



LSINF1121

Algorithmique et Structure de données
Complexité, Type-Abstrait de données, Structure Chainées

Pierre Schaus
@pschaus

Plan

- Motivation
- Organisation du cours
- Rappel Java
- Type Abstrait de données
- Complexité
- Les outils
 - Java util collection
 - IDE IntelliJ
 - Junit

Pourquoi étudier les algorithmes?

- Car c'est le fondement de l'informatique
 - Le cours LSINF1121 est un pré-requis pour beaucoup d'autres cours.
 - * Savez vous que l'algorithme Dijkstra est utilisé dans chaque router d'internet ?
- La place des algorithmes dans notre société est grandissante. On parle de « société algorithmique »
- Si vous ne maîtrisez pas les bases, vous ne passerez pas les interviews chez les GAFA



Google 1ère interview = Algorithme

Phone/Hangout interviews

During phone or Google Hangout interviews, you'll speak with a potential peer or manager. For software engineering roles, your phone/Hangout discussion will last between 30 and 60 minutes.

When answering coding questions, you'll talk through your thought process while writing code in a Google Doc that you'll share with your interviewer.

We recommend using a hands-free headset or speakerphone so you can type freely.

Your phone interview will cover **data structures and algorithms**.

Be prepared to **write around 20-30 lines of code in your strongest language**. Approach all scripting as a coding exercise — this should be clean, rich, robust code:

1. You will be asked an open ended question. Ask clarifying questions, devise requirements.
2. You will be asked to explain it in an **algorithm**.
3. Convert it to workable code. (Hint: Don't worry about getting it perfect because time is limited. Write what comes but then refine it later. Also make sure you consider corner cases and edge cases, production ready.)
4. Optimize the code, follow it with test cases and find any bugs.

For all other roles, your phone/Hangout discussion will last between 30 and 45 minutes. Be prepared for behavioral, hypothetical, or case-based questions that cover your role-related knowledge.

Type Abstrait de donnée

- Dans ce cours nous manipulons des données.
- Les données peuvent être ajoutées et traitées via une API (interface ou ensemble de méthodes + constructeur)
- L'utilisateur n'est pas obligé de connaître la représentation concrète du stockage des données (il manipule une boîte noire).
- Le but de ce cours est étudier l'implémentation des boîtes noires car toutes les implémentations ne se valent pas.



```
public static void main(String[] args) {  
    Stack<Integer> stack = null;  
    stack.push(2);  
    stack.push(3);  
    stack.pop();  
    stack.isEmpty()  
}
```

StackImpl

API

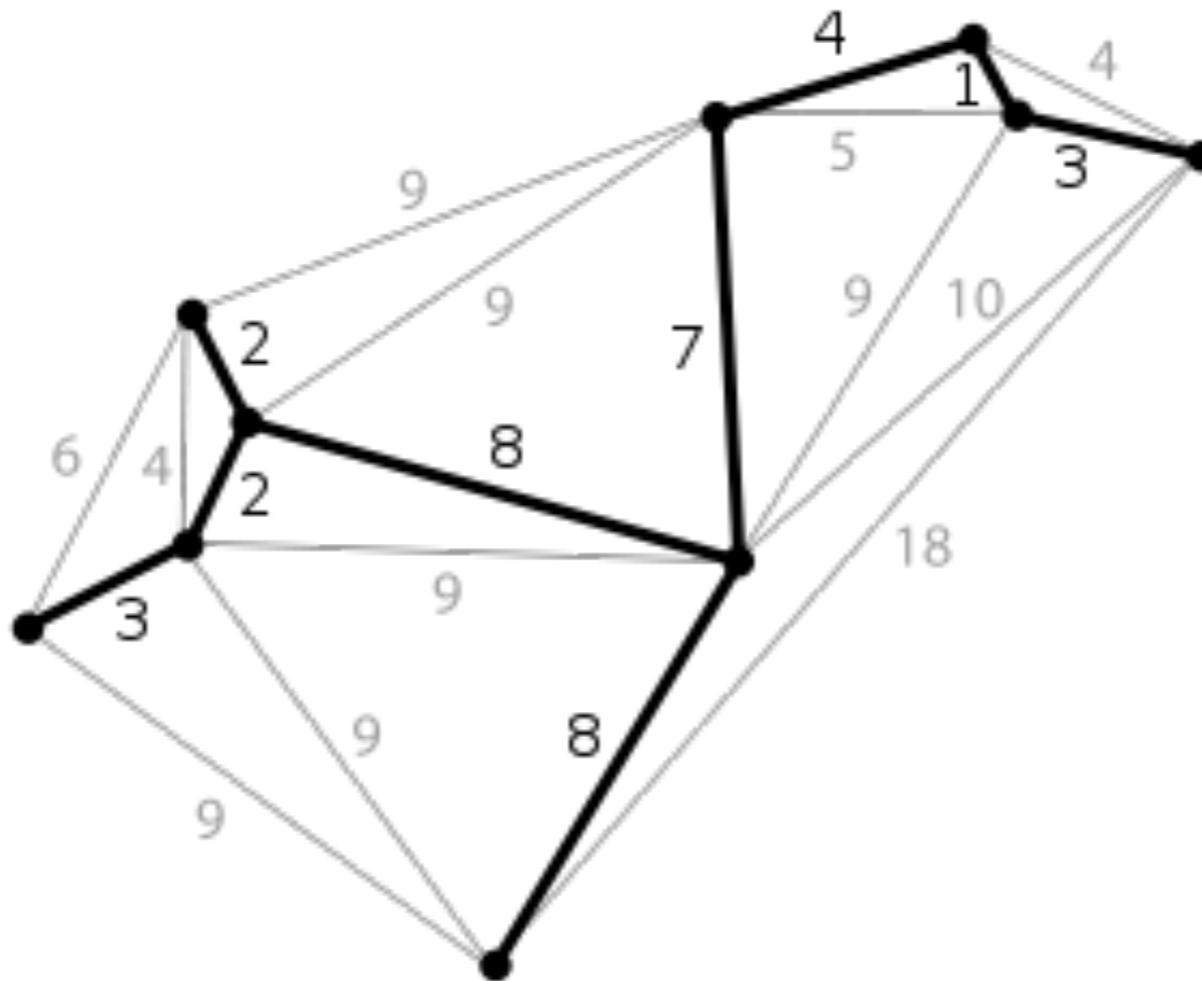
```
public interface Stack<Item> {  
    void push(Item item);  
    Item pop();  
    boolean isEmpty();  
    int size();  
}
```

Dans ce cours nous verrons

- Des types abstrait de données pour:
 - Insérer des objets ordonnés dans un ordre arbitraire et pouvoir rapidement retrouver le maximum ou retirer le maximum
 - Insérer/retirer des objets dans un ordre arbitraire sur base d'une clef ordonnée et ensuite pouvoir itérer rapidement dans l'ordre, retrouver le minimum
 - Représenter des réseaux: ajouter des noeuds, retrouver des noeuds voisins à un noeud, etc.
 - Insérer et retirer des objets dans un ordre arbitraire sur base d'une clef non nécessairement ordonnée
- Des algorithmes pour
 - Trier des données
 - Compresser des données textuelles
 - Retrouver des sous chaînes de caractères efficacement dans une grande chaîne de caractère
 - Calculer des chemins ou des arbres dans des réseaux

Exemple

- Trouver dans ce réseau (Graphe) un sous graphe qui a le plus petit poids possible et qui touche tous les noeuds



- Application: connecter des maisons à moindre coût avec de la fibre optique

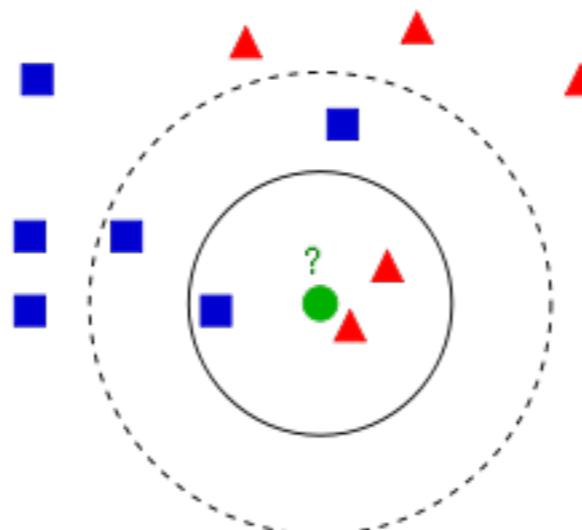
Exemple

- Moteur de recherche:
 - ▶ Vous souhaitez pouvoir stocker des millions de mots clefs et pour chacun une liste d'URLs avec les pages correspondantes.
 - ▶ Comment faire pour être capables de retourner rapidement une URL pour un mot clef ?



