

IFT 1015 - Introduction

Professeur:
Stefan Monnier

B. Kégl, S. Roy, F. Duranleau, S. Monnier
Département d'informatique et de recherche opérationnelle
Université de Montréal

hiver 2006

Au programme

- Plan de cours
 - Contact
 - Objectifs
 - Évaluation
 - Travaux pratiques
 - ...
- Début de la matière

1 page de web:

<http://www2.iro.umontreal.ca/~dift1015/A04/>

2 démonstrateurs:

dift1015@iro.umontreal.ca

3 professeurs:

{kegl,roys}@iro.umontreal.ca

● Pour rejoindre tout le monde du cours:

gift1015@iro.umontreal.ca

”Être capable d’analyser un problème, de proposer une stratégie pour le résoudre par ordinateur, et de programmer correctement cette solution en Java.”

- Analyse de problème
- Données et Algorithmes
- Approche orientée objet
- Java

Bref,

Programmation \neq Java

- **Commandes** de base d'ordinateur
 - Environnement Unix (**GNU/Linux**)
 - Éditeur de texte (**vi**, **Emacs**, ...)
 - Gestion des fichiers (**ls**, **cp**, **mv**, **rm**, ...)(→ **Séminaires Unix**)
- Aptitude en **résolution de problèmes**
- **Mathématiques** de base
- **Aucun expérience en programmation requise**

Il n'y a aucun livre obligatoire. Nous suggérons:

- Anne Tasso, *Le livre de Java premier langage*, Eyrolles, 2002 (ISBN 2-212-11100-2).
- J. Niño et F. A. Hosch, *An Introduction to Programming and Object Oriented Design*, Wiley, 2002 (ISBN 0-471-35489-9).

Autres livres utiles:

- C. Delannoy, *Programmer en Java*, Eyrolles, 2002 (ISBN 2-212-11119-3)
- Cay Horstmann, *Big Java*. John Wiley, 2002, 1^{ère} édition
- Jean-Bernard Boichat, *Apprendre Java et C++ en parallèle*, Eyrolles, 2002 (ISBN 2-212-11152-5)

Évaluation

Théorie		
Examen Intra	25%	20-02-2006
Examen Final	25%	24-04-2006
Pratique		
Examen Pratique	10%	mi-session
10 Exercices	10%	1/semaine
2 Travaux pratiques	2 × 15%	

- **Seuil** (conditions pour que les TPs et exercices comptent)
 1. Moyenne pondérée Intra et Final $\geq 40/100$
 2. Examen Pratique $\geq 40/100$

Intra et Final

- **Intra**: examen écrit, durée 2 heures, matière théorique du début à mi-session, aucune documentation n'est permise
- **Final**: examen écrit, durée 2h45, matière théorique **du début** à la fin de la session, aucune documentation n'est permise

Examen Pratique

- Évaluation directe de l'aptitude à réaliser un programme
 - se fait devant un ordinateur, en présence d'un évaluateur
 - 2-3 problèmes simples programmés au complet
 - durée à déterminer (autour de 30-45 minutes)
 - un seul examen, à la mi-session
 - droit de reprise
 - doit être réussi avec une note supérieure ou égale à 40%.

- vise à développer les aptitudes en programmation
- Problèmes simples reliés à la matière vue chaque semaine
 - doivent être réalisés seul
 - remis sous forme de rapport écrit ou de programme
 - 10 exercices (1/semaine)

Travaux pratiques

- vise à développer les aptitudes en programmation et la capacité de résoudre de plus gros problèmes
 - À réaliser seul ou en équipe de deux
 - 2 travaux pratiques (15% chacun) dans la session

Faites vos travaux vous-même!

- Première délit: Zéro pour le travail
- Seconde délit: Échec au cours

En cas de doute sur l'origine d'un travail pratique, nous nous réservons le droit de questionner un étudiant sur les détails du travail remis.

Tous les membres d'une équipe doivent participer à tous les travaux pratiques.

Pourquoi Java?

- Pourquoi pas?
 - langage moderne (**moins de détails** étranges que le C++)
 - utilise l'approche **orientée objet** (OO)
 - syntaxe de base **similaire au C et C++**
 - **bibliothèque standard** très riche
- Important : le langage a peu d'importance
 - un bon programmeur est bon dans **tous les langages**
 - il peut les **apprendre au besoin**

Références

- On réfère au premier chapitre du livre de **Anne Tasso** (*Java premier langage*) par [Tasso:1]
- On réfère au livre *Object Oriented Programming* de **J. Niño** par [Niño]

[Tasso:Introduction] et [Niño:Chapitre 1]

- Définir "Informatique"
- Qu'est-ce qu'un **algorithme**
- Qu'est-ce qu'un **ordinateur**?
- Qu'est-ce qu'un **programme**?
- Les **langages** de programmation
- Les **erreurs** de programmation
- Mon **premier programme** Java

*”L’informatique est la **science de l’abstraction**. Elle vise à créer le bon **modèle** pour un problème et conçoit une technique **”méchanisable”** pour le résoudre.”*

(Aho & Ullman)

- Science de **construire des algorithmes**

Construire un algorithme

- Un ordinateur peut ...
 - calculer
 - compter
 - trier
 - rechercher
- ... à condition qu'on **lui dise quoi faire.**

Construire un algorithme

- Un ordinateur est
 - rapide mais
 - stupide
- L'algorithme définit la marche à suivre des tâches à exécuter

Programmer = Cuisiner

Problème : comment faire une crème anglaise?

