Eléments de programmation java

Christelle CAILLOUET
(christelle.caillouet@unice.fr)

Le langage Java

- Les variables, opérateurs, expressions, instructions, blocs, contrôle de flot sont très proches de ceux du C.

- S'y ajoutent les exceptions, et des spécificités syntaxiques liées à la programmation objet, aux classes, à l'héritage, …

- Un **style de nommage** précis :
  - ***CamelCase*** pour les identificateurs
  - Première lettre en majuscule pour les classes
  - Première lettre en minuscule pour les variables et méthodes (*abscisse, getAbscisse()*)
  - Tout en majuscules pour les constantes (*TAILLE_MAX)*

Premiers pas

- Un fichier de nom `HelloWorld.java`

- **Règles :**
  - **toute classe publique doit être dans un fichier qui a le même nom que la classe**
  - **tout code doit être à l'intérieur d'une classe**

HelloWorld.java

```
public class HelloWorld {

    /**
     * Method main
     *
     *
     * @param args
     *
     */
    public static void main(String[] args) {
        // TODO: Add your code here
        System.out.println("Hello world !");
    }
}
```

- Ceci définit une classe.
- Comme il y a une méthode `main`, cette classe est *exécutable*.

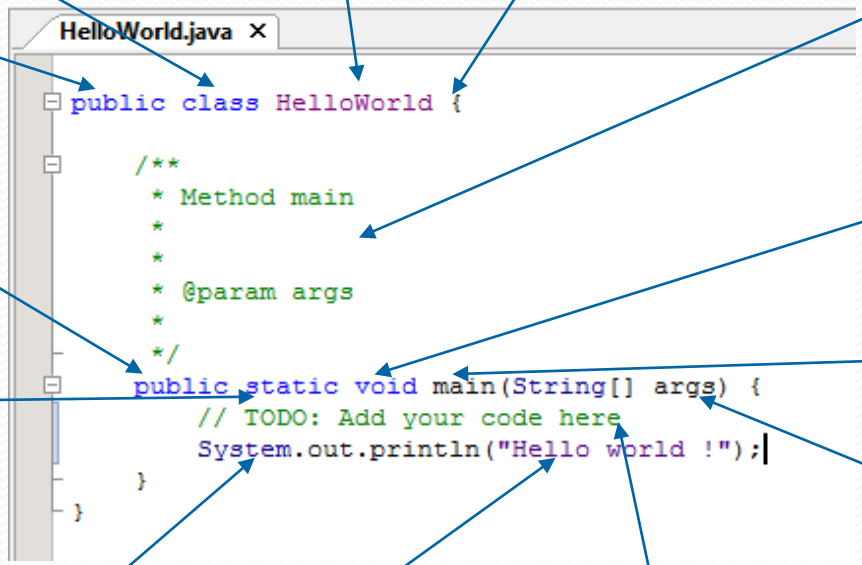
Tout code en java doit appartenir à une classe

Nom de la classe (identique au nom du fichier .java)

Accolade qui ouvre le contenu de la classe

Commentaires sur plusieurs lignes (avec balises de documentation)

Tout le monde peut exécuter ce programme



Tout le monde peut utiliser cette méthode

Cette méthode ne retourne rien

Cette méthode appartient à la classe, pas à un objet

Nom de la méthode

Nom de la méthode que nous voulons appeler (ici la méthode qui affiche des chaines de caractères à l'écran et retourne à la ligne contenue dans le package `java.lang.System`)