Exemple

- Est-ce que le nouveau point est un point bleu ou rouge ?
- On peut compter les « k » plus proche et attribuer la classe majoritaire ?
- Comment faire pour compter efficacement les « k » plus proches ?



java.util.collection

- <https://docs.oracle.com/javase/8/docs/api/index.html?java/util/Collection.html>

The screenshot shows the Java API documentation for the `java.util` package. The left sidebar lists various sub-packages and categories like `Interfaces` and `Classes`. The main content area has tabs for `Overview`, `Package` (which is selected), `Class`, `Use`, `Tree`, `Deprecated`, `Index`, and `Help`. Below these tabs, there are links for `Prev Package`, `Next Package`, `Frames`, and `No Frames`. The `Standard Ed. 7` is noted at the top right.

Package `java.util`

Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).

See: Description

Interface Summary

| Interface | Description |
|--------------------------------------|---|
| <code>Collection<E></code> | The root interface in the <i>collection hierarchy</i> . |
| <code>Comparator<T></code> | A comparison function, which imposes a <i>total ordering</i> on some collection of objects. |
| <code>Deque<E></code> | A linear collection that supports element insertion and removal at both ends. |
| <code>Enumeration<E></code> | An object that implements the <code>Enumeration</code> interface generates a series of elements, one at a time. |
| <code>EventListener</code> | A tagging interface that all event listener interfaces must extend. |
| <code>Formattable</code> | The <code>Formattable</code> interface must be implemented by any class that needs to perform custom formatting using the ' <code>s</code> ' conversion specifier of <code>Formatter</code> . |
| <code>Iterator<E></code> | An iterator over a collection. |
| <code>List<E></code> | An ordered collection (also known as a <i>sequence</i>). |
| <code>ListIterator<E></code> | An iterator for lists that allows the programmer to traverse the list in either direction, modify the list during iteration, and obtain the iterator's current position in the list. |
| <code>Map<K,V></code> | An object that maps keys to values. |
| <code>Map.Entry<K,V></code> | A map entry (key-value pair). |
| <code>NavigableMap<K,V></code> | A <code>SortedMap</code> extended with navigation methods returning the closest matches for given search targets. |
| <code>NavigableSet<E></code> | A <code>SortedSet</code> extended with navigation methods reporting closest matches for given search targets. |
| <code>Observer</code> | A class can implement the <code>observer</code> interface when it wants to be informed of changes in observable objects. |
| <code>Queue<E></code> | A collection designed for holding elements prior to processing. |
| <code>RandomAccess</code> | Marker interface used by <code>List</code> implementations to indicate that they support fast (generally constant time) random access. |
| <code>Set<E></code> | A collection that contains no duplicate elements. |
| <code>SortedMap<K,V></code> | A <code>Map</code> that further provides a <i>total ordering</i> on its keys. |
| <code>SortedSet<E></code> | A <code>Set</code> that further provides a <i>total ordering</i> on its elements. |

Class Summary

| Class | Description |
|--|--|
| <code>AbstractCollection<E></code> | This class provides a skeletal implementation of the <code>Collection</code> interface, to minimize the effort required to implement this interface. |
| <code>AbstractList<E></code> | This class provides a skeletal implementation of the <code>List</code> interface to minimize the effort required to implement this interface backed by a "random access" data store (such as an array). |
| <code>AbstractMap<K,V></code> | This class provides a skeletal implementation of the <code>Map</code> interface, to minimize the effort required to implement this interface. |
| <code>AbstractMap.SimpleEntry<K,V></code> | An Entry maintaining a key and a value. |
| <code>AbstractMap.SimpleImmutableEntry<K,V></code> | An Entry maintaining an immutable key and value. |
| <code>AbstractQueue<E></code> | This class provides skeletal implementations of some <code>Queue</code> operations. |
| <code>AbstractSequentialList<E></code> | This class provides a skeletal implementation of the <code>List</code> interface to minimize the effort required to implement this interface backed by a "sequential access" data store (such as a linked list). |
| <code>AbstractSet<E></code> | This class provides a skeletal implementation of the <code>Set</code> interface to minimize the effort required to implement this interface. |
| <code>ArrayDeque<E></code> | Resizable-array implementation of the <code>Deque</code> interface. |
| <code>ArrayList<E></code> | Resizable-array implementation of the <code>List</code> interface. |

- Un package de java qui contient la plupart des types abstrait de données. Ce package n'aura plus de mystère pour vous à la fin de ce cours.

Pédagogie

- Vous êtes au centre de votre apprentissage
- Nous ne sommes là que pour vous guider et vous proposer des activités pour acquérir la matière



Livre de référence



- 70 euros sur Amazon (paper), 26 Kindle

Le site du cours

- <https://moodleucl.uclouvain.be/course/view.php?id=7682>
pour les annonces
- <http://lsinf1121.readthedocs.io> pour le reste

Documentation LSINF1121 2018-2019 »



Sujet suivant

Organisation

Cette page

Montrer le code source

Recherche rapide

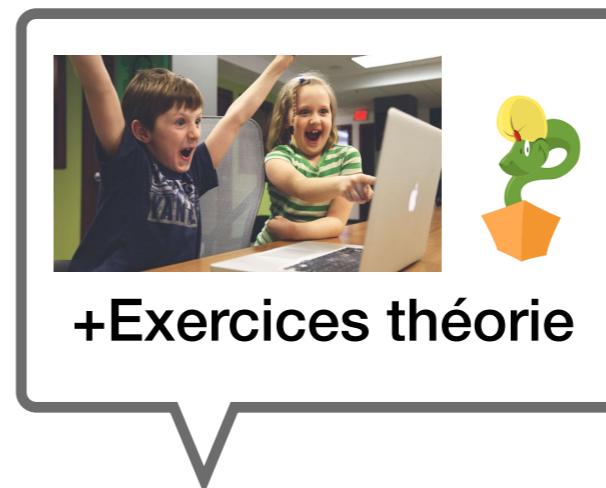
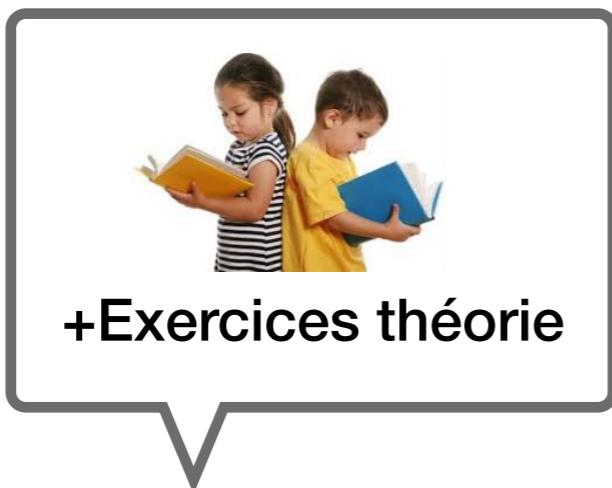
Go

LSINF1121: Algorithmique et Structures de données

- [Organisation](#)
 - [Pédagogie](#)
 - [Agenda](#)
 - [Evaluation](#)
 - [Contact et communication](#)
- [Partie 1 | Types abstraits de données, Complexité, Collections Java; Piles, files et listes liées](#)
 - [Objectifs](#)
 - [A lire](#)
 - [Exercices théoriques: première partie](#)
 - [Exercices théoriques supplémentaires](#)
 - [Exercices d'implémentation sur Inginous](#)
 - [Exercices théorique: deuxième partie](#)
 - [Ressources supplémentaires](#)
- [Partie 2 | Tri et propriétés des ensembles triés](#)
- [Partie 3 | Arbres](#)
- [Partie 4 | Dictionnaires: tables de hashages et autres implémentations](#)
- [Partie 5 | Files de priorités, union-find et compression de données](#)
- [Partie 6 | Graphes: parcours, arbres sous-tendants, et plus courts chemins](#)

Documentation LSINF1121 2018-2019 »

Organisation



Introduction
et motivation
P. Schaus

Permanence
Candix

Correction
(partielle) des
exercices
G.Derval

Permanence
Candix

Restructuration
P. Schaus

Evaluation

- Examen
 - ▶ 2 (exercices implémentation) x 8 points
 - ▶ 4 points de théorie et exercices manuels
- Mid term test
 - ▶ 1 (exercices implémentation) x 8 points
 - ▶ 2 points de théorie et/ou exercices manuels
- Mid term test ne compte dans la note finale que pour 2 points et uniquement s'il fait remonter la note (no-stress).



