Lab #5
CSE110 - Arizona State University

Topics

- Classes

Coding Guidelines

- Give identifiers semantic meaning and make them easy to read (examples numStudents, grossPay, etc).
- Keep identifiers to a reasonably short length.
- Use upper case for constants. Use title case (first letter is upper case) for classes. Use lower case with uppercase word separators for all other identifiers (variables, methods, objects).
- Use tabs or spaces to indent code within blocks (code surrounded by braces). This includes classes, methods, and code associated with ifs, switches and loops. Be consistent with the number of spaces or tabs that you use to indent.
- Use white space to make your program more readable.
- Use comments after the ending brace of classes, methods, and blocks to identify to which block it belongs.

Assignment/Lab Documentation

At the beginning of each programming assignment you must have a comment block with the following information:

```
/*-------------------------------------------------------------------------
// AUTHOR: your name
// FILENAME: title of the source file
// SPECIFICATION: description of the program
// FOR: CSE 110- Lab #5
// TIME SPENT: how long it took you to complete the assignment
//-----------------------------------------------------------*/
```

Getting Started

Object-oriented languages like Java are designed to make it easy for programmers to implement software versions of real-world objects. In learning Java, an important skill to master is the ability to represent an object in code. Objects that we model are described using Java classes, so we have chosen to begin this lab by modeling a very simple, everyday object: a door.

Part 1: Defining the Class

Write the code to create a class that models a `Door` object. Don't worry about the internal details just yet. Just give the class a name and an empty body for the moment. We will add more to the class shortly. For example, if we were creating a class to represent a `Window`, we would use the following:

```
public class Window {
    // To be filled in later
}
```

Part 2: Instance Variables

When modeling an object as a class, we also need to describe the properties it possesses. An everyday object that we wish to model always has one or more properties that describe it. For instance a door object might have a name like "Front" or "Side" to distinguish it from other doors. Another property that could describe a door is its state: "open" or "closed". Properties of objects are described in code by using nouns like "state" or "name" to create instance variables that hold values.

Add the following two instance variables to your `Door` class:

- a `String name` to hold the name of the door
- a `String state` to hold if the door is opened or closed

To protect encapsulation, be sure to make the two instance variables above `private` variables. For example, if the `Window` class example above had a name and a variable to hold information about the shades, we could have:

```
public class Window {
    // instance variables
    private String name;
    private String shades;

    // constructors and methods to be filled in later
}
```

Part 3: Constructor

Now that we have a `Door` class, we would like to be able to create some `Door` objects. Java constructors are components of a class whose purpose is to create objects of the given class and to initialize the object's instance variables. Java provides a constructor with no arguments (the "default" constructor) for every class and the `Door` class is no exception. Unfortunately, the default constructor that Java provides initializes the `state` and `name` variables to `null`. This is unacceptable.

Add a constructor for the `Door` class that receives two `String` arguments: the name of the door and its initial state. Inside the constructor, assign the two instance variables the value of the appropriate argument. Because we want to use the constructor outside of the class, make the access modifier for the constructor `public`.

Remember, the name of the constructor must match the name of the class exactly and the constructor does not have a `return` type, not even `void`.

For the `Window` example the following constructor would initialize the `name` and `shades` instance variables:

```
public Window (String newName, String newShades) {
    name = newName;
    shades = newShades;
}
```

Part 4: Accessor Methods

It is often convenient to have accessor methods that operate on a single instance variable. Here is an accessor method for the `name` variable of the `Door` class:

```
public String getName() {
    return name;
}
```

The word `String` in front of `getName()` indicates that the method returns a `String` when it is invoked. The body is simple and just returns the value of `name`. Add this method to your class and write a similar accessor method for the `state` instance variable.

Part 5: Mutator Methods

Many instance variables in a class will have corresponding mutator methods that allow you to change the value of the variable. Here is a mutator method for the `state` variable:

```
public void setState(String newState) {
    state = newState;
}
```

The word `void` in front of `setState()` indicates that the method does not return a value when it is invoked. The body is simple and copies the value of the parameter variable `newState` to instance variable `state`. Add this method to the class and write a similar mutator method for the `name` instance variable.

