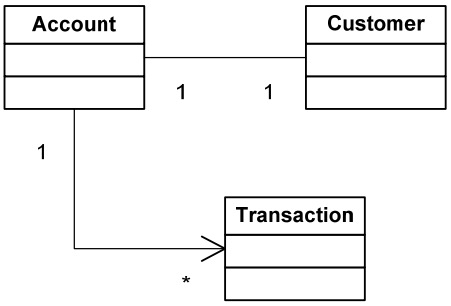
LAB 2: A Java bank account

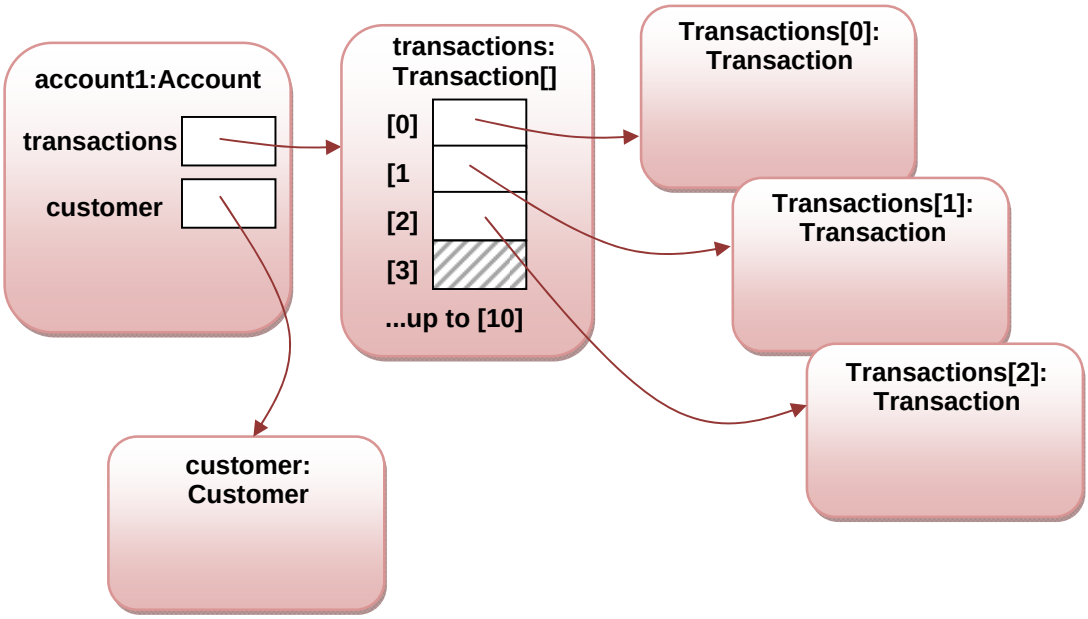
Getting started

In this lab you create and test Java classes which might form part of a bank system. The completed system will consist of three classes, `Account`, `Transaction` and `Customer`. You will be given partially complete code for the `Account` and `Transaction` classes and you will complete and test these classes.

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: Complete the Transaction class

1. Download and open the BlueJ project *lab2*. Note that there are three classes.

2. The `Customer` class is already complete. Open it in the editor and review the code.

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

4. ***// TO DO: instance variables***
   Replace this comment with the following instance variable declarations:

   - A `double` field called `amount`
   - A `String` field called `type`
   - A `String` field called `reference`
   - A `Date` field called `date`. Note that `Date` is a **library class** which represents a date, and provides many functions helpful for working with dates.

     A library class needs to be **imported** before it can be used. Add the following line at the top of your class, replacing **// TO DO: import library classes**

     ```
     import java.util.Date;
     ```

     You will see more examples of library classes in the lectures.

5. ***// TO DO: constructor***
   Replace this comment with a constructor which takes four parameters and uses these to set the values of the four instance variables.

6. ***// TO DO: getters for instance variables***
   Replace this comment with getter methods for all four instance variables (no setter methods are required)

7. ***// TO DO: getDateString method***
   In addition to the getter, it will be useful to have a method which converts `date` to a `String` with a specific date format. Replace this comment with a method `getDateString` which takes no parameters and contains the following code:

```
SimpleDateFormat formatter = new SimpleDateFormat(
      "EEE, MMM d, yyyy" );
String dateString = formatter.format(date);
return dateString;
```

Make sure you choose the correct return type for this method. You will also need to import the library class `java.text.SimpleDateFormat` with an `import` statement immediately after the first import.

8. **Test** the `Transaction` class in BlueJ as follows:

   - Create a new instance of `Transaction` in the Object Bench, entering the following values for the constructor parameters:

     - **200.00**
     - **"CREDIT"**
     - **"ref1"**
     - **new java.util.Date()** - this actually creates a new `Date` object

   - Call all the getter methods to check that all instance variables have been set correctly.

   - The `getDate` method should return a `Date` object, and you can't easily see what date it represents. Call the `getDateString` method. What date does the `Date` field represent?

