

Project 6—Implementation of Banker’s Algorithm in Dealing with Deadlocks
(Due date: 12/11/2008/Thursday)

Your name:	Date:
------------	-------

In Chapter 7 (Deadlocks), we described the Banker’s algorithm as one of the methods used for deadlock avoidance. In this project, we will write a Java program that implements the banker’s algorithm: customers request and release resources from the bank and the banker will grant a request only if it leaves the system in a safe state. A request is denied if it leaves the system in an unsafe state.

- The bank will employ the strategy outlined in the textbook whereby it will consider requests from n customers for m resources. The bank will keep track of the resources using the following data structures:
 - `int [] available;` //the available amount of each resource
 - `int [] [] maximum;` //the maximum demand of each customer
 - `int [][] allocation;` //the amount currently allocated to each customer
 - `int [][] need;` //the remaining needs of each customer
- The functionality of the bank appears in the interface defined in “Bank.java” file.
 - The implementation of this interface (in a Java file called “BankImpl.java”, which you are required to write) will require adding a constructor that is passed the number of resources initially available. For example,
 - Suppose we have three resource types with 10, 5, and 7 resources initially available. In this case, we can create an implementation of the interface using the following technique:
 - `Bank theBank = new BankImpl(10, 5, 7);`
 - The bank will grant a request if the request satisfies the safety algorithm outlined in the textbook; if granting the request does not leave the system in a safe state, the request is denied.
- Testing your implementation:
 - There is a test input data file called “infile.txt” that contains the maximum demand for each customer. The file appears as follows:


```
7,5,3
3,2,2
9,0,2
2,2,2
4,3,3
```

 - This means the maximum demand for customer0 is 7, 5, 3; for customer1, 3, 2, 2; and so forth.
 - Since each line of the input file represents a separate customer, the `addCustomer()` method is to be invoked as each line is read in, initializing the value of maximum for each customer:
 - `maximum[0][]` is initialized to 7, 5, 3,
 - `maximum[1][]` is initialized to 3, 2, 2,
 -
 - Furthermore, **Test.java** also requires the initial number of resources available in the bank. For example, if there are initially 10, 5, and 7 resources available, we invoke `Test.java` as follows:
 - **`java Test infile.txt 10 5 7`**

- The following shows the process on how to run this process, provide inputs and read the outputs from the program:

```
D:\2008Fall\CSC280_0S\assignments\project6_Sol\test>java Test infile.txt 10 5 7
*
Available = [10 5 7]

Allocation = [0 0 0][0 0 0][0 0 0][0 0 0][0 0 0]
Max = [7 5 3][7 5 3][3 2 2][3 2 2][9 0 2]
Need = [7 5 3][7 5 3][3 2 2][3 2 2][9 0 2]
RQ 2 2 1 0
*2*
*1*
*0*

Customer # 2 requesting 2 1 0 Available = 10 5 7 Approved
*
Available = [8 4 7]

Allocation = [0 0 0][0 0 0][2 1 0][0 0 0][0 0 0]
Max = [7 5 3][7 5 3][3 2 2][3 2 2][9 0 2]
Need = [7 5 3][7 5 3][1 1 2][3 2 2][9 0 2]
RQ 1 3 2 3
*3*
*2*
*3*

Customer # 1 requesting 3 2 3 Available = 8 4 7 Approved
*
Available = [5 2 4]

Allocation = [0 0 0][3 2 3][2 1 0][0 0 0][0 0 0]
Max = [7 5 3][7 5 3][3 2 2][3 2 2][9 0 2]
Need = [7 5 3][4 3 0][1 1 2][3 2 2][9 0 2]
RQ 0 2 3 1
*2*
*3*
*1*

Customer # 0 requesting 2 3 1 Available = 5 2 4 INSUFFICIENT RESOURCES
Denied
*
Available = [5 2 4]

Allocation = [0 0 0][3 2 3][2 1 0][0 0 0][0 0 0]
Max = [7 5 3][7 5 3][3 2 2][3 2 2][9 0 2]
Need = [7 5 3][4 3 0][1 1 2][3 2 2][9 0 2]
RL 2 2 1 0
*2*
*1*
*0*

Customer # 2 releasing 2 1 0 Available = 7 3 4 Allocated = [0 0 0 ]*
Available = [7 3 4]

Allocation = [0 0 0][3 2 3][0 0 0][0 0 0][0 0 0]
Max = [7 5 3][7 5 3][3 2 2][3 2 2][9 0 2]
Need = [7 5 3][4 3 0][3 2 2][3 2 2][9 0 2]
```

