Outline

- Inheritance Basics
- Inheritance in Java
- Overriding Methods
Inheritance

- A way of organizing classes
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

- A Student?
  - ID, major

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

- The 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, parent class, or super class.
Derived Class

- A specialized class that inherits properties from a base class is called a derived class, child class, or subclass.
Children Can Be Parents

- 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 \textit{once}

- Derived classes can perform the same actions as base classes without having to define the actions
  - If desired, the actions \textit{can} be redefined
    - More on this later
public class Person
{
    private String name;

    public Person()
    {
        name = "Nameless";
    }

    public void setName(String newName)
    {
        name = newName;
    }

    public String getName()
    {
        return name;
    }
}

Person
- name
+ setName(String newName) : void
+ getName() : String
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;
    }
}
The extends Keyword

```java
public class DerivedClassName extends BaseClassName {
    Declaration_of_Added_Instance_Variables
    Definition_of_Added_And_Overridden_Methods
}

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

- A derived class inherits the public instance variables and public methods of its base class
We have seen `extends` before.

```java
public class UpgradedSmiley extends JPanel
```

- The `UpgradedSmiley` class inherits the public instance variables and public methods of the `JPanel` class.
  - Don’t worry about what `JPanel` is.
    - It is a container class used in creating graphical user interfaces.
Inheritance: private v. 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);
}
```
Inheritance: private v. 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 (superclass) from a derived class
  - More details later
Overriding methods

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);
    }
}
```
Overriding methods

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 with the same name, number and types of parameters, and return type as a method in the base class, the 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.
Example: Overriding Methods

- Both Person and Student have a printInfo() method

Given:

- Student std = new Student("John Smith", 37183);
  std.printInfo();

Output

- Name: John Smith
- ID: 37183
Overriding v. Overloading

- Overriding
  - Derived class defines a method of the same name, same number and types of parameters, and same return type as a base class method

- Overloading
  - A class defines multiple methods with the same name, but with different number or types of parameters
Questions?
Logistics

Next:
  - Program 4 Discussion

Assignments:
  - Lab 6 is due on Monday