Introduction et rappels

Cours 1 / GL / LP DA2I

Cédric Lhoussaine

2019-2020

Présentation du génie logiciel

Qu'est-ce que l'informatique ?

Définition 1

Informatique : science du traitement de l'information

Qu'est-ce que l'informatique ?

Définition 1

Informatique : science du traitement de l'information

Définition 2

Computer Science : science des calculateurs

Qu'est-ce que l'informatique ?

Définition 1

Informatique : science du traitement de l'information

Définition 2

Computer Science : science des calculateurs

Définition 3

Software Engineering : génie logiciel

Axiomes fondamentaux

- traiter de l'information = faire un calcul (Leibniz, Boole, Shannon)
- tout calcul peut être fait par une machine (Babbage, Turing, Von Neumann)

Information et Calcul

Axiomes fondamentaux

- traiter de l'information = faire un calcul (Leibniz, Boole, Shannon)
- tout calcul peut être fait par une machine (Babbage, Turing, Von Neumann)

Deux étapes clefs

- fin XIXe s. : machines de traitement automatique et mécanique de l'information (Jacquard, Babbage)
- 2e G.M. : calculateurs électroniques et théorie de la calculabilité (Von Neumann, Wiener, Turing): machines universelles.

Le génie logiciel

- concevoir et réaliser des programmes d'après les spécifications des utilisateurs
- s'abstraire autant que possible de la spécificité du matériel
- savoir réutiliser ce qui existe
- faciliter la maintenance
- garantir fiabilité, sécurité, confidentialité

Objectifs du cours

- Génie logiciel : comment concevoir des applications "proprement"
- Conception par objets : comment analyser et décomposer un problème en termes de classes et d'envois de messages
- Programmation par objets :
  - pratique du langage Java
  - aperçu de quelques autres langages à objets (Python, Scala...)

Quelques rappels. . .

Algorithmique

Discipline qui étudie la décomposition d'un calcul en une série d'étapes, au moyen (dans le paradigme *impératif*):

- de variables contenant des *données* de façon plus ou moins structurée;
- d'instructions (ou commandes) représentant les opérations élémentaires réalisables à partir des variables;
- de structures de contrôle permettant d'exécuter des parties du programme sous certaines conditions ou de façon répétitive.

Les langages de programmation

- L'algorithme est *implémenté* dans un langage particulier.
- 3 principaux paradigmes:
  - *impératif*
  - *fonctionnel*
  - *objets*

Paradigme impératif

Séquence d'opérations sur des variables mutables pour construire le résultat

```
x = 5*2; x = x+3;
```

Chaque opération change l'état (= mémoire) de la machine (= ordinateur).

Exemple: Cobol, C, Pascal, etc.

Paradigme fonctionnel

Composition de fonctions appliquées aux données d'entrée

```
plus(3, fois(5,2))
```

3 concepts fondamentaux:

- valeurs (= constantes de caractère, entières, etc.)
- abstraction (fonctionnelle)
- application (d'une fonction à un argument)

Exemple: Lisp, Caml, Haskell

Paradigme objets

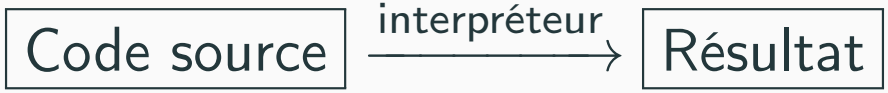
Décomposition en entités autonomes qui s'échangent des messages

```
(5.fois(2)).plus(3);
```

Exemple: Java, C++, Smalltalk, Python, Scala

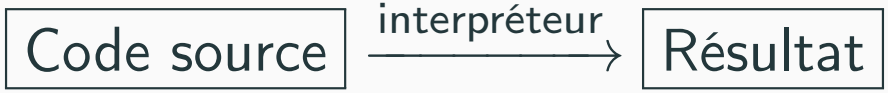
Mise en oeuvre

langages interprétés



Mise en oeuvre

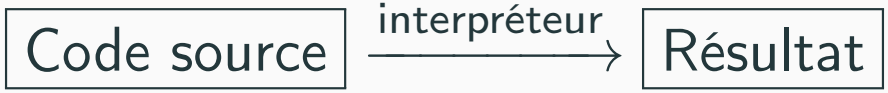
langages interprétés



langages compilés

Mise en oeuvre

langages interprétés



langages compilés

langage byte-compilé

Les erreurs de programmation

Les erreurs procèdent de divers niveaux:

