Stage de Programmation Objet

LP DAWIN

Informatique

Organisation du cours

Objectif : mise à niveau en programmation objet
Langage : Java

4 parties :

1. Bases de l'objet
2. Spécificités, interfaces et packages
3. Héritage
4. Collections et bonus

En parallèle : un projet de moteur de jeu (mode texte)

Évaluation : un petit contrôle (20') + un DS machine (3h)

Les concepts importants (en rouge) sont à connaître et maîtriser.

Quelques références

Cours de Philippe Narbel en Master 1 CSI :
`http://dept-info.labri.fr/~narbel/ProgCSI/`

Programmation

Produire un code :

- correct
- maintenable (ajouts + correctifs)

Programmation

Produire un code :

- correct
- maintenable (ajouts + correctifs)

ce qui signifie :

- extensible / modifiable / modulaire

Programmation

Produire un code :

- correct
- maintenable (ajouts + correctifs)

ce qui signifie :

- extensible / modifiable / modulaire
  - *classes, interfaces, packages, visibilité, héritage...*

Programmation

Produire un code :

- correct
- maintenable (ajouts + correctifs)

ce qui signifie :

- extensible / modifiable / modulaire
  - *classes, interfaces, packages, visibilité, héritage...*
- concis / factorisé

Programmation

Produire un code :

- correct
- maintenable (ajouts + correctifs)

ce qui signifie :

- extensible / modifiable / modulaire
  - *classes, interfaces, packages, visibilité, héritage...*
- concis / factorisé
  - *méthodes, héritage...*

Programmation

Produire un code :

- correct
- maintenable (ajouts + correctifs)

ce qui signifie :

- extensible / modifiable / modulaire
  - *classes, interfaces, packages, visibilité, héritage...*
- concis / factorisé
  - *méthodes, héritage...*
- lisible

Programmation

Produire un code :

- correct
- maintenable (ajouts + correctifs)

ce qui signifie :

- extensible / modifiable / modulaire
  - *classes, interfaces, packages, visibilité, héritage...*
- concis / factorisé
  - *méthodes, héritage...*
- lisible
  - *conventions, documentation...*

Programmation

Produire un code :

- correct
- maintenable (ajouts + correctifs)

ce qui signifie :

- extensible / modifiable / modulaire
  - *classes, interfaces, packages, visibilité, héritage...*
- concis / factorisé
  - *méthodes, héritage...*
- lisible
  - *conventions, documentation...*
- robuste

Programmation

Produire un code :

- correct
- maintenable (ajouts + correctifs)

ce qui signifie :

- extensible / modifiable / modulaire
  - *classes, interfaces, packages, visibilité, héritage...*
- concis / factorisé
  - *méthodes, héritage...*
- lisible
  - *conventions, documentation...*
- robuste
  - *tests unitaires, constantes, visibilité...*

Partie 1

Bases de l'objet

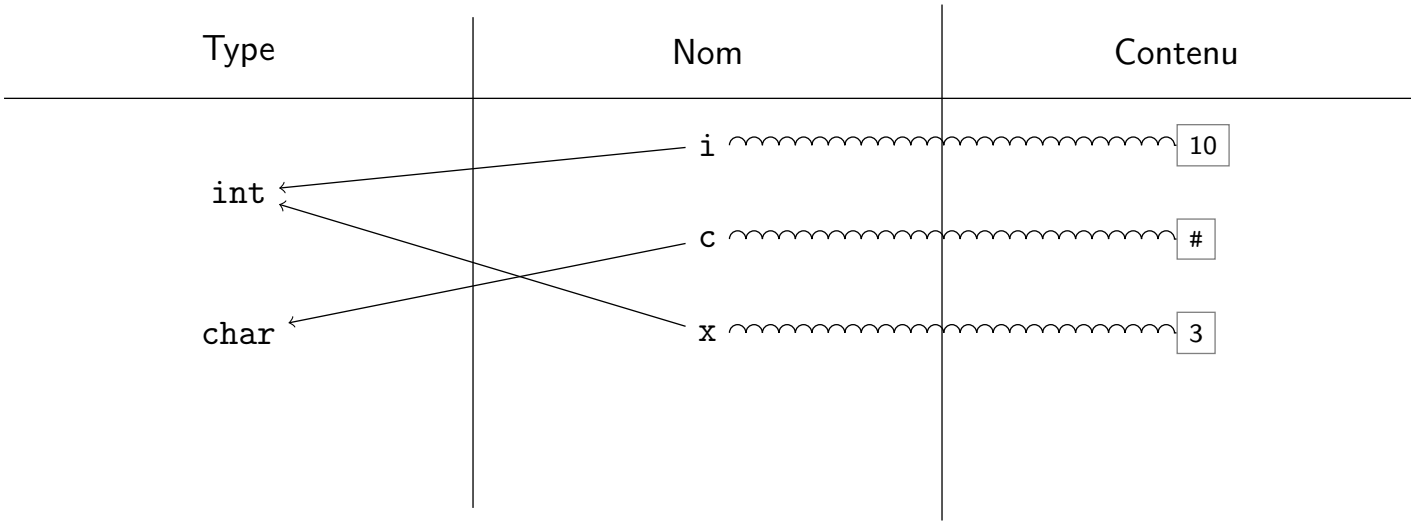
Partie 1 : Bases de l'objet

Revenons aux bases

Une variable, c'est 3 choses :

1. un type
2. un nom
3. un contenu

```
int i = 10;
char e = '#';
int x = 3;
```



Qu'est-ce qu'un objet ?

