Université A. Belkaïd
Tlemcen
Faculté des Sciences
Département d'Informatique

Corrigé-type de l'Examen final de Programmation Objet

Année universitaire 2015-2016
2e année licence informatique
Programmation Objet

Jeudi 21 janvier 2016
Durée : 1h30

Questions de cours : (4 pts)

1) `ActionListener` est une interface. (1 pt)

Explication : **`ImageViewer`** implémente (**`implements`**) l'interface **`ActionListener`**.

2) Pour qu'il n'y ait pas d'erreur à la compilation la classe `ImageViewer` doit contenir la (re)définition de la méthode `actionPerformed()` héritée de `ActionListener`. (1 pt)

Explication : **`ImageViewer`** n'étant pas une classe abstraite (pas de **`abstract`**), elle doit obligatoirement (re)définir toutes les méthodes abstraites qu'elle a héritées, sinon le compilateur affichera : `error: ImageViewer is not abstract and does not override abstract method actionPerformed(ActionEvent) in ActionListener`

3) Il n'y a pas de classe `ActionAdapter` dans l'API de Java car `ActionListener` n'a qu'une seule méthode. (2 pts)

Explication : Les classes abstraites Adapter héritent (implémentent) l'interface Listener correspondante et (re)définissent toutes les méthodes héritées avec un corps vide (un bloc d'instructions vide).
Quand une classe hérite de l'Adapter, cela lui permet d'implémenter (indirectement) l'interface Listener et redéfinir uniquement les méthodes dont elle a besoin. Les autres méthodes héritées étant déjà définies dans l'Adapter, elles ne sont plus abstraites et le compilateur ne se plaindra pas.
Dans le cas de **`ActionListener`**, il y a une seule méthode à redéfinir. Une classe **`ActionAdapter`** ne servirait donc à rien.

Exercice 1 : (9 pts)

1) Il y a 6 erreurs :

Ligne 18 : erreur à la compilation.
Explication : **`super()`** est un appel au constructeur hérité, il ne peut se faire que dans la première ligne d'un constructeur, pas dans une autre méthode.

Ligne 23 : erreur à la compilation.
Explication : La classe **`B`** hérite les méthodes de **`A`** et de **`Object`**. **`super.c()`** est un appel à la méthode **`c()`** héritée. Or il n'y a pas de méthode **`c()`** dans la classe A ni dans **`Object`**.

Ligne 33 : erreur à la compilation.
Explication : **`a2`** est déclaré comme une référence d'un objet de type **`B`**. On ne peut pas y mettre la référence d'un objet de type **`A`** car **`A`** n'est pas un sous-type de **`B`**.

Ligne 43 : erreur à la compilation.
Explication : **`a1.c()`** est un appel à la méthode **`c()`** de **`a1`**. Or le type déclaré (statique) de **`a1`** est **`A`** et **`A`** n'a pas de méthode **`c()`**.

Ligne 44 : erreur à la compilation.
Explication : **`b1.c()`** est un appel à la méthode **`c()`** de **`b1`**. Or le type déclaré (statique) de **`b1`** est **`A`** et **`A`** n'a pas de méthode **`c()`**. Même si le type dynamique de l'objet référencé par **`b1`** est **`B`** (qui a une méthode **`c()`**), le compilateur ne l'accepte pas car il n'examine que les types statiques.

Ligne 47 : erreur à l'exécution.
Explication : **`((B) a1)`** est une tentative de transtypage (cast) de l'objet **`a1`** du type statique **`A`** vers le type **`B`**. **`B`** étant un sous-type de **`A`**, le compilateur l'accepte car il n'examine que les types statiques.
Cependant l'objet référencé par **`a1`** est de type dynamique **`A`** (pas **`B`**). Le transtypage échouera donc à l'exécution en provoquant une **`ClassCastException`** (`A cannot be cast to B`).

2) Après avoir supprimé les lignes 18, 23, 33, 43, 44, 47 qui contiennent les erreurs et les lignes 37, 41, 45 et 49 qui contiennent la variable a2 qui n'existe plus (elle était déclarée en ligne 33), l'exécution du programme donne :

`a de A dans A@`*<adr. de a1>* ← *ligne 35* *("A@<adr. de a1>" désigne la chaîne renvoyée par la méthode*
`a de A dans B@`*<adr. de b1>* ← *ligne 36* *toString() de Object. Elle contient l'adresse de l'objet a1 en hexa.)*
`a de A dans B@`*<adr. de b2>* ← *ligne 38*
`b de A dans A@`*<adr. de a1>* ← *ligne 39*
`b de B dans B@`*<adr. de b1>* ← *ligne 40*
`b de B dans B@`*<adr. de b2>* ← *ligne 42*
`c de B dans B@`*<adr. de b2>* ← *ligne 46*
`c de B dans B@`*<adr. de b1>* ← *ligne 48*
`c de B dans B@`*<adr. de b2>* ← *ligne 50*

N.B. : Ceci est l'exercice 12.5 p.307 du livre de H. Bersini « La programmation orientée objet » 5e ed. Eyrolles (disponible à la bibliothèque de la faculté), que j'ai légèrement modifié.

Explications :
…

Exercice 2 : (7 pts)

1) et 2) classe Pile :

```
public class Pile
{
    private Liste liste;
    public Pile(int taille){
        liste = null; // pas de taille initiale
    }

    public boolean estVide(){
        return (liste == null);
    }

    public void empiler(ElementPile element){
        if (element == null)
            return;
        liste = new Liste(element, liste);
    }

    public ElementPile sommet(){
        if (estVide())
            return null;
        return liste.tete();
    }

    public ElementPile depiler(){
        if (estVide())
            return null;
        ElementPile sommet = sommet();
        liste = liste.decapiter();
        return sommet;
    }

    public void afficher(){
        if (estVide()) {
            System.out.println("Pile vide.");
            return;
        }
        System.out.print("(");
        liste.afficher();
        System.out.println(")");
    }

    public void empiler(int valeur){
        ElementPile element = new ElementPile(valeur);
        empiler(element);
    }
}
```

Question Bonus : méthode `afficher()` itérative :

```
    public void afficher() {
        // construction d'une copie de cette liste chaînée
        // en commençant par la tête de liste (==> ordre inverse)
        Liste inv = null;
        for(Liste p=this ; p!=null ; p=p.suivant){
            inv = new Liste(p.element, inv);
        }
        // parcours et affichage des éléments de la nouvelle liste
        for(Liste p=inv ; p!=null ; p=p.suivant){
            if (p.element != null){
                p.element.afficher();
                if(p.suivant != null)
                    System.out.print(", ");
            } else
                System.out.print("null");
        }
    }
```