24/03/2023
Version 3

JAVA

JÉRÉMY PERROUAULT

INTRODUCTION

LES BASES DE L'OBJET

INTRODUCTION

- En **JAVA**, chaque classe créée doit l'être dans un *package*
  - Si on ne précise pas le nom d'un *package*, c'est le *package* **default** qui sera utilisé
    - Laisser ce *package* par défaut est mauvais dans la plupart des architectures
      - Pour des questions d'organisation du code / relecture
      - Pour la maintenance
      - Pour l'implémentation de **Frameworks** qui se basent sur les règles et conventions **JAVA**

INTRODUCTION

- On peut créer autant de *packages* que nécessaire
  - Généralement, une application a un *package* **racine**, dans lequel se trouve les autres *sous-packages*

- Dans cet exemple, le *package* **racine** est `fr.formation`, et les *sous-packages* sont
  - `model` et `model/enumerator`
  - `repository`
  - `service`

```
package fr.formation.model;
```

```
package fr.formation.repository;
```

```
package fr.formation.model.enumerator;
```

```
package fr.formation.service;
```

INTRODUCTION

- Les *packages* fonctionnent comme des organisations de répertoires
  - Répertoires et sous-répertoires à plusieurs niveaux
- Ils sont d'ailleurs enregistrés comme ça dans l'arborescence du projet

- On peut avoir plusieurs **classes JAVA** qui portent le même nom dans le même projet
  - En fait, le nom de la **classe JAVA** complète est `nom.package.NomClasse`

```
fr.formation.model.Produit
```

- Si on veut utiliser une **classe JAVA** qui est dans le même *package*
  - Rien à faire
- Si on veut utiliser une **classe JAVA** qui est dans un autre *package* (même un *sous-package*)
  - Il faut importer la **classe JAVA** et son *package*

```
import fr.formation.model.Produit
```

INTRODUCTION

- La ligne du nom du *package* est à mettre à la toute première ligne du fichier **classe JAVA**
  - Viennent ensuite les *imports*
  - Puis la description de la **classe JAVA**

```
package fr.formation.model;

import java.util.List;
import java.util.Optional;

public class Produit {
}
```

INTRODUCTION

- Quelques règles de nommage, convention
  - Le nom du fichier *.java* est strictement le nom de la **classe JAVA**
  - Le nom de la **classe JAVA** s'écrit en `CamelCase`, sans espaces ni caractères spéciaux
  - Le nom du *package* s'écrit en `minuscule`, sans espaces ni caractères spéciaux

INTRODUCTION

- Les **classes** mises à disposition par **JAVA** se trouvent dans des *packages* particuliers
  - Pour pouvoir manipuler ces classes, il faut **importer** le ou les packages concernés
    - Seul le *package* `java.lang` de **JAVA** est implicitement importé
  - D'une manière générale, si la classe n'est pas accessible MAIS qu'elle existe belle et bien
    - Les **IDE** vous proposent de l'importer pour la manipuler

```
import java.util.ArrayList;

import java.util.*;
```

