EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Faculté Informatique et Communication
Cours de programmation aux sections IN/SC
Sam J.

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

INTRODUCTION A LA PROGRAMMATION

Corrigé test semestre 1

Exercice 2 : Conception OO et programmation [52 points]

Voir le fichier source `Simulation.java`.
BARÈME : Le barème est donné en commentaire dans ce fichier

Exercice 3 : Concepts [36 points]

1. **[6 points]**
   (a) Le programme affiche :

   ```
   A: 1
   A: 1
   B: 0
   A: 1
   B: 4
   ```

   Justification : La ligne 31 appelle le constructeur de la ligne 7 en lui passant la valeur 2 en paramètre. Ce constructeur appelle le constructeur par défaut de la même classe à la ligne 8 qui affiche `A: 1` car l'attribut `a` de `A` a été initialisé à 1. La ligne 9 s'exécute ensuite qui affect 2 à l'attribut `a` de `A`. La ligne 32 appelle le constructeur de la ligne 17 en lui passant la valeur 2 en paramètre. Ce constructeur appelle le constructeur de la ligne 7 en lui passant la même valeur lequel va à nouveau appeller le constructeur de la ligne 3 et afficher `A: 1`. La ligne 19 s'exécute ensuite qui affiche `B: 0` car l'attribut `a` de `B` n'est pas initialisé explicitemement et a donc la valeur zéro par défaut. La ligne 33 appelle le constructeur de la ligne 22 qui appelle implicitement le constructeur par défaut de `A` et qui affiche à nouveau `A: 1`. Puis la ligne 23 affecte 4 à l'attribut `a` de `B` et affiche `B: 4`

   (b) oui, on référencerait alors le paramètre du constructeur plutôt que l'attrribut du même nom.

   (c) Non car le constructeur s'invoquerait lui même (récursion infinie).

   (d) Non car `this(..)` doit être la première instruction d'un constructeur.

BARÈME :

(a) si l'affichage est correct, mettre 3 points (+ 1 point de bonus en cas d'effort de justification). Sinon, 0.5 par affichage correct (par rapport aux appels de constructeurs et 1.5 pour une justification proche de celle du corrigé).

(b) 0.5 pour la réponse et 0.5 pour la justification.

(c) 0.5 pour la réponse et 0.5 pour la justification.

(d) 0.5 pour la réponse et 0.5 pour la justification.

2. **[6 points]** Le programme affiche :

   ```
   1 5 22
   5 22
   5 22
   ```

   La ligne 28 déclare-initialise un tableau dynamique vide. Les lignes 29 à 31 y ajoutent en séquence les valeurs 1, 5 et 22. La méthode `f` affiche le contenu du tableau. C'est ce qui explique la première ligne affichée. **Les itérations sur ensemble de valeurs ne peuvent pas modifier le contenu de l'ensemble sur lequel elles itèrent**. Ceci explique pourquoi les lignes 12 à 14 de la fonction `g` n'incrémentent pas les valeurs du tableau. **Les paramètres sont toujours passés par valeur en Java**. La référence au tableau passé en paramètre ne peut être modifié mais le contenu référencé peut l'être. C'est ce qui explique pourquoi l'élément 1 n'est plus dans le tableau après exécution de la méthode `g` (en raison de sa ligne 15). C'est aussi ce qui explique pourquoi la ligne 24 de la fonction `h` ne parvient pas à changer la référence à l'objet `list` du programme principal.

   BARÈME : 1 point par affichage correct, 3 points pour les explications : 0.5 pour l'explication du premier affichage. 1.25 pour les explications de chacune des deux autres lignes proches de celles données ci-dessus. Les points importants étant ceux en gras ci-dessus.

