UFAZ / Strasbourg University

Object Oriented Programming

Year 1 – Computer Science

Extra exercises (cover all topics)

Exercise 1 – Horses

1. Create a class `Horse` containing the attributes name, color and year of birth.
2. Implement getters and setters for the member variables
3. Implement a constructor
4. Create the classes (a) `Racehorse` and (b) `Workhorse`. In the case (a), you must add field a field holding the number of races won. In the case (b), add a field that contains the number of years the horse has been working. Write the appropriate getters and setters.
5. Test your classes

Exercise 2 – Poems

1. Create a class `Poem`. A poem is characterized by a title and a number of lines.
2. Create the class constructor requiring a title and a number of lines as parameters.
3. Create these subclasses: `Couplet`, `Limerick` and `Haiku`. For all these subclasses, the constructors only requires a title, since all these poems have a constant number of lines: 2 lines for a couplet, 5 lines for a limerick and 3 lines for a haiku.
4. Test your classes

Exercise 3 – University courses

1. Create the class `Department`. In a university, a department is characterized by a name (such as CS for computer science, CHEM for chemistry, PHY for physics)
2. Create the class University. A university is characterized by a name, a city and a set of `Department`. Do you use aggregation or composition to manage the departments? (justify your answer).
3. Create the class `UniversityCourse`. A university course is characterized by a name, the `Department` holding the course, a course number and a number of credits. Do you use aggregation or composition to specify the department holding the course? (justify your answer)
4. Define the method `String toString()` in the class `UniversityCourse` so as to obtain the following output:

```
308 - Object Oriented Programming (department of Computer Science) / 3 credits
```

   Is this overriding or overloading?
5. Test your classes

Exercise 4 – Runner

1. Create an interface `Runner` and declare the following method in the interface: `public void run();`.
2. Create the classes `CoffeeMachine`, `Athlete`, `PoliticalCandidate` and implement the method `run()` in all three classes.
3. Test your classes.

Exercise 5 – Newspaper subscription

[The problem description is high-level: you must read the entire problem statement before starting working on the problem. Drawing a diagram will be highly useful. You must identify the classes, the properties of those classes and the methods. Ask yourself what relationship there is

between the objects (e.g. do you need polymorphism? Inheritance? Aggregation or composition?]

You have been hired as an IT tech at a local newspaper company. They ask you to write a java program to handle subscriptions to the paper. A subscription is characterized by the subscriber name, the subscriber address and a subscription rate. There are two types of subscription: physical subscription and online subscription. In the case of physical subscription, the address must contain at least one digit. Otherwise, an exception is thrown. If the address is valid, the subscription rate is set to $5. In the case of the online subscription, the address must contain the symbol '@'. If the address is not valid, an exception is thrown. If the address is valid, the subscription rate is set to $3. It is mandatory to set the address so the subscription rate can be computed so design your classes carefully!

• Design a class model, implement it and test it.

Exercise 6 – Chemical elements

[The problem description is high-level: you must read the entire problem statement before starting working on the problem. Drawing a diagram will be highly useful. You must identify the classes, the properties of those classes and the methods. Ask yourself what relationship there is between the objects (e.g. do you need polymorphism? Inheritance? Aggregation or composition?]

You are working on a Chemo-informatics project. One part of this project is to build an application to store chemical elements. An element has the following properties: a symbol, an atomic number and an atomic weight. You will need a constructor that requiring values for each of those properties.

• Create the corresponding class.

Then, you are asked to differentiate metal elements from non-metal elements. For a given metal/non-metal element, you must be able to display a brief description of their properties (good conductor / poor conductor ; malleable as solid / brittle, hard or soft ; etc.). It is mandatory to be able to describe an element based on its type so design your classes carefully!

• Improve your initial model by adding new classes and defining their relationships. Remember that all elements have a description of their properties.

Exercise 7 – Enrolling student to the university

[Prerequisite] exercise 3 must be completed

We now want to enroll student in the University we created previously. A `Student` is characterized by a first name, a last name, a student number and an annual tuition. The amount of tuition depends on the level of the student so you cannot set the tuition before you have defined levels. The levels are the following: `UndergraduateStudent` and `GraduateStudent`. Tuition fees are the following: $2000 for an undergraduate student and $3000 for a graduate student. It is mandatory to set the tuition fee, so design your classes carefully!

• Design and implement the student model.

Now, you can enroll students into the departments of your university. The courses they will attend are different if they are undergraduate or graduate students: add a property `level` to the course you designed previously and assign 3 courses per students enrolled in a department.

You must be able to display a student and their properties. Create and implement this functionality. As always, think about where and how you want to define/implement this functionality.

Exercise 8 – grading students

[Prerequisite] exercise 7 must be completed

Mind the code redundancy! We have learned ways to factorize code so apply these principles carefully in your model.

You now need to assign grades to your students. Graduate and undergraduate students should have a grade for each course they attend. To implement this, you can associate a course reference to a grade. Design a generic class `Pair<F,S>` that you will use to implement this association. Now, back to grading: in this university, the grades are letters (A, B, C, D, E, F). Create a method to assign a grade to a course. This method must throw the exception `UnvalidGradeException` if the grade is not one of the above letters. Modify your classes so the grades are displayed when you "print" a student.

Test your classes and methods.