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

In this section we're tasked with writing a class that permits a teacher to keep track of grades attributed to each student. Please see Figure 3 and Figure 4 for my solution. See below for another example session.

```
Enter student's grade for test #1: 50
Enter student's grade for test #2: 75
The average for Mary is: 62.50

Enter student's grade for test #1: 75
Enter student's grade for test #2: 50
The average for Mike is: 62.50

Name: Mary
Test 1: 50.00
Test 2: 75.00

Name: Mike
Test 1: 75.00
Test 2: 50.00
```

Band Booster Class

In this exercise we're tasked with writing a class that maintains the state of a band booster and keeps track of band candy sales over 3 weeks. Please see Figure 5 for my solution. Below is another example session, demonstrating how my program works.

```
Please enter a name: Brandon
Would you like to enter a another booster? [y|n]: yes
Please enter a name: Alison
Would you like to enter a another booster? [y|n]: n

    Week 1

Enter a number of sales for Brandon for this week: 50
Enter a number of sales for Alison for this week: 52

    Week 2

Enter a number of sales for Brandon for this week: 30
Enter a number of sales for Alison for this week: 60

    Week 3

Enter a number of sales for Brandon for this week: 70
Enter a number of sales for Alison for this week: 55

Summary:

    Brandon: 150
    Alison: 167
```

Representing Names

In this section we're tasked with writing a class that stores a person's first, middle, and last names. Please see Figure 6 and Figure 7 for my solution. Also, below is a sample session.

```
Enter a name: Brandon James
*** Error: Please enter a First, Middle, and Last name, separated by space
Enter a name: Brandon James       Doyle
Enter a name: Brandon James Doyle

Brandon James Doyle
Doyle, Brandon James
BJD
17

Brandon James Doyle
Doyle, Brandon James
BJD
17

The names are equal
```

Conclusion

In this lab I came across a number of new challenges. For instance, I wanted to put constraints on incoming values accepted by methods in my classes (hence the `throws IllegalArgumentException`s). I'm also trying to decide what kind of semantics make sense while I'm designing a class. I try to keep in mind how my class may be used; e.g. how can I make the API more convenient? What kind of type should I use to represent this information (again, what limits do I want to put on the inputs)?

```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 IllegalArgumentException If account number is not a list of 10 digits
     *                                  or any of the input digits are (-).
     */
    public Account(double startingBalance, String acctName, int[] acctNumber)
        throws IllegalArgumentException
    {
        this._balance = startingBalance;
        this._name = acctName;

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

        for (int i : acctNumber)
            if (i < 0)
                throw new IllegalArgumentException(
                    "*** Error: Account number entries must all be (+)");

        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 IllegalArgumentException Thrown if the name string is empty.
     */
    public void changeName(String newName) throws Exception
    {
        if (newName == "")
            throw new IllegalArgumentException("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

```java
package lab_3;

import java.util.Scanner;
import java.util.InputMismatchException;

public class Student
{
    private String _name = "";
    private double _scoreTest1 = 0.0;
    private double _scoreTest2 = 0.0;

    /* My original plan was to implement the AutoCloseable interface on Student,
     * but I've since learned that calling Scanner.close in the first instance
     * actually closes the input stream System.in as well, which is a tricky side-
     * effect that I'm not fond of. Hence, to avoid adding Scanner as a parameter
     * to the constructor, I've decided to ignore closing the Scanner since this is
     * just for an assignment.
     *
     * https://stackoverflow.com/q/13042008/3928184
     */
    private Scanner _scnr;

    /**
     * Class constructor that initialize name and scores.
     *
     * @param name   Name of the student.
     * @param score1 First test score.
     * @param score2 Second test score.
     */
    public Student(String name, double score1, double score2)
    {
        this._name = name;
        this._scoreTest1 = score1;
        this._scoreTest2 = score2;
        this._scnr = new Scanner(System.in);
    }

    /**
     * Overloaded class constructor that sets name and sets up internal Scanner.
     *
     * @param name Name of the student.
     */
    public Student(String name)
    {
        this._name = name;
        this._scnr = new Scanner(System.in);
    }

