Aide-mémoire de POOGL pour l'examen

Laure Danthony

D'après le cours de Vincent Rialle, janvier - mai 2001

Table des matières

1 Classes, objets

- En Java, chaque classe dérive d'une seule classe parente dont elle hérite. La classe mère est *Objet*. [L2 t6-7]
- Objet = identité + état + comportement [L2 t3]
- Notion d'**héritage** [L2 t19-22]
- **Encapsulation** [L2 t15-16]
- **Classe interne** [L2 t13-14]
- Une **méthode abstraite** [L2 t12] ne possède pas de définition, donc se redéfinit dans une classe dérivée. Une classe abstraite ne peut pas être instanciée (new). Une classe dérivée d'une classe abstraite ne redéfinissant pas toutes les méthodes est elle-même abstraite.
- Les **exceptions** sont des objets comme les autres [L3 t13-26]
- Entrées-sorties [L11 *]

2 Aide-mémoire de base

- Les types de base sont : *float, double, int, char, boolean*! *String* n'est pas un type simple.
- Déclaration de tableau : `int[] valeurint = new int[10]`, ils sont indexés de 0 à 9. `double[][] = new double[3][5]` a 3 colonnes et 5 lignes.
- Conversion : ad-hoc (`int valeur_int = 14`, `float valeurl=valeur_int`, `String str=''texte''+valeur_int`) ou à l'aide de méthodes (on peut vérifier auparavant à l'aide de *instanceof*.

3 Mots-clefs

- Portée des classes et des méthodes :

	Classe	Classe dérivée	Package	Public
package	oui	non	oui	non
private	oui	non	non	non
protected	oui	oui	oui	non
public	oui	oui	oui	oui

- static/final/volatile : [L3 t5-6]
  - **static** : un objet ou une variable statique n'existe qu'une fois, même si plusieurs instances de la classe ont été créées. En particulier, une méthode main est statique.
    - Une variable statique est partagée par toutes les instances d'une classe.
    - Les méthodes statiques ne peuvent pas accéder à l'objet *this*.
    - La méthode *main* ne peut faire appel qu'à des membres statiques de la classe englobante.
    - Une classe interne de méthode ne peut avoir de membre statique.
  - **final** (classes, méthodes, variables) : elles sont constantes, elles ne peuvent pas êtres redéfinies : exemple `public static final Color black`.
  - **volatile** : modifiable.

4 Héritage versus Composition

4.1 Héritage

C'est la relation de sous-classe [L5 t3-7] : A sous classe de B, que l'on schématise par la figure 1.

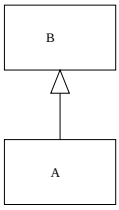

FIG. 1 – A hérite de B

Déclaration :

```
class A extends B{...
class Pomme extends Fruit{...
```

Caractéristiques :

- liaison dynamique : utilisation des méthodes redéfinies dans la sous-classe.
- polymorphisme ⇒ réutilisation du code de la sur-classe.
- **inconvénients** Si on change le code d'une classe (par exemple si on redéfinit dans *B=Fruit* la signature d'une méthode), on doit répercuter les changement sur tous les descendants.
- **inconvénients** Si on modifie l'interface de A, il faut vérifier qu'elle est compatible avec B (par exemple, les doubles définitions sont interdites).

4.2 Composition

C'est l'utilisation d'une instance de B à l'intérieur de la classe A : A est dite de premier-plan et B d'arrière-plan. On utilise le schéma de la figure 2.

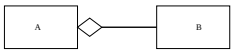

FIG. 2 – Composition

Déclaration :

```
class A {
     private B unB; ...
```

Caractéristiques :

- Notion de délégation de méthode : on peut définir des méthodes en déléguant le code à B.
- La sous-classe devient classe de premier-plan.
- La super-classe devient classe d'arrière-plan.

- **Avantage** : on peut définir des méthodes de A à partir de méthodes de B (de même nom) mais de signatures différentes [L5 t11-12].
- **Avantage** : on peut modifier A sans modifier B. Si on modifie B on doit modifier l'implémentation de A mais pas nécessairement son interface.

Comment choisir ? L5 t15-16

4.3 Interface + association + délégation

voir L5 t17-21.

5 Applet

Déclaration :

```
public class MonApplet extends java.applet.Applet{
...
```

Les méthodes d'une applet qui peuvent/doivent être redéfinies :

- **init()** : se charge avant que l'applet soit affichée ;
- **start()** : démarre l'applet ;
- **paint()** : affiche des objets de la classe *Graphics*
- **stop()** : arrête le programme (par exemple en cas de minimisation de la fenêtre, ou un autre programme a été lancé en premier-plan. Quand on maximise la fenêtre, la méthode paint est réinvoquée) ;
- **destroy()** : invoquée lorsque l'on quitte le navigateur ou que l'on charge une autre page.

