CS 2112—Spring 2014

Assignment 7

Distributed and Concurrent Programming

Due: May 7, 11:59PM

In this assignment, you will build a web service that simulates the Critter World, by building on your code from Assignment 5. You will also modify the GUI you built for Assignment 6 to communicate with this server over HTTP, the Hypertext Transfer Protocol. This separation of UI and simulation will allow multiple people to interact with the same Critter World running on the same server.

We also expect you to fix problems in your previous assignment submission. In fact, about 60% of your grade will be based on whether the functionality from Assignments 4–6 is correctly implemented.

Finally, you will implement a small concurrent data structure.

0 Changes

- Nothing significant yet.

1 Major Challenges

This assignment aims to expose you to the challenge of building distributed applications. You will also need to use a wider range of third-party libraries (which may be buggy or badly documented!). And you will need to change your already complex existing code to deal with some very different requirements.

You can also expect to learn about the Hypertext Transfer Protocol (HTTP) and JavaScript Object Notation (JSON), and the Java servlet framework. And we will ask you to write thread-safe code.

If you have kept a strong separation between the model and view/controller parts of your code during A6, this task should be relatively straightforward, though you may have to extend both model and view to implement the HTTP protocol that we have defined.

If these tasks seem daunting to you, here is our advice: start early, and make sure you attend Lab 12 to get started on servlets.

2 HTTP Overview

Disclaimer: the following description is an simplification that omits many real-world details that are not important for this assignment. CS4410 and other networking-related courses can teach you more about HTTP.

A HTTP web service is a computer program that accepts *HTTP requests* over the network and sends back *responses*, which are usually *HTML documents* that your browser can then render to

human-viewable pages. We refer to the entity that sends requests as the *client*, and the program that returns responses as the *server*. In this assignment, you need to write a server that simulates the Critter World, and modify your GUI to be a client that queries the server for the state of the World.

A HTTP request has several components, of which the most important are described below:

- URL (Uniform Resource Locator). Most people already know URL as the string you enter into your browser to locate a website, like `http://www.google.com`. To continue with this example, when you enter `http://www.google.com` in your browser, a HTTP request is sent from your computer to another computer operated by Google. This computer, known as the server, processes your request and returns a web page.

  One complication of URLs is that they often contain *query strings*. A query string is the part of a URL that contains data to be passed to web applications. For instance, this is the URL you might be directed to when you search for "hello" in Google via the browser's search bar:

  `https://www.google.com/search?client=ubuntu&channel=fs&q=hello&ie=utf-8&oe=utf-8`

  The part after "search" is the query string.

  Having received the HTTP request, the Google server looks at this query string and learns that the user is using the Ubuntu operating system (`client=ubuntu`), that the search was for "hello" (`q=hello`), and that the input and output encodings are both UTF-8 (`ie=utf-8,oe=utf8`). In this project, your client will use query strings to communicate with the server.

- Method. The "R" in URL stands for *resource*. In HTTP parlance, a web page is a resource; so is an image, a video, etc. There are many ways you can interact with a resource; the way is specified with a *HTTP method*. There are four methods. For the purpose of this project, you will need to use the following two:

  - **GET** : Request data from a specified resource.
  - **POST** : Submit data to be processed to a specified resource.

- **Body.** Sometimes, query strings alone are not enough to contain all the data you want to submit to the server. For instance, if you look at the query string above carefully, you see that it contains special characters like "=" and "& that have special meanings. What if the data you submit (e.g. a critter program) also contains these characters? There are ways to get around this limitation (for instance, try searching for "a&b=c" in Google and observe the query string), but it's a lot of hassle. And what if you want to upload a file to a server?

  Therefore, a HTTP request (particularly, a `POST` request) may also include a *body*, which is simply an arbitrary string. We will talk more about the body later.

- **Content type.** The content type specifies the encoding used by the body to represent the data contained in the body. Commonly used content types include `application/json` and `text/html`.

The response to an HTTP request also contains a content type and a body. It also includes a status code, which is a number representing a certain class of responses. For instance, 200 stands

for **OK**, meaning that the request has succeeded. Perhaps the most recognizable status code is 404: it means that the requested resource was **Not found**. In this project, any request that goes to a non-defined API should receive a 404 response.