3. **[7 points]**
   (a) **(1 pts)** non, car le constructeur de `A` est privé

   (b) **(1 pts)** non pour la même raison

   (c) **(1 pts)** non, car l'attribut `I` de `A` est `final`

   (d) **(1 pts)** non, l'accès à un membre non statique n'est pas possible dans `main`

   (e) **(1 pts)** oui, un membre statique peut être invoqué au travers du nom de la classe

   (f) **(1 pts)** oui, on peut accéder à un membre statique au travers d'une instance (ici créée anonymement)

   (g) **(1 pts)** non car l'attribut `a` de `A` est `private`.

4. **[4 points]** L'interface Game décrit un jeu du point de vue fonctionnel. `ActorGame` est une implémentation spécifique de `Game` (d'autre types de jeu pourraient avoir la même API fonctionnelle en ayant une tout autre logique d'implémentation. Le choix fait dans le projet permet d'appliquer le principe de «Program to an interface, not to an implementation». La classe qui utilise `BikeGame` en tant que `Game` et non en tant que `ActorGame` profite de l'encapsulation offerte par l'interface et se préserve des éventuelles modifications d'implémentation dans `ActorGame`. Elle s'offre aussi la possibilité d'utiliser une autre implémentation si elle le souhaite.

   BARÈME : 4 points pour une explication proche du corrigé, les points importants étant la séparation de l'aspect fonctionnel de l'implémentation + l'encapsulation offerte par la vue fonctionnelle offerte par l'interface.

5. **[6 points]**

(a) Pour « HoHoHoHoHoHo » puis « oH » :

```
** 1 3 5 7 9
5 **
Voila !
C'est parti !
```

Justification: Pour le premier cas : la ligne 11 va causer l'exécution de la fonction `f` de la ligne 22. Comme la chaîne `"HoHoHoHoHoHo"` a une taille (6) supérieure à celle de la chaîne `"oH"`, (2) l'exécution passe à la ligne 27. Les lignes de 27 à 32 comptabilisent (dans une variable `z`) le nombre d'occurences de la chaîne `"oH"` dans la chaîne `"HoHoHoHoHoHo"` et affichent les indices des positions de chaque occurence de `"oH"`. C'est ce qui cause l'affichage de `1 3 5 7 9`. La ligne 11 affiche ensuite la valeur de `z` retournée par la méthode `f` (5) suivie de `"**"`. Aucune exception n'est lancée par l'exécution de la ligne 11. Le programme passe donc à la ligne 12 et affiche «Voilà !». Le bloc `catch` est ignoré et le bloc `finally` s'exécute affichant «C'est parti !».

(b) Pour « oH » puis « HoHoHoHoHoHo » :

```
** Non !
C'est parti !
```

Justification : la chaîne `"oH"` étant de taille inférieure à `"HoHoHoHoHoHo"`, la ligne 23 va causer l'exécution de la ligne 24 et donc le lancement d'une exception. Le programme passe donc de la ligne 11 à la ligne 14 sans passer par la ligne 12. La ligne 15 affiche le message lié à l'exception lancée, c'est à dire «Non !». Le bloc `finally` est toujours exécuté ce qui affiche «C'est parti !».

BARÈME : deux fois 3 points dont 1 point (à chaque fois) pour une explication proche du corrigé.

6. [**6 points**]

(a) **(1 pts)** Faux. Une méthode abstraite n'a pas de corps (ne retourne pas de résultat concret) et il n'y aurait aucune raison qu'elle retourne la valeur zéro en particulier.

(b) **(1 pts)** Faux. On peut toujours déclarer une variable, même de type abstrait. Ce n'est par parcequ'une classe contenant une méthode abstraite est non instanciable en tant que telle qu'on ne peut pas déclarer de variable de ce type (par exemple pour accueillir une instance de sous-classe non abstraite).

(c) **(1 pts)** Vrai. IL n'est pas possible d'écrire `new Truc(...)`

(d) **(1 pts)** Faux. La méthode `f` est typiquement amenée à être redéfinie dans les sous-classes que l'on souhaite instancier.

(e) **(1 pts)** Faux. `Truc` peut avoir plusieurs sous-classes mais il n'y a aucune raison qu'elle y soit forcée.