- une classe est un type de données comportant
  - des champs (appelés attributs), et
  - des opérations (appelées méthodes)

```
class Point {
  int x; // un attribut
  int y; // un autre attribut
  void afficher() { ... } // une methode
}
```

- un objet est une instance d'une classe, donc :
  - il peut attribuer une valeur à chacun de ses attributs, et
  - les méthodes de sa classe lui sont accessibles.

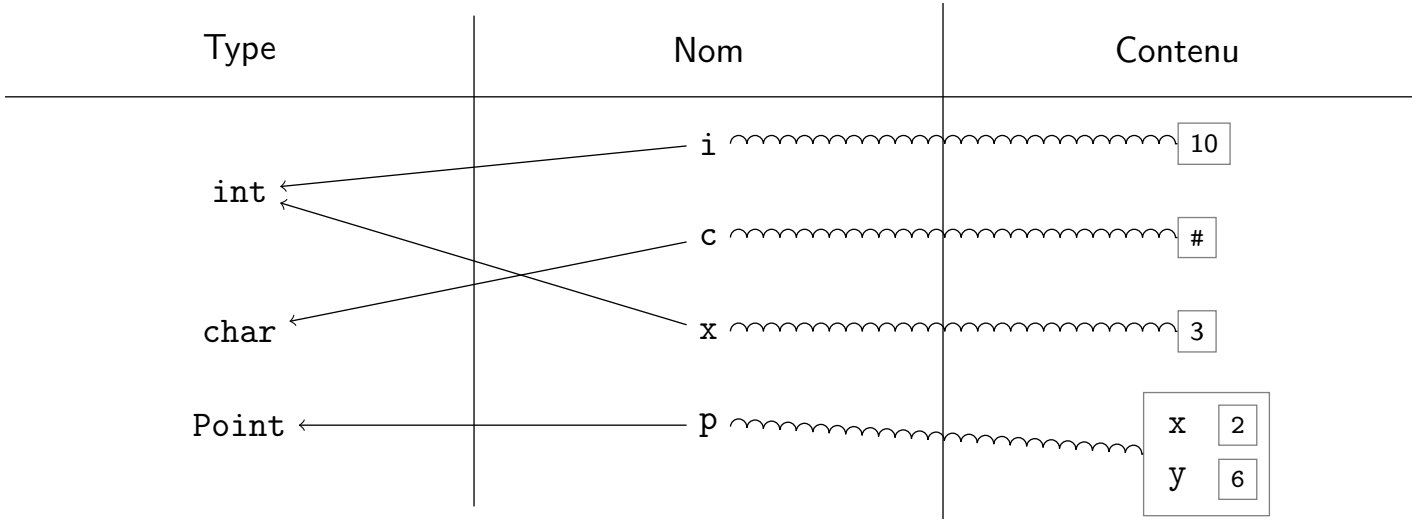
```
Point p = new Point();
p.x = 2;
p.y = 6;
p.afficher();
```

En particulier, objet = instance.

Qu'est-ce qu'un objet ?

```
class Point {
  int x;
  int y;
  void afficher() { ... }
}
```

```
int i = 10;
char e = '#';
int x = 3;

Point p = new Point();
p.x = 2;
p.y = 6;
p.afficher();
```



Programmation orientée objet

- La programmation orientée objet (POO) est un style de programmation (parmi d'autres...) où la notion d'objet est centrale.
- Chaque langage est plus ou moins adapté à la POO.
- Java est très adapté à la POO, ce fût un choix de conception. On parle alors de langage orienté objet.

Java

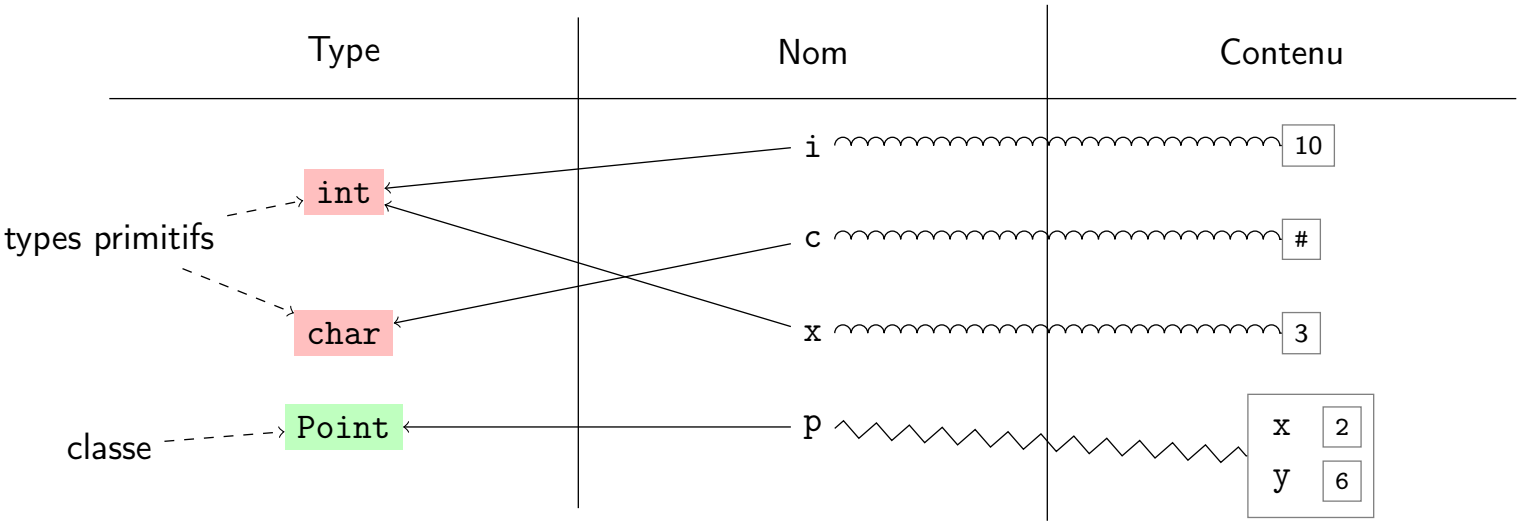
	1991	début du projet
JDK 1.0	01/1996	
JDK 1.1	02/1997	
J2SE 1.2	12/1998	
J2SE 1.3	05/2000	
J2SE 1.4	02/2002	
J2SE 5.0	09/2004	généricité, énumérations, varargs. . .
Java SE 6	12/2006	
Java SE 7	07/2011	
Java SE 8	03/2014	LTS $\rightarrow$ 2025, lambdas et streams
Java SE 9	09/2017	
Java SE 10	03/2018	
Java SE 11	09/2018	LTS $\rightarrow$ 2026
Java SE 12	03/2019	

