Programmation Java

Y. ALJ

Plan

Références

Les références pour ce cours sont :

- Head First Java, 2nd Edition : Kathy Sierra, Bert Bates (pour les exercices).
- Le cours de Gauthier Picard
  `https://www.emse.fr/~picard/cours/1A/java/`

Introduction à Java

- Java est un langage de programmation de haut niveau.
- Conçu au début des années 1990 par Sun Microsystemes et actuellement maintenu par Oracle.
- Java est un langage indépendant de la plateforme utilisée :
- On écrit le programme une fois.
- Ce programme peut être exécuté sur différentes plateformes
- L'esprit de Java : "Write Once Run Anywhere"

Premier programme

- On commence comme à chaque quand on apprend un nouveau langage de programmation par afficher "Hello World".

```
public class MaPremiereClasse {
    public static void main(String[] args) {
    System.out.println("Hello World");
    }
}
```

Premier programme : discussion

- En Java chaque ligne de code qui peut être exécutée doit être à l'intérieur d'une classe.
- Dans note exemple, on a appelé cette classe MaPremiereClasse.

```
public class MaPremiereClasse {
    public static void main(String[] args) {
    System.out.println("Hello World");
    }
}
```

Premier programme : discussion

- En Java chaque application doit avoir un point d'entrée qui est une fonction (ou méthode) appelée main.

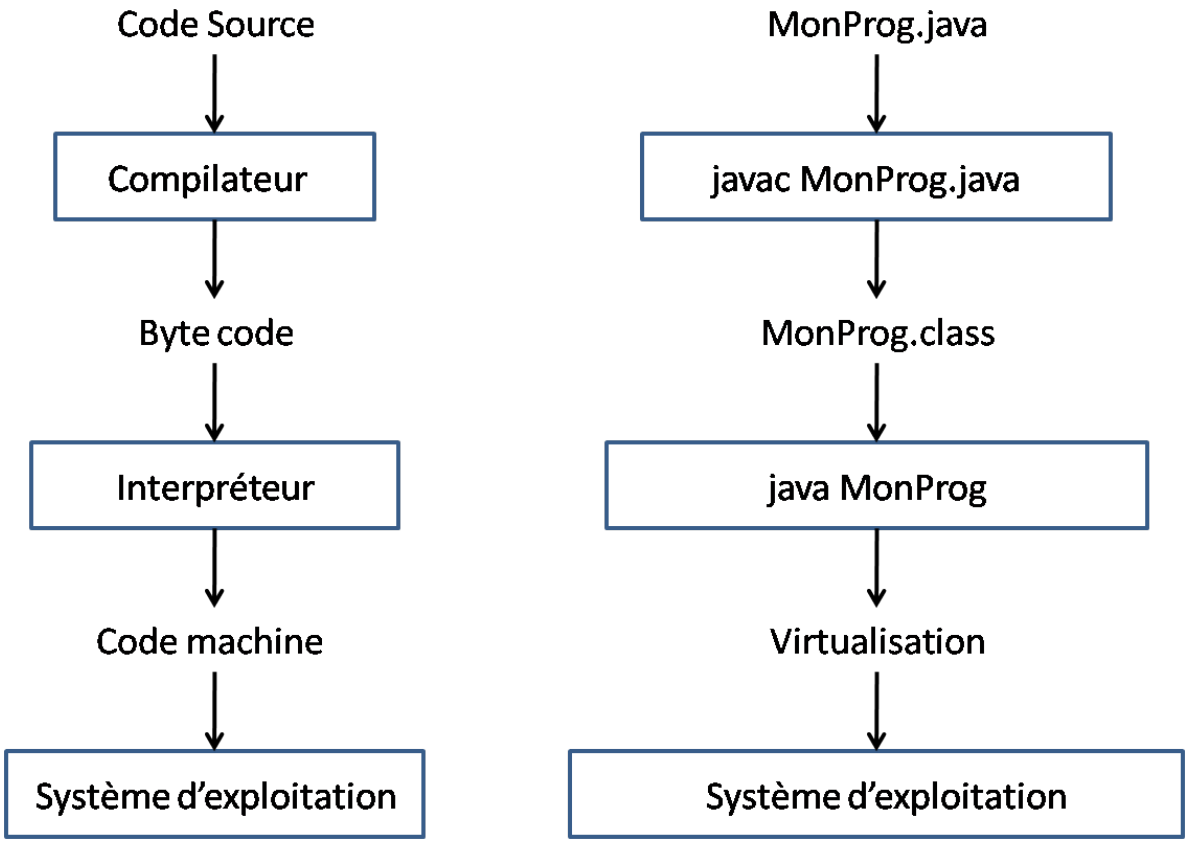
```
public class MaPremiereClasse {
    public static void main(String[] args) {
    System.out.println("Hello World");
    }
}
```