Exercices d'implementation à l'examen

- 50% de la note si l'algorithme est correct
- 50% de la note si l'algorithme est correct et a la bonne complexité
- Exemple: Ecrire une méthode pour trier un tableau en $O(n \cdot \log(n))$.
 - Réponse: code correct mais en $O(n^2)$ => 4/8
 - Réponse: code buggy en $O(n \cdot \log(n))$ => 0/8

Pourquoi Java ?

- Java est le langage le plus populaire
- Crée en 1995 chez Sun mais maintenant maintenu par Oracle depuis 2009.
- Il est très rapide et portable: il s'exécute sur une JVM
- Dans ce cours on utilisera la version 8 de Java (dernier changement sur le langage).
- Nous supposons que vous êtes familier avec Java, sinon lisez section 1.1. du livre.
 - variable d'instance, tableau, types primitifs

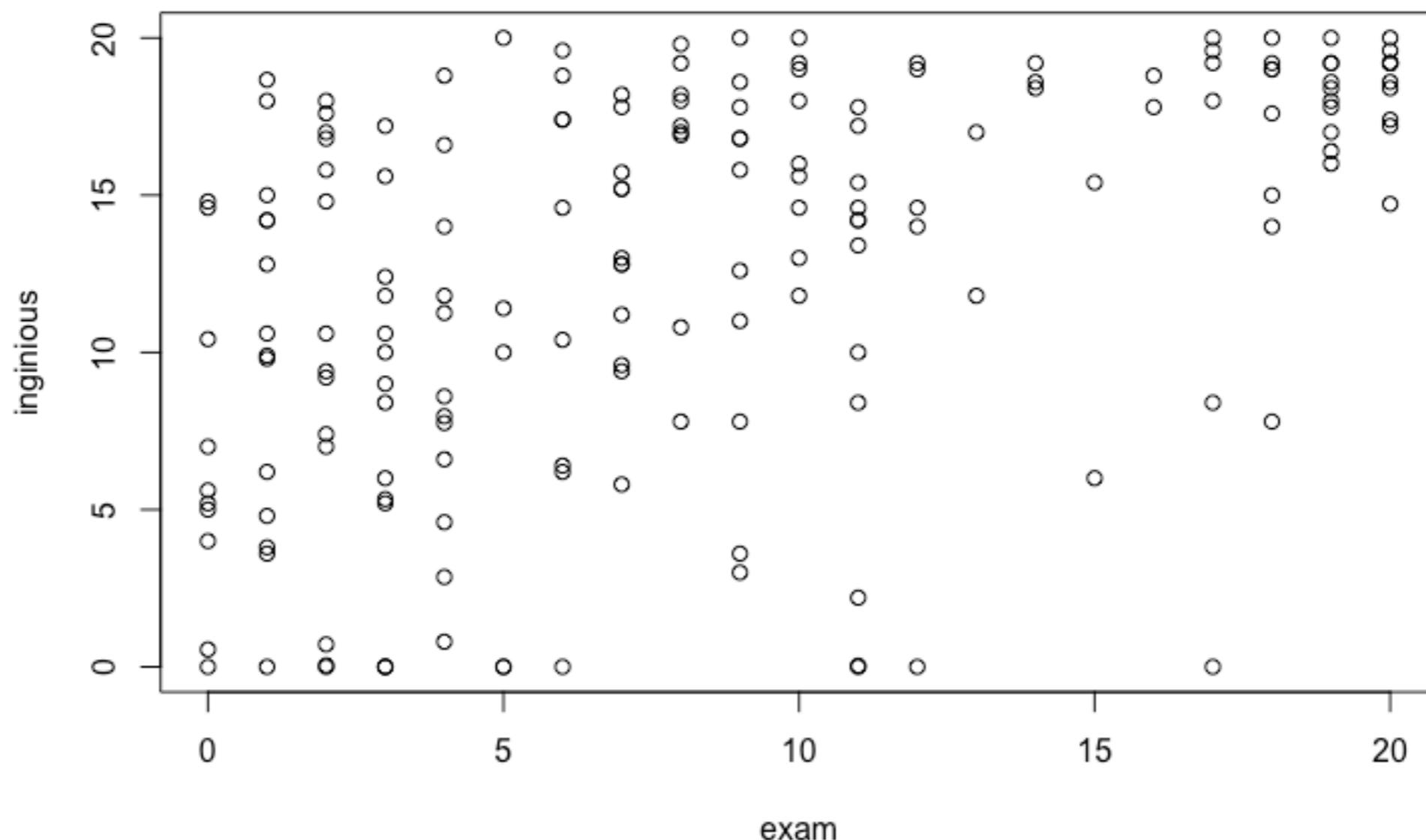


Pourquoi pas python

- Car Python est un langage haut (trop) niveau pour étudier les algorithmes. Il cache beaucoup de détail d'implémentation.
 - Exemple: les Arrays en python sont dynamiques
- Autre raison: python est lent, les structures de données et algorithmes built-in de python sont généralement implémentées en C: python list implementation <https://github.com/python/cpython/blob/master/Objects/listobject.c> (enjoy!)
- En java, toutes les collections disponibles sont implémentés en Java et peuvent être étudiées facilement. A la fin de ce cours, l'implémentation de java.util.collection n'aura plus de secrets pour vous.

Inginious, Inginious, Inginious

- Probabilité réussir examen si pas réussi inginious 20%
- Probabilité réussir l'examen si réussi inginious 50%



Rappel Objets

```
class Car {  
  
    String color;  
    int speed = 0;  
    int acceleration = 0;  
  
    Car(String color, int acceleration) {  
        this.color = color;  
        this.acceleration = acceleration;  
    }  
  
    void speedUp() {  
        speed += acceleration;  
    }  
    void brake() {  
        if (speed >= acceleration)  
            speed = speed - acceleration;  
    }  
  
    public static void main(String[] args) {  
        Car myPorshe = new Car("black",10);  
        myPorshe.speedUp();  
        Car myFerrari = new Car("red",20);  
        myFerrari.speedUp();  
        myFerrari.speedUp(); // speed is now 20  
    }  
}
```

Encapsulated instance variables

Constructor

Methods



In Java, everything is a reference*

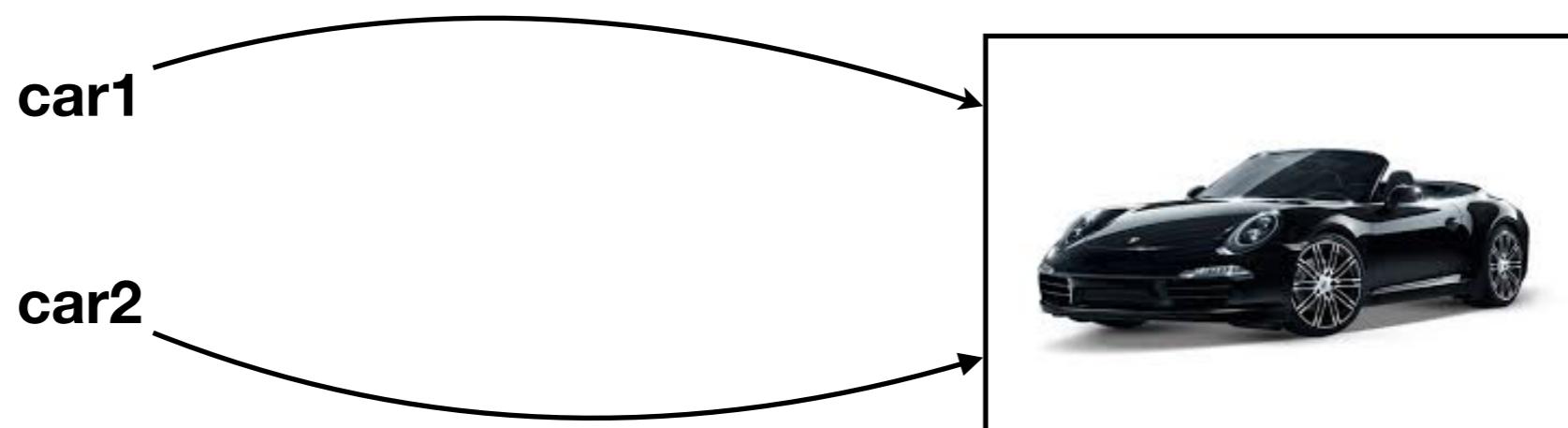
- Definition: A **reference** is a variable that refers to something else and can be used as an alias for that something else.
- No pointer arithmetic possible with reference like with C/C++ pointers. The access is safe-guarded by Java. A pointer is a reference but not the opposite. A reference is a limited pointer.
- * except primitive types: int, boolean, char, etc.

