Programmation Orientée Objets avec Java

Pr. Abdelhak LAKHOUAJA

Département de Mathématiques et Informatique
Faculté des Sciences
Oujda

a.lakhouaja@ump.ma

http://lakhouaja.oujda-nlp-team.net/

SMI-S5
Année universitaire : 2016/2017

Chapitre 1
Introduction

Introduction

Les langages orientés objets prennent en charge les quatre caractéristiques importantes suivantes :

1. Encapsulation
2. Abstraction
3. Héritage
4. Polymorphisme

La plateforme Java

Les composantes de la plateforme Java sont :

- Le langage de programmation.
- La machine virtuelle (The Java Virtual Machine - JVM).
- La librairie standard.

Langage Java

Java est compilé et interprété :

- Le code source Java (se terminant par `.java`) est compilé en un fichier Java bytecode (se terminant par `.class`)
- Le fichier Java bytecode est interprété (exécuté) par la JVM
- La compilation et l'exécution peuvent se faire sur différentes machines
- Le fichier bytecode est portable. Le même fichier peut être exécuté sur différentes machines (hétérogènes).

Premier programme en Java

Un exemple qui permet d'afficher le message `Bonjour - SMI-S5` :

```
public class Bonjour {
    public static void main(String[] args) {
        System.out.println("Bonjour - SMI-S5");
    }
}
```

- Le programme doit être enregistré (**obligatoirement**) dans un fichier portant le même nom que celui de la classe **Bonjour.java**.
- Pour compiler le programme précédent, il faut tout d'abord installer l'environnement de développement **JDK** (**J**ava **D**eveloppment **K**it).

Installation sous Linux

Installer openjdk-7, en tapant la commande :

```
sudo apt-get install openjdk-7-jdk
```

Ou bien, télécharger la version de jdk (jdk-7uxy-linux-i586.tar.gz ou jdk-7uxy-linux-x64.tar.gz) correspondant à votre architecture (32 ou 64 bits) de :

`http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html`

Décompresser le fichier téléchargé en tapant la commande :

- Systèmes 32 bits : `tar xfz jdk-7uxy-linux-i586.tar.gz`
- Systèmes 64 bits : `tar xfz jdk-7uxy-linux-x64.tar.gz`

Remplacer xy par le numéro de mise-à-jour, par exemple 55.

Installation sous Linux

Ajouter dans le fichier .bashrc (gedit ~/.bashrc) la ligne suivante :

```
PATH=~/jdk1.7.0_xy/bin:$PATH
```

Faites attention aux majuscules ! Linux (comme Java et C) est sensible à la casse (A≠a).

Installation sous Windows

Télécharger la version de jdk (jdk-7uxy-windows-i586.exe ou jdk-7uxy-windows-x64.exe) correspondant à votre architecture (32 ou 64 bits) de :

`http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html`

Exécuter le fichier téléchargé et ajouter `C:\Program Files\Java\jdk1.7.0\_xy\bin` au chemin, en modifiant la variable **path**. La valeur de **path** doit ressembler à ce qui suit :

```
C:\WINDOWS\system32;C:\WINDOWS;C:\Program Files\Java\jdk1.7.0_xy\bin
```

Installation sous Windows

Pour modifier la valeur de path :

1. cliquer sur **démarrer** puis **Panneau de configuration** puis **Système et sécurité** puis **Paramètres système avancés**
2. cliquer sur **Variables d'environnement** puis chercher dans **variables système**, **Path** et modifier son contenu.

Compilation

Dans une console, déplacez vous dans le répertoire ou se trouve votre fichier et tapez la commande suivante :

```
javac Bonjour.java
```

Après la compilation et si votre programme ne comporte aucune erreur, le fichier `Bonjour.class` sera généré.

Exécution

Il faut exécuté le fichier .class en tapant la commande (sans extension) :

```
java Bonjour
```

Après l'exécution, le message suivant sera affiché.

```
Bonjour - SMI-S5
```

Déclaration

System.out.println("Bonjour - SMI-S5") ;

- **System** : est une classe
- **out** : est un objet dans la classe System
- **println()** : est une méthode (fonction) dans l'objet out. Les méthodes sont toujours suivi de ().
- les points séparent les classes, le objets et les méthodes.
- chaque instruction doit se terminer par " ;"
- Bonjour - SMI-S5 : est une chaîne de caractères.

Première classe

Dans Java, toutes les déclarations et les instructions doivent être faites à l'intérieure d'une classe.

```
public class Bonjour
```

veut dire que vous avez déclaré une classe qui s'appelle **`Bonjour`**.

