

Développement en JAVA

Notions de base



Agenda

- Notions fondamentales
- L'objet system
- Variables et opérateurs
- Les structures conditionnels
- Conversion de type
- Les tableaux
- Les boucles
- Les énumérations

Exercice 1: premier pas

Hello World

Une classe publique Main

```
1 public class Main {  
2  
3 public static void main(String args[]){  
4     System.out.println("Hello World");  
5 }  
6 }
```

Une méthode statique, ayant pour paramètre un tableau de chaîne de caractères

Une instruction d'affichage sur la sortie standard

Commandes de base

- Le JDK fournit plusieurs commandes:
 - `java <MonFichier>`
permet d'exécuter du code Java **compilé**
 - `javac <MonFichier>.java`
permet de compiler du code **source** java en un fichier .class



La JVM n'est capable d'exécuter que du code compilé !

Notion très fondamentale

Tout programme possède:

- Des **variables** dans lesquelles sont stockées les données
- Des **instructions** qui décrivent comment les manipuler
- Des **directives** qui permettent de préparer le code avant la compilation ou l'exécution

Notions fondamentales en Java

En Java tout est **objet** (ou presque !)

C'est un langage **compilé** à typage **explicite statique fort**

Le code **compilé** est exécuté par une **machine virtuelle**

La mémoire est gérée **automatiquement**



Notions fondamentales en Java

Toute instruction se termine **par un point virgule**

Le terme fonction n'existe pas, on parle de **méthode**

Point d'entrée

- Une application peut avoir plusieurs point d'entrée:
 - l'initialisation d'un processus
 - une socket
 - une interface utilisateur ...
- Dans le cas d'un programme console c'est:
 - une **classe** possédant
 - une **méthode statique**
 - ayant pour nom **main**
 - et prenant en paramètre un **tableau de chaîne de caractère**
 - qui représente les paramètres passés au programme

Exemple

```
public class Main{  
  
    public static void main(String[] args){  
        // Affichage de l'ensemble des paramètres reçu par le programme  
        for(int i=0; i < args.length; i++){  
            System.out.println(args[i]);  
        }  
    }  
}
```

Résultat de l'exécution:

```
> Java Main 1 2 3  
1  
2  
3
```

L'objet system

- La classe `System` est instanciée en même temps que chaque programme
- Son instance est accessible depuis n'importe quel scope du programme
- Elle expose des attributs :
 - La sortie standard
`static PrintStream out`
 - La sortie erreur
`static PrintStream err`
 - L'entrée standard
`static InputStream in`
- Et des méthodes
 - Arrêt de la JVM exécutant actuellement ce programme
`static exit(int status)`
 - Lecture des variables d'environnement
`static Map<String, String> getenv()`
 - Copie d'un tableau par bloc
`void arraycopy`
`(Object src, int srcPos, Object dest, int destPos, int length)`
- Pour aller plus loin:

Lire depuis l'entrée standard

- La classe Scanner permet de lire depuis l'entrée standard
- Elle est définie dans le package `java.util` et doit être importée à l'aide de la directive `import` en début de fichier

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

- Elle offre plusieurs méthodes:

```
String nextLine() // lit la prochaine ligne
```

```
int nextInt() // lit la prochaine ligne et la parse comme un entier
```

- Voir la documentation complète sur le site d'Oracle:

<http://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html>

Lire depuis l'entrée standard

Exemple

```
import java.util.Scanner;

public class Main{

    public static void main(String[] args){

        Scanner sc = new Scanner(System.in);
        System.out.println("Quel est votre age ?");
        int value = sc.nextInt();

        if(value < 25){
            System.out.println("C'est jeune !");
        }else{
            System.out.println("C'est vieux !");
        }

        sc.close();
    }
}
```

Notion de bloc

- Un bloc est délimité par des accolades
- Il permet de définir le contexte d'une méthode, d'une classe, d'un mot clé...
- Toute instruction est définie au sein d'un bloc

Variables

- Les variables permettent de stocker une valeur de type primitif ou une référence vers un objet
- Un nom de variable doit être unique à l'intérieur d'un bloc

```
<Type de la variable> <Nom de la variable> ;
```

```
int i = 0;  
String s = new String("Hello World");  
User user = new User();
```

Variables

- Dans le cas d'une méthode, la portée et la visibilité d'une variable est réduite au bloque où elle est définie

```
if(test){  
    int i = 0;  
    // i peut être utilisé  
    i = 2;  
}  
System.out.println(i);  
// Erreur de compilation: i est non défini
```