3 JSON

JSON (JavaScript Object Notation) is a data-interchange format commonly used on the web. As an exercise, we ask you to go to json.org to learn the JSON format. It's one of the simplest data-interchange formats and should therefore serve as a good starting point for reading technical documents. In this assignment, the server and the client will mostly communicate using JSON.

You could certainly manipulate JSON objects simply as strings, but that'd be rather inconvenient. Instead, you will want to be able to convert Java objects to and from JSON objects directly. There are many libraries that do that and we ask you to pick one of them. These are two good choices:

`https://code.google.com/p/google-gson/`

`http://www.json.org/java/`

4 API Specification

In this assignment, we have defined a set of APIs (application programming interfaces) that specify how the server and the client must interact. They are described on this page: `http://docs.cs2112a7.apiary.io/`. You can inspect the APIs in detail by clicking them on the web page.

We expect that your server will be able to operate with the course staff client program, and your client will be able to operate with the course staff server. To make this possible, you will need to follow the APIs carefully. It is okay to extend the API by adding new parameters or fields, but both your client and server should be able to interoperate with a server or client respectively that does not provide this extra information and ignores it when provided by the other side.

5 Incremental updates

The `GET /world` API request is the most tricky to implement. When the `update_since` query string is specified, the server is supposed to return a *diff*: the difference between the current version of the world and the old version named by the timestemp parameter. This API allows the client to avoid downloading and rerendering the whole world for each timestep.

You may implement this feature in any way. You will get full credit as long as your implementation works. But how to implement this feature? One way is to keep track of every version of the world. This could be easily done by deep-cloning the World instance in each timestep and save the clone in an array. You then need an algorithm that takes two World instances and calculate their difference, i.e. the content of the grids that have changed between these two versions.

Saving all these worlds and comparing them would be inefficient in time and space. A better way is to keep a *log*, a data structure that records a sequence of operations or changes. In fact version-control tools like git are implemented with this idea. The first time you `commit` a file, its

entire content is saved by Git. Subsequently, when you `commit` changes to the file, the changes you made are recorded in a log. Git can use the log to reconstruct any past version of the file without having to store every version.

6 Thread Safety

Your server should support connections from multiple clients that are viewing the same world. If multiple client programs are interacting with single server, the server is likely to be handling multiple requests in parallel. If only one thread is handling all client requests, some clients may have to wait for a long time. Instead, a web server usually has a *thread pool* running in the background. Each new request is handed to a thread in the pool for processing; therefore, concurrent requests are handled by different threads. To make sure this works correctly, you will need to ensure that all shared resources are accessed in a thread-safe fashion. We will test your implementation with multiple simultaneous clients to see whether your implementation is thread-safe, so you should too.

One way to make the implementation thread-safe is to lock the entire world with a mutex. However, a *coarse-grained* lock like this may reduce concurrency to an unacceptable level. One way to improve concurrency is to use a reader–writer lock such as the one provided in the standard library: `java.util.concurrent.locks.ReentrantReadWriteLock`.

7 Project Setup

Your project will comprise both client-side code that will be a regular JavaFX application and server-side code that runs within a web server. As a starting point, we have provided an example code for a client and servlet that are configured to talk to each other. They can be found in the source release under the `demoClient` and `demoServlet` directories respectively.

Setting up a servlet project within Eclipse is a bit different from a regular Java application. You will want to have the Java Enterprise Edition plugins for Eclipse (j2ee Standard Tools, or JST) and to use the Java EE perspective. Also, you should use the Tomcat 7.0 Server as the servlet container that you run the servlet within.

To set up the servlet project:

- Go to **File** → **New** → **Other** and select **Dynamic Web Project** under **Web**. Click **Next**.
- Click **New Runtime**, and on the pop-up window, select **Apache Tomcat v7.0** under **Apache**. Then click **Finish**
- Click **Finish** to create the project. Now, go to the directory where this project is created, and replace everything under **src/** with the source code from the `demoServlet` directory. Now you should be able to refresh the project on Eclipse and see your code.
- At this point, you should be able to run the project. Open up `src/demoServlet/Servlet.java` and run it. Eclipse should launch the Tomcat server and open up a browser that shows a simple web page. You should also be able to visit `http://localhost:8080/demoServlet/`

from a regular web browser and see the page. In fact, if you find your computer's IP address and you don't have a firewall turned on, you should even be able to view this web page from another computer on the network by replacing `localhost` with the appropriate IP address.

