Présentation de l'UE

Introduction, programme *main*, types, variables séquences d'instructions simples

Faculté des sciences, Université de Montpellier

UE HAI717I - Programmation par objets

Objectifs du module

- Comprendre et appliquer les principes de base de la Programmation Orientée Objets (POO)
- Leur combinaison avec d'autres styles de programmation (impératif, fonctionnel)
- Langage à typage statique : Java
- Langage graphique pour certains schémas :UML
- Environnement de développement **eclipse** et **replit.com**

Organisation

Agenda, documents

- https ://moodle.umontpellier.fr/mod/page/view.php ?id=577531

Enseignants 2022-2023

- Marianne Huchard (Cours et TDs)
- Mathieu Lafourcade (TDs)

Modalités de contrôle des connaissances

- Session 1 :
  - Contrôle Continu : Écrit devoir surveillé, 2 ou 3 courts devoirs, (30% de la note)
  - Contrôle terminal : Écrit devoir surveillé, 2h (70% de la note)
- Session 2 :
  - Contrôle Continu : Report de la note de CC (30% de la note)
  - Contrôle terminal : Écrit devoir surveillé, 2h (70% de la note)

Contenu

Profil

- Débutants possédant des notions d'algorithmique impérative de base
- Module mutualisé : M1 ICo, Physique-numérique, Géomatique, Bioinformatique

Notions abordées

- Eléments de programmation impérative
- Classes, instances, attributs, méthodes
- Mise en œuvre des associations 1-1, n-m, énumérations
- Utilisation de classes génériques (listes)
- Données particulières : constantes, données de niveau instance vs. de niveau classe
- Héritage, spécialisation, polymorphisme, liaison statique et dynamique
- Récursivité et son mariage avec l'héritage

Pourquoi étudier et utiliser la POO ?

- un mode de pensée et de conception qui s'est étendu à de multiples domaines (BD, logiciel, prog. système, prog. IHM, ...)
- en se combinant avec d'autres paradigmes : par ex. impératif, fonctionnel, événementiel, logique, distribué
- des parentés avec des approches de représentation des connaissances (en particulier les ontologies)
- par sa capacité à monter en abstraction (la programmation s'est nourrie au travers du temps de différentes montées en abstraction)
- comme base de construction et inspiration pour de nouveaux paradigmes plus récents, comme les composants, les services et les micro-services

Avantages attendus en termes de construction de logiciel

- stabilité des artefacts construits (données "abstraites", plus stables que les fonctions)
- compréhensibilité des programmes (représentation des entités d'un domaine)
- structuration modulaire favorisant la maintenabilité
- extensibilité des programmes
- économie de développement par la réutilisation

Objectifs du premier cours

- organisation des programmes en Java
- principes de traduction des algorithmes simples composés de :
  - variables, types
  - instructions simples
  - séquences d'instructions
- début de l'approche par objets à partir du 2e cours

Structure générale d'un programme Java

```
/*
* Voici un programme simple qui affiche "Bonjour"
*/
package Cours1NotesExpress;
import java.util.Scanner;

public class cours1 { // debut de la classe

  public static void main(String[] args) { // debut du main
            System.out.println("Bonjour");
   } // fin du main

} // fin de la classe
```

Structure générale d'un programme Java

```
package Cours1NotesExpress;
```

- Le fichier se place dans un répertoire dont le nom figure derrière la directive "package".
- Vous pouvez assimiler dans un premier temps : répertoire = package.
- Les répertoires peuvent contenir des sous-répertoires, sans limitation de niveau d'imbrication.

Structure générale d'un programme Java

/*Voici un programme simple qui affiche "Bonjour"*/

//Voici un programme simple qui affiche "Bonjour"

- Dans un texte de programme, on peut placer des commentaires :
  - ceux entre / * et * / peuvent faire plusieurs lignes
  - s'ils ne font qu'une ligne, on peut les introduire par //
  - les outils comme javadoc auront des commentaires particuliers

```
/**
 * Classe exemple pour javadoc
 * @author marianne
 * @version 1.0
 */
public class cours1 { ......

// commentaire sur une ligne

/* commentaire sur
   plusieurs lignes
*/

}
```

Imports

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

- Des ressources extérieures pourront être nécessaires
  - Dans ce cas on les importe grâce à une directive import
  - Importe ici une classe Scanner (représentation du flux d'entrée)
  - La classe Scanner est rangée dans le répertoire (package) "java" et son sous-répertoire "util" d'où l'écriture java.util

Classe et fichier

public class cours1 { . . . . . }

- Le fichier porte le nom d'une structure de haut-niveau appelée une classe
- Ici le fichier s'appelle cours1.java, et la classe s'appelle cours1.
- Les accolades { .. } sont des éléments de structuration et peuvent se lire
  - { début
  - } fin
- `public` indique que la classe est visible depuis d'autres programmes y compris dans d'autres packages

