LAB 3: A Java bank account application

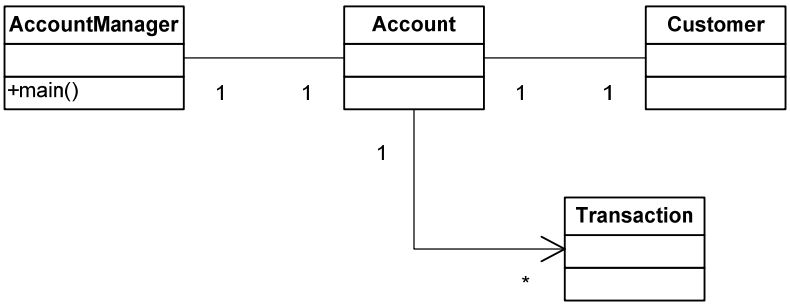
Getting started

In this lab you will complete and test a Java application to mange account transactions in a bank system. The completed application will consist of four classes, `AccountManager`, `Account`, `Transaction` and `Customer`.

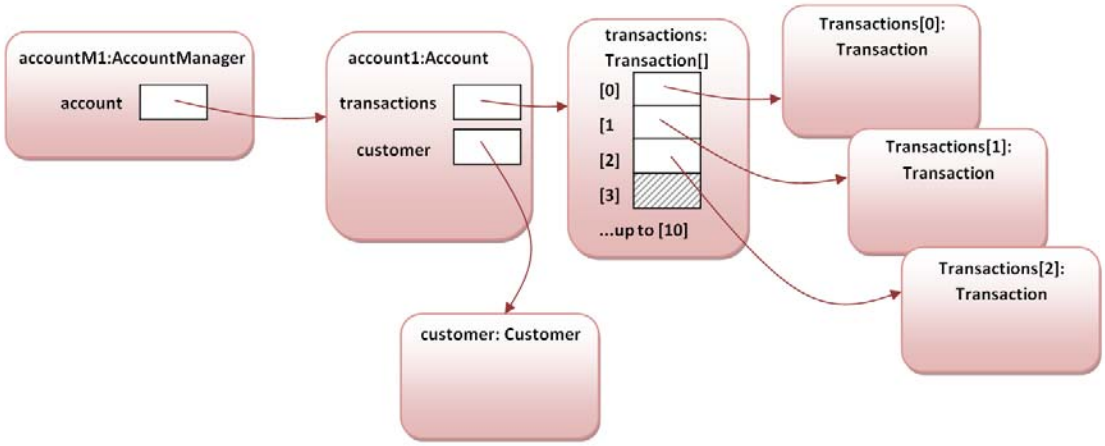
You will be given complete code for the `Account`, `Transaction` and `Customer` classes. The `Account` class is an extended version of the class you created in the previous lab, and you will test this class.

You will be given partially complete code for the `AccountManager` class, which will create a simple text-based user interface for listing, adding and deleting transactions. You will complete and test the application.

As before, an account object will contain an array of transactions and will be associated with a customer. The class diagram is shown below:



The object diagram when several transactions have been recorded might look like this:



Task 1: Unit testing: the Account class

Review class documentation:

1. Download and open the BlueJ project *lab3*. Note that there are four classes. The `Customer` and `Transaction` classes are the same as in the previous lab.

2. The `Account` class is already complete. Unlike the version in the previous lab, it does not read transactions from a file. Instead, this version is intended to be used by another class which handles the input of transaction data.

3. Open the Account class in the editor. Look at the code at the top of the file. Note that there is an array of Transaction objects whose size is defined by a constant as before. There is a field of type Customer which is initialised in the constructor as before.

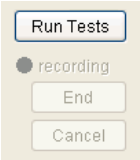
4. Switch from *Source Code* to *Documentation* view in the editor. Find the Method Summary part of the documentation page, and note that the class has methods for adding, finding and removing transactions.

5. Click on **`getTransaction`**. You should now see the detail documentation for that method. Note the **Parameters** and **Returns** information.