Types primitif

- Les types primitif sont une exception dans le langage Java puisque ce ne sont pas des objets
- Ils commencent toujours par une minuscule

Principaux types primitifs	
int	entier
small	entier court
long	entier long
float	nombre décimal
double	nombre décimal à double précision
boolean	booléen
char	caractère
byte	octet

Déclaration des type

- Un entier long doit être déclaré de la manière suivante:

```
long compteur = 960000000000L;
```

- Un nombre flottant se déclare en notant la précision

```
float prix = 12.95f;  
double longueur = 5.1335156456465d;
```

- Un booléen se déclare avec les mots clés false et true

```
boolean faux = false;  
boolean vrai = true;
```

Caractères et chaines de caractères

- Un caractère unique est toujours représenté entre simple guillemet
- Une chaine de caractère est toujours représentée entre double guillemet

```
char c1 = 'a'; // OK
char c2 = "b"; // KO
String s1 = "alphabet"; // OK
String s = 'e'; // KO
```

- La notation double guillemet est un alias permettant d'instancier la classe String

```
// Ces deux lignes sont équivalentes
String s1 = "éléphant";
String s2 = new String("éléphant");
```

Concaténation

- La concaténation entre deux chaînes de caractères s'effectue à l'aide de l'opérateur +

```
String nom = "Zinedine";  
String prenom = "Zidane";  
String nomComplet = prenom + " " + nom;
```

Opérateurs arithmétiques

- L'ensemble des « opérateurs classiques » sont supportées

Opérateurs binaire	+, -, *, /, %
Opérateurs unaire	++, --, +=, -=
Comparaison	==, !=, <, <=, >, >=
Combinaison	&&,

- On ne peut faire des traitements arithmétiques que sur des variables de même type
- Contrairement au C++ il est n'est pas possible de surcharger les opérateurs

Conditions

- Les conditions sont matérialisées en Java par le mot clé `if(condition)`
- Le mot clé `else if(condition)` permet d'enchaîner les branchements conditionnels
- Le mot clé `else` permet d'offrir un comportement par défaut

Conditions

Exemple

```
if(a == 42){
    System.out.println("C'est la bonne réponse");
}else if( a % 666 == 0){
    System.out.println("C'est diabolique");
}else if( a >= 10 && a < 1000)
    System.out.println("Deux chiffres pas mal");
}else{
    System.out.println("Pas très intéressant...");
}
```

Conversion

- Il est possible de convertir une variable d'un type vers un autre à l'aide de l'opérateur de cast ().

```
byte foo = 0x5a; // Code ASCII du a majuscule  
char bar = (char) foo;
```

- Le compilateur nous empêche d'effectuer des conversions impossibles
- Dans certains cas le type d'un objet n'est déterminé qu'à l'exécution, la vérification du compilateur n'est pas toujours suffisante
- Pour connaître le type d'un objet avant de le convertir on peut utiliser l'opérateur **instance of**.

Commentaires

- Les deux types de commentaire classiques du C/C++ sont supportés:

Bloc : /* */

Ligne: //

Exercice 2: manipulations des E/S

Tableaux

- Un tableau permet de stocké un ensemble de valeurs
 - Exemple une liste d'entier: 12, 21, 35, 22, 13
- On accède à une valeur spécifique du tableau en spécifiant l'index de la valeur
- En Java les tableaux sont indexés à partir de zéro
- Les tableaux doivent être typés statiquement, ils ne peuvent donc contenir qu'un seul type d'éléments

Tableaux

- Déclarer une variable de type tableau

```
<type du tableau>[] <nom de la variable>;
```

- Initialiser un tableau à l'aide de liste de valeurs

```
int[] ages = {10,15,20,30};
```

- Initialiser d'un tableau vide

```
int[] tailles = new int[10];
```

- Les tableaux sont des objets !

- Un tableau possède notamment un attribut publique: length

```
int[] tableau = new int[15];  
int tailleDuTableau = tableau.length;
```

Tableaux

- Accéder aux valeurs d'un tableau

```
int[] ages = {10,15,20,30};  
int age1 = ages[0]; // 10  
int age2 = ages[3]; // 30  
int age3 = ages[4]; // Erreur (ArrayOutOfBoundsException)
```

- En l'absence de valeur d'initialisation, les cases du tableau sont initialisées à partir de la valeur par défaut du type (null pour un objet, 0 pour un entier...)

