# LINFO1121 2023-2024
# Algorithmique et Structure de données
# Complexité, Type-Abstrait de données, Structure Chainées

*TA: Harold Kiossou, Alice Burlats, Achille Morenville + Tutors*

**pierre.schaus@uclouvain.be**

# 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
  - Les compétences acquises dans LINFO1121 sont utilisées dans beaucoup d'autres cours.
    - ✳ Savez vous que l'algorithme Dijkstra est utilisé dans chaque routeur d'internet ?
- La place des algorithmes dans notre société est grandissante. On parle de « société algorithmique »
- Si vous ne maitrisez 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:

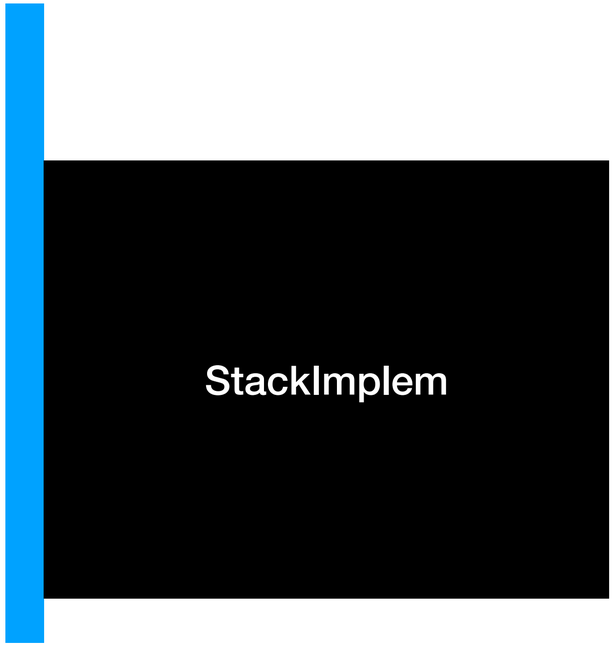
1. You will be asked an open ended question. Ask clarifying questions, devise requirements.
2. You will be asked to explain it in an **algorithm**.
3. Convert it to workable code. (Hint: Don't worry about getting it perfect because time is limited. Write what comes but then refine it later. Also make 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 boite noire).
- Le but de ce cours est étudier l'implémentation des boites noires car toutes les implémentations ne se valent pas.

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




API

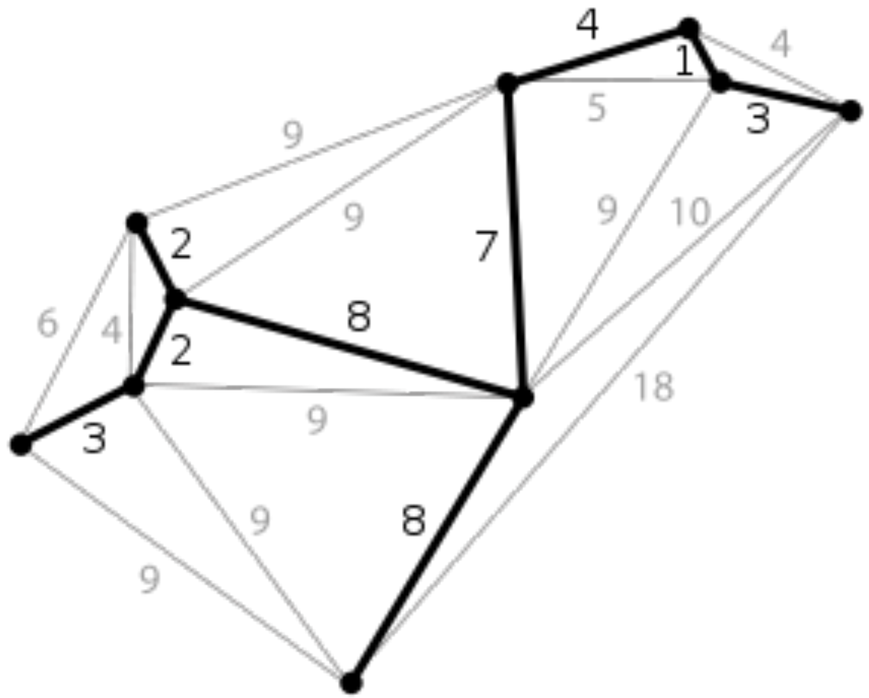
```java
public interface Stack<Item> {
    void push(Item item);
    Item pop();
    boolean isEmpty();
    int size();
}
```

# 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, ou l'objet associé à n'importe quelle clef ou ses successeurs
  - 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 chaines de caractères efficacement dans une grande chaine de caractère
  - Calculer des chemins ou des arbres dans des réseaux

# Exemple: Design de réseau

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

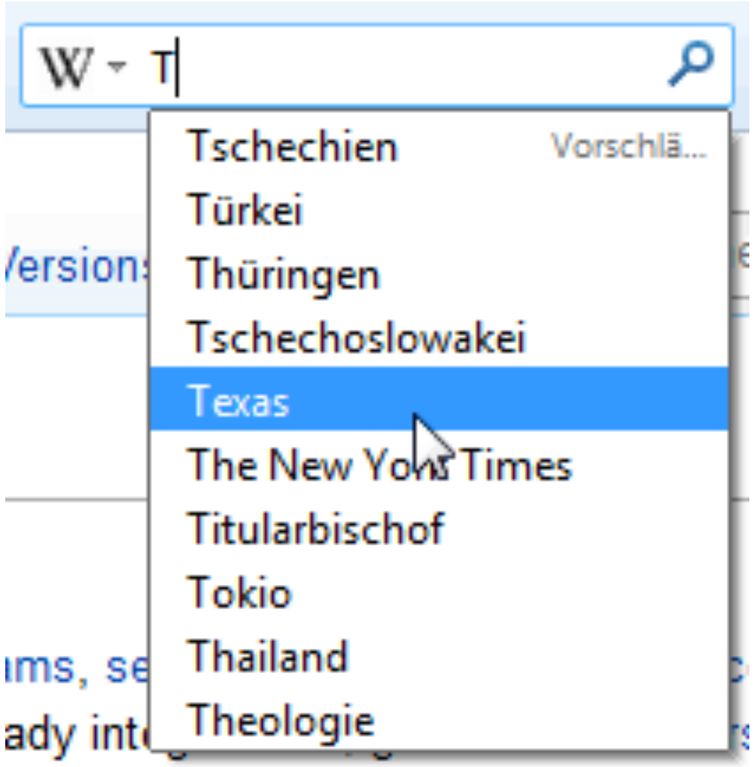



- Application: connecter des maisons à moindre coût avec de la fibre optique

# Exemple: Moteur de recherche

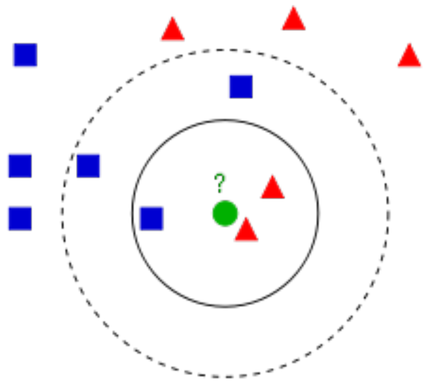
- Moteur de recherche:
  - Vous souhaitez pouvoir stocker des millions de mots clefs et pour chacun une liste d'URLs avec les pages correspondantes.
  - Comment faire pour être capables de retourner rapidement une URL pour un mot clef ?

