Lab 4: Bank Accounts

Steven K. Andrianoff
Robert Harlan
David Levine
Computer Science Department
St. Bonaventure University

Objective:

This lab introduces particular aspects of class construction including multiple constructors and overriding the *toString* and *equals* methods. The lab also shows how to *throw an exception* and how to generate an *API* for a class file using *javadoc* documentation.

Instructions:

1. Create a project named Bank. Next, create a source code file (choose New/Class) named `BankAccount.java` in the project.

Define a BankAccount class that maintains the balance, the account number, and the name of the customer (one name is sufficient) associated with a bank account. Include a three-parameter constructor which initializes the data fields and methods to deposit funds in the account, withdraw funds from the account, and obtain the current balance. At this point in time, you needn't worry about invalid parameter values. (We'll deal with that in a later step in this lab.)

Document the class with javadoc style comments that include a brief overview of the class, your name(s), and the date. Document each method with javadoc style comments that include a brief description of the method and any parameters or returns.

Answer the following questions in your lab write-up:

- What type of data should each data field store?
- Why would it be better to store the account number as a String rather than an int?
- Which methods are *accessors*, i.e. which methods access the object without changing any of the data stored therein? Which are *mutators*, i.e. which methods cause the object's state to change in some manner?

2. Write a main class with code to test your BankAccount class. Test only those features that have already been implemented: the constructor and the methods.

Print a copy of the output generated by these tests.

3. Write a second constructor that requires only an account number and a name. This constructor should initialize the balance to be 0. Have this second constructor make a call to the first constructor (you wrote in Step 1) with the appropriate values. Ask if you don't know how to do this (and I believe that most of you will not know how.)

Document the constructor with javadoc style comments and add code to the main method to test this second constructor.

Print a copy of the output generated by these tests.

4. We know from class that every class one defines in Java extends (inherits from) the base class Object. Typically one should override the *toString* method of the Object class. The purpose of the toString() method is to return a String form of the object.

Define a *toString* method with the signature:

```
public String toString( )
```

The method should return a String with the object's data in the form:

```
Account Num: <account num>, Name: <name>, Balance: <balance>
```

Document the method with javadoc style comments and add code to the main method to test the method.

Print a copy of the output generated by these tests.

5. Another method from Object that is often overridden is the *equals* method. The equals method has the signature:

```
public boolean equals(Object rhs)
```

The equals method is to compare a second BankAccount object (rhs) with this object and determine whether or not the data fields store the same values.

This code gives an example of an overridden equals method for the TimeA class. In particular you need to first check that rhs is an instance of the BankAccount class using the *instanceof* operator. Then you need to "cast" rhs to a BankAccount and then finally compare the fields of the converted rhs and this for equality. In the case of String fields you need to use the equals method.

Document the method with javadoc style comments and add code to the main method to test the method. In particular, illustrate two BankAccount objects that are not equal using *equals*. Illustrate two other BankAccount objects that are equal using *equals* but not using ==.

Print a copy of the output generated by these tests.

6. Next we want to protect against overdrawing an account. One thing that could be done is to display an error message in the withdraw() method if the amount exceeds the current balance. A better method is to throw an appropriate exception. There is an exception class IllegalArgumentException that we will use in this case. Modify the code for the withdraw method to check if the amount to be withdrawn will lead to an overdrawn account. If the account would be overdrawn throw the exception (and don't make the withdrawal). This statement will throw the exception:

```
throw new IllegalArgumentException();
```

Modify testing code in the main method to ensure that you are correctly testing these conditions.

Print a copy of the output generated by these tests.

7. Finally you are to generate a web page containing the *API* for the BankAccount class. (This is also known as the *specification* for the class.) If you have documented your code using javadoc style comments then all that is required is to run the javadoc utility on the file which will generate an html file from the documentation that can be viewed with a web browser.

The first step is to write the comments in the file in a form recognized by the javadoc utility. You should have already done this. You should have javadoc comments for the class and for each method. A javadoc comment starts with /** and ends with */. (Refer to section 1.9.3 of the textbook for further information and an example.) The class comment should include a brief description of the class along with your name(s) and the date. The comment for each method should include a brief description of the purpose of the method as well as a @return tag if the method returns a value and a @param tag for every parameter of the method. If a method throws an exception then a @throws tag should be included to identify the exception thrown.

If you know HTML you can embed HTML tags in javadoc comments to control formatting, specify fonts, etc. Most importantly, javadoc automatically provides *hyperlinks* to other classes and methods.

To run the javadoc utility on the files in your project, right-click on the project, select Export . . ., choose javadoc and accept the defaults. This will leave a folder named Doc in your project. Open the file `index.html` in this folder. This will give you the API for your BankAccount class.

Scroll through the API and read the comments you wrote. After reading them in this form go back into your code and change any comments that are not complete.

Upon completion of this step print the source code file for each of the two classes (`BankAccount.java` and the main class that you used for testing) and print the `index.html` web page.

Hand in:

The write-up you hand in for this lab should include:
- answers to the questions posed in Step 1.
- output from the tests in Steps 2-6
- a copy of the two classes as requested in Step 7.
- a copy of the API (web page) from Step 7.

Help Policy:

Help Policy in Effect for This Assignment: Group Project with Limited Collaboration

In particular, you may discuss the assignment and concepts related to the assignment with the following persons, in addition to an instructor in this course: any member of your group; any St. Bonaventure Computer Science instructor; and any student enrolled in CS 132.

You may use the following materials produced by other students: materials produced by members of your group.