Programme Main

```
public class cours1 {

      public static void main(String[] args) {
          System.out.println("Bonjour"); // affiche Bonjour
      }

}
```

- une méthode est un bloc d'instruction nommé. Ici nous observons la méthode `main` (méthode principale) qui est à l'intérieur de la classe `Cours1`
- `public` indique que la méthode `main` est visible depuis d'autres programmes y compris dans d'autres packages
- `static` sera la marque de certaines méthodes, dites "statiques"
- `void` indique qu'il s'agit d'une procédure (rien n'est retourné)

Programme Main

```
public class cours1 {

      public static void main(String[] args) {
          ....
      }

}
```

- `main` est le nom de la méthode (de l'algorithme principal)
- entre les parenthèses se trouvent les paramètres, il n'y en a qu'un ici, il est de type "tableau de chaînes (suites) de caractères" et s'appelle "args", il sert à récupérer des informations qui seraient données lors du lancement du programme (nous n'utiliserons pas cette possibilité au début)
- c'est à la première instruction que commencera l'exécution du programme quand on le lancera.

Programme Main et instruction

```
public class cours1 {

      public static void main(String[] args) {
          System.out.println("Bonjour"); // affiche Bonjour
      }

}
```

- Dans cet espace on peut mettre les instructions, par exemple, nous affichons sur la console la suite de caractères "Bonjour"
- `System.out` est le nom d'un objet qui représente la console
- `println` est un algorithme qui écrit sur la console
- `("Bonjour")` est sa liste d'arguments

Types

- Comme en algorithmique, les programmes Java manipulent des valeurs qui appartiennent à des types.
- De manière simplifiée un type est en informatique :
  - un ensemble de valeurs (appelées aussi littéraux, valeurs littérales)
  - un ensemble d'opérations admises sur ces valeurs
- En Java, on trouvera des types :
  - primitifs, simples
  - construits : tableaux, énumérations, classes

Type Entier

- Entier : int
- Ecriture des valeurs littérales : 2 3 12 -2 -3 -12
- Principales opérations + - * / % < <= > >= == !=
- Quelques expressions (affichées grâce à `System.out.println`)
  - System.out.println(2) ; // affiche 2
  - System.out.println(-2) ; // affiche -2
  - System.out.println(2+4) ; // affiche 6
  - System.out.println(2 == 4) ; // affiche false (voir ci-dessous !)

Type booléen

- Booléen : boolean
- Ecriture des valeurs littérales : `true` `false`
- Principales opérations ! && & | || == !=
- Utilisez plutôt && (et logique) et || (ou logique) car elles n'évaluent que ce qu'il faut pour conclure
- Quelques expressions (affichées grâce à `System.out.println`)
  - System.out.println(true) ; // affiche true
  - System.out.println(!true) ; // affiche false
  - System.out.println(false && true) ; // affiche false
  - System.out.println(false || true) ; // affiche true
  - System.out.println(2 >=4 && 2 >= 1) ; // affiche false
  - System.out.println(2 >=1 && 2 >= 4) ; // affiche false

Type réel

Il en existe plusieurs, ici nous considérons les réels en double précision

- Réel : double
- Ecriture des valeurs littérales : 2 -2 2.01 2.01E-3 -2.01E+3
- Principales opérations + - * / < <= > >= == !=
- Quelques expressions
  - System.out.println(2.01) ; // affiche 2.01
  - System.out.println(2.01E-3) ; // affiche 0.00201
  - System.out.println(-2.01E+3) ; // affiche -2010.0

Type Caractère

- Caractère : char
- Ecriture des valeurs littérales : 'a' 'z' '\t' '\n'
- Principales opérations + < <= > >= == !=
- Quelques expressions :
  - System.out.println('a') ; // affiche a
  - System.out.println('2') ; // affiche 2
  - System.out.println(""+'a'+'2') ; // affiche a2
  - System.out.println('a'+'2') ; // affiche 147 (code ascii de 'a' + code ascii de '2')
  - System.out.println('a'<'2') ; // affiche false (rang dans les codes ascii)
  - System.out.println('A'<'a') ; // affiche true (rang dans les codes ascii)

Le type construit "chaîne de caractères"

- Chaîne de caractères : String
- Ecriture des valeurs littérales : "abricot" "orange" "Ligne1 \n Ligne2"
- Principales opérations + equals compareTo charAt
- Quelques expressions, pour certaines la syntaxe sera clarifiée lorsque nous développerons la programmation par objets en Java
  - System.out.println("abricot") ; // affiche abricot
  - System.out.println("abricot"+"ier") ; // affiche abricotier
  - System.out.println("abricot"+'s') ; // affiche abricots
  - System.out.println("Ligne1 \n Ligne2") ; // affiche
    Ligne1
    Ligne2
  - System.out.println("abricot".compareTo("Abricot")>0) ; // affiche true
  - System.out.println("abricot".compareTo("orange")>0) ; // affiche false
  - System.out.println("abricot".equals("abricot")) ; // affiche true

