24/03/2023
Version 3

OBJET

JÉRÉMY PERROUAULT

INTRODUCTION

INTRODUCTION À L'OBJET

PROBLÉMATIQUE

- Projection de la réalité difficile, on aimerait pouvoir regrouper des entiers, des chaines de caractères, des booléens, … Bref, regrouper des infos pour en constituer une structure plus ou moins complexe
  - Et tant qu'à y être, les ordonner de façon logique

```
int[] lesAges;
char[][] lesNoms;
char[][] lesPrenoms;
```

- Réutiliser des algorithmes écrits par d'autres (ou pas) de façon simplifiée
  - Pourquoi pas masquer une complexité ?

IDÉE

- Créer un type de variable (appelé **Structure**) qui s'utilisera comme un type classique
  - On utilise le mot-clé `nouvelle` ou `nouveau` pour créer une nouvelle **Structure**
  - Toutes les variables dans la structure sont accessibles publiquement
  - L'accès se fait via le point (« . ») sur une variable de type **Structure**

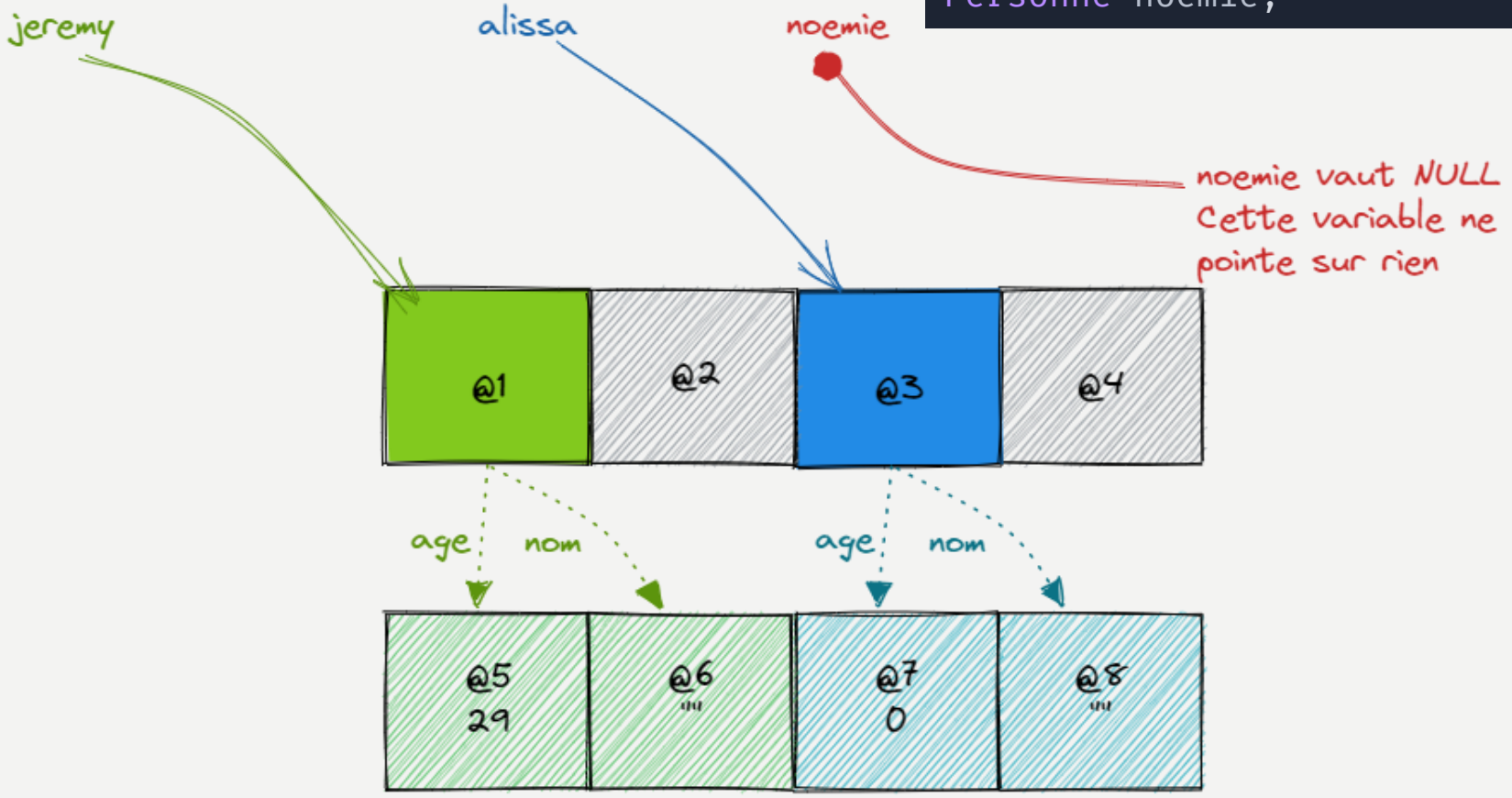
```
structure Personne {
  char[] prenom;
  char[] nom;
  int age = 0;
}
```

```
Personne maPersonne = new Personne();

maPersonne.age = 29;
```