Follow up:

Open assignment *Lab 2* in Blackboard.

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

Task 2: Completing the Account class

Set up the instance variables:

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

2. ***// TO DO: declare customer***
   Replace this comment with a declaration of an instance variable customer of type `Customer`.

3. ***// TO DO: declare array of transactions***
   Replace this comment with a declaration of an instance variable called `transactions` which is an **array** of type `Transaction`.

4. Now find the constructor of the `Account` class. There are two comments to be replaced in the constructor.

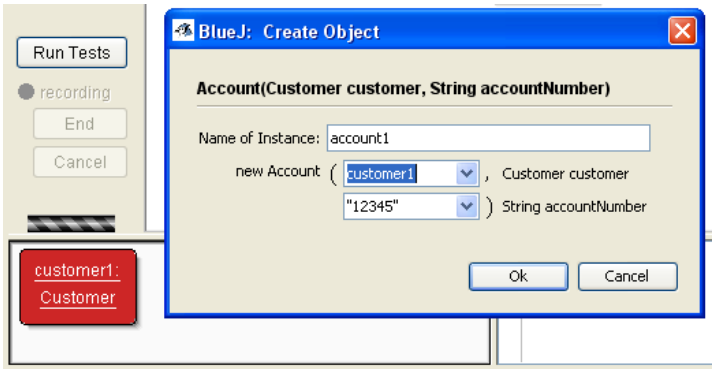
5. ***// TO DO: set customer***
   Replace this comment with a statement which sets the value of the `customer` instance variable to the relevant parameter value.

6. ***// TO DO: create new array for transactions***
   Replace this comment with a statement which initialises the `transactions` instance variable to be a new array of `Transaction` objects whose size is the constant `MAX_TRANSACTIONS`.

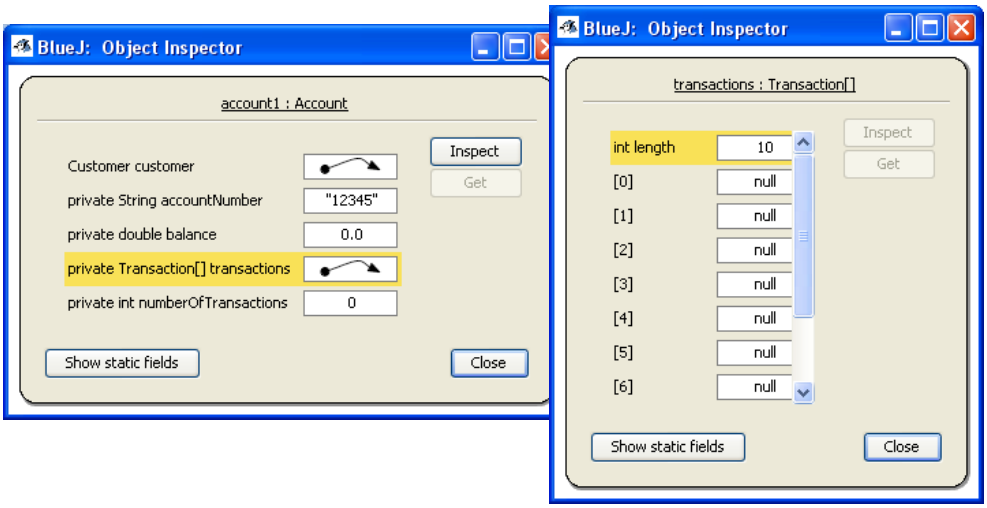
7. Compile the `Account` class. Test the `Account` class in BlueJ as follows:

   - Create a new instance of `Customer` in the Object Bench, entering the following values for the constructor parameters:
     - **"Fernando"**
     - **"Alonso"**

   - Create a new instance of `Account` in the Object Bench, with the following values for the constructor parameters:
     - **the `Customer` object in the Object Bench**
     - **"12345"**



- Inspect the `Account` object in the Object Bench. Select the `transactions` field in the Object Inspector, and now inspect this field by clicking the Inspect button. You should see something like the figure below.



- Close the Object Inspector windows.

Read transactions from a file:

1. Call the `readTransactions` method of your `Account` object. You should see an error message in the terminal window.

2. Find the `readTransactions` method in `Account`. This method reads data from a text file and uses the data to create `Transaction` objects. The file, *transactions.txt*, is inside your BlueJ project folder. Each line of the file contains the four pieces of information needed to construct a `Transaction` object, and there are three lines, as follows:

   200.00,CREDIT,ref1,1/12/2009
   550.00,DEBIT,ref2,24/12/2009
   100.00,CREDIT,ref3,7/1/2010

   The code in `readTransactions` uses some techniques which we do not have time to study in depth here, so in this exercise you will simply add some code to complete the method.

3. The reason for the error message was that the name of the file which contains the data was not specified correctly. Find the comment

   // TO DO: make filename a constant.
   In this line, replace *"wrong_name"* with the name of a constant, `FILENAME`.

