CS1007: Object Oriented Design and Programming in Java

Lecture #13
Nov 10

Shlomo Hershkop
shlomo@cs.columbia.edu

Outline

- Inheritance
- Objects
- Mouse listeners
- Car shape example

Announcements

- Next homework will be posted tomorrow.
  - See website

```
public class Employee
{
   public Employee(String aName) { name = aName; }
   public void setSalary(double aSalary)
       { salary = aSalary; }
   public String getName() { return name; }
   public double getSalary() { return salary; }

   private String name;
   private double salary;
}
```

How do we specialize the class?

- Manager class adds new method: setBonus
- Manager class *overrides* existing method: getSalary
- Adds salary and bonus

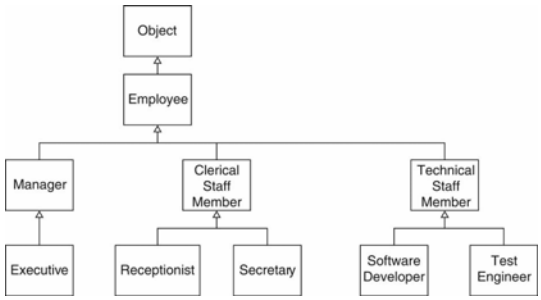
Overriding methods

- methods setSalary, getname (inherited from Employee)
- method getSalary (overridden in Manager)
- method setBonus (defined in Manager)
- fields name and salary (defined in Employee)
- field bonus (defined in Manager)

- Why is Manager a subclass?
- Isn't a Manager superior?
- Doesn't a Manager object have more fields?
- The set of managers is a subset of the set of employees

Inheritance Hierarchies

- Real world: Hierarchies describe general/specific relationships
  - General concept at root of tree
  - More specific concepts are children
- Programming: Inheritance hierarchy
  - General superclass at root of tree
  - More specific subclasses are children



Substitution Principle

- Formulated by Barbara Liskov
- You can use a subclass object whenever a superclass object is expected

Example:

Employee e;

...

System.out.println("salary=" + e.getSalary());

- Can set e to Manager reference
- Polymorphism: Correct getSalary method is invoked

Dealing with superclass

- Can't access private fields of superclass

```
public class Manager extends Employee
{
   public double getSalary()
   {
      return salary + bonus;  // ERROR--private field
   }
   ...
}
```

- Be careful when calling superclass method

```
public double getSalary()
   {
      return getSalary() + bonus;  // ERROR--recursive
   call
   }
```

super

- Use super keyword

```
public double getSalary()
 {
    return super.getSalary() + bonus;
 }
```

- super is not a reference
- super turns off polymorphic call mechanism

Super constructors

- Use super keyword in subclass constructor:

```
public Manager(String aName)
{
 super(aName); // calls superclass constructor
 bonus = 0;
}
```

- Call to super must be first statement in subclass constructor
- If subclass constructor doesn't call super, superclass must have constructor without parameters

Dealing with preconditions

- Precondition of redefined method at most as strong

```
public class Employee
{
   /**
      Sets the employee salary to a given value.
      @param aSalary the new salary
      @precondition aSalary > 0
   */
   public void setSalary(double aSalary) { ... }
}
```

- Can we redefine Manager.setSalary with precondition `salary > 100000`?
- No--Could be defeated:

```
Manager m = new Manager();
Employee e = m;
e.setSalary(50000);
```

Post conditions

- Postcondition of redefined method at least as strong
- Example: Employee.setSalary promises not to decrease salary
- Then Manager.setSalary must fulfill postcondition
- Redefined method cannot be more private. (Common error: omit public when redefining)
- Redefined method cannot throw more checked exceptions

Extending jcomponent

```
public class foo extends JComponent
{
   public void paintComponent(Graphics g)
   {
      drawing instructions go here
   }
   ...
}
```

Mouse listeners

- Attach mouse listener to component
- Can listen to mouse events (clicks) or mouse motion events
- Anyone know how?

Interface!

```
public interface MouseListener
{
   void mouseClicked(MouseEvent event);
   void mousePressed(MouseEvent event);
   void mouseReleased(MouseEvent event);
   void mouseEntered(MouseEvent event);
   void mouseExited(MouseEvent event);
}

public interface MouseMotionListener
{
   void mouseMoved(MouseEvent event);
   void mouseDragged(MouseEvent event);
}
```

- Includes a lot
- What if you just want part of it?

Extend MouseAdapter

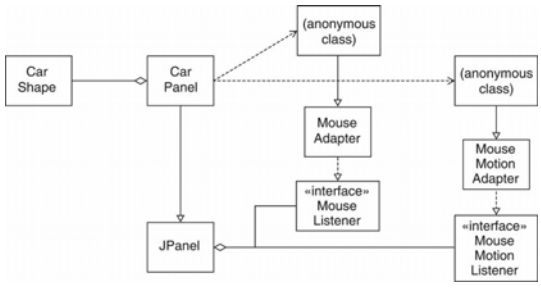
```
public class MouseAdapter implements MouseListener
{
   public void mouseClicked(MouseEvent event) {}
   public void mousePressed(MouseEvent event) {}
   public void mouseReleased(MouseEvent event) {}
   public void mouseEntered(MouseEvent event) {}
   public void mouseExited(MouseEvent event) {}
}
```

usage

```
addMouseListener(new
   MouseAdapter()
   {
      public void mousePressed(MouseEvent event)
      {
         mouse action goes here
      }
   });
```

Example: Car Mover Program

- Ch6/car/CarComponent.java
- Ch6/car/CarMover.java
- Ch6/car/CarShape.java



Next Time

- Do reading, start Homework