EN PRATIQUE

```
structure Personne {
  char[] nom;
  int age = 0;
}
```

```
Personne jeremy = new Personne();

jeremy.age = 29;

Personne alissa = new Personne();
Personne noemie;
```



SPÉCIALISATION

- Si on veut faire des actions pour une `Personne`
  - Créer un programme qui attend un paramètre de type `Personne` pour le manipuler

```
boolean isPersonneChild(Personne p) {
  if (p.age < 18) {
    return true;
  }

  return false;
}
```

SPÉCIALISATION

- Et si on allait plus loin ? Si on créait une **structure** qui a
  - Ses propres données (comme c'est le cas actuellement)
  - Ses propres fonctionnalités, son propre comportement ?

- Et on appellerait ça une **classe**
  - Et ça encapsulerait (masquerait) toute la complexité interne du fonctionnement de la classe
    - On gagne en intégrité, en lisibilité et en sécurité

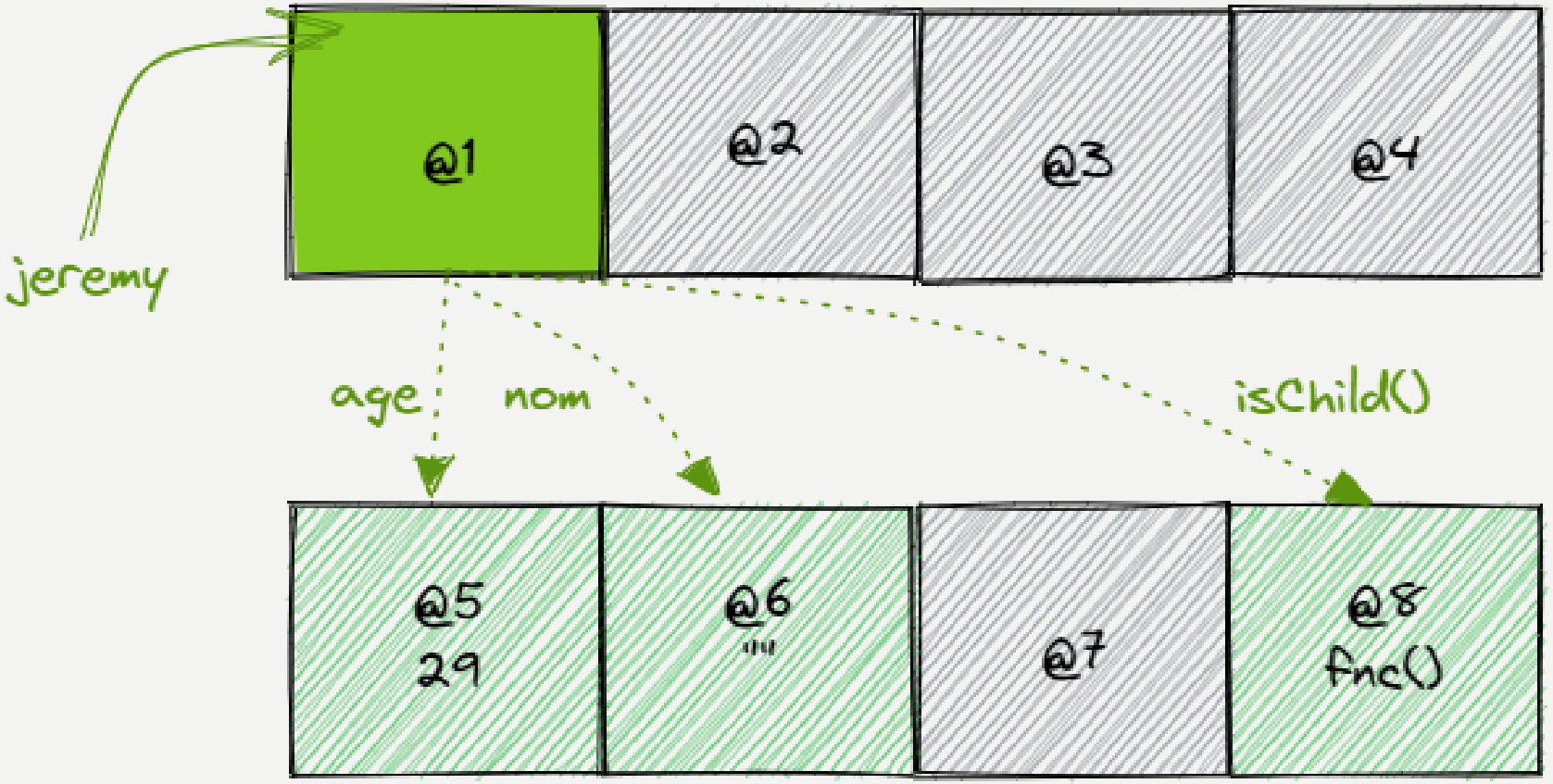
IDÉE

- Créer un type de variable (appelé **Classe**) qui s'utilisera comme un type classique
  - Et qui en plus d'avoir ses propres données (attributs), aura son propre comportement
  - Utilisation du mot-clé `this` dans une classe pour faire référence à ses propres données ou programmes

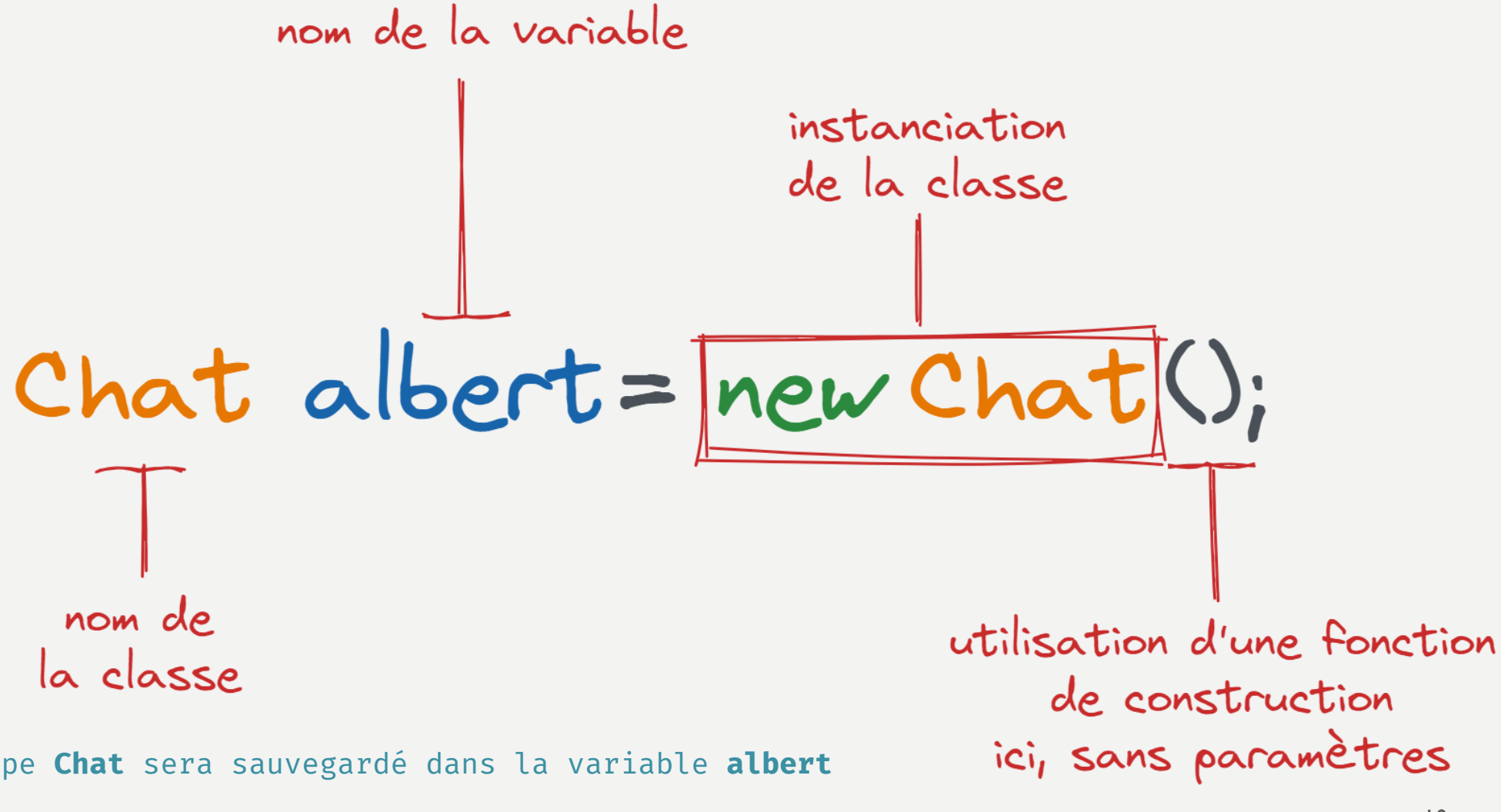
```
class Personne {
  char[] prenom;
  char[] nom;
  int age = 0;

  booleen isChild() {
    if (this.age < 18) {
      return true;
    }
    return false;
  }
}
```

```
Personne maPersonne = new Personne();

maPersonne.age = 29;

if (maPersonne.isChild()) {
  System.out.println("C'est un enfant !!");
}
```