More on references

```
public static void main(String[] args) {  
    Car car1 = new Car("black",5);  
    Car car2 = car1;  
}
```

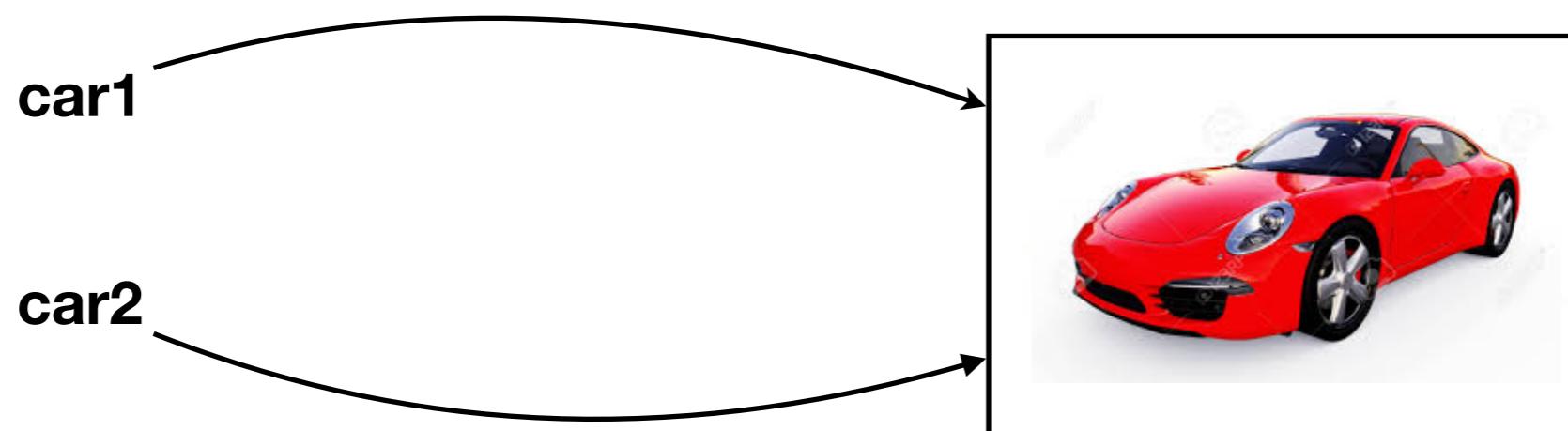
car1 is a reference to an actual
Car object

The “=” copies the reference and assign it to
car2. Thus car2 is a reference to the same
object as car1 but there is no object creation
here



More on references

```
public static void main(String[] args) {  
    Car car1 = new Car("black",5);  
    Car car2 = car1;  
    car2.color = "red";  
}
```



Object vs Primitive types

- Primitive types in Java:
 - boolean – 1 bit
 - byte – 8 bits
 - short, char – 16 bits
 - int, float – 32 bits
 - long, double – 64 bits
- Java also has the object representation of Primitive types:
 - Boolean – 128 bits
 - Byte – 128 bits
 - Short, Character – 128 bits
 - Integer, Float – 128 bits
 - Long, Double – 192 bits
- autoboxing and unboxing = From primitive types to objects and vice versa

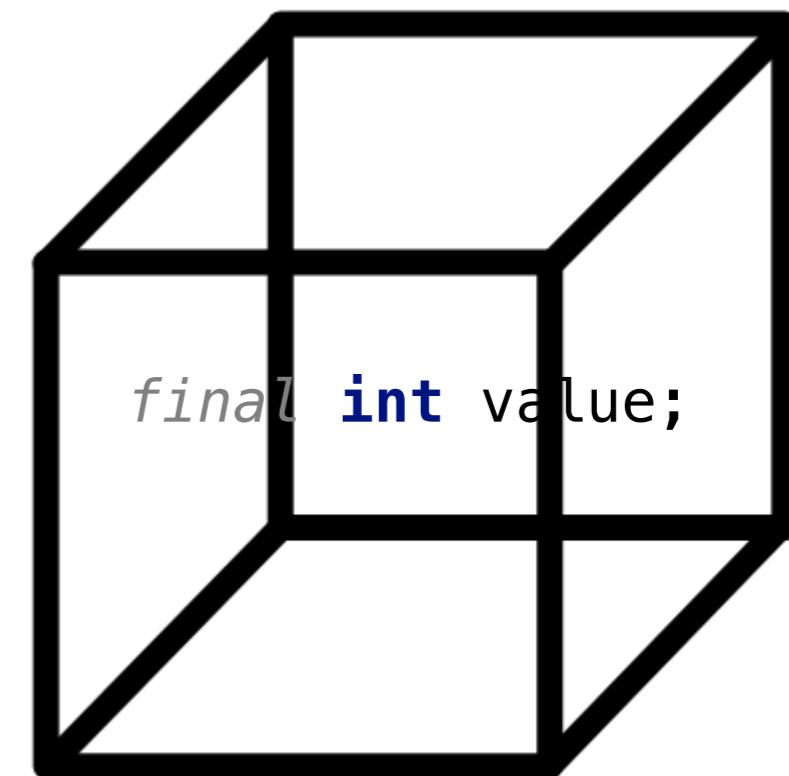
Auto and out boxing

Every object contains a single value of the corresponding primitive type. The **wrapper classes are immutable** (so that their state can't change once the object is constructed) and are final (so that we can't inherit from them).

Under the hood, Java performs a conversion between the primitive and reference types if an actual type is different from the declared one. This conversion is called **auto-boxing** and **out-boxing**

```
public static void main(String[] args) {  
    Integer n1 = 3; // auto-boxing  
    int n2 = n1; // out-boxing  
    foo(3); // auto-boxing  
}  
  
public static int foo(int n1) {  
    return n1+1;  
}
```

The integer class is just
a wrapper around an
immutable int field



Exercise

- What is printed ?

```
public static void main(String[] args) {  
    Integer n1 = 3;  
    Integer n2 = n1;  
    n2 = 4;  
    System.out.println(n1);  
}
```

Parameters are passed by value

- Java manipulate objects by reference but arguments are not passed by reference but **by value**.
- What is printed here ?

```
public static void main(String[] args) {  
    int n1 = 1;  
    int n2 = 2;  
    swap(n1,n2);  
    System.out.println(n1+" "+n2);  
}  
  
public static void swap(int n1, int n2) {  
    int tmp = n1;  
    n1 = n2;  
    n2 = tmp;  
}
```

By value means that
the value is copied

By value for Objects ?

- Java manipulate objects by reference but arguments are not passed by reference but **by value**.
- What is printed here ?

```
public static void main(String[] args) {  
    int n1 = 1;  
    int n2 = 2;  
    swap(n1,n2);  
    System.out.println(n1+" "+n2);  
}
```

```
public static void swap(int n1, int n2) {  
    int tmp = n1;  
    n1 = n2;  
    n2 = tmp;  
}
```

By value means that the values n1 and n2 are copied (fresh variables).

