Consignes relatives au déroulement de l'épreuve

Date :	31 mars 2014
Contrôle de :	**4ETI** - **Module « Conception orientée Objet – DP 1 » - 1ère session 2ème partie**
Durée :	1h 30
Professeur responsable :	Françoise Perrin
Documents :	Supports de cours et TP autorisés

LES TELEPHONES PORTABLES ET AUTRES APPAREILS DE STOCKAGE DE DONNEES NUMERIQUES NE SONT PAS AUTORISES.
Les téléphones portables doivent être éteints pendant toute la durée de l'épreuve et rangés dans les cartables.
Les cartables doivent être fermés et posés au sol.
Les oreilles des candidats doivent être dégagées.

Rappels importants sur la discipline lors des examens

La présence à tous les examens est strictement obligatoire ; tout élève présent à une épreuve doit rendre une copie, même blanche, portant son nom, son prénom et la nature de l'épreuve.
Une absence non justifiée à un examen invalide automatiquement le module concerné.
Toute suspicion sur la régularité et le caractère équitable d'une épreuve est signalée à la direction des études qui pourra décider l'annulation de l'épreuve; tous les élèves concernés par l'épreuve sont alors convoqués à une épreuve de remplacement à une date fixée par le responsable d'année.
Toute fraude ou tentative de fraude est portée à la connaissance de la direction des études qui pourra réunir le Conseil de Discipline. Les sanctions prises peuvent aller jusqu'à l'exclusion définitive du (des) élève(s) mis en cause.

Quelques recommandations :

Lisez bien tout l'énoncé avant de commencer chaque exercice.

Les questions sont numérotées et « en gras ».

Soyez synthétiques dans vos explications, dessinez si besoin.
Pour les diagrammes UML :

- Donnez une légende pour clarifier vos liens entre classes si vous n'utilisez pas la syntaxe UML.
- Préfixez les noms de classes et méthodes abstraites par « Abs ».
- Si vous mettez des commentaires pour préciser le contenu des méthodes, changez de couleur de stylo.

Pour les instructions Java, la syntaxe n'a pas d'importance pourvu qu'elle soit compréhensible. N'hésitez pas à ajouter des commentaires pour rendre votre code lisible.

Pour mémoire, la liste des différents patterns de conception et leurs intentions sont précisés dans le poly diapo 284 et suivantes.

1er exercice : 6 points

Il s'agit d'identifier les Design Patterns à mettre en œuvre dans les 3 cas d'utilisation suivants.

Dans un jeu vidéo, un seigneur de la guerre est capable d'entrainer ses troupes pour conquérir une ville. C'est le joueur (utilisateur) qui par un moyen quelconque va sélectionner la tactique qui lui parait la plus opportune, charge au seigneur de guerre de la mettre en œuvre. Il pourrait par exemple « défoncer le pont levis et escalader la muraille nord » ou bien « encercler la ville et attendre que tout le monde meure de faim » ou encore « épouser la fille du seigneur de la ville », etc.

1. **Quel Design Pattern mettriez-vous en œuvre pour modéliser ce système ? Justifiez et expliquez son mode de fonctionnement dans ce contexte.**

Dans une hiérarchie de classes d'animaux, tous les animaux ont le même comportement : ils mangent, se déplacent, se reproduisent et parlent. Les méthodes sont réellement définies soit au plus tôt dans la hiérarchie (les mammifères sont vivipares) soit dans les classes concrètes (les chiens « parlent » en aboyant, les chats en miaulant, etc.).
Ces classes sont déjà écrites, testées et utilisées dans de nombreuses applications…
Une nouvelle application voudrait manipuler des chiens qui mangent, se déplacent, etc. comme tous les autres animaux mais qui soient aussi capables de « faire le beau » et « rapporter la balle ».

2. **Quel Design Pattern mettriez-vous en œuvre pour modéliser cette extension du système tout en respectant le principe de Responsabilité Unique ? Justifiez et expliquez son mode de fonctionnement dans ce contexte.**

Le GPS (Global Positioning System) dispose de 24 satellites artificiels en orbite autour de la Terre à une distance de 12 600 miles/ 20 300 kilomètres qui émettent des signaux radio. Le schéma de leurs orbites a été "chorégraphié" de telle manière qu'un récepteur GPS se trouvant n'importe où sur Terre est toujours "visible" (et reçoit donc des signaux) par au moins 4 satellites.
À partir de ces 4 lectures de satellite, votre GPS peut calculer votre position par "trilatération".

