CSIT 150 Lab 4: CVS & Inheritance

Objectives

- Practice using interfaces
- Practice inheritance
- See how polymorphism works
- Use CVS to save your work to GitHub.

Special Requirements for this lab only: Each student must complete this lab individually. (You can work side by side with your partners.)

I. **https://education.github.com/pack** -- scroll down to the GitHub link, and click it.
II. Create a GitHub account.
III. **If you are doing this on your own computer, you may need to install Git, https://git-scm.com/downloads**

Background

In this assignment you will practice using interfaces, and inheritance. This is a good time to learn how to commit a project to a Version Control System (VCS) repository so that you can get some experience on VCSs. There are several version control systems, we will use GitHub in this lab because it is simple to configure and is already nicely integrated with IntelliJ without any additional installation steps, and it is used by most software developers around the world.

One advantage of using version control for all projects is that your project will be stored in a code repository in your account, as well as within the IntelliJ workspace on the machine or machines where you are doing your work. This replication provides some safety in the event of a hardware failure. There are several other advantages to using version control including the ability to back out of changes that you make (e.g., if you write some code that breaks your project).

Programming Practice

Follow the detail as provided below:

1. Within IntelliJ project, create a package for lab 4.
2. In the package, add an interface named `Rules`, which defines two methods: `getName`() and `getSalary`().
3. Create a class named `Employee`.
   - It implements the interface of Rules.
   - It has two data members: `name` and `salary`.
   - It has a constructor that accepts data and initializes the name and salary.
   - It has a default constructor.
   - It has a `getName`() method that outputs the name of the Employee.
   - It has a `getSalary`() method that outputs the salary of the Employee.
   - It has a toString() method to output the Employee information.

4. Create a class named `Faculty`.
   - It inherits class `Employee`
   - It has an extra *private* field named `courses`, which is an array that holds the courses taught by this faculty member. (For each course, just store the course name as a String.)

- It has a constructor that accepts three parameters: `name`, `salary`, and `courses`. Notice that name and salary can be initialized using the constructor of the superclass. Notice that the input parameter `courses` is plural.
- It overrides the `getName`() method of `Employee`, and precedes the name with the word "*Professor",* similar to the `GraduateStudent` `getName`() method demonstrated in class.
- It has a public accessor `getCourses` that returns an array of courses. Be sure to return a copy.
- It has a public accessor `getCourseNames` that gets all of the course names as a string.
- It has a public mutator to the field `courses`. Be careful with the array size, and with storing copies. See the CD Collection.
- It has a toString() method to output the Faculty information.

5. Copy the class named Test.java from Blackboard into this package, and run it. This class just has a main method. In this method, it:
   - Instantiates a `Faculty` reference to a variable of type `Faculty`. Then it calls each of the methods described above, including `getName`(), `getSalary`(), and the accessor and mutator to field `courses` in class `Faculty` and see what happens.
   - Instantiates an `Employee` variable that references a new `Faculty`. Then it calls each of the methods described above, including `getName`(), `getSalary`(), and the accessor and mutator to field `courses` in class `Faculty` and see what happens.
   - Instantiates a `Rules` variable that references a new `Faculty`. Then it calls each of the methods described above, including `getName`(), `getSalary`(), and the accessor and mutator to field `courses` in class `Faculty` and see what happens.

6. Commit your project to GitHub, by following these steps:
   a. In IntelliJ File->Settings->Version Control, add your GitHub account information.

   b. From the command prompt (Menu: Accessories/Terminal), type:
   `VCS->Import into Version Control ->Share Project on Git Hub.`

   - **Anytime you work on your project**, after making changes that you wish to commit (usually after you make sure the code runs the way you want), you can then use **`VCS->Commit Changes.`**

   - **Anytime you want to work on the project on <u>any computer (with any operating system)</u> which doesn't have the project yet,** in IntelliJ, you can work on the project, by going to: VCS->Checkout from version control. Select it from your drop down list. (Be sure that IntelliJ has your GitHub account information in the settings.

   - Any time you wish to get the most up-to-date SAVED version of the project onto any computer, select VCS->update. Be careful – this overrides any local current changes you have made and pulls the saved version from the repository.