CS116

LAB 5

Lab 5 is due on March 26 on Blackboard by 10:00 p.m time stamped. Use Notepad++ or EditPlus or equivalent text editor to create source code.

This lab assignment is worth 3 points.

Objectives:

The purpose of this laboratory assignment is to create different relationships between a number of classes and create a hierarchical tree of relationships.

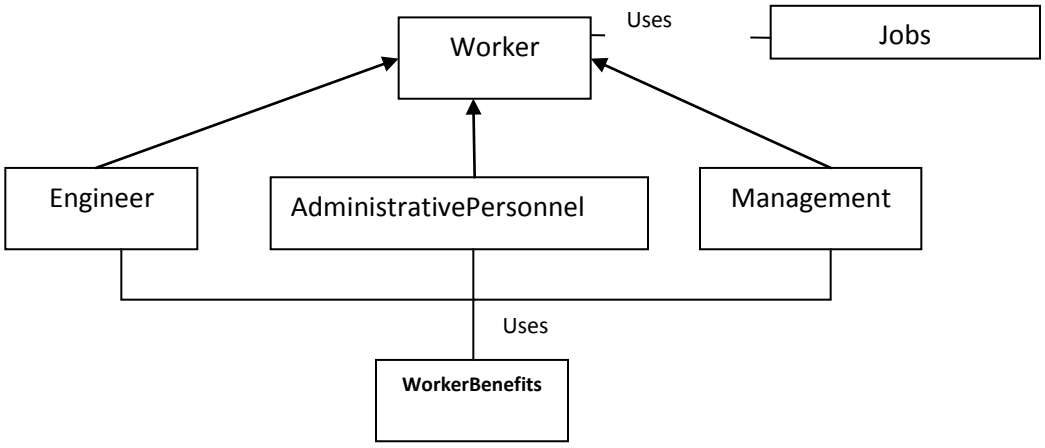
1. Inheritance.
2. Abstract classes.
3. Interfaces and simulation of multiple inheritance using interfaces.

Task 1 (2.0 points)

This is an example of simulating Multiple Inheritance as discussed in the lecture. Although this lab assignment requires a lot of classes, the code in each class is very minimal. The emphasis is on the relationships between the classes.

Thus far we have created a hierarchical tree for employees in Lab 4 –TASK 2 and we want to use it here. Packaging remains the same as in Lab. 4. Make sure that you copy YOUR classes from Task2 of Lab 4 in the folder TASK 1 of this Lab.

Goal of this Lab : In lab 4 Task 2 we created the following classes:



The objective of this lab is to create the architecture of classes needed to track the WorketBenefits and in addition the recurring expenses of a Corporation, by creating a new class called RecurringExpenses Both classes will need to be inherited by third class that brings the two types of corporate expenses (WorkerBenefits and RecurringExpenses) together.
Before we create the RecurringExpenses class some other work is needed in order to create the individual types of recurring expenses that is described further down.
When we finish the leg of the architecture that deals with recurring expenses, we will bring the two legs of the architecture (WorkerBenefits and RecurringExpenses part) together by utilizing another class which will try to inherit these two classes. We will then test this architecture with a Client class called Corporation in the second Task of this lab.

1. Description of new Classes/Interfaces

We have a corporation that pays benefits and also has recurring expenses (utilities, rent etc.). In this task we are going to use the classes created earlier in lab 4 and also create some new ones.

Revise the class WorkerBenefits as follows:

- Comment out the main method.
- Add the following method
  public double totalBenefits();
  This method first acquires the ArrayList of Worker objects and then it calculates the total of all benefits paid for the workers in the list. It returns that total of all benefits.

2. New Relationships

a. First let us create the service classes for the 3 types of Recurring Expenses: Utility, Rent and Material WHICH RESIDE IN PACKAGE **Client.Services.Enum.Help**. For each of the classes create all the required methods (constructors, accessor/mutator) for each service class including the toString method

These classes have the following attributes:
UtilityCosts class: Attributes: double electricity, double gas, double water. Add a static variable called utilcostsforallobjects of double data type and only the accessor method for this static variable. Do not pass the static variable as argument to the non default constructor.
It also has a helper method:
public double totalUtilCosts(); it calculates and returns the sum of all 3 utilities.
The method totalUtilCosts also calculates the total of all Utility objects by adding the sum of the 3 utilities to the **static** instance variable utilcostsforallobjects; The method

also displays the total cost for one object as well as the accumulated cost for all objects created up to that point of execution of this method.

