COE318 Lecture Notes Week 2 (Week of Sept 8, 2014)

Announcements

- Quiz (5% of total mark) on Wednesday, September 24, 2014. Covers weeks 1–3.
- Midterm (20% of total mark) on Wednesday, October 15. Covers weeks 1–6.

Topics

- Objects and classes: an introduction
- Notes on Lab 1
- Simplest possible object: Nothing
- Notes on Lab 2
- Text/Tutorial sections covered
- Q&A

Anatomy of an Object

We first discuss what an *object* is without reference to any particular programming language. (i.e. this discussion is applicable to any object-oriented language such as C#, C++, Java, python, Objective-C, Ruby, PHP, Groovy, etc.

- An object represents a “person, place or thing”, or an “idea, concept or algorithm” or “anything else!”
- Any object has a *type*.
    - Example: if you refer to some person as the “bestFriend” object, then “bestFriend” is of type Person.
    - Example: if your only pet is a dog and you refer to it as “myPet” then “myPet” is of type Dog.

1. An object can have **characteristics**.
    - Example: A Person object may have characteristics such as “name”, “date of birth”, “gender”, etc.
    - Example, A Dog object may have characteristics like “name”, “weight”, “breed”, etc.
    - Example: A Robot object may have characteristics like “location”, etc.

- An object can have **behaviours** (things it can do).
  - Example: A Dog object can "*speak*": it would "say" something like "woof, woof".
  - Example: A Robot object can *"move"*.
- An object whose characteristics cannot be changed is *immutable*.
- Conversely, an object whose characteristics ca change is *mutable*.
- As a general principle, immutable objects are *safer* (cause fewer bugs) than mutable objects.

The "type" of an Object is a Class name

- An *object* is a specific instance of a general concept or *class*.
- An object has *characteristics* and *behaviour.*
  - Characteristics are called **instance variables** and distinguish one object from another of the same type (or class).
  - For example, a chair may be characterized by its position in a room. So instance variables of a chair object would include its row and column numbers.
  - Similarly, a person object may be characterized by their hair colour and name and there would be corresponding instance variables such as `hairColor` and `name`.
  - A behaviour of an object is called a **method**. Methods look like C functions and can take parameters and return some data type.
  - For example, a Person object might be asked for their name with the method `getName()`.
- An object's instance variables and methods are called *members*.
  - To access an object's member append a dot (.) to the object's name followed by the member.
  - For example if an object called `leftFork` of type `Fork` has an instance variable called `colour` and a method called `lift()`, the colour is obtained with `leftFork.colour` and you get the fork to be lifted with `leftFork.lift()`.
  - If the member name has parentheses at the end, it is a method invocation.
  - Otherwise, it accesses an instance variable.
  - Until you have more knowledge, always make instance variables *private*. This means they can only by accessed by an object of the same type, not by foreign objects.
  - Note: the text book does **not** follow this convention in the initial chapters. Although all the code is correct, following this practice could mislead you and develop bad programming habits. To repeat, make instance variables private and methods public until you have learned more about Java and safe programming.
- A *class* defines a data type for objects. Java source code defines the class and sets the instance variables and methods for objects of that type.

- (Note: there is no source code for an object; only the class is defined by source code.)
- In the class source code, the instance variables are shared throughout the source code for the class. Thus if one method changes the value of an instance variable, any other method belonging to the object that uses the instance variable will obtain its updated value. In short, instance variables act like global variables within the class source code file.
- When an object is created, it gets its own copy of all the instance variables defined in the class.
- An object's own instance variables and methods can be accessed directly by name in the class's source code.
- However, it is also permissible to use the keyword `this` to access a member.
- For example, if an instance variable is called `x`, it can be accessed in the class source code as either `x` by itself or as `this.x`
- Using `this` is only necessary when there is ambiguity.
- For example, consider the following code fragment where the name `x` is used for two entirely different things: a passed parameter to a method and an instance variable:

```
private int x; //instance variable
public void setX(int x) {
    //Set the instance variable x to the parameter x's value
    this.x = x;
}
```

Notes on Lab 1

- Due date: You must submit the lab at least 24 hours before your scheduled start for lab2. If you do not, you receive zero (0) for the lab. (However, leniency will be applied for the first lab because you have not used the department's computers for this before.)
- Submission: You can submit the project as often as you like. It is recommended that you submit as soon as you have anything at all. Then re-submit once you more of the lab working. The number of times you submit is not tracked. Only the last submission is graded.
- I will show you how to get started with Lab 1 during the lecture

