Harvey Mudd College
Computer Science 60
Fall 2010

Assignment 6
Sequential Logic Simulator
Due. 11:59 p.m., Wed., 14 October 2010

Using the Java language, construct a sequential logic circuit simulator, according to the javaDoc to be found at:

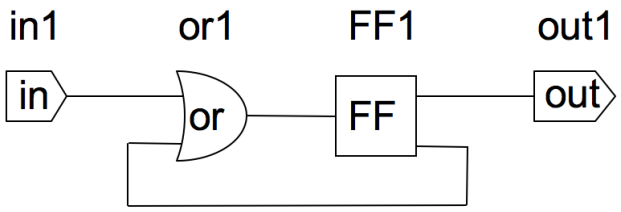
http://www.cs.hmc.edu/courses/2010/fall/cs60/sequentialLogicDoc/

Read the documents with **Frames** turned on, so that navigation between Java classes is convenient. Because this may be your first exposure to Java, I will give you some very specific hints. The general idea of sequential logic will be discussed in the classroom, but not explained here in its entirety.

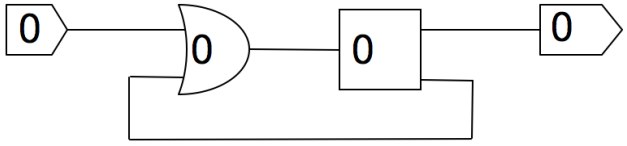
Your submission should pass all of the jUnit tests in a file that that I will provide at:

http://www.cs.hmc.edu/courses/2010/fall/cs60/CircuitTest01.java

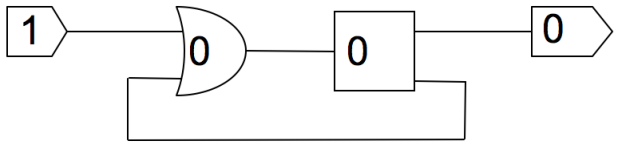
You may think of a logic circuit as a "directed graph", meaning a set of nodes connected by directed arrows. The nodes will be of various types: gates, flip-flops, and terminals. Each node has a *state* represented by a single Boolean. The diagram below is an example of a circuit that can be constructed. Inputs to a node are on the left, and outputs on the right.



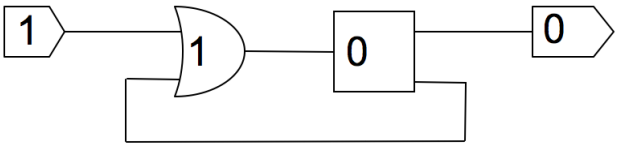
These circuits operate by virtue of a "clock", which is not shown explicitly, but which emits a never-ending sequence of "ticks". **At each tick**, each flip-flop state becomes the logic value of the unique node that is input to it. **Between ticks** gates change their state to according to the logic function of the node. For example, suppose the circuit above is in state shown below:



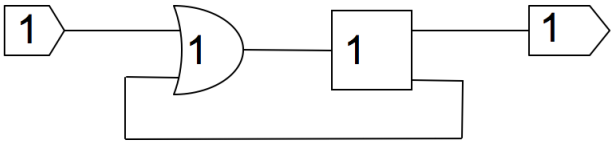
The state of the "or" stays at 0 (representing "false"), because both of its inputs are 0. Now suppose that an external agent sets the inTerminal to 1, resulting in the following state:



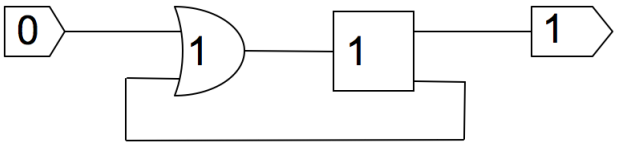
The state of the "or" will now change to 1, because at least one of its inputs is 1:



The state of the flip-flop will not change until the clock ticks. At that point, it will become 1 (representing "true"), the output of the "or", and then the state of the outTerminal will become 1 too.



Now suppose the external agent sets the inTerminal back to 0. The state of the "or" will remain at 1, because of the second input from the flip-flop.



It is easy to see that the state of the flip-flop will remain at 1 forever, regardless of what happens at the inTerminal. Thus this circuit "remembers whether it has ever seen a 1" at its input. It has a *sequential* behavior, in contrast to "combinational" logic circuits, which don't have flip-flops or loops.