Premier programme : discussion

- Pour exécuter nos programmes notre fonction main doit obligatoirement avoir cette signature :

  public static void main(String [] args)
- public : tout le monde peut accéder à cette méthode.
- static : la méthode en question peut être appelée sans instancier la classe contenant la fonction main.
- void : méthode ne renvoie aucune valeur.
- main : nom de la méthode. Exemple : **void** test () ne renvoie rien et n'a pas de paramètres.

Processus de développement d'une application Java



Explication

1. Un programmeur Java écrit son code source : c'est un fichier ".java" qui contient une classe.
2. Ce code source est alors compilé par le compilateur javac et traduit en un autre langage appelé bytecode. Le résultat est un fichier ".class". Ce bytecode n'est pas directement utilisable.
3. Le bytecode est interprété par la machine virtuelle java qui transforme ce bytecode en code machine compréhensible par le système d'exploitation.

Installation de Java

- On a besoin :
  - Pour compiler : javac.
  - Pour exécuter l'application : java.
- Deux produits java existent :
  - le JDK (Java Development Kit) : un environnement qui contient les outils pour compiler et pour exécuter les applications java.
  - le JRE (Java Runtime Environment) : un environnement d'exécution de classes java.

Outline

Familles de langages de programmation

- Chaque langage de programmation appartient à une famille de langage qui définit une méthodologie pour programmer (appelée aussi paradigme de programmation).
- Par exemple : le langage C est un langage de programmation procédurale. Un programmeur C commence par identifier l'ensemble des traitements qu'il souhaite effectuer, puis écrit des fonctions qui réalisent ces traitements.
- La programmation orientée objet POO (en anglais Object Oriented Programming) propose une méthodologie orientée sur les données.
- En POO, le programmeur identifie les objets sous forme d'un ensemble de données dans un premier temps.
- Dans un second temps, le programmeur écrit les traitements.
- Un objet en POO est une entité regroupant des données (appelées attributs) et des traitements (appelés méthodes).

Comparaison entre C et Java

Langage C	Langage Java
programmation fonctionnelle	programmation orienté objet
langage de bas niveau	langage de haut niveau
gestion de la mémoire par l'utilisateur	gestion par la JVM
+ applications rapides (temps d'exécution)	- applications moins rapides
- éventuelle mauvaise gestion de la mémoire	+ moins de problèmes de gestion de la mémoire

La multiplication en Java

```
class Multiplication{

    public static void main(String [] args){
        int a = 2;
        int b = 4;
        System.out.println("Le produit de a=" + a + " par b=" +
            b +" est " + multiply(a,b));
    }
    public static int multiply(int a, int b){
        return a*b;
    }
}
```

La multiplication en C

```c
#include <stdio.h>

int multiply(int a, int b)
{
    return a*b;
}
void main()
{
    int a = 2;
    int b = 4;
    printf("La multiplcation de a=%i par b=%i est %i", a, b,
        multiply(a,b));
}
```

Les deux approches de programmation

Approche fonctionnelle

Que doit faire mon programme ?

Approche orientée objet

De quoi être composé mon programme ?

Syntaxe d'une classe

Le code source d'un programme Java est contenu dans un ou plusieurs fichiers d'extension ".java".

- Une seule classe publique par fichier.
- Le nom du fichier doit être le même que celui de la classe.
- Par convention, le nom d'une classe commence toujours par une majuscule.

Syntaxe d'une classe

```
class <nom de la classe>{
    <contenu de la classe>
}
```

Contenu d'une classe

Contenu d'une classe

- des attributs : variables typée.
- Dans l'exemple ci-contre les attributs sont : solde et proprietaire.
- des méthodes (ou opération) : ensemble d'instructions de traitement.
- Dans l'exemple ci-contre les méthodes sont : **double** getSolde() et **void** credite(**double** val).