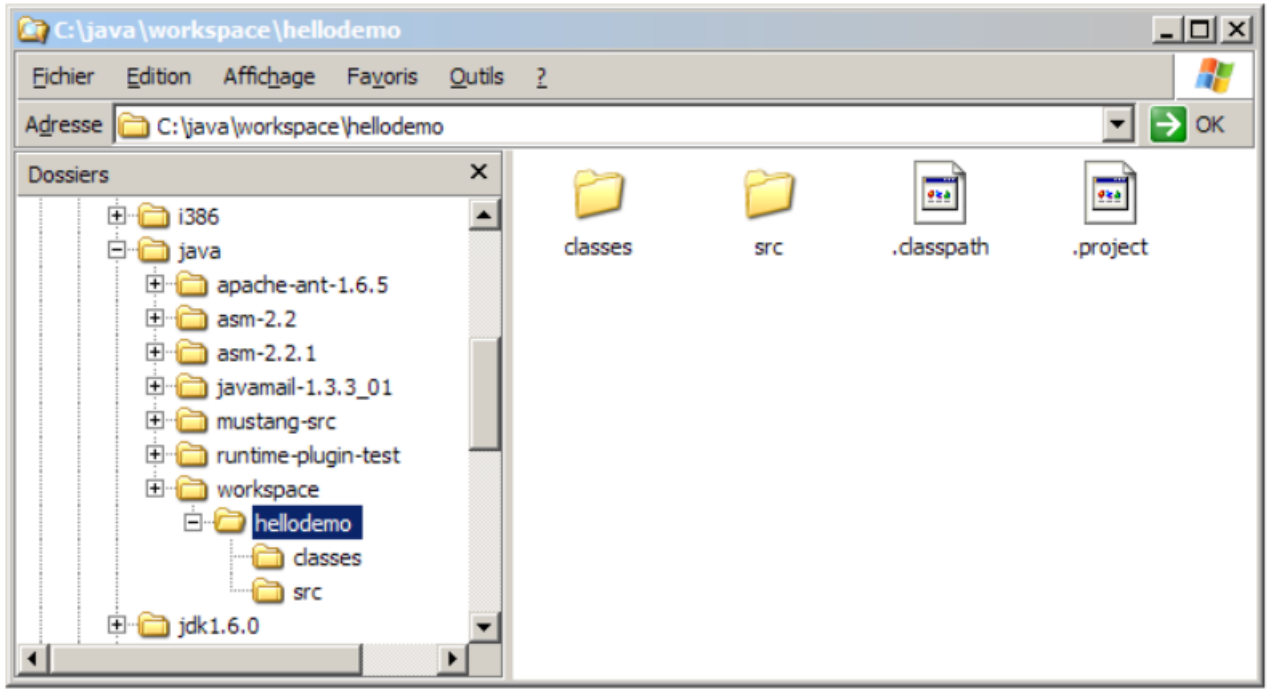
Paramètre de la méthode appelée (ici chaine de caractères à afficher)

Une ligne de commentaires

Paramètres de la méthode (ici un tableau de chaines de caractères correspondant aux arguments de la ligne de commande)

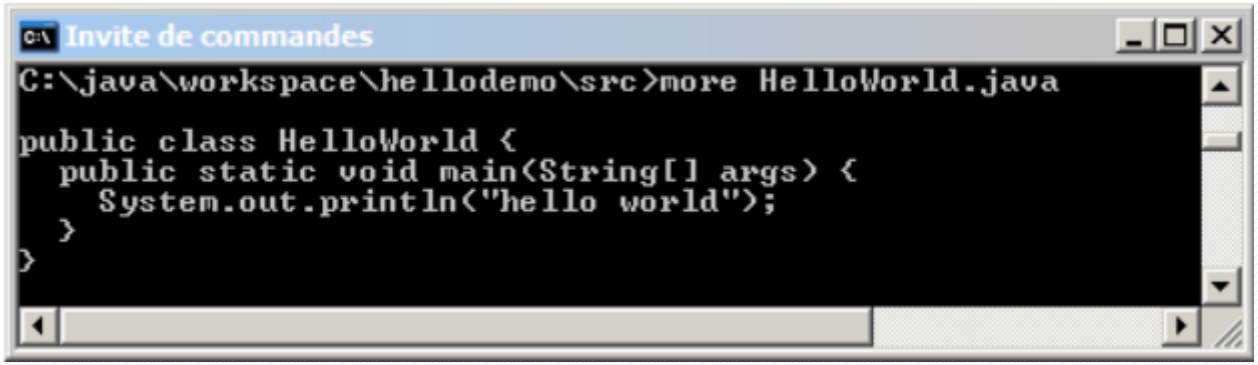
Fichiers sources et bytecode

- Habituellement, on sépare les fichiers sources (dans **src**) des fichiers binaires (dans **classes** ou **bin**)

