Assignment #3

Introduction to Object Oriented Programming with Java

Start: Week of March 8th.
Duration: 3 weeks**.**
Delivery: Via Moodle, one hour before the start of the next assignment according to your group (week of April 5th).
Weight in the final grade: 25%

The objective of this practice is to introduce object-oriented programming with the Java language. For this purpose, we will develop several classes in Java (including their tests and documentation) implementing some components for a small vehicle management application. The following Java concepts will be used:

- *Primitive data types, String, Array and reference types (objects) defined by the programmer.*
- *Simple classes for implementing abstract data types using instance variables, class variables, instance methods, class methods, and constructors.*
- *Basic input/output for reading text files and displaying text on the console*.
- *Inheritance, method overriding*
- *Introduction to collections*
- *Good programming style and comments for documentation using Javadoc.*

Section 0. Introduction

In this assignment, we will continue developing the application of the provincial traffic police department (DGT), that you already started in **section 1 of assignment #2**. Initially, we will start from the description given in assignment 2, and in the following sections of this assignment, we will add new functionality and requirements.

Section 1. Drivers and vehicle owners (3 points)

The program below is responsible for connecting vehicles with the owners who have purchased them. These owners can be a person or a company, in which case there must be a responsible person. Also, each vehicle can have a driver associated with it, who must be a person (for managing fines, as we will see later). If the driver is not specified, the default driver is set to the person who owns the vehicle or to the responsible person of the company.

In the constructor of each type of vehicle, you can optionally indicate the owner (company or person). The owner or driver of a vehicle is set using the *setOwner()* and *setDriver()* methods. People can own more than one vehicle, and when printing each person, the vehicles they own should be printed as well.

Note:

- To represent the collection of vehicles owned by a person or company, you can use a list (for example, an *ArrayList*). This is a type of collection in Java that is parameterized with the type of elements it contains. For instance: *ArrayList<Vehicle> vehicles = new ArrayList <>();* declares a variable *vehicles* that is a list of objects of type *Vehicle*. You can add a vehicle *v* to the list using the *vehicles.add(v)* method, and get the size of the list with the *vehicles.size()* method.
- Also, pay attention to the declaration of packages in the first line and the import of elements in the second line: you must organize your classes in packages following this scheme.

In this section, you must:

a) Modify the design (the class diagram) to accommodate these new requirements. Note that a good object-oriented design may need additional classes not mentioned in the tester.

b) Create the necessary Java code to cover this functionality. Don't forget to follow the Java programming style guide available in Moodle, including comments, especially those used for Javadoc.

Note: It is possible that when implementing the following sections (in the creation of driving licenses), the tester stops giving the output specified here. In that case, modify this tester (for example, adding the corresponding driving licenses to the drivers) so that it produces the expected output.

Tester section 1 (available in Moodle):

```java
package pr3.traffic.vehicles;
import pr3.traffic.drivers.*;

public class TesterPr31 {
  public static void main(String[] args) {
        Person ann     = new Person("Ann Smith", 30);
        Person louise  = new Person("Louise Lane", 17);
        Person anthony = new Person("Anthony Johnson", 27);
        Company fdinc  = new Company("Fast Delivery Inc", ann);          // Ann is responsible for FDINC

        Vehicle fleet[] = {
                new Car("Fiat 500x", 2019, "1245 HYN", true, ann),       // Ann's car, who drives it
                new Truck("IvecoDaily", 2010, "5643 KOI", 2, fdinc),     // Truck of FDINC
                new Motorcycle("Harley Davidson", 2003, "0987 ETG", false)};

        fleet[2].setOwner(ann);
        // Method setOwner should allow receiving a Company: fleet[2].setOwner(fdinc);
        System.out.println("Can Louise drive a Harley? "+fleet[2].setDriver(louise));
        // Louise, being less than 18 years old, cannot be the driver

        for (Vehicle v : fleet ) {
                System.out.println(v);
                System.out.println("------------------------");
        }

        fleet[2].setDriver(anthony);
        System.out.println(fleet[2]);
        System.out.println("People:");
        System.out.println(anthony+"\n---");
        System.out.println(ann+"\n---");
        System.out.println("Company:");
        System.out.println(fdinc);
  }
}
```