# Exemple: Auto-Complete



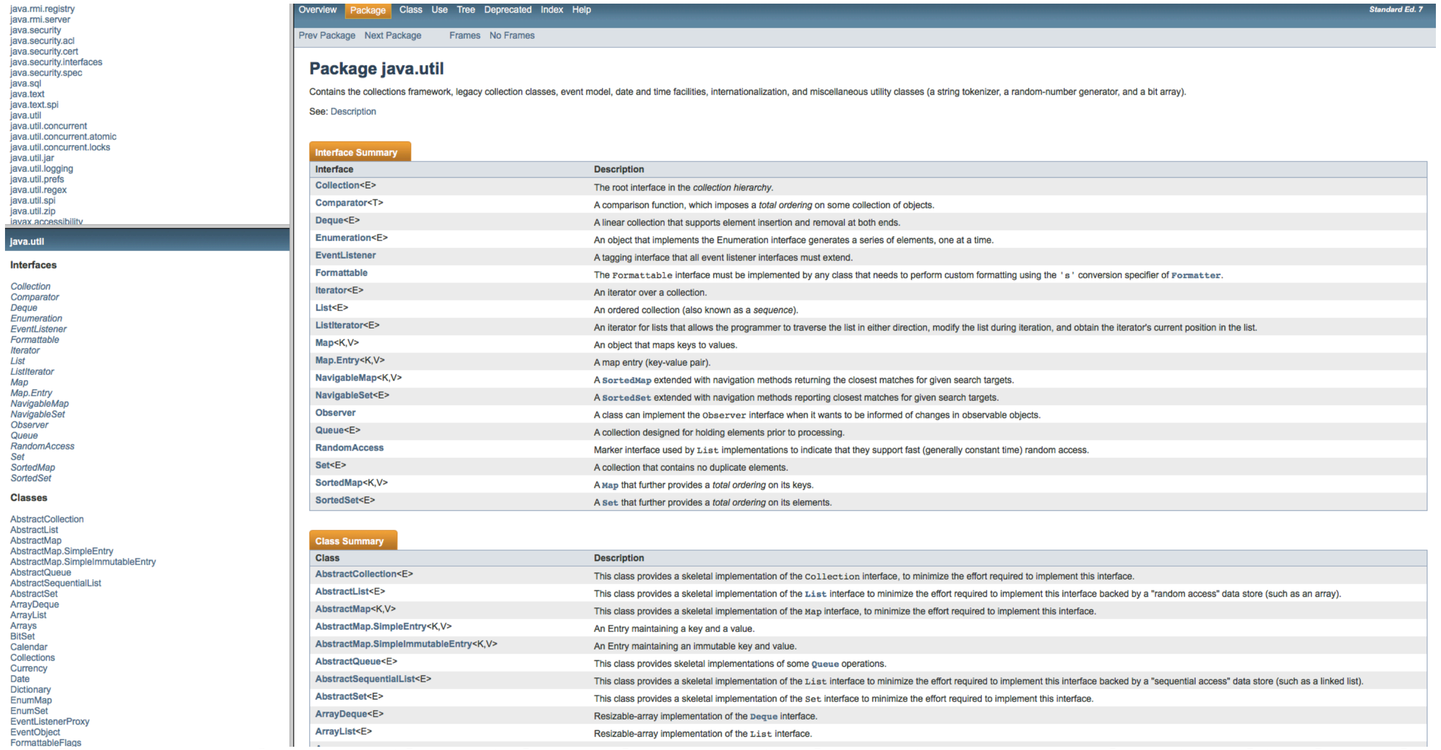
# Exemple: Machine Learning

- 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




- 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

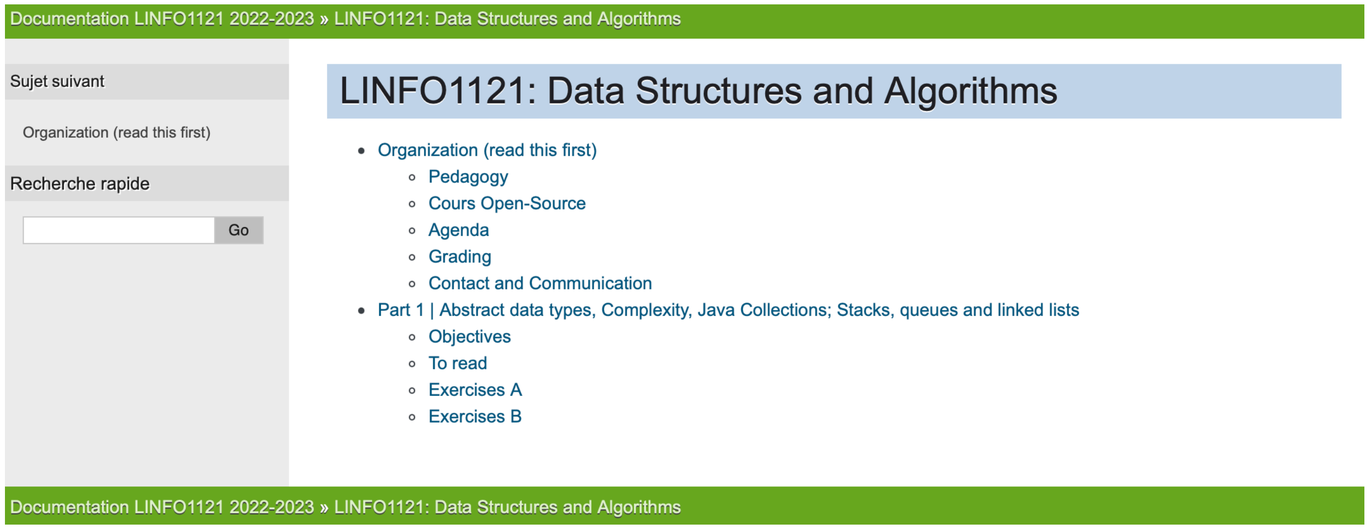
# Livre de référence


- +-65 euros sur Amazon.BE (paper)