Partie 1 : Bases de l'objet

Types en Java

Les types en Java sont : les types primitifs + les classes

(+ les interfaces, voir Partie 2)



- les liens "type $\leftrightarrow$ nom" sont **inamovibles**
  - *une variable ne change jamais de type*
- les liens "nom $\leftrightarrow$ contenu" sont :
  - **inamovibles** pour ⌇ les types primitifs
  - **changeants** pour ⌇ les classes ($\rightarrow$ dans quelques slides)

Classe : définition

Une classe permet de regrouper des informations concernant une "entité" (exemple ici : une salle, ou un rectangle), ainsi que les traitements associés.

Définir une classe

Salle.java

```
class Salle {
  int capacite;
  boolean salleMachine;
  boolean libreService;
  int numero;

  int etage() { ... }
}
```

Rectangle.java

```
class Rectangle {
  int x;
  int y;
  int largeur;
  int hauteur;

  int superficie() { ... }
}
```

En général, une classe est déclarée dans un fichier portant le même nom. Il existe quelques exceptions (classes internes...) que nous n'aborderons pas.

En Java, tout le code se trouve dans les classes.

(à l'exception des "méthodes par défaut" dans les interfaces...)

Classe : définition

Une classe n'est pas un fourre-tout !

On y regroupe les attributs et les méthodes qui concernent la même chose.

Classe : définition

Définir une classe

```
class Salle {
  int capacite;
  boolean salleMachine;
  boolean libreService;
  int numero;

  int etage() { ... }
}
```

```
class Rectangle {
  int x;
  int y;
  int largeur;
  int hauteur;

  int superficie() { ... }
}
```

Une classe constitue un nouveau type de données.
On peut donc déclarer des variables de ce type :

Déclarer une variable

```
Salle salle1;
Salle salle2;
```

```
Rectangle r1;
Rectangle r2;
```

Classe : définition

Lorsque le type est une classe :

Déclarer une variable

```
Salle salle1;
```

Vocabulaire

- avant son initialisation, cette variable prend la valeur `null`.
- une fois initialisée, la variable `salle1` contient une référence vers une instance de la classe `Salle` (nous verrons pourquoi ensuite).

On dit aussi que `salle1` est un objet de type `Salle`.

Partie 1 : Bases de l'objet

Créer un objet (= une instance)

Pour initialiser un objet, on utilise un constructeur.
Par défaut, il y a toujours un constructeur sans paramètre :

Constructeur par défaut

```
Salle salle1;
Salle salle2;
salle1 = new Salle();
salle2 = new Salle();
```

ou

```
Salle salle1 = new Salle();
Salle salle2 = new Salle();
```

Ensuite on accède à chaque attribut avec la notation `objet.attribut` :

Accès aux attributs

```
salle1.capacite = 30;
salle1.salleMachine = true;
salle1.libreService = true;
salle1.numero = 301;
```

```
salle2.capacite = 30;
salle2.salleMachine = true;
salle2.libreService = false;
salle2.numero = 303;
```

Classe : constructeurs

Définir un nouveau constructeur

```
Salle(int uneCapacite, boolean estUneSalleMachine,
      boolean estEnLibreService, int unNumero) {
  capacite = uneCapacite;
  salleMachine = estUneSalleMachine;
  libreService = estEnLibreService;
  numero = unNumero;
}
```

```
Salle salle1 = new Salle(30, true, true, 301);
```

Redéfinir le constructeur par défaut (si on veut)

Si, par exemple, on connaît des valeurs par défaut pour tous les attributs :

```
Salle() {
  capacite = 30;
  salleMachine = true;
  libreService = false;
  numero = 0;
}
```

```
Salle salle1 = new Salle();
salle1.numero = 301; // ecrase
```

Classe : constructeurs

Où placer un constructeur ?

$\rightarrow$ dans la classe !

```
class Salle {

  // Attributs
  int      capacite;
  boolean  salleMachine;
  boolean  libreService;
  int      numero;

  // Un premier constructeur
  Salle() {
    capacite = 30;
    salleMachine = true;
    libreService = false;
    numero = 0;
  }

  // Un deuxieme constructeur
  Salle(int uneCapacite, boolean estUneSalleMachine,
      boolean estEnLibreService, int unNumero) {
    capacite = uneCapacite;
    salleMachine = estUneSalleMachine;
    libreService = estEnLibreService;
    numero = unNumero;
  }
}
```

Classe : constructeurs

Un constructeur est une fonction particulière en Java, qui renvoie implicitement un nouvel objet.