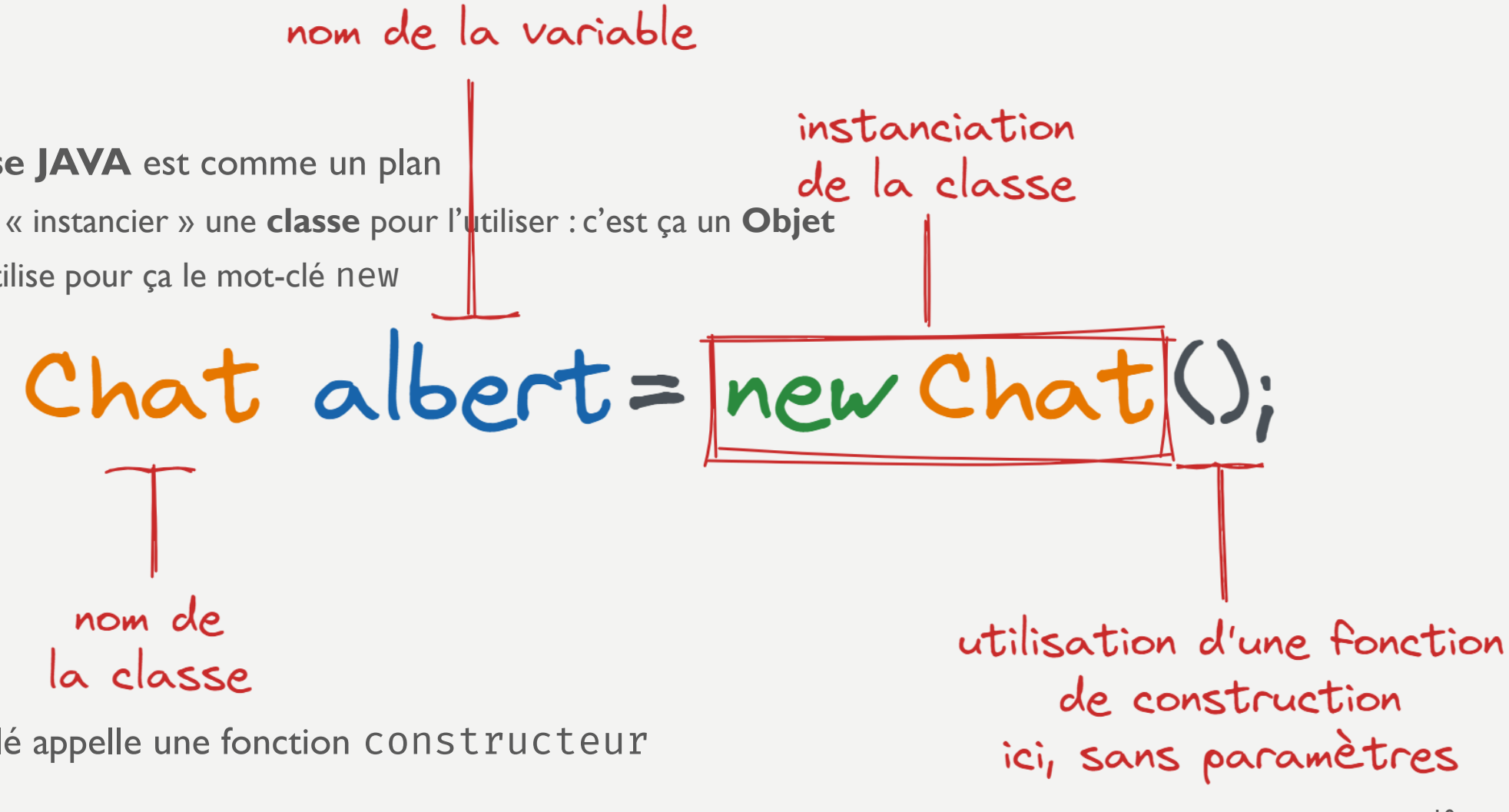
INTRODUCTION

POO	Réalité
Classe	Plans pour fabriquer un chat
Objet (instance de classe)	Le chat
Propriétés / attributs	Attributs du chat (nom, race, …)
Méthodes / comportement	Comportements du chat, ce qu'il peut faire

