Notes: Inheritance

Inheritance

- **inheritance: a way to form new classes based on existing classes, taking on their attributes/behavior**
  - a way to group related classes
  - a way to share code between two or more classes
  - one class can extend another, absorbing its data/behavior
- **superclass**: the parent class that is being extended
  - you can refer to the constructor of the super class using `super()` as the first line in a subclass's constructor
  - you can refer to methods in a super class by using `super.methodname()` in a subclass
- **subclass**: the child class that extends that superclass (and inherits the superclass' behavior)
  - Subclasses inherit all of the public and protected instance methods of the parent class
  - Subclasses inherit all of the public and protected instance and class variables
  - Subclasses can have their own instance variables
  - Subclasses can have their own static and instance methods
  - Subclasses can override the parent class's methods
  - Subclasses can contain constructors that directly invoke the parent class's constructor using the `super` keyword
- **is-a relationship**: a hierarchical connection where one category can be treated as a specialized version of another (uses the keyword `extends`)
  - All classes except for `Object` extend Object; you don't have to code it (`extends Object`), Java automatically adds it in (this is why we can call toString() on classes that we create before we define our own)
- **inheritance hierarchy**: a set of classes connected by is-a relationships that can share common code
  - multiple levels of inheritance in a hierarchy are allowed
- **polymorphism**: ability for the same code to be used with different types of objects and behave differently with each
- **override**: to write a new version of a method in a subclass that replaces the superclass's version

Code Example

Here we are creating a new class called AdministrativeAssistant, but utilizing existing code from the Employee class. By doing this, we can utilize methods from the Employee class without having to rewrite code.

```
public class EmployeeClientProgram {
   public static void main(String[] args) {
      Employee sally = new Employee();
      System.out.println("Sally works " + sally.getHours() + " hours a week.");   // 40

      AdministrativeAssistant bob = new AdministrativeAssistant();
      System.out.println("Bob works " + bob.getHours() + " hours a week.");      // 45
   }
}

public class AdministrativeAssistant extends Employee {
   // "extends Employee" --> inherit all state and behavior of an Employee
   // i.e. getSalary, getVacationDays, and getVacationForm exist here
   //      even though you don't see them

   // overrides getHours() that was inherited from Employee
   public int getHours() { return super.getHours() + 5; }

   // AdministrativeAssistant adds the takeDictation method.
   public void takeDictation(String text) {
      System.out.println("Taking dictation of text: " + text);
   }
}


public class Employee {
   // ... fields, constructors, mutators, toString

   public int getHours() { return hours; }         // works 40 hours / week
   public double getSalary() { return salary;  }   // $40,000.00 / year
   public int getVacationDays() { return vacayDays;  }     // 10 days
   public String getVacationForm() { return formColor;  }   // "yellow" form
}
```