4TYE707Y Programmation orientée objet novembre 2018

Corrigé

Exercice 1

On considère la classe `Relation` suivante, qui met en relation deux objets `a` et `b` :

```
public class Relation {
    private Object a;
    private Object b;
    public Relation(Object a, Object b) {
        this.a = a;
        this.b = b;
    }

    public Object a() {
        return a;
    }

    public Object b() {
        return b;
    }
}
```

Question 1 Modifier le code pour que la classe `Relation` soit paramètrée par un type `T` de façon que les deux objets `a` et `b` soient tous les deux de type `T`.

Réponse

```
public class Relation<T> {
    private T a;
    private T b;

    public Relation( T a, T b) {
        this.a = a;
        this.b = b;
    }

    public T a() {
        return a;
    }

    public T b() {
        return b;
    }
}
```

Question 2 Modifier le code pour que les deux objets `a` et `b` soient tous les deux d'un type `T` qui soit une sous-classe de la classe `Number` (comme `Integer`, `Float` ...).

Réponse

```
public class Relation<T extends Number> {
```

```
    private T a;
    private T b;

    public Relation(T a, T b) {
        this.a = a;
        this.b = b;
    }

    public T a() {
        return a;
    }

    public T b() {
        return b;
    }
}
```

Exercice 2

On considère l'interface `Evenement`. Pour simplifier, une date est codée par un entier :

```
public interface Evenement {
    int date();
    void changerDate(int date);
    String description();
    void changerDescription(String descr);
}
```

Question 1 Écrire la classe `Reunion` qui implémente `Evenement`. Une réunion sera de plus caractérisée par un lieu (String).
Exemple de code de test :

```
Evenement r = new Reunion(275, "seminaire", "Amphi␣LaBRI");
```

Question 2 Modifier le code de `Reunion` pour que le code

```
Evenement r = new Reunion(275, "Seminaire", "Amphi␣LaBRI");
System.out.println(r);
```

affiche

```
Seminaire, date : 275, lieu : Amphi Labri
```

Question 3 Modifier le code de `Reunion` pour que deux réunions soient égales si elles ont lieu le même jour et ont le même descriptif.
Quelle autre méthode faut-il redéfinir ? Donner le code correspondant.

Réponse aux questions 1, 2 et 3
Pour la question 2, il faut redéfinir la méthode `String toString()`.
Pour la question 3, il faut redéfinir les méthodes `boolean equals(Object o)` et `int hashCode()`.

```
public class Reunion implements Evenement {
    private int date;
    private String description;
    private String lieu;

    public Reunion(int date, String description, String lieu) {
        this.date = date;
        this.description = description;
        this.lieu = lieu;
    }

    public int date() {
        return date;
    }

    public void changerDate(int date) {
        this.date = date;
    }

    public String description() {
        return description;
    }

    public void changerDescription(String description) {
        this.description = description;
    }

    public String lieu() {
        return lieu;
    }

    public void changerLieu(String lieu) {
        this.lieu = lieu;
    }

    // Question 2
    public String toString() {
        return description + ", date : " + date + ", lieu : " + lieu;
    }

    // Question 3
    public boolean equals(Object o) {
        if (! (o instanceof Reunion))
            return false;
        Reunion r = (Reunion)o;
        return description.equals(r.description) && date == r.date;
    }

    public int hashCode() {
        return Objects.hash(description, date);
```

```
    }
}
```

Question 4 Modifier le code de `Reunion` pour que si l'on essaie de créer une réunion avec une date de valeur négative, une exception `DateNegativeException` soit levée. Donner également le code de `DateNegativeException`.

