Cours JAVA Avancé

Pierre-Henri FRICOT

Cours Algorithmique Avancé

Objectif pédagogique :

Être capable de d'utiliser les fonctionnalités avancées proposés par l'API standards JAVA.

Valider et appuyer que les notions suivantes :

- Interfaces et Polymorphisme
- Collections, Arrays, Set et List
- Table de hachage HashMap
- Expressions régulières en java
- Fonctions Lambda

Planning de la semaine

Lundi: Cours et TD

Mardi: Cours et TD

Mercredi: Présentation du projet et démarrage

Jeudi: Projet en autonomie

Vendredi: Finalisation du projet. Sprint review et Sprint rétrospective

Java Avancé

Vous avez appris précédemment appris à développer en Java. Maintenant vous allez entrer dans le monde des API JAVA avancés.

- Comment utiliser des algorithmes de tris efficaces
- Quand et Comment utiliser certaines structures de données et pas d'autres.
- Etc…

Installation

Langage de support pour les exercices et projet :

=> JAVA (>=JDK12)

https://www.oracle.com/java/technologies/javase-downloads.html

Editeur de support => IntelliJ Community Edition :

https://www.jetbrains.com/idea/download/

Interfaces et Polymorphisme

Le polymorphisme: Qui peut prendre plusieurs formes

Le polymorphisme c'est le fait qu'une variable de type « générique » puisse contenir plusieurs types d'objets différents.

```
public interface Fruit { }
```

```
public class Kiwi implements Fruit { }
```

```
public class Banana implements Fruit{ }
```

```
public class Apple implements Fruit{ }
```

Le polymorphisme peut être fait avec :

1. Des classes et de l'héritage
2. Des classes abstraites et de l'héritage
3. Des interfaces

```
public class Main {
    public static void main(String[] args) {
        Fruit fruit1 = new Banana();
        Fruit fruit2 = new Apple();
        Fruit fruit3 = new Kiwi();
    }
}
```

Une interface peut être implémentée par diffèrent types d'objets et un objet peut implémenter plusieurs interfaces.

Écrire du code polymorphe

On veut coder une classe Zoo qui aura plusieurs cages permettant d'accueillir différents types d'animaux.

Il faut donc implémenter une classe Cage permettant de contenir tous ces animaux.

```
public class Zoo {
    public static void main (String[] args) {
        Cage uneCage1 = new Cage(...);
        Cage uneCage2 = new Cage(...);
        // On ajoute un lion
        Lion unLion = new Lion(...);
        uneCage1.accueillir(unLion);
        // On ajoute un singe
        Singe unSinge = new Singe (...);
        uneCage2.accueillir(unSinge);
    }
}
```

Écrire du code polymorphe : la mauvaise solution

Ici, la surcharge est une très mauvaise solution.

Si une nouvelle espèce animale doit être prise en compte, il faudra modifier le code de la classe Cage.

```
public class Cage {
    public void accueillir(Lion l) {
        ...
    }
    public void accueillir(Singe s) {
        ...
    }
}
```

Écrire du code polymorphe : la bonne solution

La bonne solution consiste à utiliser le polymorphisme, en implémentant une méthode accueillir générique pour tout les animaux.

Son paramètre étant de type Animal on pourra l'appeler avec une référence de Lion ou de Singe.

```
public class Cage {
    public void accueillir(Animal a) {
        System.out.println(a.getNom( ) + "est en cage");
        ...
    }
}
```

Motivation

Nouvelle spécification à prendre en compte : tous les animaux ne peuvent pas aller en cage.

La méthode Cage.accueillir() doit être à même de détecter les animaux ne pouvant l'être.

```
public class Zoo {
    public static void main (String[] args) {
        Cage uneCage1 = new Cage(...);
        Homme unHomme = new Homme(...);
        uneCage.accueillir(unHomme); // => il refuse !!!
    }
}
```

Première solution

```
public class Cage {
    ...
    public void accueillir(Animal a) {
        if (a instanceof Homme)
            System.out.println(a.getNom( ) + "refuse d'aller en cage");
        return;
        ...
    }
}
```

Très mauvaise solution

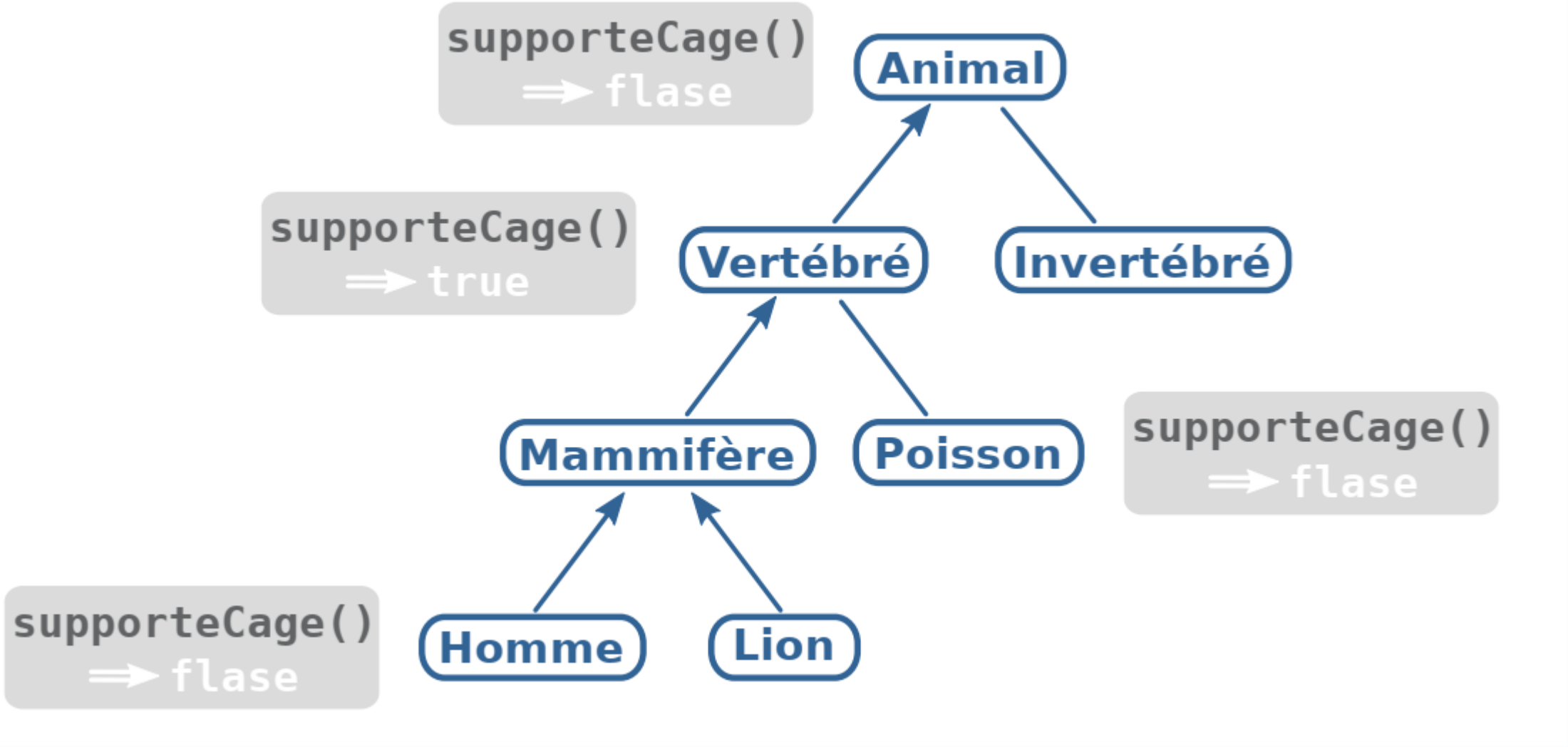
- Des connaissances propres à la classe Homme sont dans la classe Cage
- si une nouvelle espèce animale refuse d'aller en cage, il faudra modifier le code de la classe Cage