Example: A Water Pipe

Suppose we have a water pipe 1 m long and of some specified diameter *d*. The basic fluid mechanics tells us that the water flow *Q* and the pressure *P* are related as:

$$P \propto Q/d^4$$

We can define a `WaterPipe` class where its objects have instance variables `diameter` and `pressure`. We will assume that the diameter is set when the object is created (using a *constructor*) and that the pressure and flow can be changed and retrieved.

```
/*
 *  (C) 2014  Ken Clowes
 */

package coe318;

/**
 *
 * @author Ken Clowes
 */
public class WaterPipe {
    private double pressure;
    private double diameter;
    private double d4;

    public WaterPipe(double d) {
        diameter = d;
        d4 = d*d*d*d;
    }

    public double getDiameter() {
        return diameter;
    }

    public double getPressure() {
        return pressure;
    }

    public void setPressure(double p) {
        pressure = p;
    }

    public double getFlow() {
        return pressure * d4;
    }
    public void setFlow(double flow) {
        pressure = flow / d4;
    }

    public static void main(String[] args) {
        WaterPipe wp1, wp2;

        wp1 = new WaterPipe(1);
        wp2 = new WaterPipe(5);
        System.out.println(wp1.getPressure());
        wp1.setFlow(20);
        System.out.println(wp1.getPressure());
    }

}
```

Object members and the simplest possible class (Nothing)

- Objects have both instance variables and methods.
- An instance variable and a method are both referred to a *members* of the object.
- An object's member (instance variable or method) is accessed by using the varible name of the object, followed by a dot (.) followed by the member name.
- Instance variable members have no parentheses whereas method names are followed by any parameters in parentheses.
- For example, if a new Dog object is created with `Dog rover = new Dog();` then its colour instance variable can be accessed with `rover.colour` and its speak method can be invoked with `rover.speak().` (We assume that both members are “public”).
- In general, an object has zero or more instance members and 0 or more method members.
- The simplest possible class would have zero members (no instance variables and no methods.)
- Here is such a class:

```
public class Nothing {
}
```

- This class compiles correctly and can even be used.
- For example:

```
public class NothingDemo {
    public static void main(String[] args) {
        Nothing a, b;
        a = new Nothing();
        b = new Nothing();
        System.out.println(a.equals(b));
        System.out.println(a.equals(a));
        System.out.println("a: " + a);
        System.out.println("b: " + b);
    }
```

- But how can we invoke the “equals” method when the Nothing class does not define such a method???

- The answer is that all classes automatically inherit a few default mehtods from the grandparent of all classes, the `Object` class which does define methods such as "equals(...)" and "toString()"
- We shall examine how "inheritance" works in object oriented languages (including Java) later in the course.

Notes on Lab 2

- In Lab 2 you complete the code for an *immutable* class: `ComplexNumber`.
- The following code for a vector in two dimensions illustrates how to do this.

```
/*
 *  (C) 2014  Ken Clowes
 */

package demovector2d;

/**
 *
 * @author Ken Clowes
 */

/** <code>Vector2D</code> represents an immutable vector in 2-D
 * space using Cartesian coordinates.
 */
public class Vector2D {
    private double x;
    private double y;

    /** Construct a Vector2D object with the given coordinates.
     *
     * @param x The x coordinate.
     * @param y The y coordinate.
     */
    public Vector2D(double x, double y) {
        this.x = x;
        this.y = y;
    }

    /** Get the x coordinate.
     *
     * @return The x coordinate.
     */
    public double getX() {
        return x;
    }

    /** Get the y coordinate.
     *
     * @return The y coordinate.
     */
    public double getY() {
        return y;
    }

    /** Calculate the dot product between <code>this</code>
     * and <code>other</code>.
     *
     * @param other
     * @return The dot product: this . other
     */
    public double dot(Vector2D other) {
        return x * other.x + y * other.y;
    }

    /** Returns a new Vector2D equal to the sum of
     * <code>this</code> and <code>other</code>.
     * @param other
     * @return this + other
     */
    public Vector2D add(Vector2D other) {
        return new Vector2D(x + other.x, y+other.y);
    }

    @Override
    /** Returns a String representation of the vector
     * in the form: (x, y)
     * @return "(x, y)" where x and y are the coordinates.
     */
    public String toString() {
        return "(" + x + ", " + y + ")";
    }
}
```

Questions

1. Consider the Vector2D class. Suppose we wanted to obtain the length of a vector with the method `getLength()`. Implement this method. (Note: you can calculate the square root of a

number in Java with `Math.sqrt(someNumber)`.

Answers

```
1.
   public double getLength() {
         return Math.sqrt(dot(this));
    }
```