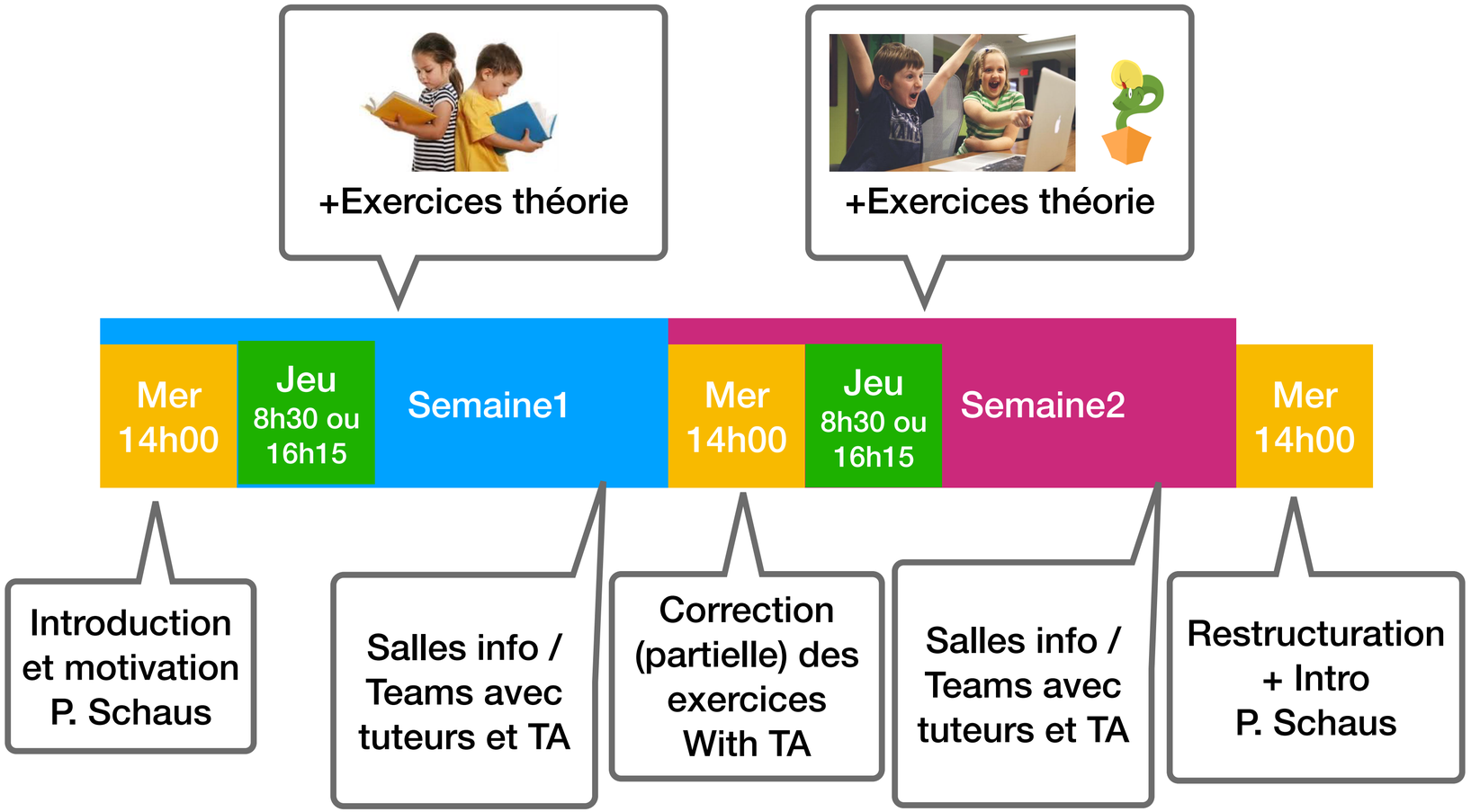
# Le site du cours

- https://moodleucl.uclouvain.be/course/view.php?id=7682 pour les annonces

- https://pschaus.github.io/LINFO1121/ pour le reste



# Organisation



# Evaluation

- Typical Examen
  - 3 (exercices implémentation) x 20/6 points
  - Eventuellement une petite question théorique

- Mid term quiz (optionnel) =
  - 1 exercices d'implémentation (2h pour tout le monde)

- Mid term quiz ne compte dans la note finale que pour 2 points et uniquement s'il fait remonter la note (no-stress).

# Programming Contest (new!)

- En fin de quadrimestre, peut également rapporter deux points si fait remonter la note

- 18 décembre 12h-16h (nous devons encore décider la taille des groupes)

# ChatGPT et IA Générative

- Les IA génératives ne peuvent être utilisées ni pour le quiz, ni pour l'examen.
- Le quiz et l'examen sont individuels, aucune discussion ni collaboration n'est autorisée le temps de l'épreuve.
- Le non-respect de ces directives peut entraîner une réduction des notes ou d'autres **sanctions académiques**.
- Les mêmes conséquences s'appliqueront à un étudiant qui partage volontairement son code ou le rend disponible à d'autres étudiants (quiz).
- Si le professeur le juge nécessaire, un entretien supplémentaire pourra également être organisé pour vérification.

# ⚠️ Plagiat au quiz = ☠️

- Chaque année nous détectons des cas de plagiat de code
- Nous utilisons des outils avancés pour les détecter et sommes aussi très bons pour les identifier.
- Le jeu n’en vaut vraiment pas la chandelle, ces étudiants ont généralement zéro à l’examen + zéro dans de nombreux autres cours. Les sanctions peuvent être bien plus graves, ces cas sont discutés devant tous les professeurs de l’EPL (réputation 📉)

# Exercices d’implementation à l’examen

Ceci est une estimation (pas une règle) …

- 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.log(n)).
  - Réponse: code correct mais en $O(n^2)$ => 4/8
  - Réponse: code buggy en O(n.log(n)) => 0/8

# Pourquoi Java ?


- Java est le langage le plus populaire
- Créé 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 gros 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!)

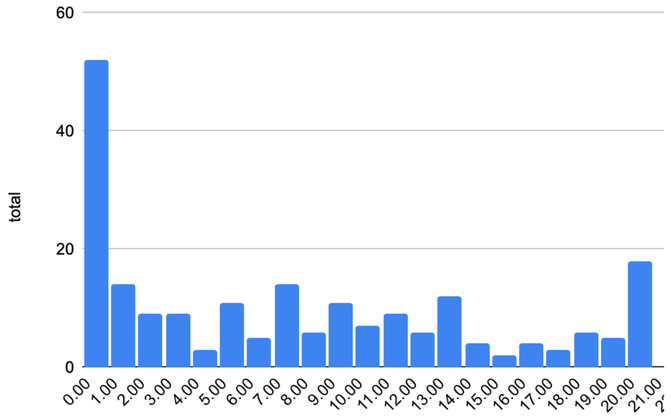
- En java, toutes les collections disponibles sont implémentés en Java et peuvent être étudiées facilement. A la fin de ce cours, l'implémentation de java.util.collection n'aura plus de secrets pour vous.