Ingrédients

- 8 jaunes d'oeufs
- 250g de sucre
- 500cl de lait
- vanille

Algorithme

- battre les jaunes et le sucre en un mélange lisse
- ajoutez peu à peu le lait porté à ébullition au mélange en brassant
- laissez cuire 12 minutes en remuant et en évitant de faire bouillir

Programmer = Cuisiner

Quel est le bon niveau de détail?

- Ingrédients
 - 8 oeufs
 - . . .
- Algorithme
 - pour chacun des 8 oeufs, faire:
 - * casser la coquille sur le bord d'un bol
 - * séparer le jaune du blanc

Dans une recette normale:

- beaucoup d'ingrédients implicites
- beaucoup d'algorithmes plus simples supposés connus

Programmer = Cuisiner

Problème : comment faire une charlotte aux poires?

Ingrédients

- 750ml de crème anglaise ([voir page 73](#))
- 500ml de crème fouettée
- 1kg de poires pochées ([voir page 22](#))
- ...

Algorithme

- Mélanger la crème anglaise et la crème fouettée
- ...

Recette complexe \Rightarrow [ingrédients complexes](#) et [algorithmes connus](#)

Qu'est-ce qu'un programme?

- Un programme est une **séquence d'instructions à effectuer**
 - "séquence" = l'**ordre** est important
 - langage = un ensemble des **instructions élémentaires**
- **Niveaux** de programmation
 - langages **machine** et assembleur
 - langages **évolués** (C, C++, Java, ...)
 - langages **spécialisés** (Oracle, SPSS, Mathematica, PHP, Tcl/Tk, Perl, Python, Awk, ...)

Qu'est-ce qu'un ordinateur

- Appareil d'usage général pour traiter de l'information
 - machine d'entrée-sortie
 - stockage et traitement de l'information
- Stockage de l'information
 - physiquement: “interrupteur” de deux états, *on* ou *off*
 - au niveau abstrait: nombres binaires, *1* ou *0*

