Algo / Git

Algorithmie

Sommaire

- Tableaux
- Fonctions
- Passage de paramètre par copie
- Complexité (Big-O)
- Récursivité
- Debug / Raccourcis IntelliJ

Le cours + projet seront faits en Java

Pourquoi Java ?

Pourquoi Java ?

- Langage fortement typé
  - Chaque variable doit explicitement déclarer son type
  - Chaque paramètre et retour de fonction doit explicitement déclarer son type
- C'est moins "fun" qu'un JS / Python, mais c'est pour vous donner des bases plus solides

Pourquoi Java ?

- C'est + facile de passer d'un langage typé à un langage dynamique que l'inverse
- Très utilisé dans le monde professionnel
  - C#
  - TypeScript
  - Kotlin
  - Rust
  - C, C++, etc

Tous les TPs sont à enregistrer dans un repo git !

A chaque TPs, commit + push

Si j'oublie, rappelez le moi !

Tableaux

Rappel des types primitifs

Type	Exemple	Taille (bits)	Description
`boolean`	`boolean b = true;`	Variable	Stocker une valeur vraie ou fasse
`byte`	`byte b = 10;`	8	Stocker une nombre <= 255
`int`	`int i = 42;`	32	Stocker un nombre entier
`long`	`long l = 42L;`	64	Stocker un (grand) nombre entier

Rappel des types primitifs

Type	Exemple	Taille (bits)	Description
`float`	`float f = 3.12f;`	32	Stocker un nombre à virgule
`double`	`double d = 3.12;`	64	Stocker un (grand) nombre à virgule
`char`	`char c = 'c';`	8	Stocker un caractère unique
`String`	`String s = "abcd";`	Variable	Stocker une chaîne de caractères

En déclarant une variable d'un type

Vous réservez une case mémoire pour cette variable

Vous pouvez déclarer un tableau de cases mémoires contigües

(qui se suivent)

Exemple de tableau

```java
// Déclaration d'un tableau de int d'une taille de 5
int[] scores = new int[5];

// Affectation de la valeur 83 à la 1ère case du tableau (index 0)
scores[0] = 83;

// Affectation de la valeur 42 à la dernière case du tableau (index 4)
scores[4] = 42;

// Stockage du contenu de la 1ère case du tableau (index 0) dans un int
int score1 = scores[0];
System.out.println(score1);

// Affichage du contenu de la dernière case du tableau (index 4)
System.out.println(scores[4]);
```

Points importants

- Un tableau a une **taille fixe** définie à la déclaration
- Un tableau peut uniquement contenir des données du type de la déclaration (int, bool, char, String, ...)
- Un tableau commence à l'index `0` et se termine à `taille - 1`
- Vous pouvez à tout moment récupérer ou affecter une valeur à une case donnée

Parcourir un tableau

```java
// Déclaration d'un tableau de taille 3 avec les éléments déjà initialisés
String[] names = {"Bob", "Alice", "Robin"};

// Boucle parcourant le tableau (i variant de 0 à 2)
for (int i = 0; i < names.length; i++) {
  // Récupération d'une valeur à une case du tableau (0, 1 puis 2)
  String s = names[i];

  // Bob, Alice, Robin
  System.out.println("name=" + s);
}
```

📣 Info Java 📣

- Le temps de ce cours, nous n'allons **pas** utiliser la POO (Programmation Orientée Objet)
- Java est **fortement** orienté objet, donc nous ferons des choses pas très *classe* (haha, j'ai écrit ça la nuit m'en voulez pas)

📣 Info Java 📣

- Un fichier Java contient toujours une `class` { }
- Tout ce qui est directement dans la `class` sera préfixé par `static`
- A l'intérieur des fonctions, **jamais** de `static`

📣 Info Java 📣

```java
class Coucou {
  // Variable globale STATIC age
  static int age = 42;

  // Fonction STATIC sayHello()
  static void sayHello() {
    // Pas de STATIC ICI
    String name = "Bob";
    System.out.println("Hello " + name);
  }
}
```

