Object Oriented Programming Methodology

Module 4 : Inheritance and Interface

Faculty In-charge:

Pragya Gupta

E-mail: pragya.g@somaiya.edu

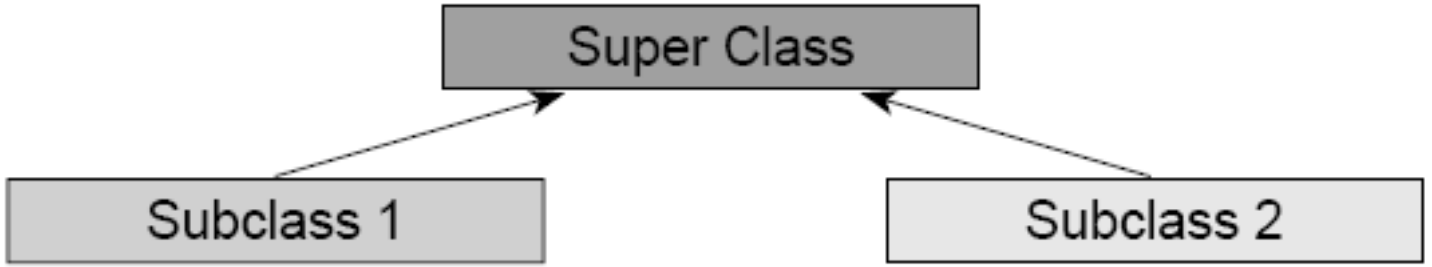
Contents

- Inheritance

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 s*ubclass*
- A subclass inherits all the properties and methods of the super class

