Carleton University
Department of Systems and Computer Engineering
SYSC 2004 - Object-Oriented Software Development - Winter 2015

Lab 9 - Classes and Interfaces, Overriding `equals` and `toString`

Objective

- To develop an inheritance hierarchy of classes that model different types employees in a company. The major differences between this week's lab and last week's are: (1) the `Employee` abstract class will implement an interface, (2) class `Employee` will override the `equals` and `toString` methods that are inherited from class `Object`, and (3) the `Company` class will be modified to reflect the changes to the employee class hierarchy. You should be able to reuse most of the code you wrote for Lab 8.

Attendance/Demo

To receive credit for this lab, you must demonstrate your work. **Also, you must submit your lab work to cuLearn**. (Instructions are provided in the *Wrap Up* section at the end of this handout.)

When you have finished all the exercises, call a TA, who will grade the code you wrote. For those who don't finish early, a TA will ask you to demonstrate whatever code you've completed, starting about 30 minutes before the end of the lab period. **Any unfinished exercises should be treated as "homework"; complete these on your own time, before Lab 10.**

Attendance/Demo

To receive credit for this lab, you must make an effort to complete the exercises, and demonstrate your work. When you have finished the exercises, call a TA, who will review the code you wrote. For those who don't finish early, the TAs will ask you to demonstrate whatever code you've completed, starting at about 30 minutes before the end of the lab period.

References

Objects First with Java, Chapters 8, 9 and 10.

Getting Started

Step 1: Create a new folder named Lab 9.

Step 2: Download company2.zip to your Lab 9 folder.

Step 3: Extract the *company2* project from the zip file into a project folder named company2.

Step 4: Open the company folder. Double-click on the BlueJ project file, which is named package. This will launch BlueJ and open the *company* project.

Background - Modelling Employees

A company has two kinds of employees. Salaried employees have a name, an ID number and an annual salary; for example, Cathy Coder has ID 123456 and earns $60,000 per year. Hourly employees have a name, an ID number and are paid an hourly wage. For example, Harry Hacker has ID 111222 and works 15 hours a week for $20.00 per hour (so he earns $300 per week).

Exercise 1

Step 1: The project contains incomplete implementations of classes named `Employee`, `SalariedEmployee` and `HourlyEmployee`. Your task is to complete the design and implementation of the classes, subject to the following constraints. **Remember to avoid smelly code; for example, unnecessary code duplication.**

- The `SalariedEmployee` and `HourlyEmployee` classes must be subclasses of the abstract `Employee` class.
- It's up to you to decide what fields are needed by `SalariedEmployee` objects and `HourlyEmployee` objects. Some of the fields should be declared in the superclass, `Employee`, and some of the fields should be declared in the subclasses. Use the knowledge of class inheritance that you've gained over the past three labs to help you decide where each field should be defined. **The visibility of all of the fields must be `private`.**
- `SalariedEmployee` and `HourlyEmployee` must have each have a two-parameter constructor that is passed the employee's name and ID number.
  - When an instance of class `SalariedEmployee` is created, the employee's name and ID must be initialized to the specified values, and any other fields in the object must be initialized to 0.
  - Similarly, when an instance of class `HourlyEmployee` is created, the employee's name and ID must be initialized to the specified values, and any other fields in the object must be initialized to 0.
- The *company2* project contains an interface named `IEmployee`, which specifies three abstract methods: `calculatePay`, `equals` and `toString`. Abstract class `Employee` implements this interface, which means that `Employee` inherits the three abstract methods from the interface. For each of the three methods, you have to decide:
  - Should the concrete method be implemented in `SalariedEmployee` and `HourlyEmployee`, but not in `Employee`?
  - Should the concrete method be implemented in `Employee` and inherited by `SalariedEmployee` and `HourlyEmployee`?
  - Should the concrete method be implemented in `Employee` and overridden in

`SalariedEmployee` and `HourlyEmployee`?

When implementing `equals`, keep the following in mind:

- ○ Two `SalariedEmployee` objects are equal if they have the same name, ID number and annual salary. Two `HourlyEmployee` objects are equal if they have the same name, ID number and hourly wage, and work the same number of hours.

- Do **not** copy the `hasSameID` method that you developed for Lab 8 into any of your classes. Because you're overriding `equals`, you don't need it.

- In addition to the methods declared in `IEmployee`, you must be able to call the following method on a `SalariedEmployee` object:

```
/**
 * Set this employee's annual salary to the
 * specified amount.
 */
public void setAnnualSalary(double salary)
```

- In addition to the methods declared in `IEmployee`, you must be able to call the following methods on an `HourlyEmployee` object:

```
/**
 * Sets this employee's hourly wage to the specified wage.
 *
 * @param wage the amount this employee earns in one hour.
 */
public void setHourlyWage(double wage)

/**
 * Sets the number of hours this employee works in
 * one week.
 *
 * @param hours the hours this employee works each week.
 */
public void setHoursWorked(double hours)
```

- You are not permitted to define any accessor (getter) methods in the `Employee`, `SalariedEmployee` and `HourlyEmployee` classes to retrieve an employee's name, ID, annual salary, hours worked or hourly wage.

