CS0445 – Data Structures

Fall 2017

Lab #1

Lab Overview

We have covered in class what an Abstract Data Type (ADT) is, that it is implemented using one or more data structures, and that it is used in an algorithm to satisfy a set of operations that the algorithm needs. In order to use and implement ADTs in Java, we would need to write Java classes. In order to write a class, you would need either to:

- Write it completely from scratch,
- Compose it using existing classes, or
- Inherit it from an existing class.

In this lab we will write a Student class by composing it from an existing Name class. We will also write a CollegeStudent class by inheriting it from the Student class. Along the way, we will be compiling our programs in the command-line, writing Java packages, and running our Java classes from the command-line as well.

Java Packages

We will create a Java package for our code for Lab 1. Let's name this package Lab1. We start by creating a folder named Lab1 (the same as the package name). Inside the Lab1 folder, we will create our Java source-code files, one for each class.

The Name class

Let's start by writing the Name class, which contains two strings, firstName and lastName. So, let's open a text file using your favorite text editor, name the file Name.java and type the following code inside. Note the javadoc tags used. For a description of javadocs and its tags, please refer to Appendix G in the textbook. The class Name has a default constructor, a parameterized constructor and setters and getters for its two instance variables.

```
package Lab1;

/**     A class to hold first name and last name of a person

   <p>Last changed: Sept. 5, 2017

   <p>CS0445 Fall 2017, Lab 1

   @author Sherif Khattab

   @version 0.1

*/

public class Name

{
```

```
private String firstName;
private String lastName;

/** Sets first name and last name to the empty string
*/
public Name()
{
  firstName = "";
  lastName = "";
}

/** Parameterized constructor
 @param firstName a string for the first name
 @param lastName a string for the last name
*/
public Name(String firstName, String lastName)
{
  this.firstName = firstName;
  this.lastName = lastName;
}

/** Setter for the firstName instance variable
 @param firstName a string for the first name
*/
public void setFirstName(String firstName)
```

```
{
   this.firstName = firstName;
}

/** Setter for the lastName instance variable
 @param lastName a string for the last name
*/
public void setLastName(String lastName)
{
   this.lastName = lastName;
}

/** Getter for the firstName instance variable
 @return the string for the first name
*/
public String getFirstName()
{
 return firstName;
}

/** Getter for the lastName instance variable
 @return the string for the last name
*/
public String getLastName()
{
   return lastName;
  }
}
```

The Main class

In order to have a program, we need to write a class with the main method. We call this class the Main class in our example. So, open up a next text document in the Lab1 folder and name it Main.java. Write the following code. Note that we need to import the Scanner class within the package java.util in order to be able to use the class within our code to read from the keyboard. Note also the use of the setters and getters of the Name class.

```
package Lab1;

import java.util.Scanner;


/**     A class to read and print a person's name
   <p>Last changed: Sept. 5, 2017
   <p>CS0445 Fall 2017, Lab 1
   @author Sherif Khattab
   @version 0.1
*/
public class Main
{
  /** The entry-point to the program
   @param args an array of strings holding the command-line arguments
  */
  public static void main(String[] args)
  {
   Name name = new Name();
```

```
    Scanner keyboard = new Scanner(System.in);
    System.out.print("Welcome to the program! Please enter your first name: ");
    name.setFirstName(keyboard.nextLine());
    System.out.print("Please enter your last name: ");
    name.setLastName(keyboard.nextLine());
    System.out.println("You entered: " + name.getFirstName() + " " + name.getLastName());
   }
  }
```

Compiling our program

To run our program, we need first to compile the Java code. Change directory into the Lab1 folder and type the following.

```
javac *.java
```

This would compile the two java files we have therein and generate two class files (if no errors were encountered by the compiler).

Running our program

To run our program, we would run the command java and give it the full class name of the Main class. The full name includes the package name, the dot, and the class Name. This has to be done at the directory above the package's directory. So, first we will have to go up one folder.

```
cd ..
java Lab1.Main
```

Generating documentation

We can use the Javadocs tool to automatically generate HTML documentation of our code. To do so we run the command

```
javadoc  -d docs Lab1
```

Here, the "-d" option allows us to specify the name of the folder to contain the automatically-generated html files. Lab1 is the folder containing our code. To open the generated documentation, we would point the web browser to the file index.html under the newly-generated folder docs.

