Software and Programming II (SP2) — Lab sheet 4

Objects and Classes

2024/25

Based upon exercises from Java for Everyone, 2e.

---

1. Object-oriented languages like Java are designed to make it easy for programmers to implement software versions of real-world objects. When we are 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 modelling a very simple, everyday object: a door.

   Write the code to create a class `Door` 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.

2. When modelling 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* for instance variables that hold values.

   Add instance variables to your `Door` class for the name of the door and its state. Experience shows that we almost always want to limit the visibility of instance variables to inside the same class, so make the access modifiers of the two variables `private`. As the `state` and `name` properties have values like *open* or *front*, the instance variables you create should be of type `String`.

3. Classes 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*.

   Declare methods for open and `close`. We usually want to allow free access to the methods a class contains, so make the access modifier for each method `public`.

4. 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 initialise the object's instance variables. Java provides a constructor with no arguments (the "default" constructor) for every class that does not define its own constructors, and the `Door` class is no exception. Unfortunately, the default constructor that Java provides initialises the `state` and `name` variables to `null`; this is not what we want.

   Add a constructor for the `Door` class that receives two arguments: the name of the door and its initial state. We want to use the constructor outside of the class, so make the access modifier for the constructor `public`.

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

   ```
   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 instance variable `state`.

6. 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 `name` variable:

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

   The word `void` in front of `setName()` 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 `newName` to the instance variable `name`.

   Add this method to the class and write a similar mutator method for the instance variable `state`.

7. Compile and run the code below (it is also available on Moodle):

```
/**
 * A class to test the Door class.
 */
public class DoorTester {

    /**
     * Tests the methods of the Door class
     * @param args not used
     */
    public static void main(String[] args) {
        Door frontDoor = new Door("Front", "open");
        System.out.println("The front door is " + frontDoor.getState());
        System.out.println("Expected:  open");
        Door backDoor = new Door("Back", "closed");
        System.out.println("The back door is " + backDoor.getState());
        System.out.println("Expected:  closed");
        // Use the mutator to change the state variable
        backDoor.setState("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 called " + backDoor.getName());
        System.out.println("Expected:  Kitchen");
    }
}
```

   Create a third `Door` object called *sideDoor* with the name property *Side* and an initial state of *closed*. Verify that the object was properly created. Use the mutator to change the state of object `sideDoor` to *open*. Verify that the mutator is working.

8. Consider the variable `state` in the class `Door` and the variable `newState` in the mutator for `state`. What kind of variable is `state`? What kind of variable is `newState`? When do these variables exist?

9. Consider the line below:

   ```
   backDoor.setState("open");
   ```

   What is the implicit parameter that is passed by this method call? What is the explicit parameter?

---

10. To give you an opportunity for extra practice with writing classes, the second part of this lab sheet is about implementing a (simplified) *vending machine* that sells cans of soft drink. To buy a can of drink, the customer needs to insert a token into the machine. When the token is inserted, a can drops from the can store into the product delivery slot. The vending machine can be filled with more cans. The goal is to track how many cans and tokens are in the machine at any given time.

    What methods would you supply for a `VendingMachine` class? Describe them informally.

11. Now translate those informal descriptions into Java method headers, such as:

    ```
    public void fillUp(int cans)
    ```

    State the names, parameter variables, and return types of the methods. Do not implement them yet.

12. What instance variables do the methods need to do their work?

    Hint: You need to track the number of cans and tokens. Declare them with their type and access modifier.

13. Consider what happens when a user inserts a token into the vending machine. The number of tokens is increased, and the number of cans is decreased. Implement a method:

    ```
    public void insertToken() {
       // Instructions for updating the token and can counts
    }
    ```

    You need to use the instance variables that you defined in the previous step.

    Be sure to check that the number of cans is greater than zero before decreasing the can count and increasing the token count (otherwise nothing is supposed to happen).

14. Now implement a method `fillUp(int cans)` to add more cans to the machine. Simply add the number of new cans to the can count.

15. Next, implement two methods, `getCanCount` and `getTokenCount`, that return the current values of the can and token counts.

16. You have now implemented all methods of the `VendingMachine` class. Put them together into a class, like this:

    ```
    public class VendingMachine {
       private your first instance variable
       private your second instance variable

       public your first method
       public your second method
       . . .
    }
    ```

17. Now complete the following tester program so that it exercises all of the methods of your class:

```
public class VendingMachineTester {
   public static void main(String[] args) {
      VendingMachine machine = new VendingMachine();
      machine.fillUp(10); // Fill up with ten cans
      machine.insertToken();
      machine.insertToken();
      System.out.print("Token count: ");
      System.out.println(machine.getTokenCount());
      System.out.println("Expected: . . .");
      System.out.print("Can count: ");
      System.out.println(machine.getCanCount());
      System.out.println("Expected: . . .");
   }
}
```

18. So far, the `VendingMachine` class does not have any constructors. Instances of a class with no constructor are always constructed with all instance variables set to zero (or `null` if they are references to objects or arrays). It is always a good idea to provide an explicit constructor.

    Provide two constructors for the `VendingMachine` class:

    (a) a constructor with no arguments that initialises the vending machine with 10 drink cans;

    (b) a constructor, `VendingMachine(int cans)`, that initialises the vending machine with the given number of cans.

    Both constructors should initialise the token count to 0, and both constructors should have `public` visibility. The first constructor should call the second one.