EN PRATIQUE



EN PRATIQUE



L'objet de type **Chat** sera sauvegardé dans la variable **albert**

EN PRATIQUE

POO	Réalité
Classe	Plans pour fabriquer une personne
Objet (instance de classe)	La personne
Propriétés / attributs	Attributs de la personne (nom, prenom, age, …)
Méthodes / comportement	Fonctionnalités de la personne, ce qu'elle peut faire

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

EXERCICE

- Créer une **classe** `Joueur`
  - Age
- Instancier un nouveau `Joueur`

LES CHAINES DE CARACTÈRES

- Une chaine de caractères est un tableau de caractères
- Introduction d'une classe `String` qui masquera la complexité de ce tableau
  - C'est un des objectifs de l'objet !

```
String maPhrase = "Une phrase !";
```

- Avec la possibilité de concaténer une ou plusieurs chaines

```
String monIntro = "Bonjour, ";
String prenom = "Jérémy";

String maPhrase = monIntro + prenom + " !";
```

```
String monIntro = "Bonjour, ";
String prenom = "Jérémy";

if (monIntro.equals(prenom)) {
  //...
}
```

- Ou de réaliser d'autres opérations, comme comparer 2 chaines

LES CHAINES DE CARACTÈRES

- Pour comparer des chaines de caractères

```java
Scanner sc = new Scanner(System.in);

String demo = "demo";
String saisie = sc.nextLine();

if (demo == saisie) {
  // Ne fonctionne pas
}

if (demo.equals(saisie)) {
  // Fonctionne !
}
```

EXERCICE

- Modifier la **classe** `Joueur`
  - Nom, Age
- Instancier un nouveau `Joueur`
  - Affecter son `nom` et son `age` (avec saisis utilisateur)
  - Afficher ses informations