Il se distingue d'une fonction classique :

- à la déclaration : pas de type de retour (il porte forcément le nom de la classe)
- lors de l'appel : mot-clé `new`

Partie 1 : Bases de l'objet

Attributs et méthodes

Dans une classe, on peut trouver :

1. des attributs,
2. des constructeurs, et
3. des méthodes.

Ainsi, dans chaque instance de cette classe :

- un attribut est une variable liée à cette instance.
- une méthode est simplement une fonction, ayant accès :
  - à ses paramètres, et
  - aux attributs de l'instance "`this`"

Attributs et méthodes

Dans une classe, on peut trouver :

1. des attributs,
2. des constructeurs, et
3. des méthodes.

Ainsi, dans chaque instance de cette classe :

- un attribut est une variable liée à cette instance.
- une méthode est simplement une fonction, ayant accès :
  - à ses paramètres, et
  - aux attributs de l'instance "`this`"

Ainsi le rôle de chacun est clair :

les attributs	stockent les données $\rightarrow$ état de l'instance
les constructeurs	initialisent les instances
les méthodes	contiennent les traitements

Attributs

Un attribut peut être de n'importe quel type...

... y compris un objet, un tableau d'objets, etc.

```
class Salle {
  int           capacite;
  boolean       salleMachine;
  boolean       libreService;
  int           numero;
  Ordinateur    ordiVideoProj; // l'ordinateur connecte au video-projecteur
  Ordinateur[]  ordisEtudiants; // tableau d'ordinateurs

  public static void main(String[] args) {
    salle301 = new Salle();
    salle301.ordiVideoProj     = new Ordinateur();
    salle301.ordisEtudiants    = new Ordinateur[30];
    salle301.ordisEtudiants[0] = new Ordinateur();
    ...
  }
}
```

```
class Ordinateur {
  int     modele;
  String  nomHote;
  OS      systeme;
}
```

Attributs

Un attribut peut être de n'importe quel type...

... y compris un objet, un tableau d'objets, etc.

```
class Salle {
  int           capacite;
  boolean       salleMachine;
  boolean       libreService;
  int           numero;
  Ordinateur    ordiVideoProj; // l'ordinateur connecte au video-projecteur
  Ordinateur[]  ordisEtudiants; // tableau d'ordinateurs

  public static void main(String[] args) {
    salle301 = new Salle();
    salle301.ordiVideoProj     = new Ordinateur();
    salle301.ordisEtudiants    = new Ordinateur[30];
    salle301.ordisEtudiants[0] = new Ordinateur();
    ...
  }
}
```

```
class Ordinateur {
  int     modele;
  String  nomHote;
  OS      systeme;
}
```

Les constantes

Un attribut ou une variable peuvent être constants.

On utilise le mot-clé `final` pour l'indiquer :

```
class Salle {
  final int CAPACITE_MAX = 300; // attribut constant
  final Ordinateur ordiVideoProj;
  ...
  Salle(Ordinateur ordiVP) {
    ordiVideoProj = ordiVP;
  }
  ...
  void updateOrdiVP(String unNomHote) {
    ordiVideoProj = new Ordinateur(unNomHote); // interdit !! (final)
    ordiVideoProj.nomHote = unNomHote; // ok : ne change pas d'instance
    final String nomNormalise = normaliser(unNomHote); // variable constante
    enregister(nomNormalise);
  }
  ...
      CAPACITE_MAX = 32; // interdit !! (final)
  ...
}
```

Sur un attribut : initialisé à la déclaration ou dans le constructeur.

S'il est de type classe : on peut modifier l'instance, mais pas affecter une nouvelle instance.

Convention : majuscules, pour les types primitifs (surtout pour les `static`, cf Partie 2).

Méthodes

Une méthode est simplement une fonction, avec potentiellement des arguments, et toujours un type de retour (`void` si rien à renvoyer).

- Elle est forcément déclarée dans une classe.
- Si une méthode `m()` est déclarée dans une classe `C`, alors elle est forcément appelée sur une instance de type `C`.

Exemple

Déclaration (dans la classe `Salle`) :

```
boolean salleMachineAu3eme(boolean estLibreService) {
  return this.salleMachine
      && this.numero / 100 == 3
      && this.libreService == estLibreService;
}
```

Invocation (potentiellement dans une autre classe) :

```
Salle s = ...;
if (s.salleMachineAu3eme(true)) {
  ...
}
```

Méthodes avec le même nom : la surcharge

On peut déclarer plusieurs méthodes avec le même nom (dans la même classe), pourvu qu'elles aient des signatures différentes.

```
boolean salleMachineAu3eme() {
  return this.salleMachine && this.numero/100 == 3;
}

boolean salleMachineAu3eme(boolean estLibreService) {
  return this.salleMachine
      && this.numero / 100 == 3
      && this.libreService == estLibreService;
}
```

C'est la notion de surcharge.