By value for object (references)?

```
public static void main(String[] args) {  
    Car myPorshe;  
    myPorshe = new Car("black",5);  
  
    tuneIntoAFerrari(myPorshe);  
  
    // myPorshe is still a reference to the black  
    System.out.println(myPorshe);  
}
```

```
public static void tuneIntoAFerrari(Car c) {  
    c = new Car("red",10); // c is now a reference to a new "red" car  
}
```

myPorshe

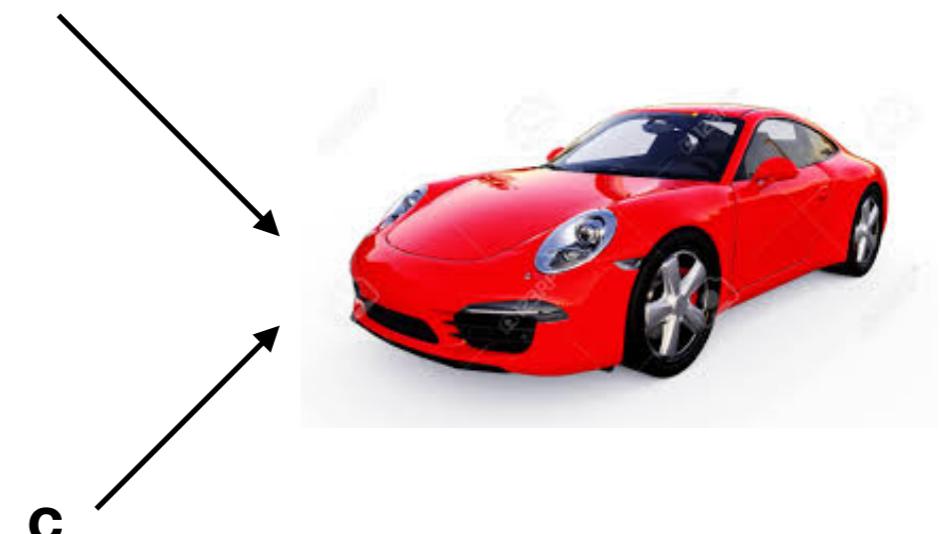
c



By value for Objects ?

```
public static void main(String[] args) {  
    Car myPorshe = new Car("black",5);  
}  
  
public static void tuneIntoAFerrari(Car c) {  
    c.color = "red";  
    c.acceleration = 10;  
}
```

myPorshe



The null reference

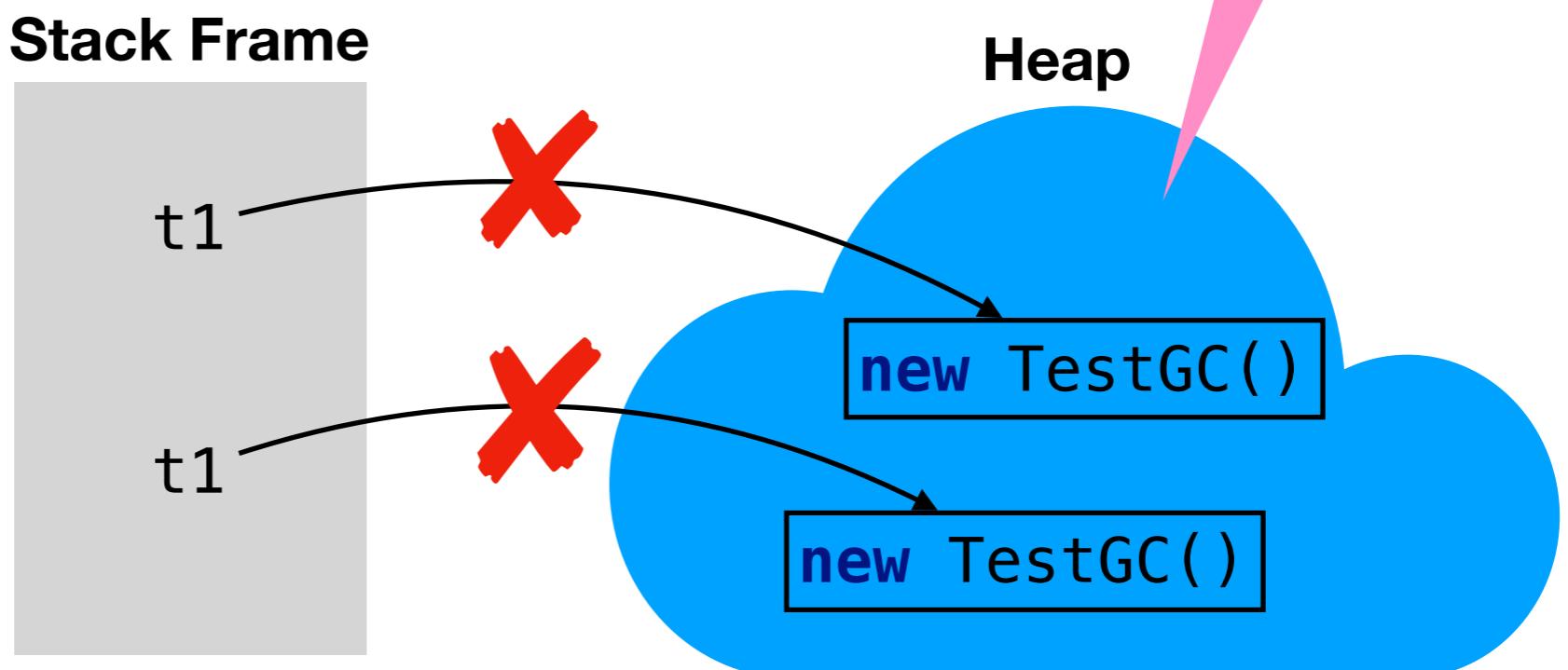
- A uninitialised reference is ***null***

```
public static void main(String[] args) {  
    Car myPorshe;  
    System.out.println(myPorshe);  
}
```

Garbage Collection Revisited

```
public class TestGC {  
    public static void main(String[] args) {  
        TestGC t1 = new TestGC();  
        TestGC t2 = new TestGC();  
  
        // Nullifying the reference variable  
        t1 = null;  
  
        // Nullifying the reference variable  
        t2 = null;  
    }  
}
```

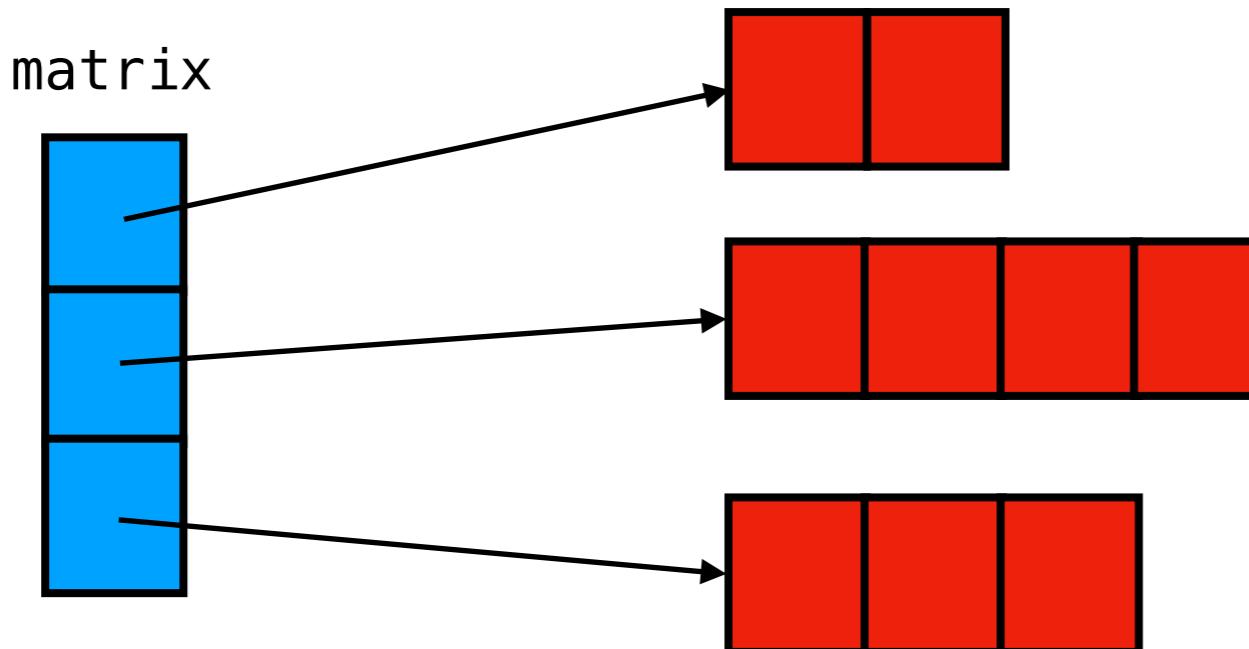
The two instances are not accessible any more, they can be GC



Java Arrays of Arrays (Rappel)

```
public class ArrayDemo {  
    public static void main(String[] args) {  
        int[][] matrix; // declaration of an array of array of int's  
        // declare an array of three references to int [] arrays  
        matrix = new int[3][];  
        // initialize each array individually  
        matrix[0] = new int [] { 1, 0};  
        matrix[1] = new int []{ 1, 0, 12, -1};  
        matrix[2] = new int []{-5, -2, 2};  
    }  
}
```

