CS 2112 Lab: Inheritance

8-10 September 2014
Inheritance Overview

- Language mechanism for extending and reusing code
- Different from subtyping!
- Two basic functions: Copying and Editing
Copying and Editing

- Copying is provided by the keyword `extends` in the method header.
- This allows you to use any functionality you included in your superclass, as long as it is public (or protected).
- You can edit existing classes by adding or changing functionality in a subclass.
- Any time you extend a class, you create a subtyping relationship where subclass `<:` superclass.
An Example

class Robot {
  ...

  public void doSomething() {
      ...
  }
}

class SmartRobot extends Robot {
  ...

  private int numSomethingsDone;

  public void doSomething() {
      ...
      numSomethingsDone++;
  }
}
Robot roboMan = new SmartRobot();
roboMan.doSomething();

Which doSomething() is called?
The static type is Robot and the dynamic type is SmartRobot.
This method is not static, so the method doSomething() of the dynamic type is called.
After this call, numSomethingsDone = 1.
class Robot {
    ...
    public void doSomething() {
        ...
    }
    public void doSomethingElse() {
        doSomething();
    }
}

Robot roboMan = new SmartRobot();
roboMan.doSomethingElse();

Now, which doSomething() is called?
Even if this call is made within a method of the superclass, the doSomething() method in the subclass will still be called. This is called late binding.
public Robot {
    static String hello() {
        return "HELLO";
    }
}

public SmartRobot extends Robot {
    static String hello() {
        return "Hello!";
    }
}

Robot roboMan = new SmartRobot();
roboMan.hello();

What is returned?
Static Methods

- The hello() method in the static type would be called, and this method would return "HELLO"
Which will work?

1
Robot roboman = new Robot();
Robot.hello();

2
Robot roboman;
roboman.hello();

1
Robot roboman = null;
roboman.hello();
Constructors

- To make sure you don’t leave anything uninitialized, Java requires that you call the superclass constructor in the first line of your subclass constructor.
- If you don’t, Java will call super() automatically.
Protected Visibility

- Visibility modifier `protected` will be accessible to the class and any of its subclasses.
- This creates a specialization interface that allows others to edit and expand your code without changing the public interface.
- Public and protected methods can be overridden, while private ones cannot.
- This is why it is good practice to create a specialization interface; you can define the way in which your code can be extended.
Inheritance Exercise

Design the inheritance hierarchy for classes associated with playing a game of chess

- Utilize interfaces, abstract classes, and super/subclasses to limit repetition of code, make your code easy to understand, and clearly define what should be open to the client, the "extenders," and what should be private.

- A couple class suggestions to get you started!

- Board, Location, Move, Piece, King, etc.