POBJ – Examen Janvier 2015

Durée : 2 heures Tous documents autorisés

PROBLÈME : LE LIVRE DONT VOUS ÊTES LE HÉROS

Un livre interactif est un livre composé de sections numérotées où le lecteur doit choisir à chaque section parmi un petit nombre de possibilités pour déterminer quelle section il va lire ensuite. En partant d'une section initiale, il parcourt selon ses choix une aventure « dont il est le héros ».

On vous confie la réalisation d'un logiciel qui permet de générer automatiquement un livre interactif en HTML à partir d'une interface graphique où un auteur se contente de rédiger le texte des différentes sections et de décrire les enchaînements possibles. Le logiciel détecte notamment les sections inaccessibles.

Lisez la totalité de l'énoncé ci-dessous avant de commencer à répondre aux questions.

1 : Sections et enchaînements

Une section contient un numéro unique qui permet de l'identifier, un texte principal écrit par l'auteur et une liste d'enchaînements qui permet au lecteur d'accéder à d'autres sections. Le nombre d'enchaînements possibles n'est pas limité a priori. Un enchaînement définit un texte (du type « si vous faites l'action X, allez vers la section Y ») et une référence vers la section à laquelle il permet d'accéder.

Pour fixer un contrat que doivent respecter les sections, on définit l'interface suivante :

```
package pobj;
import java.util.List;

public interface ISection {
      public int getNumero();
      public List<ISection> getSuivantes();
      public String getTexte();
      public void ajouterEnchainement(String texte, ISection suivante);
}
```

La classe *Section,* qui réalise *ISection*, est dotée en outre

- d'une liste de chaînes de caractères contenant les textes des enchaînements vers les sections immédiatement suivantes.
- des attributs nécessaires pour implémenter *Isection,*
- d'un constructeur qui prend en paramètre la chaîne de caractères correspondant au texte principal fourni par l'auteur. En vous inspirant de ce qui a été vu en TD, vous proposerez un mécanisme simple de numérotation croissante des sections qui assure que chaque section

soit dotée d'un numéro unique (à partir de 1) lors de sa création.

- d'une méthode `public void ajouterEnchainement(String texte, ISection suivante)` permettant d'ajouter aux suites possibles de cette section un nouvel enchaînement composé d'un texte et de la section accessible par cet enchaînement
- d'une méthode `public String getTexteEnchainement()` qui renvoie pour tous les enchaînements, le texte d'enchaînement et le numéro de la section pointée.
- d'une méthode `public String toString()` qui renvoie une chaîne de caractères contenant le numéro et le texte de la section puis le texte des enchaînements défini ci-dessus.

Q1 Proposez une implémentation de la classe *Section*.

```
package pobj;

import java.util.ArrayList;
import java.util.List;

public class Section implements ISection {
	private static int compteur = 1;
	private int id;
	private String text;
	private boolean atteinte = false;

	private List<ISection> suivantes;
	private List<String> enchainements;

	public Section(String text){
		id = compteur++;
		this.text = text;
		suivantes = new ArrayList<ISection>();
		enchainements = new ArrayList<String>();
	}

	public int getNumero(){
		return id;
	}

	public String getTexte() {
		return text;
	}


	public boolean isAtteinte() {
		return atteinte;
	}

	public void setAtteinte(boolean atteinte) {
		this.atteinte = atteinte;
	}

	@Override
	public List<ISection> getSuivantes() {
		return suivantes;
	}
	public void ajouterEnchainement(String texte, ISection suivante){
```

```
            suivantes.add(suivante);
            enchainements.add(texte);
      }
      public String getTexteEnchainement(){
            if (enchainements.size()==0) return "";
            String retour = "Enchainements :";
            for (int i=0;i<enchainements.size();i++){
                  retour += enchainements.get(i);
                  retour += " -> Section " + suivantes.get(i).getNumero() +
"\n";
            }
            return retour;
      }
      public String toString(){
            String retour = "Section "+ getNumero() + " :" + getTexte() + "\n" +
getTexteEnchainement();
            return retour + "\n";
      }
}
```

2 : Classe Livre

L'ensemble des sections est regroupé dans une classe principale *Livre*, qui contient notamment :

- une référence sur la section initiale, stockée dans une variable *init* ;
- une liste des sections du livre, stockée dans une variable *contenu ;*

Q2.1 Proposez les attributs et le constructeur de la classe *Livre* (on lui passe la section initiale et une url),

```
package pobj;

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class Livre {
      private List<ISection> contenu;
      private ISection init;
      private PageGenerator pageGen;

      public Livre(ISection init, String url){
            this.init = init;
            pageGen = new PageGenerator(url);
            contenu = new ArrayList<ISection>();
      }
```

Q2.2 Ajoutez une méthode `public void ajouterSection(ISection s)` qui ajoute une section,

```
      public void ajouterSection(ISection s){
            contenu.add(s);
```

```
}
```

3 : Sections avec objet et avec monstre

Le lecteur dispose d'un inventaire qui lui permet de stocker des objets que son personnage récolte au cours de son aventure. En effet, dans certaines sections, le personnage peut trouver des objets qui lui serviront dans la suite. Le texte d'une section avec objet est le texte d'une section classique auquel on ajoute la phrase : « Vous trouvez un nouvel objet : » suivi de la description de l'objet et de sa valeur en pièces d'or.