Objet	Classe
Objet	Instance de classe
Type d'objet	Le nom de la classe

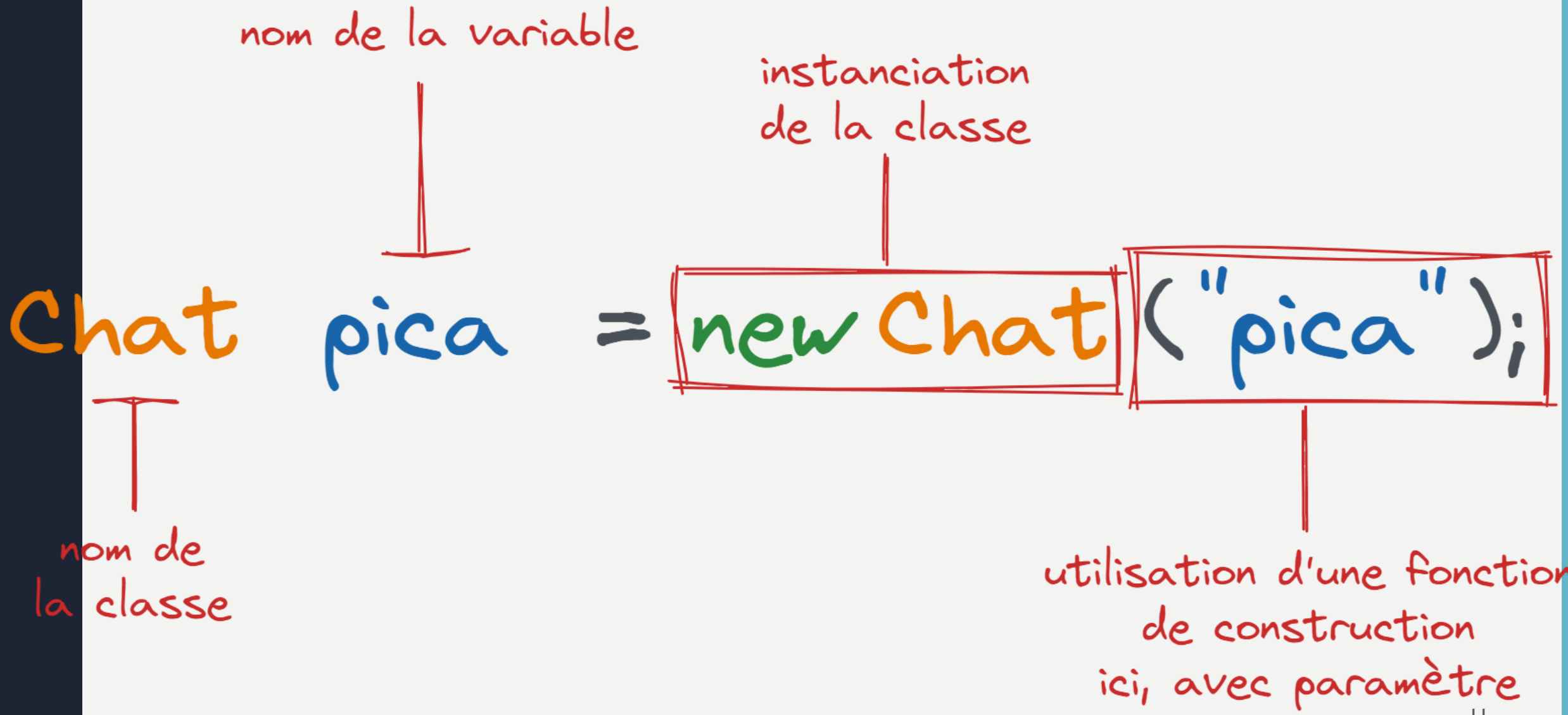
INTRODUCTION

- Une **classe JAVA** est comme un plan
  - Il faut « instancier » une **classe** pour l'utiliser : c'est ça un **Objet**
  - On utilise pour ça le mot-clé `new`



- Ce mot-clé appelle une fonction `constructeur`

INTRODUCTION



INTRODUCTION

Type algorithmique	Type (primitif) JAVA	Classe JAVA
Entier très Court	byte	(java.lang.)Byte
Entier Court	short	(java.lang.)Short
Entier	int	(java.lang.)Integer
Entier Long	long	(java.lang.)Long
Réel	float	(java.lang.)Float
Réel Long	double	(java.lang.)Double
Caractère	char	(java.lang.)Character
Booléen	boolean	(java.lang.)Boolean
Chaine		(java.lang.)String

INTRODUCTION

Classe JAVA	Description
(java.lang.)System.out	Sortie du système (console)
(java.lang.)System.in	Entrée du système (clavier)
java.util.Scanner	Scanner un flux (clavier par exemple)
java.util.ArrayList	Collection d'objets sous forme de tableau dynamique
java.util.LinkedList	Collection d'objets sous forme de liste chainée
java.util.HashMap	Liste de clés-valeurs (ou tableau associatif)
java.util.HashSet	Liste de valeurs sans doublon
java.util.Collections	[Statiques] Fonctionnalités liées aux collections

INTRODUCTION

- Dans un projet **JAVA**, il faudra une classe principale (`Application` par exemple)
  - Qui aura une méthode `main`, publique et statique
  - Et qui attend une tableau de `String` en argument

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

EXERCICE

- Avec **Eclipse**, créer un nouveau projet **JAVA**
  - Créer une classe principale `Application`
    - *Package* `fr.formation`
    - Dans la méthode `main`, reprendre l'algo précédent (création des 32 cartes et affichage)
  - Créer les classes `Joueur`, `Equipe` et `Carte`
    - *Package* `fr.formation.jeu`

