COMP1161 Lab 3 – Week of February 4

Chapter 4: Writing Classes
Lab Exercises

Objectives:

On completion of this lab you should be able to write code for a simple class given a description of the operations that the class shall perform. Specifically, you should be able to:

1. Write Java code for a simple class given a description of the class' attributes and methods.
2. Write a driver class to test another class

Topics to be covered:

- Members of a class (attributes and methods)
- Visiblity modifiers and encapsulation
- Method declaration and parameter passing
- Accessor methods and mutator methods
- Constructors
- Documenting your code with javadoc

Preamble

A description of the operatoions that a class can perform tells you **WHAT** the class can do but does not say **HOW** the class does these things. For example, you know that an Account object can be used to store information about an account including the account balance, and other information. You also know that the object allows you to carry out certain operations on an account. You can check the account balance, make a deposit to the account, or withdraw an amount from the account. The details of how these tasks are carried out are hidden from you.

When you write a Java application you create instances of predefined classes (objects) and use the methods that are defined in each class' interface to manipulate the information that each object stores. In the case where there is no predefined class that satisfies your needs you will need to write the code for the class yourself before you can use it in your application. In this lab you will view the Java code that implements the `Account` class and make some modifications to the class so that it stores more data and provides additional methods.

At the end of the lab you should be able to write a class from scratch given a description of what the class does. That is, you should be able to determine what values an object that is created from the class will store and be able to write the methods that are needed to use the object.

Startup Activities

1. Dowload the file lab3.zip
2. Extract the file *Account.java* from the zip file and save it to a location of your choice.
3. Create a package named lab3.
4. Use the command "**Insert Class from File…**" (select this from the **Edit** menu) to add the Account.java file to the package you just created. Please ensure that you are working in the lab3 package when you do this.
5. You should see the class *Account* displayed in your BlueJ window. If not then you need to go back and do tasks 1 to 3 ensuring that you follow carefully.
6. Open the Account class so that you can see the code.

Anatomy of a Class

A class has two sets of code:

1. Variables are used to store the data that an object will need to manage. These are properly referred to as **attributes**. An attribute can be a either primitive data type or a reference data type (i.e. an object).

2. The second type of code that is found in a class is code that carries out operations. This code is organized into methods. A method is a block of code with a **method header**.

```
public class Account
{
     private double balance;  //variable declaration


     public Account(double bal)// method header //
     {

     }

     public void deposit(double amt)// Method header
     {

     }

}
```

Attributes and methods that are declared in a class are referred to as **members** of the class.

Please note that the code you are viewing has been documented with directives that a special tool (the javadoc tool) uses to produce the API document for a class. These directives (e.g. @author, @version, @args, @return, etc) are embedded blocks of comments that start with "/**". This makes the code a little "heavy" on the eyes but gives you the benefit of automatic generation of automatically generating the API documentation so that others can use your class without having to see the code.

Declaring Variables (Attributes)

Java is a strongly typed language. Every variable that is to be used mst be declared prior to its use. One variable is declared in the class Account.

```
private double balance;
```

An account object will manage (know) its balance.

Data Scope

Variables that are declared outside of methods can be used by any method in the class. For example, any method in the class `Account` can use use variable `balance` in an expression (e.g. `balance = balance + 10.0`).

A variable that is declared inside a method is known as a **local** variable. It is not "known" outside of the method. A local variable only exists as long as

Visibility Modifiers

A variable declaration is of the form:

```
modifier  type  variable;
```

A modifier is a special keyword that affects some aspect of the variable. For example the keyword `private` (called a **visibility modifier**) is used in the declaration of the variable `balance`. Visibility modifiers are used with both variable declarations and method declarations. A visibility modifier is used to determine the scope of access that is allowed for a class member. Java has four visibility modifiers. These are:

`private`	this member can only be accessed (is visible) by other members of this class.
`public`	this member can be accessed by members of other classes.
`protected`	this is a special type of access modifier that is used for inheritance. More on the use of the `protected` vivibility modifier when we study inheritance and polymorphism.
`package`	a class member with this visibility modifier can be accessed by any member of the same package. Actually, this is the default access if no modifier is explicitly given. For this reason you will not normally see the word `package` explicitly used as a visibility modifier.