TP #01

- On crée le projet ensemble
  - Tout le code directement dans `main() {}`
- Déclarer un tableau de 7 cases pour stocker des scores (ex: 24)
- Stocker une score différent dans chaque case
- Parcourir le tableau et afficher son contenu
- Calculer et afficher la moyenne des scores

Fonctions

Objectif

Réutiliser du code et simplifier la lecture

Rappel

Une fonction à 3 composantes

- 1 type de retour (void, int, boolean, etc)
- 1 nom (getName, isValid, etc)
- 1 ou plusieurs paramètres

Exemples de signatures de fonctions

```java
// Fonction nommée sayHello
// Prend 1 paramètre name de type String
// Ne renvoie rien (void)
static void sayHello(String name) { }

// Fonction nommée getMaximum
// Prend 2 paramètres a et b de type int
// Renvoie un int
static int getMaximum(int a, int b) { }

// Fonction nommée isButtonEnabled
// Ne prend pas de paramètre
// Renvoie un boolean
static boolean isButtonEnabled() { }
```

Point ChatGPT / Copilot

LLMs dans l'ensemble

Disclaimer

Je ne vous empêcherai pas de les utiliser

(Je les utilise moi même, ce serait stupide)

MAIS

Soyez malins,

vous êtes là pour apprendre à CODER

Vous êtes responsable du code que vous livrez en production

DONC

Vous devez développer vos skills de dev

- Pour utiliser ChatGPT le plus efficacement
- Pour savoir quand il hallucine
- Pour l'utiliser quand c'est utile (et quand il fait n'imp)

Vous voulez progresser ?

- Vous devez être capable d'écrire 100% du code sans ChatGPT
- Une fois que vous savez faire ça, ChatGPT est un outil génial pour **gagner du temps**
- Oui on est en cours, oui vous avez une note à la fin, mais... **ON S'EN FOUT**

L'objectif réel

Être les meilleurs devs possible

Parce qu'être compétent

c'est juste + fun

Mes conseils

- Désactiver Copilot quand vous débutez (nouveau langage, framework, etc)
- Coder c'est comme apprendre d'un instrument : ça doit rentrer dans les doigts pour que ça devienne naturel

Mes conseils

- Utilisez ChatGPT comme un prof assistant sous stéroïdes qui ne dort jamais (je suis insomniaque, je fais ce que je peux) :
  - Toujours dispo
  - Donne des compléments d'explications, d'autres exemples que moi
- Pendant une explication ChatGPT vous donne une conclusion différente de la mienne : **dites-le, ça peut être super intéressant**

Exemples de prompts

- "Explique l'algo étape par étape"
- "Voici mon code. Rend plus clair / plus court et explique les changements"
- "Voici mon erreur `<PAVE ROUGE>`, explique moi la cause"

TP #02

- Ecrire une fonction qui prend en paramètre le tableau des scores et affiche son contenu
- Ecrire une fonction qui prend en paramètre le tableau des scores et qui renvoie le plus grand score du tableau
- Ecrire une fonction qui prend en paramètre le tableau des scores et qui renvoie vrai si le tableau contient au moins une valeur inférieure à 10

Documentation de code

JavaDoc

Vous écrivez déjà des commentaires

- `//` pour commenter une ligne
- `/* ... */` pour commenter plusieurs lignes

Il existe une catégorie de commentaires spéciaux

Qui explique comment utiliser vos fonctions

La JavaDoc

(Que vous avez utilisé pendant toute la piscine)

```
/**
 * Calculates the arithmetic mean of a, b, and c.
 * @param a an integer to used in the mean
 * @param b an integer to used in the mean
 * @param c an integer to used in the mean
 * @return the mean value (a + b + c) / 3
 */
int mean(int a, int b, int c) { ... }
```