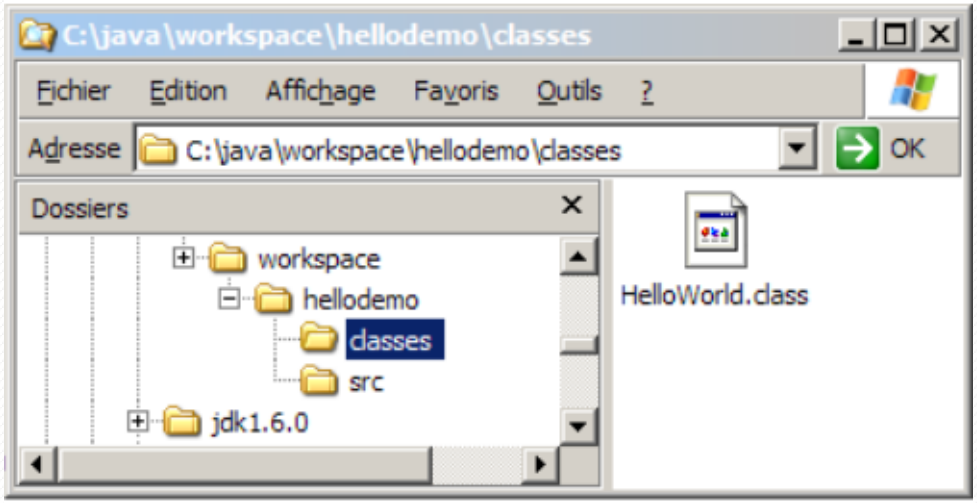


Compilation simple

- Dans **src** :



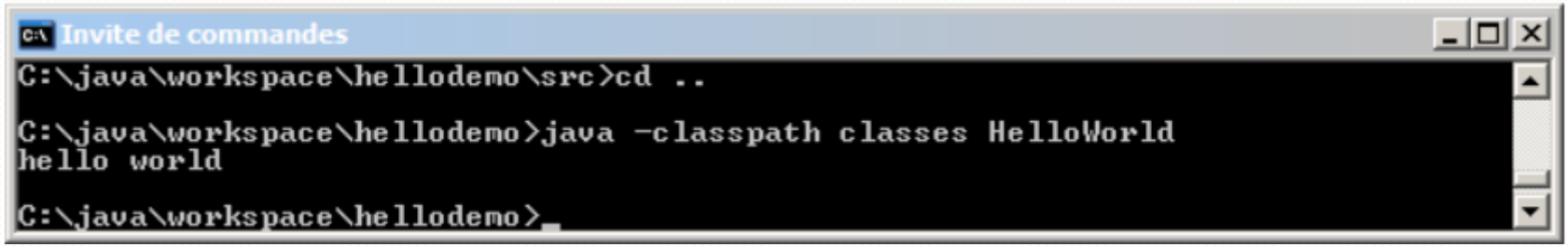
- Compiler avec la commande :
  `javac -d ../classes HelloWorld.java`



Crée le fichier **HelloWorld.class** dans **classes**

Exécution simple

- On remonte d'un répertoire



- Exécuter avec la commande :
  `java -classpath classes HelloWorld`

- On indique où se trouvent les fichiers bytecode (avec **-classpath**) ainsi que le nom de la classe qui contient le **main**

Notions de classes et objets

Définitions

- Un programme met en œuvre différents *objets*
- Une *classe* = description commune d'un ensemble d'*objets* ayant une structure de données commune et disposant des mêmes méthodes.
- Un *objet* est une **instance** de la *classe*

➔ Une application est donc une **partition de classes**. (diagramme de classes en UML)

Forte analogie avec les T.A.D.