# Ininious, Inginious, Inginious …. et IntelliJ

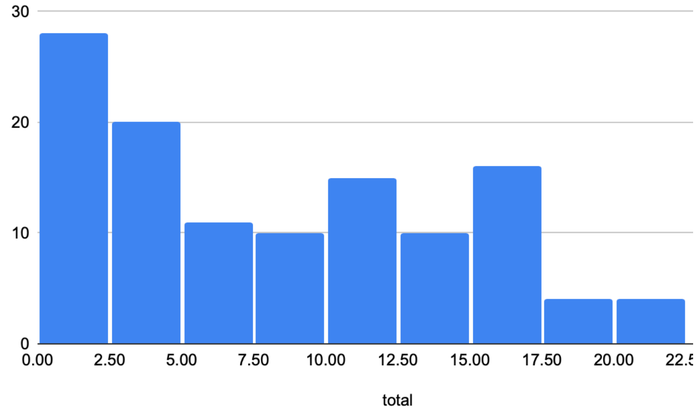
- ⚠️ C’est un cours très difficile! ⚠️

- Le seul moyen de réussir des pratiquer /programmer très régulièrement et de faire tous les exercices proposés

Janvier 2024




Aout 2024



# Raisons de l’échec à l’examen

- Manque de maitrise de Java et/ou IntelliJ (comprendre la théorie de suffit pas)

- Manque de pratique pour corriger vos bugs
  - Cela ne s'acquiert pas en lisant des solutions toute faites mais bien en étant confronté au problème de bug.

- Manque de connaissance des algorithmes en profondeur

*“Patience you must have my young Padawan.”*

# 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 40
    }
}
```

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.

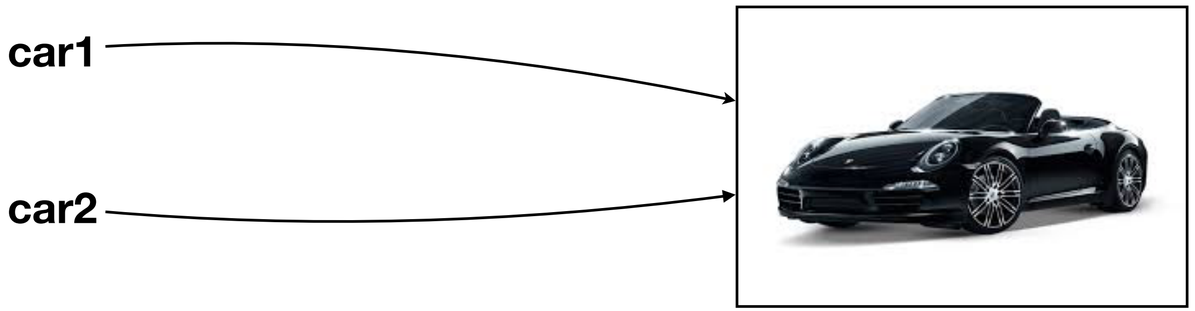
- * except primitive types: int, boolean, char, etc.

# More on references

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

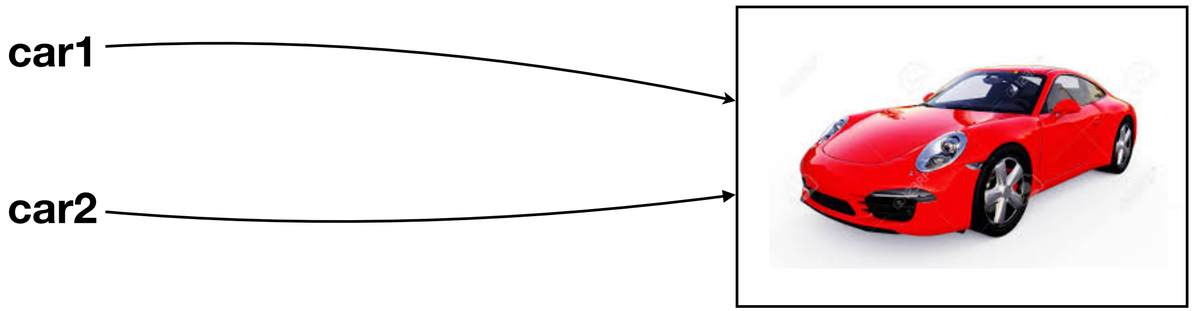
car1 is a reference to an actual Car object

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

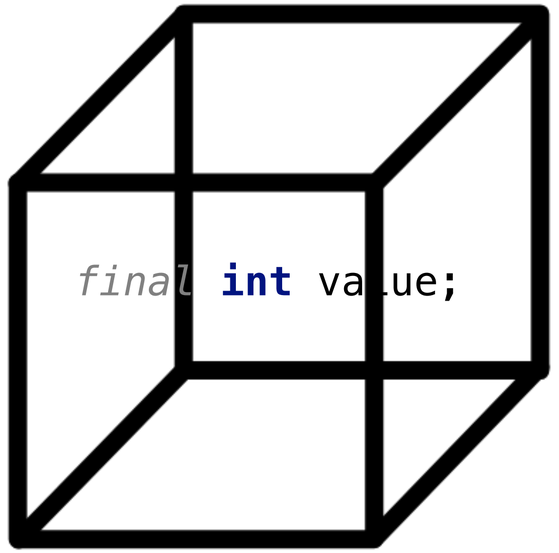
# Auto and out boxing

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(n2); // auto-boxing
}

public static int foo(Integer 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 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 are not passed by reference but **by value**.

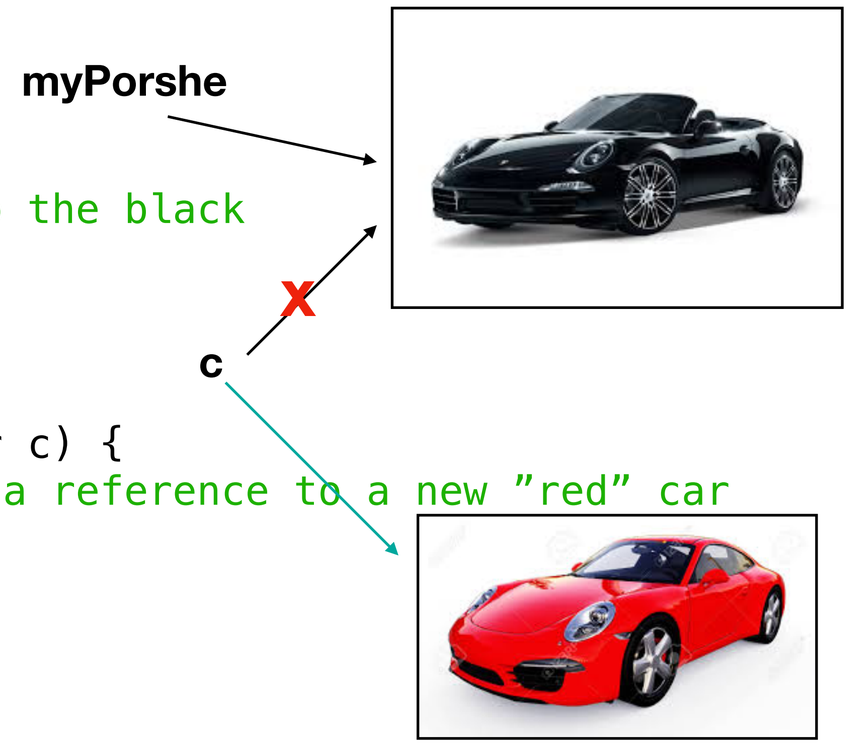
- What is printed here ?

