Lab2 Objects concepts in Java

Classes - Objects - Encapsulation - Attributes - Methods

Relationships between classes – Tables

This TP is a synthesis TP, object tables are used.

In Java, different packages can contain classes of the same name.

Exercice 0 Violation of the principle of encapsultation

with arrays of integers (type primitive : int)

1) Complete the Integers class written in an integers0 package.

```
public class Integers {
        // public integer constant of class initialized to 5
        // ...

        // array of integers of primitive type int
        // ...

        // attribute index of type integer storing
        // the first free place in the table
        // ...

        // default builder
        // creation of the integer table
        /*

        */

        // constructor taking an array of integers as parameters
        // coding a reference assignment
        // assigning 666 to the last box of the table
        /*

        */

        // method add of adding an integer to the first free place
        // beware of overflow in a static table
        /*

        */

        // toString method
        // the returned string must be of the form [ 1 2 3 0 0 ] for example.
        // from now this method will be preceded by the annotation @Override
        // this annotation tells the compiler that the toString method
        // is a redefinition of the one inherited from the Object class
        // the compiler verifies that the declaration is the same as
        // that of the inherited method
        @Override
        /*

        */
}
```

2) The execution of the following program gives :

```
integers1 : [ 1 2 3 0 0 ]
table :  [ 20 30 40 0 0 ]
integers2 : [ 20 30 40 0 666 ]
Modification of the element 30 with index 1 of the table
table :  [ 20 -1 40 0 666 ]
integers2 : [ 20 -1 40 0 666 ]
```

Explain the observed phenomenon. How can this be avoided?

```
package integers0;


public class TestIntegers {

    // the display must be in the form [ 1 2 3 0 0 ] for example.

public static void display(int [] tab) {
                        /*

          */
            }

            public static void main (String [] arg) {
                        Integers integers1 = new Integers();

                        integers1.add(1);
                        integers1.add(2);
                        integers1.add(3);
                        System.out.println("integers1 : " + integers1);

                        int tab [] = new int[Integers.SIZE];
                        tab[0] = 20;
                        tab[1] = 30;
                        tab[2] = 40;
                        System.out.print("table :  ");
                        display(tab);

                        Integers integers2 = new Integers(tab);
                        System.out.println("integers2 : " + integers2);

                        System.out.println("Modification of the element " + tab[1] + " of  index 1 in the table");
                        tab[1] = -1;
```

```
                System.out.print("table :  ");
                display(tab);
                System.out.println("integers2 : " + integers2);
        }
}
```

Exercice 1 NON Violation of the principle of encapsulation

With arrays of integers (type primitive : int)

1) Create an integer1 package and copy the classes of the integer0 package.

2) How to modify the Integers class of the exercise 0 in order to obtain the following execution?

```
entiers1 : [ 1 2 3 0 0 ]
tableau :  [ 20 30 40 0 0 ]
entiers2 : [ 20 30 40 0 666 ]
Modification of the element 30 of index 1 in the table
tableau :  [ 20 -1 40 0 0 ]
entiers2 : [ 20 30 40 0 666 ]
```

Exercice 2 Violation of the principle of encapsulation

With arrays of integers (type object : Integer)

1) Create an integer2 package and copy the classes of the integer0 package.

2) How to modify the Integers class of the exercise 0 in order to obtain the following execution?

```
entiers1 : [ 1 2 3 null null ]
tableau :  [ 20 30 40 null null ]
entiers2 : [ 20 30 40 null 666 ]
Modification of the element 30 of index 1 in the table
tableau :  [ 20 -1 40 null 666 ]
entiers2 : [ 20 -1 40 null 666 ]
```

Exercice 3 NON Violation of the principle of encapsulation

with arrays of integers (type object : Integer)

1) Create an integer package3 and copy the classes of the integer package2.

2) How to modify the Integers class of the exercise 2 in order to obtain the next execution?

```
entiers1 : [ 1 2 3 null null ]
tableau :  [ 20 30 40 null null ]
entiers2 : [ 20 30 40 null 666 ]
Modification of the element 30 of index 1 in the table
tableau :  [ 20 -1 40 null null ]
entiers2 : [ 20 30 40 null 666 ]
```

Exercice 4 Modeling an odometer

An odometer is composed of a totalizer record and a partial record. These two recordings keep the distances travelled as a real number. When a counter is created, the value of these two records is always zero. The partial record can be reset to zero (but not to any other value), when it reaches 1,000km, it is automatically reset to 0. Apart from this operation, the only way to change the value of the two records is to add kilometers while driving.

