Java Concurrency Lab Manual

Lab #1 – The Big Picture

Setup

Pre-Class:

Prior to the start of class the following steps have been completed:

1) Install a Java 8 JDK

2) Set the JAVA_HOME environment variable to the JDK installation directory

3) Install Eclipse 2020-06 IDE for Enterprise Java and Web Developers

4) Install the VMLens plugin for Eclipse, using the Eclipse Marketplace.

5) Create a Workspace folder to hold the source code

6) Set the Workspace folder as the Eclipse Workspace

7 Configure Eclipse to use the Java 8 JDK as the default Installed JRE

(Window → Preferences → Java → Installed JREs)

Pre-Lab import the starter code:

1) Open Eclipse using the shortcut on the Lab Machine Desktop

1) Download the Starter.zip file from the class website, by opening a Browser on the Lab Machine, navigating to http://www.jre-training.com/Concurrency. Right click the Starter.zip file and select Save Target as/Save Link as to download it to the Lab Machine, do not unzip it.

2) Import Starter.zip into Eclipse

(File → Import → General → Existing Projects into Workspace → Select archive file → Browse to the zip file you downloaded)

3) Build the maven-built-master project

(Right click the project in an Eclipse Explorer → Run As… → Maven install)

4) Build the custom-javadoc project

(Right click the project in an Eclipse Explorer → Run As… → Maven install)

5) Examine the source code for the SensorValueGenerator class in the sensor project, note the custom JavaDoc annotation of @Stateless. Open the Stateless.java source code in the custom-javadoc project and read the JavaDocs for this annotation. Open the index.html file in the doc folder with a Web Browser (right click → Open With → Web Browser) read the JavaDocs for all the custom JavaDoc

annotations in the custom-javadoc project. Note where each annotation is used class level/method level/ etc.

Lab Steps:

1) Start with reading the Overview below followed by the Thread Model and then the project descriptions

2) Look at the pom.xml in the maven-build-master project, pay attention to the <reporting> section to see the Maven plug-ins for the Code Review

Overview

We will be creating a simulated IOT sensor environment. A Sensor will be generating a value approximately every 3 seconds and sending that value to a SensorController using a SensorData record object. The SensorController will be managing a number of Sensors and accumulating the SensorData records into a SensorSummary record object and sending that object to the MasterSensorManager. The MasterSensorManager will be working with a number of SensorController objects and writing the contents of SensorSummary record objects to a file. Your will be designing, coding, documenting and testing Threads in the Sensor, SensorController and MasterSensorManager implementations. You will be working with Immutable, Single-Threaded and Multi-Threaded classes using SensorData, SensorSummary and SensorMaster objects.

Thread Model

The IOT Sensor simulation starts with a Java application with a main method constructing an instance of the MasterSensorManager. The following is the Thread Model for this simulation:

Thread Summary

Thread Name/Purpose	Class creating the Thread	The runnable class
main The Entry point to the simulation		MasterSensorManager implementation
sensorController<int> The Thread to control the SensorController there will be one of these for each SensorController implementation	MasterSensorManager implementation	SensorController implementation
singleSensorController<int-letter> The Thread to control the Sensor there will one of these for each SingleSensorController implementation	SensorController implementation	SingleSensorController implementation

maven-build-master

The maven-build-master project is the parent of all the code projects in our environment. This project is deployed as a pom file and has no Java source code. It does have the Code Review components and hosts the configuration files for the Code Review process. For details on how to run and interpret the Code Review results see the Code Review section of this Lab Manual.

custom-javadoc

On of the critical steps in writing multi-threaded Java code is documenting it. The custom-javadoc project contains a number of custom JavaDoc annotations that are designed to make documenting you code easier.

These annotations include:

Annotation	JavaDoc location
@Immutable	Class level
@MultiThreaded	Class level
@SingleThreaded	Class level
@Stateless	Class level
@ThreadSafe	Method level

To use these custom JavaDoc annotations in your code, you will need to build the customer-javadoc project jar file (Run As → Maven install) , and put that jar file on your classpath. The maven-build-master parent pom, has this jar file listed for you (see the tagletArtifacts entry in the javadoc plugin). Please note to build the cutom-javadoc project the tools.jar file must be on the classpath, and that jar is in the JDK not the JRE, see the pom.xml in the custom-javadoc project for details.

sensor-data

This is the Maven project that will contain the Entity classes for the IOT sensors.

sensor

The sensor project contains the Sensor interface, and a sequential implementation. You will be coding/documenting/testing a concurrent implementation of the Sensor interface. This project also hosts the SensorReporter interface that a Sensor uses to push the SensorData to the SensorContoller.

sensor-controller

The sensor-contoller project hosts the SensorContoller interface along with the SingleSensorContoller interface. An implementation of the SensorContoller interface is created my the MasterSensorManager implementation and controller a number of SingleSensorContoller instances. Each SingleSensorContoller instance will in turn control a specific Sensor instance.

master-sensor-manager

The MasterSensorManager interface controls the entire Sensor simulation. The MasterSensorManager is started by the main thread and create the output file, then creates the SensorContoller instances. The MasterSensorManager implementation will run for 3 minutes, then shut everything down.