Résumé de la JavaDoc

- Spécifié par `/**` au dessus de votre fonction
- Commence par un court résumé de ce que fait votre fonction
- Décrit les paramètres (type, valeurs autorisées, etc)
- Décrit le retour avec les valeurs possibles (et quand ces valeurs sont émises)

En documentant ainsi vos fonctions

- Vous aidez à la compréhension de vos fonctions
- Vous pouvez générer un "site web" de JavaDoc
  - Menu Tools > Generate JavaDoc > OK

TP #03

- Documenter toutes les fonctions précédemment écrites
- Générer la JavaDoc et regarder le résultat dans un navigateur

Passage de paramètre par copie

Comment fonctionne la mémoire

Qu'affiche ce programme ?

```
void update(int n) {
  n = 23;
}

public static void main(String[] args) {
  int a = 42;
  update(a);
  System.out.println(a); // 23 ou 42 ?
}
```

42

(La réponse à la vie, l'univers et le reste)

🤔 Pourquoi ? 🤔

Car en Java, les paramètres sont toujours passés par copie

```java
void update(int n) {
  // La variable n contient la valeur de a
  // mais la variable n n'est PAS la variable a
  // C'est une nouvelle variable !
  n = 23;
}

public static void main(String[] args) {
  int a = 42;
  // Ici, on va copier la valeur de a (42)
  // dans la variable n de la fonction update
  update(a);
  System.out.println(a); // 23 ou 42 ?
}
```

Fonctionnement de la mémoire d'un programme

Chaque fois que vous allouez une variable, elle est ajoutée dans une pile

Pile en anglais = Stack

Fonctionne exactement comme une pile d'assiettes

Quand vous entrez dans une fonction,

Une stack-frame est créée

Quand vous quittez une fonction,

toutes les variables de la stack-frame sont dépilées

Stack-Frame

```java
int findMax(int[] tab) {
  // Cette fonction déclare au total 3 variables (tab, max, i)
  // Ces 3 variables font partie de la stack-frame de findMax()
  int max = -1000000;
  for(int i = 0; i < tab.length; i++) {
    if (tab[i] < max) {
      max = tab[i];
    }
  }
}

public static void main(String[]args) {
  int a = 10;
  int[] t = new int[2];
  t[0] = a;
  t[1] = 43;
  // Ici on déclare 3 variables qui font partie de la stack frame de main()
  int res = findMax(t);
}
```

Complexité

En informatique, on veut savoir combien coûte l'exécution d'un algorithme

Le temps de traitement est une information importante

Mais non suffisante

Imaginons que vous fassiez un algorithme pour trouver une valeur dans un tableau trié

La méthode directe

On regarde chaque case du tableau

Avec 10 éléments, il met 0.3 secondes

Avec 1000 éléments, il met 3 secondes

Avec 100 000 éléments, il met 300 secondes

On voit que le temps de traitement est proportionnel au nombre d'éléments

On dit que l'algorithme suit une courbe linéaire

Ce temps de traitement va changer en fonction de votre CPU, la charge au moment de l'exécution, etc...

Donc, on va utiliser un autre indicateur pour calculer le coût de l'algorithme

La complexité

Attention, le mot est trompeur

TLDR;

Si je donne une entrée à mon algo de taille N, quel est l'ordre de grandeur, en fonction de N, du nombre d'opérations qu'il fera ?

En français

Quand j'augmente la taille mon entrée, mon nombre d'opération augmente comment ?

Quand on fait une recherche dans un tableau

L'opération est : combien de comparaisons je fais ?

