université de BORDEAUX

4TYE707U
Programmation orientée objet
Master MIAGE

Examen du 13 décembre 2017, durée 3h
Corrigé

Exercice - Itération

Question 1 Dans une classe `Iterateurs`, écrire une méthode de classe `public static <E> Iterator<E> iterateurTableauAvecTrous(E [] t)` qui retourne un itérateur *it* dont les éléments sont les éléments du tableau *t* différents de la valeur `null`. La méthode `remove` aura pour effet de mettre à `null` la valeur du dernier élément retourné. La méthode `forEachRemaining` exécutera son code par défaut.

Réponse

```
public class Iterateurs {
    public static <E> Iterator<E> iterateurTableauAvecTrous(E[] t) {
        return new Iterator<E>() {
            int next = 0;
            int last = -1;

            public boolean hasNext() {
                while (next < t.length && t[next] == null)
                    next++;
                return next < t.length;
            }

            public E next() {
                if (hasNext()) {
                    last = next;
                    return t[next++];
                } else
                    throw new NoSuchElementException();
            }

            public void remove() {
                if (last == -1 || t[last] == null)
                    throw new IllegalStateException();
                t[last] = null;
            }
        };
    }
}
```

Question 2 Rajouter dans la classe `Iterateurs` une méthode de classe `iterateurTableauAvecFiltre(E [] t, Predicat<E> p)` prenant en paramètre un tableau et une instance de `Predicat`. L'itérateur instancié par `iterateurTableauAvecFiltre` ne renverra que les valeurs du tableau `t` pour lesquelles `p.predicat` renvoie `true`. Comme dans la question précédente, la méthode `remove` mettra à `null` le dernier élément du tableau retourné par l'itérateur.

Réponse

```
public class Iterateurs {

    public static <E> Iterator<E> iterateurTableauAvecFiltre(E[] t, Predicat<E> p) {
        return new Iterator<E>() {
            int next = 0;
            int last = -1;

            public boolean hasNext() {
                while (next < t.length && !p.predicat(t[next]))
                    next++;
                return next < t.length;
            }

            public E next() {
                if (hasNext()) {
                    last = next;
                    return t[next++];
                } else
                    throw new NoSuchElementException();
            }

            public void remove() {
                if (last == -1 || t[last] == null)
                    throw new IllegalStateException();
                t[last] = null;
            }
        };
    }
}
```

Question 3 Créer dans la classe `Iterateurs` une constante de classe `EST_NON_NUL` de type `Predicat<Object>` et dont la méthode `predicat` renvoie `true` si l'élément passé en paramètre est non nul. Utiliser si possible une lambda-expression pour implémenter `EST_NON_NUL` .

Réponse

```
public static final Predicat<Object> EST_NON_NUL = o -> o != null;
```

Question 4 Si l'on écrit le code suivant :

```
String[] ts = { "a", null, "c" };
Iterator<String> it = Iterateurs.iterateurTableauAvecFiltre(ts, EST_NON_NUL);
```

l'erreur de compilation suivante apparait :

```
Type mismatch: cannot convert from Iterator<Object> to Iterator<String>
```

Modifier la signature de la méthode `iterateurTableauAvecFiltre` pour éviter ce problème. Donner le nouveau code d'`iterateurTableauAvecTrous` utilisant `iterateurTableauAvecFiltre`.

Réponse

```
public static <E> Iterator<E> iterateurTableauAvecFiltre(E[] t, Predicat<? super E> p) {...}
```

```
public static <E> Iterator<E> iterateurTableauAvecTrous(E [] t) {
        return iterateurTableauAvecFiltre(t, EST_NON_NUL);
}
```

Problème - Véhicule

Question 1 Modifier la classe `VehiculeImpl` de façon à ce que l'instruction

```
System.out.print(v);
```

où `v` est une instance de `VehiculeImpl` affiche sur la sortie standard le résultat suivant :
`Vehicule (`x`, `y`), vitesse = `s`, direction = `d
où x et y sont les coordonnées de la position du véhicule, s sa vitesse et d sa direction.