Mise en place d'un protocole

```
public class Cage {
    public void accueillir (Animal a) {
        if (!a.supporteCage()) {
            System.out.print(a.getNom());
            System.out.println(" refuse d'aller en cage");
            return;
        }
    }
}
```

1. Tout animal doit pouvoir répondre à ce protocole
2. Nous allons implémenter le protocole dans la hiérarchie de racine Animal, en utilisant l'héritage et en spécialisant lorsque c'est nécessaire

Protocole dans l'arbre d'héritage



Mise en place d'un protocole

```
public class Animal {
    public boolean supporteCage() { return false; }
}

public class Vertébré extends Animal {
    public boolean supporteCage() { return true; }
}

public class Homme extends Mammifère {
    public boolean supporteCage() { return false; }
}

public class Zoo {
    public static void main (String[] args) {
        Cage uneCage1 = new Cage(...);
        uneCage1.accueillir(new Lion(...));
        // Génère un appel à Vertébré.supporteCage();
    }
}
```

Motivation pour les classes abstraite

```
public class Animation {
    ...
    public void faireManger(Animal a , Nourriture n) {
        ...
        a.manger(n);
        ...
    }
    ...
}
```

Pour que le polymorphisme marche/compile il faut introduire une méthode manger() dans la classe Animal

Problème:

Quel comportement y décrire puisque la façon de manger dépend de l'espèce animale ?

Classes abstraite

Spécification le plus haut possible dans l'arbre :

```
public abstract class Animal {
    ...
    public abstract void manger(Nourriture n);
    // pas de code associé
}
```

Spécialisation là où c'est nécessaire dans l'arbre d'héritage:

```
public class Lion extends Mammifère {
    ...
    @Override
    public void manger(Nourriture n) {
        ... // code spécifique aux lions
    }
}
```

Méthodes abstraites ⇒ Classe abstraite

Définition

Un classe contenant au moins une méthode abstraite est appelée une classe abstraite et cela doit être explicitement précisé dans la déclaration avec : abstract class

Remarque(s)

- Une classe abstraite peut contenir des méthodes concrètes.
- Une classe peut être déclarée abstraite sans contenir de méthode abstraite.
- Une classe abstraite ne peut pas être instanciée, car son comportement n'est pas complètement défini

```
Animal unAnimal = new Animal (...); // ERREUR
```

Le polymorphisme : dans les listes

Une liste sans polymorphisme

A est un objet

List<A>
A
A
A
A
A

Une liste avec polymorphisme.

Les objets B, C et D étendent une interface A

```
public class Main {
    public static void main(String[] args) {
        Fruit fruit1 = new Banana();
        Fruit fruit2 = new Apple();
        Fruit fruit3 = new Kiwi();

        List<Fruit> fruits = new ArrayList<>();
        fruits.add(fruit1);
        fruits.add(fruit2);
        fruits.add(fruit3);
        fruits.add(new Banana());
        fruits.add(new Apple());
        fruits.add(new Kiwi());
    }
}
```

List<A>
B
B
C
A
D

Le polymorphisme: Dans les appelles de fonctions

Jusqu'à présent nous avons vu le polymorphisme sous l'aspect type mais ce n'est pas tout. Des langages non typé on aussi une notions de polymorphisme. C'est la notion d'appel de fonction polymorphique. Exemple :

```java
public interface Fruit {
    public void eat();
}
```

```java
public class Apple implements Fruit {
    @Override
    public void eat() {
        System.out.println("I eat an Apple");
    }
}
```

```java
public class Banana implements Fruit {
    @Override
    public void eat() {
        System.out.println("I eat a Banana");
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Fruit fruit1 = new Banana();
        Fruit fruit2 = new Apple();
        Fruit fruit3 = new Kiwi();
        fruit1.eat();
        fruit2.eat();
        fruit3.eat();
    }
}
```

```
I eat a Banana
I eat an Apple
I eat a Kiwi
```

Le polymorphisme: Dans les appelles de fonctions

De la même manière se principe s'applique dans le parcours de liste

```java
public class Main {
    public static void main(String[] args) {
        Fruit fruit1 = new Banana();
        Fruit fruit2 = new Apple();
        Fruit fruit3 = new Kiwi();
        fruit1.eat();
        fruit2.eat();
        fruit3.eat();

        List<Fruit> fruits = new ArrayList<>();
        fruits.add(fruit1);
        fruits.add(fruit2);
        fruits.add(fruit3);
        fruits.add(new Banana());
        fruits.add(new Apple());
        fruits.add(new Kiwi());
        for (Fruit fruit : fruits) {
            fruit.eat();
        }
    }
}
```

```
I eat a Banana
I eat an Apple
I eat a Kiwi
I eat a Banana
I eat an Apple
I eat a Kiwi
I eat a Banana
I eat an Apple
I eat a Kiwi
```

Exercice : Formes

Ecrire les classes nécessaires au fonctionnement du programme suivant (en ne fournissant que les

méthodes nécessaires à ce fonctionnement). :

```
public class TestFormes {
    public static void main(String[] argv) {
        Forme[] figures = new Forme[3];
        figures[0] = new Carre(2.0); // Création d'un carré de 2 cm de coté
        figures[1] = new Cercle(3.0); // Création d'un cercle de 3 cm de rayon
        figures[2] = new Carre(5.2); // Création d'un carré de 5,2 cm de coté
        for (int i = 0; i < figures.length; i++) {
            System.out.println(figures[i] + " : surface = " + figures[i].getSurface() + " cm2");
        }
    }
}
```

Sortie de ce programme :

Carré (coté 2.0 cm) : surface = 4.0 cm2

Cercle (rayon 3.0 cm) : surface = 28.26 cm2

Carré (coté 5.2 cm) : surface = 27,04 cm2

Rappel :

- Surface d'un carré = coté*coté
- Surface d'un cercle = Pi*rayon*rayon

A faire 3 fois en utilisant un système de classe, de classe abstraite et d'interface pour le type Forme

Exercice : Métiers

Ecrire les classes nécessaires au fonctionnement du programme suivant (en ne fournissant que les

méthodes nécessaires à ce fonctionnement).

```
public class TestMetiers {
    public static void main(String[] argv) {
        Personne[] personnes = new Personne[3];
        personnes[0] = new Menuisier("Paul");
        personnes[1] = new Plombier("Jean");
        personnes[2] = new Menuisier("Adrien");
        for (int i = 0; i < personnes.length; i++) {
            personnes[i].affiche();
        }
    }
}
```

Sortie de ce programme :

Je suis Paul le Menuisier

Je suis Jean le Plombier

Je suis Adrien le Menuisier

Exercice : Magic

Vous vous intéressez dans cet exercice à décrire les données d'un jeu simulant des combats de magiciens.

