

INTRODUCTION A LA PROGRAMMATION

Corrigé test semestre 1

Exercice 2 : Conception OO et programmation [55 points]

Voir le fichier source `Termites.java`.

BARÈME : Le barème est donné en commentaire dans ce fichier

Exercice 3 : Concepts [37 points]

1. [5 points]

- (a) Le constructeur de la ligne 14 qui appellera implicitement celui de la ligne 3 qui appellera à son tour celui de la ligne 6.
- (b) les attributs sont protégés au lieu de privé (mauvaise pratique). Il devrait être à la charge des constructeurs de `X` d'initialiser l'attribut `x`. Pour y remédier : déclarer les attributs en privé et remplacer la ligne 15 par `super(x)`;

BARÈME :

- (a) 1.5 (0.5 par constructeur mentionné)
- (b) 3.5 : 1 pour la critique sur les attributs protégés, 1 sur celle relative à l'initialisation de `x`. 0.5 pour les attributs en privé et 1 pour l'instruction `super`

2. [4 points] Le programme affiche :

Ragging
Bull

Les paramètres sont toujours passés par valeur en Java. La méthode `p` ne permute pas le contenu des variables `s1` et `s2`

BARÈME : 2 points pour l'affichage et 2 pour l'explication.

3. [7 points] Pour chacune des réponses ci-dessous : 0.5 pour la réponse (oui/non) et 0.5 pour la justification.

- (a) (1 pts) non, car `A` est abstraite et donc non instantiable
- (b) (1 pts) oui car `B` est instantiable et parcequ'on peut affecter un `B` à un `A` (compatibilité ascendante des types).
- (c) (1 pts) non, car on ne peut affecter un `I` à un `B` (un `B` se comporte comme un `I` mais un `I` ne se comporte pas et n'est pas forcément un `B`).
- (d) (1 pts) non, car on ne peut créer d'instance d'interfaces.
- (e) (1 pts) oui, car un objet de type `B` a aussi le type `I` (`B` implémente `J` qui descend de `I`).
- (f) (1 pts) oui, raison analogue au cas précédent.
- (g) (1 pts) non car on ne peut créer d'instance d'interfaces.

4. [4 points]

- (a) Exception est de type «checked exception» elle doit donc être soit traitée soit déclarée ce qui n'est pas le cas ici
- (b) façons possibles de corriger : i) assortir les entêtes de `approx` et `main` de la déclaration `throws Exception`, ii) traiter l'exception dans `approx` (en y mettant des blocs try/catch), iii) assortir l'entête de `approx` de la déclaration `throws Exception` et traiter l'exception dans `main` (en y mettant des blocs try/catch)

BARÈME : 1 point pour l'explication et 3 points pour une rectification correcte (d'une façon ou d'une autre). Bonus de 0.5 par méthode de résolution alternative donnée en plus.

5. [3 points] Non car la méthode ne pourrait jamais être redéfinie et par conséquent il devient impossible d'avoir des sous-classes instanciables de la classe à laquelle appartient la méthode.

BARÈME : 1.5 point pour citer le fait que la méthode ne peut pas être redéfinie et 1.5 points pour le fait que la classe ne peut alors jamais être instanciée.

6. [6 points]

- (a) (3 pts = 1 par point cité) Dans une interface, les méthodes sont forcément publiques (même si aucun modificateur n'est explicité). La ligne 6 essaie d'implémenter une méthode dictée par l'interface `Drawable` mais en restreignant les droits à `protected`. Or en Java on peut ouvrir les droits dans une hiérarchie mais jamais les restreindre.
- (b) (1 pts) La ligne 12 appelle la méthode non statique `draw` en la prefixant par la classe `Ball`, or cette syntaxe est réservée aux membres statiques
- (c) (2pts = 1.5 + 0.5) i) oui car le compilateur se plaindrait alors de l'absence de `abstract` dans la déclaration de `B` et la ligne 11 ne pourrait plus compiler non plus. ii) non car il est possible d'affecter un `Ball` à un `Drawable`.

