The file `/home/hajiamini/courses/CSC207/code/Switch.java` contains a class definition for objects of a familiar and particularly simple kind: two-position switches, like the power switch on a vacuum cleaner.

A **Switch** has one bit of internal state: It can be either in the “on” position or in the “off” position. Its repertoire of behaviors is likewise quite limited: It has a **report** method that tells the caller what the state of the switch is and a **turn** method that allows the caller to turn the switch off or on (that is, to force the state of the switch to a specified value). The one extra feature that is supported is a **toggle** method that changes the current state of the switch to the opposite value.

1. Copy the `Switch.java` into your own directory and run `javac` to compile it. Use `cp` command for copying files.

2. Write the definition for a **SwitchTester** class with a public, static **main** method that determines and reports the state of a newly created **Switch** object.

3. (1 point) In the **main** method (or, if you prefer, in a separate method that **main** invokes), create a new **Switch** object, assign it to a variable, and then assign that variable to a second variable. The textbook asserts that these “reference variables” are now referring to one and the same **Switch**. Test this claim by changing the state of the switch that one variable refers to and then inspecting the state of the switch that the other variable refers to. What result do you expect to see if the textbook’s assertion is correct?

4. Design and implement a similar test to verify or refute the textbook’s assertion that when an object (such as a **Switch**) appears as an argument in a method call, what is actually passed to the method is a reference to that object rather than a separate copy of it, so that side effects on the object’s state occurring inside the method body persist after return from the method.

5. Design and implement a similar test to determine whether two separately created switches can be toggled independently, without changing the state of the other.

6. (1 point) Design and implement a test to verify or refute the textbook’s assertion that two separately created switches are not “equal” (as determined by the `==` operator) even if they have the same internal state.

7. Design and implement a test as a static method that takes a **Switch** object as its argument and returns a String value — either "on" or "off", depending on the state of the switch. Implement this method inside **SwitchTester** class.

8. (1 point) Extend the class definition of **Switch**, adding a new public method called **sameState** that takes another **Switch** as its argument and determines whether the state of that **Switch** is the same as the state of the **Switch** within which the method is called. For example, after the declarations

   ```java
   Switch power = new Switch();
   Switch lights = new Switch();
   ```

   the call `power.sameState(lights)` should return `true` if the two switches are both on or both off, `false` if one is on and the other is off.

   Recompile the **Switch** class definition and design and implement unit tests for the **sameState** method, adding them to the **main** method of your **SwitchTester** class. Recompile and run **SwitchTester** as a program.
9. Extend the class definition of `Switch`, adding a new method called `conformToConsensus` that takes three switches as arguments and causes the `Switch` within which the method is called to adjust its internal state, if necessary, to match the state of the majority of the arguments (turning itself on if any two of the given switches are on and off if any two of the given switches are off). Create and run unit tests for the `conformToConsensus` method.

10. It is possible for a class to have two or more overloaded constructors, provided that their parameter lists are different enough that the Java compiler can determine unambiguously, by looking at the number and types of the arguments, which of the constructors is being called.

    Add to the definition of `Switch` a second constructor that takes a Boolean value as its argument and initializes the `on` field of the newly constructed `Switch` to that Boolean value.

    Revise the `SwitchTester` program so that it tests this new constructor.

11. Add two `static` fields, `OFF` and `ON`, to the definition of the `Switch` class, each denoting one of the two Boolean values that can be stored in the `on` field of a `Switch` object. (This enables application programmers to write, for instance, `'Switch.OFF'` rather than `'false'` when supplying a parameter to the `turn` method.) Initialize these fields when declaring them in the `Switch` class. (Static fields are typically declared as `public`.) Revise the `SwitchTester` program to use these new constants.