



Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

PROGRAMMATION ET TP

T1A

RÉSEAU ET SÉCURITÉ

S14 Encapsulation package

ROTEN MARC

2017/2018

Table des matières

0	Introduction	2
1	Exercice 1 Date	2
1.1	Version A avec deux attributs entiers	2
1.2	Version B avec 1 seul attribut entier	4
1.3	Question : Avantages de l'un par rapport à l'autre ?	6
2	Exercice 2 Rational Number	6
2.1	Question	8
3	Conclusion	8

0 Introduction

Avec ce travail, nous allons aborder les aspects d'encapsulation qui sont indispensables dans tout langage de programmation. Voir comment créer un objet, y accéder, poser des exceptions et récupérer ces valeurs dans notre main pour pouvoir les print.

1 Exercice 1 Date

Écrire une classe `Date` qui respecte le principe d'encapsulation et qui garantit la cohérence des données (en générant une exception si nécessaire). Supposer une année non bissextile (365 jours).

```
public class Date { // Spécification
    public Date(int day, int month) throws Exception;
    public int dayOfMonth();
    public int month();
}
```

L'arsenal de la librairie standard pour gérer les dates (p. ex. `java.util.Calendar`) n'est **pas** disponible.

Écrire deux implémentations différentes de cette classe `Date`, **sans rien changer** dans la spécification ci-dessus (exactement ces 3 éléments publics avec ces signatures-là, tout le reste étant privé); les classes seront placées dans deux packages distincts `s14.va` et `s14.vb` :

- a) une variante avec 2 attributs entiers (jour et mois)
- b) une variante avec un seul attribut entier (numéro du jour dans l'année [1..365]).

C'est un exercice important pour comprendre que la représentation interne n'est pas forcément calquée sur la spécification. L'utilisateur de la classe n'a pas à connaître ce genre de choix d'implémentation; le fait que certaines données subissent des conversions n'aura pas d'influence sur la manière d'utiliser le composant.

Dans le package `s14.test`, créer ensuite deux classes `DateTestA` et `DateTestB`.

Écrire dans chacune d'elles une méthode `main()` (avec un corps strictement identique) qui teste la variante a) respectivement la variante b) de votre classe `Date` en créant quelques dates et en affichant le résultat retourné par les deux méthodes de cette classe.

Donner enfin un argument en faveur de **chacune** de ces variantes (en quoi serait-elle "meilleure" ?).

1.1 Version A avec deux attributs entiers

Version avec la méthode avec deux attributs entiers, jour et mois

Mon main

```
public class DateWithTab {
    private int day, month;
    public DateWithTab(int day, int month) throws Exception {
        final int[] monthTab = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
        if(monthTab[month-1] < day) {
            throw new IOException("votre mois comporte trop de
                jours");
        }
        if(day < 1 || day > 31 || month < 1 || month > 12) {
```

```

        throw new IOException("vos entres ne sont pas correctes");
    }
    this.day=day;
    this.month=month;
};
public int getDayOfMonth() {
    return this.day;
}
public int getMonth() {
    return this.month;
};
}

```

ma classe DateWithTab

```

public class testDateWithTab {
    public static void main(String[] args) {
        try {
            DateWithTab Date1 = new DateWithTab(1,12);
            System.out.println("la date est "+Date1.getDayOfMonth()+"
                               / "+Date1.getMonth());
        }catch(Exception e){
            e.printStackTrace();
        }
        try {
            DateWithTab Date2 = new DateWithTab(29,2);
            System.out.println("la date est "+Date2.getDayOfMonth()+"
                               / "+Date2.getMonth());
        }catch(Exception e){
            e.printStackTrace();
        }
        try {
            DateWithTab Date3 = new DateWithTab(14,8);
            System.out.println("la date est "+Date3.getDayOfMonth()+"
                               / "+Date3.getMonth());
        }catch(Exception e){
            e.printStackTrace();
        }
        try {
            DateWithTab Date4 = new DateWithTab(0,12);
            System.out.println("la date est "+Date4.getDayOfMonth()+"
                               / "+Date4.getMonth());
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}

