CSCE 156 – Lab05: Inheritance

Handout

0. Prior to the Laboratory

1. Review the laboratory handout.
2. Read the following tutorial on inheritance in Java
   http://download.oracle.com/javase/tutorial/java/IandI/subclasses.html
3. Read about abstract methods and abstract classes in Java
   http://download.oracle.com/javase/tutorial/java/IandI/abstract.html
4. Read about interfaces in Java:
   http://download.oracle.com/javase/tutorial/java/IandI/createinterface.html

1. Lab Objectives & Topics

Upon completion of this lab you should be able to:

- Understand Inheritance and design classes and subclasses in Java
- Understand and use interfaces and abstract classes in Java
- Understand and use the `extends` keyword

2. Problem Overview

Object Oriented Programming allows you to define a hierarchy of objects through *inheritance*. Inheritance promotes code reuse and semantic relations between objects. Subclasses provide *specialization* of behavior by allowing you to *override* object methods while preserving common functionality (*generalization*) defined in the super-class.

Java is a class-based object-oriented programming language. It facilitates inheritance through subclassing by using the keyword `extends`. If class B *extends* class A, B is said to be a subclass of A (A is a superclass of B). Instances of class B are also instances of class A, defining an "*is-a*" relation between them.

Java also provides two related mechanisms related to sub-classing.

- 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.

Program

You will explore these concepts by completing a Java program that simulates a basic *weekly* payroll reporting system for the Cinco Corporation. Every employee has an employee ID, a name (first and last), and a title. Further, there are two types of employees:

- Salaried employees – Salaried employees have a base annual salary, which is subject to a 20% income tax rate (state, federal and FICA). In addition, each salaried employee receives a $100 post-tax benefits allowance.
- Hourly employees – Hourly employees have a per-hour pay rate along with a weekly total of the number of hours they worked. Hourly employees do not receive any benefit allowance. Further, there are two types of hourly employees.
    - Staff employees are directly employed by Cinco and are subject to a 15% income tax rate.
    - Temporary employees are not directly employed by Cinco, but instead are contracted through a third-party temp agency who is responsible for collecting taxes (thus no taxes are taken from their gross pay).

Employee data is stored in a flat data file and the basic parsing has been provided. However, you will need to design and implement Java classes to support and model this payroll system.

Importing Your Project

A Java project has been started to implement a basic payroll system. Import this project into your Eclipse IDE by following either of the steps below:

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/Lab05

Activity 1: Class Design & Inheritance

As indicated, you have been provided a partially completed `PayrollReport` program that parses the data file (in `data/employee.dat`) and creates instances of an `Employee`. However, the `Employee` class is empty. You will need to implement this class and any relevant subclass(es) to complete the program.

- 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 do each of these pieces of data belong?
- Identify the behavior (methods) that are common to each of these classes—what are methods that should be in the superclass? How should subclasses provide specialized behavior?
- Some state/behavior may be common to several classes—is there an opportunity to define an intermediate class?
- If you need more guidance, consult the UML diagram below on one possible design.
- To check your work, a text file containing the expected output has been provided in the project (see `output/expectedOutput.txt`)

Eclipse Tip: Many of the common programming tasks when dealing with objects can be automated by your IDE. For example: once you have designed the state (variables) of your class, you can automatically generate the boilerplate getters, setters, and constructors: when focused on your class, click Source > Generate Getters and Setters or Generate Constructors using Fields.

Java Note: In a subclass, you *must* invoke a constructor in the superclass *and* it must be done first. This rule is so that classes conform to the is-a relationship. Invoking a super class's constructor is achieved by using the `super` keyword. If there are multiple constructors, you may invoke any one of them. Note that if a constructor does not explicitly invoke a superclass constructor, the Java compiler automatically inserts a call to the no-argument constructor of the superclass. If the superclass does not have a no-argument constructor, you will get a compile-time error. Object does have such a constructor, so if Object is the only superclass, there is no problem.

Activity 2: Abstractions

Now that you have a well designed, functional implementation we will improve on the design by identifying potential abstractions that can be made. In particular, start by making the `Employee` class `abstract`. If you had a good design, then nothing should break; the model did not have any generic employee—all employees were of a specific type. If something did break in your code, rethink your design and make the appropriate changes.

Further, identify one or more methods in the Employee class that could be made abstract. That is, are there any methods in the superclass where it would not be appropriate to have a "default" definition? Make these methods abstract and again make any appropriate changes to your design as necessary.