Le nom d'une classe (**identifiant**) doit respecter les contraintes suivantes :

- l'identifiant doit commencer par une lettre (arabe, latin, ou autre), par _ ou par $. Le nom d'une classe ne peut pas commencer par un chiffre.
- un identifiant ne doit contenir que des lettres, des chiffres, _, et $.
- un identifiant ne peut être un mot réservé.
- un identifiant ne peut être un des mots suivants : `true`, `false` ou `null` . Ce ne sont pas des mots réservés mais des types primitifs et ne peuvent pas être utilisés par conséquent.

Mots réservés

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Même si **const** et **goto** sont des mots réservés, ils ne sont pas utilisés dans les programmes Java et n'ont aucune fonction.

Recommandations concernant les noms des classes

En java, il est, par convention, souhaitable de commencer les noms des classes par une lettre majuscule et utiliser les majuscules au début des autres noms pour agrandir la lisibilité des programmes.

Quelques noms de classes valides

Nom de la classe	**description**
Etudiant	Commence par une majuscule
PrixProduit	Commence par une majuscule et le le deuxième mot commence par une majuscule
AnnéeScolaire2014	Commence par une majuscule et ne contient pas d'espace

Quelques noms de classes non recommandées

Nom de la classe	**description**
etudiant	ne commence pas par une majuscule
ETUDIANT	le nom en entier est en majuscule
Prix_Produit	_ n'est pas utilisé pour indiqué un nouveau mot
annéescolaire2014	ne commence par une majuscule ainsi que le deuxième, ce qui le rend difficile à lire

Quelques noms de classes non valides

Nom de la classe	**description**
Etudiant#	contient #
double	mot réservé
Prix Produit	contient un espace
2014annéescolaire	commence par un chiffre

Méthode main

Pour être exécuté, un programme Java doit contenir la méthode spéciale **main()**, qui est l'équivalent de **main()** du langage C.

- **`String[] args`**, de la méthode **main** permet de récupérer les arguments transmis au programme au moment de son exécution.
- **`String`** est une classe. Les crochets ([]) indiquent que **args** est un tableau (voir plus loin pour plus d'informations sur l'utilisation des tableaux).
- Le mot clés **void**, désigne le type de retour de la méthode **main()**. Il indique que **main()** ne retourne aucune valeur lors de son appel.
- Le mot clés **static** indique que la méthode est accessible et utilisable même si aucun objet de la classe n'existe.
- Le mot clés **public** sert à définir les droits d'accès. Il est obligatoire dans l'instruction **public static void main(String[] args)** et peut être omis dans la ligne **public class Bonjour**.

Commentaires

Les commentaires peuvent s'écrire sur une seule ligne ou sur plusieurs ligne, comme dans l'exemple suivant :

```
/*
Premier programme en Java
contient une seule classe avec une seule methode
*/
public class Bonjour {
    //methode principale
    public static void main(String[] args) {
        System.out.println("Bonjour - SMI-S5");
    }
}
```

Commentaires pour la documentation Java

Les commentaires qui commencent par /** et se terminent par */ servent pour générer une documentation automatique.

Exemple :

```
/**
 * C'est une classe de test
 */
public class TestCommentaires {
    /**
     * Methode main (principale)
     *
     * @param args arguments
     */
    public static void main(String[] args) {
        System.out.println("Bonjour - SMI-S5 ");
    }
}
```

Commentaires pour la documentation Java

Pour générer la documentation sous format HTML, il faut taper la commande :

javadoc TestCommentaires.java

Données primitifs

Java est un langage (presque) purement orienté objets, puisqu'il permet la déclaration de types de données primitifs.

Lors de la déclaration d'une variable sans qu'aucune valeur ne lui soit affecté, la variable sera non initialisée.

Par exemple, lors de déclaration de la variable **age** de type **int** :

```
int age;
```

si vous essayez de l'utiliser dans une expression ou de l'afficher, vous allez recevoir une erreur lors de la compilation (contrairement au langage **C**) indiquant que la variable n'est pas initialisée (The local variable age may not have been initialized).

