Assignment

Overview

Our system was SO successful with the vacuum company that we have decided to release it as a library for others to use. Before we do this we need to implement (get it ;)) some controls to help make our system more programmer friendly. We need to add some interfaces, enumeration and organize our files into packages as well as adding some documentation.

Directions

To Get Started:

1. Create a project in your IDE(IntelliJ, etc.) titled YOURNAME-Assignment 10
2. Download a copy of the **A10EmployeeChecker.java** file and put it in your project folder.
    - Also here is the **EmployeeSystemV4.java** file to update and play with.
3. Take a copy of the Employee.java, HourlyEmployee.java, CommissionEmployee.java, SalaryEmployee.java, EmergencyContact and put them in the src folder of your project.

Part 1: Printable Interface

Reference the following Printable UML interface below:

<<interface>>Printable
currency : NumberFormat percent : NumberFormat
print() : void

Create a new interface called Printable. This interface has two variables of type NumberFormat. These are variables that allow us to better format everything that can implement the Printable interface. Right now we already have these variables created on the Employee abstract class so we are going to move them around.

To do this:

- Move the currency and percent NumberFormat variables from the Employee file to the Printable
    - We don't need protected or static as the interface takes care of this for us
    - Also note the classes that use these NumberFormat will give you an error until we implement it down below
- Make sure to import NumberFormat when doing this (most IDEs should ask you if you want to import it automatically)
- You can then delete the import NumberFormat on the Employee class as we don't use it anymore

Now we have the Printable interface that we can implement on the Employee class. Modify the Employee abstract class header by implementing Printable and EmployeeType. This now gives some errors but also fixes some things.

- The toString() methods on all the classes should not have any errors as they have access to the percent and currency NumberFormat objects
- Now we have an error because we don't have a print() method on any of the classes. We do have a printEmployee() function that does what we want, so instead of creating another print method, rename the printEmployee() to just print(). Doing this in the Employee class makes it so we don't have to do it in each subclass.
    - This function needs to be overridden.

Part 2: Enumeration Types

Create a new java class of type Enumeration called EmployeeType. Reference the following UML below:

<<enumeration>>EmployeeType
HOURLY SALARY COMMISSION
toString(): String

- Create the constants for each employee type.
- Add a toString() method that returns either "Hourly" "Salary" or "Commission"
- Use a switch or if statement to check this.ordinal() seeing if it is 0, 1, or 2 for Hourly, Salary, or Commission respectively

Part 3: Update the Employees for the EmployeeType Enumeration

Now that we have our EmployeeType enum, let's add it into our system. Once again here is the UML Diagram for Employee with updates:

abstract Employee.java implements Printable
-firstName : String -lastName : String -employeeNum : int -department : String -jobTitle : String -employeeType : EmployeeType
<<constructor>>Employee(String fn, String ln, int en, String dept, String job, EmployeeType et) <<constructor>>Employee(String fn, String ln, int en, EmployeeType et) <<constructor>>Employee(Employee e) <<constructor>>Employee() +getFirstName() : String +setFirstName(String fn) : void +getLastName() : String +setLastName(String ln) : void +getEmployeeNumber() : int +setEmployeeNumber(int en) : void +getDepartment() : String +setDepartment(String dept) : void +getJobTitle() : String +setJobTitle(String job) : void +getEmployeeType() : EmployeeType +setEmployeeType(EmployeeType et) : void +print() : void +**abstract** resetWeek() : void +**abstract** calculateWeeklyPar() : double +**abstract** annualRaise() : void +**abstract** holidayBonus() : void +**abstract** setPay(double pay) : void +toString() : String +equals(Object obj2) : boolean

Variables

As you see we added in a variable of type EmployeeType called employeeType.

Constructor Modifications

In the first and second constructor we need to add another parameter into the list for accepting an EmployeeType. We then take that and assign it to our employeeType variable.

In the copy constructor you need to assign it using a get method. (get method we are coding below so code it and ignore the error). So it should be something like employeeType = e.getEmployeeType().

Lastly in the default constructor we are just going to have a default of the HOURLY value. So set employeeType to the EmployeeType.HOURLY.

