ELEC 279 - Winter 2020
Introduction to Object-Oriented Programming
Lab 4 - Week 9

Interfaces in JAVA

Welcome to ELEC 279 lab session where you will learn OOP by practicing. Same as before, throughout this lab session you will be working in the *pair programming model* [1] - usually two programmers per workbench and in rare case, a group of three programmers will be permitted at the approval of a Teaching Assistants coordinating the lab. Please arrive early to the lab and get settled to start working on the lab - arriving 15 minutes after the start of a lab is considered late and could affect your participation point for the lab.

Pair Programming Model: In order to promote an equal learning environment, we will be evaluating your participation and results in the labs, which means every member of your group must participate. In the pair programming model, each of the two team members can either be the *driver* or the *navigator* and their roles are:

- *the driver:* the person typing at the keyboard
- *the navigator:* the person with the "eagle eyes," watching for mistakes and/or typos.

One of you can only assume one role at a time, and you can take turns to interchange the role during each lab session. An example is, for Task 1, one person assumes the *driver* role while the other person serves as the *navigator*. And for Task 2, the *navigator* for Task 1 becomes the *driver*, and vice versa. This is the model we will adopt for this course throughout the lab sessions in this semester.

Lab Grading and Assessment: Note that your TAs are here in this Lab to help you learn. Feel free to ask questions as the TAs will be going round to make sure all members of the group are participating. For each Lab assessment, YOU MUST demonstrate your results for each **Task** to get credit for that **Task** - do not wait until the end to show all the tasks, demonstrate each task to the TAs as you progress. Remember, the total Labs is worth 15% of your final grade.

1 Introduction

In this lab you will learn about Interfaces. An interface at it's core is a group of methods with empty bodies. Interfaces specify a set of methods that any class that implements the interface must have.

- Interfaces contain method headings or/and constant definitions only.
- Interfaces contain no instance variables nor any complete method definitions.

Java does not allow for multiple inheritance and thus interfaces can approximate this.

[1] https://tinyurl.com/ydf9mwos

2 Getting Started in eclipse

This lab has included test code that you must satisfy to get full marks. There are a couple steps that must be completed first to set up the lab in eclipse. Like last lab, **let one member be the driver and the other the navigator.**

1. Open eclipse
2. Start by creating a project called **"Lab4"**
3. Select **File→New→Java Project** to create a new project
4. Name the project **"Lab4"**
5. Click finish

Now we will add classes to our project.

1. Now we must create the main class for this project
2. Select **File→New→Class** to add a new class
3. Under **Name** enter **"Lab4"**
4. Click finish
5. Copy the text from **"Lab4.java"** on onQ to your newly created class
6. Repeat this for **"Employee.java"**, **"HourlyEmployee.java"**, and **"Invoice.java"**

We must also create a new interface.

1. Select **File→New→Interface** to create a new interface
2. Name the interface **"PayAble"**
3. Click finish

3 The Comparable Interface

We will start by implementing the Comparable<T> interface in our class HourlyEmployee so that we can easily sort HourlyEmployee objects by employees' name. The "<T>" indicates that it uses generic types and allows the interface to change the method types to be suitable for the class that implements it. You have learned and will learn more about generics in class. So far, just follow the instructions here to finish this lab.

1. In the class definition for HourlyEmployee add **Comparable<HourlyEmployee>** to the list of implemented interfaces
2. The definition should now read **"public class HourlyEmployee extends Employee implements Cloneable, Comparable<HourlyEmployee>"**

3. Now add the method **"public int compareTo(HourlyEmployee anotherWorker)"**

4. Complete the method by following what you learned in the class. Note that the compareTo method here has one parameter (anotherWorker).

   - In general, the method should return a negative number if the calling object (an hourly employee) "comes before" the parameter (i.e., anotherWorker), return a zero if the calling object "equals" the parameter, and return a positive number if the calling object "comes after" the parameter.

   - Specifically in this lab, let's define "comes before" as if the calling object's (an hourly employee) name is alphabetically before the anotherWorker's name, define "equals" as if they have exactly same name, and define "comes after" as if the calling object has a name that is alphabetically after anotherWorker's name. Given this definition, the implementation of compareTo in HourlyEmployee is actually very simple and can be implemented with one line of code. (Hint: you may invoke the compareTo method defined in the String class.)

5. Once complete, run Part 1 of the main method in Lab4 and comment out the rest

At this time it is recommended that you **switch from navigator to driver and vise-versa.** We will now add the interface Comparable<T> to the Invoice class.

1. In the class definition for Invoice add **implements Comparable<Invoice>** after the class declaration

2. The definition should now read **"public class Invoice implements Comparable<Invoice>"**

3. Now add the method **"public int compareTo(Invoice bill)"**

4. Complete the method so that an array of Invoice objects can be sorted in ascending order of amounts. (Hint: Unlike the compareTo above in which you can invoke the compareTo method defined in String, here you need to write some code with if-statement.)

5. Once complete, run Part 1 and 2 of the main method in Lab4 and comment out the rest

Stop Here Ask the TAs to evaluate your code's performance to get graded for this portion of the lab.(1 mark)

4 Creating Your Own Interface

Now go to the PayAble interface we made at the beginning of the lab. We will add two method headings needed for the PayAble interface. The interface will be needed for different types of objects such bills, invoices, and employees. In this lab, you will implement this interface in the HourlyEmployee and Invoice class, after we define this interface first. To define this interface:

1. Add the following two method headings into the PayAble interface.
2. **"public Double amountToPay()"**
3. **"public void printPayment()"**
4. Now you have defined the interface

Below we will implement PayAble in the Invoice class first.

1. In the definition of Invoice add **PayAble** to the list of interfaces it implements.
2. Now in the Invoice class, define the two methods specified in the PayAble interface
3. Complete the two methods so that printPayment prints **"Payment information for an invoice. Company name: [name-of-company]; payment [amount]."**, e.g., "Payment information for an invoice. Company name: Google; payment 200."
4. Note that the printPayment should invoke amountToPay, and amountToPay just returns the value of instance variable, amount, which is defined in Invoice.
5. Once complete, run Part 1, 2 and 3 of the main method in Lab4 and comment out the rest

Stop Here Ask the TAs to evaluate your code's performance to get graded for this portion of the lab.(1 mark)

At this time it is recommended that you **switch from navigator to driver and vise-versa.** Now we will implement PayAble in HourlyEmployee.

1. In the class definition for HourlyEmployee add **PayAble** to the list of interfaces that HourlyEmployee implements
2. Now define the two methods, printPayment and amountToPay, in the HourlyEmployee class. amountToPay here should return hours*wageRate.
3. Complete the methods so that it prints **"Payment information for an hourly employee. Employee name: [name-of-employee]; payment: [Total-Wages-For-All-Hours-Worked]"**, e.g.,"Payment information for an hourly employee. Employee name: Tina; payment: 100"
4. Ensure you use amountToPay() in printPayment()
5. Once complete, run Part 1, 2, 3 and 4 of the main method in Lab4

Stop Here Ask the TAs to evaluate your code's performance to get graded for this portion of the lab.(1 mark)