Object-Oriented Programming in Java
IFT 194: Lab 3

Brandon Doyle
bdoyle5@asu.edu
1215232174

Dr. Usha Jagannathan
Usha.Jagannathan@asu.edu

July 15, 2018

Summary

Prelab Exercises

1. Class constructors are special methods that are called when the `new` operator is used to create a new instance of a class. These methods typically set up attributes of the class.

   There are a few differences between regular methods and class constructors (special methods). First and foremost, constructors cannot have a return value, and neither is a return type specified in the method header. Secondly, constructors have the same name as the containing class. Thus, if I were to write a class `Animal`, this class' constructors would also have the name `Animal`.

2. Access or visibility modifiers, which include `public` and `private`, among others, are used to determine where a variable or method of a class may be accessed from. For example, a public method or field variable may be accessed from outside an instance, whereas in the latter case they may only be used from inside the instance.

   A programmer can decide if a variable should be public based on what that variable may store. Likewise, a programmer may make a method public if it plays a vital role in the class' interface to other classes. Private methods are typically used to provide support to public methods. We should also keep *encapsulation* in mind, which suggests that a class should not allow other classes to modify its state by "reaching in" and modifying a value. Rather, we should provide an interface, perhaps through a public method, that has the ability to modify its state. This being said, it is perfectly fine to define publicly-accessible constants as long as they are preceeded by `final`, declaring the value immutable.

   There are a lot of best-practices suggested throughout the text for this course. I completely support them, knowing that things can easily get out of hand as a code base becomes large enough that a single person can no longer maintain or extend it.

3. In this problem we consider a class that may represent a bank account.

   a. To hold information about an account balance, the name of the account holder, and an account number, I might define the following field variables.

   ```
   private double balance = 0.0;
   final private String accountHolder;
   final private int[] accountNumber;
   ```

   The `accountHolder` and `accountNumber` fields are declared `final` because I don't want them to be modifiable once the class is instantiated. Moreover, all variables are private, because I wouldn't want this information to be accessible by other classes or objects unless the proper indentification is provided or process completed.

   b. In this sub-problem, we're asked to write method headers for each of the examples provided.

      i. To withdraw a certain amount of money from the account – change the account balance and do not return a value.

      ```
      public void withdraw(double amount) { ... }
      ```

View the source of this document on GitHub.

ii. Deposit a certain amount into the account and do not return a value.

```
public void deposit(double amount) { ... }
```

iii. Get the current balance of the account.

```
public double getBalance(/* something to check credentials? */) { ... }
```

iv. Return a string with account information, including name, account number, and balance.

```
public String getAccountInfo(/* Again, check credentials? */) { ... }
```

v. Charge a $10 fee.

```
public void chargeFee(double amount) { ... }
```

vi. A constructor that requires the initial balance, name of the owner, and account number.

```
public BankAccount(double initialBalance, String owner, int[] acctNumber) { ... }
```

Bank Account Class

1. For this question, I've reproduced the code provided in the lab in Figure 1.

   a. Please see my implementation of `toString` in the aforementioned figure, which overrides `Object`'s default implementation. Also, I've used the `String` class' `format` method, which allows us to shorten this line somewhat, in addition to limiting the display of account balance to 2 decimal places.

   b. For the `chargeFee` method, I've actually used method overloading so that a default fee of $10 may be charged to an account if no arguments are supplied.

   c. See the aforementioned figure regarding changes to both `chargeFee` methods.

   d. See the aforementioned figure for my implementation of the `changeName` method. My implementation also requires the argument to have content.

