Course: CSC 115 (J)

Lab 04: Abstraction: using Inheritance, and Interfaces

Objectives

- To understand the basics of information abstraction.
- Use inheritance to reduce redundancy in your code.
- Implement an interface we have provided to make sure your program meets the required specifications and help you hide the implementation details.

Part I – Background

Abstraction is the process by which data and programs are defined such that its semantics (meaning) is shown, while hiding away the implementation details. Abstraction tries to reduce and factor out details so that the programmer can focus on a few important concepts. Abstraction simplifies complex reality by modelling classes appropriate to the problem, and works at the most appropriate level of inheritance for a given aspect of the problem.

The *inheritance* mechanism in object-oriented programming can be used to define an abstract class as the common interface. The recommendation that programmers use abstractions whenever suitable in order to avoid duplication (usually of code) is known as the abstraction principle. The requirement that a programming language provide suitable abstractions is also called the abstraction principle.

For more details, visit http://en.wikipedia.org/wiki/Abstraction_(computer_science)

Part II – Inheritance

You will be writing a program that stores a list of people associated with a course. This first part will involve creating classes to store information on students, and TAs, using inheritance to cut down on redundancy in your code.

1. Download and examine *Person.java* from connex. It includes the following features:
    - Person is an *abstract* class. This means that it has no constructor, and a user cannot directly create an instance of a Person.
    - A Person has a *name* and an *email* address.
    - *Getters* and *setter* methods for its instance variables.
    - A *toString()* method that returns a person's information as a single String with the following format:
        - name, email (for instance David Clark, cdavid@somecompany.com)

2. Download and examine *Student.java* from connex. It includes the following features:
    - A Student is a person, so the Student class inherits from/*extends* the Person class.
    - A Student has a numerical *student number* and a *numerical grade*.
    - A constructor with the following signature:
        - `public Student(String studentName, String studentEmail, int studentNumber, int grade)`
    - *Getters* and *setter* methods for its instance variables.
    - A *toString()* method that returns the student's information as a single String with the following format:
        - Student: Ackbar, admiral@rebels.com, Student#: 54354353, Grade: 94
        - **Note** how the *super* keyword is used to access methods of a *superclass*/ancestor.
    - A main method that creates a Student object and tests its toString() method. Make sure to **run** Student.java before moving on to the next step.

3. Download and complete the TA class so that it includes the following features:
    - A TA should have an employee number and a salary, both stored as integers.
    - A TA should have a student number.
    - A constructor with the following signature:
        - `public TA(String taName, String taEmail, int taEmployeeNumber, int taSalary, int taStudentNumber)`
    - *Getters* and *setter* methods for its instance variables.
    - A *toString()* method that returns the TA's information as a single String with the following format:
        - TA: Ackbar, admiral@rebels.com, Employee#: 4325432, Salary: 10000, Student#: 432423
        - You can use the **super** keyword to access methods of a **superclass**/ancestor. Use super.toString() as part of your toString() method.
    - A main method that creates a TA object and tests its toString() method. Make sure to **run** TA.java before moving on to the next step.

Part III - Implementing an Interface

Download *Course.java* and *EnrollmentList.java* from connex. EnrollmentList is an interface that contains a list of functions that your program will need to include. This part will have you complete Course.java, such that it implements all the functionality specified in the EnrollmentList interface. Complete the Course class so that it includes the following features:

- A Course should have a name, an array of People associated with it, and an integer that stores the number of people in the course.

- A constructor with the following signature:
  - `public Course(String courseName, int capacity)`
- A *toString()* method that returns the course's information as a single String with the following format:
  - CSC 115 Class List
    ```
    TA: Ackbar, admiral@rebels.com, Employee#: 4325432,
    Salary: 10000, Student#: 432423
    Student: Ackbar, admiral@rebels.com, Student#: 54354353,
    Grade: 94
    ```
- The Course class should implement the *EnrollmentList* interface. This means that it must include all the methods described in the interface, otherwise you will get compiler errors.
- Start by making shells for each method in the interface -- copy their signatures, and return dummy values where needed to get the code to compile.
- Fill in the methods one by one, making sure to test each one before you move on to the next. Use the comments provided with the interface to figure out what each method is supposed to do.

Announcement: There will be no csc-115 labs next week (the week of Oct 08 – Oct 12). You may utilize the extra time studying for your mid-term exam.

Best of luck in your exams..!

Have fun