Part 6: Other Methods

Objects also have operations which can be invoked on the object and which may change the object in some way. For instance, the operations for a door object could be "open" and "close". An operation on an object corresponds to a Java method and is described in code by using a verb like "open" or "close". Invoking a method may change the value of an instance variable. For example, invoking `close()` would change the value of the `state` variable from "open" to "closed". For example, the `open()` method could look like:

```
public void open() {
    state = "open";
}
```

Declare the following two methods in your `Door` class:

- a method named `open()` that takes no arguments, doesn't return a value and sets the value of the instance variable `state` to "open".
- a method named `close()` that takes no arguments, doesn't return a value and sets the value of the instance variable `state` to "closed".

Because we usually want to allow free access to the methods a class contains, make the access modifier for each method `public`.

Part 7: Testing your Door Class

Use the following driver program (also available from the class website under Lab 5) to test your `Door` class. This file has to be saved as `Lab5.java`. Make sure you save the file in the same folder as your `Door.java` file (or in Eclipse create both classes in the same project). Compile and run the code below:

```
/**
 A class to test the Door class.
*/
public class Lab5{
    /**
    Tests the methods of the Door class
    @param args not used
    */
    public static void main(String[] args) {
        // Create a new Door object named ?Front? with initial state ?open?
        Door frontDoor = new Door("front", "open");
        // Print a check that the constructor created the object correctly
        System.out.println("The " + frontDoor.getName() + " door is " + frontDoor.getState());
        System.out.println("Expected: open");
        // Create a second Door object with a name ?Back? and state ?closed?
        Door backDoor = new Door("back", "closed");
        // Print a check that the constructor created the object correctly
        System.out.println("The " + backDoor.getName() + " door is " + backDoor.getState());
        System.out.println("Expected: closed");
        // Test the open() method to open the backDoor
        backDoor.open();
        System.out.println("The back door is " + backDoor.getState());
        System.out.println("Expected: open");
        // Use the mutator to change the name variable
        backDoor.setName("kitchen");
        System.out.println("The back door is now called" + backDoor.getName());
        System.out.println("Expected: kitchen");
        // Complete the class as described in Part 8 of the Assignment
    }
}
```

Part 8: Instantiating a new Door Object

Create a third `Door` object in your `Lab5.java` program to represent a `sideDoor` with the name property "side" and an initial state of "open". Verify that the object was properly created by printing out its name and state and the expected values (as done above for the `frontDoor` and `backDoor`). Use the `close()` method to change the state of the `sideDoor` to "closed". Verify that the method is working by printing out the new value stored in the state and the expected value (as done for `backDoor`).

Sample Output

Below is an example of what your output should roughly look like when this lab is completed. All text in bold represents user input.

Sample Run 1:
The front door is open
Expected: open
The back door is closed
Expected: closed
The back door is open
Expected: open
The back door is called Kitchen
Expected: Kitchen
The side door is open

Expected: open
The side door is closed
Expected: closed

Submission

Submit your `Lab5.java` and `Door.java` to the Submission Server. Go to the Submission Server site located on the course website, login, then click on Lab Submissions in the left frame. Choose Lab5 from the drop-down box, click on the browse button and find where you saved your `Lab5.java` and `Door.java` on your computer. Upload the file to the site and then click on the Submit button.

Your file will be submitted and a screen will show up displaying if your program compiled and what your output is when run on some sample input.

You should then check to make sure that the actual file submitted properly and is readable to the grader. To do so click on Grades in the frame on the left of the page and then click on the 0 underneath Lab5. You will again see that your program compiled and the sample output, but you should scroll down to the bottom of the screen and make sure your file is readable as well.

Important Note: You may resubmit as many times as you like until the deadline, but we will only mark your last submission.