COMP1161 Lab 2 – Week of January 28, 2013

Chapter 3: Using Classes and Objects
Lab Exercises

Objectives:

On completion of this lab the student should be able to write a program that uses objects. Specifically, the student should be able to write Java code to:

1. Declare an object
2. Use the `new` keyword to create an instance of a class (an object)
3. Explain the difference between a reference variable and primitive data type
4. Invoke a method on an object
5. Read the API documentation for a class in order to use the class in a program

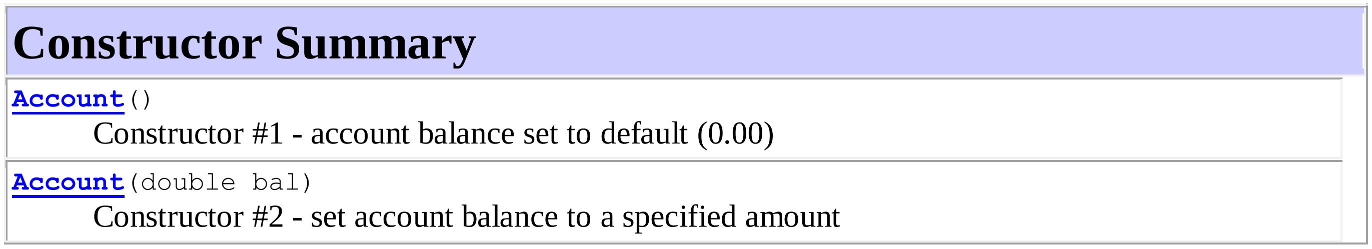
Introduction

One of the strengths of Object-Oriented Programming is that it provides strong support for software reuse. There are thousands of classes that you can use when writing a program instead of writing all code from scratch. For example, the `String` class provides functionality to strore a string and to perform various operations such as conversion of characters from upper to lower case, searching for characters in the string, and a host of other interesting things. Reuse makes it easier to write programs faster since it saves you time.

Reading Application Programming Interface (API) Documentation

As a Java programmer you will often need to use classes that are written by other programmers, or provided by the Java programming environment. In order to do this you must know what methods are provided by the class. The Application Programming Interface (API) documentation that is typically supplied with a class provides you with this information. In fact, you can get documentation for Java classes from the web site www.java.sun.com but you may find the documents there a little too complex for you at this stage. For now we will look at documentation for a simple class which has been created just for you.

1. Open the file *Account.html* on the course web site. This file contains documentation for the API to a class named `Account`. The document has several sections. For no2 we will focus on four sections:



Constructor Summary
`Account()` Constructor #1 - account balance set to default (0.00)
`Account(double bal)` Constructor #2 - set account balance to a specified amount

   a. **Constructor Summary** is a quick listing of constructors for the class. A constructor is a special method that is executed when a new instance of the class is to be created. Its purpose is to set up (initialize) the values that the newly created object will store. As shown in the documentation, there are two constructors for the `Account` class. Any one may be used depending on what values (in this case a balance) you wish to start an account with.

b. **Method Summary** is a quick listing of all the methods that you can invoke on an instance of this class. Each mehod has a name and a type. A method which is of type `void` is not expected to return a value. One of the methods shown here if of type `double`. This means that when the method is run it will return a value of type `double`.

Method Summary	
`double`	`balance()` check account balance
`void`	`deposit(double amt)` deposit an amount to account
`void`	`withdraw(double amt)` withdraw an amount from account

The other two sections of the documentation provide details of the constructors and methods that are listed in the summary sections.

c. **Constructor Details** gives a more detailed description of each constructor including a description of each argument (if any) that must be sent when the constructor is used.

d. **Method Details** gives a detailed description of each method that you can ivoke on an instance of the Account class. The details include a short description of each argument that the method expects to be sent when it is used. The details can sometimes include an example.