Dans ce jeu, il existe trois types de cartes : les terrains, les créatures et les sortilèges.

Les terrains possèdent une couleur (parmi 5 : blanc('B'), bleu ('b'), noir ('n'), rouge ('r') et vert ('v').)

Les créatures possèdent un nom, un nombre de points de dégâts et un nombre de points de vie.

Les sortilèges possèdent un nom et une explication sous forme de texte.

De plus, chaque carte, indépendamment de son type, possède un coût. Celui d'un terrain est 0.

Dans un programme Magic.java, proposez (et implémentez) une hiérarchie de classes permettant de représenter des cartes de différents types.

Ajoutez ensuite aux cartes une méthode afficher() qui, pour toute carte, affiche son coût et la valeur de ses arguments spécifiques.

Créez de plus une classe Jeu pour représenter un jeu de cartes, c'est-à-dire une collection de telles cartes.

Cette classe devra avoir une méthode piocher permettant d'ajouter une carte au jeu.

On supposera qu'un jeu comporte au plus 10 cartes. Le jeu comportera également une méthode joue permettant de jouer une carte. Pour simplifier, on jouera les cartes dans l'ordre où elles sont stockées dans le jeu.

Questions?

Structure de données

Les deux grands types de donnée

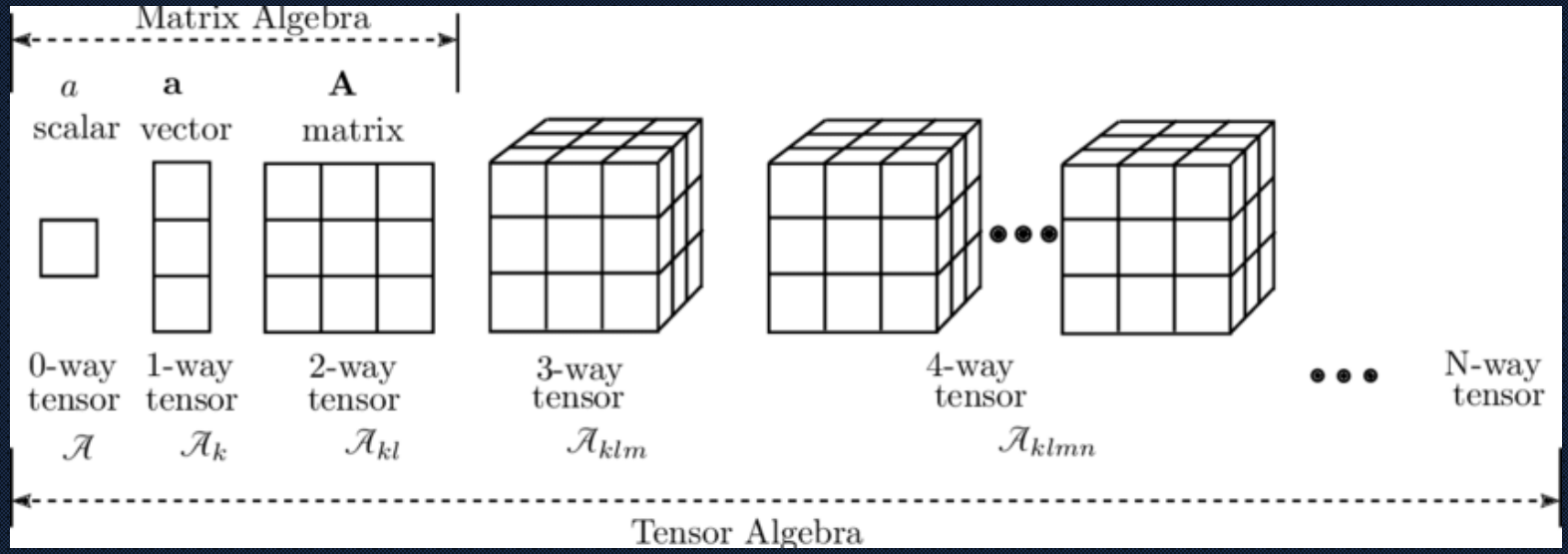
Liste (Array, vector, piles, Tensor, etc...)

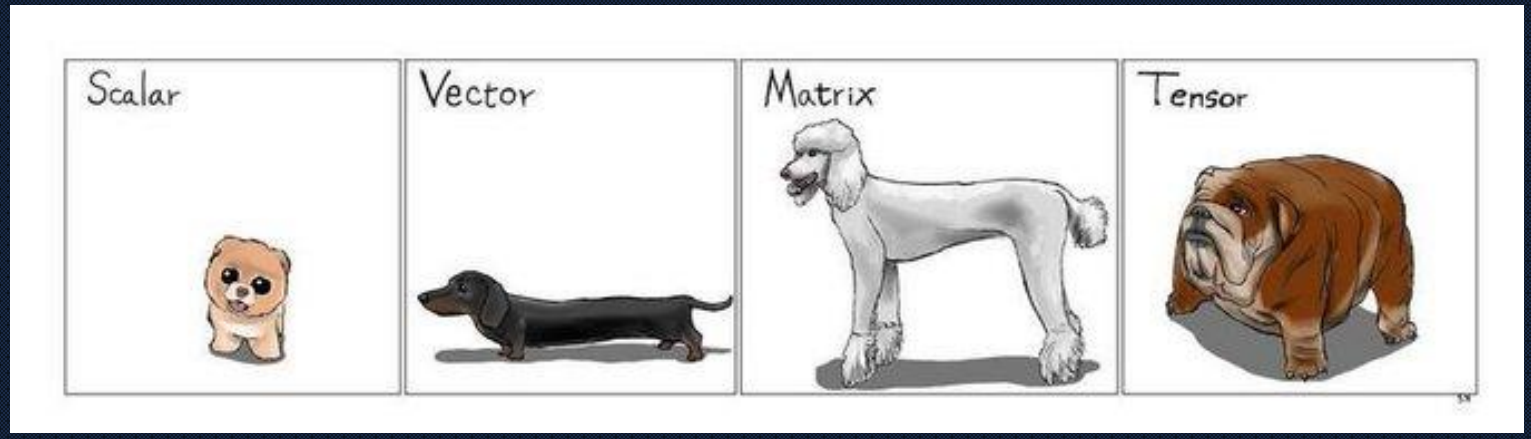
1	2	3	4	5	6	7	8	9	10

Dictionnaire (Objet, Json, BDD etc...)