Partie 1 : Bases de l'objet

Syntaxe Java

Regardons sur un exemple (personnage d'un jeu devant trouver 8 objets).
Observez : la javadoc, les tableaux, les boucles, les chaînes (String), les fonctions...

```java
package mygame;

/**
 * Character of a game, who has to find 8 items.
 */
class GameCharacter {

    /** Character's name */
    String name;

    /** Hit points of the character */
    int hp;

    /** Armor of a character */
    int armor;

    /** An array of booleans indicating which items have been found */
    boolean[] foundItem;
```

Syntaxe Java

```java
/**
 * Default constructor.
 *
 * @param aName name of the character
 */
GameCharacter(String aName) {
    name = aName;
    hp = 0;
    armor = 0;
    foundItem = new boolean[8];
    for (int i = 0; i < 8; i++) {
        foundItem[i] = false;
    }
}

/**
 * Character gets injured.
 *
 * @param damage amount of damage
 */
void injured(int damage) {
    hp -= damage; // or this.hp -= damage, but not hp -= this.damage
    armor -= 1; // does not depend on damage
}
```

Syntaxe Java

```java
/**
 * Returns the name with an additional character between letters.
 * For instance, with a dot: if name is "HeRo", it returns "H.e.R.o."
 *
 * @param spaceChar the character inserted between letters
 * @return the name with inserted spaces
 */
String nameWithSpaces(char spaceChar) {
    String withSpaces = "";
    for (int i = 0; i < name.length(); i++) {
        withSpaces = withSpaces + name.charAt(i) + spaceChar;
    }
    return withSpaces;
}

/**
 * Displays found items.
 * For instance, if items 0, 4 and 6 have been found: "0   4 6 ".
 */
void displayItems() {
    for (int i = 0; i < 8; i++) {
        if (foundItem[i]) {
            System.out.println(i);
        } else {
            System.out.println(" ");
        }
        // or, with ternary operator:
        // System.out.println(foundItems[i]?i:" ");
    }
}
```

Partie 1 : Bases de l'objet

Mini-projet : v0 $\rightarrow$ v1

- Cloner le dépôt git
- Importer les sources dans un IDE
- Compiler, exécuter

Niveau 1

1. Découper le code en classes (attributs, méthodes, constructeurs)
2. Contrôler la saisie ([a-d] ou [A-D])
3. 'q' ou 'Q' quitte le jeu

Niveau 2

1. Afficher les lettres de ligne et colonne
2. Vérifier que le code fonctionne en changeant les dimensions

```
   A B C D
a  B     W
b
c    B B W
d  W     B
```

Partie 1 : Bases de l'objet

Comment s'exécute un programme Java ?

Dans un environnement de développement, on utilise un IDE (Eclipse, NetBeans, IntelliJ...), qui masque tout : l'IDE s'occupe de la compilation et du lancement.

Dans un environnement de production, il n'y a pas d'IDE...

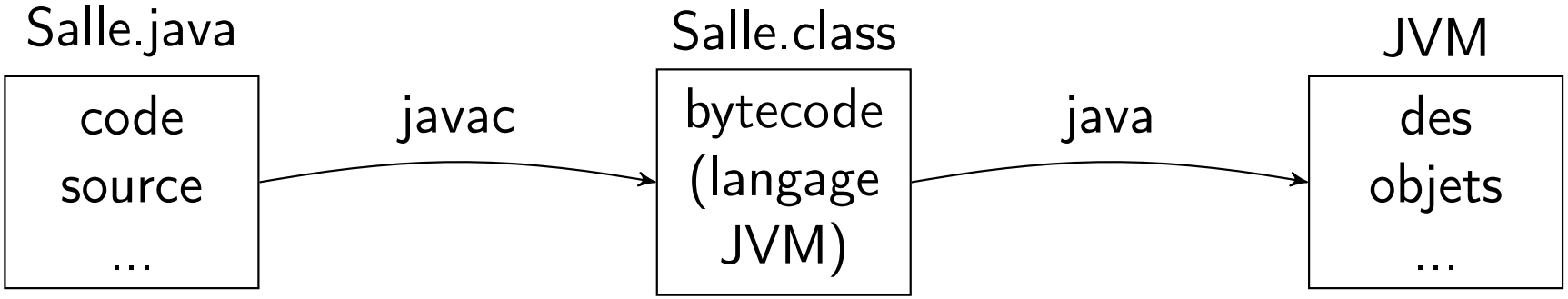
On vous montre les coulisses...

Java : compiler et exécuter

Un programme Java n'a pas besoin d'être recompilé pour chaque architecture d'ordinateur :

write once, run anywhere

Ceci est rendu possible grâce à la JVM : Java Virtual Machine.



Java : compiler et exécuter

En Java on exécute une classe. Elle doit posséder une méthode main() :

```
class Salle {

    ...

    public static void main(String[] args) {
        Salle salle301 = new Salle(30, true, true, 301);
        if (salle301.salleMachineAu3eme()) {
            System.out.println("La salle " + salle301.numero +
                " est une salle machine au 3eme.");
        }
    }
    ...
}
```

C'est le point de départ de l'exécution.

Java : compiler et exécuter

1. Compiler le fichier Salle.java (-cp = “classpath”) :

```
$ tree .
.
├── bin
└── src
    └── gestionsalles
        └── Salle.java

3 directories, 1 file
$ javac -cp src -d bin src/gestionsalles/Salle.java
$ tree .
.
├── bin
│   └── gestionsalles
│       └── Salle.class
└── src
    └── gestionsalles
        └── Salle.java

4 directories, 2 files
```

2. Lancer la classe `gestionsalles.Salle` :

```
$ java -cp bin gestionsalles.Salle
La salle 301 est une salle machine au 3ème.
```

Dur(ée d)e vie d'un objet

La JVM s'occupe de gérer la mémoire :

- créer les objets (lors d'un `new`),
- détruire les objets (via le garbage collector, lorsqu'un objet n'a plus de référence vers lui).