De même, il existe des sections dans lesquelles le personnage peut rencontrer des monstres, auquel cas on ajoute au début du texte la phrase « Vous êtes attaqué par un : » suivi du nom du monstre et de son nombre de points de vie.

Un extrait de la classe représentant les objets est fourni ci-dessous :

```
package pobj;

public class Objet {
	private String description;
	private int valeur;

	public Objet(String desc, int val){
		description = desc;
		valeur = val;
	}
	public String toString(){
		return getDescription() + " de valeur :" + valeur;
	}
	public String getDescription() {return description;}
}
```

On représente les sections avec objets par la classe *SectionAvecObjet* et les sections avec monstres par la classe *SectionAvecMonstre*.

Il existe aussi des sections dans lesquelles peuvent se trouver à la fois des objets et des monstres.

Q3.1 : Quel *design pattern* proposez-vous d'utiliser pour implémenter les sections dotées d'objets et/ou de monstres sans avoir à toucher aux définitions des classes et interfaces précédentes ?

Decorator

Q3.2 : Donnez une implémentation de la classe *SectionAvecObjet* et d'une classe intermédiaire requise pour le pattern, conforme au pattern proposé **(mais pas la classe *SectionAvecMonstre*)**.

```
package pobj;

import java.util.List;

public class SectionDecorator implements ISection {

	private ISection decorated;

	   public SectionDecorator(ISection decorated){
```

```
            this.decorated = decorated;
        }
	public List<ISection> getSuivantes() {
		return decorated.getSuivantes();
	}

	public String getTexte() {
		return decorated.getTexte();
	}

	@Override
	public int getNumero() {
		return decorated.getNumero();
	}
	@Override
	public void ajouterEnchainement(String texte, ISection suivante) {
		decorated.ajouterEnchainement(texte, suivante);

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

```
package pobj;

public class SectionAvecObjet extends SectionDecorator {

	private Objet obj;

	 public SectionAvecObjet(ISection decorated, Objet obj){
		super(decorated);
		this.obj = obj;
	    }

	public String getTexte() {
		String retour = super.getTexte()+" :\n"; //le super est critique
		retour += "Vous trouvez un nouvel objet : " + obj.getDescription() +
"\n";
		return retour;
	}
}
```

-50 % s'il manque le super dans getTexte().

4 : Parcours du graphe et sections inaccessibles

L'ensemble des enchaînements possibles du livre constitue un graphe. Dans cette partie, vous allez réaliser une fonctionnalité de parcours de ce graphe utilisée pour détecter les sections inaccessibles.

Les traitements correspondant sont regroupés au sein de la classe *Livre* définie plus haut.

Une difficulté particulière du parcours du graphe des sections provient du fait qu'il faut éviter de parcourir à nouveau le graphe à partir de sections que l'on a déjà rencontrées.

Pour cela, vous allez appliquer le principe de parcours suivant :

- le graphe est parcouru en profondeur d'abord à partir de la section initiale,
- à chaque fois qu'il rencontre une nouvelle section, l'algorithme retient qu'il l'a atteinte en la stockant dans un ensemble (classe *HashSet<>* implémentant *Set<>*) des sections atteintes;
- s'il rencontre une section déjà atteinte, l'algorithme ne continue pas le parcours à partir de cette section.

Pour les questions Q4.1 à Q4.3 vous pourrez vous appuyer sur les méthodes standards des *Collections* fournies en annexe.

Q4.1 Définissez une méthode récursive `private void calculeAccessibleRec(ISection s, Set<ISection> vus)` qui renvoie dans le second paramètre l'ensemble des sections vues à partir de la section passée dans le premier paramètre, si elle n'est pas encore vue. Si la section passée en paramètre est déjà vue, alors la méthode ne fait rien.

```
private void calculeAccessibleRec(ISection s, Set<ISection> vus) {
      if (vus.add(s)){
            for (ISection suite : s.getSuivantes()) {
                  calculeAccessibleRec(suite, vus);
            }
      }
}
```

Q4.2 En vous appuyant sur la méthode précédente, définissez une méthode `private Set<ISection> calculeAccessible()` qui renvoie en valeur de retour l'ensemble des sections accessibles à partir de la section initiale du livre.

