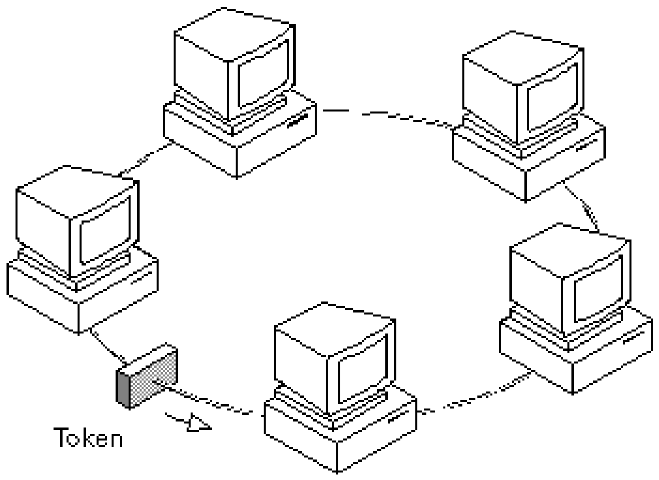
CMPSCI 187 Programming With Data Structures

Token Ring Simulator

Assigned: Monday, March 31, 2014
Due: Monday, April 11, 2014

Overview

In this programming exercise you will complete the implementation of a simple network simulator for the Token Ring protocol. Token ring is a local area network (LAN) technology whose network is logically a ring of connected workstations. More specifically, each workstation is connected one after the other by a cable (shielded twisted pair) until the last workstation is connected to the first. An empty "token" is then circulated continuously around the token ring network until a workstation needs to communicate to another workstation. When the token reaches a workstation A that needs to "talk" to another workstation B, A puts data in the token and then continues to circulate the token around the ring until it reaches workstation B. Workstation B then receives the data, updates the token to indicate that the data has been received, and then continues to circulate the token around the ring until it is once again received by the sender (A).



The sender will acknowledge that B received the data and then circulate an empty token around the ring for it to be used by the next sender (which could be A again). The token will always circulate in the same direction which is usually dictated by the hardware vendor, but it does not matter which direction (clockwise vs. counterclockwise). You can see a visual of a token ring network in action and this one allows you to modify the number of messages and how fast the simulation runs.

Files To Complete

You are expected to write an implementation for the classes discussed below. In addition, you will be required to fill in the static methods found in the `config.Configuration` class. This class is used by the token ring simulator and our testing facility to grade the classes that you implement.

In addition to this, you **MUST** include a String field in the `config.Configuration` file with your UMass student ID number. This is the 8 digit value found on your student ID card and is used to identify you in moodle. This will allow us to get your grade and feedback to you quickly. We recommend that you enter this ID value immediately after importing the project. Any class file you write that is used by your solution **MUST** be in the provided `src` folder. When we test your assignment, all files not included in the `src` folder will be ignored.

Note: When you submit your solution, be sure to remove **ALL** compilation errors from your project. Any compilation errors in your project may cause the autograder to fail and you will receive a zero for your submission.

Test Files

In the `test` folder, you are provided with JUnit test cases that will help you keep on track while completing this assignment. We recommend you run the tests often and use them as a checklist of things to do next. You are not allowed to remove anything from these files. If you have errors in these files, it means the structure of the files found in the `src` folder have been altered in a way that will cause your submission to lose points. We highly recommend that you add new `@Test` cases to these files. However, before submitting, make sure that your program compiles with the original `test` folder provided.

In addition to tests for the classes you need to implement we also provide two classes in the `test` directory for executing the simulator:

> **TokenRingSimulateClockwise10000** - tests the simulation by setting the direction of the frames in the token ring network to clockwise
> **TokenRingSimulateCounterClockwise10000** - tests the simulation by setting the direction of the frames in the token ring netowkr to counter-clockwise

You should look at these two files to understand how the simulation is constructed from the ground up. You can see the output of our solution in the files `clockwise-run.txt` and `counterclockwise-run.txt` in the main project folder.

We can represent a token ring network using a doubly circular linked list of workstations. We use a circular linked list to represent the logical circle of a token ring network and a doubly linked list so we can send the token either clockwise or counterclockwise. Your task is to implement a `Workstation` and a `DoublyCircularLinkedList`.

Support Code API

The Support Code's comments have been generated into a nicely formatted API that can be found online. It is highly recommended that you spend a day simply reading over the comments in each of the interfaces and classes provided.

Part One: Importing Project Into Eclipse

Begin by downloading the eclipse project.

You should now have a project called *assignment-token-ring-student*. it is very important that you do not rename this project as it is used during the grading process. If the project is renamed, your assignment may not be graded.

By default, your project should have no errors and contain the following root items:

> **src** - The source folder where all code you are submitting must go. You can change anything you want in this folder, you can add new files, etc...
> **src/support** - This folder contains support code that we encourage you to use (and must be used to pass certain tests). You are not allowed to change or add anything in this folder
> **test** - The test folder where all of the public unit tests are available
> **JRE System Library** - This is what allows java to run
> **JUnit 4** - A library that is used to run the test programs
> **javadoc** - A local copy of the API documentation. You can open the documentation in a web browser by opening the `index.html` file in this directory. You can also access this documentation online.
> **writeup** - A local copy of this document.

If you are missing any of the above or errors are present in the project, seek help immediately so you can get started on the project right away.

Once you have your project imported, add your 8 digit student ID number to the `config.Configuration` class.

Part Two: Implement a `DoublyCircularLinkedList` class

The fundamental data structure we will use to implement a token ring network is a circular doubly linked list. Your class must implement the `DoublyCircularLinkedList` interface found in the `structures` package.