Types numériques

En java, il existe 4 types entiers et 2 types réels :

Type	**Valeur Minimale**	**Valeur Maximale**	**Taille en octets**
byte	-128	127	1
short	$-32\ 768$	32 767	2
int	$-2\ 147\ 483\ 648$	2 147 483 647	4
long	$-9\ 223\ 372\ 036\ 854\ 775\ 808$	$9\ 223\ 372\ 036\ 854\ 775\ 807$	8
float	$-3.4 * 10^{38}$	$3.4 * 10^{38}$	4
double	$-1.7 * 10^{308}$	$1.7 * 10^{308}$	8

Types numériques

Remarques :

1. Par défaut, une valeur comme 3.14 est de type **double**, donc pour l'affecter à une variable de type **float**, il faut la faire suivre par la lettre **f** (majuscule ou minuscule).
   ```
   float pi=3.14F, min=10.1f;
   ```
2. Par défaut les entiers sont de type **int**. Pour affecter une valeur inférieure à $-2\ 147\ 483\ 648$ ou supérieure à $2\ 147\ 483\ 647$ à un **long**, il faut la faire suivre par la lettre **l** (majuscule ou minuscule).
   ```
   long test=4147483647L;
   ```

Caractères

On utilise le type **char** pour déclarer un caractère. Par exemple :

```
char c='a';
char etoile='*';
```

Les caractères sont codés en utilisant **l'unicode**. Ils occupent 2 octets en mémoire. Ils permettent de représenter **65 536** caractères, ce qui permet de représenter (presque) la plupart des symboles utilisés dans le monde.

Séquences d'échappement

Séquence d'échappement	**Description**
`\n`	nouvelle ligne (new line)
`\r`	retour à la ligne
`\b`	retour d'un espace à gauche
`\\`	\ (back slash)
`\t`	tabulation horizontale
`\'`	apostrophe
`\"`	guillemet

Type boolean

Il permet de représenter des variables qui contiennent les valeurs vrai (**true**) et faux (**false**).

```
double a=10, b=20;
boolean comp;
comp=a>b; //retourne false
comp=a<=b; //retourne true
```

Constantes

Pour déclarer une constante, il faut utiliser le mot clés **final**. Par convention, les constantes sont écrites en majuscule.

Exemple :

```
final int MAX = 100;
final double PI = 3.14;
...
final int MAX_2 = MAX * MAX;
```

Expressions et opérateurs

Comme pour le langage C, Java possède les opérateurs :

- arithmétiques usuels (+, -, *, /, %)
- de comparaison (<, <=, >, >=, ==, !=).

On retrouve aussi les expressions arithmétiques, les comparaisons et les boucles usuelles du langage C.

D'autres façons propres au langage Java seront vues dans les chapitres suivants.

Exemple 1 :

```
public class Expressions {
    public static void main(String[] args) {
        double a=10, b=20, max, min;

        max = b;
        min = a;
        if (a > b) {
            max = a;
            min = b;
        }
        //Equivalent a printf du langage C
        System.out.printf("max = %f\tmin =%f\n", max
            , min);

    }
}
```

Exemple 2 :

```
public class Expressions {
    public static void main(String[] args) {

        //Carrees des nombres impairs de 1 a 30
        for(int i=1; i<=30; i+=2)
            System.out.printf("%d^2 = %d\n",i,i*i);
    }
}
```

Exercices

Exercice 1

Parmi les identifiants suivants, quels sont ceux qui sont valides ?

a - nomEtudiant
b - prenom Etudiant
c - static
d - BONJOUR
e - 23code
f - code13
g - serie#
h - Test_Comp
i - goto

Exercice 2

Donnez le résultat des expressions suivantes :

a - 15 / 2 (7)
b - 15 % 2 (1)
c - 14 % 2 (0)
d - 31 % 7 (3)
e - 5 + 6 * 3 (23)
f - 25 + 3 / 2 (26)

Exercice 3

Donnez le résultat des expressions booléennes suivantes :

a - 4 <= 9 (true)
b - 12 >= 12 (true)
c - 3 + 4 == 8 (false)
d - 7 < 9 – 2 (false)
e - 5 != 5 (false)
f - 15 != 3 * 5 (true)
g - 9 != -9 (true)
h - 3 + 5 * 2 == 16 (false)

Exercice 4

Si j = 5 et k = 6, alors la valeur de j++ == k est :

a - 5
b - 6
c - true
d - false

Solution

false : j++ == k $\Leftrightarrow \begin{cases} j == k \\ j++ \end{cases}$

Exercice 5

Quel est le résultat de la sortie du code suivant ?

```
for(int i = 0; i < 3; ++i)
    for(int j = 0; j < 2; ++j)
        System.out.print(i + " " + j + " ");
```

a - 0 0 0 1 1 0 1 1 2 0 2 1

b - 0 1 0 2 0 3 1 1 1 2 1 3

c - 0 1 0 2 1 1 1 2

d - 0 0 0 1 0 2 1 0 1 1 1 2 2 0 2 1 2 2

Solution

0 0 0 1 1 0 1 1 2 0 2 1

Chapitre 2
Classes et objets

Introduction

Comme mentionné au chapitre précédent, Java est un langage (presque) purement orientée objets. Tout doit être à l'intérieure d'une classe.

