CSCI 310 Project
Group Assignment
Due: November 30, 2020
Points: 75

The purpose of this project is to increase your knowledge of the Bankers Algorithm, threads and synchronization by solving a problem using Java.

Project Requirements

You are to implement the project described on pages 344 – 346 of the textbook subject to the additional requirements outlined below:

1. *This assignment is a group assignment*. You are expected to work in a group with one or two other students.

2. It must be possible to specify the values of *m* (number of resource types) and *n* (number of threads) via command line arguments. The program must work correctly for values of m and n in the range 1 to 10.

3. You are not expected to implement the Bank interface specified in the textbook; rather, you can implement a different definition of the Bank class that meets these additional requirements.

4. You are not expected to use the TestHarness.java file mentioned in the textbook. Instead, you must randomly generate test scenarios by using random numbers, such as those available via the Math.random method. You should randomly generate the maximum number of resources of each type that are available as well as the maximum demand by each of the customers for each resource type; the latter demand should be generated to be less than or equal to the maximum number of resources.

5. In addition to requesting resources, it must be possible to release (or return) resources.

6. Each customer thread must go through at least three cycles of requesting resources and holding them for a random period of time (1 – 5 seconds). The second and third requests must be for resources in addition to those previously held. In satisfying this requirement, you are expected to use the Math.random and Thread.sleep methods. The requests for a number of instances of each resource type should be randomly generated so that they are less than or equal to the maximum need by the customer.

7. You must use the following Java keywords to help synchronize your program: *synchronized*, *wait*, *notify* (or *notifyAll* in place of *notify*).

8. While the program is executing, display messages that correspond to important events. These messages are useful for debugging and should help to demonstrate that the requests are processed correctly.

9. Run your program for at least two different sets of initial resources, number of customers and requests. This must include one execution with four resource types and five customer threads. The output must demonstrate that multiple customers can hold resources at the same time in a safe manner, and that additional requests that would not be safe must wait.

10. Your program output must demonstrate that when the program ends all instances of all resources types have been returned by the customer threads.

11. Your program must be compatible with the version of Java that is installed on the Virtual Machine. The version is: `1.8.0_252`. This version works well for the Java that has been discussed in this class.

Sample output:

The following output demonstrates some of the items / events that you might need to display to understand if your program is functioning correctly. You may need to display additional information.

```
C:\RUNBANK>java RunBank 4 5
Bank - Initial Resources Available:
[ 6, 7, 6, 3 ]
Bank - Max:
[ 4, 0, 0, 3 ]
[ 5, 2, 4, 0 ]
[ 2, 5, 5, 1 ]
[ 6, 2, 5, 2 ]
[ 5, 5, 4, 3 ]
Customer 0 making request:
[ 1, 0, 0, 0 ]
Bank - Safe Sequence:
[ 0, 1, 2, 3, 4 ]
Bank - Allocation matrix:
[ 1, 0, 0, 0 ]
[ 0, 0, 0, 0 ]
[ 0, 0, 0, 0 ]
[ 0, 0, 0, 0 ]
[ 0, 0, 0, 0 ]
Customer 0 request 0 granted
Customer 1 making request:
[ 0, 1, 2, 0 ]
Bank - Safe Sequence:
[ 0, 1, 2, 3, 4 ]
```

```
Bank - Allocation matrix:
[ 1, 0, 0, 0 ]
[ 0, 1, 2, 0 ]
[ 0, 0, 0, 0 ]
[ 0, 0, 0, 0 ]
[ 0, 0, 0, 0 ]
Customer 2 making request:
[ 1, 2, 0, 0 ]
Bank - Safe Sequence:
[ 0, 1, 2, 3, 4 ]
Bank - Allocation matrix:
[ 1, 0, 0, 0 ]
[ 0, 1, 2, 0 ]
[ 1, 2, 0, 0 ]
[ 0, 0, 0, 0 ]
[ 0, 0, 0, 0 ]
Customer 2 request 0 granted
Customer 3 making request:
[ 3, 0, 3, 1 ]
Bank - Safe Sequence not found
Bank:  Customer 3 must wait
Customer 4 making request:
[ 0, 3, 3, 0 ]
Bank - Safe Sequence not found
Bank:  Customer 4 must wait
Customer 1 request 0 granted
Customer 2 making request:
[ 0, 0, 4, 0 ]
Bank - Safe Sequence not found
Bank:  Customer 2 must wait
Customer 1 making request:
[ 2, 0, 0, 0 ]
Bank - Safe Sequence not found
Bank:  Customer 1 must wait
Customer 0 making request:
[ 1, 0, 0, 2 ]
Bank - Safe Sequence:
[ 0, 1, 2, 3, 4 ]
Bank - Allocation matrix:
[ 2, 0, 0, 2 ]
[ 0, 1, 2, 0 ]
[ 1, 2, 0, 0 ]
[ 0, 0, 0, 0 ]
[ 0, 0, 0, 0 ]
Customer 0 request 1 granted
Customer 0 releasing resources:
[ 2, 0, 0, 2 ]
<Requests of waiting customers may be retried>
.
.
.
```

```
Final Available vector:
[ 6, 7, 6, 3 ]
Final Allocation matrix:
[ 0, 0, 0, 0 ]
[ 0, 0, 0, 0 ]
[ 0, 0, 0, 0 ]
[ 0, 0, 0, 0 ]
[ 0, 0, 0, 0 ]
```

One report in a single PDF document is to be turned in by each team. Turn in the report to the D2L Assignment folder for the Project. It is to include the following materials:

- your starID; and the names of all team members.
- output that demonstrates that your program meets the requirements. This output must be **annotated** to explain the changes that it shows.
- a discussion of:
  - key aspects of the program design; such as use of Java synchronization.
  - the output -- demonstrate that the program is functioning correctly for each test set.
- a listing (source code) of the program.
- a user manual that explains how to run the program.
- A copy of the program (.class files, and .java files).

Also, leave a copy of the program (.class and .java files) in your Coursefile work folder for this class in a folder called Project.

Grading

Grading of this project will be divided into two parts:

a. A group score (g) consisting of up to 80% of the available points (60 points).
b. An individual (i) score.

These two parts are added together to determine your total score for this project.

The group score is based on the following items (with approximate percentage weight):

a. Readability (10%): readability of code (is code documented, are indentation conventions followed, use of self-documenting variable names, etc.)
b. Correctness (70%): Does your program meet the requirements; this includes correct execution. Does your written report adequately describe your project; this includes the discussion of your design and the discussion of your test case output.
c. Testing (20%): The completeness of your testing will be examined.

Each group member is to assign points to the other group members, but not to him/herself. Suppose there are n group members and n*(0.20 * 75) points to be assigned (pool_points). Then each person would assign ((n-1)/n) * pool_points to the other members of the group. This assignment is to be based on the contribution of each group

member; in making this decision, consider both the effort expended by each group member and their effectiveness. A signed form with this point distribution is to be given to the instructor by each group member when the project is turned in. The instructor will compute an average potential score, p_k for each group member based on the input received. The actual individual score will then be determined by: $i = p_k * (g / 60)$.

Name: _____

Group : _____

Project: _____

a. Number of people in group: _____ (n)

b. Available Points to Assign: _____ (n – 1) * 0.20 * 75

For each person in the group, except yourself, assign points to that person based on their effort and effectiveness in the project. Note that the sum of the points must not exceed the value computed on line-b above.

Name	Assigned Points	Justification

Signature: _____