

CMI - L3 : Programmation

BULOUP Frank

Aix Marseille Université
Institut des Sciences du Mouvement



Plan de cette séquence

- 1 Structure d'un ordinateur
- 2 Que peut-on faire avec un ordinateur ?
- 3 Les langages de programmation haut niveau - Java

Programmation

- 1 Structure d'un ordinateur
- 2 Que peut-on faire avec un ordinateur ?
- 3 Les langages de programmation haut niveau - Java



Le microprocesseur

- Éléments de base : le transistor
- De nos jours, plus d'un milliard par puce
- Loi de Moore : doublement du nombre de transistors tous les deux ans



Pour les programmes et les données

Mémoire volatile (RAM)

- rapide et chère
- perd la mémoire !



Pour les programmes et les données

Mémoire volatile (RAM)

- rapide et chère
- perd la mémoire !

Mémoire permanente (Disques)

- plus lente, grande capacité et moins chère au kilo octet
- garde données et programmes à l'extinction



Pour les programmes et les données

Mémoire volatile (RAM)

- rapide et chère
- perd la mémoire !

Mémoire permanente (Disques)

- plus lente, grande capacité et moins chère au kilo octet
- garde données et programmes à l'extinction

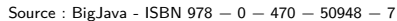
On ne sait pas encore allier grande capacité et rapidité

100



100





Programmation

- 1 Structure d'un ordinateur
- 2 Que peut-on faire avec un ordinateur ?
- 3 Les langages de programmation haut niveau - Java

Utiliser des logiciels



- Bureautique
- Calculs
- Jeux
- etc.

Utiliser des logiciels



- Bureautique
- Calculs
- Jeux
- etc.

**Les ordinateurs sont très flexibles :
les taches qu'on peut leur faire faire sont variées**

Créer des logiciels

Un ordinateur doit être programmé pour réaliser un ensemble de tâches complexes et cohérentes. Chacun de ces ensembles peut être appelé **programme**.

Créer des logiciels

Un ordinateur doit être programmé pour réaliser un ensemble de tâches complexes et cohérentes. Chacun de ces ensembles peut être appelé **programme**.

La programmation, c'est quoi ?

Concrètement, une tâche d'un programme est une succession de très nombreuses instructions extrêmement simples exécutées très rapidement.

Créer des logiciels

Un ordinateur doit être programmé pour réaliser un ensemble de tâches complexes et cohérentes. Chacun de ces ensembles peut être appelé **programme**.

La programmation, c'est quoi ?

Concrètement, une tâche d'un programme est une succession de très nombreuses instructions extrêmement simples exécutées très rapidement.

La programmation consiste à écrire ces instructions dans un langage compréhensible pour un ordinateur.

Une instruction élémentaire, c'est quoi ?

Exemples :

- Charger une valeur d'une zone mémoire RAM spécifique dans le CPU
- Multiplier deux nombres de zone mémoire RAM spécifique et placer le résultat dans une zone mémoire spécifique
- Si le resultat de l'opération précédente est positif, continuer à l'instruction présente dans une zone mémoire spécifique

Une instruction élémentaire, c'est quoi ?

Exemples :

- Charger une valeur d'une zone mémoire RAM spécifique dans le CPU
- Multiplier deux nombres de zone mémoire RAM spécifique et placer le résultat dans une zone mémoire spécifique
- Si le resultat de l'opération précédente est positif, continuer à l'instruction présente dans une zone mémoire spécifique

Ces instructions (le programme) sont chargées en mémoire RAM lorsque le programme est lancé. Elles sont ensuite lues et exécutées par le CPU. Elles ont toutes un code binaire unique.

À quoi ressemble le code binaire d'une instruction ?

10110000 01100001

À quoi ressemble le code binaire d'une instruction ?

10110000 01100001

En pratique, les instructions sont codées sur 32 ou 64 bits. Il en est de même pour les adresses mémoires (zones mémoires).

C'est ce qu'on appelle le **code machine**

À quoi ressemble le code binaire d'une instruction ?

10110000 01100001

En pratique, les instructions sont codées sur 32 ou 64 bits. Il en est de même pour les adresses mémoires (zones mémoires).

C'est ce qu'on appelle le **code machine**

Exemple de code machine

Voici un exemple de code machine exprimé en hexadécimal :

0x14200003
0x3c01003d
0x34280914
0x21280001
0x00094021

- Sur combien de bits est exprimé ce code ?
- Qu'en pensez-vous ?

Le langage machine est performant mais il est...

- difficilement compréhensible pour un humain
- sujet à de nombreuses erreurs
- d'une maintenance très difficile
- difficilement portable (spécifique à chaque CPU)

Le langage machine est performant mais il est...

