université de BORDEAUX

4TYE707U
Programmation orientée objet
Master MIAGE

Examen du 13 décembre 2017, durée 3h
Aucun document autorisé

Justifier clairement et succinctement vos choix, et décrivez brièvement vos algorithmes lorsque vous le jugez nécessaire.

Toute réponse difficile à comprendre sera considérée comme fausse.

Vous pouvez rajouter du code (méthode, donnée ou classe) non demandé à chaque fois que vous le jugerez nécessaire. Écrivez entièrement ce code, en particulier dans le cas d'exceptions, de classes abstraites ou d'interfaces.

Vous pouvez toujours considérer une classe ou une méthode demandée dans une question précédente comme disponible, même si vous n'avez pas traité cette question. Il vous est également conseillé de lire le sujet entièrement avant de commencer votre travail.

Si vous souhaitez utiliser une classe (ou une méthode particulière) de l'API Java dont vous ne vous souvenez pas du nom, donnez-lui un nom explicite et précisez-le clairement sur votre copie.

Vous pouvez à chaque fois que vous le jugez compréhensible utiliser des abréviations à la place des noms complets de classe et méthode ou encore '...' pour remplacer du code non modifié.

Les instructions `import` et `package` ne seront pas précisées.

Exercice - Itération

L'interface `java.util.Iterator` est rappelée ci-dessous :

```
public interface Iterator<E> {

        /** Returns true if the iteration has more elements. */
        boolean hasNext();

        /** Returns the next element in the iteration.
        Throws NoSuchElementException if the iteration has no more elements*/
        E next();

        /** Removes from the underlying collection the last element returned by
        this iterator (optional operation). This method can be called only once
        per call to next(). The behavior of an iterator is unspecified if the
        underlying collection is modified while the iteration is in progress in any
        way other than by calling this method.
        The default implementation throws an instance of
        UnsupportedOperationException and performs no other action. */
        default void remove();

        /** Performs the given action for each remaining element until all
        elements have been processed or the action throws an exception.
        The default implementation behaves as if:
```

```
        while (hasNext())
            action.accept(next());
        */
        default void forEachRemaining(Consumer<? super E> action);
}
```

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.

Exemple : le code suivant

```
String[] ts = { "a", null, "c" };
Iterator<String> it = Iterateurs.iterateurTableauAvecTrous(ts);
while (it.hasNext())
    System.out.print(it.next() + " ");
System.out.println();
it.remove();
for (String s: ts)
    System.out.print(s + " ");
System.out.println();
```

affichera sur la sortie standard

```
a c
a null null
```

On souhaite généraliser le processus. Pour cela, on propose une interface `Predicat` :

```
public interface Predicat<E> {
    public boolean predicat(E element);
}
```

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.

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` .

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`.

Problème - Véhicule

On considère l'interface `Vehicule` ci-dessous :

```
public interface Vehicule {
        /** Vitesse du véhicule en km/h */
        public double vitesse();
        /** Direction du véhicule en radian, relativement à l'axe des x */
        public double direction();
        /** Position du véhicule */
        public Point2D position();
        /** Fait rouler le véhicule pendant une durée donnée en heure.
         Cette méthode modifiera la position du véhicule, en fonction
         de sa vitesse et de sa direction. */
        public void rouler(double duree);
}
```

et son implémentation par défaut `VehiculeImpl` :

```
public class VehiculeImpl implements Vehicule {

        protected double vitesseMaximum;
        protected double vitesse = 0;
        protected double direction = 0;
        protected Point2D position = new Point2D.Double(0., 0.);

        private void verifierVitessePositive(double vitesse) {
                if (vitesse < 0) {
                        throw new IllegalArgumentException("Vitesse négative impossible");
                }
        }

        public VehiculeImpl(double vitesseMaximum) {
                verifierVitessePositive(vitesseMaximum);
                this.vitesseMaximum = vitesseMaximum;
        }

        public void changerDirection(double direction) {
                this.direction = direction % (2 * Math.PI);
        }
```

```
        public void changerVitesse(double vitesse) {
                verifierVitessePositive(vitesse);
                this.vitesse = Math.min(vitesseMaximum, vitesse);
        }

        public double direction() {
                return direction;
        }

        public double vitesse() {
                return vitesse;
        }

        public double vitesseMaximum() {
                return vitesseMaximum;
        }

        public Point2D position() {
                return position;
        }

        public void rouler(double duree) {
                double distance = duree * vitesse;
                double x = Math.cos(direction) * distance;
                double y = Math.sin(direction) * distance;
                position.setLocation(position.getX() + x, position.getY() + y);
        }
}
```

Vous aurez également besoin des méthodes suivantes de la classe `java.awt.geom.Point2D` :

```
double getX();
double getY();
void setLocation(double x, double y);
```

qui respectivement

- retourne la coordonnée en x d'une instance de `Point2D`,
- retourne la coordonnée en y d'une instance de `Point2D`,
- modifie les coordonnées (x, y) d'une instance de `Point2D`.

`Point2D.Double` est une classe interne à `Point2D` qui fournit une implémentation de `Point2D`.

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.

On souhaite maintenant avoir une nouvelle classe `VehiculeAvecAcceleration` qui implémente `Vehicule` et contient une nouvelle méthode :

```
/** Modifie la vitesse du véhicule */
void accelerer(double acceleration) { ... }
```

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.

On considère maintenant l'interface `Surface` ci-dessous :

```
public interface Surface {
        /** teste si un point appartient à la surface. */
        public boolean contient(Point2D p);

        /** teste si il est possible d'aller du point p1 au point p2
         * en utilisant une ligne droite sans sortir de la surface.
         * Lève une exception IllegalArgumentException si p1 ou p2 ne
         * sont pas dans la surface.
         */
        public boolean sontConnexes(Point2D p1, Point2D p2);
}
```

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.

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`.

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.

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

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.

Pour éviter ce problème, nous proposons une autre approche. Nous souhaitons que l'on puisse créer des véhicules observables. Chaque fois qu'un véhicule observable sera déplacé, ses observateurs en seront notifiés.

Vous trouverez ci-dessous les méthodes de la classe `Observable` que vous aurez à utiliser ainsi que l'interface `Observer`.

```
public class Observable {
...
    public void addObserver(Observer o);
    public void notifyObservers(Object arg);
    protected void setChanged();
}

public interface Observer {
   public void update(Observable o, Object arg);
}
```

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`.

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`.

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.