In modeling circuit behavior, we will, on each "tick" update the gates (non-flip-flops) to their implied values, then update the flip-flops. In order to do this efficiently, updating proceeds from inTerminals toward the flip-flops, i.e. in a topological order of the nodes. In this assignment, making sure that updates occur in the correct order will be the responsibility of the user. In a future assignment, we will automate this requirement.

The corresponding logic simulator construction is shown in the following unit test:

```
/**
 * Test sequential circuit that remembers whether
```

```
     * input was ever true.
     */

    public void test01l()
    {
      sequentialLogic.Circuit circuit =
            new sequentialLogic.Circuit("test01l");

      circuit.addNode("in01",  "inTerminal");
      circuit.addNode("or01", "or");
      circuit.addNode("FF01", "FF");
      circuit.addNode("out01",  "outTerminal");

      circuit.connect("in01", "or01");
      circuit.connect("or01", "FF01");
      circuit.connect("FF01", "or01");
      circuit.connect("FF01", "out01");

      circuit.setValue("in01", false);
      circuit.update("or01");
      circuit.update("FF01");
      circuit.update("out01");
      assert( !circuit.getValue("out01") );
                          . . . Truncated for Brevity . . .
  }
```

Each test consists of the following parts:

- Calling the constructor for class Circuit
- Adding nodes to the circuit
- Adding connections to the circuit
- Simulation of the circuit thus constructed

If the new Circuit has been assigned to variable `circuit`, then a command such as

```
circuit.addNode("or01", "or");
```

adds a node of type `"or"`, giving it name `"or01"`. Each node must be named uniquely. This system should generate an error if an attempt is made to name more than one node the same. This restriction is needed because names are the way of identifying nodes.

The types of nodes are listed below. Each corresponds to a separate Java class, as you can infer from the Javadoc.

- Node
  - OneInputNode
    - FlipFlop
    - OneInputGate
      - Invertor
      - InTerminal
      - OutTerminal
  - MultiInputNode
    - MultiInputGate
      - AndGate
      - OrGate
      - XorGate

Connections are made from the output of one node to the input of another. In this problem, all the gates each represent symmetric functions, so the order of inputs does not matter. A command such as

```
circuit.connect("or01", "FF01");
```

connects the output of the node named `"or01"` to the input of the node named `"FF01"`. It is required that both nodes have been added before they can be connected. If this is not the case, an error should be generated.

Once all connections have been made, simulation can take place. One or more input terminals have their states set using a command such as:

```
circuit.setValue("in01", false);
```

Individual nodes have their values updated to correspond to changed inputs by a command such as:

```
circuit.update("or01");
```

A related command:

```
circuit.update();
```

updates all gates in the entire circuit. Currently the circuit elements need to be added in "topological sort" order, from the InTerminals to the OutTerminals, in order to be able to use this update. In a future assignment, we will look at relaxing this restriction, by automating the topological sort.

The Java classes for nodes are arranged in an "inheritance hierarchy", according to functionality. The hierarchy is a tree, as shown earlier. You can verify the hierarchy from the Javadoc by noting the word `extends` in the subordinate classes. When one class extends another, it can use the methods of classes higher up in the hierarchy, unless such use has been prohibited by notation `private` .

You must implement some of the code for these classes. I will leak selected code to show you how it is done.

I will provide all code for these.	**I will provide some code for these. You will provide the rest.**	**You will provide all code for these.**
OrGate	Circuit	AndGate
Error		XorGate
FlipFlop		Invertor
OneInputNode		Connector
OneInputGate		InTerminal
Node		MultiInputNode
OutTerminal		MultiInputGate

For example, look at the code for Node and OneInputNode to see how to handle MultiInputNode. The Circuit class will use a HashTable to associate Nodes with Strings. However, a HashTable does not maintain any particular order, so you will need a separate structure, such as a LinkedList to handle this.

It will be like completing a partially-completed jigsaw puzzle, and hopefully more fun!

End-user errors in this assignment are reported by calling

```
new Error(...message...);
```

This will terminate the program. Later we will consider friendlier ways to report errors.

When your assignment is complete, make sure you've omitted nothing by deleting all .class files, recompiling, and running the unit tests one final time. You will get no credit if your sources do not compile.

Given that you've done this final check, delete all .class files one more time. Then "zip" all of the .java files, including the unit tests, into a single file, a06.zip and submit it on the submission site.