```

```

la date est 1 / 12
java.io.IOException: votre mois comporte trop de jours
la date est 14 / 8
    at s14va.DateWithTab.<init>(DateWithTab.java:8)
    at s14va.testDateWithTab.main(testDateWithTab.java:14)
java.io.IOException: vos entrées ne sont pas correctes
    at s14va.DateWithTab.<init>(DateWithTab.java:11)
    at s14va.testDateWithTab.main(testDateWithTab.java:26)

```

on voit que pour la première date, tout ok
la deuxième date, le 29 février, l'erreur apparait correctement
la 3ème date est OK
la dernière date 0 décembre, on a la bonne erreur qui apparaît

1.2 Version B avec 1 seul attribut entier

classe contenant mon main

```

public class TestDate {
    public static void main(String[] args) {
        try {
            Date Date1 = new Date(1,12);
            System.out.println("Date 1 nous sommes actuellement le
                "+Date1.getNumberOfDayInTheYear()+" me jour de
                l'anne");
        } catch (Exception e){
            e.printStackTrace();
        }
        try {
            Date Date2 = new Date(29,2);
            System.out.println("Date 2 nous sommes actuellement le
                "+Date2.getNumberOfDayInTheYear()+" me jour de
                l'anne");
        } catch (Exception e){
            e.printStackTrace();
        }
        try {
            Date Date3 = new Date(14,8);
            System.out.println("Date 3 nous sommes actuellement le
                "+Date3.getNumberOfDayInTheYear()+" me jour de
                l'anne");
        } catch (Exception e){
            e.printStackTrace();
        }
        try {
            Date Date4 = new Date(0,12);
            System.out.println("Date 4 nous sommes actuellement le
                "+Date4.getNumberOfDayInTheYear()+" me jour de
                l'anne");
        }
    }
}

```

```

    }catch(Exception e){
        e.printStackTrace();
    }
}
}

```

classe Date Version B

```

public class Date {
    int day,month,numberofDayInTheYear;
    final int[]monthTab = {31,28,31,30,31,30,31,31,30,31,30,31};

    public Date(int day, int month) throws Exception{
        int numberofDayInTheYear =0;
        if(monthTab[month-1]<day) {
            throw new IOException("votre mois comporte trop de
                jours");
        }
        if(day<1||day>31||month<1||month>12) {
            throw new IOException("vos entres ne sont pas correctes
                ");
        }
        numberofDayInTheYear = computeMonth(month)+day;
        this.day=day;
        this.month=month;
        this.numberofDayInTheYear=numberofDayInTheYear;
    };
    public int getDayOfMonth() {
        return this.day;
    }
    public int getMonth() {
        return this.month;
    };
    public int getNumberofDayInTheYear() {
        return this.numberofDayInTheYear;
    };
    private int computeMonth(int month) {
        int res = 0;
        for(int i=0;i<month-1;i++) {
            res+=monthTab[i];
        }
        return res;
    }
}

```

résultat à la console

```

la date est 1 / 12
java.io.IOException: votre mois comporte trop de jours
la date est 14 / 8
    at s14va.DateWithTab.<init>(DateWithTab.java:8)
    at s14va.testDateWithTab.main(testDateWithTab.java:14)
java.io.IOException: vos entrées ne sont pas correctes
    at s14va.DateWithTab.<init>(DateWithTab.java:11)
    at s14va.testDateWithTab.main(testDateWithTab.java:26)

```

1.3 Question : Avantages de l'un par rapport à l'autre ?

Je trouve que les deux implémentations ont du sens, de toute façon l'utilisateur n'a pas à connaître le genre de l'implémentation. Il passe toujours en paramètre le mois et le jour.

2 Exercice 2 Rational Number

Écrire une classe `RationalNumber` pour représenter des nombres rationnels, avec les opérations prévues par la spécification suivante (voir également l'exemple d'utilisation).

Commencer par définir les attributs, puis écrire le corps du constructeur et le corps de chaque méthode. Il n'est pas demandé de simplifier les fractions.

Si les attributs sont déclarés `private`, la méthode `add(RationalNumber a)` a-t-elle accès aux attributs de l'objet `a` ?