- difficilement compréhensible pour un humain
- sujet à de nombreuses erreurs
- d'une maintenance très difficile
- difficilement portable (spécifique à chaque CPU)

Première solution

⇒ langages de bas niveau

Comme l'assembleur par exemple (Cf. ce lien)

Le langage machine est performant mais il est...

- difficilement compréhensible pour un humain
- sujet à de nombreuses erreurs
- d'une maintenance très difficile
- difficilement portable (spécifique à chaque CPU)

Première solution

⇒ langages de bas niveau

Comme l'assembleur par exemple (Cf. ce lien)

- Un compilateur traduit le tout en langage machine
- Conserve, à peu de chose près, les mêmes inconvénients...

Programmation

- 1 Structure d'un ordinateur
- 2 Que peut-on faire avec un ordinateur ?
- 3 Les langages de programmation haut niveau - Java

Réponse partielle aux premières difficultés

Les langages de haut niveau procéduraux :

C, Pascal, COBOL, Fortran etc.

Mais :

- les programmes ont augmentés en complexité
- un même programme doit toujours être recompilé pour fonctionner sur une autre architecture

Solutions aux difficultés

Langages interprétés et orientés objets

Python, PHP, Ruby, Java etc.

Solutions aux difficultés

Langages interprétés et orientés objets

Langage interprété - Le cas de Java

Un compilateur génère un langage machine (bytecode) indépendant de l'architecture de l'ordinateur et donc du CPU.

Langages interprétés et orientés objets

Un compilateur génère un langage machine (bytecode) indépendant de l'architecture de l'ordinateur et donc du CPU. Une Machine Virtuelle Java (JVM), dépendante de l'architecture du système hôte, comporte un interpréteur.

Langages interprétés et orientés objets

Un compilateur génère un langage machine (bytecode) indépendant de l'architecture de l'ordinateur et donc du CPU. Une Machine Virtuelle Java (JVM), dépendante de l'architecture du système hôte, comporte un interpréteur. Cet interpréteur traduit le bytecode en langage machine propre au système hôte.

Langages interprétés et orientés objets

Un compilateur génère un langage machine (bytecode) indépendant de l'architecture de l'ordinateur et donc du CPU. Une Machine Virtuelle Java (JVM), dépendante de l'architecture du système hôte, comporte un interpréteur. Cet interpréteur traduit le bytecode en langage machine propre au système hôte.

Apporte une méthodologie de conception et de programmation :

- robustesse (tolérance au changement)
- modularité
- lisibilité

1

Caractéristiques

Une variable :

- porte un nom
- est d'un type donné
- occupe une zone mémoire RAM dont la taille dépend du type
- contient une information qui peut changer

Déclaration

Type *nomVariable*;

Type *nomVariable* = valeurInitiale ; valeurInitiale est appelée littéral par opposition au concept de variable.

```
1  int var1;  
2  int var2 = 10;  
3  double var3 = 10.23;
```

Exemples de déclarations de variables en Java

- La variable *var1* est de type entier.
- *var2* est de type entier. Elle est initialisée avec le littéral 10.
- *var3* est de type double. Elle est également initialisée avec le littéral 10.23.

Les types entiers

- byte, sur un octet
- short, sur 2 octets
- int, sur 4 octets
- long, sur 8 octets

La représentation est au format binaire complément à deux

Les types entiers

- byte, sur un octet
- short, sur 2 octets
- int, sur 4 octets
- long, sur 8 octets

La représentation est au format binaire complément à deux

Les types flottants

- float, sur 4 octets
- double, sur 8 octets

La représentation est au format IEEE754

Les autres types primitifs

- boolean, (true|false) taille non spécifiée, minimum 1 bit
- char, sur 2 octets (binaire naturel), caractères

```
1  int var1; // Initialized to zero by default
2  int var2 = 10;
3  double var3 = 10;
4  boolean var4; // Initialized to false by default
5  boolean var5 = true;
6  char var6 = 'c'; // Initialized with letter c
```

Exemples de déclarations de variables en Java

Les littéraux entiers

Par défaut, les littéraux entiers sont de type *int*. On peut les typer *long* en les suffixant de la lettre l ou L. Exemple :

```
long var = 234L ;
```

Les littéraux flottants

Par défaut, les littéraux flottants sont de type *double*. On peut les typer *float* en les suffixant de la lettre f ou F. Exemple :

```
float var = 2.34f ;
```

On manipule souvent des variables de types différents
⇒ transformation de type

```
1  int var1 = 10;  
2  long var2 = 10;  
3  double var3 = var1 + var2;
```

Exemples de transtypages

- Ligne 2 : le littéral 10, de type *int*, est affecté à une variable de type *long* ⇒ transtypage
- Ligne 3 : les deux variables sont de types différents. Le compilateur transtype *var1* en *long* avant de compiler la somme
- Ligne 3 : le résultat de la somme, de type *long*, est affecté à un *double* ⇒ transtypage

Les transtypages précédents sont dits implicites. Les suivants sont explicites :

```
1  int var1 = 128;
2  long var2 = 456;
3  int var3 = var1 + (int) var2;
4  byte var4 = (byte) var1;
```

Exemples de transtypages explicites

- Ligne 3 : les deux variables sont de types différents. On force le compilateur à transtyper *var2* en *int* avant de compiler la somme $\Rightarrow$ altération possible de l'information.
- Ligne 4 : On force à transtyper *var1* en *byte* avant d'affecter le résultat à *var4* $\Rightarrow$ altération possible de l'information.