Expected output section 1:

```
Can Louise drive a Harley? false
Car diesel, model Fiat 500x, number plate: 1245 HYN, purchase date 2019, with 4 wheels, index:C owner: Ann Smith driver:
Ann Smith
------------------------
Truck with 2 axles, model IvecoDaily, number plate: 5643 KOI, purchase date 2010, with 4 wheels, index:B owner: Fast
Delivery Inc driver: Ann Smith
------------------------
Motorcycle, model Harley Davidson, number plate: 0987 ETG, purchase date 2003, with 2 wheels, index:C owner: Ann Smith
driver: Ann Smith
------------------------
Motorcycle, model Harley Davidson, number plate: 0987 ETG, purchase date 2003, with 2 wheels, index:C owner: Ann Smith
driver: Anthony Johnson
People:
Anthony Johnson
---
Ann Smith owner of:
Car diesel, model Fiat 500x, number plate: 1245 HYN, purchase date 2019, with 4 wheels, index:C owner: Ann Smith driver:
Ann Smith
Motorcycle, model Harley Davidson, number plate: 0987 ETG, purchase date 2003, with 2 wheels, index:C owner: Ann Smith
driver: Anthony Johnson
---
Company:
Fast Delivery Inc (responsible: Ann Smith) owner of:
Truck with 2 axles, model IvecoDaily, number plate: 5643 KOI, purchase date 2010, with 4 wheels, index:B owner: Fast
Delivery Inc driver: Ann Smith
```

Section 2. Reading fines from text files (1.5 points)

In this section, you must develop a *FineReader* class with a *read()* static method that reads a text file with the fines line by line. The file should return a list with all the data about the fines in the file. The return type of method *read()* must be compatible with *List<Fine>*, for instance, *ArrayList<Fine>*. Remember that to use these collections you must include an *import* statement similar to the one at the beginning of the tester. The *read()* method receives a parameter that indicates the name of the text file to read, with the following structure:

Structure of the file with input data for tester 2 (available in Moodle, file `fines_radar1.txt`):
NUMBER_PLATE; FINE_TYPE; POINTS

Each line in the file has a series of fields separated by ";" (which we can assume will never appear in the content of any field). The fields in each line describe all the attributes of a fine: the number plate of the vehicle, a description of the type of fine, and the points to be subtracted from the driving license. After decomposing each line read into fields, a *Fine* object must be created and added to the output list. As the *FineReader* class is a utility class (for file reading), we shouldn't allow creating objects of that type.

Tester section 2 (available in Moodle):

```java
package pr3.traffic.fines;
import java.util.List;

public class FineTester {
  public static void main(String[] args) {
        List<Fine> fines = FineReader.read("fines_radar1.txt");

        for (Fine m : fines)
                System.out.println(m+"\n------------");
  }
}
```

Expected output section 2:

```
Fine [plate=1245 HYN, Fine type=Speeding, points=2]
------------
Fine [plate=5643 KOI, Fine type=Improper parking, points=1]
------------
Fine [plate=0987 ETG, Fine type=Driving in the opposite direction, points=12]
------------
Fine [plate=1245 HYN, Fine type=Forbidden overtaking, points=2]
------------
Fine [plate=0987 ETG, Fine type=Improper parking, points=1]
------------
```

Section 3. Driving licenses (1.5 points)

In this section we will create a class representing the driver's *license*. We will be able to associate licenses to people, and we will verify that they meet the requirements to be a driver of the different classes of vehicles the license *permits* to drive.

