

Examen - HAI712I ("Ingénierie Logicielle") - M1 Informatique

Année 2023-2024, Session 1, janvier 2023.

Durée : 2h00. Documents non autorisés. Il est possible de répondre aux questions de toute section sans avoir répondu aux autres ; mais la lecture dans l'ordre préalable est vivement conseillée. Il y a 4 grandes sections rapportant respectivement environ : section 1-6 points, 2-4, 3-5, 4-5. Une réponse excellente rapporte du bonus hors barème - et inversement une réponse au hasard ou incohérente ou hors sujet vaut malus (donc quand on ne sait pas, ne pas répondre vaut mieux qu'écrire 50 lignes). La concision et le style des textes et programmes sont pris en compte.

Contexte

Considérons comme prétexte un jeu de type "Pacman". Les éléments d'une partie standard sont un héros, un *pacman*, et les éléments que le héros peut rencontrer que nous nommons les "*obstacles*"; les obstacles sont des *fantômes*, des *cerises* ou des *points jaunes* (voir figure). Les *points jaunes* et les *cerises* sont fixes. Les *fantômes* sont mobiles; leurs déplacements sont aléatoires. Un *pacman* est mobile, son déplacement est contrôlé par le joueur (via le clavier par exemple).

Règles. (i) Un héros peut rencontrer lors de ses déplacements les obstacles. (ii) Lorsqu'un *pacman* rencontre un *point jaune*, il le mange et son score augmente; lorsque tous les points jaunes ont été mangés, le niveau courant est terminé et le jeu passe au niveau suivant. (iii) Lorsqu'un *pacman* rencontre un *fantôme*, il perd une vie; s'il n'a plus de vies, la partie est terminée. (iv) Lorsqu'un *pacman* rencontre une *cerise*, il devient invincible par les fantômes pendant un certain laps (durée) de temps. Pendant ce laps de temps, une rencontre d'un *pacman* avec un *fantôme* envoie ce dernier en prison et rapporte des points supplémentaires; à la fin du laps de temps, le *pacman* reprend son comportement standard.

Nous nous intéressons à une réalisation informatique de ce jeu, et par extension à la réalisation d'un framework extensible permettant l'intégration aisée de nouveaux types de héros d'obstacles ayant des comportements spécifiques lors des rencontres. A cet effet nous imaginons les classes ci-après (le décalage à droite signifiant "est sous-classe de", par ex. *Obstacle* est sous-classe de *ElementJeu*).

On crée un nouveau jeu (voir listing 1) en créant une instance de la classe *Jeu* et en y injectant des éléments de jeu. La détection des rencontres est implantée par le moteur du jeu (méthode *moteur()* de la classe *Jeu*).

```
Jeu
ElementJeu
  Héros
    Pacman
    AutreTypeDeHéros
  Obstacle
    Fantôme
    AutreTypeDeFantome
  Cerise
  PointJaune
  AutreTypeObstacle
```

un pacman en jeu une cerise 4 fantômes



un point jaune réserve de vies du pacman

1 Framework version No1, bases réutilisation et patterns

Dans cette première version du framework, à chaque rencontre le moteur envoie un message *rencontrer(argument)* au héros courant (un *pacman* dans la version standard) avec comme argument l'objet rencontré, instance d'une sous-classe de la classe *Obstacle*. L'exemple du listing 1 affiche à l'écran : "un pacman a rencontré un fantôme".


```

1 public class Jeu {
2     protected Héros héros;
3     protected List<Obstacle> obstacles = new ArrayList<Obstacle>();
4     public void injectHéros(Héros h) { héros = h;}
5     public void injectObstacle(Obstacle o) {obstacles.add(o);}

7     public static void main(String[] args) {
8         Jeu g = new Jeu(); //un jeu avec un pacman et un fantôme
9         g.injectHéros( new Pacman() );
10        g.injectObstacle( new Fantôme() );}
11        g.moteur()}

```

Listing 1 – classe Jeu de la version 1

La méthode `rencontrer(Obstacle)` définie sur `Héros` (voir listing 2) affiche simplement au terminal quel héros rencontre quel obstacle, sans rien faire de plus dans cette version.

```

1 public abstract class Héros extends Element {
2     public abstract String getDescription();
3     public void rencontrer(Obstacle o){
4         System.out.println(this.getDescription() + " a rencontré un " + o.getDescription()); } }

```

Listing 2 – classe Héros de la version 1

- A) Ce framework est-il adaptable, sans modification de son code (donné par les listings 1 et 2), à toute extension sous la forme d'un nouveau sous-type de `Héros` ou d'`Obstacle`? Après l'ajout, le framework doit continuer à fonctionner et intégrer le traitement des instances des nouveaux types si elles y ont été injectées. Si oui comment et si non pourquoi?
- B) Donnez le code d'une extension que vous inventerez.
- C) Dans cette version du framework,
 1) où sont les inversions de contrôle?
 2) à quoi servent-elles?
- D) 1) Dans le listing 1, listez deux cas d'affectation polymorphique.
 2) Pourquoi n'est-il pas possible de réaliser un framework sans utiliser d'affectation polymorphique? Utilisez l'exemple du listing 1 pour expliquez votre réponse.
- E) 1) Quels sont les types statiques des variables `this` et `o` dans la méthode `rencontrer(Obstacle)` du listing 2?
 2) Quels sont les types dynamiques des variables `this` et `o` durant une exécution de la méthode `rencontrer(Obstacle)` du listing 2?

2 Framework version No2 - Spécialisation de méthodes

On souhaite maintenant implanter le fait que dans une nouvelle version un peu plus avancée du jeu, on doit exécuter des méthodes `rencontrer(...)` différentes suite à des rencontres entre un héros et des obstacles de différentes sortes.