    /**
     * Prompt the user for grades for tests 1 and 2.
     */
    public void inputGrades()
    {
        this._scoreTest1 = getGrades(1);
        this._scoreTest2 = getGrades(2);
    }

    /**
     * Support method to remove duplicate code.
     *
     * @return Input grade of the student.
     */
    private double getGrades(int testNumber)
    {
        double grade = 0.0;

        while (true)
        {
            System.out.print("Enter student's grade for test #" + testNumber + ": ");
            try {
                grade = this._scnr.nextDouble();
                break;
            } catch (InputMismatchException ex) {
                System.out.println("*** Error: Please enter a float");
                this._scnr.next();
            }
        }

        return grade;
    }
```

```
    /**
     * Print the student's average grade.
     *
     * @return The student's average.
     */
    public double getAverage()
    {
        double average = (this._scoreTest1 + this._scoreTest2) / 2;

        return average;
    }

    public String getName()
    {
        return this._name;
    }

    @Override
    public String toString()
    {
        return "Name: " + this._name
                + String.format("\nTest 1: %.2f", this._scoreTest1)
                + String.format("\nTest 2: %.2f", this._scoreTest2);
    }
}
```

Figure 3: Student.java

```
package lab_3;

public class Grades
{
    public static void main(String[] args) throws Exception
    {
        var student1 = new Student("Mary");
        student1.inputGrades();
        System.out.println(String.format("The average for Mary is: %.2f",
                student1.getAverage()));

        System.out.println();

        var student2 = new Student("Mike");
        student2.inputGrades();
        System.out.println(String.format("The average for Mike is: %.2f",
                student2.getAverage()));

        System.out.println("\n" + student1);
        System.out.println("\n" + student2);
    }
}
```

Figure 4: Grades.java

```java
package lab_3;

import java.util.Scanner;
import java.util.ArrayList;
import java.util.InputMismatchException;

public class BandBooster
{
    private String _name;
    private int _boxesSold;

    /**
     * Class constructor.
     *
     * @param name  Name of the band booster.
     * @throws IllegalArgumentException  Thrown if the name is an empty String.
     */
    public BandBooster(String name) throws IllegalArgumentException
    {
        if (name.length() < 1) {
            throw new IllegalArgumentException("Please enter a valid name");
        }

        this._name = name;
        this._boxesSold = 0;
    }

    public String getName()
    {
        return this._name;
    }

    /**
     * Update the number of sales at the band booster.
     *
     * @param additionalBoxes Number of additional boxes sold.
     */
    public void updateSales(int additionalBoxes)
    {
        this._boxesSold += additionalBoxes;
    }

    @Override
    public String toString()
    {
        return String.format("%s: %d", this._name, this._boxesSold);
    }

    /**
     * Create instances of the BandBooster class and track sales for 3 weeks.
     *
     * @param args Not used.
     * @throws IllegalArgumentException Will not be thrown here.
     */
    public static void main(String[] args) throws IllegalArgumentException
    {
        try (var scnr = new Scanner(System.in)) {
            var boosters = new ArrayList<BandBooster>();

            // Enter names
            do {
                boosters.add(new BandBooster(getName(scnr)));
            } while (enterNames(scnr));

            // Loop for 3 weeks
            for (int i = 1; i <= 3; ++i) {
                System.out.println("\n    Week " + i + "\n");

                // Enter data for each booster
                for (BandBooster booster : boosters) {
                    booster.updateSales(getSale(scnr, booster.getName()));
                }
            }

            // Print information about each booster to the console
            System.out.println("\nSummary: \n");
            for (BandBooster booster : boosters)
                System.out.println("    " + booster);
        }
    }
```

```
    /**
     * Determine if the user would like to enter another name.
     *
     * @param scnr The scanner instance to be used for parsing input.
     */
    private static boolean enterNames(Scanner scnr)
    {
        System.out.print("Would you like to enter a another booster? [y|n]: ");
        String input = scnr.nextLine();
        return !input.matches("^[^Yy].*");
    }