COLLECTIONS

LES DIFFÉRENTES COLLECTIONS

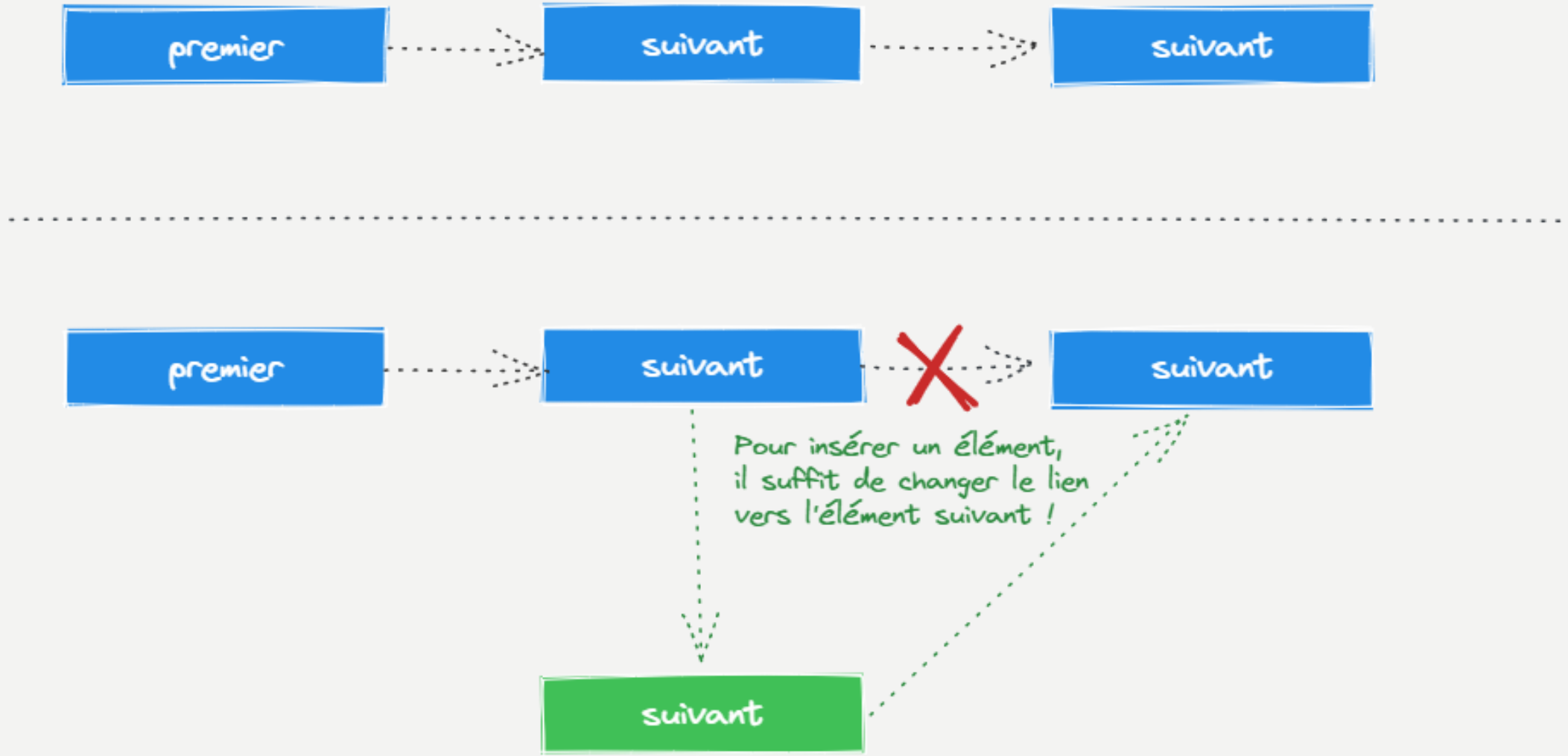
LES COLLECTIONS

- On distingue plusieurs catégories de `Collection`
  - `List`
    - `ArrayList`
    - `LinkedList`
  - `Map`
    - `HashMap`
    - `LinkedHashMap`
  - `Set`
    - `HashSet`
    - `LinkedHashSet`

LA LISTE CHAINÉE (LINKED)

- Très rapide pour insérer et supprimer des données
  - Il n'y a pas de tableau
  - Pas besoin de créer un décalage
- Plus lente pour une lecture
  - Il n'y a pas de tableau
  - Parcourir chaque élément jusqu'à trouver le bon élément