Partie 1 : Bases de l'objet

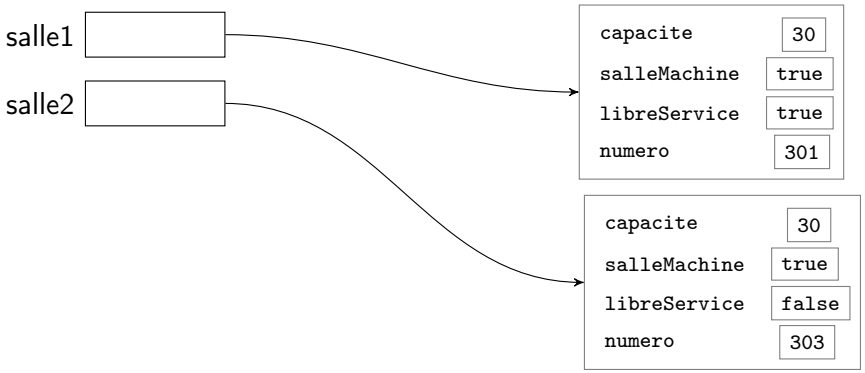
Références

En Java, lorsqu'on accède à des objets, on manipule toujours des références vers les objets (instances) :

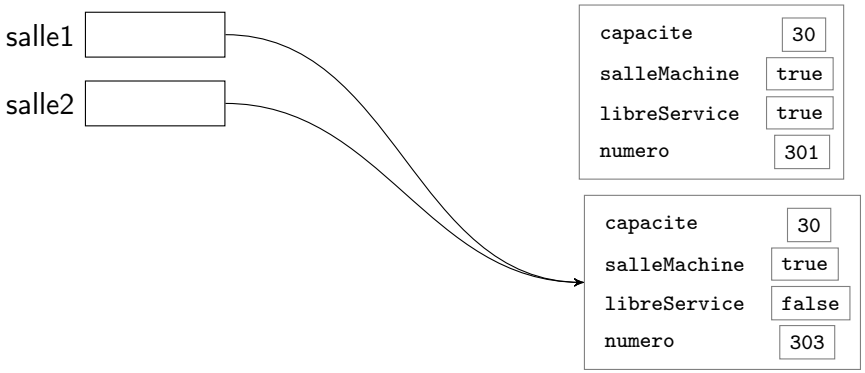
```
Salle salle1 = new Salle( 30, true , true ,  301);
Salle salle2 = new Salle( 30, true , false , 303);
```

Références

```
Salle salle1 = new Salle( 30, true, true, 301);
Salle salle2 = new Salle( 30, true, false, 303);
```




```
salle1 = salle2;
```



Passage de référence en paramètre

Une conséquence est que les objets passés en paramètre d'une fonction peuvent être modifiés (lecture / écriture)...

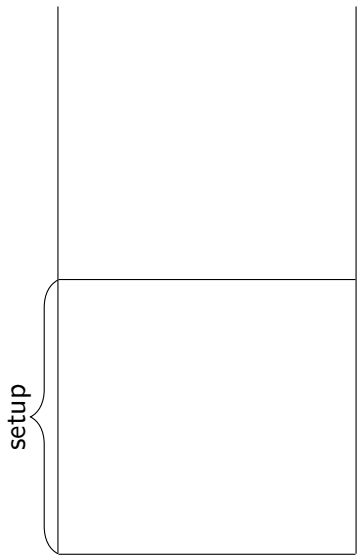
... contrairement aux types primitifs, qui sont passés par copie (lecture).

```
/**
 * Demenage une salle d'un etage vers un nouvel etage.
 */
void demenage(Salle s, int etageAvant, int etageApres) {
  // si la salle est a l'etage concerne
  if (s.numero / 100 == etageAvant) {
    // on demenage ! 301 devient 201
    s.numero = (s.numero % 100) + (etageApres * 100);
  }
}
void setup() {
  Salle salle1 = new Salle( 30, true, true,  301);
  Salle salle2 = new Salle( 30, true, false, 303);
  int etageSrc = 3;
  int etageDst = 2;
  println("Numero avant :", salle1.numero);
  demenage(salle1, etageSrc, etageDst);
  println("Numero apres :", salle1.numero);
}
```

→ pourquoi ?