7. [4 points] pour coder selon le principe «code to an interface not to an implementation», cela laisse le loisir de changer l'implémentation de `mylist` (en `LinkedList` par exemple) sans affecter le reste du code utilisant `mylist`. **BARÈME** : 1.5 pts pour un argument analogue à celui cité + 2.5 pour l'argument de changer l'implémentation

8. [4 points]

- (a) (2.5 pts = 1.5 pour l'explication et 1 pour la réparation) la liste des comptes fournie en paramètre du constructeur de `Bank` provient de l'extérieur et est directement affecté à l'ensemble des comptes de la banque. Il est possible à celui qui a créé cette liste depuis l'extérieur de la modifier et donc d'impacter la banque sans passer par son API. Pour la réparer il faudrait que la ligne 12 affecte à l'ensemble des comptes une copie de la liste passée en paramètre. Ceci implique de devoir créer un constructeur de copie dans `Account`.
- (b) (1.5pts) La classe `Account` peut devenir une classe imbriquée de la classe `Bank`.

Exercice 3 : Déroulement de programme [23 points]

Le programme affiche :

Justification : Pour B, il faut dire

- les valeurs sont `a=3`, `A.b=5` et `B.b=3` pour les attributs (appel au constructeur par défaut de B)
- appel de `B.m1` (donne 9)
- appel de `B.m2` (donne $25=5*5$) B a deux attributs `b` et l'appel `getB()` retourne la valeur de celui hérité de A
- appel de `A.m1` donne la même chose que `B.m1` car résolution dynamique des liens.
- appel de `A.m2` donne la même chose que `B.m2` car résolution dynamique des liens.
- appel de `A.m3` car pas de `m3` dans B (utilise `B.b` et appelle `B.m1` et `B.m2`, donne $37=3+25+9$)

Pour C, il faut dire (comme pour B)

- les valeurs sont `a=3`, `A.b=4`, `B.b=3`, `c=5` pour les attributs (appel au constructeur par défaut de C)
- appel de `C.m1` (donne $30=10*3$)
- appel de `C.m2` (donne $14=3*3+5$) C a deux attributs `b` et l'appel `getB()` retourne la valeur de celui hérité de B car `getB` polymorphique)
- appel de `C.m1` polymorphisme (donne $30=10*3$)
- appel de `C.m2` polymorphisme (donne 14)
- appel de `A.m3` car pas de `m3` dans C (utilise `B.b` et appelle `C.m1` et `C.m2` car polymorphisme, donne $47=3+30+14$)
- le reste des appels sur `b` est analogue

Pour D, il faut dire

- les valeurs sont 3 et 4 (appel au constructeur par défaut de C) et 5 pour `c` (constructeur par défaut de D qui appelle le constructeur de la ligne 26)
- appel de `D.m1()` : `B.m1` + la définition par défaut de `I.m1` (donne $19=3*3+10$)
- appel de `D.m2()` (= `B.m2` + `B.getB`) qui donne $20=5 * 4$)
- appel de `D.m2(a,b)` (= `D.m1` à cause du polymorphisme + `B.m2`) donne $19 + 5*4 = 39$
- appel de `D.m2(a,c)` (= `C.m1` + `C.m2`) donne $19 + 14 = 33$
- appel de `D.m2(c,c)` (= `C.m1` + `C.m2`) donne $30 + 14 = 44$
- appel de `A.m3` car pas de `m3` dans D (utilise `B.b` et appelle `D.m1` car polymorphisme et `B.m2` car pas de `m2` dans D donne $42=3+19+20$)

BARÈME :

- 1 par ligne d’affichage correcte (hormis les @@@)
- 1 pour tous les @@@
- 1 point par cas B,C,D si l’on voit au travers des explications que les mécanismes sont globalement bien compris.
- Si une ligne d’affichage est incorrecte ou absente on donnera 0.5 point si l’explication correspondante est correcte (invocation des bonnes méthodes)