Rent class: Attributes: String location, int yearOfLeasing, int buildingNumber. Add a static variable rentcostforallobjects of double data type and only the accessor method for this static variable. Do not pass the static variable as argument to the non default constructor.
It also has a method:
public double rent(); It calculates the rent as follows:
location is parsed to a double and used in the expression:
buildingRent=locationDouble*yearOfLeasing*150/buildingNumber;
The method adds the buildingRent to the static instance variable rentcostforallobjects. Therefore if multiple Rent objects are created the static variable holds the sum of all (why? Think about it). It also displays the cost of the building and the accumulated rent cost of all objects.

Material class: Attributes: String materialName, double unitPrice, int quantity. It has a static variable double materialcostforallobjects. Add only the accessor method for this static variable and do not pass it as argument in the non default constructor.
It also has a method :
public double materialPrice(); It calculates the price of each of each material object by multiplying the unit price times the quantity minus the discount of 20% if the quantity is higher or equal to 100 or 10% otherwise, and returns it. It also keeps track of the total accumulated price for all Material objects by adding the material price calculated to the materialcostforallobjects instance static instance variable. It also displays the materil price for the object and the accumulated material cost for all objects.

b. The class RecurringExpenses resides in the package Client.Services.Enums.Help and it is responsible for keeping track of the various operating expenses such as: utilities, rent, materials etc. and calculating the total Recurring Expenses for all three types of recurring expenses (similar to WorkerBenefit class getting all benefits). It can only inherit one class directly using the keyword extends, and that is the class Rent. The other two will be inherited via interfaces. Therefore the class RecurringExpenses should inherit Rent class and implement the two interfaces listed below:
    i. Create the interface UtilityCostsInterface that defines the abstract method for the Rent class: public double getUtilityExpenses();
    ii. Create the interface MaterialCostsInterface that defines the abstract method for the Materials class: public double getMaterialExpenses();
    iii. Create the corresponding Impl classes: for the two interfaces:
    Class UtilImpl which implements the interface UtilityCostsInterface and inherits the class UtilityCosts

and
class MaterialImpl which implements the interface MaterialCostsInterface and inherits the class Material.
Each one of these classes implements the corresponding method from the interface by utilizing the corresponding accessor method from each of the classes UtilityCosts and Material **that returns the value of their static variable**. Remember that the value of the static variable holds the total value for utilities or material for all objects of these classes.

Note: The two interfaces and the two Impl classes also reside in the same package as the RecurringExpenses class.

Keep in mind that the class RecurringExpenses implements the methods in the corresponding interface by using an object of the corresponding Impl class. This is done as described in the example discussed in class.

If you were absent you need to get a copy of the diagram discussed in class as an example.

The class RecurringExpenses has a method called totalRecurringExpenses which calculates and returns as a double the sum of all 3 types of accumulated recurring expenses (rent, utility and material) for all objects created. It also displays the total expenses for all objects (see sample output).

The class also has a method called createExpenses that returns void and takes no arguments. In this method create the following 6 objects:
UtilityCosts object with values: 100.00, 100.00, 100.00
UtilityCosts object with values: 150.00 150.00, 150.00
Rent object with values: "2", 3, 4
Rent object with values: "4", 1, 1
Material object with values: "Paper", 3.0, 1000
Material object with values: "Ink", 2.0, 200

Make sure that for each object you invoke the corresponding method from that class is order to calculate the total **for that object** only and also set the accumulated value for all objects created in that category of expense.
Display the corresponding total value for each object (see the sample output).

c. **Draw an architectural diagram** that shows how all these new classes (The Utility,Rent , Materil classes, the interfaces, the corresponding Impl classes and the

RecurringExpenses class) are related. Use rectangles to represent the classes and interfaces and arrows to show their inheritance relationships. If a class only uses another class via an object of that class (without an inheritance relationship) draw a line without an arrow to the class being used and write the word ***uses*** next to the line. An arrow indicates inheritance (the arrow points towards the super class or the interface).