(f) **(1 pts)** Vrai. En Java une classe contenant une méthode `abstract` doit être déclarée comme `abstract`.

Exercice 3 : Deroulement de programme [22 points]

Le programme affiche :

```
* Royaume des Elfes *
Size: 20
Money: 30

* Royaume des Nains *
Size: 30
Money: 10

-----
* Royaume des Elfes *
Size: 30
Money: 50

* Royaume des Nains *
Size: 50
Money: 50
```

Justification :

- **(1 pts)** la ligne 74 déclare une variable `elves` de type `Kingdom` et lui affecte un objet de type `Alfheim` (sous-classe de `Kingdom`). Cet objet est initialisé au moyen du constructeur de la ligne 29 qui appelle celui de la ligne 10. Le « royaume » elves a alors pour surface 20 et pour richesse 30.
- **(1 pts)** la ligne 75 déclare une variable `dwarfs` de type `Kingdom` et lui affecte un objet de type `Nidavel` (sous-classe de `Kingdom`). Cet objet est initialisé au moyen du constructeur de la ligne 45 qui appelle celui de la ligne 10. Le « royaume » dwarfs a alors pour surface 30 et pour richesse 10.
- **(0.5 pts)** la ligne 76 (ou 84 avec adaptation du contenu) fait appel à la méthode de la ligne 35 et affiche
  ```
  * Royaume des Elfes *
  Size: 20
  Money: 30
  ```
- **(0.5 pts)** La ligne 77 (ou 85 avec adaptation du contenu) fait appel à la méthode de la ligne 50 et affiche
  ```
  * Royaume des Nains *
  Size: 30
  Money: 10
  ```
- **(0.5 pts)** la ligne 81 construit un objet `cb` de type `CityBuilder` en faisant appel au constructeur de la ligne 67.
- **(2.25 pts)** la ligne 81 faut appel à sa méthode `launch` en lui passant l'objet `cb` en paramètre. Ceci cause l'appel de la méthode de la ligne 67 avec `elves` passé en paramètre. La ligne 68 cause l'appel de `grow(10,20)` sur un objet de type `Alfheim` (polymorphisme). Comme la méthode `grow` n'est pas redéfinie dans `Alfheim`, c'est la définition de la super-classe qui est utilisée. La surface du royaume `elves` est augmentée de 10 (la faisant passer à 30) et sa richesse de 20 (la faisant passer à 50).
- **(2.25 pts)** la ligne 82 faut appel à sa méthode `launch` en lui passant l'objet `cb` en paramètre. Ceci cause l'appel de la méthode de la ligne 67 avec `dwarfs` passé en paramètre. La ligne 68 cause l'appel de `grow(10,20)` sur un objet de type `Nidavel` (polymorphisme). La méthode `grow` de la ligne 55 est invoquée et la surface du royaume dwarfs est augmentée de 10*2 (la faisant passer à 50) et sa richesse de 20*2 (la faisant passer à 50 aussi).
- **(comptabililisé plus haut)** les lignes 84 et 85 font appels aux méthodes `toString` des classes `Alfheim` et `Nidavel` respectivement et cause les affichages

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
TEST INTRODUCTION A LA PROGRAMMATION – IN/SC
J. Sam
21 Décembre 2017

```
* Royaume des Elfes *
Size: 30
Money: 50

* Royaume des Nains *
Size: 50
Money: 50
```

BARÈME :

— 2 point pour le premier affichage du royaume des elfes
— 2 points pour le premier affichage du royaume des nains
— 5 points pour le second affichage du royaume des elfes
— 5 points pour le second affichage du royaume des nains
— ajoutez les points tels que suggérés pour chaque élément de justification ci-dessus.

Note : pour les points d'explications : ne soyez pas trop sévère sur le niveau de détail utilisé pour expliquer ; ce qui compte c'est d'être convaincu qu'ils ont compris ce qui se passe vraiment.