    /**
     * Get the name of a band booster.
     *
     * @param scnr The Scanner instance to be used for parsing input.
     * @return A name.
     */
    private static String getName(Scanner scnr)
    {
        String name;

        while (true)
        {
            System.out.print("Please enter a name: ");
            name = scnr.nextLine();
            if (name.length() != 0)
                break;
        }

        return name;
    }

    /**
     * Get a number of sales from the user.
     *
     * @param scnr The Scanner instance to be used for parsing input.
     * @return A number of sales.
     */
    private static int getSale(Scanner scnr, String name)
    {
        int sales;

        while (true)
        {
            System.out.print("Enter a number of sales for " + name
                    +  " for this week: ");
            try {
                sales = scnr.nextInt();
                break;
            } catch (InputMismatchException ex) {
                System.out.println("*** Error: Please enter an integer");
                scnr.next();
            }
        }

        return sales;
    }
}
```

Figure 5: BandBooster.java

```java
package lab_3;

public class Name
{
    private String _first, _middle, _last;

    public Name(String first, String middle, String last)
            throws IllegalArgumentException
    {
        if (first.length() == 0 || middle.length() == 0 || last.length() == 0)
            throw new IllegalArgumentException("Must provide valid names");
        this._first = first;
        this._middle = middle;
        this._last = last;
    }

    public String getFirst()
    {
        return this._first;
    }

    public String getMiddle()
    {
        return this._middle;
    }

    public String getLast()
    {
        return this._last;
    }

    public String getFirstMiddleLast()
    {
        return this._first + " " + this._middle + " " + this._last;
    }

    /**
     * Get the name in ‘Last, First Middle‘ format.
     *
     * @return A string containing the name in the specified format.
     */
    public String getLastFirstMiddle()
    {
        return this._last + ", " + this._first + " " + this._middle;
    }

    public boolean equals(Name other)
    {
        // Thanks for suggesting the ‘String.equalsIgnoreCase‘ method!
        return this._first.equalsIgnoreCase(other.getFirst()) &&
               this._middle.equalsIgnoreCase(other.getMiddle()) &&
               this._last.equalsIgnoreCase(other.getLast());
    }

    /**
     * Get the name’s initials.
     *
     * @return A string containing the starting letters of each portion of the name.
     */
    public String initials()
    {
        // Why shouldn’t we use String.charAt?
        return this._first.substring(0, 1).toUpperCase() +
               this._middle.substring(0, 1).toUpperCase() +
               this._last.substring(0, 1).toUpperCase();
    }

    /**
     * Get the number of characters in the name.
     *
     * @return Number of characters.
     */
    public int length()
    {
        return this._first.length() + this._middle.length() + this._last.length();
    }
}
```

Figure 6: Name.java

```java
package lab_3;

import java.util.Scanner;
import java.util.ArrayList;

public class TestNames
{
    public static void main(String[] args)
    {
        try (var scnr = new Scanner(System.in)) {
            var names = new ArrayList<Name>();

            // Get each name
            String[] fstName = getName(scnr);
            String[] sndName = getName(scnr);

            // Create each name
            names.add(new Name(fstName[0], fstName[1], fstName[2]));
            names.add(new Name(sndName[0], sndName[1], sndName[2]));

            // Print info about each name
            for (Name name : names)
            {
                System.out.println();
                System.out.println(name.getFirstMiddleLast());
                System.out.println(name.getLastFirstMiddle());
                System.out.println(name.initials());
                System.out.println(name.length());
            }

            if (names.get(0).equals(names.get(1)))
                System.out.println("\nThe names are equal");
            else
                System.out.println("\nThe names are different");
        }
    }

    /**
     * Get a valid name from the user for creating a Name object. This must
     * include a First, Middle, and Last name, separated by spaces.
     *
     * @param scnr Scanner instance for parsing input data.
     * @return A valid name.
     */
    public static String[] getName(Scanner scnr)
    {
        String[] nameParts;

        while (true)
        {
            System.out.print("Enter a name: ");
            nameParts = scnr.nextLine().split("\\s+");
            if (nameParts.length != 3) {
                System.out.println("*** Error: Please enter a First, Middle, and"
                        + " Last name, separated by space");
                continue;
            } else {
                break;
            }
        }

        return nameParts;
    }
}
```

Figure 7: TestNames.java