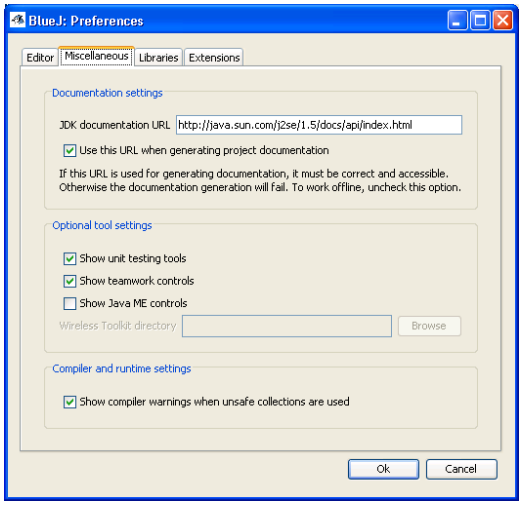
6. Switch back to *Source Code* view. Find the `getTransaction` method. Note how the Javadoc comment for the method relates to the documentation produced.

Create a test class:

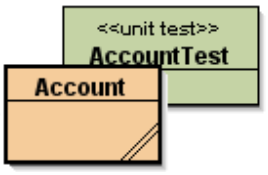
1. Make sure that the unit testing tools in BlueJ are active. You should see the following controls in the main BlueJ window:



   If not, select the Tools > Preferences menu option. Select the Miscellaneous tab and check the Show Unit Testing Tools option (see figure on next page).



2. Right-click the `Account` class in the BlueJ class diagram. Select Create Test Class from the menu. A new unit test class, `AccountTest`, will be added to the project.

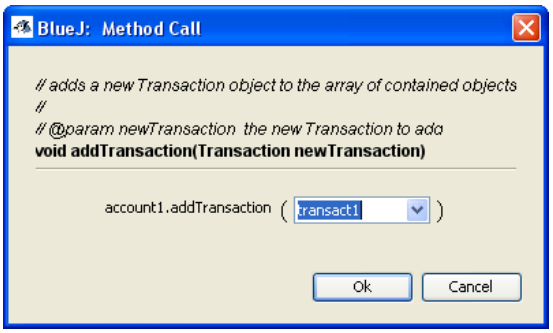


3. Now create some test objects in the Object Bench:

   - A `Customer` object with parameter values: **"Michael", "Schumacher"**
   - A `Transaction` object with parameter values: **200.00, "CREDIT", "ref1", new java.util.Date()**
   - A `Transaction` object with parameter values: **100.00, "DEBIT", "ref2", new java.util.Date()**
   - A `Transaction` object with parameter values: **300.00, "CREDIT", "ref3", new java.util.Date()**
   - An `Account` with parameter values: **customer1** (your Customer object in the Object Bench), **"12345"**

4. Now add the transactions to the account:

   - Right-click on the `Account` object in the Object Bench and call its `addTransaction` method. Give your first `Transaction` object as the parameter in the Method Call dialog (see figure on next page).



- Add your other two `Transaction` objects to the `Account` in the same way. Inspect the `Account` object to confirm that there are three objects in its `transactions` array.

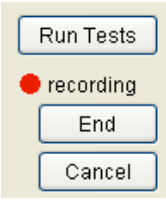
5. Right-click the `AccountTest` class and select Object Bench to Test Fixture. Your objects should disappear from the Object Bench

6. Open the `AccountTest` class in the editor and review the code in the setup method.

7. Your test class is now able to set up a collection of test objects which can be used to test the `Account` class. Right-click the `AccountTest` class and select Test Fixture to Object Bench. A set of objects should appear in the Object Bench. Inspect the `Account` object in the Object Bench to confirm that there are three objects in its `transactions` array.

Create a test method in the test class:

1. Right-click on the `AccountTest` class and select Create Test Method. Name the method `testAddTransaction`. This test will check that transactions have been correctly added to the account. The red "light" shows that BlueJ is now recording your test, and your test objects will appear in the Object Bench if they are not already there.

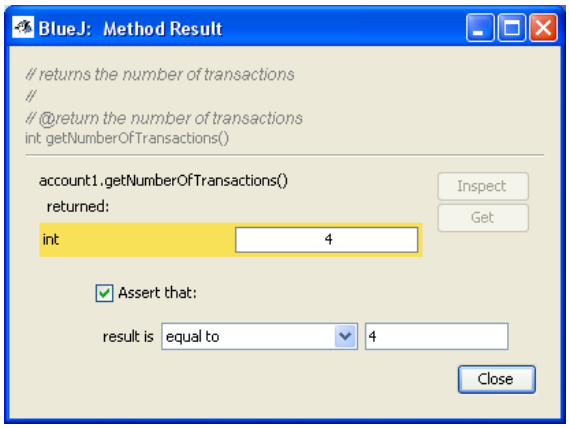


2. Create a new `Transaction` object with parameter values:
200.00, "CREDIT", "test", new java.util.Date()

3. Use the `addTransactions` method to add this `Transaction` to the `Account`.

   The account should now have four transactions and its balance *should* be £600 – actually there is a bug in the code for Account, so the balance will not be £600! You will fix this bug later on.