Tableaux multi dimensionnels

- Les tableaux multi dimensionnels (tableaux de tableau) sont également supportés

```
<type du tableau> [][] <nom de la variable>;
```

```
// Le poids de deux populations différentes  
int poids [][] = { {75, 60, 67, 90}, {45, 67, 32, 90} };
```

- Mais ils sont peut utilisés car peu pratiques par rapport à l'API collection



Les boucles

- Les boucles permettent de répéter une ou plusieurs **instructions** un nombre fini ou infini de fois
- Une boucle s'exécute tant qu'une condition est remplie

La boucle while

- La boucle `while` permet d'exécuter une ou plusieurs instruction tant qu'une condition est vraie

```
while(condition){  
    // instruction(s)  
}
```

```
while(true){  
    System.out.println("Au secours une boucle infinie");  
    sleep(1000); // le programme s'arrête pendant une seconde  
}
```

```
int notes[] = { 20, 15, 13, 8, 2, 12 };  
int i = 0;  
while(i < notes.length){  
    // Affiche toutes les notes du tableau  
    System.out.println(notes[i]);  
    i++;  
}
```


La boucle do..while

- La boucle do..while vérifie la condition d'arrêt après la première exécution de la boucle
- Peu fréquemment utilisée

```
do{  
    // instructions  
}while(condition);
```

```
boolean b = false;  
do{  
    System.out.println("Voulez vous vraiment quitter ce programme ?");  
    boolean b = sc.nextBoolean();  
}while(!b);
```

La boucle for

- La boucle for permet d'exécuter une ou plusieurs instruction un nombre limité de fois

```
for(initialisation;condition;incrément){  
    // instructions  
}
```

- Est très utile pour parcourir l'ensemble des valeurs d'un tableau

```
int notes[] = { 20, 15, 13, 8, 2, 12 };  
for(int i = 0;i < notes.length;i++){  
    // Affiche toutes les notes du tableau  
    System.out.println(notes[i]);  
}
```

Exercice 3: manipulations des tableaux et des boucles

L'instruction switch

- Le switch permet d'énumérer les différentes valeurs que prend une variable et pour lesquels un traitement doit être effectué

```
switch(variable){  
    case value1:  
        instruction;  
        break;  
    default:  
        instruction;  
}
```

- Le mot clé case permet de définir le comportement à adopter si la variable prend une valeur particulière
- Le mot clé break permet de sortir de la structure conditionnel
- Le mot clé default permet de définir un comportement par défaut.

L'instruction switch

Exemple

```
int age = sc.nextInt();

switch(age){
    case 25:
        System.out.println("Un quart de siècle !");
        break;
    case 50:
        System.out.println("Un demi siècle !");
        break;
    default:
        System.out.println("Rien de spécial à fêter...");;
}
```

Enumérations

- On utilise les énumérations pour faire une liste de valeur possible
- Les cas d'usage sont nombreux: gérer une liste de code d'erreur, un ensemble de labels...
- L'utilisation d'enums permet d'éviter la duplication de valeur en dur dans le code
- Les énumérations sont des classes particulières
 - Elles doivent donc être définies dans leur propre fichier

Enumérations

Exemple

```
public enum President {  
    HOLLANDE,  
    SARKOZY,  
    CHIRAC,  
    MITTERAND  
}
```

```
public class Main {  
    public static void main(String args[]){  
        for(int i=0; i < President.values().length; i++){  
            System.out.println(President.values[i]);  
        }  
    }  
}
```



Enumérations

- Une énumération est une classe ne possédant pas de constructeur public
- On peut donc lui définir des attributs et des méthodes

Exemple avancé

```
public enum President {  
    DEGAULE("Général De Gaule"),  
    POMPIDOU("Georges POMPIDOU"),  
    VGE("Valérie Giscard D'Estaing"),  
    MITTERAND("François Mitterand"),  
    CHIRAC("Jacques Chirac"),  
    SARKOZY("Nicolas Sarkozy"),  
    HOLLANDE("François Hollande");  
  
    private final String nom;  
  
    private President(String nom){  
        this.nom = nom;  
    }  
  
    public String getNom() {  
        return nom;  
    }  
}
```

```
switch (p) {  
    case SARKOZY:  
        lobby.verserDesPotsDeVin();  
        break;  
    case CHIRAC:  
    case HOLLANDE:  
        lobby.offrirUnRepas();  
        break;  
    case VGE:  
        lobby.seduire();  
        break;  
    default:  
        lobby.attendreLeSuivant();  
}
```

Exercice 4: manipulations des énumérations