Module M112-M113 / Langage C

- Organisation hétérogène interne des données (**struct**)
- Accessibilité implicite externe à tous les champs
- Allocation dynamique des instances
- Traitements décrits par fonctions publiques (**fonctions**, **procédures**)
- Nécessité de prévoir les moyens de destruction en mémoire (**free**)

Module M213 / Langage Java

- Le mot clé **class** remplace *struct*, **attributs** remplace *champs*
- Introduction de spécificateurs de portée (**public, protected, private**)
- Pas d'accessibilité externe aux champs (**private**)
- Méthodes de création d'instances (**constructeurs**)
- Allocation dynamique des instances (**objets**)
- Consultation/modification des attributs par **accesseurs**
- Encapsulation des fonctions (**méthodes**)
- Gestion mémoire par ***garbage collector***

Exemple du T.A.D. Etudiant

Module M112-M113 / Langage C

```
typedef struct {
        /* chaines de caractères */
        char nom[10];
        char prenom[10];
        int age;
        float note;
} etudiant;
```

Module M213 / Langage Java

```
public class Etudiant{
        private String nom;
        private String prenom;
        private int age;
        private double note;
}
```

Classe et objet Etudiant

- La classe **`Etudiant`** représente plusieurs choses :
  - Une *unité de compilation* : la compilation d'un fichier contenant la classe `Etudiant` produira un fichier *Etudiant.class*
  - La *définition du type Etudiant* : il est ensuite possible de créer des variables `Etudiant etud`
  - Un *moule* pour la création d'objets de type Etudiant : définition de l'ensemble des attributs décrivant l'état de l'objet, et de l'ensemble des méthodes décrivant son comportement

➔ Chaque objet de la classe **`Etudiant`** :
  - Dispose de **son propre état** (la valeur de ses attributs)
  - Répond au **même comportement** (via les méthodes de classe)

```java
public class Livre{
    private String titre;
    private Lecteur emprunteur;

    public Livre(String t, Lecteur l){
        titre = t;
        emprunteur = l;
    }

    public Livre(String t){
        titre = t;
        lecteur = null;
    }

    public void setTitre(String t) {
        titre = t;
    }

    public String getTitre() {
        return titre;
    }

    public Date emprunte(Lecteur lec) {
        if (emprunteur == null) {
            emprunteur = lec;
            return new Date();
        }
        else return null;
    }

    public String toString(){
        if (emprunteur == null)
            return "Livre "+titre+" non emprunte.";
        else
            return "Livre "+titre+"emprunte par "+emprunteur.toString();
    }
}
```

Attributs

Constructeurs

Mutateur (*Setter*)

Accesseur (*Getter*)

Méthodes

Instanciation

- Mot-clé *new* pour instancier un nouvel objet de la classe :

```
Livre monLivre = new Livre(”Germinal”);
```

- Invocation d’un constructeur existant.

- Sans instanciation, on ne peut pas utiliser les données et méthodes de la classe ! (sauf cas *static*…)

Invocation de méthode

- En Java, une méthode ne peut pas être invoquée seule, elle est toujours appelée **sur un objet** (ou une classe, pour les méthodes *static*)

- Un point « . » sépare le nom de la méthode de l'objet sur lequel elle est invoquée :

```
String titreDuLivre = monLivre.getTitre();
```

- Le mot-clé ***this*** désigne, en cours d'exécution d'une méthode, l'objet (ou la classe) sur lequel elle est appelée :

```
public Livre(String t, Lecteur l){
    titre = t;
    emprunteur = l;
}

public Livre(String titre, Lecteur emprunteur){
    this.titre = titre;
    this.emprunteur = emprunteur;
}
```

Invocation de méthode

- Passage des paramètres :
  - Les attributs de l'objet de la classe sont accessibles directement dans les méthodes de la classe