4. Call the `getNumberOfTransactions` method of the `Account` object. There should now be four transactions, so in the Method Result dialog choose to **assert that the method result is equal to 4.**



5. Click the End button to stop recording.

6. You have recorded a test method which checks the number of transactions after adding a new transaction. We would like it to also check that the account balance is updated correctly, and that the new transaction is in the correct place in the array. To do this, you will add code to the test method rather than recording.

7. Open the `AccountTest` class in the editor, and find the `testAddTransaction` method.

8. Add the following code to the method

```
assertEquals(600.00, account1.getBalance());
assertEquals("test",
    account1.getTransactions()[3].getReference());
```

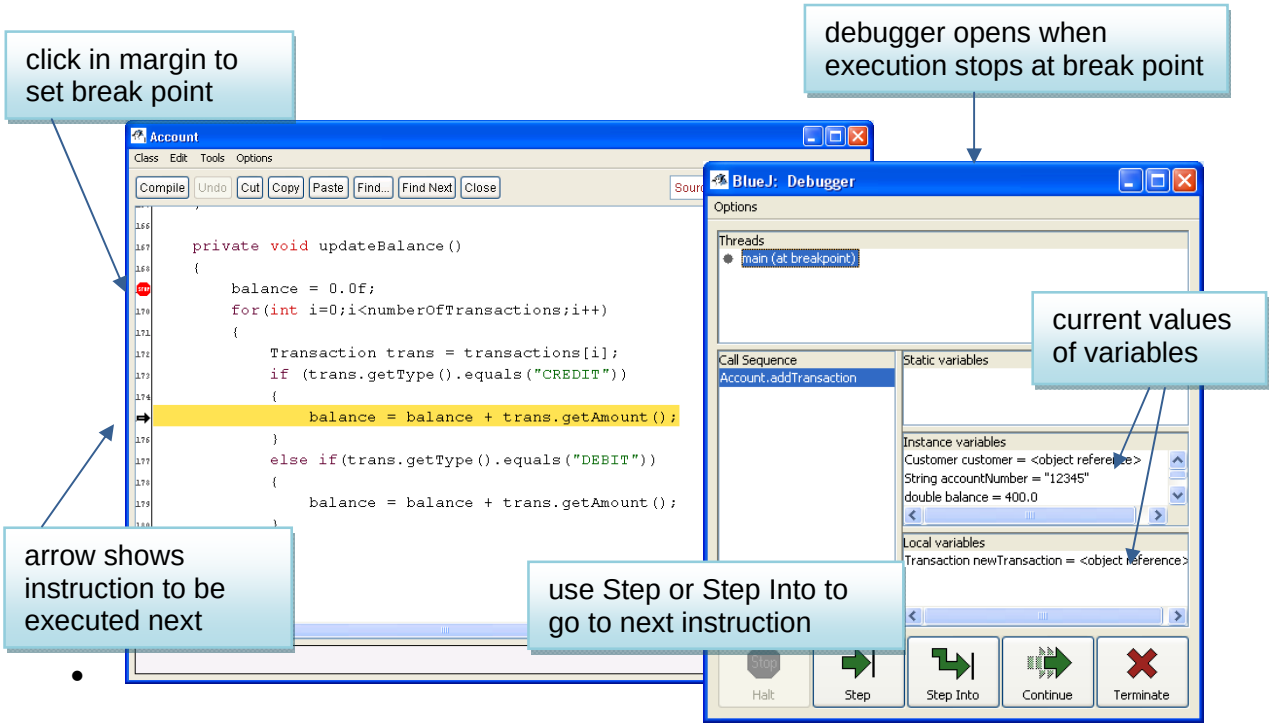
   The second line gets the value of the reference field of the fourth element of the transactions array.

9. Click the Run Tests button. The Test Results window should show a test failure as there is a bug in the code which you have been given for the Account class.