```
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).

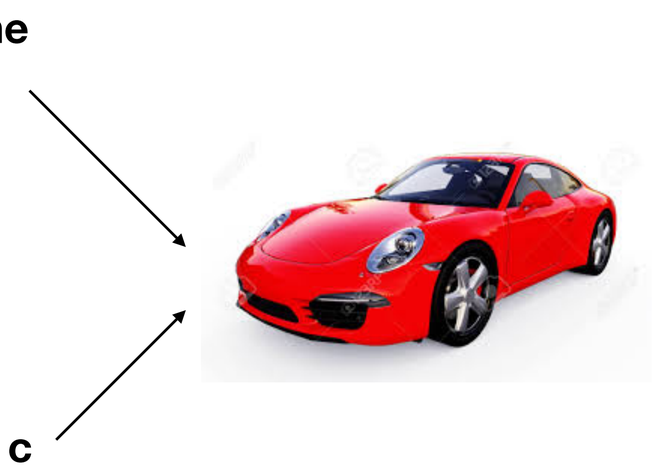
# By value for object (references)?

```
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
}
```



# 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;
}
```



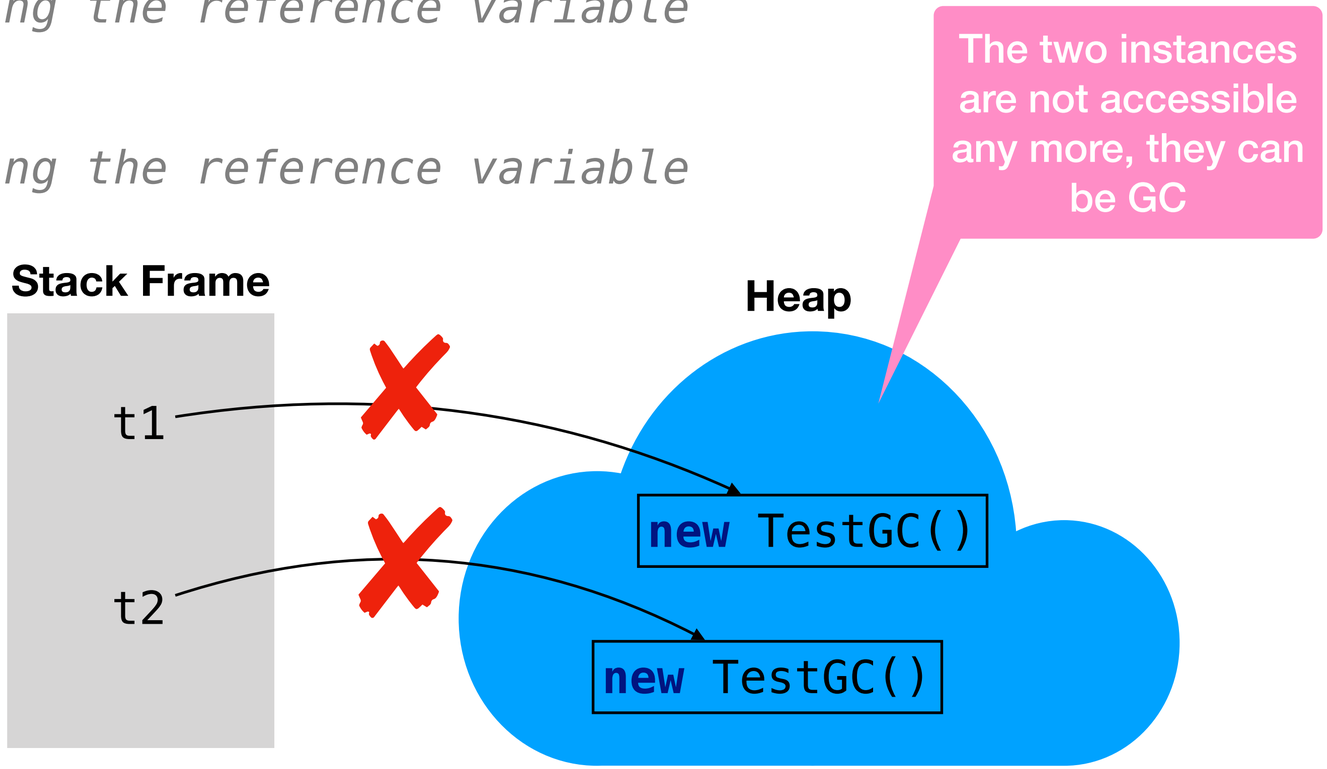
# The null reference

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

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

# Garbage Collection Revisited

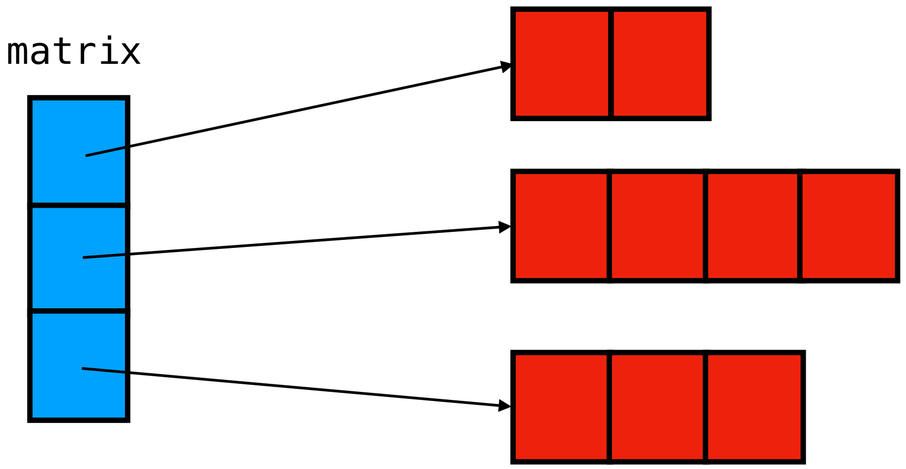
```
public class TestGC {
    public static void main(String[] args) {
        TestGC t1 = new TestGC();
        TestGC t2 = new TestGC();