4. You will need to define the constant `FILENAME`. Go back to the top of the `Account` class and find the definition for the constant `MAX_TRANSACTIONS`. Find the comment

   // TO DO: define constant for filename
   Replace this comment with a new `String` constant definition for `FILENAME = "transactions.txt"`.

NOTE: file input/ouput is a common source of **errors**, if for example you try to read a file that does not exist. Errors can cause a program to "crash" if not handled properly. Note that most of the code in this method is placed inside a `try` block, with a `catch` block to handle and warn about any errors, or **exceptions**, which occur. With this structure, the program can continue after a file error.

```
try
{
      Code which may cause an error...
}
catch(Exception e)
{
      Handle the error...
}
```

5. Return to the `readTransactions` method. The code reads each line of the file, within a while loop, and uses the data to set the values for four variables:

   - `amount`
   - `type`
   - `reference`
   - `date`

   The number of lines read so far is stored in a variable `count` which is incremented each time round the loop.

6. ***// TO DO: create transaction and store in array***
   Replace this comment with a statement which constructs a new `Transaction` object using these values as parameters.

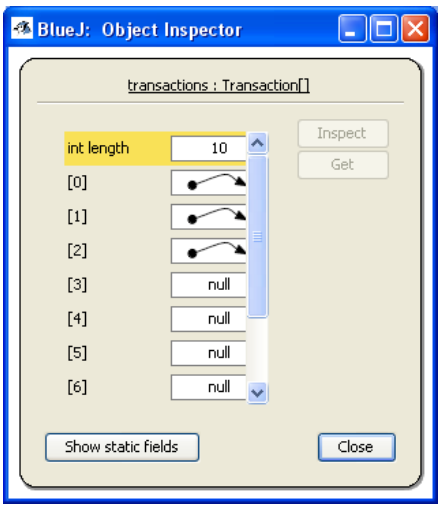
7. Add a statement immediately after this which sets the element of the `transactions` array with index `count` to refer to the `Transaction` object you have just constructed.

8. ***// TO DO: update number of transactions***
   Replace this comment, which is positioned after the end of the loop when all transactions have been read, with a statement which sets the value of the `numberOfTransactions` instance variable to `count`.

9. Compile the `Account` class. Test the `Account` class in BlueJ by creating `Customer` and `Account` objects as in the previous test.

   - Now call the `readTransactions` method of the `Account` object. You should get a reassuring message in the terminal window.

   - Inspect the `Account` object and its `transactions` field as before. You should see that the array now has some objects in it.



Displaying transactions and updating the balance

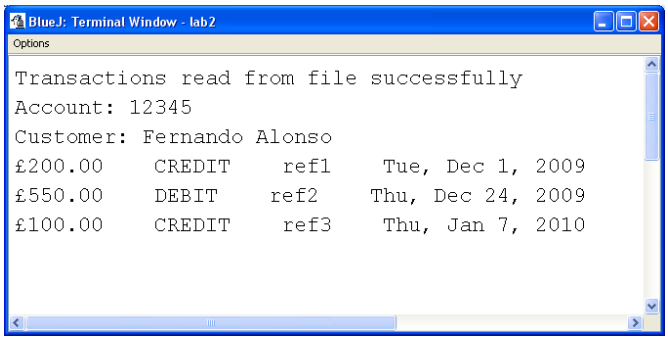
1. Find the `displayTransactions` method in the `Account` class.

2. ***// TO DO: loop through transactions and display***
   Replace this comment with a `for` loop which prints out the details of each transaction in the `transactions` array. Use the following code inside your `for` loop (your loop count variable should be called `i`):

```
Transaction trans = transactions[i];

System.out.format("£%4.2f    %s    %s    %s\n",
    trans.getAmount(),
    trans.getType(),
    trans.getReference(),
    trans.getDateString());
```

3. Compile the `Account` class. Repeat the previous test, and this time also call the `displayTransactions` method. You should see the following output:



4. Find the `updateBalance` method in the `Account` class.

5. ***// TO DO: loop through transations and add amount to/subtract***
   // amount from balance
   Replace this comment with a `for` loop which adds the amount of each transaction to the `balance` instance variable if the transaction type is "*CREDIT*", or subtracts it if it is "*DEBIT*". Use the following code to add or subtract the amount:

```
if (trans.getType().equals("CREDIT"))
{
    balance = balance + trans.getAmount();
}
else if(trans.getType().equals("DEBIT"))
{
    balance = balance - trans.getAmount();
}
```

NOTE: the correct way to check whether strings are equal is to use `equals` – you should not use `==` for strings, as this does not always work as you would expect.

5. Compile the `Account` class. Repeat the previous test, and this time also call the `getBalance` method. You should get a return value of -250.00.

Follow up:

Continue with assignment *Lab 2* in Blackboard.

Copy and paste your Java code for the `Account` 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.