Introduction to Computer Science 2

Lab 3: Inheritance

Learning Goals:

- To learn the principle of inheritance.
- To learn how the principle of inheritance allows us to reuse code.
- To learn how to extend existing and new classes.

Exercise 1 (6.5 points)

In this lab, we will make a simple simulation of a hospital clinic. Our first step in building our model is to simulate the most important people in our hospital, these being doctors and patients. Since we might want to add more roles to our simulation in the future, such as nurses, janitors, and secretaries, we choose to implement an abstract class `Person`, containing general methods and instance fields shared by all people in the hospital clinic. Using this principle of inheritance allows us to reuse our code for all the different person objects we might have in our simulation.

1. Start by implementing an **abstract** class `Person`, with the following requirements:

- Two instance fields: a `name` of type `String`, and an `age` of type `int`.
- A parametric constructor, which takes a `name` and an `age` as an input, and sets the relevant instance fields to these values.
- Accessor and mutator methods for both `name` and `age` instance fields.
- The methods `equals` and `toString` - these are methods of Java's `Object` class (See * below for more details), which you need to override here.

2. After implementing the abstract class described above, your next task is to implement the class `Doctor`, whose objects represent our clinic's doctors. Derive this class from the class `Person`.
A `Doctor` object requires:

- Two instance fields `name` and `age`, inherited from class `Person`.
- Two additional fields: `specialty` (e.g. Pediatrician, Obstetrician etc.) of the type `String`, and a `double` field `officeVisitFee`.
- A parametric constructor, whose inputs are used to initialize all four instance fields.
- Accessor and mutator methods for the `specialty` and `officeVisitFee` fields.
- The methods `equals` and `toString`, which override the methods `equals` and `toString` from the class `Person`.

3. Next, implement a class `Patient`, whose objects represent the patients at our simulated clinic. Derive this class from the class `Person`.
A `Patient` object requires:

- Two instance fields `name` and `age`, inherited from class `Person`.
- An additional instance field `identificationNumber`, of the type `String`.
- A parametric constructor which initializes all three instance fields.
- An accessor and a mutator method for the field `identificationNumber`.

- The methods `equals` and `toString`, which override the methods `equals` and `toString` from the class `Person`.

4. Test class `Doctor` by creating a `main` method inside of it. Within this method, start by creating a `Doctor` object. Then, set a new specialty and new office visit fee for this doctor, and print the relevant information by using the output from the `toString` method applied to the object. Finally, test whether the object is equal to itself.

5. In a similar fashion, test class `Patient` by creating a `main` method inside of it. In this method, first create a `Patient` object. Then, set a new age and new identification number and print the information obtained from applying the `toString` method to the object. Finally, test whether the object is equal to itself.

*. Please define your classes and methods using the formats described in Appendices A1, A2, and A3. The definitions of methods `equals` and `toString` are given in the application public interface of class `Object` - these are standard built-in methods from the class `Object`, which you need to override (these aren't included in the appendices, since they have a default format). For more details, please have a look at https://docs.oracle.com/javase/tutorial/java/landl/objectclass.html.

Exercise 2 (3.5 points)

1. This exercise builds upon Exercise 1 above. In this exercise, we want to introduce a new class into our simulation of the hospital clinic, an object of which represents a hospital bill. This class, appropriately named `Bill`, must contain the following:

- Three instance fields: a `Doctor` field for the doctor, a `Patient` field for the patient, and a `double` field named `amountDue` that represents the amount (in euros) to be paid.
- A constructor with two input parameters: one for the `Patient` object and one for the `Doctor` object.
  - This constructor must set the field `amountDue` to be equal to the `Doctor` object's office visit fee.
- An accessor method `getAmountDue` which returns the amount to be paid.
- A `toString` method for the class `Bill`. This method has to output a `String` object that contains the names of the doctor and patient, and the amount due.

2. Next, implement a class `BillContainer`, which contains all the hospital clinic's bills. An object of this class can accept objects of the class `Bill` sequentially and provide the total fee for all the `Bill` objects added thus far. For this purpose, supply the class with a default constructor and a mutator method `addBill` that allows for an object of the class `Bill` to be added. In addition provide:

- An accessor method `totalFee` that outputs the total fee for all the `Bill` objects added to the bill container.
- A `toString` method that outputs a `String` object that is a concatenation of `String` objects outputted sequentially by the method `toString` over all the `Bill` objects added to the bill container.

3. Finally, test class `BillContainer` by creating a `main` method inside this class:

- Start by creating 2 `Doctor` objects and 2 `Patient` objects.
- Then, create 4 `Bill` objects for each pair of `Doctor` and `Patient` objects, and subsequently add them to a `BillContainer` object.
- Print the information obtained from the outputs of the `toString` and `totalFee` methods executed over the `BillContainer` object.

**. Please define your classes and methods using the formats described in Appendices B1 and B2. The definition of method `toString` is given in the application public interface of class `Object`, see * above for more information.

Appendix A1: Definitions of the Methods of Class `Person`

```
public Person(String name, int age)

public String getName()

public int getAge()

public void setName(String name)

public void setAge(int age)
```

Appendix A2: Definitions of the Methods of Class `Doctor`

```
public Doctor(String name, int age, String specialty, double officeVisitFee)

public String getSpecialty()

public double getOfficeVisitFee()

public void setOfficeVisitFee(double officeVisitFee)

public void setSpecialty(String specialty)
```

Appendix A3: Definitions of the Methods of Class `Patient`

```
public Patient(String name, int age, String identificationNumber)

public String getIdentificationNumber()

public void setIdentificationNumber(String identificationNumber)
```

Appendix B1: Definitions of the Methods of Class `Bill`

```
public Bill(Patient patient, Doctor doctor)

public double getAmountDue()
```

Appendix B2: Definitions of the Methods of Class `BillContainer`

```
public BillContainer()

public void addBill(Bill bill)

public double totalFee()
```

Honor code, coding style, and deliverable:

Try to solve the exercises with what you already know. You are welcome to expand your program to do extra things but they are not mandatory.

Plagiarism is not allowed! We will run sophisticated software that automatically detects similarities on source code among students. All plagiarism incidents will be immediately reported to the Board of Examiners!

Submission!

Submit your java files to canvas.
Ask your instructor in case there is a problem with your submission.

DO NOT SEND SUBMISSIONS VIA EMAIL
YOUR LAB WILL NOT GET GRADED!