Pas nécessairement la même taille



Iterating over an array (with indices)

```
int[][] matrix; // declaration

matrix = new int[][]{ { 1, 0, 12, -1},
                     { 7, -3, 2, 5},
                     {-5, -2, 2, -9}
};

matrix = new int[3][];
matrix[0] = new int [] { 1, 0, 12, -1};
matrix[1] = new int []{ 1, 0, 12, -1};
matrix[2] = new int []{-5, -2, 2, -9};

for (int i = 0; i < matrix[0].length; i++) {
    for (int j = 0; j < matrix[i].length; j++) {
        System.out.println("entry " + i + "," + j + "=" + matrix[i][j]);
    }
}
```

Iterating on values

```
int[][] matrix; // declaration

matrix = new int[][]{{1, 0, 12, -1},
                     {7, -3, 2, 5},
                     {-5, -2, 2, -9}}
};

matrix = new int[3][];
// initialize each array
matrix[0] = new int[]{1, 0, 12, -1};
matrix[1] = new int[]{1, 0, 12, -1};
matrix[2] = new int[]{-5, -2, 2, -9};

for (int[] line : matrix) {
    for (int v : line) {
        System.out.println(v);
    }
}
```

Java Exercice

```
public class Main {  
  
    public static void main(String[] args) {  
        int [] a = new int[10];  
        foo(a);  
        System.out.println(Arrays.toString(a));  
    }  
  
    public static void foo(int [] a) {  
        a[0] = 2;  
    }  
}
```



Refresh
My Memory

- L'output de ce programme est ?
 - ▶ [2, 0, 0, 0, 0, 0, 0, 0, 0, 0]
 - ▶ [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
 - ▶ NullPointerException

Java Exercice

```
public class Main {  
  
    public static void main(String[] args) {  
        int a = 7;  
        foo(a);  
        System.out.println(a);  
    }  
  
    public static void foo(int a) {  
        a = 6;  
    }  
}
```



Refresh
My Memory

- L'output de ce programme est ?
 - ▶ 6
 - ▶ 7
 - ▶ autre

Java Exercice



Refresh
My Memory

```
public class Main {  
  
    public static void main(String[] args) {  
        String [] a = new String[5];  
        System.out.println(a[0].isEmpty());  
    }  
}
```

- L'output de ce programme est ?
 - ▶ 0
 - ▶ « »
 - ▶ NullPointerException

For (logical equality testing) use equals

```
public static void main(String[] args) {  
  
    Integer i1 = 2;  
    Integer i2 = 2;  
  
    System.out.println(i1.equals(i2)); // true (because of caching)  
  
    i1 = 160;  
    i2 = 160;  
  
    System.out.println(i1.equals(i2)); // false  
  
    String s1 = "LSINF1102";  
    String s2 = "LSINF1102";  
  
    System.out.println(s1.equals(s2)); // true because of string constant  
    "interning"  
  
    String s3 = s1+"";  
  
    System.out.println(s1.equals(s3)); // false  
  
    Car c1 = new Car("black",5);  
    Car c2 = new Car("black",5);  
  
    System.out.println(c1.equals(c2)); // false (by default same behaviour as ==  
}  

```

You can redefine it

User Defined Equals

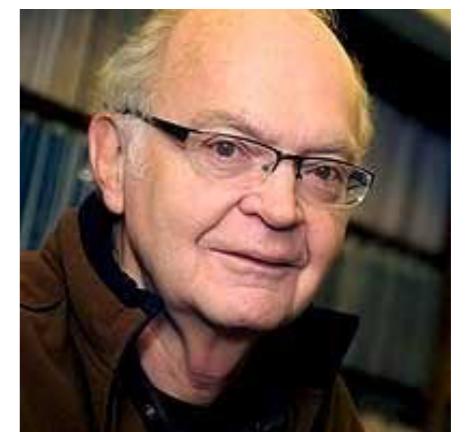
```
class Car {  
  
    String color;  
    int speed = 0;  
    int acceleration = 0;  
  
    Car(String color, int acceleration) {  
        this.color = color;  
        this.acceleration = acceleration;  
    }  
  
    @Override  
    public boolean equals(Object obj) {  
        if (obj instanceof Car) {  
            Car c1 = (Car) obj;  
            return color.equals(c1.color) && speed == c1.speed;  
        }  
        return false;  
    }  
  
    public static void main(String[] args) {  
  
        Car c1 = new Car("black",5);  
        Car c2 = new Car("black",5);  
  
        System.out.println(c1.equals(c2)); // true  
    }  
}
```

Some strange behaviours

```
public static void main(String[] args) {  
    int v1 = 2;  
    int v2 = 2;  
  
    System.out.println(v1 == v2); // true  
  
    Integer i1 = 2;  
    Integer i2 = 2;  
  
    System.out.println(i1 == i2); // true (because of caching)  
  
    i1 = 160;  
    i2 = 160;  
  
    System.out.println(i1 == i2); // false  
  
    String s1 = "LSINF1102";  
    String s2 = "LSINF1102";  
  
    System.out.println(s1 == s2); // true because of string constant "interning"  
  
    String s3 = s1+"";  
  
    System.out.println(s1 == s3); // false  
}
```

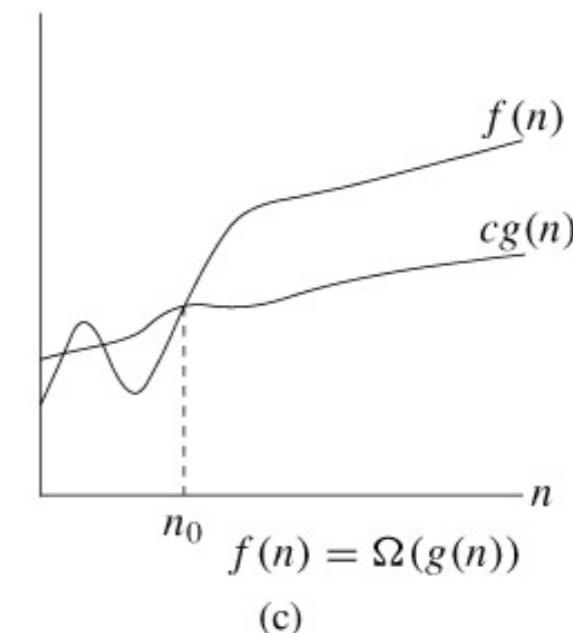
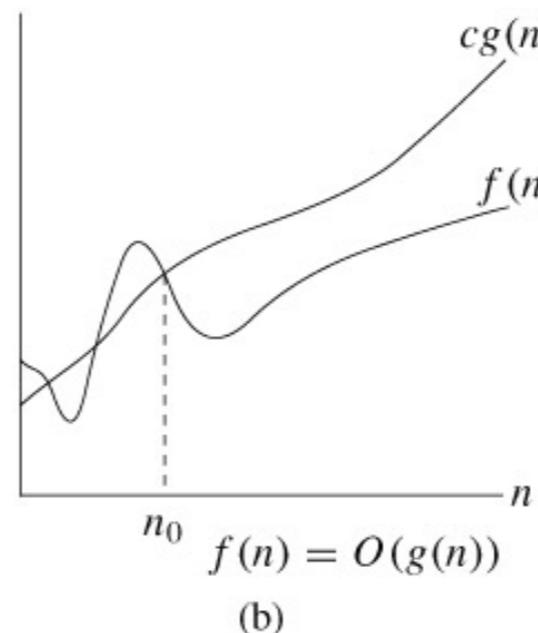
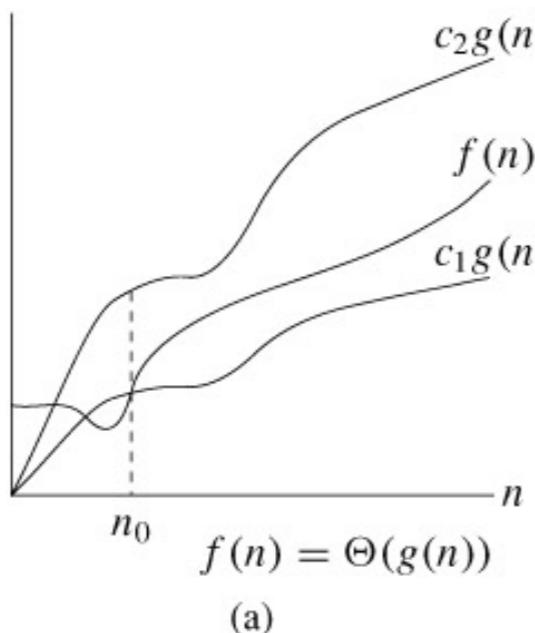
La complexité des algorithmes

- D.E.Knuth a postulé qu'il est possible de construire un modèle mathématique pour décrire le temps d'exécution des algorithmes. En effet le temps repose sur deux facteurs principaux:
 1. Le coût d'exécuter chaque instruction (difficile à estimer, dépend de la JVM, de l'OS).
 2. La fréquence d'exécution de chaque instruction (on peut la calculer car elle dépend uniquement de l'algorithme et des données d'entrées).



