COMP 110
Inheritance

Darrell Bethea
June 6, 2011
Announcements

- Program 4 due Friday
  - Testing-debugging help online

- Final exam
  - Comprehensive
  - Monday, 6/13, 8–11 AM
  - SN014
Questions?
Today in COMP 110

- Inheritance
We have discussed before how classes of objects can have relationships.
Inheritance

- Define a general class

- Later, define specialized classes based on the general class

- These specialized classes inherit properties from the general class
Inheritance

- What are some properties of a Person?
  - Name, height, weight, age

- How about a Student?
  - ID, major

- Does a Student have a name, height, weight, and age?
  - Yes, Student inherits these properties from Person
The *is-a* relationship

- This inheritance relationship is known as an *is-a relationship*

- A Doctoral student *is* a Grad student
- A Grad student *is* a Student
- A Student *is* a Person

- Is a Person a Student?
  - Not necessarily!
Our general class is called a **base class**
- Also called a **parent class** or a **superclass**

**Examples:**
- Person, Transportation
A specialized class that inherits properties from a base class is called a **derived class**
- Also called a **child class** or a **subclass**

**Examples:**
- **Student** is-**a** Person
- **Employee** is-**a** Person
- **Car** is-**a** form of Transportation
- **Animal** is-**a** form of Transportation
Child (derived) classes can be parent (base) classes

- Student is a child class of Person
- Student is also the parent class of Undergrad and Grad
Why is inheritance useful?

- Enables you to define shared properties and actions once
- Derived classes can perform the same actions as base classes without having to redefine the actions
  - If desired, the actions can be redefined – more on this later
public class Person {
    private String name;
    public Person() {
        name = "No name yet";
    }
    public void setName(String newName) {
        name = newName;
    }
    public String getName() {
        return name;
    }
}
public class Student extends Person {
    private int id;
    public Student() {
        super();
        id = 0;
    }
    public Student(String stdName, int idNumber) {
        setName(stdName);
        setID(idNumber);
    }
    public void setID(int idNumber) {
        id = idNumber;
    }
    public int getID() {
        return id;
    }
}

Person
- name
+ setName(String newName) : void
+ getName() : String

Student
- id
+ setID(int idNumber) : void
+ getId() : int
The `extends` keyword

```java
public class Derived_CLASS_NAME extends Base_CLASS_NAME {
    Declaration_of_Added_Instance_Variables
    Definitions_of_Added_And_Overridden_Methods
}
```

```java
public class Student extends Person {
    // stuff goes here
}
```

- A derived (child) class inherits the **public** instance variables and **public** methods of its base (parent) class
public class SmileyMain extends JApplet

- The SmileyMain class inherits the public instance variables and public methods of the JApplet class
private vs. public

- **private** instance variables and **private** methods in the base class are NOT inherited by derived classes

- This would not work:

```java
public Student(String stdName, int idNumber)
{
    name = stdName; // ERROR! name is private to Person
    setID(idNumber);
}
```
private vs. public

- **private** instance variables of the base class CAN be accessed by derived classes using the base class’ **public** methods

- This works:

```java
public Student(String stdName, int idNumber)
{
    setName(stdName);  // OK! setName is a public method in Person
    setId(idNumber);
}
```
The super keyword

- Used to call a constructor of the base class (remember, a base class is also known as a superclass)
- More details later
What if the class Person had a method called printInfo?

```java
public class Person {
    // a bunch of other stuff
    // ...
    public void printInfo() {
        System.out.println(name);
    }
}
```
What if the class Student also had a method called printInfo?

```java
public class Student extends Person {
    // a bunch of other stuff
    // ...
    public void printInfo() {
        System.out.println("Name: " + getName());
        System.out.println("ID: " + getID());
    }
}
```
Overriding methods

- If Student inherits the printInfo() method and defines its own printInfo() method, it would seem that Student has two methods with the same signature
  - We saw before that this is illegal, so what’s the deal?
Overriding methods

- Java handles this situation as follows:
  - If a derived class defines a method that matches one in the base class in:
    - Name,
    - Number and types of parameters, and
    - Return type
  - Then derived class’ method **overrides** the base class’ method
  - The method definition in the derived class is the one that is used for objects of the derived class
Both Person and Student have a printInfo() method

Student std = new Student("John Smith", 37183);
std.printInfo(); // calls Student’s printInfo method,  
// not Person’s

Output would be:
Name: John Smith
ID: 37183
Overriding vs. overloading

- If a derived class defines a method of the same name, same number and types of parameters, and same return type as a base class method, this is **overriding**
- You can still have another method of the same name in the same class, as long as its number or types of parameters are different: **overloading**