Programmation Orientée Objet

Initiation à Java

Saber HENI
saber.heni02@univ-paris8.fr
http://handiman.univ-paris8.fr/~saber/

Plan du cours

- Chapitre 1 : Introduction au langage Java
- Chapitre 2 : Classes et objets
- Chapitre 3 : Concepts de base de l'OO
- Chapitre 4 : La gestion des exceptions
- Chapitre 5 : Les classes de base dans Java
- Chapitre 6 : Les collections

UNIVERSITÉ PARIS 8
VINCENNES-SAINT-DENIS

Chapitre 2

Classes et objets

Les méthodes

- Le principal avantage des méthodes est de pouvoir factoriser le code.
- Java offre une panoplie de méthodes prédéfinies (natives) organisés dans des classes.
- Mais.
  - Vous pouvez aussi créer vos propre méthodes.

Méthodes natives - 1

- Exemple de méthodes concernant les chaînes de caractères.
  - La méthode toLowerCase() permet de transformer tout caractère alphabétique en son équivalent minuscule.

    public String toLowerCase () {…}
  - la méthode toUpperCase() est simple. Elle transforme donc une chaîne de caractères en capitales.

    public String toUpperCase () {…}
  - La méthode length() renvoie la longueur d'une chaîne de caractères (en comptant les espaces).

    public int length () {…}
  - La méthode equals() permet de vérifier (donc de tester) si deux chaînes de caractères sont identiques.

    public boolean equals (String s) {…}

Méthodes natives - 2

- Le résultat de la méthode charAt() sera un caractère : il s'agit d'une méthode d'extraction de caractère. Le premier caractère est d'indice 0 et elle prend en argument des entiers.

public char charAt (int pos) {…}

- La méthode substring() extrait une partie d'une chaîne de caractères.

public String substring (int debut) {…}

public String substring (int debut, int fin) {…}

```
String chaine = new String("Master MIASHS 2015");

String chaine2 = chaine.substring(7, 12);
//Permet d'extraire "MIASH"

String chaine3 = chaine.substring(7);
//Permet d'extraire « MIASHS 2015"
```

Méthodes natives - 3

- La méthode **indexOf()** explore une chaîne de caractères à la recherche d'une suite donnée de caractères.
  - renvoie la position (ou l'index) de la sous-chaîne passée en argument.
  - la méthode **indexOf()** explore à partir du début de la chaîne
- **lastIndexOf()** explore en partant de la fin, mais renvoie l'index à partir du début de la chaîne.

public int indexOf (String sousChaine) {…}

public int lastIndexOf (String sousChaine) {…}

Méthodes natives - 4

```
String mot = new String("anticonstitutionnellement");
int n = 0;

n = mot.indexOf('t');           //n vaut 2
n = mot.lastIndexOf('t');       //n vaut 24
n = mot.indexOf("ti");          //n vaut 2
n = mot.lastIndexOf("ti");      //n vaut 12
n = mot.indexOf('x');           //n vaut -1
```

Méthodes natives - 5

- Exemple de méthodes de la classe ***Math***.

```
double X = 0.0;
X = Math.random();
//Retourne un nombre aléatoire
//compris entre 0 et 1, comme 0.00013857463293710 58

double sin = Math.sin(120);     //La fonction sinus
double cos = Math.cos(120);     //La fonction cosinus
double tan = Math.tan(120);     //La fonction tangente
double abs = Math.abs(-120.25);
//La fonction valeur absolue (retourne le nombre sans le signe)
double d = 2;
double exp = Math.pow(d, 2);    //La fonction exposant
//Ici, on initialise la variable exp avec la valeur de d élevée
//au carré
//La méthode pow() prend donc une valeur en premier paramètre,
//et un exposant en second
```

Créer une méthode - 1

- Exemple de méthodes que vous pouvez créer.

```
public static double arrondi (double a, int b) {

        return (double) ( (int) (a * Math.pow(10,  b) + 0.5)) / Math.pow(10,
b);

}
```

  - Tout d'abord, il y a le mot clé **public**.
    - C'est ce qui définit la portée de la méthode
  - Ensuite, il y a **static**.
  - Juste après, nous voyons **double**.
    - Il s'agit du type de retour de la méthode. Pour faire simple, ici, notre méthode va renvoyer un double.
  - Vient ensuite le nom de la méthode **arrondi**.
    - C'est avec ce nom que nous l'appellerons.

