# 44-542 Object-Oriented Programming Your Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Practice Lab Exam 2 - 5 pts

**The Goal**

You will implement **two classes** to track of the operations of a bank. First, implement the **Customer** class to store information about a customer waiting at the bank. Second, implement the **Bank** class to track which customers are waiting and store the account balances for each customer.

For each Java file that you create, you must include your first and last name as a comment at the beginning of the program file, using the **@author** notation.

We are providing driver classes for you that will handle input and output. You may use println() in your two classes to help you debug, but try looking at the test output from the driver class first. You must comment out any print methods in your code, before your submit your work. If not, we will deduct points.

We also strongly recommend that when you create the Bank class you create stubs for all of the methods and then implement and test them one at a time. Creating stubs allows your project to compile.

Remember: Save early, Save often!

**The Process**

1. Download and unzip LabExam2Practice from the course website. Open the project in NetBeans. Right click on the project and rename it to LastNameLabExam2Practice where LastName is *your* last name. (For example, Ajay Bandi would rename it to BandiLabExam2Practice.)

2. Create the Customer class in the **labexam2practice** package. Complete the class using the specifications given below.

**Customer Specification**

* The **Customer** class will have three private attributes:
  + **name** – a String that holds the name of the customer.
  + **transactionValue** – an int that gives the value of the transaction the customer wants to make. (Don’t worry about the monetary unit. For this application, they will be arbitrary.)
  + **wait** – an int that is the amount of time that the customer has been waiting in line. (The time units are also arbitrary.)
* The **Customer** class will have a two argument constructor with parameters for **name** and **transactionValue**. Set **wait** to zero.
* The **Customer** class will have a getter method for each of the three attributes.
* The **Customer** class will have a toString method. Use **Insert Code…** in NetBeans to generate this method for you.
* The **Customer** class will have a method that updates wait
  + **public void bumpWaitTime()** – increment wait by one.

3. **Test** the Customer class by running the **TestCustomer** class provided.

*You are implementing classes that should work with any interface/client that follows the specifications. TestCustomer reads lines from the script file Customer.txt. Each line invokes a method of the Customer class and gives expected results. Do not change TestCustomer. If you see syntax errors in TestCustomer, most likely your class does not yet meet the specifications given for the Customer class.)*

*INPUT:*

**Customer.txt**

Customer:Constructor John 10

Customer:getName Name should be John

Customer:getTransactionValue Value should be 10

Customer:getWait Wait time should be 0

Customer:bumpWaitTime Wait time should now be 1

Customer:bumpWaitTime Wait time should now be 2

Customer:toString The customer should hold the values John, 10 and 2

*When TestCustomer runs, it prints what it is testing, the expected result and the actual result from your class. We have saved that output into the file Customer\_output.txt, which you can see below or look at in the files tab of the project.*

*OUTPUT:*

Customer\_output.txt

Testing Customer:Constructor John 10

EXPECTED:

RESULT: Created customer Customer{name=John, transactionValue=10, wait=0}

Testing Customer:getName

EXPECTED: Name should be John

RESULT: Customer name is John

Testing Customer:getTransactionValue

EXPECTED: Value should be 10

RESULT: Customer transaction value is 10

Testing Customer:getWait

EXPECTED: Wait time should be 0

RESULT: Customer wait time is 0

Testing Customer:bumpWaitTime

EXPECTED: Wait time should now be 1

RESULT: Customer wait time is 1

Testing Customer:bumpWaitTime

EXPECTED: Wait time should now be 2

RESULT: Customer wait time is 2

Testing Customer:toString

EXPECTED: The customer should hold the values John, 10 and 2

RESULT: Customer toString is Customer{name=John, transactionValue=10,wait=2}

4. Create the Bank class in the labexam2practice package. We strongly recommend that you create a method stub (just an empty method with the correct signature that compiles) for each of the methods in the Bank Specification below.

5. Complete the methods one at a time and test as you go by running the **TestBank** class provided.

**Bank Specification**

* The Bank class will have four private attributes:
  + **waiters** – a **Queue<Customer>** that holds all the customers waiting to be served.
  + **accounts** – a **Map<String,Integer>** from customer names to the amount in their account.
  + **served** – an int that records the number of customers that have been served.
  + **totalWaitTime** – an int that records the total wait time for all customers that have been served.
* The Bank class will have a no-arg constructor that initializes each of the attributes.
  + Set **waiters** to an empty **LinkedList<Customer>**.
  + Set **accounts** to an empty **HashMap<String,Integer>**.
  + Set **served** to zero.
  + Set **totalWaitTime** to zero.
* The Bank class will have a toString method. Use the following definition. (It is the method that NetBeans generates with \n\t added in two places to make it more readable during testing.)