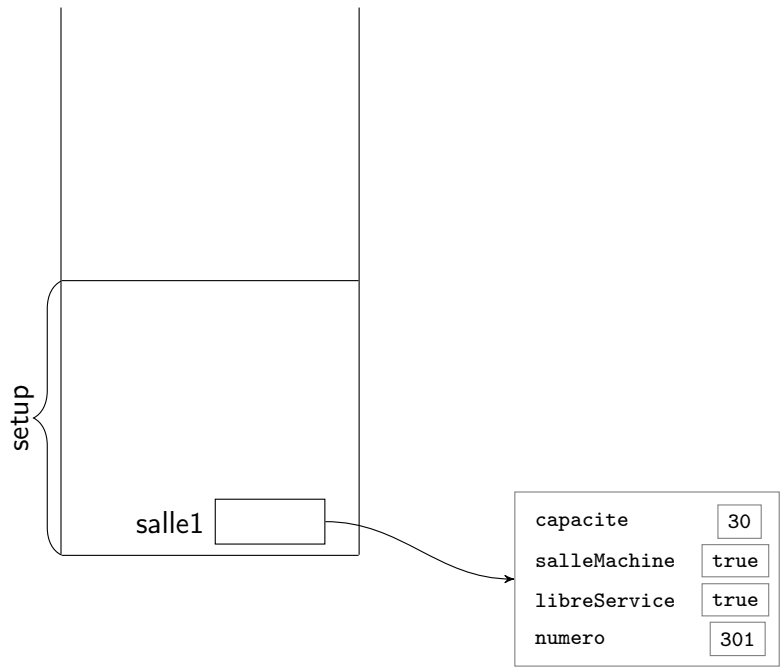
Passage de référence en paramètre

```
void demenage(Salle s, int etageAvant, int etageApres) {
  // si la salle est a l'etage concerne
  if (s.numero / 100 == etageAvant) {
    // on demenage ! 301 devient 201
    s.numero = (s.numero % 100) + (etageApres * 100);
  }
}

void setup() {
  Salle salle1 = new Salle( 30, true, true, 301);
  Salle salle2 = new Salle( 30, true, false, 303);
  int etageSrc = 3;
  int etageDst = 2;
  println("Numero avant :", salle1.numero);
  demenage(salle1, etageSrc, etageDst);
  println("Numero apres :", salle1.numero);
}
```



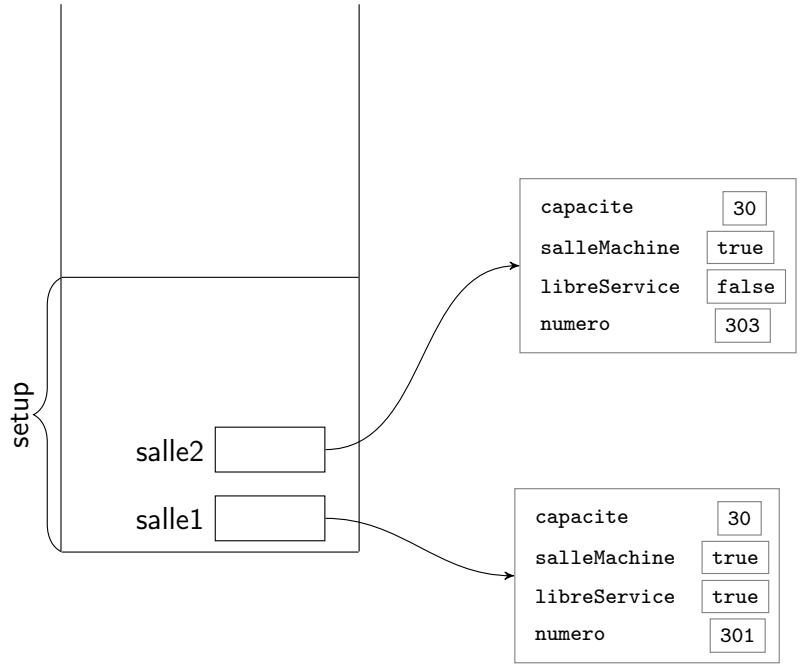
Passage de référence en paramètre

```
void demenage(Salle s, int etageAvant, int etageApres) {
  // si la salle est a l'etage concerne
  if (s.numero / 100 == etageAvant) {
    // on demenage ! 301 devient 201
    s.numero = (s.numero % 100) + (etageApres * 100);
  }
}

void setup() {
  Salle salle1 = new Salle( 30, true, true, 301);
  Salle salle2 = new Salle( 30, true, false, 303);
  int etageSrc = 3;
  int etageDst = 2;
  println("Numero avant :", salle1.numero);
  demenage(salle1, etageSrc, etageDst);
  println("Numero apres :", salle1.numero);
}
```



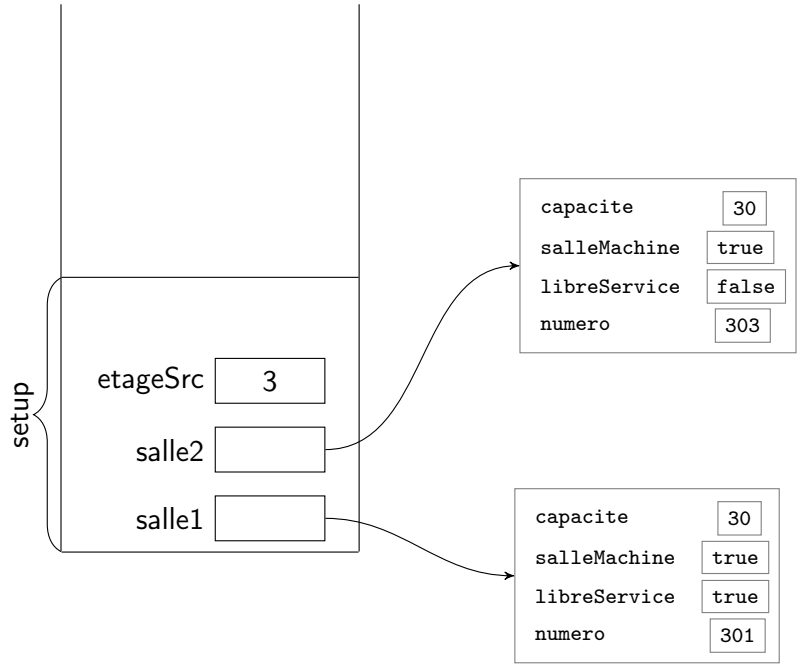
Passage de référence en paramètre

```
void demenage(Salle s, int etageAvant, int etageApres) {
  // si la salle est a l'etage concerne
  if (s.numero / 100 == etageAvant) {
    // on demenage ! 301 devient 201
    s.numero = (s.numero % 100) + (etageApres * 100);
  }
}

void setup() {
  Salle salle1 = new Salle( 30, true, true, 301);
  Salle salle2 = new Salle( 30, true, false, 303);
  int etageSrc = 3;
  int etageDst = 2;
  println("Numero avant :", salle1.numero);
  demenage(salle1, etageSrc, etageDst);
  println("Numero apres :", salle1.numero);
}
```