There are other types of modifiers that are used in Java. For example, the modifier `final` is used to declare a constant.

Declaring Methods

A method is like a function in non-OOP languages except that a method is invoked by using the name of an object followed by the name of the method with a period (".") separating the two. Recall that you used the following code in a previous lab:

```
acct2.deposit(100.00);
```

This code is an example of **invoking** a method on the object `acct2` which is of type `Account` (an instance of the `Account` class). The desired action is that the balance in `acct2` should be increased by 100. For example, if the account balance was 100 before the method was invoked it should be 200.00 after the method is invoked.

The programmer who wrote the `Account` class wrote code to do this. He/she did this by declaring the method `deposit` and writing code to change the balance.

A method declaration is of the form:

```
modifier type methodname(parameters….)
```

Locate the method declaration for the deposit method. You should see the following code:

```
public void deposit(double amt)
{
     balance = balance + amt;

}
```

The visibility modifier for this method is `public` meaning that members of other classes (specifically methods) can invoke this method.

The type `void` means that the method is not expected to return a value.

The variable `amt` is a referred to as a formal parameter. It is a "place holder" for a value that the method is invoked with. In other word, when you invoke the method `deposit` with the value 100.00, that value is used whever the placeholder appears to achieve the operation `balance = balance + 100`. The value 100.00 that you invoke the method with is referred at as an **actual parameter**, or an **argument**.

The variable `balance` is the same variable that was declared at the top of the class. It is an **instance variable** meaning that each instance of `Account` that is created has its own balance. You will learn about another type of variable that is called a **class variable** later on in your course.

Now look at the code for the the method `withdraw`. You should see that this method operates by subtracting an amount from the account balance.

Accessor Methods vs. Mutator Methods

A method that changes a value (or values) of an object is referred to as a **mutator** method. As you can see both the `deposit` and `withdraw` methods are mutators.

An **accessor** method is a method is used to query a value that the object knows. The method `balance()` is an accessor method. Just like the other methods it can be called from any other class but its type is `double`. You will recall that you used this method as follows:

```
System.out.println(acct2.balance());
```

When this statement is executed, the balance() method is invoked on the object acct2. Now look at the code for this method. It says:

```
return balance;
```

This means that the object responds by returning the value that is stored in the variable `balance` (the instance variable).

So, if the account balance is 200.00 then the print statement will result in the value 200.0 being printed.

QUICK REVIEW

Please ensure that you understand the following concepts before proceeding:

- Attributes
- Methods
- Members of a class
- Visibility members
- Accessors
- Mutators
- The `return` statement
- Local variables
- Instance variables

Constructors

A constructor is a special method that is run whenever an object is created. Its purpose is to set up the values in a newly created object. You will recall that you previously wrote code of the form:

```
Account   acct2 = new Account(100.00);
```

The keyword `new` instructs the Java environment to create a new `Account` object (an instance of the class `Account`). The class name is used along with parentheses. The parentheses are used to cause invocation of a constructor.

There can be several constructors defined for a class. The number and types of arguments that are used in the invocation determine which one of the constructors should be run. In this case you provided a single value of type `double`.

Now examine the the `Account` class code. You will see that there are two methods which are simply named `Account`. One does not have any parameters. The other declaration of a method named `Account` has a single parameter named `bal` which is of type `double`. This is the method that will be run in this case. The code that is executed is simply:

```
balance = bal;
```

Thus, the newly created account will have its balance set to the value (the argument) that was given to the constructor (in this case, the value 100).

Construtors are quite interesting, and quite often misunderstood. Here are few points:

- A constructor declaration does carry a type. Neither should the type void be used since technically a constructor is not regarded as either a mutator or an accessor method.