Créer une méthode - 2

- Puis arrivent les arguments de la méthode **double a et int b**.
  - Ce sont en fait les paramètres dont la méthode a besoin pour travailler.
  - Ici, nous demandons d'arrondir le double **«a»** avec **«b»** chiffres derrière la virgule.
- Finalement, vous pouvez voir une instruction **return** à l'intérieur de la méthode.
  - C'est elle qui effectue le renvoi de la valeur, ici un double.

Si vous placez une de vos méthodes à l'intérieur de la méthode main ou à l'extérieur de votre classe, le programme ne compilera pas.

Créer une méthode - 3

- Les méthodes ne sont pas limitées en nombre de paramètres;
- Il en existe trois grands types :
  - les méthodes qui ne renvoient rien. Les méthodes de ce type n'ont pas d'instruction **return**, et elles sont de type **void** ;
  - les méthodes qui retournent des types primitifs (double, int etc.). Elles sont de type double, int, char etc.
    - Celles-ci possèdent une instruction **return** ;
  - les méthodes qui retournent des objets. Par exemple, une méthode qui retourne un objet de type String.
    - Celles-ci aussi comportent une instruction **return**.

Créer une méthode - 4

Exercice

Nous allons reprendre l'exercice de la moyenne olympique du TD 1. On vous demande de factoriser le code en le répartissant sur 5 méthodes :

- La première permet la conversion et le remplissage du tableau
- Une deuxième méthode permettant le tri du tableau.
- La troisième méthode qui fait juste la permutation de deux variables.
- Une quatrième méthode qui fait le calcul de la moyenne olympique.
- La dernière permet l'affichages des résultats.

Dans la méthode main vous faites seulement la déclaration du tableau et l'appel des quatre méthodes.

La surcharge des méthodes - 1

- En Java il possible de déclarer deux ou plusieurs méthodes avec le même nom à condition que :
  - Les types et/ou le nombre de paramètres soient différents.

- Est-il possible de déclarer deux méthode de même nom et de même nombre et types de paramètres mais de type de retour différent?
  - La réponse est non.
    - Le compilateur ne tient en compte, pour différencier quelle méthodes est appelée, que du type et du nombre d'arguments passés entre les parenthèses.

La surcharge des méthodes - 2

- **Exemple :**

```
public void draw(String s)  {
        ...
}

public void draw(int i) {
        ...
}

public void draw(double f) {
        ...
}

public void draw(int i, double f) {
        ...
}
```

Les objets en java - 1

- L'idée de base de la programmation orientée objet est de rassembler dans une même entité appelée objet les ***données*** et les ***traitements*** qui s'y appliquent.
  - ***Comment créer un objet?***
  - **Il faut définir la structure de l'objet.**

Les objets en java - 2

- Un objet ou **instance** est une variable *presque* comme les autres. Il faut notamment qu'il soit déclaré avec son type.
- Le type d'un objet est un type complexe (par opposition aux types primitifs entier, caractère, …) qu'on appelle une classe.

Les classes en java - 1

- Une classe regroupe :
  - Un ensemble de **données** dites **attributs** ou encore **variables d'instance** (qui peuvent être des variables primitives ou des objets)
  - Un ensemble de traitements de ces données et/ou de données extérieures à la classe, dites **méthodes**.
- Une classe sert *généralement* à créer des objets
  - On dit : Instancier la classe.

Les classes en java - 2

- Les objets contiennent des attributs (**variables d'instance**) et des méthodes.
- Les attributs sont des variables ou des objets nécessaires au fonctionnement de l'objet.
- La classe est la description d'un objet.
- Un objet est une instance d'une classe.
- Pour chaque instance d'une classe, le code est le même, **seules les données sont différentes à chaque objet**.

Les classe en java - 3

- **Exemple**

```
public class Rectangle {
        private int longueur ;
        private int largeur ;
        private int origineX ;
        private int origineY ;

        public void déplacer (int x, int y) {
                origineX += x ;
                origineY += y ;
        }

        public int surface () {
                return this.longueur * this.largeur ;
        }
}
```

Les modificateurs de visibilité des classes - 1

- Une classe peut être marqué
  - Public ou default.

