



# INTRODUCTION A LA PROGRAMMATION

## Corrigé de l'examen final

### Exercice 1 : Conception OO et programmation [50 points]

Correction : voir le fichier `Simulation.java`.

See the file `Simulation.java`.

**BARÈME :** Le détail du barème pour la conception est donné dans la fiche de correction. Si la conception est trop différente mais tient la route, la rubrique “Points ajoutés (autres conception ou bonus)” comptabilise les points donnés (à la place des points de détails de chaque sous-rubrique). Il y a une rubrique “Bonus globaux divers” et “Malus globaux divers” pour comptabiliser des points additionnels ou à retrancher globalement (usage de l’annotation `@Override` (+1), commentaires soignés etc.).

**EN :** The details of the grading schema for the conception part are given in the `exercice1.ods` correction form. If the conception is too different but still makes sense, fill the heading “Added points (other design or bonus)” instead of the detail points in each subheading. There is a section “Bonus globaux divers” and “Malus globaux divers” to count points or to deduct globally (use of the annotation `@Override` (+1), neat comments etc.).

### Question 4

1. car sa méthode `update` n’a pas besoin d’avoir accès à une donnée de type `Environnement`. Le `update` de `Updatable` a une autre sémantique que celui présent dans `Environnement` (il fait évoluer *dans un environnement*).

**EN :** because its `update` method does not need to have access to a data of type `Environnement`. The `update` of `Updatable` has another semantic than the one present in `Environnement` (it makes evolve in an environment).

2. Pour garantir que l’`Environnement` garde la propriété sur les plantes passées en paramètre il faudrait en créer des copies défensives (et donc faire une copie profonde pour les feuilles), ce qui est assez lourd. Si l’on ne fait rien il y aura faille d’encapsulation (des plantes peuvent être modifiées depuis l’extérieur de l’environnement).

**EN :** To guarantee that the `Environnement` keeps the property on the plants passed in parameter it would be necessary to create defensive copies (and thus to make a deep copy for the leaves), which is rather heavy. If nothing is done, there will be an encapsulation leak (plants can be modified

---

from outside the environment).

3. La question était mal posée : l'énoncé voulait dire en empêchant de faire évoluer accidentellement la *source de lumière* (ce point n'a pas été relevé pendant l'examen). La réponse correcte en l'état est donc : il n'est pas possible de le faire car `addLight` a accès sans restriction à toutes les méthodes de `Environment` y compris `update`.

**EN :** The question was badly stated : the statement meant to say "by preventing the *source of light* from accidentally evolving" (this point was not raised during the exam). The correct answer as it stands is therefore : it is not possible to do this because `addLight` has unrestricted access to all methods of `Environment` including `update`.

**BARÈME :** Pour la question 3, mettre un bonus de 2 points s'il y a une discussion censée du type : par contre on pourrait empêcher `addLight` d'invoquer `update` sur le paramètre de type `Light` en faisant implémenter un interface `LightEnvironmentView` à `Light` laquelle n'offrirait pas d'accès à la méthode `update` et en passant en paramètre `LightEnvironmentView` au lieu de `Light`.

**EN :** For question 3, put a bonus of 2 points if there is a meaningful discussion of the type : on the other hand one could prevent `addLight` from invoking `update` on the parameter of type `Light` by making implement an interface `LightEnvironmentView` to `Light` which would not offer access to the method `update` and by passing in parameter `LightEnvironmentView` instead of `Light`.

## Exercice 2 : Concepts de base[35 points]

### 1. [ 5 points]

- (a) L'exécution de la ligne 23 affiche :

`X:0.0`

`Y:0.0`

Justification : Le constructeur de la ligne 13 appelle celui de la ligne 4. Au moment où les lignes 5 et 14 sont appelées, l'attribut `x` de `X` vaut 2 et celui de `Y` vaut zéro (valeur par défaut). Le `getX()` appelé à la ligne 5 est celui de `Y` : on est en train de construire un `Y`, d'où l'affichage du zéro 2 fois.

- (b) pour afficher 10, il faut remplacer la ligne 14 par

```
System.out.println("Y: " + super.getX());
```

**Détail barème :** donné directement dans la fiche de correction.

### 2. [10 points]

- (a) *Une classe abstraite ne peut avoir que des méthodes abstraites* : incorrect.
- (b) *Une classe imbriquée peut accéder à un membre privé de sa classe englobante* : correct
- (c) *Une classe ne peut pas étendre directement plusieurs classes* : correct.
- (d) *Une classe peut implémenter plusieurs interfaces* : correct.
- (e) *Une classe abstraite ne peut pas implémenter une interface* : incorrect
- (f) *Une méthode dans une classe abstraite ne peut pas appeler de méthode abstraite* : incorrect.
- (g) *Une interface n'a que des constructeurs par défaut* : incorrect.
- (h) *Une classe abstraite ne peut pas avoir que des constructeurs par défaut* : incorrect.
- (i) *Une interface peut étendre une autre interface* : correct.
- (j) *Tous les membres d'une interface sont public* : correct.

---

**Détail barème :** donné directement dans la fiche de correction.

3. [11 points]

(a) Le programme affiche :

```
1 2
6 7
6 9
```

Justification : les objets **a** et **b** sont construits avec les constructeurs par défaut par défaut leur attributs **a** et **b** valent les valeurs par défaut 1 et 2 (d'où le premier affichage). Les appels des lignes 19 et 20 ajoutent 5 à chacun de ces attributs d'où l'affichage de 6 7 La méthode **m** affecte au paramètre **a** la valeur du paramètre **b**. Ce paramètre pointe donc vers l'objet **b** de **main** dont la valeur de l'attribut vaut 7. Ajouter 2 à cet attribut via **a** dans **m** revient à modifier l'objet **b** de **main** (son attribut prend la valeur 9). Par contre on reassigne un nouvel objet au paramètre **b** dans **m**, la modification **b.add(3)** modifie cet objet et non le **b** de **main** (passage par valeur systématique en Java).