Getters/Setters

We need both a getter and setter for our new employeeType variables

toString()

Lastly we are going to add in to the end of our toString() method so that it prints

```
Type: {employeeType}
```

To do this we just need to call the employeeType.toString() that you set up earlier.

Updating Hourly, Salary and Commission Employee classes

Constructors

Each super() constructor call is going to error as we need to now send the EmployeeType to it. We are not taking this on through the parameter because as we know an Hourly employee is HOURLY, Salary is SALARY and Commission is COMMISSION.

So in each of the constructors that have super() calls you need to add the respective parameters.

- EmployeeType.HOURLY
- EmployeeType.SALARY
- EmployeeType.COMMISSION

Part 4: Packages

Now add some organization. Refactor/Move our code files around into packages

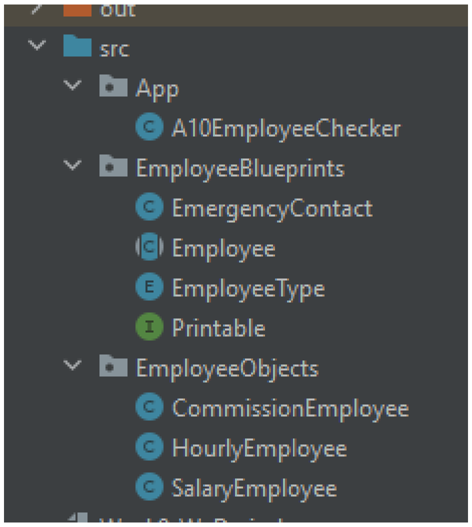
Right click on the project and create three new packages

- EmployeeBlueprints
- EmployeeObjects
- App

Move the Interfaces, enumeration, abstract Employee, and EmergencyContact into the blueprints folder. Then move the Hourly, Salary and Commission employees in the objects package. Lastly move the A10EmployeeChecker.java into the App package

When you do this make sure that IntelliJ (and most IDEs) should have a pop up window asking if you want to move and "Refactor" the code. This should automatically add in the imports/package statements on the file. If not you will need to add "package PACKAGENAME;" and "import PACKAGENAME.CLASSNAME;" at the top of each file that uses a file.

Your project should look like this:



Part 5: Javadoc Comments

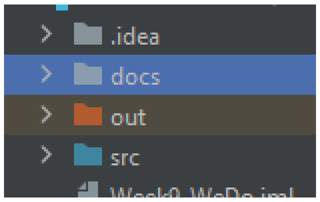
For this part you should have Javadoc comments EVERYWHERE on your project. **For each class there should be a header with your name as the author and then a version. (You can use 1 for version number)**

- Employee
- Printable
- EmergencyContact
- EmployeeType
- HourlyEmployee
- SalaryEmployee
- CommissionEmployee

For each method you should have a comment that explains if there are any parameters or returns. You also need a quick comment on what each method does. Remember if there are no @**params** or @**returns** you don't need to have those tags.

View your Javadoc comments when you are done and make sure they look good/make sense.

Build the Javadoc Comments in IntelliJ and put them in a folder called **"docs"** in your project. Check the build and make sure the only errors are ones NOT in your files that you have created.



Check Your Program

Running the Employee Checker should produce these EXACT results:

```
**** Salary Employees ****
Name: Steve Rodgers
ID: 3781
Department: Sales
Title: Manager
Type: Salary
Salary: $64,325.00

Name: Bruce Banner
ID: 7589
Department: Service
Title: Manager
Type: Salary
Salary: $43,876.00

**** Hourly Employees ****
Name: Tony Stark
ID: 5749
Department: Service
Title: Lead Service Manager
Type: Hourly
Wage: $32.85
Hours Worked: 0.0

Name: Thor Odinson
ID: 1281
Department: Operations
Title: Product Stocker
Type: Hourly
Wage: $12.76
Hours Worked: 0.0

**** Commission Employees ****
Name: Clint Barton
ID: 6847
Department: Sales
Title: Customer Representative
Type: Commission
Rate: 2.65%
Sales: $0.00
```