On oublie donc la version 1 précédente de la classe `Héros` et de la méthode `rencontrer(Obstacle)` et on propose une nouvelle version No 2 (voir listings 3 et 4) pour tenter de distinguer les cas. Pour le moteur de jeu (listing 4), prenez garde aux types des variables.

- A) Pour réaliser la distinction des cas, pour quelle raison de génie logiciel a-t-on choisi de définir plusieurs méthodes `Rencontrer(...)` et pas une seule utilisant des tests explicites via l'opérateur `instanceOf`?
- B) Expliquez vos réponses aux deux questions suivantes en terme de : "envois de message", "type statique", "type dynamique" "redéfinitions de méthodes", etc. Il n'y a aucun code à écrire.
- 1) Quelle est la méthode invoquée par l'appel `h.rencontrer(f)` du listing 4?
- 2) Même question que "1)" ci-avant en supposant que l'on n'ait pas défini la méthode `"public abstract void rencontrer(Fantôme f)"` sur la classe `Héros`.
- 3) Même question que "1)" ci-avant, en supposant que la ligne 2 du listing 4 soit remplacée par : `"Obstacle f = new Fantome();" .`

```

1 public abstract class Héros extends ElementJeu {
2     public void rencontrer(Obstacle r) {System.out.println("Suis-je utile?");}
3     public abstract void rencontrer(Fantôme f);
4     public abstract void rencontrer(Cerise c);
5     ... }

7 public class Pacman extends Héros {
8     public void rencontrer(Fantôme f){
9         System.out.println("le pacman courant rencontre un fantôme"); }

11    public void rencontrer(Cerise c){
12        System.out.println("le pacman courant rencontre une cerise"); } ... }

```

Listing 3 – classes Héros et Pacman de la version 2

```

1 Héros h = new Pacman();
2 Fantôme f = new Fantôme(); Cerise c = new Cerise();
3 while (jeuNonFini){
4     //à chaque fois que le pacman rencontre un
5     // fantôme (resp. une cerise), on exécute :
6     h.rencontrer(f); // (resp. h.rencontrer(c);)
7     ...}

```

Listing 4 – Simulation du moteur de jeu pour la version 2

3 Gérer les changements de comportement des héros (pour la version No 2) du Framework

On se place dans le contexte de la version 2 du coeur du framework (listings 3 et 4) et on s'intéresse au problème indépendant de la réalisation du changement de comportement d'un héros, déclenché lorsqu'il rencontre une cerise et faisant que, si le héros est un pacman, pendant un certain laps de temps et avant de redevenir normal, il envoie les fantômes qu'il rencontre en prison.

- A) Listez les -avantages- et -inconvénients- du schéma State appliqué à ce problème.
- B) Listez les -avantages- et -inconvénients- du schéma Decorator appliqué à ce problème.
- C) Sur la base de vos réponses précédentes, faites un choix motivé entre State et Decorator. Puis donnez les éléments clé du code de votre solution (n'écrivez que les parties du code en rapport avec la question posée). L'aspect "comptage du temps écoulé" ne fait pas partie de la question et pourra être représenté par des commentaires dans le code. En fonction du schéma choisi, commentez les points importants (où sont les changements d'état ou bien où sont les ajouts et retraits de décorations).

POUR LA SUITE TOURNEZ SVP →

4 Framework version No 3, une version réaliste du moteur de jeu.

La version 2 précédente du framework pose un problème pour l'écriture du moteur de jeu, il est en fait nécessaire de pouvoir stocker chaque nouvel objet rencontré, quel que soit sa classe, dans une variable de type **Obstacle** et de traiter tous les cas via un même envoi de message. Pour ce faire, on garde les classes **Héros** et ses sous-classes inchangées par rapport à la version 2 (listings 3 et 4) et on réalise la simulation du moteur de jeu du listing 5.

```
1  Héros h = new Pacman();
2  while (jeuNonFini){
3      Obstacle o = ... //appel d'une méthode qui rend le prochain obstacle rencontré, quel qu'il soit
4      h.rencontrer(o);
5  }
```

Listing 5 – Simulation du moteur de jeu pour la version 3

- A) En supposant que la variable **o** du listing 5 référence à l'exécution une instance de **Fantôme**, pourquoi l'appel **h.rencontrer(o)** invoque-t-il la méthode **rencontrer(...)** de la classe **Héros** et pas celle de la classe **Pacman** ?
- B) Suite au problème indiqué par la question précédente (l'invocation de la mauvaise méthode fait que le système ne fonctionne pas), sans modifier la classe **Héros** ni ses sous-classes, proposez une modification du reste du système pour que l'ensemble fonctionne correctement avec toutes les sortes de héros et d'obstacles existants (invoquer la bonne méthode sur la bonne (sous-)classe de **Héros**).

Il faut conserver cette variable **o** de type **Obstacle** dans le code du moteur de jeu (listing 5) mais vous pouvez modifier son code .

- C) Montrez que la solution mise en place en B permet, sans modification de son code, l'intégration a posteriori d'une nouvelle classe d'obstacles, par exemple les Bananes qui font déraiser un pacman. Si elle ne le permet pas, modifiez là.
- D) Indiquez, sans écrire de code, si la solution mise en place en B ou C est compatible avec la solution pour le changement de comportement (State ou Décorateur) mis en place sur les Héros à la section 3. On supposera que la solution mise en place pour la version 2 du framework est correctement transposée à la version 3. A priori les classes **Héros** et **Pacman** ne sont pas modifiées par le passage à la version 3.
- E) Connaissez vous une règle générale pour qu'un système intégrant n hiérarchies de classes indépendantes supporte l'ajout de nouveaux types de données sans modification du code existant.