(b) non il ne compile plus car l'instruction 27 devient illicite.

(c) Oui car **B** n'est étendue par aucune classe, non car si **A** est finale elle ne peut plus être étendue par **B**.

**Détail barème :** donné directement dans la fiche de correction.

4. [9 points]

(a) correcte **C2** et **t C1** sont dans le même paquetage

(b) incorrect **C4** et **C1** ne sont pas dans le même paquetage

(c) incorrect elle est protégée et **C4** et **c1** ne sont pas dans le même paquetage

(d) correct car **C3** étend **C1**

(e) correct car ils sont dans le même paquetage.

**Détail barème :** donné directement dans la fiche de correction.

## Exercice 3 : Déroulement de programme [15 points] (bonus)

1. (a) Oui cela ne pose pas de problème.

(b) Non car il n'y a pas de méthode **c(int)** dans **R0**.

(c) Oui car on a casté l'objet en **R1** où la méthode existe. (attention dans l'énoncé il manquait une paire d'accolades : l'instruction aurait dû être : **System.out.println(((R1)r0).c(2)).** Ceux qui répondent "non car le cast devrait se faire comme ceci **((R1)r0).c(2)**" ont tous les points.

2. Le programme affiche :

```
***/** **/** */**
360
**/** **/** */**
300
**/** **/** */**
240
```

---

Justification :

- (a) (2 point) ligne 76 le programme crée une instance `r0` de `R1` et la stocke dans une variable de type `R0` (ok : compatibilité ascendante des types). Le constructeur de `R1` est appelé qui appelle celui de `R0` en lui passant en paramètres `{3, 2, 1}` et `Rank.High`. le constructeur de `R0` itère sur chaque élément de `lst=3, 2, 1` pour créer les objets `S` contenant respectivement les chaînes `***/**`, `**/**`, `*/**` : la valeur en dessus de `/` est une `String` de `lst[i]` `'*'`, la valeur en dessous est retournée par la méthode `shining` du type énuméré sur `Rank.High`. Ces objets `S` sont stockées dans l'attribut `c` de l'objet `r0`
- (b) (1 point) ligne 77 le programme crée une instance `rint` de `R1`, initialise son attribut `r` à `Rank.Medium`, et la stocke dans une variable de type `I` (ok car `R1` descend de `R0` qui implémente `I`). La procédure de construction se fait de façon analogue et ce sont les chaînes `***/**`, `**/**`, `*/**` qui sont stockées dans l'attribut `c`.
- (c) (1 point) ligne 78 le programme crée une instance `r1` de `R1`, initialise son attribut `r` à `Rank.Low` et la stocke dans une variable de type `R1`. La procédure de construction se fait de façon analogue et ce sont les chaînes `***/**`, `**/**`, `*/**` qui sont stockées dans l'attribut `c`.
- (d) (1 point) la ligne 80 appelle la méthode `p` de la ligne 84 en lui passant `r0` en paramètre. La ligne 85 appelle la méthode `toString` de la ligne 58 (polymorphisme).
- (e) (2 points) la méthode `toString` traduit le contenu de la liste `c` en une chaîne de caractère correspondant `***/**`, `**/**`, `*/**` devient `***/** **/** */**`.
- (f) (2 points) La ligne 86 appelle la méthode `c()` de la ligne 52 sur `r0`. cette dernière calcule un score relatif à l'attribut `c` : la méthode `p()` de la ligne 70 est appelée lors du calcul (polymorphisme), la méthode `c()` de la ligne 11 est aussi appelée. Le calcul effectué est somme de `p() * s.c()` pour chaque `s` de l'ensemble `c`. `p()` retourne la constante `I.CST` qui vaut 2. `s.c()` retourne `10 (I.super.c()) * longueur de la chaîne stockée dans s`. Le calcul effectué pour l'ensemble `***/**`, `**/**`, `*/**` est donc  $10 \cdot 7^2 + 10 \cdot 6^2 + 10 \cdot 3^2 = 360$ .
- (g) (1 point) les lignes 81 et 82 font pareil mais pour les objets `rint` et `r1`. `***/**`, `**/**`, `*/**` produit la chaîne `***/** **/** */**` et le score  $10 \cdot 6^2 + 10 \cdot 5^2 + 10 \cdot 4^2 = 300$ .
- (h) (1 point) la ligne 82 fait pareil mais pour l'objet `r1`. `***/**`, `**/**`, `*/**` produit la chaîne `***/** **/** */**` et le score  $10 \cdot 5^2 + 10 \cdot 4^2 + 10 \cdot 3^2 = 240$

**BARÈME :** Pour les sous-questions (voir fiche de correction). Pour l'affichage :

- 1.5 point par ligne d'affichage correcte
- Pour la justification 11 points (voir les points par éléments d'explication suggéré dans le corrigé ci-dessus). Si les explications sont plus verbeuses et/ou très différentes dans le forme du corrigé, donnez les points au pro rata des lignes de code (bien) expliquées.

Attention au mode de calcul : on peut avoir tous les points sans donner l'affichage explicitement si on a donné des explications complètement correctes. On peut avoir presque tous les points s'il y a eu juste l'affichage mais pas les justifications.