Déclaration d'une classe

- Une classe est crée en utilisant le mot clés **class**.
- Elle peut contenir des méthodes (fonctions), des attributs (variables).

Pour illustrer ceci, nous allons créer le prototype de la classe **Etudiant** :

```
class Etudiant {
    //Declarations
    //attributs et methodes
}
```

Remarques :

1. on peut définir plusieurs classes dans un même fichier à condition qu'une seule classe soit précédée du mot clés **public** et que le fichier porte le même nom que la classe publique ;
2. une classe peut exister dans un fichier séparé (qui porte le même nom suivi de **.java**) ;
3. pour que la machine virtuelle puisse accéder à une classe contenant la méthode main, il faut que cette classe soit publique.

Définition des attributs

Nous supposons qu'un étudiant est caractérisé par son nom, son prénom, son cne et sa moyenne.

```
class Etudiant {
    private String nom;
    private String prenom;
    private String cne;
    private double moyenne
}
```

Par convention, les noms des attributs et des méthodes doivent être en minuscule. Le premier mot doit commencer en minuscule et les autres mots doivent commencer en majuscule (**ceciEstUneVariable**, **ceciEstUneMethode**).

Remarque

La présence du mot clés **private** (privé) indique que les variables ne seront pas accessibles de l'extérieure de la classe où elles sont définies. C'est possible de déclarer les variables non privé, mais c'est déconseillé.

Définition des méthodes

Dans la classe étudiant, nous allons définir trois méthodes :

- **initialiser()** : qui permet d'initialiser les informations concernant un étudiant.
- **afficher()** : qui permet d'afficher les informations concernant un étudiant.
- **getMoyenne** : qui permet de retourner la moyenne d'un étudiant.

Exemple

```
class Etudiant {
    private String nom, prenom, cne;
    private double moyenne;
    public void initialiser(String x, String y,
        String z, double m) {
         nom = x;
         ...
    }
    public void afficher() {
         System.out.println("Nom : "+nom);
         ...
    }
    public double getMoyenne() {
         return moyenne;
    }
}
```

Remarques

- On peut définir plusieurs méthodes à l'intérieure d'une classe.
- La présence du mot clés **public** (publique) indique que les méthodes sont accessibles de l'extérieure de la classe. C'est possible de déclarer des méthodes privés.
- Il existe d'autres modes d'accès aux variables et aux méthodes qu'on verra plus loin.

Utilisation des classes

Après déclaration d'une classe, elle peut être utilisée pour déclarer un objet (variable de type classe) à l'intérieure de n'importe quelle méthode. Pour utiliser la classe étudiant :

```
Etudiant et;
```

Contrairement aux types primitifs, la déclaration précédente ne réserve pas de place mémoire pour l'objet de type **Etudiant** mais seulement une référence à un objet de type **Etudiant**. Pour réserver de la mémoire, il faut utiliser le mot clés **new** de la façon suivante :

```
et = new Etudiant();
```

Utilisation des classes

Au lieu de deux instructions, vous pouvez utiliser une seule instruction :

```
Etudiant et = new Etudiant();
```

A présent, on peut appliquer n'importe quelle méthode à l'objet **et**. par exemple, pour initialiser les attributs de **et**, on procède de la façon suivante :

```
et.initialiser("Oujdi","Mohammed","A8899",12.5);
```

Dans l'exemple suivant, on va utiliser la classe **ExempleEtudiant**, pour tester la classe **Etudiant**.

Exemple :

```
public class ExempleEtudiant {
    public static void main(String[] args) {
         double moy;
        Etudiant et = new Etudiant();
        et.initialiser("Oujdi","Ali","A8899",12.5);
        et.afficher();
        et.initialiser("Berkani","Lina","A7788",13);
        moy = et.getMoyenne();
        System.out.println("Moyenne : "+moy);
    }
}
class Etudiant {
    ...
}
```

Exécution :

Le résultat de l'exécution du programme précédent est le suivant :

```
Nom : Oujdi
Prenom : Ali
CNE : A8899
Moyenne : 12.5
Moyenne : 13.0
```

Initialisation des objets

Lors de la création d'un objet, tous les attributs sont initialisés par défaut. Dans les sections suivantes on verra comment initialiser les attributs lors de la création d'un objet.