La convention en anglais

Big-O Notation

(ça vient des maths)

Avec une recherche linéaire

- **Meilleur cas O(1)** => C'est le 1er, temps constant => 1 comparaison
- **Pire cas O(n)** => C'est le dernier => j'ai n cases => n comparaisons
- **Cas moyen O(n/2)** => En moyenne je ferais n/2 comparaisons

La complexité d'une recherche linéaire est donc

O(n/2)

Maintenant un(e) collègue propose une autre façon de chercher

(Une recherche binaire que vous allez implémenter)

Avec 10 éléments, il met 0.9 secondes

Avec 1000 éléments, il met 2 secondes

Avec 100 000 éléments, il met 5 secondes

Au doigt mouillé

On dirait que le temps augmente de moins en moins

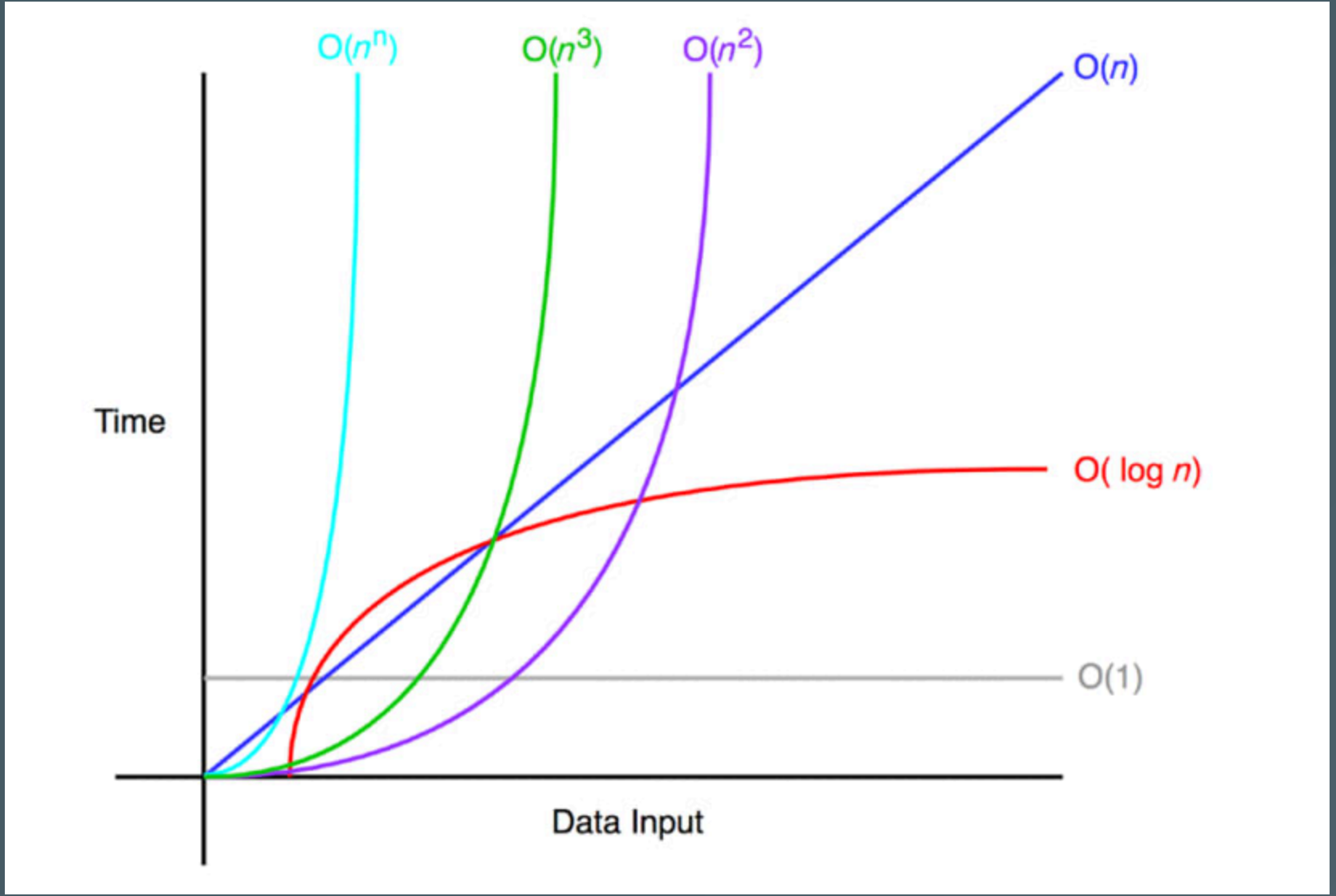
Avec une recherche binaire

- **Meilleur cas O(1)** => C'est le 1er, temps constant => 1 comparaison
- **Pire cas O(n)** => C'est le dernier => j'ai n cases => n comparaisons
- **Cas moyen O(log n)** => En moyenne je ferais log n comparaisons