Les différents types primitifs en Java

Type	Contenu	Valeur par défaut	Taille
boolean	true ou false	false	1 bit
char	Unicode	\u0000	16 bits
byte	Entier signé	0	8 bits
short	Entier signé	0	16 bits
int	Entier signé	0	32 bits
long	Entier signé	0	64 bits
float	Virgule flottante	0.0	32 bits
double	Virgule flottante	0.0	64 bits

[Voir ici pour plus de détails](#)

Opérateur

- de calcul
- d'assignation
- d'incrémentation
- de comparaison
- logiques
- bit à bit
- de rotation de bits
- ternaire

[Voir ici pour plus de détails](#)

Operateurs	Nom	Exemple (int x = 10)
+	addition	$x + 7 = 17$
-	soustraction	$x - 7 = 3$
*	multiplication	$x * 7 = 70$
/	division	$x / 7 = 1$
%	modulo	$x \% 7 = 3$
=	affectation	$x = 12$

Operateurs	Nom	Exemple (int x = 10)
+=	addition	$x+=7 \Leftrightarrow x = x + 7$
-=	soustraction	$x-=7 \Leftrightarrow x = x - 7$
=	multiplication	$x=7 \Leftrightarrow x = x * 7$
/=	division	$x/=7 \Leftrightarrow x = x / 7$
%=	modulo	$x\%=7 \Leftrightarrow x = x \% 7$

Opérateurs	Nom	Exemple (int x = 10)
++	addition	$y = x++ \Rightarrow y = 10$
--	soustraction	$y = --x \Rightarrow y = 9$

Operateurs	Nom	e.g. ($x = 10$, $y = 5$)
==	test d'égalité	$x==y$ retourne false
<	test d'infériorité stricte	$x<y$ retourne false
<=	test d'infériorité	$x<=y$ retourne false
>	test de supériorité stricte	$x>y$ retourne true
>=	test de supériorité	$x>=y$ retourne true
!=	test d'inégalité	$x!=y$ retourne true

Operateurs	Nom	e.g. (x = true, y = false)
	Ou	x y retourne true
&&	Et	x&&y retourne false
!	Non	!y retourne true

Opérateurs	Nom	e.g. ($x = 10$, $y = 5$)
	Ou	$x y$ retourne 0
&	Et	$x\&y$ retourne 15
$\wedge$	Ou exclusif	$x \wedge y$ retourne 15

Opérateurs	Nom	e.g. ($x = 10$)
$\ll$	Décalage à gauche	$x \ll 1$ retourne 20
$\gg$	Décalage à droite avec signe	$x \gg 1$ retourne 5
$\ggg$	Décalage à gauche avec zéros	$x \ggg 1$ retourne 5

`variable = (condition) ? (valeur si vraie) : (valeur si fausse);`

```
1  int var1 = 10;
2  int var2 = 11;
3  int var3 = (var1 > var2) ? var1 : var2;
4  //var3 = max(var1, var2) = 11
```

Exercice I - Premiers pas avec Eclipse

Utilisation d'Eclipse :

- Lancer Eclipse
- Créer un projet Java nommé **FirstStep**
- Créer un fichier "brouillon" dans ce projet

Exercice II - Déclaration et manipulation de variables

Dans ce fichier brouillon :

- Déclarer 4 variables (*var0*, *var1*, *var2* et *var3*) de type entier. *var0* doit être initialisée à 2, *var1* à 10 et *var3* à 5.
- Effectuer le calcul suivant : $var2 = var1 + 2var3 + var0$
- Afficher le résultat en utilisant la méthode `System.out.println("... ")` : $var2 = var1 + 2var3 + var0$

Exercice III - Transtypage de variables

Dans un nouveau fichier brouillon, effectuer et afficher le calcul suivant : $var0 = var1 + var2$, lorsque $var1$ et $var2$ sont de type double et sont initialisées à 0.6 et 0.4 respectivement. Y a-t-il un problème ? Si oui, résolvez-le par deux puis une opération de transtypage. Commentez chacun des deux résultats.

Exercice IV - Calcul de surfaces

Créer un petit programme qui calcule les aires d'un rectangle et d'un cercle à partir de la largeur, de la hauteur et du rayon.