A license is associated with a number of points (initially 12), which may decrease when offenses are committed. The licenses must have a unique numerical identifier, created automatically by the system. A license allows driving several types of vehicles (one or more), depending on the *permit*. For simplicity, we will consider 3 types of permit: A (for driving motorcycles), B (for driving cars), and C1 (for driving trucks). The minimum age required for permits A and B is 18 years, while for C1 is 23. You must ensure that when a license is assigned to a person, s/he has the minimum age for all permits in the driving licenses. Besides, when a driver is assigned to a vehicle, the driver must have the appropriate license (otherwise, the driver remains unassigned). Finally, the number of points on a license cannot be less than 0.

The tester below contains three tests exercising some of the characteristics of the licenses. Please, add more tests that execute the classes created in more depth. This style of testing is used to create unit tests, and frameworks like JUnit (https://junit.org/) make them easy to write. JUnit will be explained in later sessions (and in PADSOF) but you can use it in this part to create the tests.

Note: Remember to make an extensible and flexible design, enabling to easily add new types of permits (with different required ages) and allowing to easily change the required ages.

Tester section 3 (available in Moodle, you must add imports and decide in which package to place this class)

```java
// Place this tester in the most suitable package, and import the necessary classes
public class TesterLicense {
  private void testYoungerThan18CannotHavePermitA() {
        Person ann = new Person("Ann Smith", 17);
        License c  = new License(PermitKind.A);
        System.out.println("Test: YoungerThan18CannotHavePermitA");
        System.out.println(c);
        System.out.println(ann.setLicense(c));           // should return false, since Ann is not 18 years old
  }
  private void testYoungerThan23CannotHavePermitC1() {
        Person ann = new Person("Ann Smith", 19);
        License c  = new License(PermitKind.A, PermitKind.C1);
        System.out.println("=================\nTest: YoungerThan23CannotHavePermitC1");
        System.out.println(c);
        System.out.println(ann.setLicense(c));           // should return false, since Ann is not 23 years old
  }
  private void testLicenseForVehicleKind() {
        Person ann = new Person("Ann Smith", 24);
        ann.setLicense(new License(PermitKind.A, PermitKind.C1));
        Car c = new Car("Fiat 500x", 2019, "1245 HYN", true, ann);
        System.out.println("=================\nTest: LicenseForVehicleKind");
        System.out.println(c);                        // Ann is not the driver, since it has no car permit
        ann.getLicense().addPermit(PermitKind.B);
        c.setDriver(ann);
        System.out.println(c);                        // Now she is
        System.out.println(ann.getLicense());         // license
  }
  public static void main(String[] args) {
        TesterLicense tap3 = new TesterLicense();
        tap3.testYoungerThan18CannotHavePermitA();
        tap3.testYoungerThan23CannotHavePermitC1();
        tap3.testLicenseForVehicleKind();
  }
}
```

Expected output:

```
Test: YoungerThan18CannotHavePermitA
License [id=0, permits=[A], points=12]
false
=================
Test: YoungerThan23CannotHavePermitC1
License [id=1, permits=[A, C1], points=12]
false
=================
Test: LicenseForVehicleKind
Car diesel, model Fiat 500x, number plate: 1245 HYN, purchase date 2019, with 4 wheels, index:C owner: Ann Smith driver:
not registered
Car diesel, model Fiat 500x, number plate: 1245 HYN, purchase date 2019, with 4 wheels, index:C owner: Ann Smith driver:
Ann Smith
License [id=2, permits=[A, C1, B], points=12]
```

Section 4. Fine processing (1.5 points)

Next, we create the functionality to process fines, subtracting the corresponding points from the license of the drivers of the fined vehicles. To do this, you must create a *FineProcessor* class that receives a list of vehicles to consider in the constructor. The *process()* method will receive the list of fines, iterate over them, and subtract the corresponding points. The processor should print the following messages:

- A message with the points each driver loses for each fine.
- A message if the driver reaches zero points.
- A message if the driver had the license suspended, or the fine causes the license to be suspended (see below).
- A message if the vehicle does not have an associated driver (and therefore, the DGT will initiate legal actions against the owner).