Réponse

```
public class Reunion implements Evenement {
    private int date;
    private String description;
    private String lieu;

    public Reunion(int date, String description, String lieu) throws DateNegativeException
        if (date < 0)
            throw new DateNegativeException(date);
        this.date = date;
        this.description = description;
        this.lieu = lieu;
    }

    public int date() {
        return date;
    }

    // Remarque : il faut également rajouter le "throws DateNegativeException"
    // dans la déclaration de changerDate(int date) dans l'interface Evenement
    public void changerDate(int date) throws DateNegativeException {
        if (date < 0)
            throw new DateNegativeException(date);
        this.date = date;
    }

    // Le reste est inchangé ...
}

public class DateNegativeException extends Exception {
    private int date;

    public DateNegativeException(int date) {
        this.date = date;
    }

    public int date() {
        return date;
    }

    public String getMessage() {
        return "date negative : " + date + " impossible";
    }
```

```
}
```

Question 5 On considère une autre classe, `Anniversaire` qui implémente `Evenement`. Un objet de cette classe est caractérisé par la date et le nom de la personne (description). Il n'est pas demandé d'écrire cette classe.
Comment factoriser le code d' `Anniversaire` et `Reunion` ? Écrire le code factorisé et le nouveau code de `Reunion`.

Réponse
On crée une classe (abstraite) pour factoriser le code. Il faut penser à factoriser également `equals(Object o)` et `toString()`

```
public abstract class EvenementAbstrait implements Evenement {
    private int date;
    private String description;

    public EvenementAbstrait(int date, String description) throws DateNegativeException {
        if (date < 0)
            throw new DateNegativeException(date);
        this.date = date;
        this.description = description;
    }

    public int date() {
        return date;
    }

    public void changerDate(int date) throws DateNegativeException {
        if (date < 0)
            throw new DateNegativeException(date);
        this.date =  date;
    }

    public String description() {
        return description;
    }

    public void changerDescription(String description) {
        this.description =  description;
    }

    public String toString() {
        return description + ", date : " + date;
    }

    public boolean equals(Object o) {
        if (! (o instanceof EvenementAbstrait))
            return false;
        EvenementAbstrait r = (EvenementAbstrait)o;
        return description.equals(r.description) && date == r.date;
```

```
    }

    public int hashCode() {
        return Objects.hash(description, date);
    }
}



public class Reunion extends EvenementAbstrait {
    private String lieu;

    public Reunion(int date, String description, String lieu) throws DateNegativeException
        super(date, description);
        this.lieu = lieu;
    }

    public String lieu() {
        return lieu;
    }

    public void changerLieu(String lieu) {
        this.lieu = lieu;
    }

    public String toString() {
        return super.toString() + ", lieu : " + lieu;
    }

    public boolean equals(Object o) {
        if (! (o instanceof Reunion))
            return false;
        return super.equals(o);
    }
}
```

Exercice 3

Qu'affiche le code suivant si on exécute la méthode `main()` de la classe `B` ?

```
class A {
    public String f() {
        return "A";
    }
}

class B extends A {
    private A a;

    public B(A a) {
```

```
        this.a = a;
    }

    public String f() {
        return super.f() + "B" + a.f();
    }

    public static void main(String[] args) {
        A a = new A();
        System.out.println(a.f());
        A a1 = new B(a);
        System.out.println(a1.f());
        A a2 = new B(a1);
        System.out.println(a2.f());
    }
}
```

Réponse

L'affichage sera :

```
A
ABA
ABABA
```

En effet,

- `a.f()` appellera la méthode `f()` de la classe `A`, qui affiche `A`.
- `a1.f()` appellera la méthode `f()` de la classe `B` qui elle-même appelera d'abord la méthode `f()` de sa sur-classe (donc celle de la classe `A`) qui affiche `A`, puis affichera `B`, puis appellera `a.f()` qui affiche `A`.
- `a2.f()` appellera la méthode `f()` de la classe `B` qui elle-même appelera d'aborde la méthode `f()` de sa sur-classe (donc celle de la classe `A`) qui affiche `A`, puis affichera `B`, puis appellera `a1.f()` qui affiche `ABA`.