Introduction to Inheritance

Chris La Pilla
CompSci 6
Summer 2007

Code reuse

- Recall interface types
  - Interface consists of a list of methods without implementation
  - Interface itself contains no state or behavior
- Interfaces allow us to write code once that will work with all implementing classes
- But what if we have multiple classes that have nearly identical state or behavior?

Inheritance

- We can write a class that inherits the state and behavior of an existing class
- The new class can then:
  - Add new behaviors and state information
  - Override existing behaviors
- The new class is said to be a subclass, and the class it inherits from is said to be the superclass

Inheritance in Java

- Create a subclass by using the extends keyword:
  public class SavingsAccount extends BankAccount {
  ...
  }
- Example together: extending Wizard class for Wizards with different attacking strategies

Subclass Construction

- Recall that the constructor is responsible for initializing state
- When extending a class, we add new instance variables to its state, but we still retain all of the instance variables from the superclass
- We therefore need to initialize both the new instance variables and those defined in the superclass
- We already have a method responsible for initializing the variables in the superclass (which?)

Subclass construction

- Use the super keyword followed by parameters to invoke the superclass constructor:
  super(param1, param2, ...);
- This must be the first line of the subclass constructor
- The rest of the subclass constructor is used to initialize instance variables added by the subclass
- Can also use the super keyword followed by a period to access other methods
Subclass and Superclass types

- We know that a variable of a particular type can reference objects of that time
- That variable can also reference an object of any subclass of that type
- E.g. if TabbyCat extends Cat, a Cat variable can reference Cat objects or TabbyCat objects

Converting between subclass and superclass types

- Rules for variables with regards to subclass/superclass types is similar to rules for interfaces
- A subclass can be stored in a variable with a superclass type without casting
- If we want to convert a superclass type to a subclass type, we must use a cast:
  TabbyCat tabby = new TabbyCat();
  Cat cat = tabby; //no cast needed
  TabbyCat tabby2 = (TabbyCat) cat;

Polymorphism with inheritance

- If we override a method in a subclass, which method is actually called?
- The method called is always based on the actual type of the object, NOT the type of the reference:
  Cat cat = new TabbyCat();
  cat.meow();
  – If TabbyCat class overrides the meow method, then the meow method of TabbyCat is executed
  – If TabbyCat doesn’t override this method, then the meow method of Cat class is executed

Access Specifiers Revisited

- public: variable/method can be accessed anywhere
- private: variable/method can be accessed only within this class (but NOT within subclasses!)
- protected: variable/method can be accessed:
  – within this class
  – within any subclass of this class
  – within any class in the same package (see Ch 9.9)
- Important: if you do not include an access specifier, the variable/method has package access

Protected

- Protected instance variables can lead to the same problems as using public instance variables discussed previously (like what?)
- Better to keep instance variables private and use methods to access or modify them

Object class

- All classes you create are actually subclasses
- If you do not explicitly specify a superclass using extends, the superclass is Object
- This means that you can have a variable of type object that can hold a reference to an object of any class type
  – Is this useful?
Object class

- `Object` defines methods that all classes inherit; some important ones are:
  - `equals(Object obj)`: tests whether this object is the same as `obj`
  - `toString()`: returns a string representation of this object
  - `clone()`: returns a copy of this object
- The default implementations of these methods aren't very useful; if you want your own classes to use them, you must override them (see Ch 13.8)