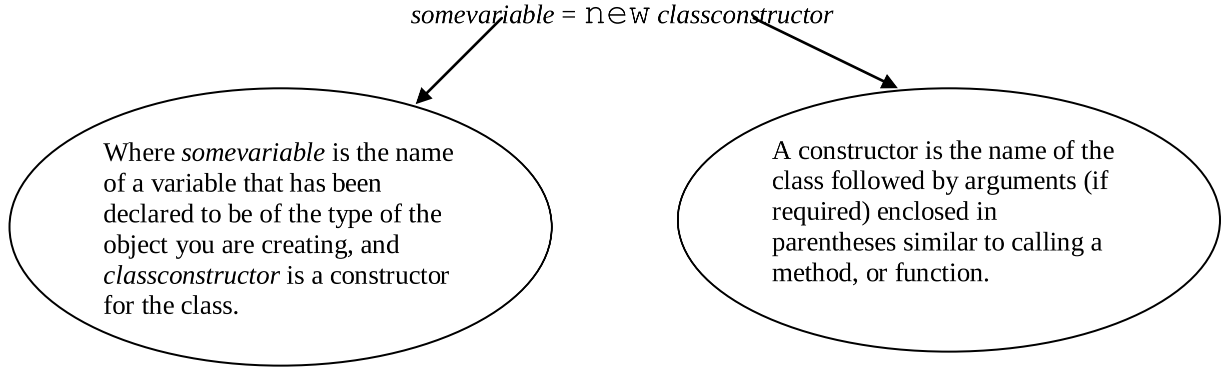
BEFORE PROCEEDING PLEASE DO THE FOLLOWING:

1. Create a java package for lab2.
2. Download the file `Account.java` from the course website. Ensure that the file is saved to the lab2 folder on your computer *(right-click on the file on the course website, select the 'save as', then find your folder for lab2 and save it there).* It is important that this be done properly as a mistake will mean that you program will not know about the `Account` class.

Creating an object

The Java keyword `new` is used to create a new instance of a class (an object). In typical use, the keyword is used in the form:

somevariable = `new` *classconstructor*



Where *somevariable* is the name of a variable that has been declared to be of the type of the object you are creating, and *classconstructor* is a constructor for the class.

A constructor is the name of the class followed by arguments (if required) enclosed in parentheses similar to calling a method, or function.

There are two different versions of the constructor for the `Account` class. One version does not require any argument. The other requires a single argument of type `double`. The following code segments illustrate how each of the constructors might be used:

```
            …
Account acc1, acc2, acc3;// Declare three Account objects

            …
acc1 = new Account();    // Using constructor #1 – nor arguments

            …
acc2  =  new Account(100); // Constructor #2 – balance in account is 100.00
            …
double  amount = 125.00;      // a variable of type double
acc3 = new Account(double * 0.90); //Argument is an expression that gives
            …                        // a double as its value
```

Question: What is the new balance in each of the accounts above?

Commuicating with an Object (Using Methods)

Your program communicates with an object by invoking the object's methods. The `Account` object has three methods that can be used to: 1) deposit money to the account; 2) withdraw money from the account; or 3) check the balance on the account.

In order to use a method you write the name of the object, followed by a period (a dot), then the name of the method (with arguments, if required).

Here are some examples of using the methods that are available for an Account object. You may find it useful to create a program with this code as practice for the upcoming exercises.

```
            ….
Account myAccount = new Account(100);                             //1
oldBalance = myAccount.balance();                                 //2
myAccount.deposit(200);                                           //3
myAccount.withdraw(50);
double newBalance = myAccount.balance();
System.out.println("Balances: + oldBalance + " " +newBalance);
System.out.println(myAccount.balance());                          //5
            ….
```

The numbers given as comments illustrate the following::

1. You can declare a reference variable and initialize it at the same time just as you would with a primitive variable by including the instantiation in the statement;
2. The value that is returned by a method can be used in an expression.
3. Using a `void` method can be thought of as instructing the object to carry out an action for which no response is needed.
4. Again, the value that is returned by a method can be used in any place where the type of value that is returned may be used.