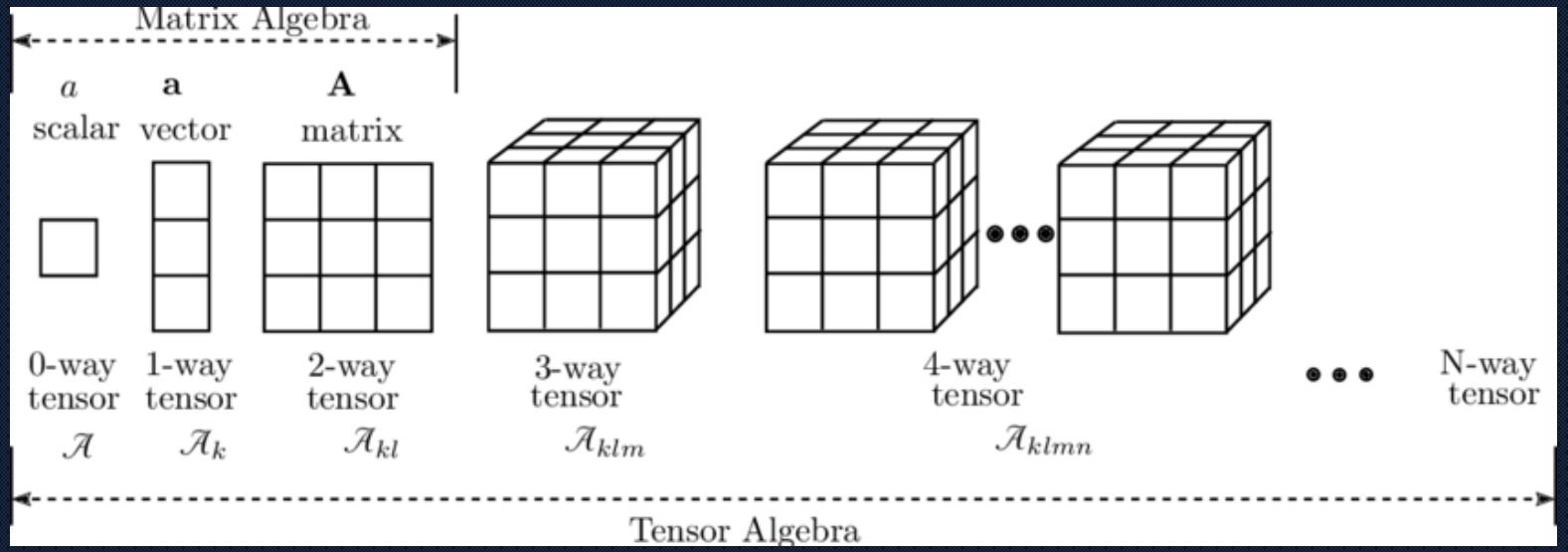
Key	Value
A	0
B	1
C	2
D	3
E	4

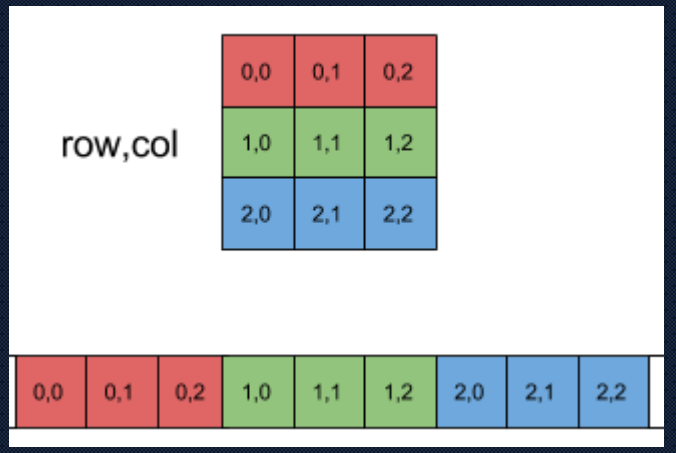
Liste au sens large : vocabulaire





Exercice 1D vers 2D





1	2	3	4	5	6	7	8	9

1	2	3
4	5	6
7	8	9

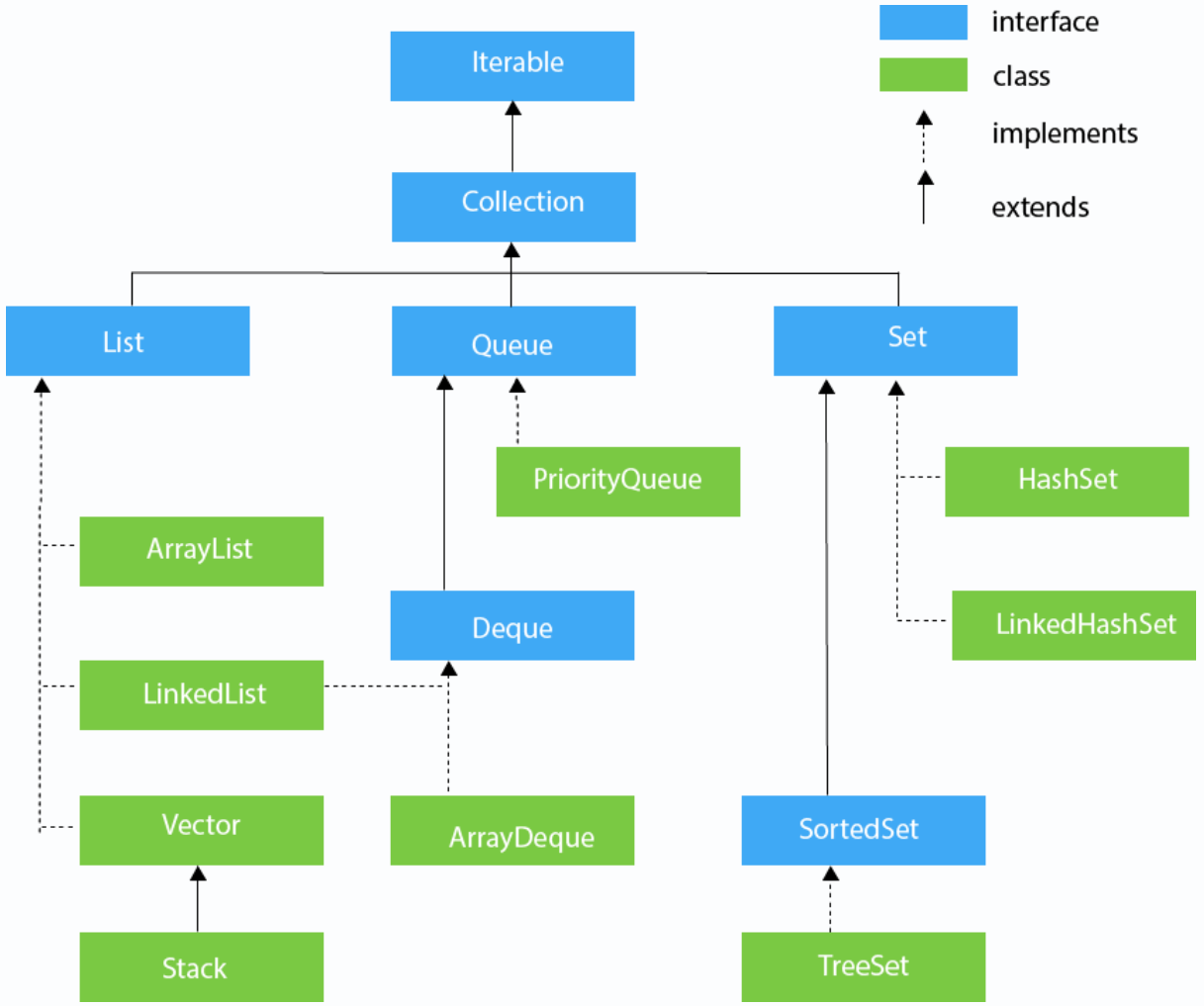
Transformer une liste en matrice.
Note on considère la matrice carrée.
Pour cela codez la fonction suivante :

```
public int get(List<Integer> data , int i, int j, int matrixSize)
```

Questions?

