

Mettons Java au travail

Conception Orientée Objets

Email : David.Teller@univ-orleans.fr

Aujourd'hui L'objectif de cette séance de TDs est de réviser les bases de Java et de la Programmation Orientée Objets. Progressivement, au cours des semaines qui viennent nous allons avancer vers la conception d'un jeu de Démineur complet, avec interface graphique, graphismes et sons.

Note À partir de la deuxième séance, les TDs seront notés. Entre deux séances, je vous suggère fortement de relire et de comprendre les corrections précédentes. J'interrogerai.

Note Lisez rapidement toute la fiche avant de commencer les exercices.

Note Les exercices marqués d'une étoile * doivent impérativement avoir été faits d'une séance sur l'autre, puisque nous nous appuierons sur eux pour construire le Démineur. J'interrogerai et vous aurez besoin de les maîtriser pour l'examen de fin de semestre.

1 Révisions générales

1.1 Langage impératif

Le langage Java fait partie de la famille des *langages de programmation impératifs*, ce qui signifie notamment qu'il fait appel à des variables, à des affectations, à des tableaux et des boucles itératives. Revoyons la signification de tous ces mots.

Variables

Exercice 1. Qu'est-ce qu'une *variable* ? Qu'est-ce qu'une *affectation* ?

Exercice 2. En Java, que signifie $x = x+1$?

Tableaux

Exercice 3. Qu'est-ce qu'un *tableau* ? À quoi servent les tableaux ?

Exercice 4. En Java, comment crée-t-on un tableau ?

Exercice 5. En Java, que signifie $a[0] = 1$?

Exercice 6. Comment est représenté un tableau en mémoire ?

Boucles

Exercice 7. Qu'est-ce qu'une boucle ? À quoi servent les boucles ?

1.2 Langages compilés

Le langage Java fait aussi partie de la famille des *langages de programmation compilés*, ce qui signifie que le code source passe par une étape de compilation avant d'être exécuté.

Exercice 8. Qu'est-ce qu'un *code source* ? Que fait la phase de *compilation* d'un programme ?

Exercice 9. Que fait la phase d'*analyse de types* d'un programme ?

Exercice 10. Comment fonctionne un langage *non compilé* ?

Exercice 11. Quels sont les avantages et les inconvénients de la compilation ?

2 (Ré-)introduction à la POO

La Programmation Orientée Objets (ou POO) est une technique de programmation moderne qui permet de produire des programmes plus faciles à comprendre, plus faciles à étendre et plus faciles à réutiliser dans d'autres programmes. En particulier, la Programmation Orientée Objets s'avère extrêmement utile pour le développement d'interfaces graphiques (toute l'interface de Windows XP, de Mac OS X ou de KDE est construite à l'aide de la POO) et pour le développement de projets de grande taille (ex. Windows XP entier, Mozilla/Firefox, ...)

De nombreux langages de programmation permettent plus ou moins de POO : Java combine POO et programmation procédurale (que vous avez étudiée l'an dernier), OCaml combine POO et programmation fonctionnelle (que vous étudierez au deuxième semestre), C++ combine POO et programmation procédurale orientée système, etc.

2.1 Diagrammes de classes

Définition 1. *Diagramme de classes*

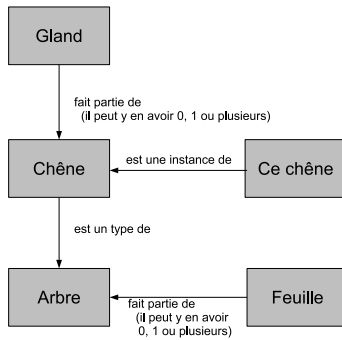
Un diagramme de classe est un graphe constitué de noms liés par les relations suivantes :

- “est un type de” (“A est un type de B” si tous les éléments de A sont des éléments de B – on dit aussi que “B généralise A” ou “la classe A est une spécialisation de la classe B”, ou encore “A est une sous-classe de B”)
- “fait partie de la définition de” (A “fait partie de la définition de” B si tous les éléments de B contiennent A – on dit aussi que “A fait partie de B” ou “B a un champ A”)
- “est une instance de” (A “est une instance de” B si A est l'un des éléments de l'ensemble B – on dit aussi que A “est un des” B)

Un diagramme de classes sert à décrire le contexte d'un problème sous la forme de relations entre objets. Nous compléterons plus tard cette définition d'un diagramme de classes pour ajouter l'état d'un objet.

Exemple 2.

- Chêne est un type d'Arbre
- Feuille fait partie d'Arbre
- Gland fait partie de Chêne
- Le chêne de la cour est une instance de Chêne.



Exercice 12. Dessinez un diagramme de classes pour les noms suivants :

- L'ensemble de toutes les voitures.
- L'ensemble de tous les véhicules.
- L'ensemble de toutes les bicyclettes.
- L'ensemble des objets.
- Mon vélo.
- Un vélo-moteur.
- Une roue.
- Un volant.
- Un passager.
- Une vitesse.
- L'ensemble des lampes à huile.
- L'ensemble des objets qui consomment de l'huile ou de l'essence.
- Le fait de conduire.

Exercice 13. Dessinez un diagramme de classes pour les noms suivants :

- Carte à jouer
- Main de poker
- Valeur de la carte
- As
- Roi
- Dame
- Trèfle
- Pique
- Cœur
- Carreau
- Joueur de poker
- Joueur humain.
- Joueur contrôlé par l'ordinateur.
- Combinaison de cartes
- Une paire
- Deux paires

Note : vous allez devoir ajouter au moins un nom à la liste.

Exercice 14. (*) Dessinez un diagramme de classes pour un jeu de démineur (cf. figure ?):

- Case.
- Case minée.
- Coordonnées (x,y).
- Taille du plateau.
- Plateau.
- Nombre total de mines.
- Nombre total de drapeaux.
- Drapeau.
- Point d'interrogation.
- Temps écoulé.
- Nombre de mines dans des cases adjacentes.



Figure 1.

2.2 Des diagrammes aux classes

Le diagramme de classes de l'exercice 13 peut parfaitement servir au développement d'un jeu vidéo de poker. De même, celui de l'exercice 14 va nous servir au développement d'un jeu de démineur, au cours du semestre.

Une fois le diagramme de classes dessiné, il est assez simple de le traduire sous la forme d'un squelette de programme Java – certains logiciels sont d'ailleurs capables de faire cela presque automatiquement.

Ainsi, du diagramme de l'exemple 2, on peut tirer le code suivant :

```
//Dans le fichier Feuille.java
public class Feuille
{
    //...
}

//Dans le fichier Arbre.java
public class Arbre
{
    Feuille feuillage[]; //Les feuilles de l'arbre
    //...
}

//Dans le fichier Gland.java
public class Gland
{
```

```
    //...
}

//Dans le fichier Chene.java
public class Chene extends Arbre
{
    Gland fruits[]; //Les glands du chêne
    //...
}
```

Exercice 15. Qu'est-ce qu'un *champ* d'un objet ?

Exercice 16. Pour le moment, quels sont les champs de la classe Arbre ?

Exercice 17. Qu'avons-nous ajouté par rapport au diagramme de classe initial ? Que manque-t-il ? Comment pourrions-nous compléter ce code source pour obtenir le diagramme complet ?

Exercice 18. De la même manière, transformez le diagramme de l'exercice 12 en code source.

Exercice 19. Idem pour le diagramme de l'exercice 13.

Exercice 20. (*) Idem pour le diagramme de l'exercice 14.

Note : nous allons nous réserver de ce code source dans les séances qui suivent.