10. Try to find the bug and fix it (*hint:* look at the code in the `updateBalance` method)

    If you wish, you can try using the BlueJ **Debugger** to help, as follows:

    - Right-click the `AccountTest` class and select Test Fixture to Object Bench. Your test objects should appear in the Object Bench.
    - Create a new `Transaction` object with parameter values: **200.00, "CREDIT", "test", new java.util.Date()**
    - Set a **break point** at the first line of code inside the `updateBalance` method.
    - Use the `addTransactions` method to add this `Transaction` to the `Account`. The code execution should stop at the breakpoint.
    - Use the Step button to execute the code line-by line and observe how the value of the `balance` instance variable changes each time round the loop. Think about your test objects and consider what the value of balance *should* be each time.



11. When you think you have fixed the code, run the test again to confirm this.

Create an additional test method:

1. Add another test method to the `AccountTest` class to test the `removeTransaction` method of `Account`. You can record your test or write code for it as you prefer.

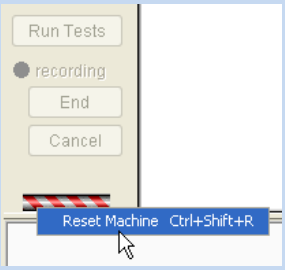
   Note that when tests run, the `setup` method is **run before each test method**. This means that when your new test starts, the account will contain the three transactions defined in `setup`.

2. Click Run Tests again. This will run both test methods. Confirm that both methods are now working correctly.

NOTE: Terminating running code

Sometimes a program **stops responding** and "hangs" indefinitely. If your code running in BlueJ appears to hang, then you can stop execution in one of two ways:

If code is currently executing, the Work Indicator bar in the main BlueJ windows will be red and white, as shown. You can right-click the bar and select Reset Machine.



If the Debugger is open, you can click the Terminate button:

Follow up:

Open assignment *Lab 3* in Blackboard.

Copy and paste your Java code for the `AccountTest` class into the appropriate box in the assignment, and answer the two questions which follow.

Task 2: Completing and testing the application

Look at the AccountManager class:

1. Open the `AccountManager` class in the editor. There are several comments in the code indicating what has to be done to complete the class.

2. Review the instance variables. Note that there is a variable called `reader`, of type `Scanner`. `Scanner` is a library class which allows keyboard input to be read from the terminal window. There is also an instance variable called `account` of type `Account`. This will be the account that the application manages.

3. Find the `printMenu` method. Note that this method prints out a menu with a set of options.

4. Find the `start` method. Review the code and note the following:

   - There is a **while** loop which causes the menu to be displayed repeatedly until the value of finished becomes true

   - The `Scanner` variable, `reader`, is used to get the user's menu choice

   - The code `opt = Integer.parseInt(input)` is used to convert the input to an integer.

     This is done inside a **try-catch** block – why do you think this is?

   - An **if** statement with a series of **else if** statements is used to select the action to be taken for each possible user input.

     Incidentally, there is actually a better way of doing this, using a **switch** statement, which you will see in your lectures

   - The quit option simply sets `finished` to `true`.

Complete the main method:

1. ***// TO DO: set up account and launch application***
   Find the `main` method. This will be the entry point for running the application. Replace this comment **by writing code which**:

   - Creates a new `Customer` object called `customer` with parameter values: **"Rubens", "Barrichello"**

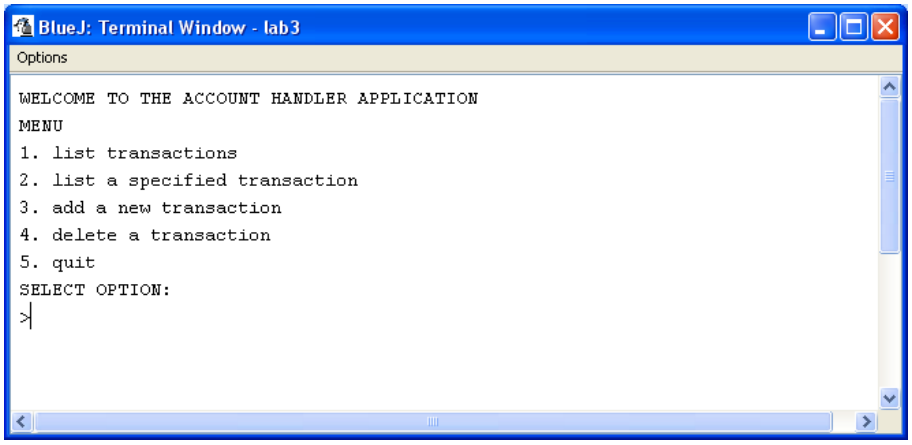
   - Creates a new `Account` object called account with parameter values: **customer, "88888"**

- Creates a new `AccountManager` object called `manager` with parameter value **account**
- Launches the application by calling the `start` method of `manager`

This will allow the application to be run. It will display the menu, but the options will not be fully implemented yet.

