Computer Engineering Fundamentals
Assignment 2.

Engr 9859, 2013

Due Oct 3 @ 11:00PM.

For each question you will be marked on programming style as well as correctness. To see my opinion about what constitutes good programming style see http://www.engr.mun.ca/~theo/Courses/ds/pub/style.pdf. In short:

- All .java files must be professionally commented; in particular, each file should contain a comment header that gives your name, student number, and mun email address. Each subroutine and class should have a comment at the start of it. I encourage you to use the "javadoc" conventions for comments.
- Code and comments must be consistently indented; tab stops should be set every 4 characters.
- Names must be chosen carefully and spelled correctly. (Use names starting with lower case letters for variables and methods; use names starting with upper case letters for classes and interfaces.)
- Use subroutines to avoid redundant coding.
- Keep control structures and data structures simple.

All classes must be tested by you prior to being submitted. You are welcome to share test code with each other.

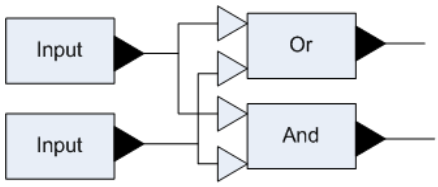
The assignment is to be done alone. Each file should contain the following declaration in comments near the top. "This file was prepared by [your name here]. It was completed by me alone.". If you obtained help in doing the assignment, do not include this declaration, but rather an explanation of the nature of any help that you received in doing the assignment.

Q0. In this assignment you will create a family of related concrete classes around an abstract class. Together we will create a simulator for synchronous (i.e. 'clocked') digital circuits. I have done some of the work; your job is to implement the device classes.

Ok, here is how it goes: A network is a collection of devices and nets. A net is basically a wire that transports a boolean value between devices. Devices do the computation.

Each device owns a fixed set of ports. For example a 2-input AND gate owns 3 ports. Each port is classified as an input port or an output port. For example, a 2-input AND gate has 2 input ports and 1 output port. Relative to each device, ports are known simply by number, counting from 0. For example, for a 2-input AND gate, port 0 is the output, while ports 1 and 2 are input ports.

Each net is connected to a number of ports. There are two rules that the code should check[1] at run-time: Each net is connected to at most 1 output port; and, at simulation time, each net is connected to exactly 1 output port. Also each input port should be connected to exactly 1 net, although the code need not check this.



A network with 4 device, 4 nets, and 8 ports

The simulator works as follows. The values are stored on the ports. For each clock cycle, the simulator does the following.

- First, it sends a 'clock' message to each device. The 'clock' message tells the device that a clock edge is passing. In response to the 'clock' message, each device updates its output ports' values as appropriate.

- Second, it alternates between sending a 'update' message to each net and an 'update' message to each device, until the entire network has settled to a stable state. Each net responds to an 'update' message by copying the value on its output port to all its input ports.[2] Each device responds to a 'update' message by calculating a new value for its each of its output ports based on its input ports. Once the network has settled —i.e. once no ports change in response to an 'update' message— the simulator prints the values of all the nets and proceeds to the next clock cycle.

The simulator uses 3 valued boolean logic. The three values are FALSE, TRUE, and UNKNOWN.

Here is description of all the devices.

- 'Input devices'. Input gates have one output port and no input ports. These are used to feed a sequence of values into the net. In response to the first clock edge, the input device outputs the first value, in response

[1] These checks are made using the Assert.check method. I've already implemented these checks.

[2] The terms 'output port' and 'input port' are relative to the device. To the net, an 'output port' is its input source and its 'input ports' are its output destinations.

to the second clock edge, the input device outputs the second value, and so on.

- 'and', 'or' and 'not' gates. These have one output port (port 0) and 1 or 2 input ports (ports 1 and, if needed, 2). Gates ignore clock edges. In response to each "update" message, they update their output ports according to the following truth tables

and	F	U	T
F	F	F	F
U	F	U	U
T	F	U	T

or	F	U	T
F	F	U	T
U	U	U	T
T	T	T	T

not	
F	T
U	U
T	F

- D-flip-flops. D stands for 'delay'. D-flip-flops have one output port (port 0) and one input port (port 1). On each clock edge, the input value is transferred to the output. On update, there is no change to the output.

I've implemented most of this. It remains to implement the 5 types of devices described above, including the ports, and to finish writing the factory class that creates them. As you can see, the devices have many things in common; **you should use an abstract class to capture what is common to all devices.**

For details about devices and ports, read the following interfaces: PortInterface, DeviceInterface, InputDeviceInterface in package digital.interfaces.

All classes must be in package digital.implementation.

Wherever possible, preconditions should be checked using the Assert.check method in package assertions.

Advice: Start by implementing the PortInterface interface and the InputDeviceInterface; also complete the corresponding methods in DigitalFactory then you should be able to run progam SimulateCircuit0. Next tackle D-flip-flops, being careful to put anything in common between input devices and D-flip-flops in an abstract class. You should then be able to run SimulateCircuit1. Finally implement the gates, making use of your abstract class.

Q1.

For this question, you will write a part of a graphical user interface program. See Horstman and Cornell for information about Java's Swing library. You are to write a class called gomoku.Model that implements interface called gomoku.ModelIntf. This class tracks the state of a game of Gomoku. This is essentially tic-tac-toe except that (a) it is played on a bigger board and (b) the goal is to put 5 of your own token in a row. I will distribute the ModelIntf and also a simple GUI class called gomoku.View. You may modify either of these types, if you like, but I will be testing your model against the original ModelIntf file.