Topic 7:
Inheritance

Reading: JBD Sections 7.1 - 7.6
A Quick Review of Objects and Classes

- An object is an abstraction that models some thing or process

- Examples of objects:
  - Students, Teachers, Classes, Majors
  - Cars, Drivers, Races, Entries
  - Instruments, Measurements, Standards
  - Lists, Queues, Stacks

- Each object has:
  - Identity (distinguishes it from all other objects)
  - State (represented by instance variables)
  - Behavior (represented by instance methods)
Review of Objects and Classes (continued)

- A class is a user-defined type
- Every object is an instance of a class
- A class is like a "template" for creating objects
  - To create an object, use the `new` keyword
- A class has:
  - One or more constructors (to initialize new objects)
  - Static variables
    - Properties of the class as a whole.
    - One copy of variables for the whole class.
    - Invoke methods using the class name.
  - Static methods
  - Instance variables ("fields")
    - Represent the state and behavior of an object.
    - Each object has its own copy of these variables.
    - Invoke methods using an object reference.
  - Instance methods
Solving Problems with Objects

- If you write a description of your problem, the nouns in the description are probably objects

- What do you need to remember about each object?
  - Define instance variables (fields)
  - Write a constructor to initialize the fields of a new object
  - Declare the fields *private* if you do not want users to manipulate them directly

- What can each object "do"?
  - Write instance methods for these behaviors
  - Decide which methods are *public* and which are *private*

- Is there some data or function that belongs to the class as a whole, not to any individual object?
  - Define static variables and static methods
Example of a class declaration

class Student {
    private String name;
    private String major;
    private double gpa;
    private boolean coursesCompleted;
    public Student(String name, String major) {
        this.name = name;
        this.major = major;
        this.gpa = 0.0;
        this.coursesCompleted = false;
    }
    public String getName() { return name; }
    public String getMajor() { return major; }
    public double getGpa() { return gpa; }
    // more methods go here
}
More methods of the Student class

class Student {
    // variables, constructor, accessors go here
    public void setGpa(double gpa) {
        this.gpa = gpa;
    }
    public void setCoursesCompleted() {
        this.coursesCompleted = true;
    }
    public String toString() {
        return "Student " + name;
    }
    public boolean readyToGraduate() {
        if (gpa >= 2.0 && coursesCompleted)
            return true;
        else return false;
    }
}

CMPS 12A Winter 2009 UCSC
Using the Student class

```java
Student s1 = new Student("Peter", "Physics");
Student s2 = new Student("Michelle", "Math");
Student[] a = {s1, s2};
s2.setGpa(3.5);
s2.setCoursesCompleted();
for (Student s : a) {
    if (s.readyToGraduate())
        System.out.println(s +
            " is ready to graduate.");
    else
        System.out.println(s +
            " is not ready to graduate.");
}

Student Peter is not ready to graduate.
Student Michelle is ready to graduate.
```
Inheritance

- What if some existing class is pretty close to what you need, but not exactly right?
- Example: A graduate student is a student with an extra requirement: she must write a thesis
- It's wasteful and redundant to create a whole new class for graduate students
- A better approach: create GradStudent as a subclass of Student
- A subclass "inherits" all the fields and methods of its superclass, PLUS:
  - It can add more fields and methods
  - It can override fields and methods
Declaring a subclass

class GradStudent extends Student {
    private String thesisTopic;
    private boolean thesisCompleted;
    // add some more methods here
}

- GradStudent is a subclass of Student
  - A class may have more than one subclass
    (GradStudent, RemoteStudent, PartTimeStudent, etc.)

- Student is the superclass of GradStudent
  - A class can have only one superclass

- The "is-a" test:
  - A GradStudent is a Student
Objects in Storage

- A instance of a subclass looks just like an instance of its superclass, plus some extra fields.

Stack

- Name
- Value

Heap

- name
- major
- gpa
- coursesCompleted
- thesisTopic
- thesisCompleted

the Student part

the GradStudent part
A Class Hierarchy

• Many subclasses may be created to define a class hierarchy:

```
Person
   |
   Teacher
   |
   Tenured
   Teacher

   |
   Student
   |
   Graduate
   Student

   |
   Staff
```

• Define each field at the highest level where it applies

• An object may be created at any level of the hierarchy
Another Example of a Type Hierarchy

- Apply the "is-a" test at every level
Another Example of a Type Hierarchy

Vehicle
- Boat
  - Sailboat
  - Motor Boat
- Car
  - Gas-Powered Car
  - Electric Car
- Airplane
  - Jet Plane
  - Propeller Plane
Advantages of Inheritance

- Enables re-use of code
- Avoids redundancy
- Models relationships found in the real world
- Makes programs easily extensible to new cases
A subclass needs a constructor