Run the application in BlueJ:

1. Right-click the `AccountManager` class in the BlueJ class diagram and select **void main(String[] args)**. Click OK in the Method Call dialog. The application will start up in the BlueJ Terminal Window, and the menu will be displayed.



You can run the application in this way to test each of the menu options as you complete the code to make them work correctly.

Complete the AccountManager class:

Add code as described in each of the following steps to complete the AccountManager class. You can test your code after each of these steps by running the application as described above.

1. ***// TO DO: list account details***
   Find the `listDetails` method. Replace this comment with a call to the `printDetails` method of `account`.

2. ***// TO DO: get transaction with specified reference***
   Find the `listDetailsOfTransaction` method. This method reads user input into the variable `reference`. Replace the comment with code which gets the relevant transaction from `account` and prints its details.

3. ***// TO DO: get reference***
   Find the `recordTransaction` method. This method reads user input into variables which are then used to construct a `Transaction` object. The code to read `amount` and `type` is given. Replace the comment with code to read the value of `reference`.

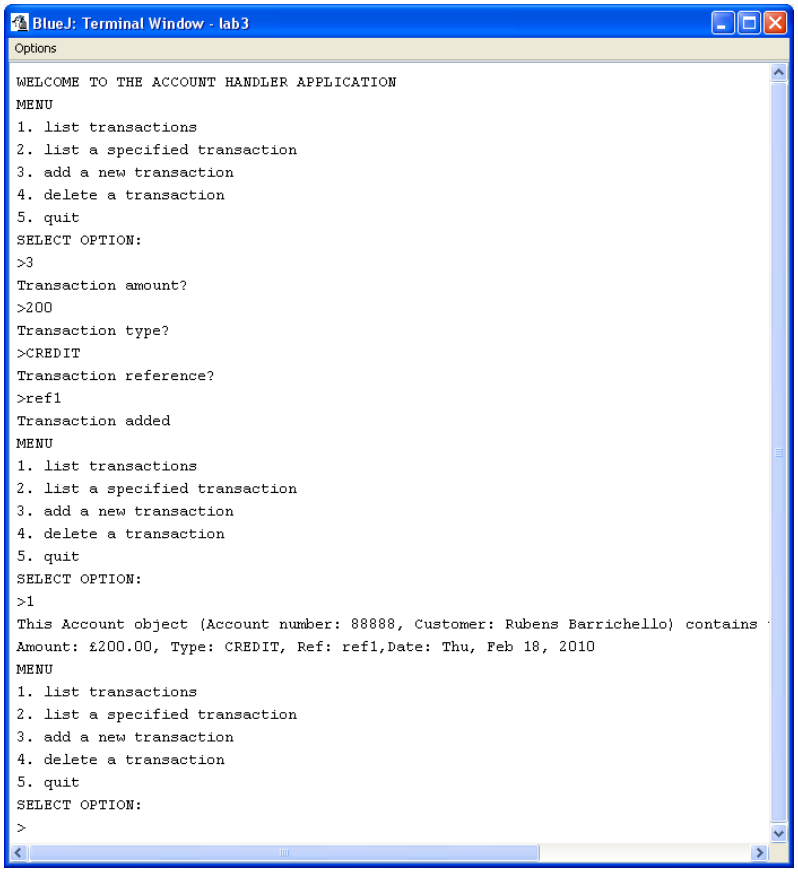
   // TO DO: create transaction and add to account
   Look further down the `recordTransaction` method. Replace this comment with code which constructs a new `Transaction` object and adds it to `account`.

Test the application in BlueJ:

Run the completed application and test by selecting menu options and entering suitable data to:

- Add some new transactions
- List the account details
- List details of one of the transactions
- Delete a transaction
- List the account details again

Example output during a test is shown below:



Note that this is **system testing**, of the completed application, not unit testing.