- A constructor has the same name as its class. Consequently, it is the only method you will write whose name will start with an uppercase letter.

- Constructors can be quite complex and can use Java structurers (e.g. if, while, etc.) that are found in other types of methods except that you will not find the `return` keyword in a constructor's code.

- Actually, you are not compelled to write a constructor for a class. Java provides a default constructor (one that takes no arguments) for every class. The default constructor will assign defaults to all attributes. Numeric attributes are given the value zero. Reference type variables are set to null and boolean primitives are set to false.

Encapsulation

The attributes of a class are almost always declared as `private`. In some special cases (e.g. for constants) public access is allowed.

The reason for making attributes private this is that an object should be *self-governing*. That is, it should take responsibility for managing the data that it holds. Allowing unrestricted access to attributes would be dangerous as other objects could access these attributes and make unregulated changes to their values. Instead, an object should only other objects to access its attributes through its methods. That way other objects can only make changes to attributes as allowed by methods.

Documenting a Class

BlueJ allows you to toggle between viewing source code and viewing documentation. The control that does this is at the top right hand corner of your BlueJ window. Try this now to see what the API documentation for your class looks like. The doeumentation is autoamatically generated from the special tags that are inserted in comments that start with the special characters "/**". A block of comments that start with these special characters is placed just a before method declaration is used to explain what the method does. The comments can include the following tags:

@author	you can type your name in the space after the tag.
@version	you can write a version number. Try 2.00a
@param	use this tag to describe the purpose of each method parameter.
@return	use this tag to describe the value (if any) that a method returns.

Try changing some of the text that now after these tags and toggle the view to see how the documentation is automatically updated to reflect your changes.

The tool that generates this documentation is called **javadoc**. When you supply a program to another programmer you will not always give him/her the source code. Instead you will normally give him/her a .jar file with just the byte-code. Since the programmer cannot look at the source he/she must rely on this documentation to know how to use the class. That is, what information must be provided to a constructor, and what methods can be used with the class.

Lab Exercises

The following exercises will require that you make improvements to the `Account` class. You can use your MyATM class to test the class. Of course, since some of the methods of the `Account` class will change, you will also haveto make changes to the code of the `MyATM` class where needed.

1. Modify the `withdraw` method so that it only changes the account balance if the balance is more than the amount being withdrawn. Also, change the type of the method so that it returns the amount actually withdrawn. For an unsuccessful withdrawal the value returned will be 0.

2. Modify the `Account` class so that an account can store a customer number. A customer number is an alphanumeric code consisting of 4 alphabetic characters and 4 digits. For example, "CUST0001". Use an attribute of type `String` to store the name.

3. Write a third constructor which has two parameters being the customer number, and starting balance, in that order. Remember, the constructor should not have a type, nor should it be `void`. At the same time modify the MyATM class so that both accounts bear the name "Usain Bolt".

4. Modify the `toString` method of the `Account` class to include the customer number.

5. Modify the class so that it keeps track of the highest balance achieved. For example, if the account starts with a balance of 100 and there is a deposit of 75 followed by a withdrawal of 100 then the highest balance is 175.*[Hint you wil have to add an attribute to do this. You will also have to modify an existing methody.]*

6. Write the method `accumulateInterest` which accepts a single value of type `double` being an interest rate. The method shall calculate interest on the highest balance, add the interest to the account balance, and set the highest balance to 0.

7. Add a menu option to the `MyATM` application to calculate interest for both accounts.

8. Document the updated `Account` class so that the javadoc tool will produce the correct API documentation.

9. Submit a **.jar** file containing the `MyATM` class and the `improved` Account class. Please remember to include source code.

Discussion

There are almost 250 students in this course. Barring cheating, you would not expect that every `Account` class which is submitted after this lab will have the same code. Yet one would expect that any of your colleague's `Account` class can be used with your application. What OOP principles ensure that this is so?