6 Interface Graphique et événements

6.1 Les outils

- **Architecture modèle-vue-contrôleur** : [l7 t3]
- **Evénements** : chaque objet source qui doit être à l'origine d'un événement doit disposer d'écouteurs d'actions (sous-classe de *EventListener*, utiliser "implements"), ou d'adapteurs (*Adapter*) : dans ce deuxième cas on n'est pas obligé de réécrire tous les cas d'événements [L7 t10-12]
- **Contexte graphique** : [L7 t15] IUG = interface graphique utilisateur.

Il faut distinguer :

- La mise en place dans des conteneurs graphiques (composants, conteneurs et gestionnaires de mise en page).
- La connexion des évenements système et les actions à lancer (notion de gestion d'événement).

6.2 Exemple important de gestion d'application

Une bonne façon de segmenter le travail est la suivante : il faut séparer :

- la classe de l'interface graphique (mise en page avec les adaptateurs, c'est-à-dire que l'on fait appel à la classe Aiguillage pour les implémenter) ;
- la classe de l'application, qui contient les méthodes de l'application en elle-même et la méthode main ;

- la ou les classes Aiguillage qui implémentent les écouteurs, ils sont un relai entre l'appel de l'écouteur à l'aide d'un événement graphique et les actions associées qui sont des méthodes de l'application.

7 Threads et communications Internet

- Introduction aux threads [L1 t26] et notion de sécurité [L1 t27]
- exemple d'utilisation [L10 t1]
- adressage commun ⇒ programmation concurrente et synchronisation.
- on peut étendre la classe *Thread* ou implémenter la classe *Runnable* (cas des applets par exemple).
- Quelques méthodes de thread :
  - **run()** : il faut la redéfinir en donnant le code à faire faire au thread ;
  - **start()** : appelle *run* mais retourne immédiatement ;
  - **yield()** passe son tour ;
  - **setPriority(nb)** fixe une priorité entre 1 et 10 (faible priorité) ;
  - **join()** : lorsqu'on atteint `t.join` on est certain que t est terminé ;
  - **notify()** : permet de débloquer un thread.
- mot-clef **synchonized** : [L10 t19-26] lorsqu'une méthode d'instance est déclarée *synchronized*, elle vérouille l'instance concernée (mais cela n'empêche pas cette même méthode d'être invoquée sur un autre objet.
- Thread et Applet [L10bis] : Applet peut implémenter *Runnable*, il faut jongler avec les rôles conjugués de *run()* (thread) et *init(), start(), stop(), destroy()* de l'applet.

8 UML

Description d'UML [Leçon 8]

- Il permet d'offrir un cadre coceptuel de représentation pour la modélisation de processus, de systèmes, de mécanismes ... ;
- ce sont des symboles graphiques qui permettent d'incarner les concepts ;
- des outils comme *Together* permettent de générer du code directement à partir de diagrammes.

8.1 Représentation des relations

- **Relation de spécification :** c'est la relation classe/sous-classe. (figure 3)
- **Relation d'association :** connexion sémantique bidirectionnelle. (figure 4)
- **Relation d'agrégation :** A est un ensemble de Bs. (figure 5)
- **Relation de composition :** A est composé de tous les Bs. (figure 6)

REMARQUE : la distinction entre ces deux dernières relations n'existe pas en JAVA.

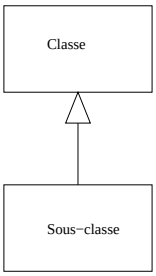

FIG. 3 – Spécification

FIG. 4 – Association

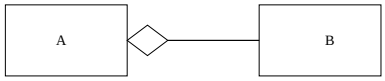

FIG. 5 – Agrégation

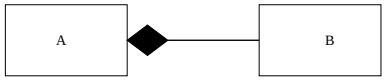

FIG. 6 – Composition

8.2 Différents diagrammes

Différents diagrammes qui ont des objectifs différents :

- **Diagrammes de classes** [L8 t15-16] : représentation des relations entre classes (en général).
- **Diagrammes d'objet** [L8 t 17-18] : représentation des objets et de leurs relations à un instant donné de l'exécution.
- **Cas d'utilisation** [L8 t19-23] : description en terme d'acteurs et d'actions, simulation du système global.
- **Diagramme de séquence** [L8 t24-26] : représentation en termes temporels.
- **Diagramme de collaboration** [L8 t27-28] : objets et messages, représentation en termes spatiaux-temporels.
- **Diagramme d'état-transition** [L8 t29-31] : représentation des états des objets par un automate dont les transitions sont étiquetées par des actions.