Type	boolean	char	byte	short	int	long
valeur par défaut	false	`'\u0000'`	(byte)0	(short)0	0	0L

Type	float	double	objet
valeur par défaut	0.0f	0.0	null

Portée des attributs

Les attributs sont accessibles à l'intérieure de toutes les méthodes de la classe. Il n'est pas nécessaire de les passer comme arguments.

A l'extérieure des classes, les attributs privés (**private**) ne sont pas accessibles. Pour l'exemple de la classe **Etudiant**, une instruction de type :

```
Etudiant et = new Etudiant();
moy = et.moyenne;
```

aboutit à une erreur de compilation (`The field Etudiant.moyenne is not visible`).

Surcharge des méthodes

On parle de surcharge, lorsque plusieurs méthodes possèdent le même nom. Ces différentes méthodes **ne doivent pas** avoir le même nombre d'arguments ou des arguments de **même** types. On parle de **signature** de la méthode.

Exemple :

On va ajouter à la classe **Etudiant** trois méthodes qui portent le même nom. Une méthode qui contient :

1. trois arguments de types **double** ;
2. deux arguments de types **double** ;
3. deux arguments de types **float**.

Exemple

```
class Etudiant {
    ...
    //calcul de la moyenne de trois nombres
    public double calculMoy(double m1, double m2,
        double m3) {
            ...
    }
    //calcul de la moyenne de deux nombres (doubles)
    public double calculMoy(double m1, double m2) {
            ...
    }
    //calcul de la moyenne de deux nombres (float)
    public double calculMoy(float m1, float m2) {
            ...
    }
}
```

Utilisation de la classe Etudiant

Dans l'exemple suivant, on va utiliser la classe **ExempleEtudiant**, pour tester la classe **Etudiant** modifiée.

Exemple

```
public class ExempleEtudiant {
    public static void main(String[] args) {
        double moy;
        Etudiant et = new Etudiant();
        et.initialiser("Oujdi","Ali","A8899",12.5);
        //Appel de calculMoy(double, double, double)
        moy = et.calculMoy(10.5,12,13.5);
        //Appel de calculMoy(double, double)
        moy = et.calculMoy(11.5,13);
        //Appel de calculMoy(float, float)
        moy = et.calculMoy(10.5f,12f);
        //Appel de calculMoy(double, double)
        //13.5 est de type double
        moy = et.calculMoy(11.5f,13.5);
    }
}
```

Conflit

Considérons la classe **Etudiant** qui contient deux méthodes. Chaque méthode contient :

1. deux arguments, un de type **double** et l'autre de type **float** ;
2. deux arguments, un de type **float** et l'autre de type **double**.

```
class Etudiant {
    ...
    public double calculMoy(double m1, float m2) {
        ...
    }

    public double calculMoy(float m1, double m2) {
        ...
    }
}
```

Exemple d'utilisation

```
public class ExempleEtudiant {
    public static void main(String[] args) {
        double moy;
        Etudiant et = new Etudiant();
        et.initialiser("Oujdi","Ali","A8899",12.5);

        //Appel de calculMoy(double m1, float m2)
        moy = et.calculMoy(11.5,13f);

        //Appel de calculMoy(float m1, double m2)
        moy = et.calculMoy(10.5f,12.0);
```

Exemple d'utilisation

```
        //11.5 et 13.5 sont de type double
        //Erreur de compilation
        moy = et.calculMoy(11.5,13.5);

        //11.5f et 13.5f sont de type float
        //Erreur de compilation
        moy = et.calculMoy(11.5f,13.5f);
  }
}
```

Exemple d'utilisation

L'exemple précédent conduit à des erreurs de compilation :

- `moy = et.calculMoy(11.5,13.5)` aboutit à l'erreur de compilation : `The method calculMoy(double, float) in the type Etudiant is not applicable for the arguments (double, double);`
- `moy = et.calculMoy(11.5f,13.5f)` aboutit à l'erreur de compilation : `The method calculMoy(double, float) is ambiguous for the type Etudiant.`