3. **Quel Design Pattern est mis en place par l'afficheur du GPS de votre voiture pour vous indiquer le bon chemin à suivre pour vous rendre à votre destination ? Justifiez et expliquez son mode de fonctionnement dans ce contexte.**

2ème exercice : 8 points

Le programme suivant permet de modéliser une représentation objet d'un document XML.
Chaque élément d'un document XML est délimité par une balise de début et une balise de fin.
Les éléments peuvent être simples ou composés par d'autres éléments qui peuvent être simples ou composés à leur tour.
Un élément simple la forme suivante : <tag>valeur</tag>
Exemple de document XML :

```
<?xml version="1.0" encoding="utf-8" ?>
<BIBLIO SUBJECT="XML">
      <BOOK ISBN="9782212090819" LANG="fr" SUBJECT="applications">
            <AUTHOR>
                  <FIRSTNAME>Jean-Christophe</FIRSTNAME>
                  <LASTNAME>Bernadac</LASTNAME>
            </AUTHOR>
            <AUTHOR>
                  <FIRSTNAME>François</FIRSTNAME>
                  <LASTNAME>Knab</LASTNAME>
            </AUTHOR>
            <TITLE>Construire une application XML</TITLE>
            <PUBLISHER>
                  <NAME>Eyrolles</NAME>
                  <PLACE>Paris</PLACE>
            </PUBLISHER>
            <DATEPUB>1999</DATEPUB>
      </BOOK>
      <BOOK ISBN="9782212090529" LANG="fr" SUBJECT="général">
      ...
      </BOOK>
      ...
</BIBLIO>
```

Vous disposez des classes suivantes :

```
public abstract class ElementXML {
    protected String tag;
    public ElementXML(String tag) {
        this.tag = tag;
    }
    public String getTag() {
        return tag;
    }
    public abstract void setValeur(String valeur);
    public abstract String getValeur();
    public abstract void ajouteEnfant(ElementXML enfant);
    public abstract List<ElementXML> getElementsEnfants() ;
}

public class ElementXMLSimple extends ElementXML {
    protected String valeur;
    public ElementXMLSimple(String tag) {
        super(tag);
    }
    public void setValeur(String valeur) {
        this.valeur = valeur;
    }
    public String getValeur() {
        return valeur;
    }
    public void ajouteEnfant(ElementXML enfant) {
    }
    public List<ElementXML> getElementsEnfants() {
        return null;
    }
}
```

```java
public class ElementXMLCompose extends ElementXML {
    protected List<ElementXML> enfants = new ArrayList<ElementXML>();
    public ElementXMLCompose(String tag) {
        super(tag);
    }
    public void setValeur(String valeur) {
    }
    public String getValeur() {
        return null;
    }
    public void ajouteEnfant(ElementXML enfant) {
        enfants.add(enfant);
    }
    public List<ElementXML> getElementsEnfants() {
        return enfants;
    }
}
```

1. **Expliquez quel est le Design Pattern mis en œuvre par cette architecture.**

Afin de répondre à de nouveaux besoins, l'architecture mériterait d'être enrichie de nouvelles fonctionnalités :

- 1 méthode qui retourne le nb d'éléments composant un élément composé :
  `int getNombreEnfants();`
- 1 méthode qui retourne l'élément à l'index passé en paramètre :
  `ElementXML getEnfant(int index);`
- 1 méthode qui retourne le 1er élément ayant pour tag celui passé en paramètre :
  `ElementXML getPremierEnfantAvecTag(String tag);`

Un développeur propose l'approche suivante pour répondre à ce nouveau besoin :

```java
public class Mystere implements Element {
    protected ElementXML elementXML;
    public Mystere (ElementXML livre) {
        this.elementXML = livre;
    }
    public int getNombreEnfants() {
        return elementXML.getElementsEnfants().size();
    }
    public ElementXML getEnfant(int index) {
        List<ElementXML> enfants = elementXML.getElementsEnfants();
        return enfants.get(index);
    }
    public ElementXML getPremierEnfantAvecTag(String tag) {
        List<ElementXML> enfants = elementXML.getElementsEnfants();
        for (ElementXML elt : enfants) {
            if (elt.getTag().equals(tag))
                return elt;
        }
        return null;
    }
}
```

2. **Expliquez quel est le Design Pattern mis en œuvre par cette nouvelle classe.**

On vous demande de répondre au même besoin en mettant en œuvre cette fois le DP Decorator :

3. **Modélisez la solution (diagramme de classes UML).**

4. **Ecrivez en Java les interfaces/classes nécessaires.** Si des méthodes sont identiques à certaines de la classe précédente, écrivez simplement { idem classe `Mystere` }

5. **Dans quels cas d'utilisation recommanderiez-vous l'approche Decorator ou l'approche « Mystere » ? Justifiez.**