Collections, Arrays, Set et List

Tour d'horizon des types



Iterable et Iterator

Un Iterable n'a qu'une seul fonction abstraite : Iterator<T> iterator() qui renvois un objet de type interface Iterator voici comment est composée cette interface:
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Iterable.html

Method	Description
public boolean hasNext()	It returns true if the iterator has more elements otherwise it returns false.
public Object next()	It returns the element and moves the cursor pointer to the next element.
public void remove()	It removes the last elements returned by the iterator. It is less used.

```
public static void main(String[] args) {
    Iterable<String> iterable = new ArrayList<String>();
    Iterator<String> iterator = iterable.iterator();
    while (iterator.hasNext()) {
        String next = iterator.next();
    }
}
```

Collection

Collection ajoute la notion d'ajout et de suppression d'éléments. En plus de la notion d'Iterable.
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Collection.html

Method	Description
public boolean add(E e)	It is used to insert an element in this collection.
public boolean addAll(Collection<? extends E> c)	It is used to insert the specified collection elements in the invoking collection.
public boolean remove(Object element)	It is used to delete an element from the collection.
public int size()	It returns the total number of elements in the collection.
public void clear()	It removes the total number of elements from the collection.
public boolean contains(Object element)	It is used to search an element.
public boolean isEmpty()	It checks if collection is empty.

List

Les listes ajoutent la notion d'index aux collections.
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/List.html

Method	Description
add(int index, E element)	Inserts the specified element at the specified position in this list (optional operation).
get(int index)	Returns the element at the specified position in this list.
remove(int index)	Removes the element at the specified position in this list (optional operation).
set(int index, E element)	Replaces the element at the specified position in this list with the specified element (optional operation).

List: Exercice

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/List.html
Implémentez l'interface List. Implémentez les méthodes suivantes et mettre des bouchons sur les autres méthodes. Implémentez aussi les fonction de l'interface Iterator. Vérifiez que ça marche avec la syntaxe suivante:

Method
add(int index, E element)
get(int index)
remove(int index)
set(int index, E element)

```
for (String s : iterable) {
    System.out.println(s);
}
```

SET

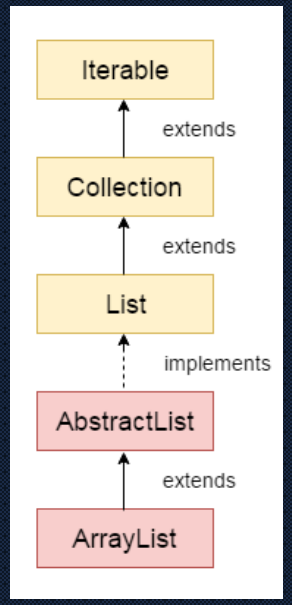
Les « Set » ajoutent la notion d'unicité des éléments.
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Set.html

Method	Description
add(E e)	Adds the specified element to this set if it is not already present (optional operation).

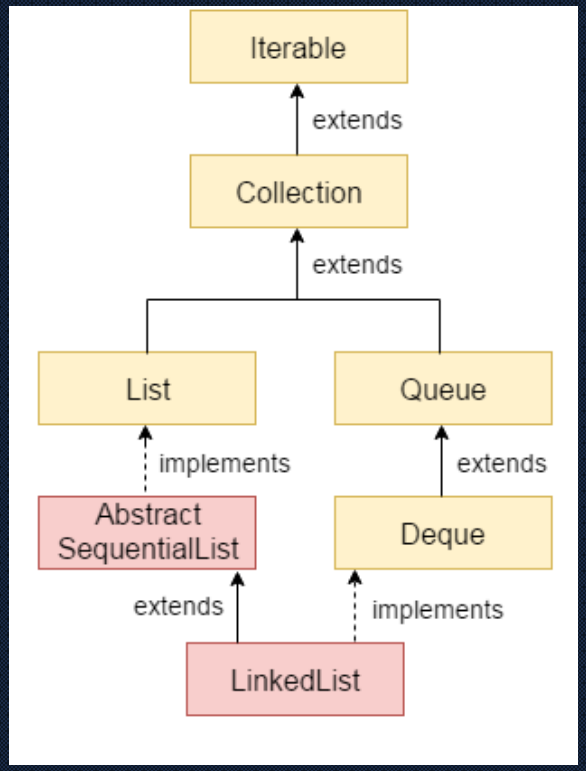
Exercice créer un Set et ajouter les éléments suivant puis afficher le résultat:

1	1	2	1	3	3	4	4	5

Les implémentations de List



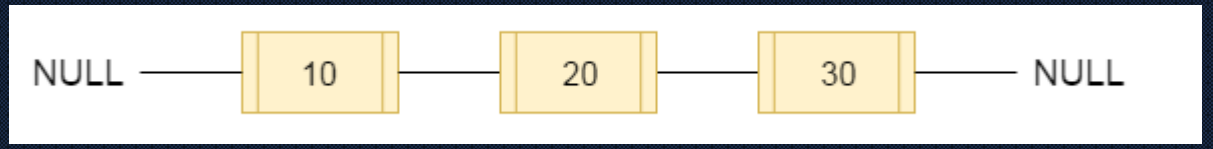
ArrayList



LinkedList

```
int[] array = new int[3];
```

10	20	30



Operations sur les list : Tri

https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html

Example d'utilisation pour trier une liste

Un "Comparator" est une fonction qui prends deux paramètres. Cette fonction est utilise pour déterminer si un Objet est plus grand ou plus petit qu'un autre selon la règle suivante:

Si il retourne un entier négatif => le premier argument est plus petit que le second

Si il retourne 0 => le premier argument est égale au second

Si il retourne un entier positif => le premier argument est supérieur au second

```
List<Integer> data = new ArrayList<>(Arrays.asList(42, 2, 53, 24, 5, 16, 7, 78, 9));
Collections.sort(data, (a, b) -> {
    if (a < b) {
        return -1;
    } else if (a > b) {
        return 1;
    }else{
        return 0;
    }
});
```

Exercice: trier une liste dans l'ordre décroissant

Operations sur les list : Exercice

Compter le nombre d'éléments dans une liste d'entiers qui sont égale à une valeur « numberToCount » donné en paramètre :

```
int count(List<Integer> data, int numberToCount)
```

Operations sur les list : Exercice

Faire une méthode qui cherche dans une liste d'objets celui qui contient un nom donné.