2. See Figure 2 for my implementation of `ManageAccounts.java`. The program's output is as follows.

```
Joe's new balance: 600.0
New balance: 950.00
Sally's new balance: 950.0

Name: Sally
Acct Number: 0000000001
Balance: 940.00

Name: Joseph
Acct Number: 0000000002
Balance: 565.00
```

Tracking Grades

Band Booster Class

Representing Names

```java
package lab_3;

import java.util.Arrays;

public class Account
{
    private double _balance;
    private String _name;
    final private int[] _acctNumber;

    /**
     * Class constructor.
     *
     * @param startingBalance Starting balance of the account.
     * @param acctName        Name associated with the account.
     * @param acctNumber      Number of the account (unique).
     * @throws java.lang.Exception If account number is not a list of 10 digits.
     */
    public Account(double startingBalance, String acctName, int[] acctNumber)
        throws java.lang.Exception
    {
        this._balance = startingBalance;
        this._name = acctName;

        if (acctNumber.length != 10)
            throw new Exception("*** Error: Account number length must be 10 "
                    + "digits, received" + acctNumber.length);

        this._acctNumber = acctNumber;
    }

    /**
     * Class constructor that initializes the balance to 0.
     *
     * @param acctName        Name associated with the account.
     * @param acctNumber      Number of the account (unique).
     * @throws java.lang.Exception If account number is not a list of 10 digits.
     */
    public Account(String acctName, int[] acctNumber)
        throws java.lang.Exception
    {
        this(0, acctName, acctNumber);
    }

    /**
     * Withdraw an amount from the account.
     *
     * @param amount The amount to be withdrawn.
     */
    public void withdraw(double amount)
    {
        if (this._balance >= amount) {
            this._balance -= amount;
            System.out.println(String.format("New balance: %.2f", this._balance));
        } else {
            System.out.println("Insufficient funds");
        }
    }

    /**
     * Deposit some amount into the account.
     *
     * @param amount The amount to deposit.
     */
    public void deposit(double amount)
    {
        this._balance += amount;
    }

    /**
     * Get the balance currently contained within the account.
     *
     * @return The balance associated with this account (instance).
     */
```

```
    public double getBalance()
    {
        return this._balance;
    }

    /**
     * Return a string with a summary of the account's information.
     */
    @Override
    public String toString()
    {
        // Convert the int[] array storing the account number to a String
        String acctNumber = Arrays.toString(this._acctNumber)
                                  .replaceAll("\\[|\\]|,|\\s", "");

        // Create a nicely formatted String
        return String.format("Name: %s\nAcct Number: %s\nBalance: %.2f", this._name,
                             acctNumber, this._balance);
    }

    /**
     * Charge a fee to the account, ignoring overdraft.
     *
     * @param amount The amount charged to the account.
     */
    public double chargeFee(double amount)
    {
        this._balance -= amount;
        return this._balance;
    }

    /**
     * Default fee; overloads former method.
     */
    public double chargeFee()
    {
        // Deduct $10 from the account.
        return chargeFee(10);
    }

    /**
     * Change the name on the account.
     *
     * @param newName    Name we'd like to change the acctName to.
     * @throws Exception Thrown if the name string is empty.
     */
    public void changeName(String newName) throws Exception
    {
        if (newName == "")
            throw new Exception("newName cannot be empty");
        this._name = newName;
    }
}
```

Figure 1: Account.java

```java
package lab_3;

public class ManageAccounts
{
    /**
     * Test our Account implementation.
     *
     * @param args Not used.
     * @throws Exception Not important in this example.
     */
    public static void main(String[] args) throws Exception
    {
        // Create an account for Sally
        var acct1 = new Account(1000, "Sally", new int[] {0, 0, 0, 0, 0, 0, 0, 0, 0, 1});

        // Create an account for Joe
        var acct2 = new Account(500,  "Joe", new int[] {0, 0, 0, 0, 0, 0, 0, 0, 0, 2});

        // Deposit $100 into Joe's account
        acct2.deposit(100);
        System.out.println("Joe's new balance: " + acct2.getBalance());

        // Withdraw $50 from Sally's account
        acct1.withdraw(50);
        System.out.println("Sally's new balance: " + acct1.getBalance());

        // Charge fees to both accounts
        acct1.chargeFee();
        acct2.chargeFee(35);

        // Change the name on Joe's account to Joseph
        acct2.changeName("Joseph");

        // Print summaries of both accounts
        System.out.println();
        System.out.println(acct1);
        System.out.println();
        System.out.println(acct2);
    }
}
```

Figure 2: ManageAccounts.java