Java Lab questions

Q: July 27

For Lab 2, I created an Address class, which contains:

streetAddress

city

stateOrProvince

country

postalCode

I agree that I should add the following elements:

Additional info (Box number, Suite, Apt., etc)

Preferred phone number

Preferred e-mail address

The main part I want to focus on is the streetAddress part because I used that to help follow the conventions on the following webpage for United States Addresses.

http://www.upu.int/fileadmin/documentsFiles/activities/addressingUnit/usaEn.pdf

Do you still want us to make our Address class as you stated instead? This would change streetAddress into the following:

Street number

Street name

Street designator (St., Ave., Blvd., etc.)

Your proposed change would make it so several valid address types on the webpage above would be rejected.

A: You are correct that the format I have would not accept all forms of US addresses. However, since I am the customer, you should use the format I specify.

Q: July 25

I’m a bit confused about how the signContract method of PerformanceGroupCompany is supposed to work. The method is located on an abstract superclass of Company and PerformanceGroup, but needs to “Sign contracts”. Contracts contain references to subclasses of PerformanceGroupCompany. Doesn’t including this break OOD principles since it would involve the superclass ‘knowing’ about its child classes? Or am I misunderstanding what this method is supposed to do?

Could you please give me some ideas on how this method is supposed to operate?

A: The signContract method can be in the superclass because the method can be left unimplemented in the superclass and only implemented in the subclasses. If it is left unimplemented in the superclass, then the superclass would be declared abstract. Since we are not using PerformanceGroupCompany to make objects, then it is fine for it to be an abstract class.

Q: July 24

In your instructions, you specify the following:

I. Interface for CalculatePay as

double CalculatePay(double Amt1, double Amt2);

and go on to specify how it should be used for various performance hall personnel.

The method for CalculatePay for a performance hall staff member is to pass in the pay scale per hour and the hours worked for one week. The theatre staff member's weekly pay is :

pay per hour \* hours per week

The method for CalculatePay for a performance hall manager is to pass in the weekly salary and the hours worked for a week. The hall manager's weekly pay is :

If hours per week >= 40

Pay = weekly salary

Else

Pay = weekly salary \* (hours per week/40)

The method for CalculatePay for a performance hall agent is to pass in the weekly contract amount and the hours worked for a week. The hall agent's weekly pay is :

If hours per week >= 10

Pay = weekly contracted amount

Else

Pay = 0

However, the way it is used it is always on a Person and never on anything else. In addition, you call for the following class definition for Person calls for:

E. Each Person has the following attributes EXAMPLE

1. First name Jaya

2. Last name Smith

3. Identification number 006 – can be left blank

4. Address (in an Address Object – see Address class below)

5.Date of Birth (in a PDate object – see PDate class below)

6. Job Title Manager or Agent or …

7. Pay Type enumerated as SALARY, HOURLY, CONTRACTED, OTHER

8. Hours worked per week

9. Pay rate per hour

10. Weekly salary

11. Weekly contract amount

The way I’m thinking you envisioned this, it appears that Person is meant to be implemented as Payable and for the performance hall personnel to individually be set to their proper pay type. However, Person, where the function would be implemented, already has privileged access to its own members, thus negating the need to pass anything in. The implemented calculatePay(double, double) function would also have access to the Person’s “Pay Type” (#7) and would be able to determine the proper pay amount itself. All that would be practically necessary to implement this is to use a switch to determine the Person’s pay type and then use Person’s members to determine the proper pay.

I would like to ask if I could use a different prototype for calculatePay:

double calculatePay();

This way, the implementation of calculatePay could be left up to the entity in question and open the interface up for more uses. This new calculatePay function could also determine total pay for Contracts, PerformanceGroupCompanys, Companys and PerformanceGroups to determine payouts for those entities.

May I use this prototype for calculatePay instead?

A: For this particular situation, I would like you to use calculatePay for determining pay for Persons and for determining pay to PerformanceGroupCompany s as well. You may create a polymorphic version of calculatePay with no inputs but it should also be part of the interface that has the version with two inputs. You may create an adapter class if desired and use that – similar to the adapter classes for the various input events.

Q: July 24

Are we permitted to do file I/O using the Serializable interface?

A: For Lab 3 there is no reason to use the Serializable interface. The file input for this lab will be a text file of input data values that you would read in, in order to be able to perform the various menu operations on them.

You can think of this program as having an initial part where various pieces of data must be put into the system such as the information about the performance hall, about each group, company, and performance, and all of the associated people and accounts. After this data has been entered, then the user can select actions from the menu to perform.

Q: Lab 2 questions

Are we permitted to use generics for Lab 2? For example, I’d like to define the following:

public class PerformanceGroupCompany<E extends Enum<E>> {

// other members and methods here

abstract public E getType();

abstract public void setType(E type);

}

enum CompanyType {

ADVERTISING\_CO,

CATERER,

RENTAL\_CO,

OTHER\_CO

};

pubic class Company extends PerformanceGroupCompany<CompanyType> {

private CompanyType type;

public Company() {

// other initialization

type = CompanyType.OTHER\_CO;

}

public CompanyType getType() {

return type;

}

public void setType(CompanyType type) {

this.type = type;

}

// other members and methods

}

A: Generics were not allowed for Lab #2 but are allowed for Lab #3

Q: July 22

The lab 3 requirements say that “output all information to the user in a GUI window”. I know we addressed it in class, but I was a bit hazy on the answer. Does this mean we’re abandoning the console the primary means of communication with the user and switching entirely to a GUI interface? For example, my current program asks for user input using the console. Now I should ask using a GUI window? Or should I only worry about implementing GUI windows for lab requirements that specifically call for “output”? Or something else…?

A: The pieces of information that are displayed to the user by the program should be shown in GUI windows rather than the console interface. There is initial input data to be read in from a file as mentioned in the previous questions. Once the user starts selecting items from the menu, your program may use either the GUI windows or console window to receive input.