Utilisez les fonctions standards et le code ci dessous.

```
Sample search(List<Sample> data, String name)
```

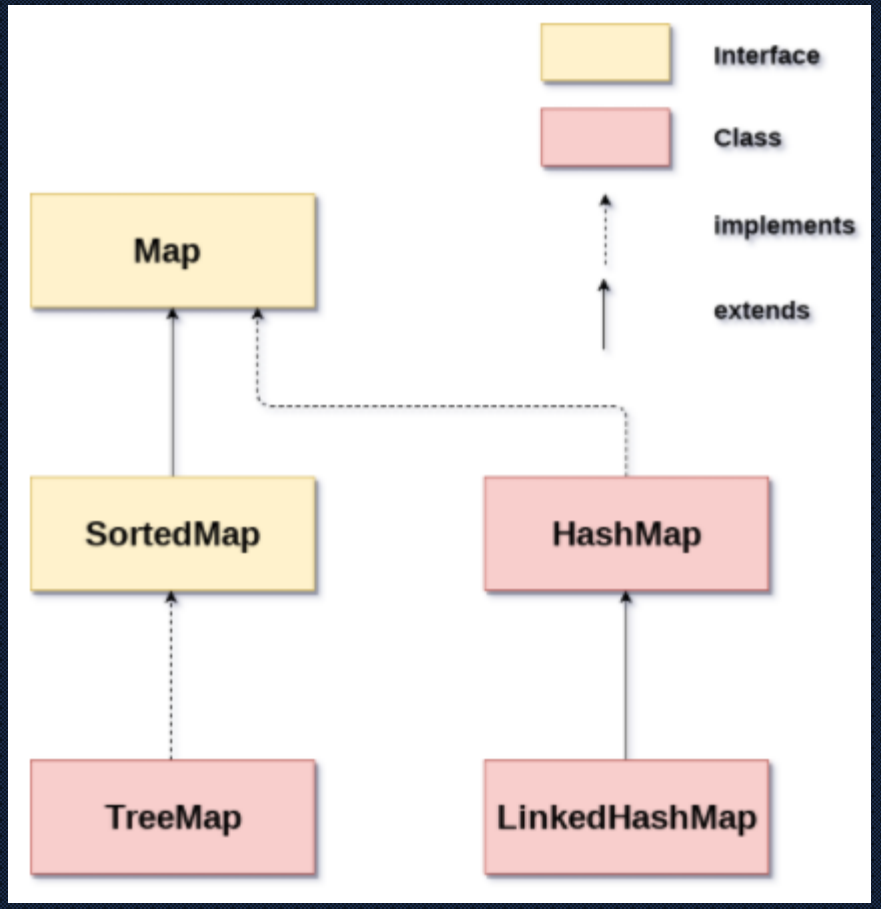
```
public class Sample {
    static List<Sample> createDataSet(){
        List<Sample> ret = new ArrayList<>();
        ret.add(new Sample("Nicholas Douffet",78));
        ret.add(new Sample("Agnès Mailloux",49));
        ret.add(new Sample("Lotye Reault",34));
        ret.add(new Sample("Jeanette Pelletier",24));
        ret.add(new Sample("Pierre Petit",30));
        return ret;
    }

    public String name;
    public int age;

    public Sample(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

Table de hachage HashMap

Map et HashMap

Une Map est un dictionnaire qui associe une définition (Valeur) à un mot (Clé / Key)

Dans les Map les Objets ne sont pas trié par key alors qu'ils le sont dans les SortedMap.

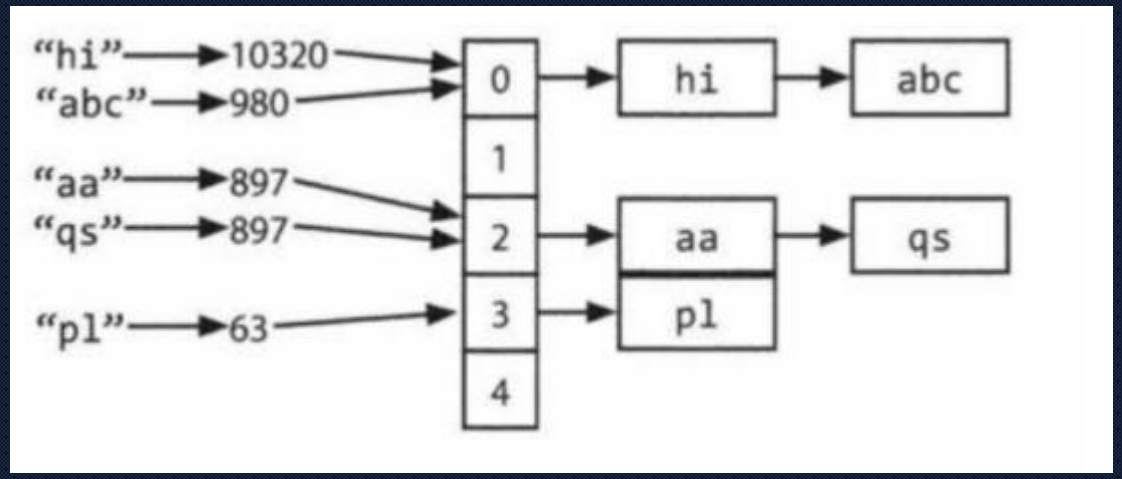


HashMap: comment ça marche

1.Tout d'abord, calculez le code de hachage de la clé, qui sera généralement un int. Les deux objets différents peuvent avoir le même code de hachage, car il peut y avoir un nombre infini d'éléments et un nombre fini d'entiers.

2.Ensuite, calculez l'index dans le tableau à l'aide du code de hachage en utilisant le modulo comme hashCode (key)% array_length. Ici, deux codes de hachage différents peuvent correspondre au même index.

3.Obtenez la liste liée à cet index calculé ci-dessus. Stockez l'élément dans cet index. L'utilisation d'une liste chaînée est importante en raison des collisions : vous pouvez avoir deux clés différentes avec le même code de hachage ou deux codes de hachage différents qui correspondent au même index.



HashMap: Avantage

Aussi rapide qu'une ArrayList et plus souple qu'une LinkedList

Complexité Temporelle

Étant donné que différentes clés peuvent être mappées sur le même index, il existe un risque de collision. Si le nombre de collisions est très élevé, le pire des cas d'exécution est O (N), où N est le nombre de clés. Cependant, nous supposons généralement une bonne implémentation qui réduit les collisions au minimum, auquel cas le temps de recherche est en O(1).

Complexité Spatial

Une ArrayList à un complexité spatial O(N) pour l'ajout d'élément car pour ajouter une donnée à une ArrayList il faut créer un nouveau tableau copier les données de ce tableau dans le nouveau puis supprimer l'ancien. Une HashMap n'a pas ce problème et est O(1) dans ce cas.

Exercice : Interface Set

Method
add(E element)
contains(Object o)
iterator()
remove(Object o)

Implémentez l'interface Set "ensemble" en utilisant un HashMap

https://docs.oracle.com/javase/8/docs/api/java/util/Set.html

Implémentez les méthodes suivantes et mettre des bouchons sur les autres méthodes. Implémentez aussi les fonction de l'interface Iterator.

Expressions régulières en java

Qu’est ce que les expressions régulière

Une expression régulière (regex) définie un motif de recherche pour une chaine de caractères. Ce motif est décrit par une chaine de caractères contenant des caractères spéciaux.

Une regex peut être utilisé pour chercher, éditer et manipuler du texte ont dit que la regex est appliquée à un texte.

Une regex est appliquée sur un texte de gauche à droite. Une fois un motif trouvé il ne peut être réutilisé. Par exemple : la regexp “aba” vas matcher “abababababa” deux fois seulement (aba_aba__)

Où trouver la doc

Doc sur comment utilizer les regex.

https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html

Outils pour tester les regexp

https://www.freeformatter.com/java-regex-tester.html

Exemple avec « . »

“.” Match tous caractères uniques.

Exemple match “.” sur le text “aabb”

Group details:

Match #	Group index	Start index	End index	Group content
1	0	0	1	a
2	0	1	2	a
3	0	2	3	b
4	0	3	4	b

Exemple avec « + »

“+” Match le caractère qui précède une ou plusieurs fois

Exemple match “a+” sur le text “aabb”

Group details:

Match #	Group index	Start index	End index	Group content
1	0	0	2	aa

Comment utiliser les expressions régulières en Java

Les objets String disposent des méthodes suivantes pour une String s.

Method	Description
s.matches("regex")	Evaluates if "regex" matches s. Returns only true if the WHOLE string can be matched.
s.split("regex")	Creates an array with substrings of s divided at occurrence of "regex". "regex" is not included in the result.
s.replaceFirst("regex", "replacement“)	Replaces first occurance of "regex" with "replacement.
s.replaceAll("regex", "replacement“)	Replaces all occurances of "regex" with "replacement.

Exercice regex basics – Est-ce un chiffre?

Implémentez StringUtils.isDigit(String) qui doit retourner « true » si la string qui lui est donnée est un chiffre entre 0 et 9. « false » sinon.

Exercice : Validation d'un code PIN.

Un distributeur permet des code PIN allant de 4 et 6 chiffres. Le code PIN ne peut contenir que des codes PIN d'exactement 4 et 6 chiffres.

Si la « String » passée en paramètre est un PIN valide alors vous retournez « true » sinon vous retournez « false ».

Exemples :

"1234" --> true

"12345" --> false

"a234" --> false

Exercice : Validation d'un nom d'utilisateur.

Ecrire une fonction qui permet de valider un nom d'utilisateur.

Les caractères autorisés sont :

Les lettres minuscules.

Les chiffres.

Les « underscore ».

La taille doit être entre 4 et 16 caractères (inclus).

Exercice : validation d'une heure

Ecrire une fonction qui valide une heure au format 24h.

Exemple:

OK:

- 01h00
- 1h00
- 12h00

Pas OK:

- 24:00

Pas d'espaces.

Exercice : IPv4?

Implémentez une fonction « ipv4_address(String) » qui retourne « true » si elle est au format adresse IPv4. 4 nombres de (0 à 255) séparé par des « . ».

Vous ne devez accepter que les la représentation simple pas de compléments de 0 type « 001 », pas d'espaces.

Exercice : prix

Implémentez la fonction « to_cents(String) » qui prend en entré un prix au format suivant « 1,23€ » ou « 1€23 » et qui doit renvoyer le nombre de centimes ou « null » si la « string » est mal formatée.

Exercice : Censurer une phrase dans un texte

Censurer toutes les phrases qui continent « ananas » et « pizza ».

Examples :

“J’ai mange une pizza ananas." --> “***"

“J’adore la pizza à l’ananas." --> “***"

“Pizza et ananas sont un super mélange." --> “***"

“L'eau ça mouille. J’adore la pizza à l’ananas. Les chats c’est fun. " --> “L'eau ça mouille.***. Les chats c’est fun."

Bonus replacer la phrase par un nombre d’étoiles correspondants aux nombres de caractères dans la phrase.

Fonctions Lambda

Fonctions lambda

Le but d'une fonction lambda est de simplifier la syntaxe du java elle n'ajoute aucun avantage fonctionnels (rapidité d'exécution ou autre)

Fonctions lambda: Syntaxe

La syntaxe d'une expression lambda est composée de trois parties :

1. un ensemble de paramètres, d'aucun à plusieurs
2. l'opérateur ->
3. le corps de la fonction

Elle peut prendre deux formes principales :

1. (paramètres) -> expression;
2. (paramètres) -> { traitements; }

L'écriture d'une expression lambda doit respecter plusieurs règles générales :

1. zéro, un ou plusieurs paramètres dont le type peut être déclaré explicitement ou inféré par le compilateur selon le contexte
2. les paramètres sont entourés par des parenthèses et séparés par des virgules. Des parenthèses vides indiquent qu'il n'y a pas de paramètre
3. lorsqu'il n'y a qu'un seul paramètre et que son type est inféré alors les parenthèses ne sont pas obligatoires
4. le corps de l'expression peut contenir zéro, une ou plusieurs instructions. Si le corps ne contient d'une seule instruction, les accolades ne sont pas obligatoires et le type de retour correspond à celui de l'instruction. Lorsqu'il y a plusieurs instructions alors elles doivent être entourées avec des accolades

Fonctions lambda : Comment les utiliser

A utiliser lorsque la fonction demande en paramètre une interface fonctionnelle (Une interface qui ne spécifie qu'une seule méthode) :