– **public** : Une classe publique est visible dans tout l'univers de Java

– **default** : Par défaut, une classe est non-publique : elle n'est accessible que depuis les classes du même paquetage.
  - Si vous ne mettez aucun modificateur de visibilité devant la déclaration d'une classe, vous aurez une visibilité niveau paquetage.

Les paquetages

- Les classes Java sont regroupées en paquetages (packages en anglais)
- Ils correspondent aux « bibliothèques » des autres langages
- Les paquetages offrent un niveau de modularité supplémentaire pour :
  - réunir des classes suivant un centre d'intérêt commun
  - la protection des attributs et des méthodes.

La visibilité des membres d'une classe - 1

- Les modificateurs de visibilité agissent comme le montre le tableau suivant lorsqu'ils sont appliqué sur une classe ou l'un de ses membres.

Élément	Autorisations
Variable	Lecture et écriture
Méthode	Appel de la méthode
Classe	Instanciation d'objets de cette classe et accès aux variables et méthodes de classe

La visibilité des membres d'une classe - 2

- Ci-dessous un tableau récapitulant la porté (Visibilité) d'un membre d'une classe marqué à chaque fois par l'un des quatre modificateurs de visibilité :

	public	protected	défaut	private
Dans la même classe	Oui	Oui	Oui	Oui
Dans une classe du même package	Oui	Oui	Oui	Non
Dans une sous-classe d'un autre package	**Chapitre 3**			
Dans une classe quelconque d'un autre package	Oui	Non	Non	Non

La visibilité des membres d'une classe - 3

- **Encapsulation :** Tout ce qui participe à l'implémentation des services doit être privé (afin de permettre la modification de l'implémentation des services sans risquer d'impacter les autres classes).
  - Protéger les données
  - Ne permettre l'accès à ces donner qu'à travers les méthodes.

- En d'autres termes, toutes les variables d'instance (attributs) d'une classe doivent être déclarés ***private.***

Les constructeurs - 1

- Reprenons l'exemple du rectangle. Comment peut-on appeler la méthode surface?

```
1. Rectangle rectangle1;
2. rectangle1  = new Rectangle ();
3. int surface1 = rectangle1.surface();
4. System.out.println(surface1);
```

- Donc, Il faut instancier la classe Rectangle et puis appeler la méthode désiré.
- Mais, le résultat est toujours égale à 0.

Les constructeurs - 2

- **Commentaires**
  - Il est nécessaire de définir la déclaration d'une variable ayant le type de l'objet désiré. La déclaration est de la forme
    - **nom_de_classe** nom_de_variable;

```
Rectangle rectangle1;

String s;
```

  - L'opérateur **new** se charge de créer une instance de la classe et de l'associer à la variable.

```
rectangle1 = new Rectangle ();
```

Les constructeurs – 3

- L'opérateur **new** est un opérateur de haute priorité.
  - Permet d'instancier des objets et d'appeler une méthode particulière de cet objet (Le constructeur).
  - Il fait appel à la machine virtuelle pour obtenir l'espace mémoire nécessaire à la représentation de l'objet ;
  - puis appelle le constructeur pour initialiser l'objet dans l'emplacement obtenu.
  - Au final, il renvoie une valeur qui référence l'objet instancié.

Les constructeurs - 4

UNIVERSITÉ PARIS 8 VINCENNES-SAINT-DENIS

- **Questions**
  - Rectangle ()?
    - C'est une méthode particulière appelée **constructeur**.
  - Comment on a pu appeler un constructeur non déclaré dans la classe Rectangle?
    - Si vous ne déclarer aucun constructeur, la JVM fournit un constructeur par défaut (sans paramètres).
    - En revanche, dès que vous avez créé un constructeur, n'importe lequel, la JVM ne crée plus le constructeur par défaut.
  - Est-ce qu'on peut créer nos propres constructeurs?
    - Oui, et vous pouvez même les surcharger.

Les constructeurs – 5

- **Exemple**

```
public Rectangle (int long, int larg) {
      longueur = long;
      largeur = larg;
}

public Rectangle (int long, int larg, int x, int y) {
      longueur = long;
      largeur = larg;
      origineX = x;
      origineY = y;
}
```

Les constructeurs – 6

- **this et this ()**

```
public Rectangle (int longueur , int largeur ) {
        this.longueur = longueur;
        this.largeur = largeur ;
}

public Rectangle (int longueur , int largeur , int origineX, int origineY)
{
        this (longeur, largeur);
        this.origineX = origineX;
        this.origineY = origineY;
```

Les constructeurs – 7

this et this ()

- **this :** désigne l'objet courant.
  - this.largeur : est utilisé lorsqu'il y'a confusion entre le nom du paramètre et celui de la variable d'instance et désigne l'attribut largeur de l'objet courant.

