université de BORDEAUX

4TYE707U
Programmation orientée objet
Master MIAGE

Examen du 13 décembre 2016, 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 - Boîte

On considère le code ci-dessous :

```
public class Boite {
        private Object contenu;

        public Boite(Object contenu) {
                // à compléter
        }

        public Object contenu() {
                // à compléter
        }
}
```

Question 1 Compléter le code.

Question 2 Modifier le code pour que le type de l'objet contenu dans la boîte soit générique.

Question 3 Modifier le code de la question 2 pour que si l'on exécute l'instruction

```
System.out.println(b)
```

où `b` désigne une instance de `Boite` contenant un objet `o`, le résultat soit l'affichage sur la sortie standard du message suivant :

```
Boîte contenant <chaîne décrivant l'objet contenu>
```

Par exemple :

```
Boite<String> bs = new Boite<>("Bonjour");
System.out.println(bs)
```

affichera

```
Boîte contenant Bonjour
```

Question 4 Modifier le code pour que deux boîtes soient égales si les objets qu'elles contiennent sont égaux (au sens de la méthode `equals`). Quelle autre méthode héritée de la classe `Object` est-il dans ce cas souhaitable de modifier ? Proposer également un code pour cette méthode.

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 Écrire dans une classe `Iterations` une méthode de classe
`public static <E> Iterator<E> iterationContinue(E valeur)`.
L'itérateur obtenu renverra indéfiniment la valeur `valeur` passée en paramètre. Les méthodes `remove()` et `forEachRemaining()` devront lever l'exception `UnsupportedOperationException` (*attention, ce n'est pas le comportement par défaut de* ***`forEachRemaining()`***).

Question 2 On souhaite maintenant pouvoir manipuler plusieurs valeurs dans l'itération. La nouvelle déclaration de la méthode `iterationContinue` sera donc :
`public static <E> Iterator<E> iterationContinue(E... valeurs)`.
L'itérateur obtenu retournera les valeurs passées en paramètre une par une et recommencera à la première une fois toutes les valeurs retournées. Par exemple, le code suivant :

```
Iterator<Integer> it = iterationContinue(1, 2, 3);
for (int i = 0; i < 10; i++)
    System.out.print(it.next() + " ");
```

affichera sur la sortie standard :

```
1 2 3 1 2 3 1 2 3 1
```

Donner les modifications à apporter au code.

Exercice - Observable

On considère l'interface `Valeurs` et les deux classes `ValeursImpl` et `Histogramme`.

```
public interface Valeurs {
    public int taille();
    public double somme();
    public double valeur(int i);
    public void changerValeur(int i, double v);
}

public class ValeursImpl implements Valeurs {
    private double[] valeurs;
    private double somme;

    public ValeursImpl(double[] valeurs) {
        this.valeurs = valeurs;
        somme = 0;
        for (int i = 0; i < valeurs.length; i++)
            somme += valeurs[i];
    }

    public double valeur(int i) {
        return valeurs[i];
    }

    public void changerValeur(int i, double v) {
        somme += v - valeurs[i];
        valeurs[i] = v;
    }
```

```
    public int taille() {
        return valeurs.length;
    }

    public double somme() {
        return somme;
    }
}

public class Histogramme extends JComponent {
    private Valeurs val;
    private Color[] colors = { Color.RED, Color.BLUE, Color.GREEN, Color.CYAN,
        Color.MAGENTA, Color.PINK };

    public Histogramme(Valeurs val) {
        this.val = val;
    }

    public void paintComponent(Graphics g) {
        int w = getWidth();
        int h = getHeight();
        if (isOpaque()) {
            g.setColor(getBackground());
            g.fillRect(0, 0, w, h);
        }
        int largeur = w / val.taille();
        double hauteurUnitaire = h / val.somme();
        for (int i = 0; i < val.taille(); i++) {
            int hr = (int) (hauteurUnitaire * val.valeur(i));
            g.setColor(colors[i % colors.length]);
            g.fillRect(i * largeur, h - hr, largeur, hr);
        }
    }
}
```

Question 1 Montrer que dans la classe `ValeursImpl`, il est possible de modifier le résultat retourné par la méthode `valeur(int i)` sans utiliser `changerValeur(int i, double v)` et donc sans que la variable `somme` soit mise à jour. Corriger ce défaut d'encapsulation.

Question 2 Modifier le constructeur `ValeursImpl(double[] valeurs)` pour qu'il lève une exception `ValeurNegativeException` si une des valeurs passées dans le tableau `valeurs` est négative.
Donner également le code de la classe `ValeurNegativeException`.

Question 3 On souhaite également qu'il soit impossible de passer une valeur négative à la méthode `changerValeur(int i, double v)`. Ecrire les modifications à apporter.

On souhaite que les instances d'`Histogramme` soient automatiquement mises à jour. Pour cela, on va utiliser l'interface `java.util.Observer` et la classe `java.util.Observable`, que l'on rappelle partiellement ci-dessous :

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

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

Question 3 Ecrire une classe `ValeursObservables` qui rend une instance de `Valeurs` observable. La classe `ValeursObservables` devra donc étendre la classe `Observable` et implémenter l'interface `Valeurs` en utilisant une instance de `Valeurs` déjà existante.

Question 4 Ecrire une classe `HistogrammeObservateur` qui permet de disposer d'histogrammes observant l'instance de `Valeurs` qu'ils affichent. En d'autre termes, en utilisant le code de la question 3, tout appel de la méthode `changerValeur(int i, double v)` sur l'instance de `Valeurs` affichée par l'histogramme devra entraîner une mise à jour de celui-ci.