CSCE 156 – Lab07: Abstract Classes, Interfaces and Generics

0. Prior to the Laboratory

1. Review the laboratory handout.
2. Read about abstract methods and abstract classes in Java
http://download.oracle.com/javase/tutorial/java/IandI/abstract.html
3. Read about interfaces in Java:
http://download.oracle.com/javase/tutorial/java/IandI/createinterface.html
4. Read about the Generics in Java:
https://docs.oracle.com/javase/tutorial/java/generics/

1. From the GIT repository:

1. Go to File -> Import -> GIT -> Projects from GIT.
(Note: if the "GIT" import source is not listed, you might have to install a GIT plugin for Eclipse)
2. Select "Clone URI" and enter the repository URI as the following:
https://git.unl.edu/csce-156/Lab07

2. Objectives

Following the lab, you should be able to:

- Understand and use Abstract classes in Java
- Understand and use Interfaces in Java
- Use the concepts of Abstract classes and interfaces to efficiently and effectively solve problems
- Use Java generics

3. Problem Overview

A brief description of Abstract Classes and Interfaces in Java is given below:

- Abstract Classes - Java allows you to specify that a class is **abstract** using the keyword ***abstract***. In an abstract class, you can define not only normal methods (which you provide a "default" implementation for) but you can also define abstract methods: methods that you do not need to provide an implementation for. Instead, it is the responsibility of non-abstract subclasses to

provide an implementation. In addition, if a class is abstract, you are prevented from instantiating any instances of it.

- Interfaces – Java allows you to define interfaces, which are essentially pure abstract classes. Interfaces specify the methods that a class must provide in order to implement the interface. Java allows you to define an **interface** that specifies the public interface (methods) of a class. Classes can then be defined to implement an interface using the ***implements*** keyword. One major advantage of interfaces is that it does not lock your classes into a rigid hierarchy; however objects that implement an interface can still be considered to have the *is-a* relationship. In addition, interfaces can be used to achieve multiple-inheritance in Java as classes can implement more than one interface.
- Generics – Generics enable *types* (classes and interfaces) to be parameters when defining classes, interfaces and methods. These type parameters provide a way to re-use the same code with different inputs. By using generics, one can implement generic algorithms that work on collections of different types, can be customized, and are type safe and easier to read. They allow stronger type checks at compile time and hence add stability to code by making more of your bugs detectable at compile time.

Activity 1: Program Design and Abstraction

In this activity you will design a Java program that calculates a basic weekly payroll report of a Company named "*Notion Inc.*" using the concepts of Abstract class and Interface. Every employee in the company has an employee ID, a name (first and last), and a title. Further, there are two types of employees:

- Permanent employees - Permanent employees have a base weekly salary and may have overtime hours. For any overtime, they are subject to overtime payments, which are calculated as hours, multiplied by overtime rates.
- Temporary employees - Temporary employees are hourly workers and have a per-hour pay rate. Their weekly pay is calculated as payrate multiplied by the weekly total of the number of hours they worked.

The structure and hierarchies of classes for this program is a design decision left to the students to decide. But you are required to use some sort of abstract class/method to accomplish this. Some of the points to note are-

- Identify the different classes necessary to model each of the different types of employees in the problem statement
- Identify the relationship between these classes
- Identify the state (variables) that are common to these classes and the state that distinguishes them—where does each of these pieces of data belong?

- Identify the behavior (methods) that is common to each of these classes—what are methods that should be *abstract*? How should subclasses provide specialized behavior?
- Some state/behavior may be common to several classes—is there an opportunity to define an intermediate class?

The driver class "`PayrollDemo.java`" is provided for you. It creates objects of each type of employees and prints their names and salaries. A partially implemented abstract class "`Employee.java`" is also given as a starting point.

Activity 2: Adding an interface

In this activity you will complete the implementations of Employee and its subclasses (if any!). The company sells various software and hardware products. Each product has a name, type, cost, and count of items sold in the week. Create new class(es) for the Products. Every week, the company computes the productivity of all its employees and its products. Additionally, it also sends all information of its products in JSON format to its auditors. Both Employee and Products must implement the following two interfaces:

- Serializable – this interface has a method called "`serializeToJSON`" which takes two arguments, an object of the class that implements the Serializable interface and a file object to write to. This method needs to parse and export the object (the first argument) into JSON format and write it to the file (the second argument).
  Hint: you may find using the json library useful.

- CompanyPortfolio – this interface has a method "`computeProductivity`", its implementation for Employee class(es) and Product class(es) are left for the students. However, your implementation should do what it is "supposed to do" as the name of the method suggests. An example could be:
  - Employee: Salary divided by number of hours in the week.
  - Product: Cost*(Count of Items Sold) /number of hours in the week.

The driver class "`PortfolioDemo`" is provided for you.

Useful Tip: *See if you can use Eclipse to auto-generate a lot of methods (e.g., getters,setters,constructors and toString, etc.) and save time.*

Activity 3: Using Java Generics

In this activity you will investigate the advantages of using generics in Java. A generic class template "`CompanyList`" is provided with a generic ArrayList which can be used to store employees and products. This type of the ArrayList is decided when an object of the class is instantiated.

Lab Handout: Abstract Classes and Interfaces- Summer 2018

In the `GenericsDemo` class, instantiate two company lists for employees, one for a specific employee type (i.e. generic) and second one without any type (non-generic). Then add two different types of employees to both. Investigate what happens. Try to print the list using the "print" method in the `CompanyList` class. Also, try to retrieve the employees that you've added from each company list.