You can also access the servlet using the `demoClient` application. If you run this application with three arguments, it will send a `POST` message that changes the message stored on the servlet.

Another tip: go to the **Project** menu and select **Build Automatically** to make sure the latest changes to your code are always compiled.

The class `Servlet` in the `demoServlet` package contains methods `doGet` and `doPost` that handle the respective HTTP requests and generate responses. You will need to write your own versions of these methods that dispatch to the appropriate code to handle the various API calls.

8 Ring buffers

As a separate implementation task, we are asking you to implement a thread-safe *ring buffer*. Each network request from a client spawns a new thread on the server. A large number of threads could overwhelm the server, so HTTP servers like Tomcat implement a *thread pool* containing a fixed number of threads that handle requests. Client requests are placed into a queue that threads in the thread pool draw their work from.

The interface `java.util.concurrent.BlockingQueue` describes a thread-safe queue abstraction that can be used to implement a thread pool. This interface allows *producer* threads to add new items to the queue and *consumer* threads to remove items. We are asking you to build a data structure that implements this interface. You do not need to use it in your project implementation.

You will implement your thread-safe queue as a *ring buffer*, a data structure commonly used in distributed systems with limited memory. A ring buffer is a fixed size queue implemented using an array and two integer indices (and some number of locks or condition variables). Items added to the queue are inserted to the array and the tail index is incremented. If there is insufficient space in the array (as determined by comparing the two indices), then adding to the queue fails. When items are popped from the queue, the item in the slot indicated by the head index is returned and the head is incremented.

If a producer tries to put an item into the queue but the array is full, the producer thread should block until there is space. Conversely, a consumer that tries to pop an item from an empty queue should block until a producer pushes a new item onto the queue.

Your thread-safe version of this data structure must implement all of the methods from the interface `java.util.concurrent.BlockingQueue` listed below. Any other methods may throw an `UnsupportedOperationException`. You must use an array to implement this and are not allowed to use any classes from the `java.util` package or subpackages. The provided class `student.util.RingBufferFactory` should be updated to use your implementation.

Required Methods

From `Collections`: `add`, `contains`, `equals`, `isEmpty`, `iterator`, `size`.

From `Queue`: all methods.

From `BlockingQueue`: all methods.

Tips: Testing concurrent code is very tricky because it never runs the same way twice. You will want to develop a test harness for the ring buffer that tests it with at least several threads issuing method calls concurrently. And you want to test both concurrent producers and concurrent consumers.

One nice trick for catching concurrency bugs is to stick random delays throughout your code, conditioned on a static, final boolean variable so they can be turned off when you are not debugging. Random delays will cause your program to explore a wider variety of schedules of different threads. Generous use of assertions is also highly recommended as a way to catch race conditions and other bugs.

It will be tempting to give the job of writing the ring buffer to one member of the project group. However, writing this kind of concurrent code is very tricky. You are likely to get it wrong unless you and your partner have both gone over the code carefully together. We recommend you work together closely on this part.

9 Your Task

To summarize, your tasks include:

- Learn JSON and choose a JSON parser library.
- Implement a thread-safe web server that runs the simulation and responds to HTTP requests according to the given API.
- Update your GUI to be a HTTP client that talks to your server.
- Implement a thread-safe ring buffer.

All application and GUI features that were described in A6 should be supported in A7 as well. And remember that your server needs to be able to handle multiple clients: you should be able to launch multiple instances of your GUI, and they should all show the same view of the world.

You are free to use any libraries compatible with Java 7, including any third-party libraries.

10 Karma

If you want to go beyond what is required in this assignment, there are many possibilities. You could add user management and authentication, so many users can be given more limited access to a world with a smaller number of administrative users. You could make the client application into a Java applet that runs inside the web browser. You could make the world able to grow indefinitely as critters explore, so that a large number of users could be supported without colliding with each other. And you probably have many good ideas of your own.