Spécification

```

public class RationalNbTest {
    public static void main(String[] args) {
        RationalNumber x, y;
        x = new RationalNumber(3, 4);
        y = new RationalNumber(9, 7);
        x.multiply(y);
        y.add(4);
        System.out.println("As float : " + x.toFloat());           // 0.9642
        System.out.println("As string: " + x.toString());          // "27/28"
        System.out.println(x + " = " + y + " ? " + x.equals(y));
        System.out.println(x + " < " + y + " ? " + x.lessThan(y));
    }
}

```

```

public class RationalNumber {
    public RationalNumber(int n, int d);

    public void multiply(RationalNumber a);
    public void divide (RationalNumber a);
    public void add (RationalNumber a);
    public void subtract(RationalNumber a);
    public void multiply(int i);
    public void add (int i);
    public boolean lessThan(RationalNumber a);
    public boolean equals (RationalNumber a);
    public float toFloat ();
    public String toString();
}

```

Exemple d'utilisation

classe contenant mes méthodes en page suivante.

```

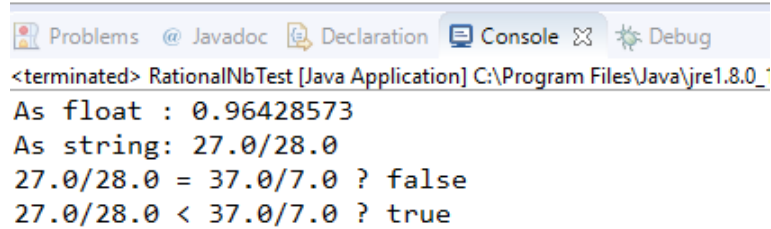
public class RationalNumber {
    float numerator,denominator;
    public RationalNumber(int n, int d){
        this.numerator=n;
        this.denominator=d;
    }
    public void multiply(RationalNumber a) {
        this.numerator *=a.numerator;
        this.denominator*=a.denominator;
    }
    public void divide (RationalNumber a) {
        this.numerator /=a.numerator;
        this.denominator/=a.denominator;
    }
    public void add (RationalNumber a) {
        this.denominator *= a.denominator;
        this.numerator = this.numerator *a.denominator +
            a.numerator*this.denominator;
    }
    public void subtract(RationalNumber a) {
        this.denominator *= a.denominator;
        this.numerator = this.numerator*a.denominator -
            a.numerator*this.denominator;
    }
    public void multiply(int i) {
        this.numerator*=i;
    }
    public void add (int i) {
        this.numerator = this.numerator + this.denominator*i;
    }
    public boolean lessThan(RationalNumber a) {
        if((this.numerator/this.denominator)<(a.numerator/a.denominator))
        {
            return true;
        }else return false;
    }
    public boolean equals (RationalNumber a) {
        if((this.numerator/this.denominator)==(a.numerator/a.denominator))
        {
            return true;
        }else return false;
    }
    public float toFloat (){
        return this.numerator/this.denominator;
    }
    public String toString(){
        String A;
        A = this.numerator+"/"+denominator;
    }
}

```



```
        return A;
    }
}
```

résultat à la console



The screenshot shows the 'Console' tab of a Java IDE. The title bar indicates the application is 'RationalNbTest [Java Application]' running on 'C:\Program Files\Java\jre1.8.0_1'. The output text is as follows:

```
<terminated> RationalNbTest [Java Application] C:\Program Files\Java\jre1.8.0_1
As float : 0.96428573
As string: 27.0/28.0
27.0/28.0 = 37.0/7.0 ? false
27.0/28.0 < 37.0/7.0 ? true
```

2.1 Question

Si les attributs sont déclarés private, la méthode `add(RationalNumber a)` a-t-elle accès aux attributs de l'objet `a`.

oui, car les pointeurs d'ou leur noms pointent sur les objets, il est même conseillé de mettre les attributs en Private pour éviter que nimportequi n'aie accès à ces attributs. Pour ccéder à ces attributs, il est conseillé de passer par des Getters.

3 Conclusion

Grâce à ce Travail, on a pu mieux comprendre les aspects concernant l'encapsulation et la manipulation de nos propre objets, tout en posant des conditions pour ne pas avoir d'incohérences dans nos données