Lecture à partir du clavier

Pour lire à partir du clavier, il existe la classe **Scanner**. Pour utiliser cette classe, il faut la rendre visible au compilateur en l'important en ajoutant la ligne :

```
import java.util.Scanner;
```

nextShort()	permet de lire un **short**
nextByte()	permet de lire un **byte**
nextInt()	permet de lire un **int**
nextLong()	permet de lire un **long**. Il n'est pas nécessaire d'ajouter L après l'entier saisi.
nextFloat()	permet de lire un **float**. Il n'est pas nécessaire d'ajouter F après le réel saisi.
nextDouble()	permet de lire un **double**
nextLine()	permet de lire une ligne et la retourne comme un **String**
next()	permet de lire la donnée suivante comme **String**

Exemple :

```
import java.util.Scanner;
public class TestScanner
{
    public static void main(String[] args)
    {
        String nom;
        int age;
        double note1,note2,moyenne;

        /*clavier est un objet qui va permettre la
           saisie clavier
        vous pouvez utiliser un autre nom (keyb,
           input, ...)*/
        Scanner clavier = new Scanner(System.in);
```

Exemple :

```
            System.out.print("Saisir votre nom : ");
            nom = clavier.nextLine();
            System.out.print("Saisir votre age : ");
            age = clavier.nextInt();
            System.out.print("Saisir vos notes : ");
            note1 = clavier.nextDouble();
            note2 = clavier.nextDouble();
            moyenne = (note1+note2)/2;
            System.out.println("Votre nom est " + nom +
                ", vous avez " + age + " ans et vous avez
                 obtenu "+ moyenne);
            clavier.close(); //fermer le Scanner
        }
}
```

Problèmes liées à l'utilisation de **nextLine()**

Reprenons l'exemple précédent et au lieu de commencer par la saisie du nom, on commence par la saisie de l'age.

```
Scanner clavier = new Scanner(System.in);
System.out.print("Saisir votre age : ");
age = clavier.nextInt();
System.out.print("Saisir votre nom : ");
nom = clavier.nextLine();
System.out.print("Saisir vos notes : ");
note1 = clavier.nextDouble();
note2 = clavier.nextDouble();
moyenne = (note1+note2)/2;
System.out.println("Votre nom est " + nom +
    ", vous avez " + age + " ans et vous avez
     obtenu "+ moyenne);
```

L'exécution du précédent programme est la suivante :

```
 Saisir votre age : 23
Saisir votre nom : Saisir vos notes : 12
13
Votre nom est , vous avez 23 ans et vous avez obtenu
12.5
```

Lors de la saisie de l'age, on a validé par **Entrée**. La touche **Entrée** a été stocké dans **nom** !

Pour éviter ce problème, il faut mettre `clavier.nextLine()` avant nom = clavier.nextLine();.

```
public class TestScanner
{
    public static void main(String[] args)
    {
        ...
        age = clavier.nextInt();
        System.out.print("Saisir votre nom : ");
        clavier.nextLine();
        nom = clavier.nextLine();
        ...
    }
}
```

Chapitre 3
Constructeurs

Introduction

On a vu dans le chapitre 2, que pour initialiser les attributs de la classe **Etudiant**, on a définit une méthode **initialiser()**.

```
class Etudiant {
    private String nom, prenom, cne;
    private double moyenne;

    //Initialisation
    public void initialiser(String x, String y,
        String z, double m) {
         nom = x;
         prenom = y;
         cne = z;
         moyenne = m;
    }
    ...
}
```

Introduction

Cette façon de faire n'est pas conseillé pour les 2 raisons suivantes :

1. pour chaque objet créé, on doit l'initialisé en appelant la méthode d'initialisation ;
2. si on oublie d'initialiser l'objet, il sera initialisé par défaut, ce qui peut poser des problèmes lors de l'exécution du programme. Du fait que le programme sera compilé sans erreurs.

Pour remédier à ces inconvénients, on utilise les **constructeurs**.

Définition

Un **constructeur** est une méthode, sans type de retour, qui porte le même nom que la classe. Il est invoqué lors de la déclaration d'un objet.

Une classe peut avoir plusieurs constructeurs (**surcharge**), du moment que le nombre d'arguments et leurs types n'est pas le même.

Exemple

```
class Etudiant {
    private String nom, prenom, cne;
    private double moyenne;

    //Constructeur
    public Etudiant(String x, String y, String
        z, double m) {
        nom = x;
        prenom = y;
        cne = z;
        moyenne = m;
    }
    ...
}
```

Exemple

Pour créer un objet et l'initialiser, on remplace les deux instructions :

```
Etudiant et = new Etudiant();

et.initialiser("Oujdi","Ali","A8899",12.5);
```

par l'instruction :

```
Etudiant et = new Etudiant("Oujdi","Ali","A8899",12.5);
```

Utilisation de this

Dans les arguments du constructeur **Etudiant**, on a utilisé : x, y, z et m. On peut utiliser les mêmes noms que les attributs privés de la classe en faisant appel au mot clés **this**.