```
public interface Comparator<T> {
    int compare(T o1, T o2);
}
```
java.util.Comparator

```
public interface Runnable{
    void run();
}
```
java.lang.Runnable

```
public interface ActionListener extends EventListener{
    void actionPerformed(ActionEvent e);
}
```
java.awt.event.ActionListener

```
public interface Callable<V>{
    V call();
}
```
java.util.concurrent.Callable

```
public interface PrivilegedAction<V>{
    T run();
}
```
java.security.PrivilegedAction

Fonctions lambda : Exemples

```
BiFunction<Integer, Integer, Long> additionner = (val1, val2) -> (long) val1 + val2;

Consumer<String> afficher = (String param) -> System.out.println(param);

Runnable monTraitement = () -> {
  System.out.println("traitement 1");
  System.out.println("traitement 2");
};

Consumer<int> impair = n -> n % 2 != 0;
Runnable mon123 = () -> 123 ;
Runnable mon123 = () -> { return 123};
```

Exercice List et Lambda

1- Créer une liste d'entier aléatoire.

2 - Trier une liste d'entiers répartis de manière aléatoire en utilisant une lambda pour le Comparator et pour afficher la liste.

Présentation du projet

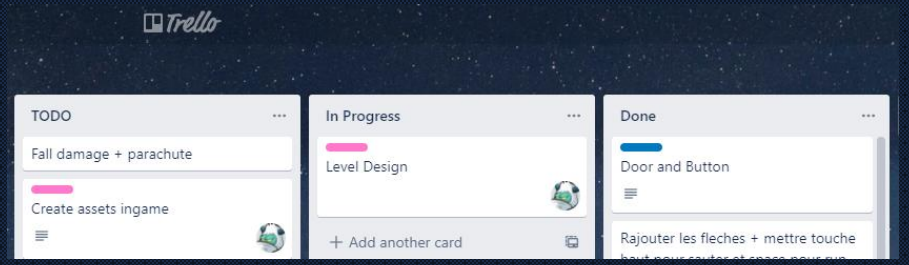
Smart Webcam

Le client souhaite avoir un logiciel qui quand il prend les images de la webcam les analyses et donne une information sur ce qui est présent à l'écran. (Comme le fait par exemple l'application google Lense)

Contraintes techniques

- Développement en java.
- Projet gradle.
- Commentaire et/ou code self-explanatory.
- Le code doit être mis sur Github.

Recommandations organisationnelles



- Utilisation d'un Trello.
- Groupe de 3 personnes.
- Des commits régulier un par story par exemple. Toutes les personnes du groupe font des commit.

Notation

Des points sont donnés pour chaque story remplie.

Le plagia évident (j'ai accès au Github n'oubliez pas) et les problèmes de groupe (coopération etc..) si j'ai besoin d'intervenir seront pénalisés.

Points complémentaires seront attribués sur:

1) Clarté du code. (Utilisation de fonctions commentaire etc…)
2) Clarté des commits. (Commentaire de commits. Pas tous les commits au dernier moment. Pas une personne qui fait tout les commit)

La notation pourra être ajustée mais cela vous donne un ordre d'idée.

Ce qui est attendu

1) Préparer la démo de chaque story que vous aurez réalisée (~20 min par groupe) pour vendredi.
2) Un lien de votre GitHub.

Comment utiliser TensorFlow et le réseau de neurones

1) Le réseau de neurones que nous allons utiliser permet de classifier les images.
2) Il prend en entrée une image sous forme de Tensor. Voir la méthode TFUtils.byteBufferToTensor. Le premier paramètre représente les octets contenues dans le fichier image au format JPEG.
3) Il donne en sortie une liste de probabilité. Chaque probabilité de cette liste correspond à un label présent dans le fichier « inception5h/labels.txt ».
4) Le label avec la plus haute probabilité correspond à l'image. Une probabilité proche de 100% indique que le RDN est sûr du résultat.
5) Vous pouvez exécuter le model du RDN en utilisant la fonction TFUtils. executeModelFromByteArray(graphDef, input) Le paramètre graphDef correspond aux bytes contenue dans le fichier. « inception5h/tensorflow_inception_graph.pb ». Le param input correspond à l'image à classifier au format Tensor.
6) Des images d'exemple sont présentes dans le dossier « tensorPics » elles fournissent une classification correcte avec ce réseau de neurones.
7) Importer avec gradle la dépendance suivante:

-
```
compile group: 'org.tensorflow', name: 'tensorflow', version: '1.15.0'
```

Stories

Story 1: Détecter le type d'une image

EN TANT QUE PO

JE VEUX pouvoir détecter le type d'une image donnée en utilisant le réseau de neurones fournit (TensorFlow)

AFIN DE vérifier que le fonctionnement du réseau de neurone est bien compris

Definition of Done

Charger une image (prendre les images d'exemple) et renvoyer la description de cette image dans la console.

Story 2 : Affichage

EN TANT QUE PO

JE VEUX pouvoir afficher l'image et sa description/classification dans une fenêtre java FX.

AFIN D' Avoir un meilleur affichage du résultat du réseau de neurones.

Definition of Done

Une fenêtre s'ouvre avec une image et en dessous la classification de l'image.

JavaFX: Configuration gradle

```
plugins {
    id 'java'
    id 'application'
    id 'org.openjfx.javafxplugin' version '0.0.9'
}

mainClassName = 'launcher.Launcher'
group 'org.example'
version '1.0-SNAPSHOT'

repositories {
    mavenCentral()
}

javafx {
    version = "15.0.1"
    modules = [ 'javafx.controls','javafx.swing' ]
}

dependencies {
    testCompile group: 'junit', name: 'junit', version: '4.12'
}
```

Story 3 : Sauvegarde et sélection des images

EN TANT QUE client

JE VEUX Enregistrer les images qui correspondent à la description donnée dans un dossier.

AFIN DE sauvegarder les résultats du logiciel.

Definition of Done

La définition doit être configurable dans un text field. Le pourcentage de reconnaissance et le fichier où sauvegarder doivent aussi être configurable. Les labels sont composés de plusieurs mots. La description est bonne si un des mots de la description est valide.

Ajouter un paramètre pour limiter la sauvegarde à une toute les X secondes.

Story 4 : Sélectionner la source

EN TANT QUE client

JE VEUX pouvoir sélectionner la source de l'analyse du fichier soit image (avec un file browser), soit la caméra.

AFIN DE simplifier l'utilisation du logiciel

Definition of Done

Pouvoir sélectionner la source de l'analyse du fichier soit image (avec un file browser. Soit la camera (seulement dans le cas où la story Webcam est faite).

Story 5 : Utilisation de la webcam

EN TANT QUE client

JE VEUX Afficher la description/classification de ce que voit la webcam.

AFIN DE visualiser le résultat de ce que pense voir la webcam.

Definition of Done

Une fenêtre s'ouvre avec le flux de la webcam affiché en direct et en dessous la classification de l'image qui évolue en direct (par exemple mise à jour 1 fois par seconde) en fonction du flux d'image de la webcam.

Utilisation de la webcam

Vous pouvez utiliser la Library JavaCV pour gérer la webcam. Voir http://bytedeco.org/javacv/apidocs/org/bytedeco/javacv/OpenCVFrameGrabber.html

Ajouter la dep graddle suivante :

```
implementation group: 'org.bytedeco', name: 'javacv-platform', version: '1.5.4'
```

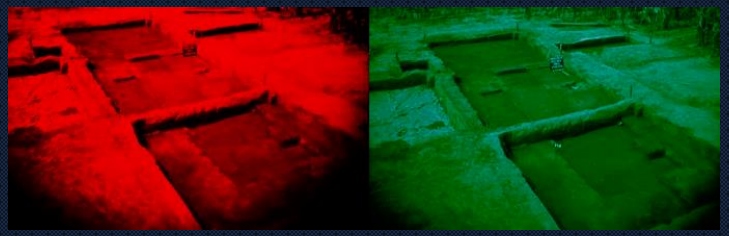
Pour convertir une Frame en BufferedImage vous pouvez utiliser: Java2DFrameConverter

http://bytedeco.org/javacv/apidocs/org/bytedeco/javacv/Java2DFrameConverter.html

Pour convertir une BufferedImage en Image (utilisable par JavaFX) vous pouvez utiliser SwingFXUtils

https://docs.oracle.com/javafx/2/api/javafx/embed/swing/SwingFXUtils.html

Story 6: Appliquer un style en fonction de la classe trouvé



EN TANT QUE client

JE VEUX Un menu qui me permette d'appliquer automatiquement un filtre de couleur sur l'image avant de l'enregistrer.

AFIN DE modifier dynamiquement les images capturé par la caméra.

Definition of Done

Dans ma fenêtre Java FX j'ai une liste dans laquelle je peut ajouter un couple « nom de classifier / filtre à appliquer ». Le filtre est un filtre de couleur simple avec un paramètre JavaFX pour choisir le type de couleur.

Story 7: Plus de style

EN TANT QUE client

JE VEUX Ajouter à ma liste de styles possible le fait d'ajouter un cadre sur mon image (Image PNG) a coller sur l'image de base

AFIN DE pouvoir ajouter un style cadre sur mon image.

Definition of Done

Dans ma fenêtre Java FX j'ai une liste dans laquelle je peut ajouter un couple « nom de classifier / filtre à appliquer ». En plus du filtre de couleur je peut ajouter un autre style de type cadre qui prend en paramètre l'image à coller.

Story 8: Plus de style 2

EN TANT QUE client

JE VEUX Ajouter à ma liste de styles possible le fait d'ajouter un cadre qui colle une image sur l'image source à une coordonné en pixel donnée.

AFIN DE pouvoir ajouter un style cadre sur mon image.

Definition of Done

Dans ma fenêtre Java FX j'ai une liste dans laquelle je peut ajouter un couple « nom de classifier / filtre à appliquer ». En plus des autres filtres je peut ajouter un autre style de type tampon qui prend en paramètre l'image à coller et un position x et y sur l'image de destination.

Have Fun