Réponse

Il faut redéfinir la méthode `toString()` :

```
public String toString() {
    return "Vehicule (" + position.getX() + ", " + position.getY() +
    ") vitesse = " + vitesse + ", direction = " + direction;
}
```

Question 2 Proposer une implémentation de `VehiculeAvecAcceleration` utilisant la classe `VehiculeImpl` par héritage. Pour la méthode `accelerer(double acceleration)`, la vitesse sera modifiée de la valeur du paramètre `acceleration` qui pourra être positive ou négative. La vitesse ne pourra toutefois ni descendre en dessous de 0, ni dépasser la vitesse maximum du véhicule.

Réponse

```
public class VehiculeAvecAcceleration extends VehiculeImpl {

    public VehiculeAvecAcceleration(double vitesseMaximum) {
        super(vitesseMaximum);
    }

    public void accelerer(double acceleration) {
        changerVitesse(Math.max(0, vitesse + acceleration));
    }
}
```

Question 3 Ecrire une classe `SurfaceRectangle` qui implémente `Surface`. Cette surface sera composée des points d'un rectangle dont la coordonnée haut-gauche, la largeur et la hauteur seront donnés à la construction. A noter que deux points inclus dans un même rectangle sont toujours connexes.

Réponse

```
public class SurfaceRectangle implements Surface {

    private double x, y, largeur, hauteur;

    public SurfaceRectangle(double x, double y, double largeur, double hauteur) {
        this.x = x;
```

```
        this.y = y;
        this.largeur = largeur;
        this.hauteur = hauteur;
    }

    private void testePoint(Point2D p) {
        if (! contient(p)) {
            throw new IllegalArgumentException(p + " n'est pas dans le rectangle");
        }
    }

    public boolean sontConnexes(Point2D p1, Point2D p2) {
        testePoint(p1);
        testePoint(p2);
        return true;
    }

    public boolean contient(Point2D p) {
        return p.getX() >= x && p.getX() <= x + largeur && p.getY() >= y && p.getY() <= y + haut
    }
}
```

Question 4 Modifier la classe `SurfaceRectangle` pour que deux de ses instances soient égales si elles ont le même point haut-gauche, la même largeur et la même hauteur. *Indication : on pensera à bien redéfinir toutes les méthodes nécessaires héritées de la classe* ***`java.lang.Object`***.

Réponse

Il faut redéfinir la méthode `equals(Object o)` ainsi que la méthode `hashCode()`.

```
public boolean equals(Object o) {
        if (!(o instanceof SurfaceRectangle))
                return false;
        SurfaceRectangle sr = (SurfaceRectangle) o;
        return sr.x == x && sr.y == y && sr.hauteur == hauteur && sr.largeur == largeur;
}

public int hashCode() {
        return Objects.hash(x, y, hauteur, largeur);
}
```

Question 5 Proposer une classe `SurfaceAvecVehicule` qui implémente `Surface` et dont les instances sont créées à partir d'un véhicule et d'une surface déjà existants, fournis à la construction. Cette classe contiendra également une méthode `void deplacerVehicule(duree)` qui appellera la méthode `rouler(double duree)` du véhicule passé en paramètre. Si le véhicule sort de la surface lors d'un déplacement, une exception `AccidentException` sera levée. Si la position du véhicule n'est pas incluse dans la surface lors de sa création, une exception `IllegalArgumentException` sera levée.

Réponse

```
public class SurfaceAvecVehicule implements Surface {

    private Vehicule vehicule;
    private Surface surface;

    public SurfaceAvecVehicule(Surface s, Vehicule v) {
```

```
        if (!s.contient(v.position()))
            throw new IllegalArgumentException();
        this.vehicule = v;
        this.surface = s;
    }

    public boolean contient(Point2D p) {
        return surface.contient(p);
    }

    public boolean sontConnexes(Point2D p1, Point2D p2) {
        return surface.sontConnexes(p1, p2);
    }

    public void deplacerVehicule(double duree) throws AccidentException {
        vehicule.rouler(duree);
        if (!surface.contient(vehicule.position()))
            throw new AccidentException(this, vehicule);
    }
}
```

Question 6 Proposer une implémentation de la classe `AccidentException`.

Réponse

```
public class AccidentException extends Exception {

    private Surface surface;
    private Vehicule vehicule;

    public AccidentException(Surface s, Vehicule v) {
        super("Accident !!! Le véhicule " + v + " a quitté sa surface " + s);
        this.surface = s;
        this.vehicule = v;
    }

    public Surface surface() {
        return surface;
    }

    public Vehicule vehicule() {
        return vehicule;
    }
}
```

Question 7 Montrer que l'implémentation actuelle n'est pas satisfaisante car un véhicule inscrit dans une surface peut être déplacé sur une position à l'extérieur de cette surface sans que l'exception `AccidentException` soit levée.

