Subclasses can provide new implementation of method using method overriding, keeping the method names and signatures same as that of the parent class.
- The super keyword can be used to access the overridden methods, variables and even the constructors of the super class.
- Abstract classes are used to force the subclasses to override abstract methods and provide body and code for them.
- The final keyword is used to create constants and disallow inheritance.