Exemple

```
class Etudiant {
    private String nom, prenom, cne;
    private double moyenne;

    //Constructeur
    public Etudiant(String nom, String prenom,
        String cne,
            double moyenne) {
        this.nom = nom;
        this.prenom = prenom;
        this.cne = cne;
        this.moyenne = moyenne;
    }
    ...
}
```

Surcharge des constructeurs

On va modifier la classe **Etudiant** pour qu'il possède deux constructeurs :

1. un à trois arguments ;
2. l'autre à quatre arguments.

Exemple 1

```
class Etudiant {
    private String nom, prenom, cne;
    private double moyenne;

    //Constructeur 1
    public Etudiant(String nom, String prenom,
        String cne) {
        this.nom = nom;
        this.prenom = prenom;
        this.cne = cne;
    }
```

Exemple 1

```
    //Constructeur 2
    public Etudiant(String nom, String prenom,
        String cne, double moyenne) {
        this.nom = nom;
        this.prenom = prenom;
        this.cne = cne;
        this.moyenne = moyenne;
    }
    ...
}
```

Exemple 1

```
Etudiant et = new Etudiant("Oujdi","Ali","A8899"
    ,12.5);

//l'attribut moyenne est initialisé par défaut (0.0)
Etudiant et1 = new Etudiant("Berkani","Lina","A7799"
    );
```

Exemple 2

Dans le constructeur 2 de l'exemple 1, trois instructions ont été répétées. Pour éviter cette répétition, on utilise l'instruction :

this (arguments)

L'exemple 1 devient :

Exemple 2

```
class Etudiant {
    ...
    //Constructeur 1
    public Etudiant(String nom, String prenom,
        String cne) {
         this.nom = nom;
         this.prenom = prenom;
         this.cne = cne;
    }
    //Constructeur 2
    public Etudiant(String nom, String prenom,
        String cne, double moyenne) {
         //Appel du constructeur 1
         this(nom, prenom, cne);
         this.moyenne = moyenne;
    }
```

Remarque

L'instruction

this (arguments)

doit être la première instruction du constructeur. Si elle est mise ailleurs, le compilateur génère une erreur.

Constructeur par défaut

Le constructeur par défaut est un constructeur qui n'a pas d'arguments.

Exemple

```
public class ExempleEtudiant {
    public static void main(String[] args) {
        //Utilisation du constructeur par défaut
        Etudiant et = new Etudiant();
    }
}
class Etudiant {
    ...
    //Constructeur par defaut
    public Etudiant() {
        nom = "";
        prenom = "";
        cne = "";
        moyenne = 0.0;
    }
    //Autres constructeurs
```

Remarques

1. Si aucun constructeur n'est utilisé, le compilateur initialise les attributs aux valeurs par défaut.
2. Dans les exemples 1 et 2 de la section «surcharge des constructeurs», l'instruction

```
Etudiant et = new Etudiant();
```

n'est pas permise parce que les deux constructeurs ont des arguments.

3. Un constructeur ne peut pas être appelé comme les autres méthodes. L'instruction

```
et.Etudiant("Oujdi","Mohammed","A8899");
```

n'est pas permise.

Constructeur de copie

Java offre un moyen de créer la copie d'une instance en utilisant le constructeur de copie. Ce constructeur permet d'initialiser une instance en copiant les attributs d'une autre instance du même type.

Exemple

```
public class ExempleEtudiant {
    Etudiant et1 = new Etudiant("Oujdi","Ali","A88");
    Etudiant et2 = new Etudiant(et1);
}
class Etudiant {
    ...
    //Constructeur de copie
    public Etudiant(Etudiant autreEt) {
        nom = autreEt.nom;
        prenom = autreEt.prenom;
        cne = autreEt.cne;
        moyenne = autreEt.moyenne;
    }
    ...
}
```

Remarque

et1 et **et2** sont différents mais ont les mêmes valeurs pour leurs attributs.

Chapitre 4
Généralités

Attributs statiques

Les variables statiques sont appelés « variables de classe ». Elles sont partagées par toutes les instances de la classe. Pour chaque instance de la classe, il n'y a qu'une seule copie d'une variable statique par classe. Il n'y a pas de création d'une nouvelle place mémoire lors de l'utilisation de « new ».