Step 2: Create a few different `SalariedEmployee` and `HourlyEmployee` objects and place them on the object bench. Interactively test these objects.

Step 3: Run the unit tests in `SalariedEmployeeTest` and `HourlyEmployeeTest` and correct any bugs.

Exercise 2

Step 1: Read the Javadoc comments for class `Company` and finish the implementation of the class. This class must use **one** `ArrayList` object to store **all** the company's employees. Notice that `lookupEmployee` and `addEmployee` both have a parameter of type `IEmployee` (**not** **`Employee`**). A variable or parameter of type `IEmployee` is a *polymorphic* variable: it can contain a reference to an instance of `SalariedEmployee` or an instance of `HourlyEmployee`, because both classes are subclasses of abstract class `Employee`, which implements the `IEmployee` interface. This means that we can pass references to `SalariedEmployee` objects and `HourlyEmployee` objects to `lookupEmployee` and `addEmployee`.

When defining the `lookupEmployee`, `addEmployee` and `calculatePayroll` methods, you are not permitted to call `getClass` or use Java's `instanceof` operator to determine if an employee is a salaried employee or an hourly employee.

You may want to refer to the API for `ArrayList`:

> http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html

Step 2: Run the unit tests in `CompanyTest` and correct any bugs.

Exercise 3

Step 1: Close the *company2* project. Make a copy of the company2 folder, and rename it company3. Open the *company3* project in BlueJ. For this exercise, modify the code in the *company3* project, not the *company2* project.

Step 2: In Exercise 1, you overrode the `equals` method. This method considers two employees to be equal only if all their fields are equal. Some software designers would argue that two `SalariedEmployee` objects or two `HourlyEmployee` objects employees should be considered equal as long as their employee IDs are equal; in other words, their other fields should not be considered when determining equality.

Modify `equals` to use this revised definition of equality. Where should this method should be defined? For example, you could:

- define `equals` in `SalariedEmployee` and `HourlyEmployee`;
- define `equals` in `SalariedEmployee` and `HourlyEmployee`, and have these methods call an `equals` method that is defined in `Employee`;
- define `equals` in `Employee`, and have your `SalariedEmployee` and `HourlyEmployee` classes inherit the method.

From an object-oriented perspective, which is the best solution?

Step 2: Run the unit tests and correct any bugs.

Wrap-Up

1. With one of the TAs watching, run the JUnit tests in `SalariedEmployeeTest`, `HourlyEmployeeTest` and `CompanyTest`. Do this for both projects. The TA will note how many test cases pass. The TA will review your solutions to the exercises, assign a grade (Satisfactory, Marginal or Unsatisfactory) and have you initial the demo/sign-out sheet.

2. The next thing you'll do is package the *Company3* project in a *jar* (Java archive) file named company3.jar. (You do not have to submit your *Company2* project). To do this:

   2.1. From the menu bar, select Project > Create Jar File... A dialog box will appear. Click the Include source and Include BlueJ project files check boxes. A check-mark should appear in each box. Do not modify the Main class field.

   2.2. Click Continue. A dialog box will appear, asking you to specify the name for the jar file. Type company3 or select the BlueJ icon named company3 in the list of files. **Do not use any other name for your jar file** (e.g., lab9, my_project, etc.).

   2.3. Click Create. BlueJ will create a file named company3 that has extension .jar. (Note: you don't type this extension when you specify the filename in Step 2.2; instead, it's automatically appended when the jar file is created.) The jar file will contain copies of the Java source code and several other files associated with the project. (The original files in your company3 folder will not be removed).

3. Before you leave the lab, log in to cuLearn and submit company3.jar. To do this:

   3.1. Click the Submit Lab 9 link. A page containing instructions and your submission status will be displayed. After you've read the instructions, click the Add submission button. A page containing a File submissions box will appear. Drag company3.jar to the File submissions box. Do not submit another type of file (e.g., a .java file, a RAR file, a .txt file, etc.)

   3.2. After the icon for the file appears in the box, click the Save changes button. At this point, the submission status of your file is "Draft (not submitted)". If you're ready to finish submitting the file, jump to Step 3.4. If you instead want to replace or delete your "draft" file submission, follow the instructions in Step 3.3.

   3.3. You can replace or delete the file by clicking the Edit my submission button. The page containing the File submissions box will appear.

      3.3.1. To overwrite a file you previously submitted with a file having the same name, drag another copy of the file to the File submissions box, then click the Overwrite button when you are told the file exists ("There is already a file called..."). After the icon for the file reappears in the box, click the Save changes button.

      3.3.2. To delete a file you previously submitted, click its icon. A dialogue box will appear. Click the Delete button., then click the OK button when you are asked, "Are you sure you want to delete this file?" After the icon for the file disappears, click the Save changes button.

3.4. Once you're sure that you don't want to make any changes to the project you're submitting, click the Submit assignment button. A Submit assignment page will be displayed containing the message, "Are you sure you want to submit your work for grading? You will not be able to make any more changes." Click the Continue button to confirm that you are ready to submit your lab work. This will change the submission status to "Submitted for grading".