- **this () :** est un appel interne (dans la même classe) d'un constructeur et entre le parenthèses on mets les arguments (le cas échéant).
  - this (longeur, largeur) : est un appel au constructeur qui prend en argument deux entiers.
  - Que faire pour appeler le deuxième constructeur à 4 entiers en arguments?

Les constructeurs – 8

- Un constructeur est une méthode invoquée lors de la création d'un objet.
- Cette méthode, qui peut être vide, effectue les opérations nécessaires à l'initialisation d'un objet.
- Chaque constructeur doit avoir le même nom (Première lettre en majuscule) que la classe où il est défini et n'a aucune valeur de retour (c'est l'objet créé qui est renvoyé).

Les constructeurs – 9

- **Remarque importante :**
  - En Java, la notion de pointeur est transparente pour le programmeur.
  - Il faut néanmoins savoir que toute variable désignant un objet est un pointeur.
  - Le passage de paramètres aux méthodes en java est toujours un passage par **valeur**.

Accès aux variables et aux méthodes

- Pour accéder à une variable associée à un objet, il faut préciser l'objet qui la contient.
- Le symbole '.' sert à séparer l'identificateur de l'objet de l'identificateur de la variable.
- Une copie de la longueur d'un rectangle dans un entier temp s'écrit :

```
int temp = rectangle1.longueur ;
```

- La même syntaxe est utilisée pour appeler une méthode d'un objet. Par exemple :

```
rectangle1.deplacer(10,-3);
```

Variables et méthodes de classe - 1

- Dans certains cas, il est plus judicieux d'attacher une variable ou une méthode à une classe plutôt qu'aux objets instanciant cette classe.
  - Par exemple, la classe java.lang.Integer possède une variable MAX_VALUE qui représente la plus grande valeur qui peut être affectée à un entier.
  - Or, cette variable étant commune à tous les entiers, elle n'est pas dupliquée dans tous les objets instanciant la classe Integer mais elle est associée directement à la classe Integer.
  - Une telle variable est appelée **variable de classe**.

Variables et méthodes de classe - 2

- De la même manière, il existe des **méthodes de classe** qui sont associées directement à une classe.
- Pour déclarer une variable ou méthode de classe, on utilise le mot-clé **static** qui doit être précisé avant le type de la variable ou le type de retour de la méthode.
- La classe **java.lang.Math** nous fournit un bon exemple de variable et de méthodes de classes.

```
Math.PI;
Math.random();
```

Variables et méthodes de classe - 3

- **Exemple**

```
class Compteur {
        private static int cptStatique = 0;
        private int cpt = 0;
        Compteur() {
                cpt++;
                cptStatique++
                System.out.println(cpt);
                System.out.println(cptStatique)
        }
}
```

```
new Compteur();
new Compteur();
new Compteur();
```

Accesseurs et mutateurs - 1

- Un accesseur ((en) getter) est une méthode qui va nous permettre d'accéder aux variables de nos objets en lecture.
  - Grâce aux accesseurs, vous pourrez afficher les variables de vos objets.
  - Les accesseurs ont toujours la forme

```
public typeDeLAttribut getNomDeLAttribut () {
    return l'attributAAccéder;
}
```

Accesseurs et mutateurs - 2

- Un mutateur ((en) setter) nous permettra d'en faire de même en écriture !
  - Grâce aux mutateurs, vous pourrez les modifier.
  - Les mutateurs ont eux aussi une forme particulière, qui est la suivante

```
public void setNomDeLAttribut (typeDeLAttribut identificateur) {
        attributAModifier = identificateur;
}
```

Remarque

Si l'attribut est statique alors l'accesseur ou le mutateur doit être statique aussi.

Si l'attribut est de type boolean alors le nom de l'accesseur devient isNomDeLAttribut.