Complexities

- $T(n)$ pour $n \in N = \{0, 1, 2, \dots\}$ est une fonction du temps de calcul d'un algorithme pour une entrée de taille n
- Etant donné une fonction $g(n)$ pour $n \in N = \{0, 1, 2, \dots\}$
 - $\Theta(g(n)) = \{ f(n) : \text{il existe les constantes positives } c_1, c_2 \text{ et } n_0 \text{ telles que } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \}$
 - $O(g(n)) = \{ f(n) : \text{il existe les constantes } c \text{ et } n_0 \text{ telles que } 0 \leq f(n) \leq c.g(n) \text{ for all } n \geq n_0 \}$
 - $\Omega(g(n)) = \{ f(n) : \text{il existe les constantes } c \text{ et } n_0 \text{ telles que } 0 \leq c.g(n) \leq f(n) \text{ for all } n \geq n_0 \}$
- We should write $T(n) \in \Theta(g(n))$ but write $T(n) = \Theta(g(n))$

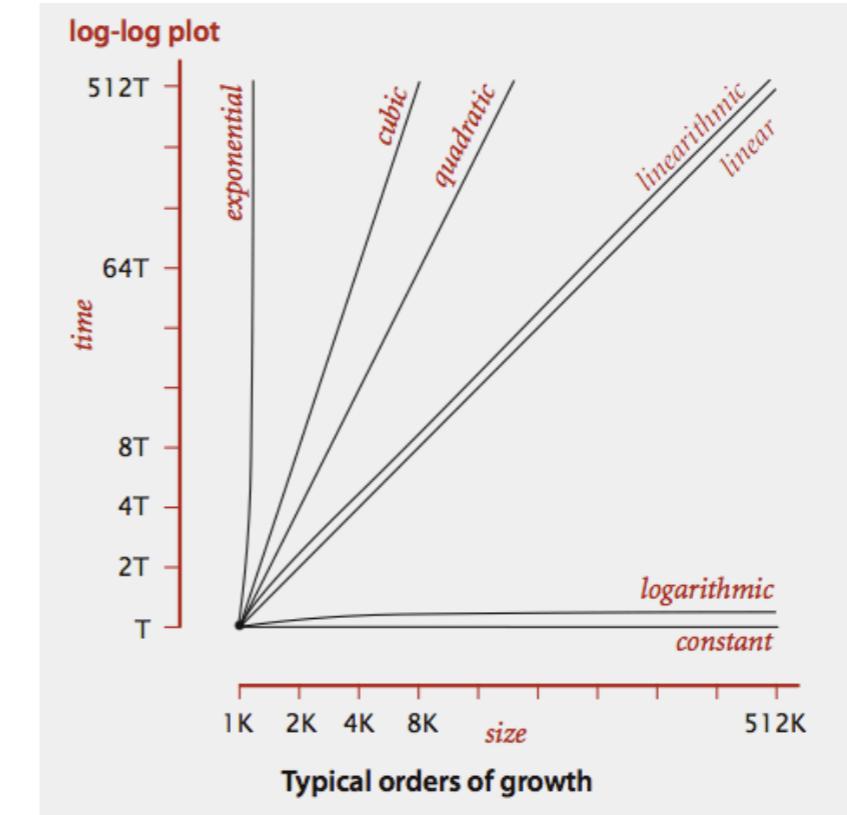
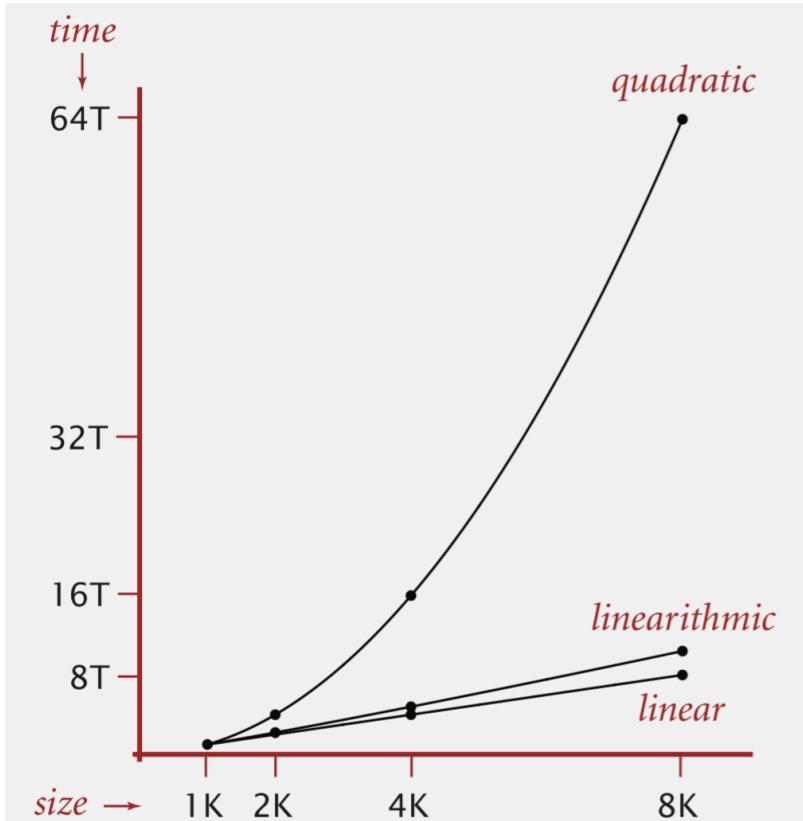


La notation « ~ »

- Le livre utilise une définition de complexité différente et moins standard:
 - $\sim g(n) = \{ f(n) : \lim_{n \rightarrow \infty} f(n)/g(n) = 1 \}$
 - On écrit $f(n) \sim g(n)$ pour exprimer que $f(n)/g(n)$ tend vers 1 lorsque n augmente c'est à dire que pour de grandes valeurs de n, f et g se comportent de la même manière.
 - Pour la plupart des algorithmes f(n) a la forme $a.f(n)$ avec $f(n) = n^b(\log(n))^c$ appelé « **order of growth function** »

En pratique

- Vous pouvez considérer que $O(n^2)$ devient impraticable assez rapidement pour $n > 10^6$



| | insertion sort $O(n^2)$ | | | merge sort $O(n \log(n))$ | | | |
|--------|-------------------------|--------|---------|---------------------------|------------|--------|--------|
| taille | 10^3 | 10^6 | 10^9 | taille | 10^3 | 10^6 | 10^9 |
| temps | instantané | 2.8 h | 317 ans | temps | instantané | 1 s | 18 min |

Exemple d'gorithme 3SUM

Nombre de triplets dont la somme est égale à 0

```
public static int count(int[] a) {
    int n = a.length;
    int count = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i+1; j < n; j++) {
            for (int k = j+1; k < n; k++) {
                if (a[i] + a[j] + a[k] == 0) {
                    count++;
                }
            }
        }
    }
    return count;
}
```

Complexité ?

Un algo pour calculer la complexité?

- Est-ce que nous aurions pu concevoir un algorithme capable de trouver la complexité de ...
 - `public static int count(int[] a)`
- ... sans analyser le code, uniquement en mesurant le temps à l'aide de `System.currentTimeMillis()` pour différents input ?
- La réponse est oui: C'est le « doubling ratio test »



Doubling ratio test*

Si $T(N) \sim a \underline{N^b} \log(N)$ alors $T(2N)/T(N) \sim 2^b$

- Preuve:
- $a(2N)^b \log(2N)/(aN^b \log(N)) = 2^b(1 + \log(2)/\log(N)) \sim 2^b$
- Donc si on mesure $T(2N)/T(N)$ et qu'on prend le logarithme on mesure b.
- Exemple: 3SUM: Le ratio tend vers 8 donc $b = \log(8) = 3$

| N | time | ratio |
|------|------|-------|
| 1000 | 0.1 | |
| 2000 | 0.8 | 8 |
| 4000 | 6.4 | 8 |
| 8000 | 51.1 | 7.98 |

Exam Question 2019

- Mesures de temps pour différentes tailles d'input

| N | time (s) |
|-----|----------|
| 100 | 5 |
| 200 | 40 |
| 400 | 320 |
| 800 | 2560 |

- Quelle est la « order of growth » function ?