Complexité recherche binaire

O(log n)

Graphe des complexités



TP #04

- Implémenter une fonction de recherche binaire
- Cette fonction prend en paramètre
  - Un tableau d'entiers trié par ordre croissant
  - Un nombre entier à rechercher
- Cette fonction renvoie l'index de la case contenant le nombre
- Si la valeur n'est pas trouvé, elle renvoie -1
- Cette fonction est commentée !

Productivité sur IntelliJ

(Je fais ça ici pour soulager les cerveaux)

Mode Debug

Vous pouvez exécuter le programme pas à pas

Mode Debug

1. Mettre un point d'arrêt (dans la gouttière à coté du numéro de ligne)
2. Exécuter l'application en mode debug (insecte vert dans la toolbar)
3. Kiffer

Raccourcis pratiques

Raccourci	Nom	Description
Shift + F6	Rename	Renommer un symbole (variable, fonction) ainsi que toutes ses occurrences
Ctrl + Shift + /	Toggle comment	Commenter / Décommenter toutes les lignes sélectionnées (⚠️ custom Robin)
⌘ + B	Aller sur le symbole	Aller a la définition de la fonction, de la variable

Raccourcis pratiques

Raccourci	Nom	Description
⌥ + ⌘ + L	Formatter le fichier	Remttre l'indentation au propre, les espaces, etc
Shift Shift	Chercher partout	Chercher un texte dans tout le projet
⇧ + ⌘ + A	Chercher une action	A partir d'un nom, voir si une action (raccourci) existe

Récursivité

Jusque là vous avez écrit des fonctions qui ont des boucles

Mais il existe des fonctions qui fonctionnent...

différemment

Une fonction qui s'appelle elle même !

Imaginons une fonction qui affiche un compte-à-rebours

Compte-à-rebours boucle

```
static void countDownLoop(int start) {
    for (int i = start; i > 0; i--) {
        System.out.println(i + "...");
    }
    System.out.println("FINISHED");
}

public static void main(String[] args) {
    countDownLoop(3); // 3, 2, 1, FINISHED
}
```

Et si on voulait la même chose...

Sans la boucle for ?

Compte-à-rebours récursif

```
static void countDownRecursive(int start) {
    if (start > 0) {
        System.out.println(start + "...");
        countDownRecursive(start - 1);
    } else {
        System.out.println("FINISHED");
    }
}

public static void main(String[] args) {
    countDownRecursive(3); // 3, 2, 1, FINISHED
}
```

Mais pourquoi s'infliger ça ?

Pour certaines catégories de problèmes

Il est beaucoup plus facile de le faire en récursif

Dès que vous parcourez des arbres

Fichiers, structures, etc

Exemple pour compter tous les fichiers d'un dossier

```
public static int countFiles(File f) {
    if (f.isFile()) {
        return 1;
    }

    int count = 0;
    File[] files = f.listFiles();
    for (int i = 0; i < files.length; i++) {
        File fileOrDir = files[i];
        count += countFiles(fileOrDir);
    }
    return count;
}
```

Attention, en récursivité, si vous oubliez la condition de sortie...


PROBLEMS WITH RECURSION ?
↓Please take one↓
WITH
ECURSIO
ease take or
WHAT THE-
PROBLEMS WITH RECURSION ?
↓Please take one↓
smbc-comics.com

TP #05

- Implémenter les 2 versions du compte-à-rebours
  - Supprimer la condition if dans la version récursive (en gardant le traitement)
  - Que se passe-t-il ?
- Implémenter la recherche binaire en version récursive