Keep the following in mind when processing point subtractions:

- If a license has fewer points than those that must be subtracted for the offense, it is assigned 0 points.
- If a license has zero points and its driver commits an offense, the license will be suspended (and remains with 0 points). A driver with a suspended license cannot drive, and the DGT will initiate legal actions against him.

For testing purposes, the following program should produce the output below.

Tester Section 4 (available in Moodle, you must add imports and decide in which package to place this class)

```java
public class TesterFines {
  public static void main(String[] args) {
        Person ann = new Person("Ann Smith", 30);                   // Ann
        Person anthony = new Person("Anthony Johnson", 27);
        Company fdinc = new Company("Fast Delivery Inc", ann);    // Ann is responsible for FDINC

        ann.setLicense(new License(PermitKind.B, PermitKind.C1));
        anthony.setLicense(new License(PermitKind.A));

        Vehicle fleet[] = {
                new Car("Fiat 500x", 2019, "1245 HYN", true, ann),    // Ann's car, who drives it
                new Truck("IvecoDaily", 2010, "5643 KOI", 2, fdinc), // FDINC's car
                new Motorcycle("Harley Davidson", 2003, "0987 ETG", false, anthony)};

        FineProcessor pm = new FineProcessor(Arrays.asList(fleet));
        pm.process(FineReader.read("fines_radar1.txt"));
  }
}
```

Expected output:

```
Driver Ann Smith loses 2 points
Driver Ann Smith loses 1 points
Driver Anthony Johnson loses 12 points
Driver Anthony Johnson remains with 0 points
Driver Ann Smith loses 2 points
Driver Anthony Johnson loses 1 points
License suspended for driver Anthony Johnson
```

<u>Note:</u> In this section, you must create more test cases exercising the rest of the conditions (e.g., vehicles without a declared driver).

Section 5. Expanding the original design (2.5 points)

Expand the original application design to include the concept of ITV (Technical Vehicle Inspection). All vehicles must pass an ITV periodically after a certain age, measured from the year of purchase. The periodicity of the ITV may be different for each type of vehicle:

- Cars and motorcycles from 4 years old, every two years; and from 10 years old, annually.
- Trucks up to 2 years old are exempt from ITV; from 2 to 6 years old they should pass the ITV every 2 years; from 6 to 10 years old every year and If they are more than 10 years old, every 6 months.

Each vehicle must keep a list of the ITVs it has passed, and each ITV must contain information on the date, garage of the inspection, and a comments field. The garage is defined by a name, an address and a province. Methods should be included to manage and consult the ITV status of a vehicle and to consult the time remaining until the next inspection. In addition, we want to obtain all vehicles that have passed an ITV at a certain workshop.

This new functionality has implications for other parts of your design: if the fine processor detects an infraction of a vehicle with an expired ITV, it should print a message on the screen, subtract an additional point to the driver of the vehicle, and print the infraction in a file “ITV_expired.txt” with the same format as the fine file.

<u>Note:</u> In this section, you have more freedom for your design (we do not give you a tester), but you must design testers yourself with test cases that exhaustively test this new functionality.

Submission Rules:

- A single ZIP file must be delivered with everything that is requested below, named P3_GR<group_number>_<student_name>.zip. For example, Marisa and Pedro, from group 2213, would deliver file: P3_GR2213_MarisaPedro.zip.
- The file should contain:
    - A ***src*** directory with all the Java code in its **final version of section 5**, including the test data and additional testers that you developed in the sections that require it. Please remember to modify the tester(s) of section 1 to add licenses if necessary.
    - A ***doc*** directory with the generated documentation
    - A ***txt*** directory with all the text data files used and generated in the tests
    - A PDF file with the **class diagram**, an explanation and a brief justification of the decisions that have made in the development of this assignment, the main problems that have faced and how they have solved, as well as the pending problems.