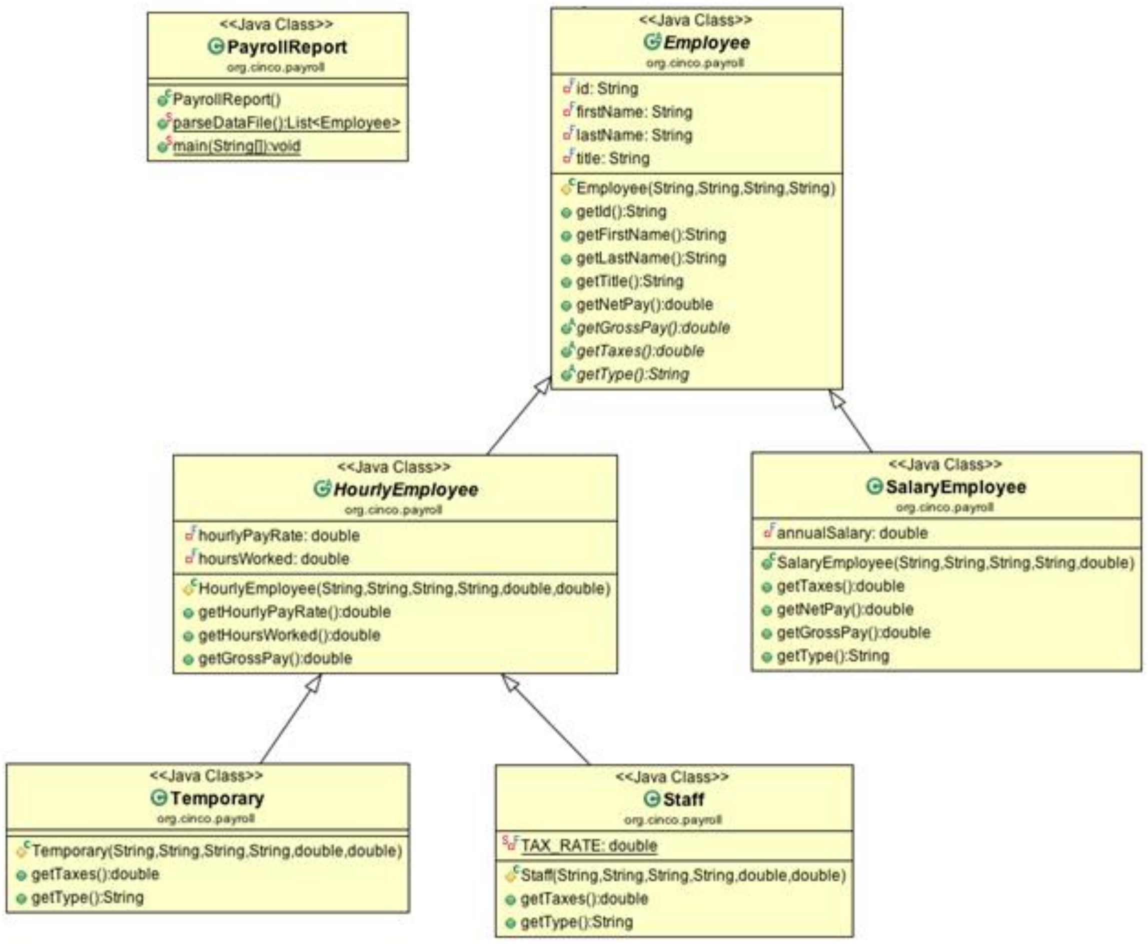

Figure 1 UML Diagram of a potential design for the Cinco Corporation payroll system

Advanced Activities (Optional)

1. The `PayrollReport` class uses an `ArrayList` to hold instances of `Employee` objects. When it generates the report, it does so in the order that the instances were parsed from the data file. Change this so that the payroll report prints in order of the total net pay in decreasing order. To do this, familiarize yourself with `Comparators` in Java by reading the following tutorial: http://docs.oracle.com/javase/tutorial/collections/interfaces/order.html
   Then, create a `Comparator` for `Employee` instances and use one (or both) of the following methods.
   a. Use the Comparator you created and call the `Collections.sort` method to sort the `ArrayList`

   b. Use the `Comparator` you created and change the `ArrayList` collection to a `TreeSet` instance (a `TreeSet` maintains an ordering as elements are added to it).

2. Unified Modeling Language (UML) is a common tool in Software Engineering that provides a visualization of the relationships between software components (subsystems, components, classes, workflows, use cases, etc.). Sometimes design of systems is done in UML and then tools can automatically generate Java (or other language) code conforming to the design. Conversely, UML diagrams (like that in Figure 1) can be automatically generated from an existing code base using various tools. In this exercise you will familiarize yourself with UML and use such a tool to generate a UML diagram for your design.
   a. Read the following tutorial on using UML for class diagrams: http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/
   b. Install an Eclipse plugin for UML and generate a UML diagram for your project. The choice is yours, but one (free) possibility is ObjectAid UML:
      - Installation instructions: http://www.objectaid.com/installation
      - Generate class diagram instructions: http://www.objectaid.com/class-diagram