- After you typed “java Test infile.txt 10 5 7” with an Enter key, you may:
 - Type “*” key for the program to show the current state information;
 - Provide Request with “RQ customer# r1 r2 r3”
 - For example: “RQ 0 2 1 3” means “customer0 requests resources 2 1 3”.
 - Release resources with “RL customer# r1 r2 r3”
 - For example, “RL 2 3 0 2” means “customer2 releases resources 3 0 2”
 - At anytime, you can type “*” to display the current state of the system.

What you need to do in this project:

1. Download the accompanied zipped file and expand it and you will find the following files:
 - a. “Bank.java”—the interface that you will implement upon
 - b. “Customer.java”—the class that defines the customer info.
 - c. “infile.txt”—input data file that gives the maximum demand for each customer
 - d. “Test.java”—used for testing your implementation
 - e. “BankImpl.java”—a java program that you will need to complete. This program is supposed to implement the interface “Bank.java”. At this time, only a skeleton is provided and it is your job to complete and test it.
2. Complete the “BankImpl.java” based on the “Bank.java” interface and the data structures mentioned above (*available*, *maximum*, *allocation*, and *need* arrays at the beginning).
3. You will definitely use “Test.java” to test your implementation:
 - a. Read this program thoroughly. You will get some ideas on how to implement “BankImpl.java.”
 - b. Don’t change to code.
4. Test you program by using:
java Test infile.txt 10 5 7
5. Provide some requests and releases to the program and record the results by taking a screenshot (use the sample output shown in the previous page).

Important:

- You may use or NOT use “Customer.java” and “SleepUtilities.java” for this project.
- You need to use “Bank.java” and “Test.java” and do NOT change them.
- In addition to above compressed source java files, a **readme** file (PDF or Doc or DOCX format) is required for your submission. Check the following on how to submit your project.
 - In the sample code, the “thread” name, in fact, means a “customer”. To use “thread”, this sample code can be readily modified to a multithreaded program which is NOT required in this project. Synchronized method is NOT required.
- At least one screenshot are required in your Readme file.

=====How To Submit—Read Carefully, Pease!=====

1. Create a directory “**project6_YourLastName**” (you must use this format for the directory name for this project; **Use Your Last Name. For example, if your** last name is Smith, you should create directory with the name of “project3_Smith”
 2. Create “**project6src**” subdirectories under “project6_YourLastName” directory.
 - a. Under these subdirectory, you can put **ONLY** java files (source) files. This should be clean and comprehensive—that is, I will javac *.java and I can test your code.
 3. If you have used some IDE, you can compress the package files in other subdirectories than the above six ones and tell me how to run in the **readme** file.
 4. A “**readme**” file is required for the project write-up that tells how to compile in which IDE (not required if not having used any IDE but a simple command line), result screenshots , ... keep this readme simple!
 - a. This “readme” must reside in the “**project6_YourLastName**” dir in the format of .txt, .pdf, or .doc/docx.
 5. Compress the “**project6_YourLastName**” dir and its contents into a zipped/rar-ed file with same name.
 6. Submit the compressed file to the instructor by email.
 7. Double check your work before submission. Significant penalty (10—100 points)will be applied if your submission does not follow the above instruction!
-