        // Nullifying the reference variable
        t1 = null;

        // Nullifying the reference variable
        t2 = null;
    }

}
```



# 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.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

```java
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;
    }
}
```


- 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

```java
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;
    }
}
```


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

# Java Exercice


```java
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

# Reference testing

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

    Integer i1;
    Integer i2;

    i1 = 160;
    i2 = 160;

    System.out.println(i1 == i2); // false

    String s1 = "LSINF1102";

    String s3 = s1+"";

    System.out.println(s1 == s3); // false

    Car c1 = new Car("black",5);
    Car c2 = new Car("black",5);

    System.out.println(c1 == c2); // false
}
```

# Some strange behaviour's

```java
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

}
```

# For (logical equality testing) use equals

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

    Integer i1;
    Integer i2;

    i1 = 160;
    i2 = 160;

    System.out.println(i1.equals(i2)); // true

    String s1 = "LSINF1102";

    String s3 = s1+"";

    System.out.println(s1.equals(s3)); // true

    Car c1 = new Car("black",5);
    Car c2 = new Car("black",5);

    System.out.println(c1.equals(c2)); // false (by default same behaviour as ==)
}
```

# 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
    }

}
```

# 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).

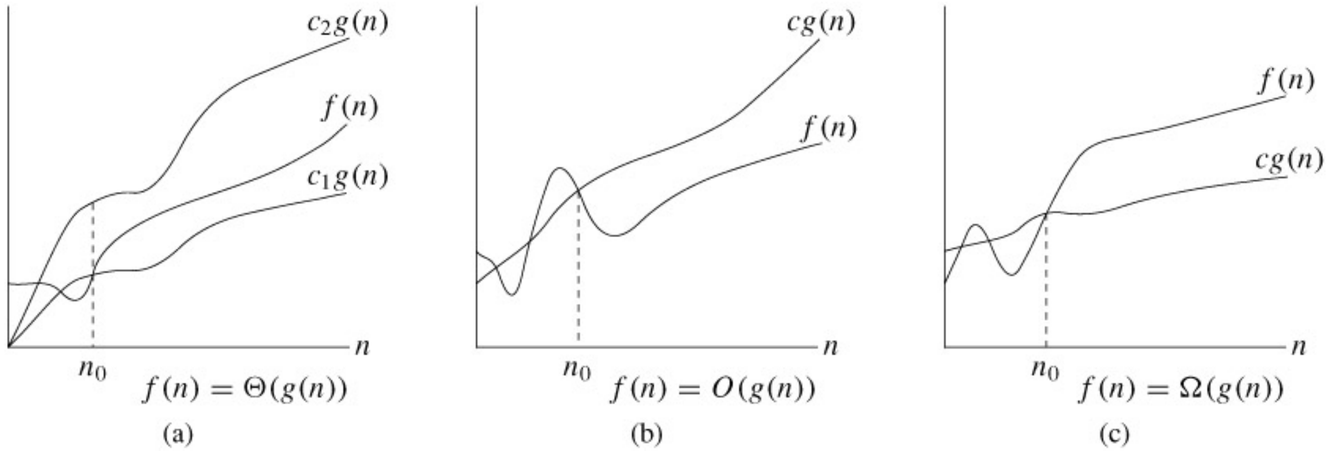
2. La fréquence d'exécution de chaque instruction (on peut la calculer car elle dépend uniquement de l'algorithme et des données d'entrées).

# Complexities

- T(n) pour n∈N={0,1,2,…} est une fonction du temps de calcul d'un algorithme pour une entrée de taille n

- Etant donné une fonction g(n) pour n∈N={0,1,2,…}
  - $\Theta$(g(n))={ f(n) : il existe les constantes positives $c_1$,$c_2$ et $n_0$ telles que 0≤ $c_1$g(n) ≤ f(n) ≤ $c_2$g(n) for all n ≥ $n_0$ }
  - $O$(g(n))={ f(n) : il existe les constantes c et $n_0$ telles que 0≤ f(n) ≤ c.g(n) for all n ≥ $n_0$ }
  - $\Omega$(g(n))={ f(n) : il existe les constantes c et $n_0$ telles que 0 ≤ c.g(n) ≤ f(n) for all n ≥ $n_0$ }

- We should write T(n)∈$\Theta$(g(n)) but write T(n)=$\Theta$(g(n))

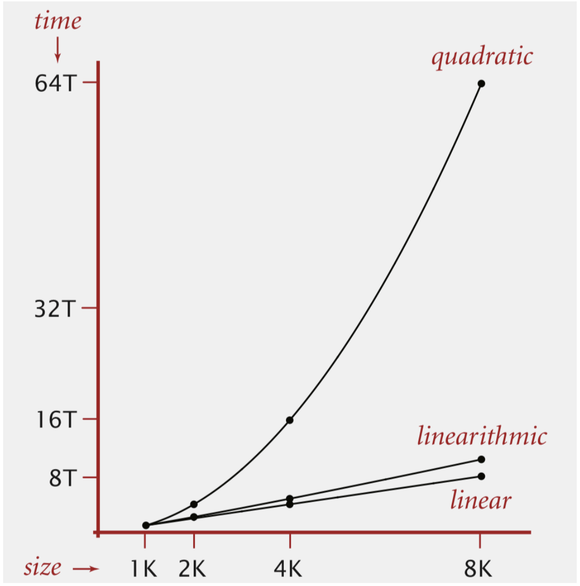


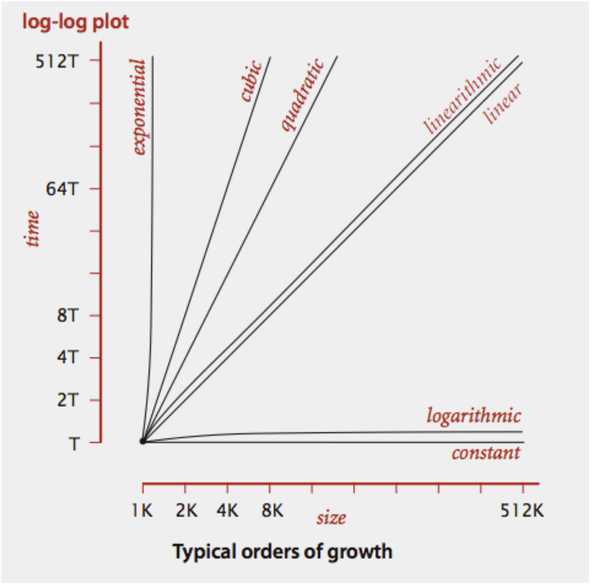
# La notation « ~ »

- Le livre utilise une définition de complexité différente et moins standard:
  - $\sim g(n)=\{ f(n) : \lim_{n\to\infty} f(n)/g(n) = 1 \}$
  - 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 algorithme 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(n2) | | | merge sort O(n log(n)) | | |
|---|---|---|---|---|---|---|
| taille | $10^3$ | $10^6$ | $10^9$ | $10^3$ | $10^6$ | $10^9$ |
| temps | instantané | 2.8 h | 317 ans | instantané | 1 s | 18 min |

# Exemple d’algorithme 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) ~ a $N^b$ log($N$) alors T(2$N$)/T($N$) ~ $2^b$**

- Preuve:
- $$\frac{T(2N)}{T(N)} = \frac{a(2N)^b log(2N)}{aN^b\ log(N)} = 2^b \frac{log(2) + log(N)}{log(N)} = 2^b(1 + \frac{log(2)}{log(N)}) \sim 2^b$$
- Donc si on mesure T(2N)/T(N) pour N assez grand 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 |

*** log est en base 2**

# 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


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

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

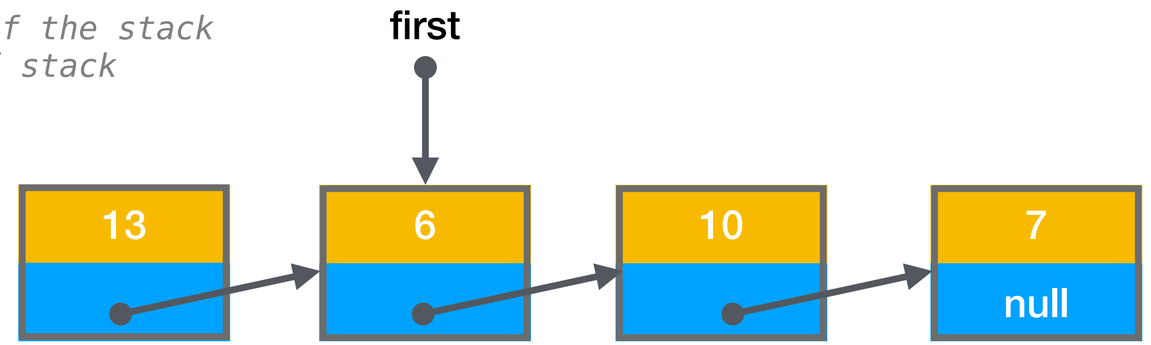
# Java Exercice


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

- Quelle est 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
    }
}
```