Pour déclarer une variable statique, il faut faut utiliser le mot clés static .

Exemple :

```
class Jeu {
    static int meilleurScore = 0;
    int score = 0;
    void calculScore() {
        score += 10;
        if (meilleurScore < score)    meilleurScore
            = score;
    }
}
```

Exemple :

```
public static void main(String[] args) {
  System.out.println(Jeu.meilleurScore);//Affiche 0
  Jeu.meilleurScore++;
  System.out.println(Jeu.meilleurScore);//Affiche 1
  Jeu j = new Jeu();
  j.calculScore();
  System.out.println(Jeu.meilleurScore);//Affiche 10
  //ou bien
  System.out.println(j.meilleurScore); //Affiche 10
  j.calculScore();
  System.out.println(Jeu.meilleurScore);//Affiche 20
 }
```

Constantes final et static

Une constante commune à toutes les instances d'une classe peut être déclarée en « final static ».

```
class A{
    final static double PI=3.1415927;
    static final  double Pi=3.1415927;
}
```

La classe « **Math** » fourni les constantes statiques **Math.PI** (égale à 3.141592653589793238462) et **Math.E** (égale à 2.718281828459045235360).

Constantes final et static

Dans la classe « **Math** », la déclaration de **PI** est comme suit :

```
public final static double PI = 3.14159265358979323846;
```

PI est :

- public, elle est accessible par tous ;
- final , elle ne peut pas être changée ;
- static , une seule copie existe et elle est accessible sans déclarer d'objet Math.

Méthodes statiques

Une méthode peut être déclaré statique en la faisant précédé du mot clés static. Elle peut être appelée directement sans créer d'objet pour cette méthode. Elle est appelée « méthode de classe ».

Il y a des restrictions sur l'utilisation des méthodes statiques :

Restrictions

- elles ne peuvent appeler que les méthodes statiques ;
- elles ne peuvent utiliser que les attributs statiques ;
- elles ne peuvent pas faire référence à this et à super.

Exemple :

```
class Calcul {
    private int somme;
    static int factorielle(int n) {
        //ne peut pas utiliser somme
        if (n <= 0)
            return 1;
        else
            return n * factorielle(n - 1);
    }
}
```

Exemple :

```
public class MethodesClasses {
    public static void main(String[] args) {
        Scanner clavier = new Scanner(System.in);
        int n;
        System.out.print("Saisir 1 entier :  ");
        n = clavier.nextInt();
        System.out.print("Factorielle :  " + n +
        " est  :" + Calcul.factorielle(n));
    }
}
```

Classe Math

La classe « **Math** » fourni les méthodes statiques **sin**, **cos**, **pow**, ...

Fin de vie des objets

Un objet (ou une variable) est en fin de vie lorsqu'il n'est plus utilisé. Il est hors porté.

Exemple 1 :

```
{
    int x=12; // x est accessible
    {
        int q;
        q=x+100; // x et q tous les deux sont
            accessibles
    }
    x=6;
    // x est accessible
    q=x+2; // Erreur: q est hors de portee
}
```

Attention

Ceci n'est pas permis en Java

```
{
    int x=12;
    {
        //illegale en Java (Duplicate local variable
            x)
        //valable en C, C++
        int x=96;
    }
}
```

Exemple 2 :

```
public class FinVie {
    public static void main(String[] args) {
        afficherUnEtudiant();
    }

    static void afficherUnEtudiant() {
        Etudiant et = new Etudiant("Oujdi", "Ali", "
            A20");
        System.out.println(et);
    }
}
```

La référence associé à **et** n'est plus utilisée par contre l'objet référencé par **et** existe toujours mais reste inaccessible.

Ramasse miettes (Garbage collector)

Contrairement au langage C où on la fonction **free** qui permet de libérer la mémoire occupée par un pointeur, en Java, il n'y a pas de méthode qui permet de libérer la mémoire occupée par un objet non référencé.

Par contre il existe un processus qui est lancé automatiquement (de façon régulière) de l'exécution d'un programme Java et récupère la mémoire non utilisé. Ce processus s'appelle le **ramasse miettes** (Garbage collector en anglais).

L'utilisateur peut appeler le ramasse miette en appelant la méthode System.gc();.

Exemple :

```
public class FinVie {
    public static void main(String[] args) {
        ...
        afficherUnEtudiant();
        System.gc();
        ...
    }

    static void afficherUnEtudiant() {
        Etudiant et = new Etudiant("Oujdi", "Ali", "
            A20");
        System.out.println(et);
    }
}
```