Summary of relationships:

- From Lab 4: Abstract superclass Worker (it has an abstract method for benefitsCalculation that gets implemented by each of its subclasses). It uses the enum Jobs.
- From Lab 4:Engineer inherits Worker.
- From Lab 4:Management inherits Worker.
- From Lab 4: AdministrativePersonnel inherits Worker.
- From Lab 4 (with the modifications requested above): WorkerBenefits class. Uses the hierarchical relationship of Worker. Remember that "uses" means that WorkerBenefits instantiates objects of the different worker types. It does NOT inherit.
- New :Material class
- New: UtilityCosts class
- New:Rent class
- New: MaterialCostsInterface
- New:UtilityCostsInterface
- New: UtilImpl class implements the interface UtilityCostsInterface and inherits UtilityCosts
- New:MaterialsImpl implements interface MaterialCostsInterface and inherits Materials
- New: RecurringExpenses which implements the two interfaces and inherits the Class Rent. It also uses the classes UtilImpl and MaterialImpl.

Note: The term "uses" means that a class has to create an object of another class to invoke its methods i.e. class A uses class B means that in class A we will instantiate an object of B to be able to invoke its methods.

Answer (draw the diagram of the relationships in document called relationships.doc and submitted with your code):

Task 2 (1.0 points)

Let us continue building the architecture of our classes. Our goal is to create a class that calculates the total expenditures of a corporation. Total expenses being the sum of the worker benefits plus the recurring expenses. We will call this class AllExpenses. AllExpenses is going to inherit the class WorkerBenefits (and thus inherit the method totalSalaries) and also inherit via an interface the method totalRecurringExpenses() from the class RecurringExpenses.

1. Write the code for the new classes

d. We are also going to create an interface called ***ExpenseInterface*** that has the abstract method and resides in Client.Services.Enums.Help:

```
public  abstract double getRecurringExenses();
```

This way we are going to give the ability to access the total operating expenses from the class RecurringExpenses.

e. We are going to need a class called ***ExpenseImpl*** which inherits ***RecurringExpenses*** and implements our interface ExpenseInterface and resides in package Client.Services.Enums.Help:

- In this class implement the method *getRecurringExpenses*() by using a call to the totalRecurringExpenses() method of class **RecurringExpenses** . The call captures the returned value of the total operating expenses (remember that this class has full access to the public methods of its superclass) and in turn returns it.

a. Next create a class ***AllExpenses*** which ***inherits WorkerBenefits*** and ***implements the interface ExpenseInterface***. It has no main method and it resides in package Client.Services.Enums.Help .
   - It has no attributes:
   - It has a default constructor that does nothing.
   - This class also implements the method ***getRecurringExpenses*** (since it is also implementing the interface). To implement it, however, it uses the implementation from class ExpenseImpl:
     - We instantiate an object of ExpenseImpl call it eximpl and then invoke: eximpl.getRecurringExpenses() (it invokes the implementation in class ExpensesImpl and captures its returned value). I would suggest that you instantiate the object eximpl globally in the class (at the top of the class outside any method).

   - It also has the method *public double getTotalExpenses()*
     In this method we call the totalBenefits method of WorkerBenefits class and the *getRecurringExpenses* method of AllExpenses class to calculate the total of salaries and operating expenses. It returns that value.
   - This class, however, has to activate the recurring expenses. Therefore, another method is needed that calls the method createExpenses of the RecurringExpenses class in order to instantiate the various operational expenses objects and set their attributes values. You can name this method also createExpenses. Disply the recurring expenses (for each object again-See sample output).

b. Next let us create a class called ***MainClass which resides in the Client package.*** This class has a **main method** and acts as a client class to the structure of classes that we have created thus far.

   Note: THIS CLASS MUST MAKE ALL INVOCATIONS WITH AN OBJECT OF AllExpenses CLASS. NO OTHER OBJECT (Penalty -0.6 points if it does not work that way)!