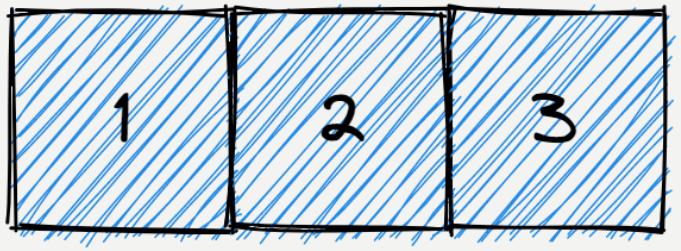
LA LISTE CHAINÉE (LINKED)

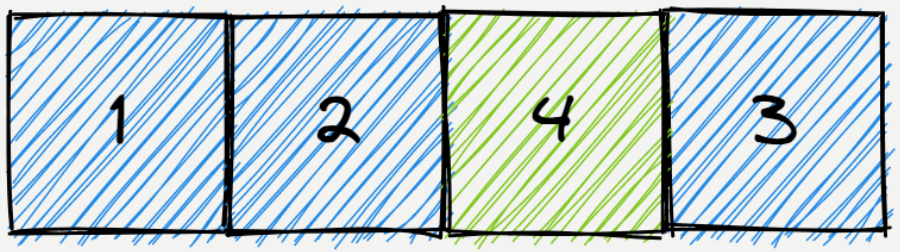


LE TABLEAU DYNAMIQUE (ARRAY)

- Très rapide à lire et à parcourir
  - C'est un tableau, accès direct via l'indice
- A éviter si la liste est trop variable
  - Beaucoup d'insertion / de suppression à faire dans un laps de temps court
  - L'allocation d'un nouveau tableau est faite à chaque ajout / suppression

LE TABLEAU DYNAMIQUE (ARRAY)