```
private Set <ISection> calculeAccessible() {
      Set <ISection> reach = new HashSet<ISection>();
      calculeAccessibleRec (init, reach);
      return reach;
}
```

Q4.3 Ajoutez une méthode *`public List<ISection> detecterInaccessible()`* qui renvoie la liste des sections inaccessibles à partir de la section initiale du livre.

```
public List<ISection> detecterInaccessible() {
      // on cree une copie propre des sections du Livre
      List<ISection> inacc = new ArrayList<ISection>(contenu);
      // calcul de reach
      Set <ISection> vus = calculeAccessible();
      // diff ensembliste
      inacc.removeAll(vus);
      return inacc;
}
```

5 : Génération du livre en HTML

Pour générer le livre en HTML, on souhaite ré-utiliser la classe sur étagère *PageGenerator* dont le code est fourni ci-dessous.

```
package pobj;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
```

```
import java.io.Writer;
import java.util.List;

public class PageGenerator {
	private String urlRoot;

	public PageGenerator(String url){
		this.urlRoot = url;
	}
public void createPages(List<IPage> pages) throws IOException{
		for (IPage page : pages){
			// NB : Levée d'IOException si le dossier "urlRoot" n'existe
pas ou n'est pas ouvert en écriture
			Writer out = new OutputStreamWriter(new
FileOutputStream(urlRoot+page.getNumber()+".html"));
			out.write("<html><body>" + page.getHTMLTitle());
			out.write(page.getContent());
			out.write("</body></html>");
			out.close();
		}
	}
}
```

On souhaite appliquer ce code sans le modifier à des objets de type *ISection*. Notamment, le titre HTML des pages sera « <h2> Section numéro » suivi du numéro de la section suivi de « </h2> ».

Q5.1 Définissez une interface *IPage* supportant le code ci-dessus.

```
package pobj;

public interface IPage {
	public int getNumber();
	public String getHTMLTitle();
	public String getContent();
}
```

Q5.2 Quel *design pattern* proposez-vous d'utiliser pour réaliser cette spécification ?

Adapter

Q5.3 Proposez le code d'une classe qui permet d'implémenter le pattern que vous avez choisi.

```
package pobj;

public class SectionToPage implements IPage {
	private ISection section;

		public SectionToPage(ISection section){
			this.section = section;
		}
	@Override
	public String getHTMLTitle() {
		return "<h2>Section numero " + section.getNumero()+"</h2>";
	}

	@Override
	public String getContent() {
		return section.getTexte();
```

```
	}
	@Override
	public int getNumber() {
		return section.getNumero();
	}

}
```

Q5.4 Dans la classe *Livre*, ajoutez une méthode `public void genererPages()` qui fait le nécessaire pour générer le livre en HTML conformément à la spécification ci-dessus. Si l'url donnée au *PageGenerator* est invalide, l'exception « *IOException* » est levée par le *PageGenerator*, auquel cas votre application doit afficher « url invalide » dans le terminal et s'arrêter.

```
	public void genererPages(){
		ArrayList<IPage> pages = new ArrayList<IPage>();
		for (ISection s : contenu){
			pages.add(new SectionToPage(s));
		}
		try {
			pageGen.createPages(pages);
		} catch (IOException e) {
			System.out.println("url invalide");
			System.exit(-1);
		}
	}
```

6 : Programme principal

Q6.1 Dans une classe *Main*, écrivez la méthode principale de l'application qui :

- crée deux sections initiale et finale dont le texte est respectivement « bienvenue », et « fin »,
- crée une section dont le texte est « milieu » et qui contient deux objets : une « épée » de valeur 10 et un « bouclier » de valeur 20 ;
- ajoute des enchaînements : la section initiale pointe sur la section finale avec le texte « choix début », la section centrale pointe aussi sur la section finale avec le texte « choix milieu »,
- crée le livre avec la section initiale et lui ajoute les sections précédemment créées,
- détecte et affiche les sections inaccessibles,
- crée le *PageGenerator* avec l'url « ./livre » et crée les pages HTML avec ce *PageGenerator*.

```
ackage pobj;

import java.util.List;

public class Main {
	public static void main(String args[]){
		Section init = new Section("bienvenue");
		Section fin = new Section("fin");
		Section centrale = new Section("milieu");
```

```
            SectionAvecObjet obj = new SectionAvecObjet(centrale,new
Objet("epee",10));
            SectionAvecObjet obj2 = new SectionAvecObjet(obj,new
Objet("bouclier",20));
            init.ajouterEnchainement("choix fin",centrale);
            fin.ajouterEnchainement("retour debut",init);
            Livre l = new
Livre(init,"/home/sigaud/Bureau/Docs/Cours/LI314/current_repository/2013-
2014/exam/test/l");
            l.ajouterSection(init);
            l.ajouterSection(fin);
            l.ajouterSection(obj2);
            System.out.println(l);
            List<ISection> l2 = l.detecterInaccessible();
            System.out.println("Sections inaccessibles :" + l2);
            l.genererPages();
      }
}
```