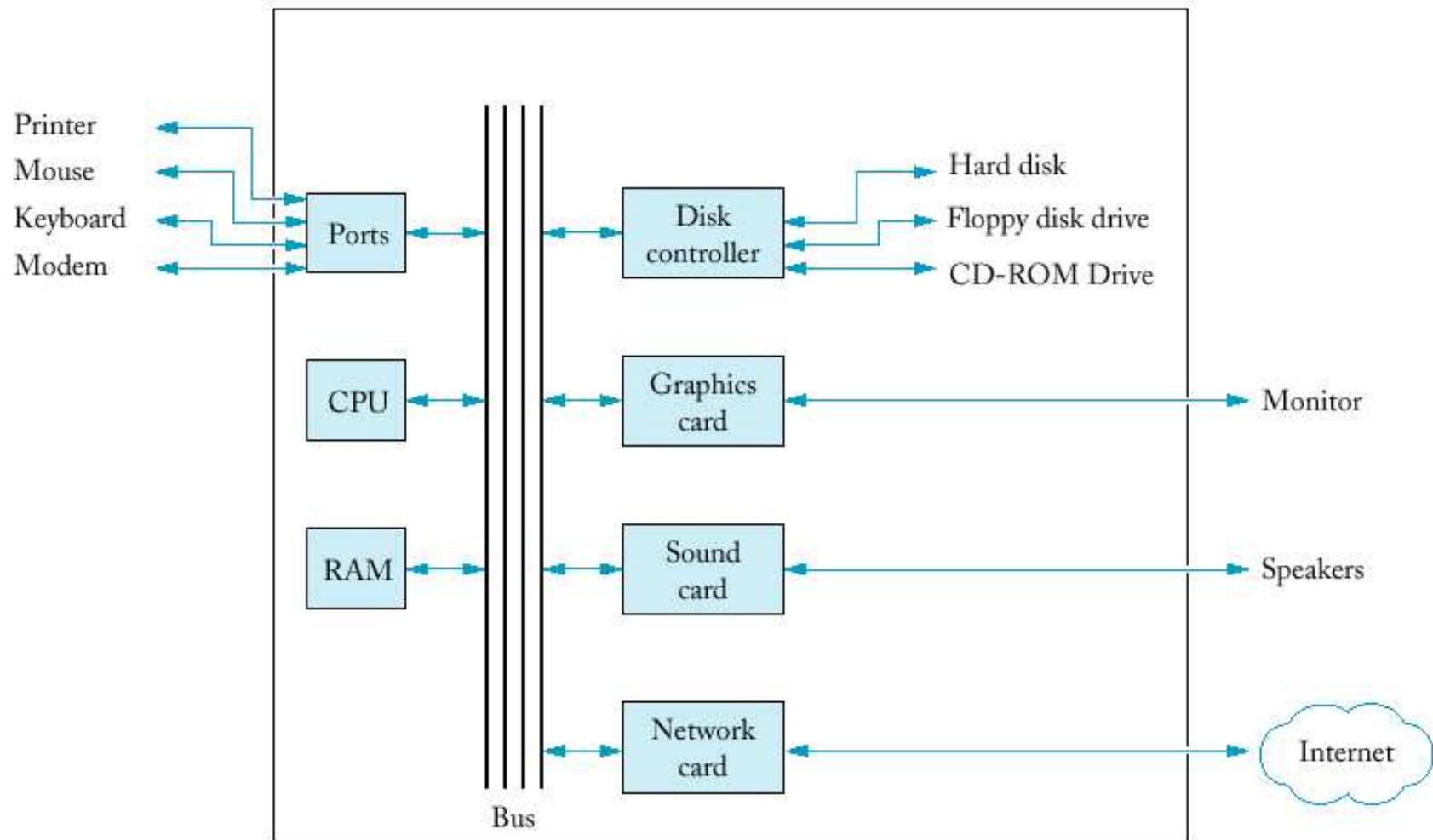
Qu'est-ce qu'un ordinateur

- Stockage des données
 - nombres entiers dans un intervalle donné:
 $0 = 0000$, $9 = 1001$, $15 = 1111$
 - caractères (code ASCII): 'A' = 01000001, 'B' = 01000010
 - images noir et blanc: 1 bit/pixel
 - etc.

Architecture de l'ordinateur

- Unité centrale (central processing unit – CPU)
 - le “cerveau” qui exécute les instructions
- Mémoire (random access memory – RAM)
 - une collection des tiroirs numérotés
- Périphériques
 - clavier, écran, carte de réseau, etc.

Diagramme schématique d'un ordinateur



Programme de style “recette”

- Tâche: **additionner deux nombres** et afficher le résultat à l'écran
 - le CPU a un **accumulateur**: mémoire “locale” pour les paramètres et les résultats du calcul

Programme de style “recette”

Programme:

1. **Lire** le premier nombre du clavier et le placer dans la mémoire.
2. **Lire** le deuxième nombre du clavier et le placer dans la mémoire.
3. **Charger** le premier nombre de la mémoire dans l’accumulateur.
4. **Ajouter** le deuxième nombre de la mémoire au nombre qui se trouve dans l’accumulateur, et placer le résultat dans l’accumulateur.
5. **Stocker** le nombre qui se trouve dans l’accumulateur dans la mémoire.
6. **Afficher** le résultat sur l’écran.

Ordinateur imaginaire simplifié

Mémoire (RAM)

- **taille** de 32 octets
- **adresses** de 5 bits

CPU

- **accumulateur**: mémoire “locale” de 8 bits
- **5 instructions élémentaires**: `read`, `write`, `load`, `save`, `add`
- chaque instruction a **un paramètre**

Ordinateur imaginaire simplifié

Instructions élémentaires du CPU

read n : lire un nombre du clavier et le placer dans la mémoire à l'adresse n

write n : afficher sur l'écran le nombre qui se trouve dans la mémoire à l'adresse n

load n : mettre le nombre qui se trouve dans la mémoire à l'adresse n dans l'accumulateur

save n : mettre le nombre qui se trouve dans l'accumulateur dans la mémoire à l'adresse n

add n : ajouter n au nombre qui se trouve dans l'accumulateur, et placer le résultat dans l'accumulateur

Programme assembleur

```
1. read 20
2. read 21
3. load 20
4. add 21
5. save 22
6. write 22
```

Programme en langage machine

Codes binaires des instructions:

```
read = 000
write = 001
load = 010
save = 011
add = 100
```

Programme:

```
1. read 20 = 00010100
2. read 21 = 00010101
3. load 20 = 01010100
4. add 21 = 10010101
5. save 22 = 01110110
6. write 22 = 00110110
```

Programme en langage machine

- Exécution du programme:
 - écrire le programme dans un fichier
 - charger le programme du fichier dans la mémoire à l'adresse 5
 - commander au CPU d'interpréter le contenu de la mémoire à partir de l'adresse 5 jusqu'à l'adresse 10 comme un programme
 - exécuter le programme instruction par instruction