- Define in Java a Counter class with :
  - two instance attributes: totalizer and partial
  - a class constant, initialized at 1000
  - a constructor without parameter
  - the public read access method for both recordings
  - the public write access method for partial registration
  - a public method **resetPartial** to reset to zero the field partial
  - a public method **add**, which increments the two fiels : number of

    kilometers (a real) specified as input parameter
  - the toString method, which returns a textual representation of a counter and its state according to the following example :

    counter = [totalizer = 500 | partial = 108]

Tests: activate tests on the counters in the provided test program.

######## TESTS of COUNTERS ########

```
compteur = [Totalisateur = 0 | Partiel = 0];
compteur = [Totalisateur = 200 | Partiel = 200];
compteur = [Totalisateur = 500 | Partiel = 500];
compteur = [Totalisateur = 500 | Partiel = 0];
compteur = [Totalisateur = 650 | Partiel = 150];
```

Exercice 5 Modeling a vehicle

A vehicle consists of a license plate number, meter, tank gauge, and mileage. Vehicles will be represented by instances of the Vehicle class.

The registration number of a new vehicle is determined by the value of a register common to all vehicles; the value of this register is automatically incremented by 1 each time a new vehicle is created. This register must not be accessible outside the Vehicle class. The registration number of a vehicle must also not be accessible outside the class, but its value can be obtained by a reading method from any class.

The odometer is the one modeled in Part I. It is created during the instantiation of a vehicle. A reference to this counter is obtained by a getCounter method of the Vehicle class.

Tank capacity is 50.0 liters for all vehicles and does not vary. It must not be accessible from outside the class.

The fuel tank gauge indicates how much fuel is left in the tank. It is specific to each vehicle. It must not be accessible from outside the class, but its value can be obtained by a getJauge reading method from any class. Each time the tank is filled, this gauge is reset to the value of the tank capacity; it is possible to fill only a fraction of the tank.

The consumption of a vehicle (number of liters of fuel per 100 kilometers travelled) is specific to each vehicle and is fixed at the creation of the vehicle. This consumption must not be accessible from outside the class.

The three operations that can be done on a vehicle are: filling up the gas tank, refueling and driving a certain distance. While driving you consume fuel, so you can only drive the distance allowed by the tank and the consumption of the vehicle; the meter is only incremented by the distance actually travelled. We decide to drive until the tank is empty, then we fill up the tank.

- According to the above information, define in Java a Vehicle class including :
  - the class attributes, the class constants
  - instance attributes
  - a constructor whose parameter is consumption
  - methods for read access to instance attributes, according to the degree of accessibility prescribed above
  - the methods : fillingTheTank, taking care not to exceed the capacity of the tank, and fillingToCapacity.
  - a method, drive, which takes as a parameter the distance you want to drive and returns the distance actually travelled (distances are specified by a real number).

- the toString method, which returns a textual representation of a vehicle and its state, like in the following example :

  Vehicle 45 : counter = [totalizer = 500 | partial = 108] ; tank gauge = 9.00533

Tests: activate tests on vehicle in the given test program

```
######## TESTS DES VEHICULES ########
Vehicule 0 : compteur = [Totalisateur = 0 | Partiel = 0];; jauge = 50.0

100.0
Le vehicule 0 a parcouru 100.0
Vehicule 0 : compteur = [Totalisateur = 100 | Partiel = 100];; jauge = 44.7

Le vehicule 0 a parcouru 300.0
Vehicule 0 : compteur = [Totalisateur = 400 | Partiel = 400];; jauge = 28.8

Le vehicule 0 a parcouru 543.39
Vehicule 0 : compteur = [Totalisateur = 943 | Partiel = 943];; jauge = 50.0

Le vehicule 0 a parcouru 200.0
Vehicule 0 : compteur = [Totalisateur = 1143 | Partiel = 143];; jauge = 39.4

Vehicule 0 : compteur = [Totalisateur = 1683 | Partiel = 683];; jauge = 10.77

Vehicule 0 : compteur = [Totalisateur = 1683 | Partiel = 683];; jauge = 50.0

Vehicule 0 : compteur = [Totalisateur = 1943 | Partiel = 943];; jauge = 36.22

Votre réservoir a une capacité insuffisante pour mettre 600 litres d'essence
Vehicule 0 : compteur = [Totalisateur = 1943 | Partiel = 943];; jauge = 36.22

Votre réservoir a une capacité insuffisante pour mettre 16 litres d'essence
Vehicule 0 : compteur = [Totalisateur = 1943 | Partiel = 943];; jauge = 36.22

Vehicule 1 : compteur = [Totalisateur = 0 | Partiel = 0];; jauge = 50.0
```