Exemple

```
class CompteBancaire{
    String proprietaire;
    double solde;
    double getSolde(){
        return solde;
    }

    void credite(double val){
        solde = solde + val;
    }
}
```

Outline

Types de données primitifs

Les mêmes types qu'en C

- **short** : les entiers signés sur 16 bits.
- **int** : les entiers signés sur 32 bits.
- **long** : entiers signés sur 64 bits.
- **float** : réels sur 32 bits.
- **double** : réels sur 64 bits.
- **char** : charactères sur 16 bits.
- **void** : utilisé pour le type de retour de fonctions.

Plus deux nouveaux types

- **boolean** : prend les valeurs **true** ou **false**.
- **byte** : : les entiers signés sur 8 bits.

Plus une classe

- String

Exemple

```
boolean result = true;
char capitalC = 'C';
byte b = 100;
short s = 10000;
int i = 100000;
double d1 = 123.4;
// meme valeur que d1, mais avec la notation scientifique
double d2 = 1.234e2;
float f1  = 123.4f;
```

Portée d'une variable

```
class Portee{
    int a;
    public void test(int b){
        if(a>b){
            int c = b;
            b = a;
            a = c;
        }
        System.out.println(b);
    }
}
```

L'entier *a* est visible dans toute la classe Portee.

Portée d'une variable

```
class Portee{
    int a;
    public void test(int b){
        if(a>b){
            int c = b;
            b = a;
            a = c;
        }
        System.out.println(b);
    }
}
```

L'entier *b* est visible à l'intérieur de la méthode test.

Portée d'une variable

```
class Portee{
    int a;
    public void test(int b){
        if(a>b){
            int c = b;
            b = a;
            a = c;
        }
        System.out.println(b);
    }
}
```

L'entier *c* est visible à l'intérieur du bloc `if`.

Outline

Accès aux attributs d'une classe

Accès aux attributs

- Les attributs sont des données de la classe.
- Ils sont définis à l'intérieur d'une classe.
- L'accès aux attributs se fait en créant un objet de la classe `Myclass` et en utilisant le point appliqué à l'objet créé.

Exemple

```
public class Myclass {
 // ceci est un attribut
  int x = 5;
  public static void main(String[] args) {
  // instanciation de myObj a partir
  // de la classe MyClass
    MyClass myObj = new MyClass();
    // acces a l attribut x et modification de sa
          valeur
    myObj.x = 40;
    // affichage de la nouvelle valeur de l
          attribut
    System.out.println(myObj.x);
  }
}
```

Outline

Les méthodes en Java

- Les méthodes Java sont définies à l'intérieur des classes.
- Les méthodes définissent des traitements à faire.

Exemple : apple de méthode

```java
public class MyClass {
  // ma methode
  public void myMethod() {
    System.out.println("appel de methode");
  }
  // methode Main
  public static void main(String[] args) {
    // cree an objet de MyClass
    MyClass myObj = new MyClass();
    // Call the public method on the object
    myObj.myMethod();
  }
}
```

Les méthodes `static` en Java

- `static` désigne une méthode qui n'appartient pas à une instance particulière de la classe.
- Les méthodes `static` appartiennent à la classe elle-même.

Exemple : Définition

```
public class MyClass {
  // Static method
  public static void myStaticMethod() {
    System.out.println("Static methods can be
        called without creating objects");
  }

  // non static method
  public void myPublicMethod() {
    System.out.println("Public methods must be
        called by creating objects");
  }

  // Main method
  public static void main(String[] args) {
    myStaticMethod(); // Call the static method
    // myPublicMethod(); This would compile an
        error

    MyClass myObj = new MyClass(); // Create an
        object of MyClass
    myObj.myPublicMethod(); // Call the public
        method on the object
  }
}
```

Outline

Constructeurs

- Un constructeur Java est une **méthode spéciale** qui est utilisée pour construire/initialiser des objets issus de la classe.

Propriétés

Un constructeur :

- doit avoir le **même nom** que la classe.
- **n'a pas de valeur de retour**.
- n'est pas obligatoire.
- Plusieurs constructeurs peuvent exister dans la même classe mais avec des arguments différents.

