FINAL PROGRAMMING GRADING

INFORMATION

To run the tests, you will need to run the FactoryAuthorizer.jar. The port by default is 6687, but you may need to change this. Enter the port that the student uses in their factory code. This should probably be in FactoryWorker.java.

Replace the Decoder.jar from the workspace, manually add the one provided. This will ensure that the student did not make their own in an attempt to cheat since it is possible to modify.

TESTS

AUTHORIZATION

Steps

1. Run FactoryAuthorizer.jar
2. Run FactoryServer, and load in a Factory.
3. Run FactoryClient, and connect to the FactoryServer.
4. View the output of the FactoryAuthorizer.

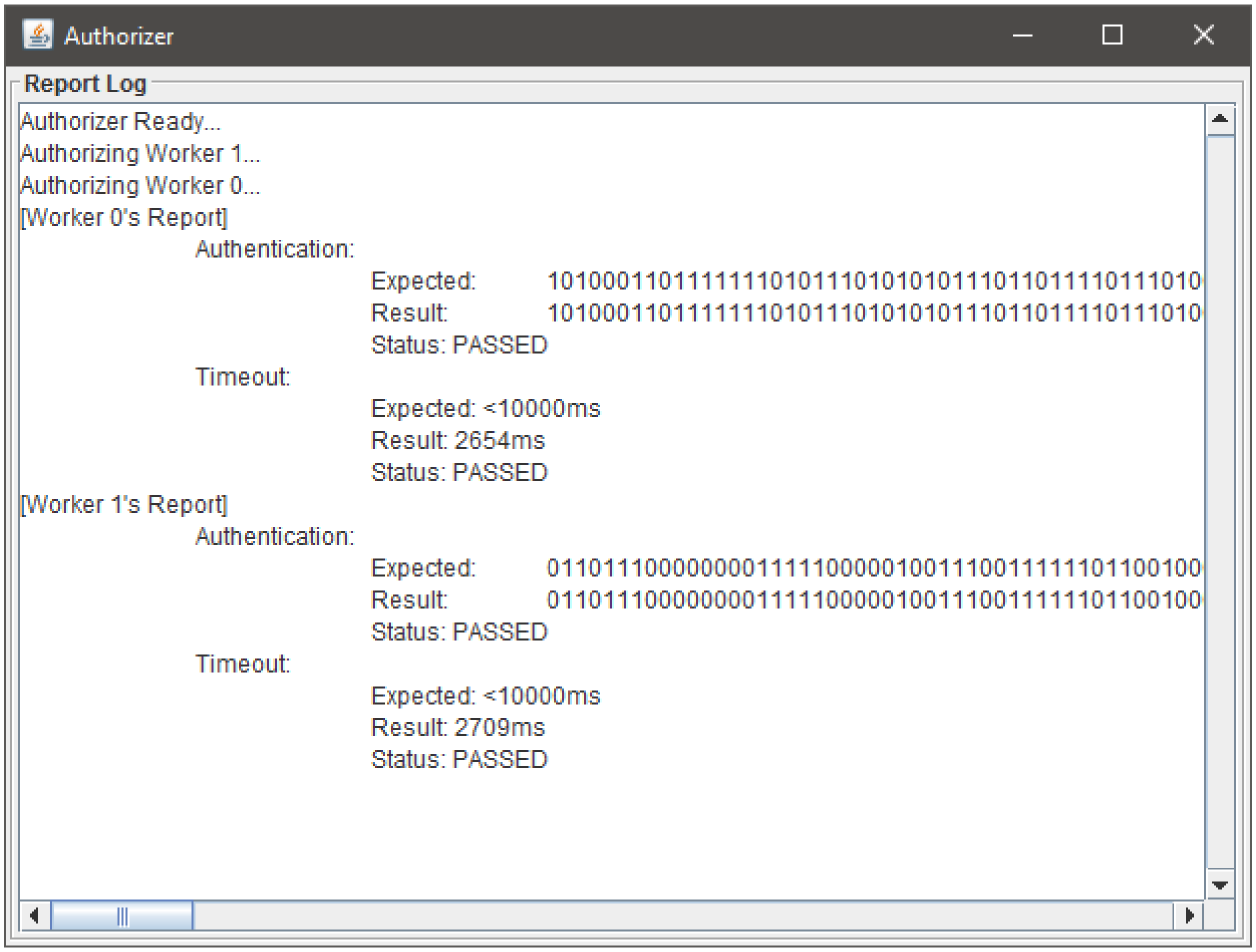
Notes

Inspect the code to ensure the student used *Decoder.resolve(String)*.