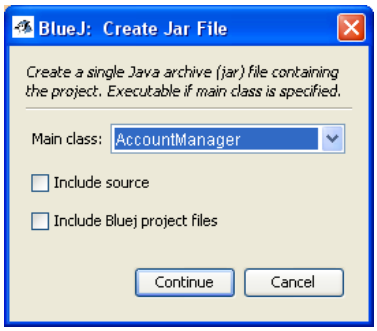
However, even once the application is complete, unit tests should be repeated whenever any part of the code is changed, for example if a new version is created to add new features or fix bugs.

Run the application from the command prompt:

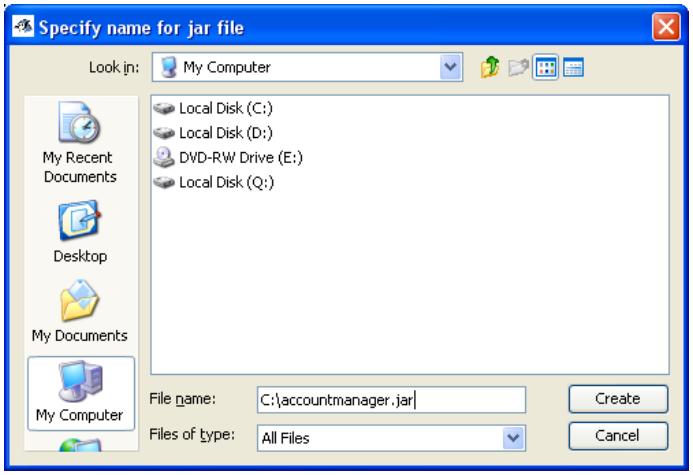
You do not expect users of your application to run it in BlueJ. Applications are usually run by **clicking on an icon** or **typing a command at a command prompt**. You will now prepare your application to be run at the system command prompt.

1. Select the Project > Create Jar File... menu option in BlueJ. This will package the contents of your project into a single, executable file, called a **Jar**. This is similar to a Windows .exe file.

   The `main` method, which is the entry point which the operating system needs to launch the application is in the `AccountManager` class, so you need to specify that this is the main class in the Create Jar File dialog.



2. Click Continue. Choose a location for the Jar file, for example the root folder of the C: drive. Save the Jar file as **accountmanager.jar**.



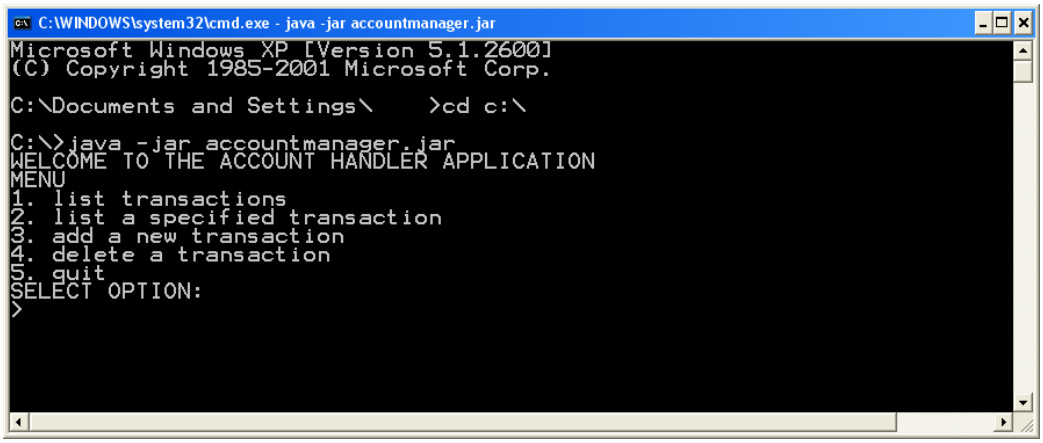
3. Open a system command prompt – in Windows, select the Run option from the Start menu, and enter **cmd** in the Run box.

4. In the command prompt window, change directory to the location where you stored the Jar, for example by entering **cd c:**

5. Enter the command:

   java –jar accountmanager.jar

   If Java is correctly configured on your computer your application should launch in the command prompt window as shown below. You can do system testing as before.



Follow up:

Continue with assignment *Lab 3* in Blackboard.

Copy and paste your Java code for the `AccountManager` class into the appropriate box in the assignment, and answer the two questions which follow.

If you have not completed all tasks, then paste the code as it is at the point you have reached.

use Step or Step Into to go to next instruction