Assignment #3

Text-based Payroll Application

Filenames: Payroll.java, Employee.java, Utility.java

Create a project named Payroll. In this project, we will create two classes – Payroll and Employee – and use a third class that you can download from our course web site. Payroll will contain the logic of our payroll application. The Employee class will be a re-usable class that we can use to represent an Employee object. The Utility class contains several class methods that you will end up using throughout the semester.

1. Create your Employee class. Important note: Since this is not the class that represents the main functionality of our application, it will NOT have a main method. Don't check the checkbox for a main method.

2. Your Employee class needs one instance variable (property):

public double payRate;

3. You need a constructor method so your application program can instantiate an Employee object. Your constructor should receive a double, which is this employee's payrate. The constructor needs to assign that parameter to your instance variable for more persistence. (Remember that the name of a constructor method must match the name of its class: Employee. Also, it is not allowed to specify a "return data type", not even "void").

```
public Employee(double thisPayRate)
{
□□this.payRate=thisPayRate;
}
```

4. You need a method named calculatePay that receives an int (hoursWorked) and returns a double (grossPay). This method should calculate the employee's grossPay (hoursWorked times payRate).

```
public double calculatePay(int hoursWorked)
{
□double grossPay=hoursWorked * this.payRate;
□return grossPay;
```

```
}
```

5. Create a class that will contain the main logic of your application. Name this class Payroll.

6. Initially, inside your main method define a hard-coded hoursWorked and payRate:

```
int hoursWorked=40;
double payRate=8.75;
```

7. Inside your main method, instantiate an Employee object, passing payRate to the constructor method. Then call the calculatePay method of that Employee object (it's an instance method), passing hoursWorked as your argument, to calculate the gross pay. Display the gross pay to the screen.

```
Employee employee=new Employee(payRate);
double grossPay=employee.calculatePay(hoursWorked);
System.out.println(grossPay);
```

Now, that's a pretty good start, but there are a number of things that you can now do:

8. Modify the program to accept both hoursWorked and payRate from the keyboard. One of our early course packet DOS-based programs uses a readInt method, something like:

```
int hoursWorked=Utility.readInt();
```

Utility.java is on our course web site. Download a copy of Utility.java. Open it in a text editor.

Back in Eclipse, create a new class named Utility. Delete any automatic code that Eclipse puts in that file, and copy/paste the entire Utility.java code from your text editor into this new class in Eclipse.

9. You will, of course, want to display a prompt before reading the input.

```
System.out.println("Hours worked?");
int hoursWorked=Utility.readInt();
```

10. Use Utility.readDouble to accept the payRate from the keyboard.

11. One of the reasons for defining the Employee class is that it separates the functionality of the Employee object from the functionality of your application program. The person who writes the application program doesn't need to know anything about calculating pay; he or she just needs to call the Employee class's calculatePay method. The person who maintains the Employee class needs to know how to calculate grossPay.

Modify the Employee class's calculatePay method to pay time and a half for overtime. That is, if the employee works over 40 hours, pay 1.5 times the payRate for any hours over 40.

Pretend that you are two different people (this will be easy if you actually believe that you ARE two different people). One of your personas is working on the application program, and your other persona (or one of your other personas) maintains the Employee class. You have just demonstrated that the application programmer doesn't need to know the inner workings of calculating gross pay. The application programmer didn't change his (or her, or its) application program; the other programmer changed the Employee class. The application programmer didn't have to recompile his program. As a matter of fact, the application programmer may not even know that anything has changed. But his program is working!

This demonstrates encapsulation. All of the functionality of an Employee object is encapsulated in the Employee class. It also demonstrates the concept of the class's interface. All the application programmer needs to know is the interface with the Employee class: if there are any public properties, the application programmer may manipulate those directly. If there are any public methods, what do they receive? and what do they return? The application programmer does not need to know the inner workings of those methods.

12. You could add a flat tax rate as a property of the Employee class.

```
private double taxRate=.10;
```

Now, if that is a flat taxRate, applied to all Employee objects evenly, does EACH Employee object need its own taxRate, or is there one taxRate for all Employee objects? Although either would work, it seems wasteful to have a separate copy of the value .10 for each Employee object when one copy would do, so you should change that line to:

```
private static double taxRate=.10;
```

You could write a calculateTax method now. This method would receive a double (grossPay) and return a double (netPay). When you use taxRate in your calculations, use Employee.taxRate since taxRate is a class variable (non-static properties are called instance variables; static properties are called class variables).

If you want to prevent the value of taxRate from being changed during the program, you can declare taxRate as "final". By convention, final variables are written in all upper case characters, using the underscore character to separate "logical words":

```
private static final double TAX_RATE=.10;
```

13. Try to think of some other things that you could add to this example.

14. Upload your .java files from this project (including Utility.java) to your mislab account.

Your link to your assignment #3 (in your index.html) should display your Employee.java file. It's in the src subdirectory of your project. Be sure to test your page from http://mislab.business.msstate.edu to verify that your links work.

Created 2019-01-09 23:24:30 UTC by Aaron Kimbrell
Updated 2019-02-07 00:12:10 UTC by Aaron Kimbrell