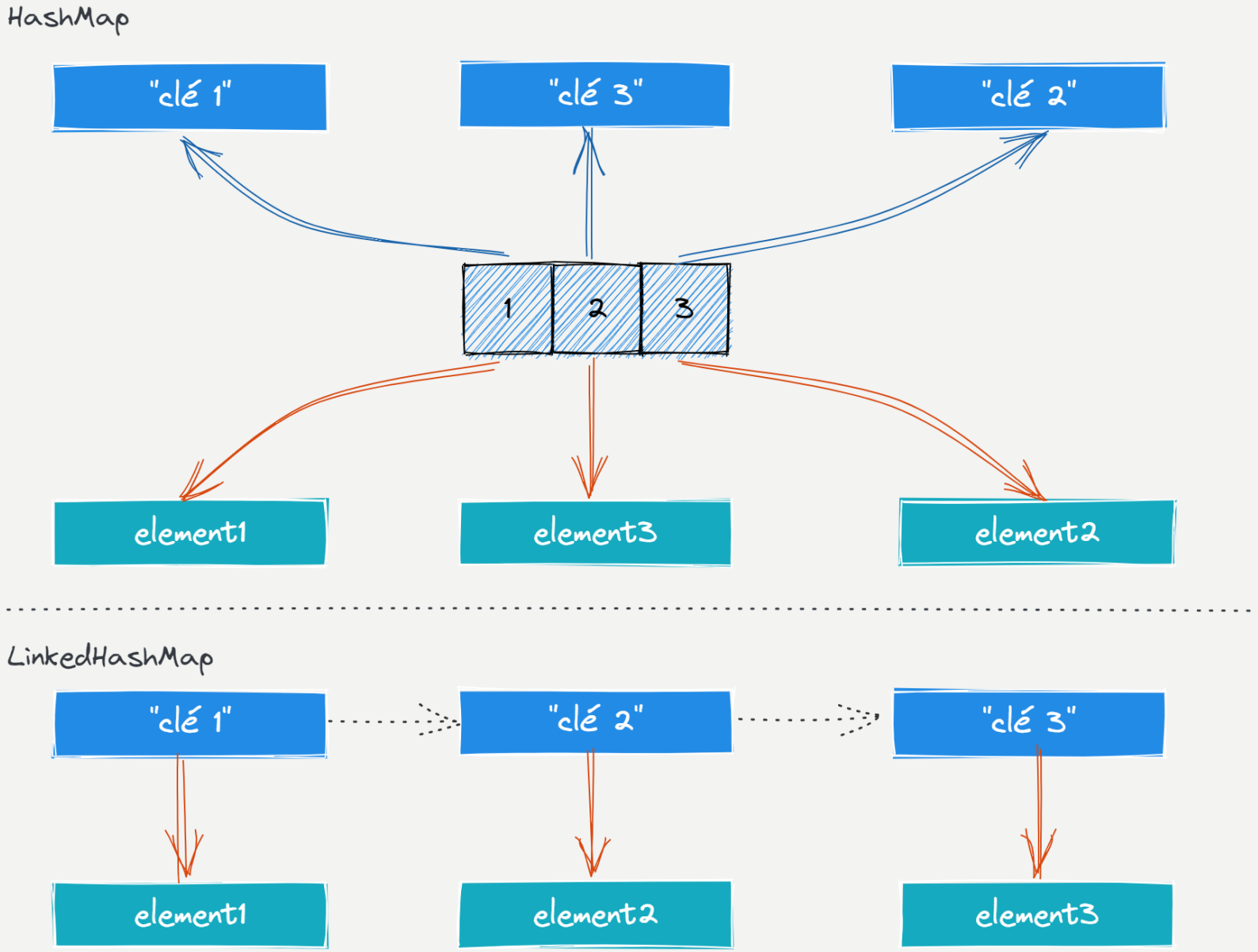


Pour insérer un élément,
il faut créer un nouveau tableau
et déplacer les éléments existants !

LE TABLEAU ASSOCIATIF (MAP)

- Fonctionne avec un couple clé-valeur
  - Comme si l'indice d'un tableau devenait par exemple une chaine de caractère (ou autre)
    - monTab["jeremy"] au lieu de monTab[5]
  - Une seule clé unique
    - Mais une valeur peut être affectée à plusieurs clés différentes
- Bien plus gourmand en mémoire
  - Il faut stocker la clé en plus de la valeur
- Plus lent pour une lecture
  - Il faudra parcourir toutes les données jusqu'à trouver la clé recherchée
- `LinkedHashMap` permettra de garder l'ordre d'insertion

LE TABLEAU ASSOCIATIF (MAP)



LE TABLEAU SANS DOUBLON (SET)

- Derrière le tableau dynamique il y a un tableau
- Derrière le tableau sans doublon il y a un tableau associatif
- Permet de ne pas autoriser les valeurs en double (références ou `Comparables`)
- Pas d'accès direct à une valeur

PROGRAMME

EN JAVA OBJET

ECRIRE UN PROGRAMME

- Les sous-programmes s'appellent désormais des `méthodes`
  - Elles sont toutes dans une **classe JAVA**
- Chaque `méthode` doit avoir une portée définie
  - `public` accessible par tous
  - `private` accessible uniquement par la classe l'ayant définie (sa propre méthode)
  - `protected` accessible par la classe l'ayant définie, et ses enfants (principe d'héritage)

```
public void maMethode() { }

private void maMethodeInterneQueYaQueMoiQuiYaAcces() { }
```

  - Si la portée n'est pas précisée, c'est une portée `package` qui est utilisée par défaut (à éviter)

ECRIRE UN PROGRAMME

- Le mot-clé `this` désigne l'instance en cours de manipulation
  - `this` donne accès à tous les attributs et à toutes les méthodes de l'instance

ECRIRE UN PROGRAMME

- Déclaration d'une **classe** (un fichier par classe)

```
public class Produit { }
```

- Déclaration des attributs

```
public class Produit {
    private String nom;
    private float prix;
}
```

! Tout attribut doit être privé ou protégé !

ECRIRE UN PROGRAMME

- Déclaration des accès aux attributs
  - En lecture et/ou en écriture
  - `Getters` / `Setters`
  - Principe d'encapsulation

```java
public class Produit {
  private String nom;
  private float prix;

  public String getNom() {
    return this.nom;
  }

  public void setNom(String nom) {
    this.nom = nom;
  }

  public float getPrix() {
    return this.prix;
  }

  public void setPrix(float prix) {
    this.prix = prix;
  }
}
```

EXERCICE

- Dans le programme principal
  - Modifier l'âge d'un joueur existant : donner un âge négatif …
  - … Puis, faire en sorte de verrouiller cette possibilité : l'âge doit être strictement supérieur à 0 !

- Mettre en place les modifications pour encapsuler les toutes les données
  - Attributs privés
  - Ajouter les `getters` et/ou `setters` partout où c'est nécessaire
  - Faire les autres modifications dans le programme principal

Les **`getters`** et **`setters`** peuvent être générés depuis **`Eclipse`**
`> Clique-droit > « Source » > « Générer les getters/setters »`