Programme en langage machine

Le cycle d'exécution:

1. $i \leftarrow 5$
2. lire le contenu de l'adresse i de la mémoire
3. décoder les premiers 3 bits pour obtenir l'instruction `inst`
4. décoder les derniers 5 bits pour obtenir le paramètre `n`
5. exécuter `inst n`
6. $i \leftarrow i + 1$
7. **Si** ($i == 11$) **Alors** STOP **Sinon** GOTO 2

Avantages

- très rapide
- accès direct aux composantes matérielles

Inconvénients

- difficile à écrire et lire (comprendre)
- détection des erreurs très difficile

Langage de haut niveau

Avantages

- isole le programmeur du langage machine
- instructions plus riches et plus **compréhensibles**
- concepts plus **abstrait**s (variable, structure, ...)

Inconvénients

- incompréhensible pour l'unité centrale (CPU)
- doit être **traduit** (ou interprété) en langage machine pour s'exécuter
→ **compilation**

Programme

```
int num1,num2,sum;  
num1 = read();  
num2 = read();  
sum = num1 + num2;  
print(sum);
```

- instructions plus **compréhensibles**
- instructions de **niveau plus élevé**
- **variables**: références abstraites au contenu de la mémoire

Erreurs de programmation

Ordinateur = exécutant

- l'ordinateur est **incapable** de distinguer une "bonne" instruction d'une "mauvaise"
- l'ordinateur **exécute tout** ce qu'on lui demande d'exécuter
- c'est la **responsabilité du programmeur** de s'assurer que l'ordinateur fait vraiment ce qu'on veut qu'il fasse

Stratégies de programmation

Approche **procédurale**: *PROGRAMME = ensemble de fonctions*

- exemples: Pascal, FORTRAN, C

Technologie “**pré-industrielle**”

- **besoin d'experts** très compétents pour le développement et la maintenance
- le développement est **long**
- les morceaux de programmes **ne sont pas interchangeables**

Stratégies de programmation

Approche **orientée objet**: *PROGRAMME = ensemble d'objets*

- exemples: Ada, C++, Java

Avantages

- permettre la **conception** de haut niveau
- le développement est **moins long**, plus **sûr**
- les programmes sont plus **faciles à comprendre**
- plus **facile d'échanger les morceaux** de différents programmes

Avantages

- simplicité, sécurité
- portabilité (indépendance du hardware et du système d'exploitation)

Désavantages

- les programmes simples sont tout de même assez compliqués
- vitesse d'exécution (de moins en moins problématique)

Le premier programme

Le fichier Hello.java:

```
public class Hello
{
    public static void main(String[] args)
    {
        // display a greeting in the console window
        System.out.println("Hello, World!");
    }
}
```

- **Compilation:** >javac Hello.java
- **Exécution:** >java Hello
- **Sortie:** Hello, World!

La **grammaire** du langage

- sensible à la casse (aux majuscules/minuscules)
- indépendant de la **mise en page**

Éléments syntaxiques:

mots réservés: `public class static void`

identificateurs: `args Hello main String System out println`

littéraux: `"Hello World!"`

ponctuation: `{ } ; [] ()`

commentaires: `// display a greeting in the console window`

La définition d'une classe

L'entête:

```
public class NomDeClasse
```

- `public` = tout le monde peut utiliser la classe
- `class` = unité de base des programmes OO
- une classe par fichier
- `class NomDeClasse` doit être dans le fichier
`NomDeClasse.java`

La définition d'une classe

Le corps:

```
{  
    . . .  
    . . .  
}
```

- contient un ensemble de méthodes, délimité par { }

Conventions

- le nom de classe: NomDeClasse
- l'indentation de { }
- l'indentation de . . .

La définition d'une méthode

L'entête:

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

- machine d'**entrée-sortie**
- `main`: nom de méthode
- `void`: aucune sortie
- `String[] args`: le **paramètre** (entrée)
- `String[]`: le **type** du paramètre
- `args`: le **nom** du paramètre
- **convention**: `nomDeParametre`

La définition d'une méthode

Le corps:

```
{  
    // display a greeting in the console window  
    System.out.println("Hello, World!");  
}
```

- contient une **séquence d'instructions**, délimitée par { }
- // display a greeting in the console window:
commentaire
- System.out.println("Hello, World!"): **appel** de méthode
- les instructions sont terminées par ;

L'appel d'une méthode

En général:

```
nomDObjet.nomDeMethode(<liste des paramètres>)
```

- `System.out`: l'objet qui représente le terminal (l'écran)
- `println`: la méthode qui imprime son paramètre (+ une fin de ligne) sur un stream
- `"Hello, World!"`: le paramètre de `println`

La méthode `main`

- `"java Hello"` exécute la méthode `main` dans la classe `Hello`