Accesseurs et mutateurs - 3

Exemple

```
public class Ville {
        private String nomVille;
        private String nomPays;
        private int nbreHabitants;

//************** ACCESSEURS **************
        public String getNomVille() {
                return nomVille;
        }
        public String getNomPays() {
                return nomPays;
        }
        public int getNbreHabitants() {
                return nbreHabitants;
        }
```

Accesseurs et mutateurs - 4

Exemple (Suite)

```
//*************** MUTATEURS ***************
        public void setNomVille(String pNom) {
                nomVille = pNom;
        }
        public void setNomPays(String pPays) {
                nomPays = pPays;
        }
        public void setNbreHabitants(int nbre) {
                nbreHabitants = nbre;
        }

}
```

Accesseurs et mutateurs - 5

Exercice

1) ***La méthode comparer(Ville V1) :*** Elle prend une ville en paramètre, pour pouvoir comparer les variables *nbreHabitants* de l'objet appelant la méthode et de celui passé en paramètre pour nous dire quelle ville est la plus peuplée.

- **Implémentez cette méthode.**

```
public String comparer(Ville v1){}
```

Accesseurs et mutateurs - 6

2) La méthode toString() : Celle-ci nous renvoie un objet de type String. Elle fait référence aux variables qui composent l'objet appelant la méthode, toujours grâce à **this**, et nous renvoie donc une chaîne de caractères qui nous décrit l'objet en énumérant ses composants.

- **Implémentez cette méthode.**

```
public String toString (){}
```

Le ramasse miettes (Garbage Collector) - 1

- En Java, la destruction des instances est assurée par le Garbage collector.
- Lorsqu'une instance n'est plus accessible a partir des variables, elle peut être détruite automatiquement par GC.

- Le ramasse miettes est la technique de Java permettant la gestion automatique de la mémoire.
- Le rôle du GC est de s'assurer qu'il y aura toujours de l'espace libre dans le tas (heap) pour créer des nouveaux objets.

Ramasse miettes (Garbage Collector) - 2

- **Exercice 1 :** Combiens d'objet a été crée et combiens d'objet est éligible au GC quand la JVM atteint la ligne du commentaire.

```
class Lien { Lien l ; }
class ProgrammePrincipal {
        public static Lien creerCycle ( ) {
                Lien lien1 = new Lien ( ) ;
                Lien lien2 = new Lien ( ) ;
                Lien lien3 = new Lien ( ) ;
                lien1.l = lien2 ; lien2.l = lien3 ;  lien3.l = lien1 ;
                return lien1 ;
        }
        public static void main (String arg [ ]) {
                Lien lien = creerCycle ( ) ; lien = null ;
                 // ici
        }
}
```

Ramasse miettes (Garbage Collector) - 3

- **Exercice 1 :** Combiens d'objet a été crée et combiens d'objet est éligible au GC quand la JVM atteint la ligne du commentaire.

- **Correction :**
  - **Objets créés = 3**
  - **Objets éligible au GC = 3**

Ramasse miettes (Garbage Collector) - 4

- **Exercice 2 :** Même question.

```
class Autre { }
class Lien { Lien l ; Autre aut = new Autre (); }
class ProgrammePrincipal {
        public static Lien creerCycle ( ) {
                Lien lien1 = new Lien ( ) ;
                Lien lien2 = new Lien ( ) ;
                Lien lien3 = new Lien ( ) ;
                lien1.l = lien2 ; lien2.l = lien3 ; lien3.l = lien1 ;
                return lien1 ;
        }
        public static void main (String arg [ ]) {
                Lien lien = creerCycle ( ) ;
                 Autre a = lien.aut;  lien = null ;
                // ici
        }
}
```

Ramasse miettes (Garbage Collector) - 5

- **Exercice 3 :** Même question.

```
class Lien { Lien l ; static String s="Bonjour"; }

class ProgrammePrincipal {
        public static Lien creerCycle ( ) {
                Lien lien1 = new Lien ( ) ;
                Lien lien2 = new Lien ( ) ;
                Lien lien3 = new Lien ( ) ;
                lien1.l = lien2 ; lien2.l = lien3 ;  lien3.l = lien1 ;
                return lien1 ;
        }
        public static void main (String arg [ ]) {
                Lien lien = creerCycle ( ) ; lien = null ;
                 // ici
        }
}
```

UNIVERSITÉ PARIS 8
VINCENNES-SAINT-DENIS

See You Next Time!