Le type construit "chaîne de caractères"

Deux points d'attention

- ne pas comparer les chaînes avec l'opérateur d'égalité ==
  utiliser equals et compareTo
  - == compare les localisations en mémoire des chaînes (ce qui nous intéresse rarement)
  - `equals` compare les contenus (ce qui nous intéresse)

- lorsqu'une chaîne est attendue, par exemple comme paramètre de System.out.println, elle est transformée en chaîne

  Ex. System.out.println("L'article "+ 5 + "est vendu") ;

  Les chaînes et l'entier 5 sont concaténées

Déclarer une variable

- < Type > <nomVariable> ;
- Comprenez pour le moment une variable comme une petite boîte dont le nom est le nom de la variable
- Son format (sa dimension) est proportionné au format du type de la variable. Quelques déclarations de variables :
  - int i ;
  - double d ;
  - String s ;

Affecter une valeur à une variable

- on utilise le symbole =
- attention aux étourderies consistant à le confondre avec la comparaison ==
  - Avec `int i;` i est une petite boîte dans laquelle on met l'entier 4
    `i = 4;`
  - on ne peut pas mettre une valeur booléenne dans i
  - Avec `double d;` d est une petite boîte dans laquelle on met le réel 3.7E+1
    `d = 3.7E+1;`
  - Avec `String s;` s est une petite boîte dans laquelle on met le mot "fraises"
    `s = "fraises";`

On peut combiner ....

```
i = i+2;
```

- utilise le contenu de i, qui est 4, lui ajoute 2 et remet le tout dans i
- i vaut 6 après cette instruction

Affichage d'une expression

- Sans passer à la ligne

  ```
  System.out.print(s+" à la chantilly - ") ;
  ```

- En passant à la ligne (notez **ln** après `print`)

  ```
  System.out.println("charlotte aux "+s) ;
  ```

Saisie d'une valeur

Elle va nous demander plus d'efforts et l'ajout d'un nouvel objet représentant le "clavier" ou plus exactement le flux d'entrée du programme.
On se rappelle que l'on a importé cette ressource tout en haut de notre programme par `import java.util.Scanner;`

```
Scanner clavier = new Scanner(System.in);

// on veut saisir une valeur et la mettre
// dans la boîte (variable) i :

i = clavier.nextInt();

// il existe une variété d'opérations de type "next"
// suivant le type de la variable que l'on veut saisir

// quand on a fini, on "ferme" la connexion avec le clavier :

clavier.close();
```

Appeler une autre méthode

Dans cette partie, nous appelons une autre méthode, qui a été réalisée en-dehors du main. Ce procédé, que vous avez vu en algorithmique, consistant à séparer en plusieurs algorithmes les calculs complexes, permet de structurer le programme et de réutiliser du code. Pour cela, les méthodes doivent réaliser des algorithmes bien délimités.

```
System.out.println("Entrer un montant à payer");
double montantAPayer = clavier.nextDouble();

System.out.println("Entrer un montant de taxe entre 0 et 100");
double montantTaxe = clavier.nextDouble();

System.out.println("Montant incluant la taxe = "
                + calculMontantTTC(montantAPayer,montantTaxe));
```

Réaliser une autre méthode

Et maintenant nous définissons cette méthode

```
/* METHODE calculMontantTTC
*
* Méthode (sous-programme) qui va retourner,
* pour un montant hors taxe (HT) à payer
* et un montant de taxe (en pourcentage entre 0 et 100),
* le montant total à payer (TTC)
* Entrée : un nombre réel représentant un montant HT (mHT)
*          un nombre réel représentant une taxe (tax)
* Résultat : le montant incluant la taxe
* Méthode : calculer la formule mHT * (1 + tax/100) et la retourner
*
*/

public static double calculMontantTTC(double mHT, double tax) {
    return mHT * (1 + tax/100);
}
```

Pour travailler chez vous

- Installer eclipse sur son propre ordinateur
  https ://moodle.umontpellier.fr/mod/page/view.php ?id=131074

- Alternativement, utiliser https ://replit.com/
  - codage en ligne par le navigateur, pas d'installation
  - un environnement moins riche, un peu lent, mais facile à prendre en main
  - créez un compte personnel
  - création d'un dépôt avec New repl (en Java) et remplir le `main`
  - le dépôt peut être public ou privé et il est partageable sur invitation pour travailler à plusieurs
  - vous pouvez exécuter en appuyant sur triangle vert (Run), l'exécution apparaîtra dans la fenêtre à fond noir à droite
  - Le code de ce cours est dans le dépôt :
    `https://replit.com/@mariannehuchard/2022HAI717POO#Main.java`
    vous pouvez le copier chez vous en choisissant Fork puis le modifier ensuite