Building the Student Class by Composition

Having written the Name class, we move on into writing the Student class. A student would have a first name, a last name, and an ID. Instead of writing the first name and last name code from scratch, we will use the Name class to handle that. This is called composition. We are basically composing the Student class from the already-existing Name class. The composition models a has-a relationship. That is to say a student has a name. So, let's have a look at the Student class. Note that we are defining an object of class Name as an instance variable inside the Student class and we are using methods of the Name class, for example to set and get the first and last name.

```
package Lab1;
/**    A class to hold student info
   <p>Last changed: Sept. 5, 2017
   <p>CS0445 Fall 2017, Lab 1
   @author Sherif Khattab
   @version 0.1
*/
public class Student
{
  private Name name;
  private String ID;

  /** Sets first name, last name, and ID to the empty string


  */
  public Student()
  {
   name = new Name();
```

```
 ID = "";
}

/** Parameterized constructor
 @param firstName a string for the first name
 @param lastName a string for the last name
*/
public Student(String firstName, String lastName)
{
   name.setFirstName(firstName);
   name.setLastName(lastName);
}

/** Setter for the firstName instance variable
 @param firstName a string for the first name
*/
public void setFirstName(String firstName)
{
   name.setFirstName(firstName);
}

/** Setter for the lastName instance variable
 @param lastName a string for the last name
*/
public void setLastName(String lastName)
```

```
    {
      name.setLastName(lastName);
    }

    /** Getter for the firstName instance variable
     @return the string for the first name
    */
    public String getFirstName()
    {
     return name.getFirstName();
    }

    /** Getter for the lastName instance variable
     @return the string for the last name
    */
    public String getLastName()
    {
     return name.getLastName();
    }
  }
```

Exercise

Modify the Main class so that it uses the Student class to read a student's first name, last name, and ID from the keyboard, then outputs what the user has typed in.

Building the CollegeStudent Class by Inheritance

Let's assume that we want to create a class to hold the information of a college student. A college student would need an extra field for the student standing (Freshman, Sophomore, Junior, or Senior). Instead of creating the CollegeStudent class from scratch, we can use the already-existing Student class. We will use class inheritance for this task. Inheritance models a is-a relationship. That is to say, a college student is a student. Let's have a look at the code for the CollegeStudent class noting the use of the keyword extends. Note also the use of the keyword super to call the constructor of the superclass (or parent class), which is Student in this case. The super keyword is used to call the default and parametrized constructors. Note also that we don't need to write setters and getters for the first name, last name, and ID. These are already inherited from the Student class.

```
package Lab1;

/**   A class to hold college student info. A college student is a student and has an extra field for the
standing (Freshman, Sophomore, Junior, or Senior).

  <p>Last changed: Sept. 5, 2017

  <p>CS0445 Fall 2017, Lab 1

  @author Sherif Khattab

  @version 0.1

*/

public class CollegeStudent extends Student

{

  private String standing;


  /** Sets first name, last name, ID, and standing to the empty string

  */

  public CollegeStudent()

  {

   super();

   standing = "";

  }
```

```
/** Parameterized constructor
 @param firstName a string for the first name
 @param lastName a string for the last name
 @param ID a string for the student ID
 @param standing a string for the student standing
*/
public CollegeStudent(String firstName, String lastName, String ID, String standing)
{
 super(firstName, lastName, ID);
  this.standing = standing;
}

/** Setter for the standing instance variable
 @param standing a string for the standing
*/
public void setStanding(String standing)
{
   this.standing = standing;
}

/** Getter for the standing instance variable
 @return the string for the student standing
*/
public String getStanding()
```

```
        {
         return standing;
        }
      }
```

Exercise

Instead of declaring the standing instance variable as a string, define an enumerated type for the possible values of student standing (Freshman, Sophomore, Junior, and Senior) and make standing a variable of the enumerated type.

Exercise

Modify the Main class so that it uses the CollegeStudent class to read a student's first name, last name, ID, and standing from the keyboard, then outputs what the user has typed in.