   - Get the list of Workers from WorkerBenefits class.
   - Invoke the method displayData() of WorkerBenefits and output the data on the various employees (after you get hold of the ArrayList so that it can be passed a sargument).
   - Invoke the method createExpenses of the AllExpenses class in order to create and display the recurring expenses objects.

- Next invoke the totalbenefits method of the ***WorkerBenefits*** class and capture the sum of the salaries . Output the total salaries paid via System.out.println("The sum of all salaries paid was:"+…….);
- Output the value of the total of recurring expenses via System.out.println("The total operating expenses were:"+……..);
- Next invoke the method getTotalExpenses() and capture the total of all expenses. Output the value via System.out.println("The total of all expenses were:"+…….);

Note: Observe the sample output. Since we are also outputting the total value of each operational expense object as well as the accumulated total for that type of the operational expense object, you will need to insert some System.out statements in the proper methods of each operational expenses class. Therefore, go back to these classes and make the corresponding changes.

2. Compile the classes and run the program.
3. Amend the diagram described in part 2c of Task 1 to include the additional classes/interfaces described in Task 2:

SAMPLE OUTPUT:

---------- Run Java ----------
---------- Java Interpreter ----------
C:\CS116\SPRING2014\Labs\Lab5\Lab5Solution>java Client.MainClass
FROM THE INVOCATIONS IN MAIN CLASS
The benefit is 1800.0 The name is: John Eng1 The Job type type is ELECTRICAL_ENGINEER The id is: 1
The benefit is 2000.0 The name is: Jim Man1 The Job type type is ENGINEERING_MANAGER The id is: 2
The benefit is 2050.0 The name is: Jim Man2 The Job type type is ADMINISTRATIVE_MANAGER The id is: 3
The benefit is 950.0 The name is: Helen Adm1 The Job type type is ADMINISTRATIVE_SECRETARY The id is: 4
The benefit is 2040.0 The name is: Maria Eng2 The Job type type is MECHANICAL_ENGINEER The id is: 5
The benefit is 1800.0 The name is: Jim Man3 The Job type type is ENGINEERING_MANAGER The id is: 6
The benefit is 950.0 The name is: John Doe1 The Job type type is ADMINISTRATIVE_ASSISTANT The id is: 7
The benefit is 3600.0 The name is: Mary Eng3 The Job type type is ELECTRICAL_ENGINEER The id is: 8
The benefit is 1115.0 The name is: Nick Doe2 The Job type type is ADMINISTRATIVE_ASSISTANT The id is: 9

The benefit is 2500.0 The name is: Ann Man4 The Job type type is ADMINISTRATIVE_MANAGER The id is: 10

FROM THE ALLEXPENSES CLASS

util cost for this object=300.0
total utilities of all objects=300.0
The firts utility object has: 300.0
util cost for this object=450.0
total utilities of all objects=750.0
The second utility object has: 450.0
Building rent=225.0
Total rent for all objects is =225.0
The first rent object has: 225.0
Building rent=600.0
Total rent for all objects is =825.0
The second rent object has: 600.0
Material price=2400.0
Total material cost=2400.0
The first Material object has: 2400.0
Material price=320.0
Total material cost=2720.0
The second Material object has: 320.0

BACK TO MAIN CLASS

The total benefits paid is: 18805.0
The total recurring expenses are:4295.0
The total recurring expenses is: 4295.0
The total recurring expenses are:4295.0
The sum of all benefits and recurring expenses is: 23100.0

Output completed (0 sec consumed) - Normal TerminationOutput completed (0 sec consumed).

Submission instructions

- In your submission you must include
    a. This document with the answers (copies of the outputs as requested above).
    b. The source code files and the compiled files (the package) from Task 1 in Task1 folder.
    c. The source code files and the compiled files (the package) required for Task 2 in the Task2 folder.

- Zip all files and name the zip file using your first name followed by your last name followed by lab1.

i.e. George_KayLab6.zip

- Upload the file on Blackboard in assignment folder “Lab 6”.
-