Solution possible :

```
1  //Rectangle parameters
2  double width = 10;
3  double height = 10;
4  //Compute rectangle perimeter
5  double perimeter = 2*width + 2*height;
6  //Print result for rectangle
7  System.out.printf("Rect. \u00perim. \u00=%f\n", perimeter);
8
9  //Circle parameters
10 double radius = 10;
11 //Compute circle perimeter
12 //Don't need to redefine "perimeter" variable
13 perimeter = 2*3.14*radius;
14 //Print result for circle
15 System.out.printf("Circ. \u00perim. \u00=%f", perimeter);
```

Exercice V - Critique de la programmation "classique"

Faites une analyse critique du petit programme précédent

Exercice V - Critique de la programmation "classique"

Faites une analyse critique du petit programme précédent

Critiques du petit programme

- Absence de lien évident entre largeur et hauteur
- Absence de lien évident entre ces variables et les calculs
- Les données et traitements sont séparés

absence de lien sémantique entre données et traitement

Regrouper données et traitements dans une même entité

⇒ C'est le concept de **classe**

Rectangle

width : double

height : double

getPerimeter() : double

getSurface() : double

Circle

radius : double

getPerimeter() : double

getSurface() : double

Rectangle

width : double

height : double

getPerimeter() : double

getSurface() : double

Circle

radius : double

getPerimeter() : double

getSurface() : double

C'est la notion d'encapsulation

Données et traitements ont été "encapsulés" dans une classe

Les paquetages, c'est quoi ?

- Structuration des classes selon une arborescence (répertoires)
- Les noms des paquets sont sources d'informations
- Participent à l'encapsulation (éclaircissement plus tard !)

fr.amu.fss.l3.cmi.geometry

Rectangle

width : double

height : double

getPerimeter() : double

getSurface() : double

Circle

radius : double

getPerimeter() : double

getSurface() : double

Exercice VI - Mise en oeuvre sous Eclipse

Sous Eclipse

- Créer un nouveau projet nommé **Geometrie**
- Créer un package **geometry** dans ce projet
- Créer les classes **Rectangle** et **Circle** dans ce package
- Ajouter les attributs (variables) et les méthodes (traitements) à ces classes

Exercice VI - Mise en oeuvre sous Eclipse

Sous Eclipse

- Créer un nouveau projet nommé **Geometrie**
- Créer un package **geometry** dans ce projet
- Créer les classes **Rectangle** et **Circle** dans ce package
- Ajouter les attributs (variables) et les méthodes (traitements) à ces classes

Visibilité des attributs d'une classe

Les méthodes d'une classe ont accès aux attributs de cette classe.

Exercice VI - Mise en oeuvre sous Eclipse

Sous Eclipse

- Créer un nouveau projet nommé **Geometrie**
- Créer un package **geometry** dans ce projet
- Créer les classes **Rectangle** et **Circle** dans ce package
- Ajouter les attributs (variables) et les méthodes (traitements) à ces classes

Visibilité des attributs d'une classe

Les méthodes d'une classe ont accès aux attributs de cette classe.

Mais je fais quoi avec ces classes maintenant moi ???!!!

Comment créer un rectangle de hauteur 2 et de largeur 3 ?

J'utilise le **constructeur** de la classe et j'accède aux attributs et méthodes à l'aide de l'opérateur "."

```
1 Rectangle rect = new Rectangle();  
2 rect.width = 12;
```

Exercice VII - Mise en oeuvre sous Eclipse suite. . .

- Créer une nouvelle classe nommée **Test** dans le package **geometry**. Cette classe comportera une méthode **main(...)**
- Dans cette méthode, créer un cercle et un rectangle puis afficher leurs périmètre et surface.

Exercice VIII - Et le bytecode, où est-il ?

- Dans un terminal (une console), aller dans le répertoire contenant les fichiers compilés
- Utiliser la commande `javap -c` pour obtenir le bytecode

Cf. [ce lien](#) pour plus de détails

Exercice X - Création de classes

- ❶ Créez une classe nommée **Trinomial** comportant trois attributs de type double pour les coefficients du polynôme $ax^2 + bx + c$. Vous appellerez ces attributs **a**, **b** et **c**.
- ❷ Prévoyez les méthodes **getFirstZero()** et **getSecondZero()** qui renvoient les valeurs des deux racines de ce polynôme.
- ❸ Testez cette classe dans les cas suivants :
 - a) $a = 1, b = -3, c = -18$
 - b) $a = 1, b = 6, c = 9$

Pour répondre au problème des racines complexes, nous allons utiliser la librairie Apache Common Math.

- ❹ Télécharger et installer cette librairie dans votre projet.
- ❺ Modifier votre code pour prendre en compte le cas des racines complexes.