Java Exercice



Refresh
My Memory

```
public static void foo(int n) {  
    int [] a = new int[n];  
}
```

- Quelle est la complexité $T(n)$ de `foo` où n est la valeur donnée en argument ?
 - ▶ $\Theta(n)$
 - ▶ $\Theta(1)$
 - ▶ Autre

Java Exercice



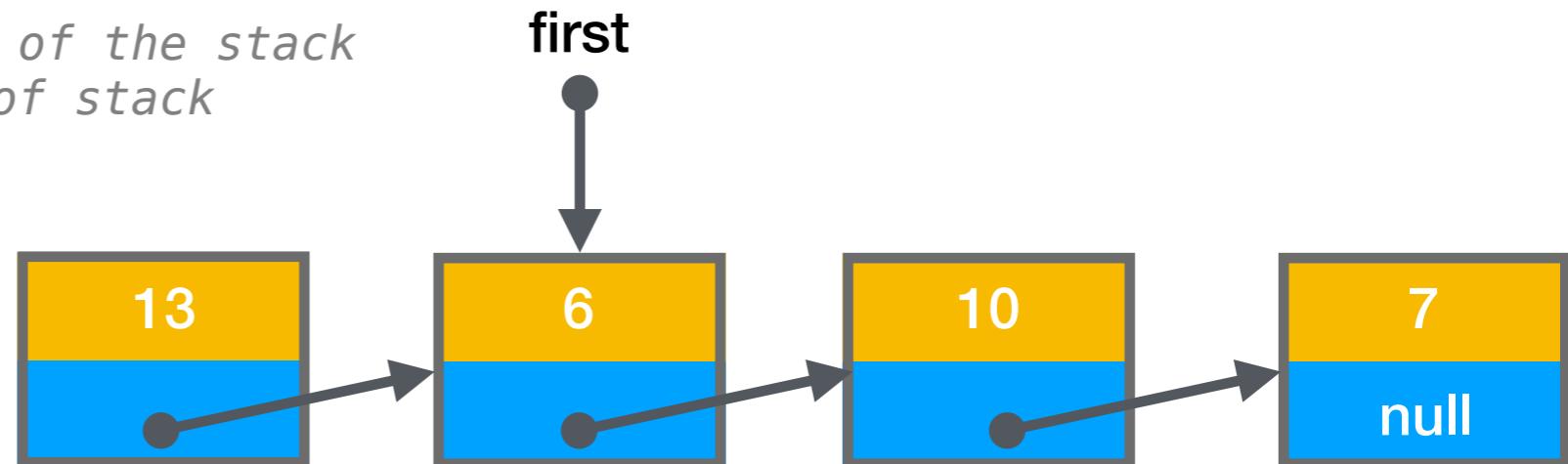
Refresh
My Memory

```
public static int foo(int [] a) {  
    return a.length;  
}
```

- Quelle est la complexité $T(n)$ de `foo` où n est la taille du tableau ?
 - $\Theta(n)$
 - $\Theta(1)$
 - Autre

Stack avec structure chainée

```
public class LinkedStack<Item> {  
  
    private int n;          // size of the stack  
    private Node first;     // top of stack  
  
    // helper linked list class  
    private class Node {  
        private Item item;  
        private Node next;  
    }  
  
    public LinkedStack() {  
        first = null;  
        n = 0;  
        assert check();  
    }  
    public boolean isEmpty() { return first == null; }  
    public int size() { return n; }  
    public void push(Item item) {  
        Node oldfirst = first;  
        first = new Node();  
        first.item = item;  
        first.next = oldfirst;  
        n++;  
    }  
    public Item pop() {  
        if (isEmpty()) throw new NoSuchElementException("Stack underflow");  
        Item item = first.item;           // save item to return  
        first = first.next;              // delete first node  
        n--;  
        return item;                    // return the saved item  
    }  
}
```



push(13)

Complexité pour
le push/pop?

Stack avec un tableau

- Insertion: si on arrive à la limite du tableau, on double la taille du tableau et y recopie tous les éléments



- $n=5$

push(10)

Stack avec un tableau

```
public class ResizingArrayStack<Item> {  
    private Item[] a;                      // array of items  
    private int n;                         // number of elements on stack  
  
    public ResizingArrayStack() {  
        a = (Item[]) new Object[2];  
        n = 0;  
    }  
    public boolean isEmpty() { return n == 0; }  
    public int size() { return n; }  
    private void resize(int capacity) {  
        Item[] temp = (Item[]) new Object[capacity];  
        for (int i = 0; i < n; i++) {  
            temp[i] = a[i];  
        }  
        a = temp;  
    }  
    public void push(Item item) {  
        if (n == a.length) resize(2 * a.length); // 2*size if necessary  
        a[n++] = item;                         // add item  
    }  
    public Item pop() {  
        if (isEmpty()) throw new NoSuchElementException("Stack underflow");  
        Item item = a[n - 1];  
        a[n - 1] = null;                      // Important for Garbage Collection  
        n--;  
        // shrink size of array if necessary  
        if (n > 0 && n == a.length / 4) resize(a.length / 2);  
        return item;  
    }  
}
```

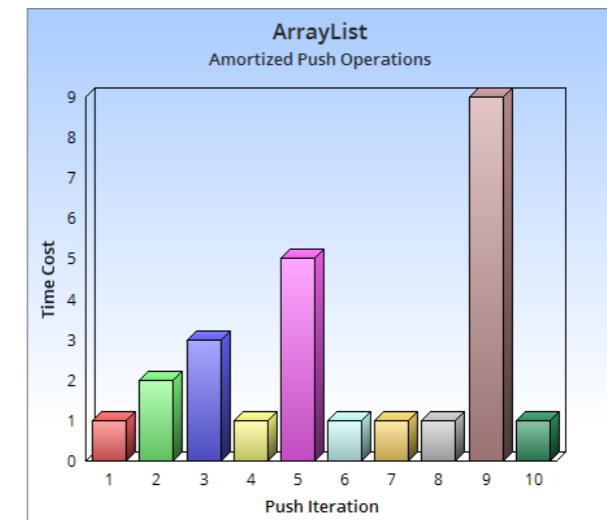
Complexité du
push/pop?

Complexité amortie

- Le push peut prendre $O(1)$ s'il ne faut pas doubler la taille ou $\Theta(n)$ dans le cas où il faut doubler la taille.
- Une analyse intéressante est la **complexité amortie** (amortized time-complexity) pour $n+1$ opérations push() lorsque le tableau a une taille de n

- Cela coutera en moyenne par operation

$$O(1)*n + O(n)/(n+1) = O(1)$$



- Mais attention la complexité d'un push arbitraire est bien $\Omega(1)$ et $O(n)$ où n est le nombre d'éléments dans la Stack.

Complexité attendue

- Pour certains algorithmes la complexité dépend de l'input (des valeurs) et pas seulement de la taille de l'input.
- Exemple: Quicksort est en $O(n^2)$ si les valeurs sont déjà triées mais en $O(n \log(n))$ pour des valeurs triées aléatoirement initialement. Pour se prémunir du pire cas, il est préférable de mélanger aléatoirement les valeurs. On parle alors de **complexité attendue** (expected time complexity) en $O(n \log(n))$

Pour cette semaine

<https://lsinf1121.readthedocs.io/>

- Si besoin: se rafraîchir en java, écrire une petite classe.
- Se familiariser avec IntelliJ et être capable d'écrire un petit test junit.
- Lire les chapitre 1.1-1.4 (pas besoin de lire ce que vous connaissez)
- Lire le rappel sur la complexité
- Préparer les exercices théoriques pour le TP
- L'assistant ne donnera du feedback que si vous avez réellement lu le livre et préparé les exercices ?

One more thing ...

- Le cours est open-source <https://bitbucket.org/pschaus/lisinf1121/src/master/> (format restructured text)
- Un bonus de 1 points sera attribué aux étudiants qui contribuent activement à l'amélioration du cours via des pull-request (l'assistant pourra montrer s'il y a une demande)
 - correction de fautes, proposition d'exercices, etc.
 - <https://www.atlassian.com/git/tutorials/making-a-pull-request>