Inheritance

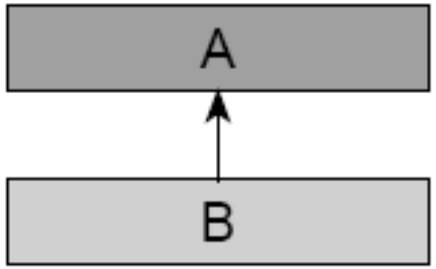


Types of Inheritance

- Single
- Multiple
- Multilevel
- Hierarchical
- Hybrid

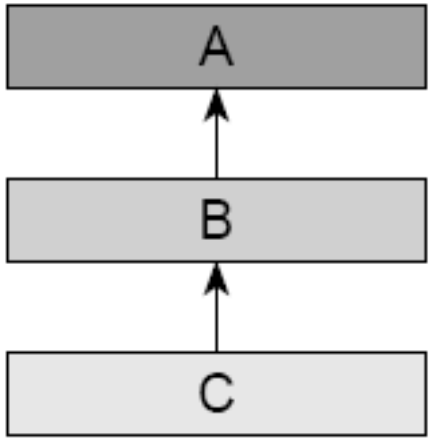
Single Level Inheritance

- Classes have only one base class



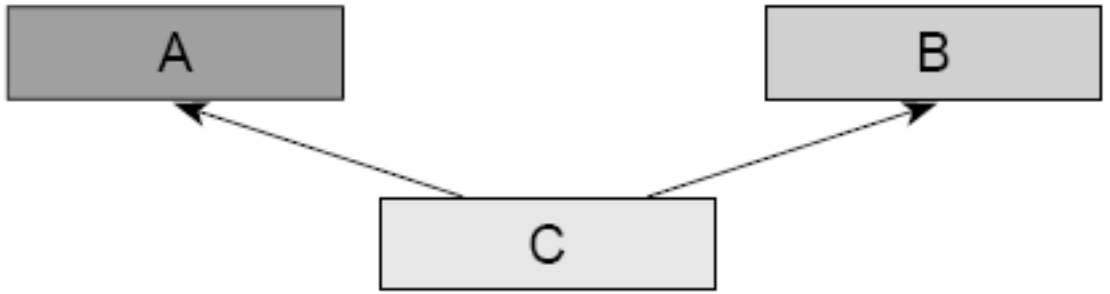
Multi Level Inheritance

- 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



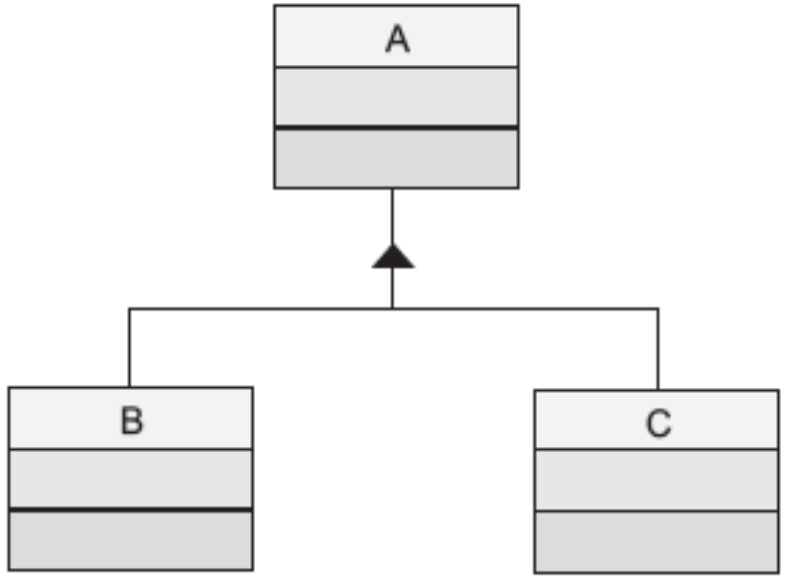
Multiple Inheritance

- A class can inherit from more than one unrelated class



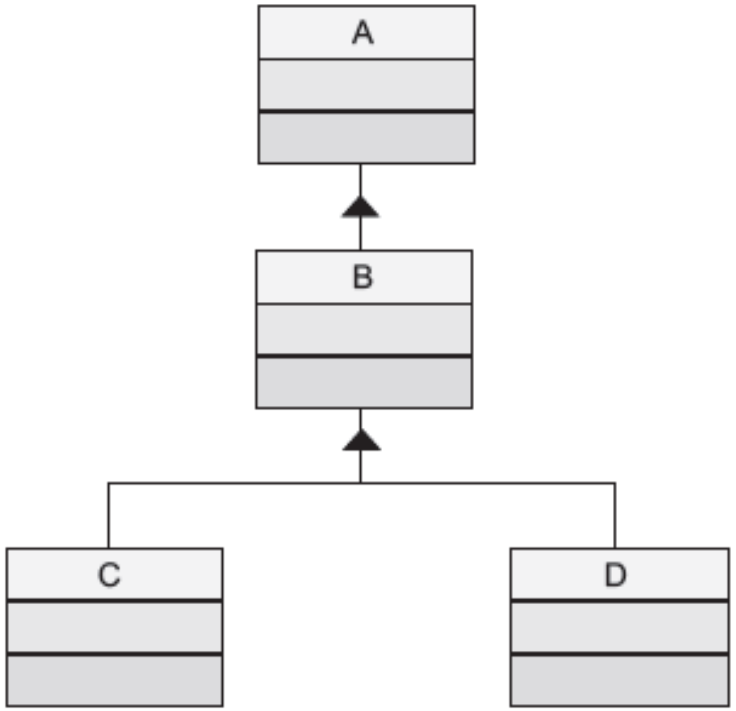
Hierarchical Inheritance

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



Hybrid Inheritance

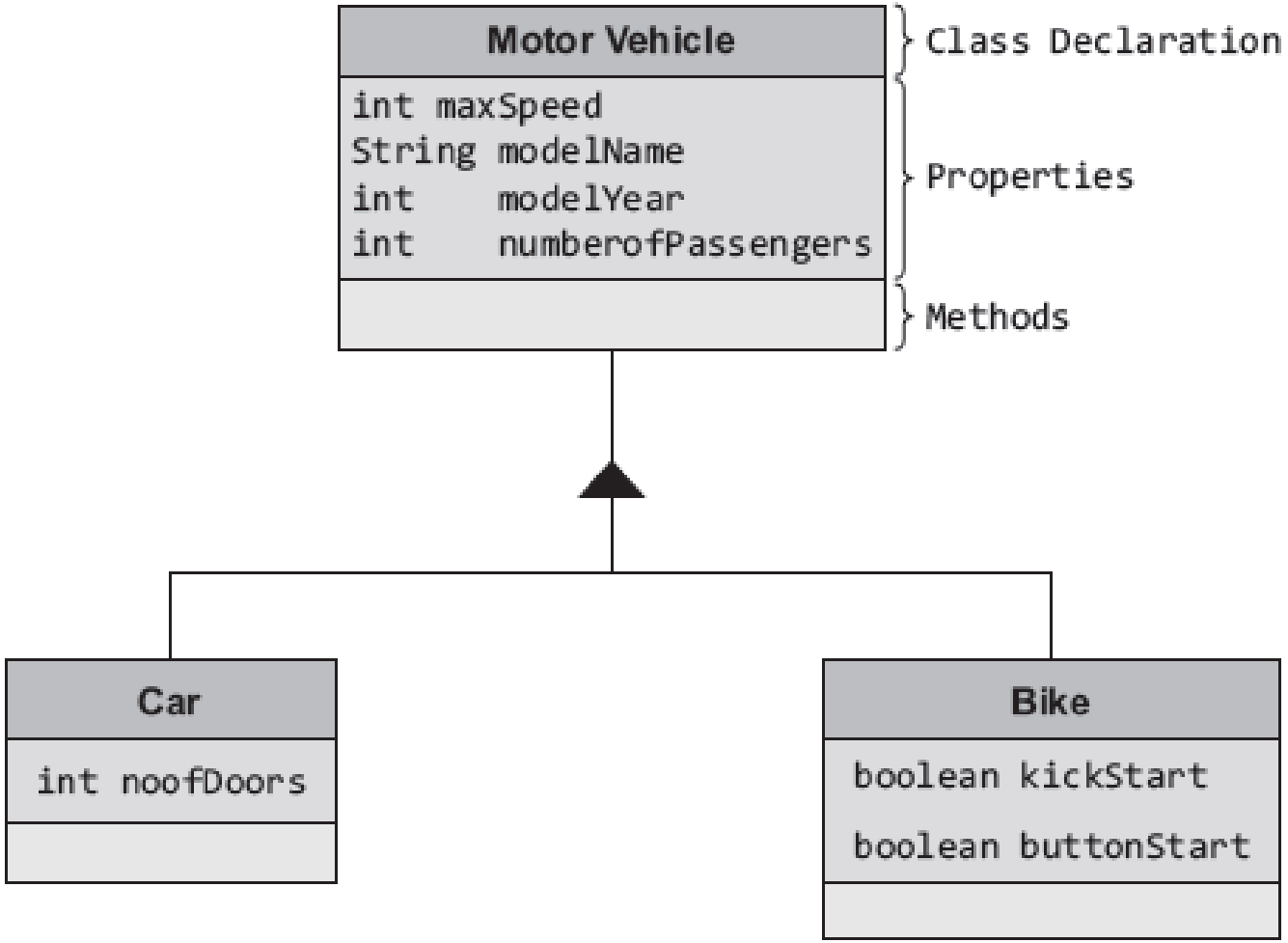
- is any combination of the above defined inheritances



Deriving Classes

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

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)***

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

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

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 {}

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
- contain 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 subclasses does not override the abstract methods of the abstract class, then it is mandatory for the subclasses 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

Shadowing Vs Overriding

- Shadowing of fields
  - occurs when variable names are same
  - It may occur when local variables 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.

Interfaces

- Interface is a collection of methods which are public and abstract by default
- The implementing objects have to inherit the interface and provide implementation for all these methods
- *multiple inheritance* in Java is allowed through interfaces
- Interfaces are declared with help of a keyword *interface*

Syntax for Creating Interface

```
interface interfacename
   {
   returntype methodname(argumentlist);
   …
   }
```

The class can inherit interfaces using implements keyword