Exemple constructeur

```
// Creer une classe appelee MyClass
public class MyClass {
  //on cree un attribut de la classe
  int x;
  // on cree un constructeur de la classe
  // il va initialiser l attribut x a 5
  public MyClass() {
  // on initialise l attribue de cette
  // classe a 5
    x = 5;
  }
  public static void main(String[] args) {
  // on cree un objet de la classe MyClass
  // ceci va appeler le constructeur cree
    MyClass myObj = new MyClass();
    //On affiche la valeur de x
    System.out.println(myObj.x);
  }
}
```

Outline

Accesseurs ("getters")

- Accesseur (ou "getter") : une méthode qui permet de récupérer le contenu d'un attribut de la classe.

```
class MaClasse{
    private TypeDeMaVariable maVariable;
    public  TypeDeMaVariable getMavariable(){
        // je renvoie maVariable
        return maValeur;
    }
}
```

Mutateurs ("setters")

- Mutateur (ou "setter") : une méthode qui permet de modifier le contenu d'un attribut de la classe.

```
class MaClasse{
    private TypeDeMaVariable maVariable;
    public  void setMaVariable(TypeDeMaVariable maValeur){
        // je mets a jour maVariable
        maVariable = maValeur;
    }
}
```

Intérêt des accesseurs/mutateurs

1. Contrôler la validité des valeurs qu'on affecte aux membres.
   - Si une valeur ne convient pas on peut la rejeter dans la définition du mutateur (voir exemple ci-dessous).
   - Variable peuvent être "read-only" si on utilise les accesseurs ou "write-only" si on utilise les mutateurs.

Exemple

```
class Personne{
    private int age;
    public void setAge(int nouvelAge){
        if (nouvelAge > 200){
            System.out.println("error");
        }
        else{
            age = nouvelAge;
        }
    }
}
```

Outline

Encapsulation

C'est le processus de regrouper données (attributs) et traitement (méthodes) dans la même unité.
Ce procédé stipule que les données d'une classes doivent être cachées des autres classes.
On accède aux données via les accesseurs/mutateurs. En deux phrases :

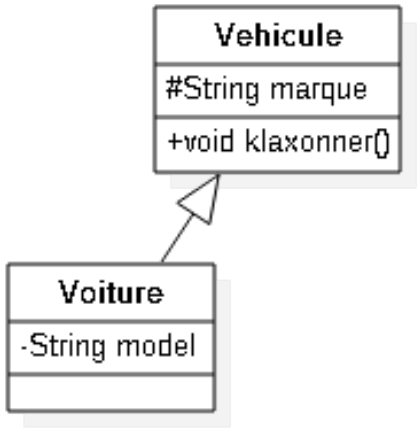
- Déclarer les données (attributs) d'une classe comme `private`.
- Fournir les accesseurs/mutateurs comme `public` pour accéder aux attributs privés.

Outline

Héritage (inheritence)

Héritage

- Concept fondamental de la POO.
- Consiste à créer une classe à partir d'une classe existante.
- La nouvelle classe ainsi créée est appelée sous-classe (ou classe fille).
- La classe existante est appelée super classe ou classe mère.




```
class Vehicule{
    // corps de la classe
        Vehicule
    ...
}
class Voiture extends Vehicule{
    // corps de la classe
        Voiture
    ...
}
```

Héritage (inheritence)

Héritage

- La classe fille hérite des attributs + méthodes de la classe mère
- Dans l'exemple : la classe voiture hérite de l'attribut `marque` et de la méthode `klaxonner`.
- La classe fille pourra avoir de nouveaux attributs/méthodes qui lui sont spécifiques.
- Dans l'exemple : la classe voiture a un attribut spécifique (`modele`).

Pourquoi l'héritage ?

- Pour la ré-utilisabilité du code.

Exemple héritage :

```
class Vehicule{
    /* protected pour pouvoir acceder a cet
          attribut depuis la classe fille */
    protected String marque  = "Renault";
    public void klaxonner() {
        System.out.println("beep beep");
    }
}
public class Voiture extends Vehicule{
    private String modele = "Clio";
    public static void main(String [] args) {
        Voiture v = new Voiture();
        System.out.println("Je suis une " + v.
              marque + " " + v.modele);
        v.klaxonner();
    }
}
```

Empêcher l'héritage : `final`

Si on veut empêcher l'héritage d'une classe, on utilise le mot clé `final`.