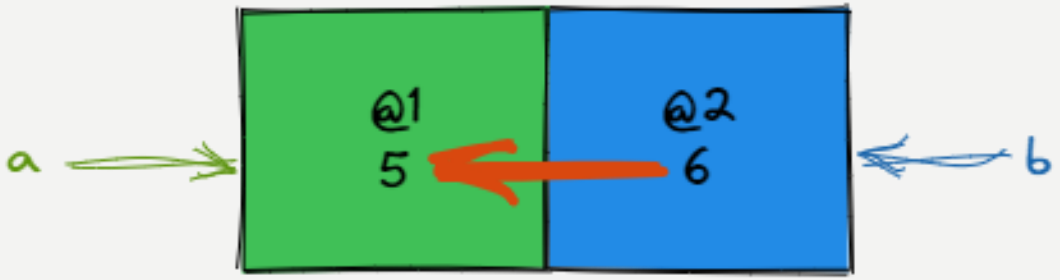
EN PRATIQUE

- Lorsqu'une variable est passée en paramètre d'un sous-programme
  - Une copie de la variable passée en paramètre est faite
  - Le paramètre est en fait une autre adresse mémoire, mais avec la même valeur
    - Elle a été copiée
    - On dit qu'on transmet par valeur

- En **Object**, la variable passée en paramètre d'un sous-programme n'est plus copiée
  - On a accès à la référence mémoire de l'objet
  - Chaque modification sur les attributs de cet objet impact donc l'objet qui a été transmit
    - On dit qu'on transmet par référence

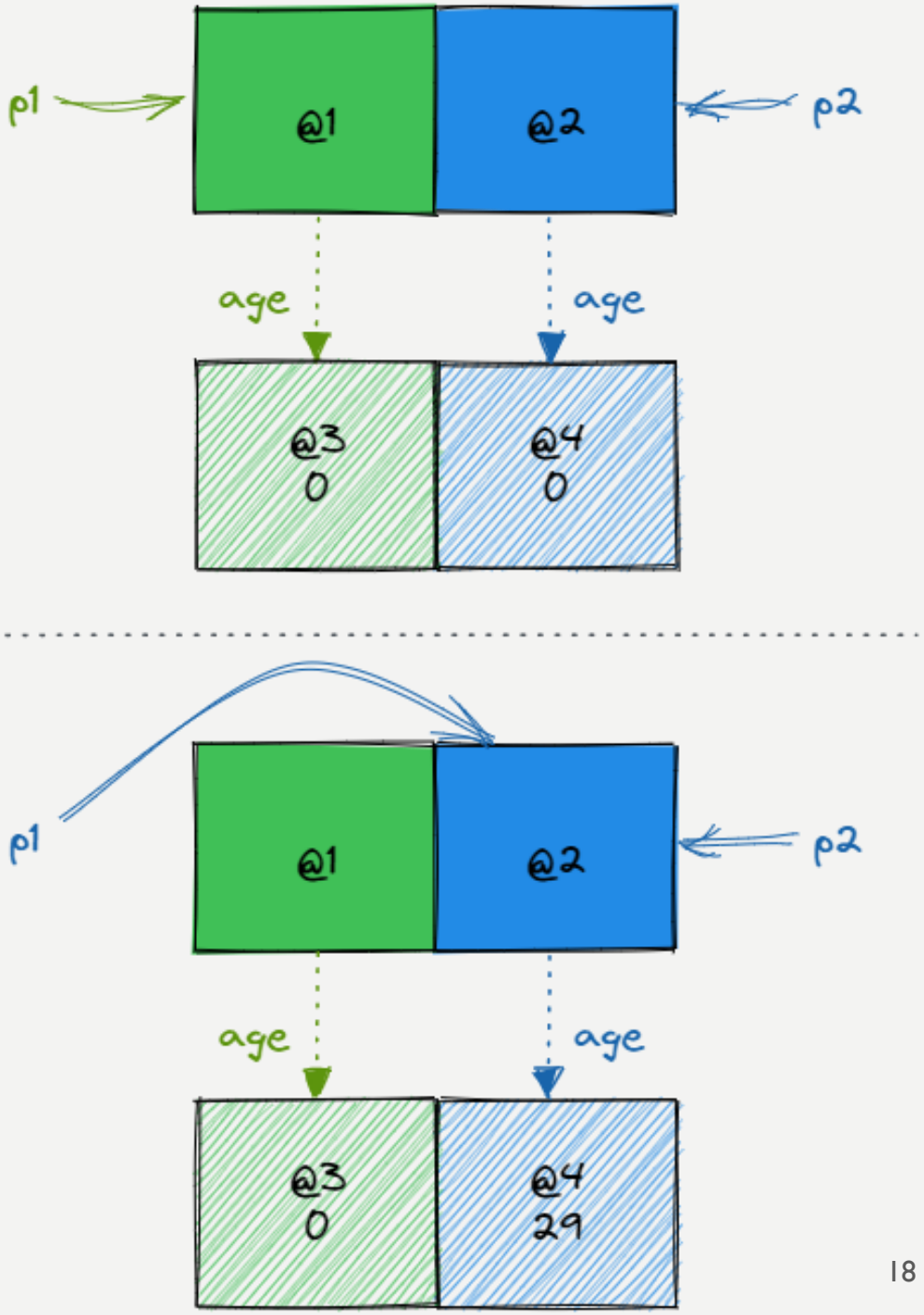
EN PRATIQUE

```java
int a = 5;
int b = 6;

a = b;
b = 7;

System.out.println(a);

//a = 6
```