    ➔ on dit que les données sont encapsulées dans la classe où est définie la méthode

```
public String toString(){
    if (emprunteur == null)
        return "Livre "+titre+" non emprunte.";
    else
        return "Livre "+titre+"emprunte par "+emprunteur.toString();
}
```

Emprunteur et titre sont les attributs de l'objet Livre, ils peuvent être utilisés directement dans les méthodes de la classe Livre

Encapsulation

➔ Regroupement des données dans une classe et accessibilité de celles-ci

- Permet de :
  - imposer des règles et limitation de visualisation ou de manipulation de l'objet en cachant ou non l'existence des données et méthodes aux autres objets à l'aide des mots-clés : *public, protected, private*
  - centraliser les contrôles sur l'état de l'objet

Modificateur du membre	`private`	*défaut*	`protected`	`public`
Accès depuis la classe	Oui	Oui	Oui	Oui
Accès depuis une classe du même package	Non	Oui	Oui	Oui
Accès depuis une sous-classe	Non	Non	Oui	Oui
Accès depuis tout autre classe	Non	Non	Non	Oui

Constantes

- Le mot-clé `final` signifie en Java : ***affectation unique***

- L'affectation se fait :
  - Dès la déclaration
    ```
    private final int x = 100;
    ```
  - Au plus tard dans le constructeur.

- Le compilateur vérifie que la variable :
  1. A bien été initialisée (et ce quelque soit le constructeur),
  2. N'a été affectée qu'une seule fois.

- En fait, on verra plus tard que `final` peut s'appliquer sur des attributs, des méthodes, et même des classes…

Variables et méthodes de classe

- **Attachées à une classe plutôt qu'à un objet**
  - Existe même si aucun objet n'est instancié
  - Déclarées avec le préfixe `static`
  - Appel à partir du nom de la classe

- Variable :
  - valeur propre à la classe
  - Même valeur pour tous les objets de la classe

- Méthode :
  - Peut être exécutée même si aucun objet n'existe

```java
public final class Math{
    ...
    public static final double PI = 3.14159265358979323846;
    ...
    public static double toRadians(double angdeg){
        return angdeg / 180.0 * PI;
    }
    ...
}


public class MathMain{
    public static void main(String[] args){
        System.out.println("pi = "+Math.PI);
        System.out.println("90° = "+Math.toRadians(90));
    }
}
```

Cas particulier de la méthode *main*

- Méthode d'entrée de l'exécution du programme java
- Peut être située dans une classe définissant un objet (par exemple *Livre.java*)
- Généralement se trouve dans une classe à part (i.e. *TestLivre.java*)

- Signature formelle de la méthode unique et imposée :

```
public static void main(String[] args)
```

Comment écrire ce constructeur ?

```
public class Pixel {
    public final static int ORIGIN = 0;
    private int x;
    private int y;

    public Pixel(int x, int y) {
        // A compléter !!
    }
}
```

- ?

Comment écrire ce constructeur ?

```
public class Pixel {
    public final static int ORIGIN = 0;
    private int x;
    private int y;

    public Pixel(int x, int y) {
        // A compléter !!
    }
}
```

- **Réponse :**

```
this.x = x;
this.y = y;
```

Variables et méthodes de classe (*static*)

- Nous utilisons depuis le début une variable de classe et une méthode de classe. Pouvez-vous trouver lesquelles ?

HelloWorld.java

```
public class HelloWorld {

    /**
     * Method main
     *
     *
     * @param args
     *
     */
    public static void main(String[] args) {
        // TODO: Add your code here
        System.out.println("Hello world !");
    }
}
```

Variables et méthodes de classe (*static*)

- Nous utilisons depuis le début une variable de classe et une méthode de classe. Pouvez-vous trouver lesquelles ?

- **Réponses :**
  - Méthode `main` : appelée directement à partir d'une classe
  - Dans l'instruction `System.out.println(…)`, nous faisons appel à la variable `out` de la classe `java.lang.System`
    - C'est un objet représentant la sortie standard
    - Sur cet objet, on appelle la méthode `println` permettant d'afficher une chaîne de caractères