You will notice that this interface extends the `CircularLinkedList` interface, which in turn extends the interface `ListInterface` defined by the book. You should create a new class in the `structures` directory that implements the `DoublyCircularLinkedList` class.

To test your implementation you will need to complete the `newList()` method found in the `config.Configuration` class. Read the comments for this method to see what you need to do. After you correctly implement this method to return an instance of your circular doubly linked list implementation you can run the JUnit test case, `structures.DoublyCircularLinkedListTest`, in the `test` directory.

Part Three: Implement the `BasicWorkstation` class

This next part will have you complete the implementation of the token ring simulator. In this simulation environment any entity that can be simulated will implement the `Simulatable` interface. This interface requires an implementing class to provide a single method called `tick()`. This method allows each simulatable object to simulate itself when it is called by the `simulate` method (found in the `Engine` class) to advance the simulation by one time step. The `Engine` class controls the simulation and also provides methods to record information about the simulation.

There are three classes that implement the `Simulatable` interface,

NetworkInterface - this class represents a network interface card (NIC). It is the hardware that sits between the token ring network and the workstation.
TokenRing - this class represents the token ring network. The implementation uses a doubly linked list of `Workstation` objects to represent the network. You should study this class to understand how it interacts with the `Workstation` class to help you with your implementation.
Workstation - this is the class that you must extend. It is an abstract class that represents a workstation in the network. It is associated with a specific network interface card that it uses to communicate with the token ring network.

You will need to extend the `Workstation` class to provide a concrete implementation called `BasicWorkstation`. The following outlines the implementation details for the important methods.

BasicWorkstation(NetworkInterface nic)

The constructor for this class. You will need to keep a reference to `nic` so it can be used by methods described below. You will also want to create an instance variable for a queue data structure. You are not required to implement the queue in this assignment, rather, you can use Java's built-in LinkedList class which implements the Queue interface. Consult the documentation to see which methods you need to use to treat the linked list as a queue. You will use this queue to store messages (`Message`) that the workstation will be sending to other workstations on the same token ring network.

getNIC()

Returns the network interface of this workstation.

compareTo(Workstation o)

This is a required method of the `Comparable` interface. This method should compare workstations by their id. The id of a workstation is provided by the `Workstation` class.

equals(Object obj)

You need to override this method. The default implementation found in the `Object` class defines equality by reference. Your implementation should define equality as being two workstations that have the same id.

sendMessage(Message m)

This method is used by the `Engine` class to deliver a message to the workstation from the simulation environment. The `Message` should be queued in your queue data structure.

tick()

This method defines what a workstation can do during a single time step. You should refer to the `TokenRing` class to see what it does to get a better idea of how to implement this method. In general, a workstation will do nothing unless it has a frame available on its network interface (NIC) (see the `hasFrame` method in the `NetworkInterface` class. If the NIC has a frame the workstation will read the frame from the NIC.

Next, the workstation will need to determine what to do with the frame. This depends on which type of frame it is. If the frame is a `TokenFrame` (see the `Frame` interface) and the workstation does not have any messages queued up it simply writes the frame back to the NIC. If the frame is a `TokenFrame` and the workstation does have messages queued up you will want to dequeue a message from the queue, create a new `DataFrame`, and write the data frame to the NIC. You will also want to update the number of messages sent by this workstation by calling `incMsgSent()`, a method defined by the `Workstation` class. This will record the number of messages sent by the simulation and generated as output in the end.

If the frame we read from the NIC is a `DataFrame` we will want to determine if the contained message should be delivered to this workstation. You can get the `Message` from the `DataFrame` by invoking its `getMessage()` method. Once you have the message there are four possible scenarios:

1. The message's receiver has the same id as this workstation. If this is the case you will want to print out the message:

   ```
   message MESSAGE received by RECEIVER; sent by SENDER
   ```

   The MESSAGE is the result of calling `Message.toString()`, the RECEIVER is the id of the receiver (this workstation), and SENDER is the id of the sender. All of this information can be obtained from the `Message` object.

2. The message's sender has the same id as this workstation and the data frame was received (`DataFrame.wasReceived()`) by the destination workstation. If this is the case we need to acknowledge the message by printing the following message:

   ```
   message MESSAGE acknowledged by sender SENDER from destination RECEIVER
   ```

   Now that the message was sent and received we will follow the token ring protocol by writing a `TokenFrame` to the NIC. You can get access to the token frame using `TokenFrame.TOKEN`. Lastly, record that the message was delivered by invoking the `incMsgDelivered()` method, implemented by the `Workstation` class.

3. The message's sender has the same id as this workstation and the data frame was not received by the destination workstation. If this is the case you will print out the following to indicate that the message will be dropped:

   ```
   message MESSAGE dropped; destination not reachable
   ```

   Finish this case off by writing the `TokenFrame.TOKEN` to the NIC.

4. The message's receiver does not have the same id as this workstation. In this case this workstation is not the intended recipient so we write the received frame back to the NIC. The token ring will then forward it along to the next host.

Part Four: Complete the `config.Configuration` class

This one is easy. In order for us to gain access to your implementations you need to complete the methods found in the `config.Configuration` class. Each of the methods are static and describe what you must return for the simulation and tests to work properly. You will notice that there is a method called `newList()` that will return your circular doubly linked list, the method `newTokenRing()` that will return a token ring network (we have provided this one), and `newWorkstation()` that will return a new workstation. Each of these methods must be filled in properly for your submission to be graded.

Part Five: Export and Submit

When you have finished and are ready to submit, export the entire project. Be sure that the project is named *assignment-token-ring-student*. Save the exported file with the zip extension (any name is fine). Log into Moodle and submit the exported zip file.