If it is not used, give a zero for passing timeout and authentication.

```java
Socket mSocket = new Socket("127.0.0.1", 6687);

BufferedReader mReader = new BufferedReader(new InputStreamReader(mSocket.getInputStream()));
BufferedWriter mWriter = new BufferedWriter(new OutputStreamWriter(mSocket.getOutputStream()));

//write the name to the authorizer
mWriter.write(mLabel);
mWriter.newLine();
mWriter.flush();

String[] toParse = new String[20];
String[] answers = new String[20];
String answer = "";

//read the 20 messages from the authorizer
for(int i = 0; i < 20; ++i) toParse[i] = mReader.readLine();
//decode the messages concurrently
Thread[] threads = new Thread[20];
for(int i = 0; i < 20; ++i) {
        int num = i;
        threads[i] = new Thread(() ->{
                answers[num] = Decoder.resolve(toParse[num]);
        });
        threads[i].start();
}
//wait for all to be solved, then combine them
for(int i = 0; i < 20; ++i) threads[i].join();
for(int i = 0; i < 20; ++i) answer += answers[i];

//send the authorizer the final message
mWriter.write(answer);
mWriter.newLine();
mWriter.flush();
mSocket.close();
```

Inspect the Authorizer, you should see “Authorizing Worker n…”, followed by the reports.



```
Authorizer Ready...
Authorizing Worker 1...
Authorizing Worker 0...
[Worker 0's Report]
		Authentication:
				Expected:	1010001101111111010111010101011101101110111010
				Result:		1010001101111111010111010101011101101110111010
				Status: PASSED
		Timeout:
				Expected: <10000ms
				Result: 2654ms
				Status: PASSED
[Worker 1's Report]
		Authentication:
				Expected:	0110111000000001111100000100111001111101100100
				Result:		0110111000000001111100000100111001111101100100
				Status: PASSED
		Timeout:
				Expected: <10000ms
				Result: 2709ms
				Status: PASSED
```

Points to deduct

- Workers do not authorize with their name -1.0%
- Workers did not pass the authentication -2.5%
- Workers did not pass the timeout -2.5%

TOOLBOX LOCATION AND BEHAVIOR

Steps

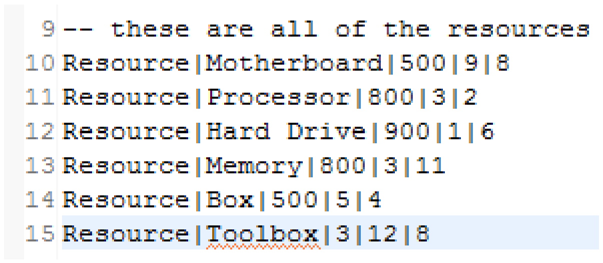
1. Run FactoryAuthorizer.jar
2. Run FactoryServer, and load in a Factory.
3. Run FactoryClient, and connect to the FactoryServer.
4. Edit the coordinates of the Toolbox resource in the factory that is loaded.
5. Reload the factory.
6. Watch the factory run to completion.

Notes

You should see the location of the toolbox change each time the factory is loaded on the server.

The location is the last two digits.

Edit these, but be sure they are valid, don't place it on a location that is already used.



```
-- these are all of the resources
Resource|Motherboard|500|9|8
Resource|Processor|800|3|2
Resource|Hard Drive|900|1|6
Resource|Memory|800|3|11
Resource|Box|500|5|4
Resource|Toolbox|3|12|8
```

Watch the solution video to get an idea of how the workers will behave with the toolbox.

Make sure that there are less toolboxes than there are workers in the factory configuration.

Make sure that the workers wait there when the number of toolboxes is 0.

Make sure that the workers are able to pick up the dropped off toolboxes.

Points to deduct

- The toolbox location is not dictated by the factory server. -1.0%
- Workers do not wait for an available toolbox when they are all in use -0.5%
- Workers do not use toolboxes once available again -0.5%

TOOLBOX CODE INSPECTION

Steps

1. Find and inspect the code that deals with the toolboxes.

Notes

This is a sample solution for the Factory Toolbox:

```java
public class FactoryToolbox extends FactoryResource {

        private Semaphore mNumberToTake;

        FactoryToolbox(Resource inResource) {
                super(inResource);
                mNumberToTake = new Semaphore(inResource.getQuantity());
        }

        public void takeOne() throws InterruptedException {
                mNumberToTake.acquire();
                super.takeResource(1);
        }

        public void leaveOne() {
                super.takeResource(-1);
                mNumberToTake.release();
        }
}
```

Note the use of a semaphore, this is what the student should have used.

```java
public class FactoryToolbox extends FactoryResource {

        private Lock mLock;
        private Condition mHasToolboxesCondition;

        FactoryToolbox(Resource inResource) {
                super(inResource);
                mLock = new ReentrantLock();
                mHasToolboxesCondition = mLock.newCondition();
        }

        public void takeOne() throws InterruptedException {
                mLock.lock();
                while(mResource.getQuantity() == 0) {
                        mHasToolboxesCondition.await();
                }
                super.takeResource(1);
                mLock.unlock();
        }

        public void leaveOne() {
                mLock.lock();
                super.takeResource(-1);
                mHasToolboxesCondition.signalAll();
                mLock.unlock();
        }
}
```

If the student used a lock, and waited on a condition, this is also acceptable.

The following is an example of an unacceptable solution:

```java
public class FactoryToolbox extends FactoryResource {

        FactoryToolbox(Resource inResource) {
                super(inResource);
        }

        public void takeOne() throws InterruptedException {
                while(mResource.getQuantity() == 0) Thread.sleep(10);
                super.takeResource(1);
        }

        public void leaveOne() {
                super.takeResource(-1);
        }
}
```

If the student even declared a Lock, Condition, or Semaphore, they will get credit for evidence of thread-safe code. This at least shows they were headed in the right direction.

Points to deduct

- No evidence of thread-safe code -1.0%
- Taking resources was not done with thread safe practices -0.75%
- Putting back resources was not done with thread safe practices -0.25%