Exemple

```
final class Vehicule {
   ...
}

//la ligne suivante provoque une erreur de compilation
//error: cannot inherit from final Vehicule
class Voiture extends Vehicule {
   ...
}
```

Outline

Polymorphisme

- Polymorphisme = Poly + formes.
- Un objet peut avoir plusieurs formes.
- Rappel : l'héritage permet à une classe fille d'hériter des attributs et des méthodes à partir d'une classe mère (et d'avoir de nouveaux attributs/méthodes pour les classes filles).
- Le polymorphisme permet aux méthodes héritées d'avoir un comportement spécifique à la classe fille.

```
class Animal{
    void cri() {
        System.out.println("cri d'un animal");
    }
}

class Chat extends Animal{
    void cri() {
        System.out.println("miou miou");
    }
}

class Chien extends Animal{
    void cri() {
        System.out.println("how how");
    }
}

public class TestPolymorphisme {
    public static void main(String[] args) {
        Animal a = new Animal();
        Animal b = new Chien();
        Animal c = new Chat();
        a.cri();
        b.cri();
        c.cri();
    }
}
```

Outline

Package

Définition

Un package est une collection de classes rangées dans le même répertoire.

Pourquoi les packages ?

1. Structuration des projets : un package est associé à un dossier.
2. Conflits de nom : on peut définir deux classes qui portent le même nom. Seule la notion de package permet de différencier les deux d'un point de vue compilation.

On distingue deux types de packages :

1. Packages Java.
2. Packages définis par l'utilisateur.

Les Packages Java

- Ce sont des packages de classes déjà définies.
- On peut les utiliser librement.
- La liste des packages se trouve ici :
  `https://docs.oracle.com/javase/8/docs/api/`

Exemple

Pour utiliser un package, on écrit

```
import package.nom.Class;    // Importer une seule classe
import package.nom.*;    // Importer toutes les classes
```

Exemple : on veut lire ce que l'utilisateur tape au clavier.
On va utiliser la classe `Scanner`. Sa documentation est décrite ici :
`https://docs.oracle.com/javase/8/docs/api/index.html`

1. Quel est le package qu'il faut utiliser.
2. Ecrire une classe TestScanner qui permet de lire un entier le stocke dans une variable `age` et lui ensuite une chaine de caractères et le stocke dans une variable `nom`.
3. Créer une classe `Personne` avec un constructeur qui prend en entrée ces deux variables et qui initialise ces attributs.
4. Créer une nouvelle classe `TestPersonne` qui permet de tester cette classe en demandant à l'utilisateur de saisir le nom et l'âge de la personne au clavier.

Packages Java

Les classes du langage Java sont organisées en packages :

- `java.lang` : Les classes de base et la gestion des processus (importé par défaut)
- `java.util` : Des structures de données pratiques.
- `java.io` : La gestion des flux comme l'accès aux fichiers
- `java.math` : Calcul sur des nombres.
- `java.awt` : Composants graphiques : boutons, fenêtres, etc
- `javax.swing` : Une Alternative aux composants graphiques AWT.
- `java.applet` : Tout pour faire des applets.
- `java.net` : Permet des connections réseau IP.
- `java.sql` : Utilisation de bases de données relationnelles en SQL
- ...

Outline

Accessibilité

L'accessibilité est la possibilité de restreindre l'accès à certaines données (attributs, méthodes ou classes).

Mode	Signification
`public`	cet élément est accessible de partout sans restriction. Certaines classes (la classe qui contient le main) doivent être publiques pour exécuter l'application
`protected`	cet élément est accessible uniquement aux classes d'un package et à ses classes filles.
`private`	cet élément est accessible uniquement à l'intérieur de la classe où il est délcaré
Par défaut (ou package)	accessible uniquement aux classes d'un package

Outline

`try ...catch`

- Quand on exécute un code java, plusieurs erreurs à l'exécution peuvent se produire.
- Quand cela se produit, Java arrête l'exécution du programme et génère un message d'erreur.

Pour que java lance une exception (une erreur), on utilise les expressions `try` et `catch`

1. L'expression `try` permet de tester un bloc de code.
2. L'expression `catch` permet de définir un bloc de code à faire au cas où il y a une erreur dans le bloc `try`.

Exemple

Exemple sans gestion d'erreur

```
public class MyClass {
  public static void main(String
      [ ] args) {
    int[] myNumbers = {1, 2, 3};
    System.out.println(myNumbers
        [10]); // error!
  }
}
```

Exemple gestion d'erreur avec les exceptions

```
public class MyClass {
  public static void main(String
      [ ] args) {
    try {
      int[] myNumbers = {1, 2,
          3};
      System.out.println(
          myNumbers[10]);
    } catch (Exception e) {
      System.out.println("
          Something went wrong.
          ");
    }
  }
}
```