**public String toString() {**

**return "Bank{" + "waiters=" + waiters +**

**", \n\taccounts=" + accounts +**

**", \n\tserved=" + served +**

**", totalWaitTime=" + totalWaitTime + '}';**

**}**

* The Bank class will have the following methods:
  + **public double averageWait()** – Return the **totalWaitTime** divided by **served**.
  + **public String addToLine(String name, int transactionValue)** – This method will create a new customer and add them to the back of the **waiters** queue. Return the string "Customer added to line: " concatenated with the customer.
  + **public String serve()** – If the waiters queue is empty, return the string "No one waiting". Otherwise, remove the customer from the front of **waiters**. Add the wait time for that customer to the **totalWaitTime**. Increment **served** by one. Call the private method **recordTransaction** with that customer. Finally, return the string "Served customer: " concatenated with the customer.
  + **public String bumpTime()** – Call **bumpWaitTime()** on each customer in the **waiters** queue. Return the string "All wait times increased".
  + **public int numberWaiting()** – Return the number of items in the **waiters** queue.
  + **private void recordTransaction(Customer c)** – This private method updates the **accounts** map for a customer using the name as the key. If this customer is already in the map, add the customer’s **transactionValue** to the existing value in the map. If there is no existing account, create a new mapping from the name to the **transactionValue**.
  + **public List<String> customersOver(int low)** – Return a list of the names of the customers in the **accounts** map whose map value is greater than **low**.

*TestBank will ask you for a name of a file. In the output window, type* ***SmallBank.txt*** *and hit Enter to run the test client. (You can see SmallBank.txt at the end of this document.* ***We have not included SmallBank\_output.txt, but you can view it under the Files tab.****) SmallBank tests each of the methods at least once and gives the expected results for each invocation. Work on your Bank class until your output matches the expected SmallBank results.*

*When SmallBank.txt works, run TestBank and type* ***LargeBank.txt.*** *LargeBank does many more method invocations and only gives expected results after some of the steps (marked by \*\*\*\* so you can find them easily). It does give the expected state at the very end. You can also check your output against LargeBank\_output.txt.*

6. Submit your solution by following the steps below:

Save your files in NetBeans.

Zip your entire project. (It should be called LastNameLabExam2Practice where LastName is *your* last name.)

Submit the zip file to the LabExam2Practice dropbox.

Download the zip file you submitted.

Look in the zip file and verify that Customer.java and Bank.java are correct. If not, resave your project in NetBeans and resubmit.

**More about the Test Programs (We will do something similar for the regular lab exam.)**

The first token in each line specifies the method to invoke. Any arguments follow. The remainder is a comment where the expected behavior can be specified. When TestCustomer is run for the model solution, the result is:

Note that you do not need to write a driver class or any main methods. While it might be tempting to change the code in the two driver classes TestCustomer and TestBank, don’t do it. Your code will be compiled and executed with our driver classes. Any changes you make to the driver classes will NOT count for your solution. If your code does not work with our drivers, we will deduct marks.

SmallBank.txt

Bank:Constructor

Bank:addToLine Laverne 15 should add Laverne to the back of the line with value 15

Bank:addToLine Shirley 25 should add Shirley to the back of the line with value 25

Bank:addToLine Chachi -10 should add Chachi to the back of the line with value -10

Bank:serve should remove Laverne from the line and create a map from Laverne to 15;

numberServed is now 1

Bank:serve should remove Shirley from the line and create a map from Shirley to 25;

numberServed is now 2

Bank:addToLine Joanie 17 should add Joanie to the back of the line with value 17

Bank:bumpTime should make the wait time for Chachi and Joanie increase to 1

Bank:serve should remove Chachi from the line and create a map from Chachi to -10; numberServed is now 3 and total wait is 1

Bank:averageWait should be 0.33333333

Bank:bumpTime should make the wait time for Joanie increase to 2

Bank:addToLine Chachi 9 should add Chachi to the back of the line with value 9;

Bank:bumpTime should make the wait times for Joanie and Chachi increase to 3 and 1

Bank:serve should remove Joanie from the line and create a map from Joanie to 17; \ numberServed is now 4 and total wait is 4

Bank:bumpTime should make the wait time for Chachi increase to 2

Bank:serve should remove Chachi from the line and update the map from Chachi to -1;

numberServed is now 5 and total wait is 6

Bank:serve should return the string "No one waiting" and the state of the Bank should be unchanged

Bank:averageWait should be 1.2

Bank:customersOver 15 should be a list containing Shirley and Joanie (order unknown)

Bank:customersOver 5 should be a list containing Shirley, Joanie and Laverne

Bank:customersOver 30 should be an empty list

Bank:addToLine Fonzie 100 should add Fonzie to the back of the line with value 100;

accounts should be unchanged