- class classname implements interfacename{}

Variables in Interface

- They are implicitly *public*, *final*, and *static*
- As they are *final*, they need to be assigned a value compulsorily
- Being *static*, they can be accessed directly with the help of an interface name
- as they are *public* we can access them from anywhere

Extending Interfaces

- One interface can inherit another interface using the *extends* keyword and not the *implements* keyword
- For example,

  interface A extends B { }

Interface Vs Abstract Class

Interface	Abstract Class
Multiple inheritance possible; a class can inherit any number of interfaces.	Multiple inheritance not possible; a class can inherit only one class.
implements keyword is used to inherit an interface.	**extends** keyword is used to inherit a class.
By default, all methods in an interface are **public** and **abstract**; no need to tag it as **public** and **abstract**.	Methods have to be tagged as **public** or **abstract** or both, if required.
Interfaces have no implementation at all.	Abstract classes can have partial implementation.
All methods of an interface need to be overridden.	Only abstract methods need to be overridden.
All variables declared in an interface are by default **public**, **static**, and **final**.	Variables, if required, have to be declared as **public**, **static**, and **final**.
Interfaces do not have any constructors.	Abstract classes can have constructors
Methods in an interface cannot be static.	Non-abstract methods can be static.

Similarities between Interface and Abstract Class

- Both cannot be instantiated, i.e. objects cannot be created for both of them
- Both can have reference variables referring to their implementing classes objects
- Interfaces can be extended, i.e. one interface can inherit another interface, similar to that of abstract classes (using extends keyword)
- **static** / **final** methods can neither be created in an interface nor can they be used with abstract methods