Aliasing

Try the following code and discuss the results you receive. You will have to add code to display the account balances.

```
                 …
Account johnAcc, maryAcc;

johnAcc = new Account(100);
maryAcc = new Account();

johnAcc.withdraw(50);
maryAcc.deposit(100);

maryAcc = johnAcc;

maryAcc.withdraw(50);
                 …
```

In the above code segment, both John and Mary initially have their individual accounts. Then Mary and John got married, now Mary and John's have the same account (they now both have access to the same data). We can see where aliasing is appropriate here, as both John and Mary can manipulate the same data.

Comparing Objects

The == operator is used to compare values of two primitive data types. For example, the following statements should not cause any surprises. Try it is you wish.

```
        ....
int x = 25;
int y = 30;

if (x == y)
      System.out.println("They are equal");
else
      System.out.println("Not equal");
```

Can you explain why the following code, though quite similar, will never compare the two objects x and y.

```
Account x = new Account(100);
Account y = new Account(100);

if (x == y)
      System.out.println("They are equal");
else
      System.out.println("Not equal");
```

Ok. Remember! An object is a reference data type. This means that comparing two objects by using the == operator will not actually compare the objects themselves. Instead, the == operator will compare the object ids.

Every properly written class provides a method called the `equals` method that can be used to compare two objects. Unfortunately, the `Accounts` class we are using does not have an implementation of this method.

Instead we will use the String class to demonstrate. To compare two strings `s1`, and `s2`, the code would be:

```
            if (s1.equals(s2))
```

Similarly, you can check to see if a string object has a given value as follows:

```
            if (s1.equals("Hello")
```

Here is a really crazy looking case. I wonder what the result is (true/false?)

```
            if ("Hello".equals("Hello"))
```

You will learn how to write an `equals` method later on in this course when you learn how to write classes. For now you should just remember how to use it for comparing two strings.

Before you go on to the exercises try the following:

1. Go to the web site www.java.sun.com
2. Select the API link and chose JSE 6 to display the API documentation for classes that are provided by the Java standard environment.
3. Browse to find the `String` class. You will notice that the class has several constructors and quite a few methods.

<u>LAB EXERCISES (Please upload using submission link)</u>

Exercise #1

Create a class named `MyAtm` that will allow you to carry out transactions on your accounts. The program should:

1. Create two `Accounts` objects, one for a savings account, the other for a chequing account. You can name the variables as you like..
   .
2. Display the following text menu:

```
                    WELCOME TO MY ATM

          DS   -  Deposit to Savings Account
          DC   -  Deposit to checquing Account

          WS   - Withdraw from Savings Account
          WC   - Withdraw from Chequing Account

          BL   - Display Account Balances

          Q    - Quit the program

     Select an option ->
```

3. Input the user's choice. You may find it preferable to use the `nextLine()` method of the Scanner class to do this.

4. If the user has input the string "BL" then the program should display the account balances. Format the display nicely to your own liking.

5. Otherwise, the program should prompt the user to enter an amount and carry out the appropriate deposit/withdrawal on the savings/chequing account..

Exercise #2

Modify your program so that the user does not necessarily have to input upper case letters. (Hint: use the to `UpperCase()` method of the `String` class to do this).

Exercise #3

Modify your program to include an option to transfer funds from the savings account to the chequing account. Selecting this option should cause the program to prompt for an amount then to complete the transfer.

Discussion Questions

a) Test the program to see whether or not it is possible to withdraw more money than there is in an account. How would you overcome this problem if you were the author of the class?
b) You will most likely have used the `nextDouble()` method of the `Scanner` class to input the amount of mony that is to be deposited/withdrawn in each case. Without necessarily writhing code, can you suggest an approach that will safeguard against this happening? Some answers might be found in the documentation for the `Double` class (the wrapper class for valuesof type double).