- First, the subclass constructor calls the constructor of its superclass
  - Use the keyword `super` for this
  - This initializes the inherited fields
- Then the subclass constructor initializes its own fields

```java
class GradStudent extends Student {
    private String thesisTopic;
    private boolean thesisCompleted;
    public GradStudent(String name, String major, String topic) {
        super(name, major);
        this.thesisTopic = topic;
        this.thesisCompleted = false;
    }
}
```
A subclass can add new methods

class GradStudent extends Student {
    private String thesisTopic;
    private boolean thesisCompleted;
    public GradStudent(String name, String major, String topic) {
        super(n, m);
        this.thesisTopic = t;
        this.thesisCompleted = false;
    }
    void setThesisCompleted() {
        this.thesisCompleted = true;
    }
}
Overriding Methods

- A subclass can override (replace) a method of its superclass
- The subclass can still invoke the superclass method by using the keyword `super`

```java
class GradStudent extends Student {
    private String thesisTopic;
    private boolean thesisCompleted;
    public String toString() {
        return "Grad Student " + this.getName();
    }
    public boolean readyToGraduate() {
        if (super.readyToGraduate() && thesisCompleted)
            return true;
        else return false;
    }
}
```
**Method Selection**

- When a method is called, Java automatically invokes the method of the most specific type

```java
Student s1 = new Student("Bill", "Politics");
s1.readyToGraduate();
```

invokes the method of the Student class

```java
GradStudent gs2 = new GradStudent("Hillary", "Law", "Health Care Reform");
gs2.readyToGraduate();
```

invokes the method of the GradStudent class
Subtype Substitution

- Since a GradStudent is a Student, it can be used wherever a Student is expected

```java
Student s;
GradStudent gs;
s = new Student("Bill", "Politics");
gs = new GradStudent("Hillary", "Law", "Reforming Health Care");
s = gs;    // OK
```

- ... but not vice versa

```java
gs = s;    // Not OK!
```

- A superclass object cannot be assigned to a subclass variable because it does not have all the required fields
Polymorphism  (Greek: "Many Forms")

- An array (or other collection) can contain objects of various subtypes
- For each method call, the most specific applicable method is selected

```java
Student s = new Student("Bill", "Politics");
GradStudent gs = new GradStudent("Hillary", "Law", "Reforming Health Care");
Student[] students = {s, gs};
for (Student s : students)
    if (s.readyToGraduate())
        System.out.println(s + " is ready.");
    else System.out.println(s + " is not ready.");

Student Bill is not ready.
Grad Student Hillary is not ready.
```
Why is polymorphism important?

- It keeps type-specific behavior where it belongs (in the class definition, not in the application)
- It makes your code extensible to new cases
  - Example: Suppose you have a program that computes the tuition for every Student. You don't have to add new cases to this program, as long as every new subtype of Student provides its own tuition() method.
Overloading vs. Overriding

"Overloading" is method selection based on the static (compile-time) types of the arguments

```java
Book b = new Book("War and Peace");
Drink c = new Drink("Decaf Latte");
int charge = price(b) + price(c);
```

"Overriding" is method selection based on the dynamic (run-time) type of objects in a class hierarchy

```java
Student s = new Student(...);
GradStudent gs = new GradStudent(...);
Student[] students = {s, gs};
for (Student s: students)
    System.out.println(s.readyToGraduate());
```
Abstract Classes

- An abstract class is a class that declares some method signatures without any bodies.
- The bodies must be provided by subtypes.
- Example (from this week's homework):

```java
abstract class Shape {
    abstract double area();
    public abstract String toString();
}

class Circle extends Shape {
    double area() { ... }
    String toString() { ... }
}
```

- Every subclass of Shape must provide its own area() and toString() methods.
Interfaces

- An interface is a named set of method signatures
  
  ```java
  interface Size {
    public double height();
    public double weight();
  }
  ```

- A class can declare that it implements an interface (it must provide all the required methods)
  
  ```java
  class Mouse implements Size {
  ...
  }
  class House implements Size {
  ...
  }
  class Blouse implements Size {
  ...
  }
  ```
Interfaces (continued)

- You can declare an array (or other collection) of different kinds of objects that all implement a common interface:

  ```java
  Mouse m1 = new Mouse( ... );
  House h1 = new House( ... );
  Blouse b1 = new Blouse( ... );
  Size[] things = {m1, h1, b1};
  for (Size s:things)
      System.out.println("Weight is " + s.weight());
  ```
Multiple Inheritance

- Some languages (e.g., C++) allow a class to have multiple superclasses

- Java does not support multiple inheritance

- But in Java, a class can implement as many interfaces as it likes
What makes a language "object-oriented"?

- Abstraction
  - In Java: Classes and methods

- Encapsulation
  - In Java: Private fields and methods

- Inheritance
  - In Java: Subclasses

- Polymorphism
  - In Java: Dynamic method selection