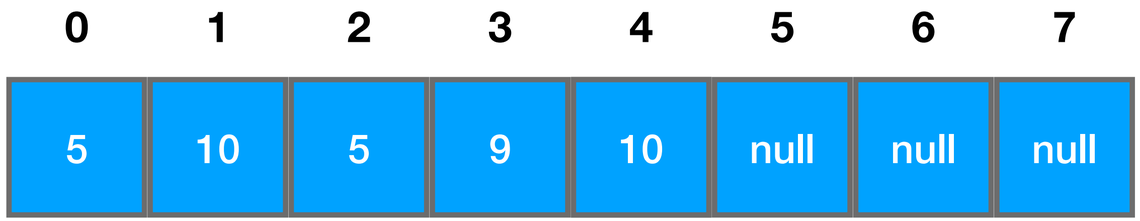



push(13)

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

# Stack avec un tableau

```java
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) {
        a = java.util.Arrays.copyOf(a, capacity);
    }
    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;
    }
}
```

```java
Item[] temp = (Item[]) new Object[capacity];
for (int i = 0; i < n; i++) {
    temp[i] = a[i];
}
a = temp;
```

Complexité du push/pop?

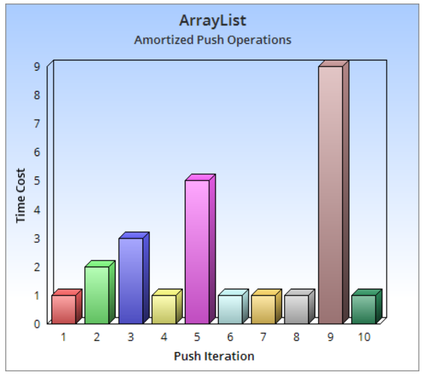
# Complexité amortie

- Le push peut prendre 0(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) ou 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 $O(n \log(n))$

# Iterator Pattern

- Iterator: Provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

# Iterator Interface

```java
public interface Iterator<E> {
    /**
     * Returns {@code true} if the iteration has more elements.
     * (In other words, returns {@code true} if {@link #next} would
     * return an element rather than throwing an exception.)
     *
     * @return {@code true} if the iteration has more elements
     */
    boolean hasNext();
    /**
     * Returns the next element in the iteration.
     *
     * @return the next element in the iteration
     * @throws NoSuchElementException if the iteration has no more elements
     */
    E next();
    /**
     * Removes from the underlying collection the last element returned
     * by this iterator (optional operation).  This method can be called
     * only once per call to {@link #next}.
     * <p>
     * The behavior of an iterator is unspecified if the underlying collection
     * is modified while the iteration is in progress in any way other than by
     * calling this method, unless an overriding class has specified a
     * concurrent modification policy.
     * <p>
     */
    default void remove() {
        throw new UnsupportedOperationException("remove");
    }
}
```

My advise is not to implement the “remove” because it is optional and complicates a lot the code of an iterator

# Iterator Usage

```
List<Integer> list = new LinkedList<>();
list.add(1);
list.add(2);
list.add(3);

Iterator<Integer> iterator = list.iterator();
while (iterator.hasNext()) {
    System.out.println(iterator.next());
}
```

# Iterable Interface

```
public interface Iterable<T> {
    /**
     * Returns an iterator over elements of type {@code T}.
     *
     * @return an Iterator.
     */
    Iterator<T> iterator();

}
```

# Iterable and for-loops

```
List<Integer> list = new LinkedList<>();
list.add(1);
list.add(2);
list.add(3);

for (Integer i : list) {
    System.out.println(i);
}
```

```
Iterator<Integer> iterator = list.iterator();
while (iterator.hasNext()) {
    System.out.println(iterator.next());
}
```

# Structure

```
public class LinkedStack<T> implements Iterable<T> {
    private Node<T> top;
    private int size = 0;

    private static class Node<T> {
        // see prev
    }

    public void push(T item) {
        // see prev
    }
    public T pop() {
        // see prev
    }
    public boolean isEmpty() {
        return top == null;
    }
    public int size() {
        return size;
    }

    @Override
    public Iterator<T> iterator() {
        return new LinkedStackIterator();
    }

    private class LinkedStackIterator implements Iterator<T> {

    }
}
```

Node is a static inner/nested class:

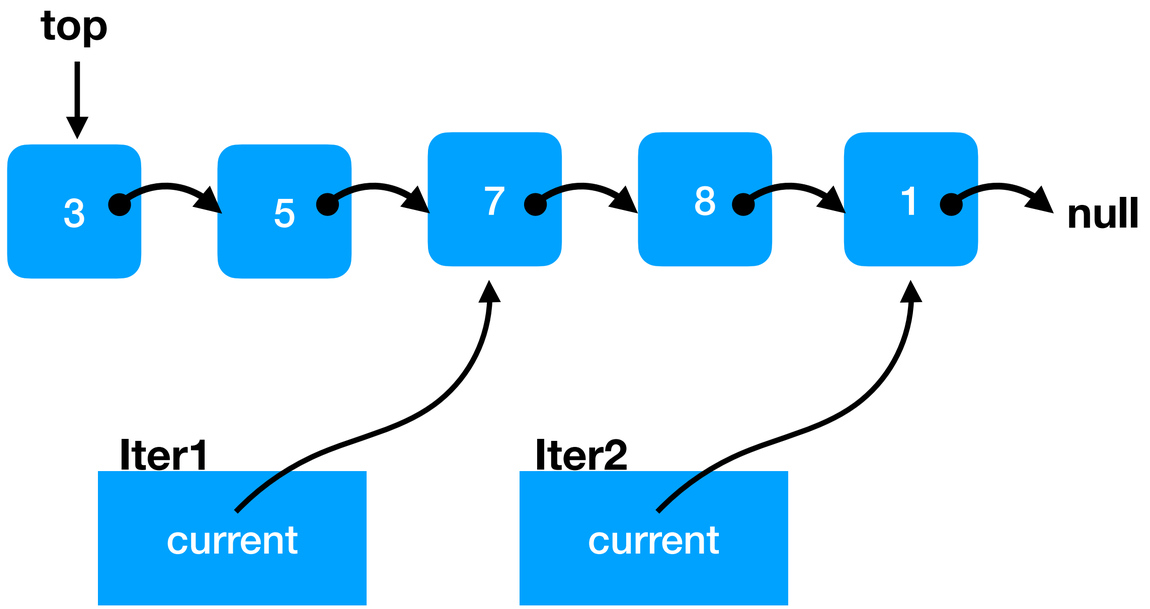
- It doesn't need to exist in its own file Node.java
- The outer-class LinkedBag has access to its private fields (no getter/setter)

The Iterator a non static inner/nested class:

- Its existence it bind to a particular instance of LinkedStack.
- It has access to the instance variables of its attached LinkedStack instance.

# LinkedBag Iterator

```
public class LinkedStack<T> implements Iterable<T> {

    private Node<T> top;
    private int size = 0;

    private static class Node<T> { // see prev }

    public void push(T item) { // see prev }
    public T pop() {// see prev }
    public boolean isEmpty() {return top == null;}
    public int size() {return size;}

    @Override
    public Iterator<T> iterator() {
        return new LinkedStackIterator();
    }
    private class LinkedStackIterator implements Iterator<T> {
       private Node<T> current = top;
       @Override
       public boolean hasNext() {
           return current != null;
       }
       @Override
       public T next() {
           if (current == null) throw new IllegalStateException("No more items");
           T item = current.item;
           current = current.next;
           return item;
       }
    }
}
```



# Iterator design problem

- Two strategies for implementing a proper iterator:
  - **Fail-Fast**: they throw *ConcurrentModificationException* if there is **structural modification** of the collection.
  - **Fail-Safe**: they don't throw any exceptions if a collection is structurally modified while iterating over it. This is because, they operate on the clone of the collection, not on the original collection and that's why they are called fail-safe iterators.

- Let us fix our iterator to implement the "Fail-Fast" strategy.

```
public static void main(String[] args) {
    Stack<String> stack = new LinkedStack<String>();
    stack.push("Computer");
    stack.push("Table");
    Iterator ite = stack.iterator();
    ite.next();
    ite.next();
    stack.push("Table");
    if (ite.hasNext()) {
        System.out.println("how come you have some next !?!?");
    }
}
```

# FailFast iterator for LinkedStack

```
public class LinkedStack<T> implements Iterable<T> {
    private Node<T> top;
    private int size = 0;
    private int modCount = 0;  // Modification count

    public void push(T item) {
        Node<T> oldTop = top;
        top = new Node<>(item, oldTop);
        size++;
        modCount++;
    }

    public T pop() {
        if (top == null) throw new IllegalStateException("Stack is empty");
        T item = top.item;
        top = top.next;
        size--;
        modCount++;
        return item;
    }

    @Override
    public Iterator<T> iterator() {
        return new LinkedStackIterator();
    }
}
```

Step 1: introduce modification counter

```java
public class LinkedStack<T> implements Iterable<T> {
    private Node<T> top;
    private int size = 0;
    private int modCount = 0;  // Modification count

    @Override
    public Iterator<T> iterator() {
        return new LinkedStackIterator();
    }

    private class LinkedStackIterator implements Iterator<T> {
        private Node<T> current = top;
        private final int expectedModCount = modCount;

        @Override
        public boolean hasNext() {
            if (expectedModCount != modCount) {
                throw new ConcurrentModificationException();
            }
            return current != null;
        }

        @Override
        public T next() {
            if (expectedModCount != modCount) {
                throw new ConcurrentModificationException();
            }
            if (current == null) throw new IllegalStateException("No more items");

            T item = current.item;
            current = current.next;
            return item;
        }
    }
}
```

Step 2: snapshot of the counter at the creation of the Iterator

Step 3: verify for modifications between creation and iterator usage by comparing the counter

Step 3: verify for modifications between creation and iterator usage by comparing the counter

# Each iterator has its own counter

```
LinkedStack<Integer> stack = new LinkedStack<>();

stack.push(1);
stack.push(2);
stack.push(3);

// modCount should be 3

Iterator<Integer> iter1 = stack.iterator(); // expectedModCount = 3

stack.push(4);  // modCount = 4;

Iterator<Integer> iter2 = stack.iterator(); // expectedModCount = 4

iter1.hasNext(); // 3 != 4 💥
```

# Pour cette semaine

https://pschaus.github.io/LINFO1121/

- Si besoin: se rafraichir 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 + Deux exercices Inginious
- L'assistant ne donnera du feedback que si vous avez réellement lu le livre et préparé les exercices ?

# One more thing …


- Les exercices sont open-source
  - https://github.com/pschaus/algorithms_exercises_students (exercices inginious)
  - https://github.com/pschaus/algorithms_exercises (solution des exercises inginious)

- N'hésitez pas à contribuer activement à l'amélioration du cours (correction de fautes, tests, proposition d'exercices) via des pull-request
  - https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/creating-a-pull-request