Réponse

En effet, il est possible d'appeler directement la méthode `rouler(double duree)` du véhicule passé en paramètre à la création d'une instance de `SurfaceAvecVehicule`. De cette façon, le véhicule pourra quitter la surface sans qu'une exception soit levée.

Question 7 Proposer une classe `VehiculeObservable` qui hérite d'`Observable` et implémente `Vehicule`. Utiliser la classe `VehiculeImpl` par délégation pour implémenter les méthodes de `Vehicule`.

Réponse

```
public class VehiculeObservable extends Observable implements Vehicule {

    private VehiculeImpl delegue;

    public VehiculeObservable(double vitesseMaximum) {
        delegue = new VehiculeImpl(vitesseMaximum);
    }

    public double vitesse() {
        return delegue.vitesse();
    }

    public double direction() {
        return delegue.direction();
    }

    public Point2D position() {
        return delegue.position();
    }

    public void changerDirection(double direction) {
        delegue.changerDirection(direction);
    }

    public void changerVitesse(double vitesse) {
        delegue.changerVitesse(vitesse);
    }

    public void rouler(double duree) {
        delegue.rouler(duree);
        setChanged();
        notifyObservers();
    }

    public String toString() {
        return delegue.toString();
    }
}
```

Question 8 Proposer une nouvelle version de la classe `SurfaceAvecVehicule` qui implémente l'interface `Observer` et qui sera instanciée en utilisant une instance de `VehiculeObservable`. Le véhicule sera bougé en utilisant directement ses méthodes (en utilisant sa propre référence) et non en passant par la surface. La nouvelle classe `SurfaceAvecVehicule` n'implémentera que la méthode `update` et lèvera une exception `IllegalStateException` qui étend la classe `RuntimeException`, en cas de sortie du véhicule de la surface. Justifier en deux ou trois lignes pourquoi il n'est pas possible d'utiliser la classe `AccidentException`.

Réponse

```
public class SurfaceAvecVehicule implements Observer {

    private VehiculeObservable vehicule;
    private Surface surface;
```

```
    public SurfaceAvecVehicule(Surface s, VehiculeObservable v) {
        if (!s.contient(v.position()))
            throw new IllegalArgumentException();
        this.vehicule = v;
        v.addObserver(this);
        this.surface = s;
    }

    public void update(Observable o, Object arg) {
        if (!surface.contient(vehicule.position()))
            throw new IllegalStateException();
    }
}
```

Il n'est pas possible d'utiliser la classe `AccidentException` dans la méthode `update` car on ne peut pas rajouter une exception si elle n'est pas déclarée dans l'interface implémentée (ici `java.util.Observer`).

Question 9 Modifier la classe `VehiculeObservable` pour que si une exception est levée lors de l'exécution de la méthode `rouler(double duree)`, le véhicule reste à sa position d'origine.

Réponse

```
public class VehiculeObservable extends Observable implements Vehicule {

    private static class VehiculeDelegue extends VehiculeImpl {
        public VehiculeDelegue(double vitesseMaximum) {
            super(vitesseMaximum);
        }

        public void changerPosition(Point2D p) {
            position.setLocation(p);
        }
    }

    private VehiculeDelegue delegue;

    public VehiculeObservable(double vitesseMaximum) {
        delegue = new VehiculeDelegue(vitesseMaximum);
    }

    public double vitesse() {
        return delegue.vitesse();
    }

    public double direction() {
        return delegue.direction();
    }

    public Point2D position() {
        return delegue.position();
    }

    public void changerDirection(double direction) {
        delegue.changerDirection(direction);
```

```
    }

    public void changerVitesse(double vitesse) {
        delegue.changerVitesse(vitesse);
    }

    public void rouler(double duree) {
        Point2D p = (Point2D)delegue.position().clone();
        delegue.rouler(duree);
        setChanged();
        try {
            notifyObservers();
        } catch (Exception e) {
            delegue.changerPosition(p);
        }
    }

    public String toString() {
        return delegue.toString();
    }
}
```