EN PRATIQUE

```java
Personne p1 = new Personne();
Personne p2 = new Personne();

p1 = p2;
p2.age = 29;

System.out.println(p1.age);

//p1.age = 29
```



La première personne sera supprimée par le GarbageCollector de **JAVA** car plus référencée nulle part.

EXERCICE

- Démonstration **JAVA**

APPLICATION

- Programmation procédurale
  - Définition de fonctions, de programmes
  - Dirigée par les traitements
  - Processus
    - Point d'entrée unique (traitement)
    - En fonction des cas : appelle un traitement ou un autre

- Programmation Orientée Objet
  - Définition d'objets qui interagissent entre eux
  - Dirigée par les données
  - Réutilisable
  - Processus
    - Point d'entrée unique (objet)
    - Instancie d'autres objets et les utilise

Meilleure séparation des données et des fonctions qui les manipulent

APPLICATION

- 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

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

- … Mais pourquoi `static` ?

STATIQUE …

- Dans certains cas, on vent éviter d'instancier pour accéder à des données ou des méthodes
  - Les données et/ou les sous-programmes doivent être `Statique`
  - En **JAVA**, c'est le cas de la classe principale qui possède un programme `main` statique !

STATIQUE …

```
class Personne {
  char[] prenom;
  char[] nom;
  int age = 0;

  booleen isChild() {
    if (this.age < 18) {
      return true;
    }
    return false;
  }

  static Personne creerPersonne(int age) {
    Personne maPersonne = new Personne();
    maPersonne.age = age;
    return maPersonne;
  }
}
```

```
Personne pers = Personne.creerPersonne(29);

pers.isChild();
```

EXERCICE

- Créer une **classe** `Equipe`
  - Nom, 2 joueurs
  - Avec une méthode statique
    - Qui crée une nouvelle `Equipe`
    - Qui attend le nom de l'équipe et les 2 joueurs

- Instancier 4 nouveaux joueurs (sans saisie des informations)
- Instancier 2 nouvelles équipes (sans saisie des informations)
- Affectez les joueurs à une équipe

COLLECTIONS

LES TABLEAUX EN OBJET

LES COLLECTIONS

- Un gros problème des tableaux, c'est qu'ils ont une taille fixe
  - Obligé de créer un nouveau tableau pour l'agrandir
  - Obligé de décaler les cases si on veut insérer ou supprimer une valeur
  - …
- Introduction de la `Collection` d'objets

```
ArrayList<Personne> mesPersonnes = new ArrayList<>();

Personne p1 = new Personne();
Personne p2 = new Personne();

mesPersonnes.add(p1);
mesPersonnes.add(p2);

mesPersonnes.get(0);
mesPersonnes.remove(0);
```

LES COLLECTIONS

- Parcourir une `Collection` (boucle `Pour`)

```
for (int i = 0; i < mesPersonnes.size(); i = i + 1) {
  System.out.println(mesPersonnes.get(i).nom);
}
```

- Parcourir une `Collection` (boucle `PourChaque`)
  - *Pour chaque* `Personne` *qu'on appelle* `p` *dans* `mesPersonnes`

```
for (Personne p : mesPersonnes) {
  System.out.println(p.nom);
}
```

EXERCICE

- Ajouter la **classe** `Carte`
  - nom (`String`), valeur (`int`)
- Créer une liste de 32 cartes (valeurs et noms de 0 à 31)
- Afficher le nom de chaque carte
  - Boucle `Pour` classique
  - Boucle `PourChaque`