COP 3330, Spring 2013

Inheritance III

Instructor : Arup Ghosh
03-01-13

School of Electrical Engineering and Computer Science
University of Central Florida

Today

- Abstract classes
  - A hybrid of normal classes and interfaces

Recap

- The `Object` class is at the root of Java's inheritance hierarchy.
  - Therefore, a Java object of any class is-a `Object`.

- The `protected` keyword allows subclasses to inherit non-public members without exposing it to a client.

- The `final` keyword prevents classes from being extended, and methods from being overridden.

Abstract classes

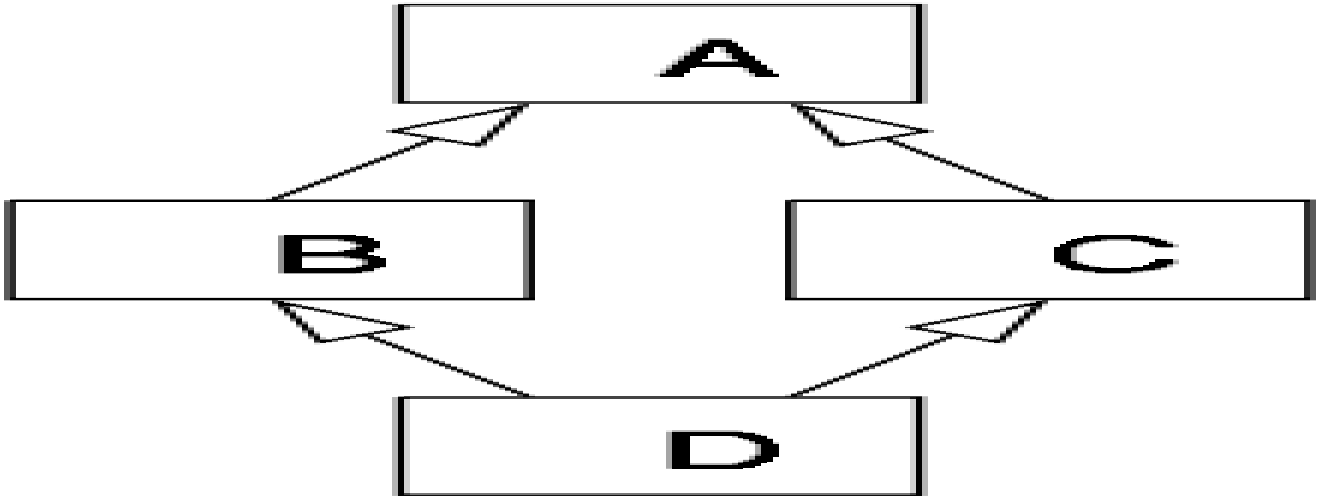
- By adding the modifier `abstract` to the class declaration, we can make a class that behaves a little like an interface.
- It can have fields and normal methods.
- At the same time, it can define *abstract methods*, that have no bodies.
  - Use the `abstract` modifier to indicate these methods.
  - Interface methods are also abstract, but we don't bother with the keyword there – since they can't be anything else.
- Bodies must be provided for them by any subclass.
  - Unless the subclass is abstract too, of course.

Abstract classes

- We cannot instantiate abstract classes – *even if they have constructors!*
- Since they lack some method bodies, they are incomplete, and their objects cannot be built (Abstract classes can't be instantiated), but they can be used as types due to polymorphism.
- Like interfaces, they provide a *framework* for building other classes.

- Unlike interfaces, they can provide some state and behavior.
  - In the form of their fields and non-abstract methods, which are inherited by their subclasses.

The diamond problem



(wikipedia)
The "diamond problem" (sometimes referred to as the "deadly diamond of death") is an ambiguity that arises when two classes B and C inherit from A, and class D inherits from both B and C. If D calls a method defined in A (and does not override the method), and B and C have overridden that method differently, then from which class does it inherit: B, or C?

Who inherits what?

- Things like this get troublesome, so Java doesn't allow multiple inheritance.

- You can only have one superclass, but you may implement as many interfaces as you like.
  - This gets us most of the expressive power, without any of the problems.

Example

- In a graphics library, we can have an abstract Figure class.
- Its subclasses can be Rectangle, Triangle, etc.
- Shape provides some basic functionality, and some abstract methods that its descendants can supply implementations for.
- We will write code.

Polymorphism

- As with interfaces, we can use an abstract type as the type of a variable.

- Because Rectangle is-a Shape, we can say:
  - `Figure r = new Rectangle(5,10);`

- Also,
  - `Figure t = new Triangle(6,5);`

- Figure can be a superobject inside of Rectangle, but only because the overridden abstract methods provide the missing pieces.

Multiple inheritance

- Some languages (notably C++) allow *multiple inheritance*.
- This means a class can have more than one superclass.
- E.g., a Dog might inherit from both Mammal and Pet.
- It has expressive power, but causes too much trouble.
  - For instance, the diamond problem