- à la compilation : erreurs de syntaxe
- durant l'exécution : erreurs de sémantique
- résultat d'exécution : erreurs algorithmiques

Types de données

Les données et variables sont (*parfois*) associées à un type (entier, caractère, booléen, . . .) qui :

- les étiquette
- restreint leur emploi
- donc limite les erreurs de programmation

Intérêt d'un langage fortement typé

- toute donnée possède un type
- la déclaration d'une variable passe par un type
- tout opérateur ou instruction ne peut s'utiliser que sur un ensemble précis de types
- les conversions de types sont contrôlées
  - implicitement autorisées si "naturelles"
  - explicites et restreintes si "risquées"
- l'utilisateur peut définir ses propres types

Toute vérification liée aux types est faite par le compilateur

Java

La langage Java

C'est un langage "récent" (1995 !)

- approche bytecode donc portable
- Compilateur : `javac`

```
$ javac MonFichier.java
```

  produit des fichiers `*.class`
- Interpréteur : `java`

```
$ java MaClasse
```

  si on dispose d'un fichier `MaClasse.class`
- Système de documentation : `javadoc` produit des pages HTML
- Archiveur : `jar`
- Debugger : `jdb`

Types primitifs en Java

Types correspondants à des valeurs: `int`, `char`, `boolean`, `double`...

Exemples :

```
int i = 5 ;
```

`i` contient 00000000000000000000000000000101

```
char c = 'A';
```

`c` contient 0000000001000001 (65)

Types "objets" en Java

Types objets = références = adresses

- classes `String`, `Integer`, `Object`...
- tableaux `int[]`, `String[][]`...

Exemples:

```
String s = "ABC";
```

s: 32a8d → @32a8d: "AbC"

```
Date d = new Date(8, 4, 2006);
```

d: 6f217 → @6f217: Datejour=8; mois=4;année=2006

La variable ne contient que l'adresse où les données sont rangées, `null` si pas d'objet

Conversions de types

Certains types peuvent être convertis en d'autres:

- soit *implicitement*: type limité $\rightarrow$ type plus général

```
int i = 3;
float x = i;
char c = 'A';
int a = c;
```

- soit *explicitement* : type général $\rightarrow$ type plus limité

```
double x = 3.5;
int i = (int) x;
char c = (char) 65;
```

Pas de conversion entre types primitifs et objets ! !

Programmer en Java : paquetages

Hiérarchie de paquetages (packages) = modules structurés de façon arborescente.

- `java.lang.*` contient les classes "de base"
- `java.awt.*` ou `javax.swing.*` : interfaces graphiques
- `java.util.*` : collections, calendriers, etc.

Utilisation

```
import le.chemin.du.package.Classe;
```

Création

```
package le.chemin.du.package;
```

Un peu de méthode

Écrire lisiblement

- indentation
- choix des noms de variables et de méthodes
- respect des conventions de casse : `maVariable`, `maMethode()`, `CONSTANTE`, `Classe...`
- commentaires "classiques" :
  - `// blabla...` en fin de ligne
  - `/* blabla... */` n'importe où
- documentation traitée par `javadoc`:

```
/** Cette classe sert à ...
 * Elle s'utilise ...
 * <B>IMPORTANT :</B> ...
*/
```

Organisation

- ne pas négliger la phase d'analyse-conception
- tester les fonctions une par une !
  - ce qui doit marcher (évidemment !)
  - ce qui ne doit pas marcher
- séparation entre fichiers sources, exécutables, et documentation
- création ou utilisation de *packages* (bibliothèques)
- pour les gros projets en équipe :
  - gestionnaires de versions : Subversion, GIT...
  - test unitaires systématiques JUnit,...

Et pour finir...

L'évolution des techniques (matérielles et logicielles) impose de savoir s'adapter:

- apprendre des concepts plutôt que des techniques
- apprendre à apprendre