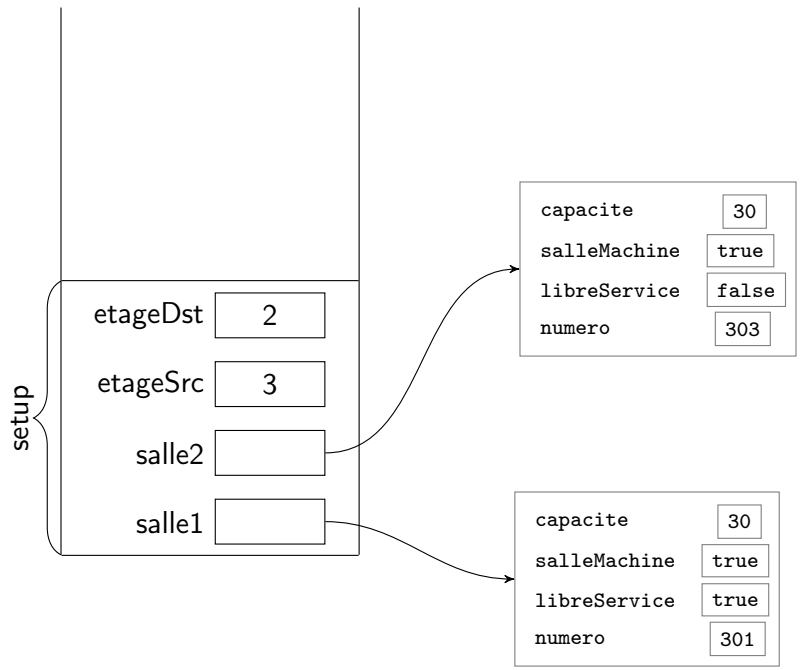
Passage de référence en paramètre

```
void demenage(Salle s, int etageAvant, int etageApres) {
  // si la salle est a l'etage concerne
  if (s.numero / 100 == etageAvant) {
    // on demenage ! 301 devient 201
    s.numero = (s.numero % 100) + (etageApres * 100);
  }
}

void setup() {
  Salle salle1 = new Salle( 30, true, true, 301);
  Salle salle2 = new Salle( 30, true, false, 303);
  int etageSrc = 3;
  int etageDst = 2;
  println("Numero avant :", salle1.numero);
  demenage(salle1, etageSrc, etageDst);
  println("Numero apres :", salle1.numero);
}
```



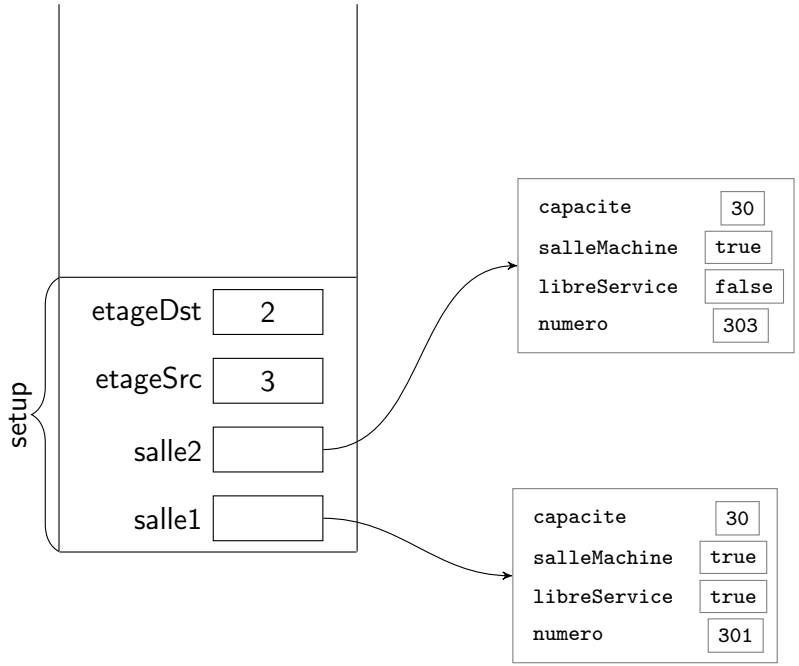
Passage de référence en paramètre

```
void demenage(Salle s, int etageAvant, int etageApres) {
  // si la salle est a l'etage concerne
  if (s.numero / 100 == etageAvant) {
    // on demenage ! 301 devient 201
    s.numero = (s.numero % 100) + (etageApres * 100);
  }
}

void setup() {
  Salle salle1 = new Salle( 30, true, true, 301);
  Salle salle2 = new Salle( 30, true, false, 303);
  int etageSrc = 3;
  int etageDst = 2;
  println("Numero avant :", salle1.numero);
  demenage(salle1, etageSrc, etageDst);
  println("Numero apres :", salle1.numero);
}
```



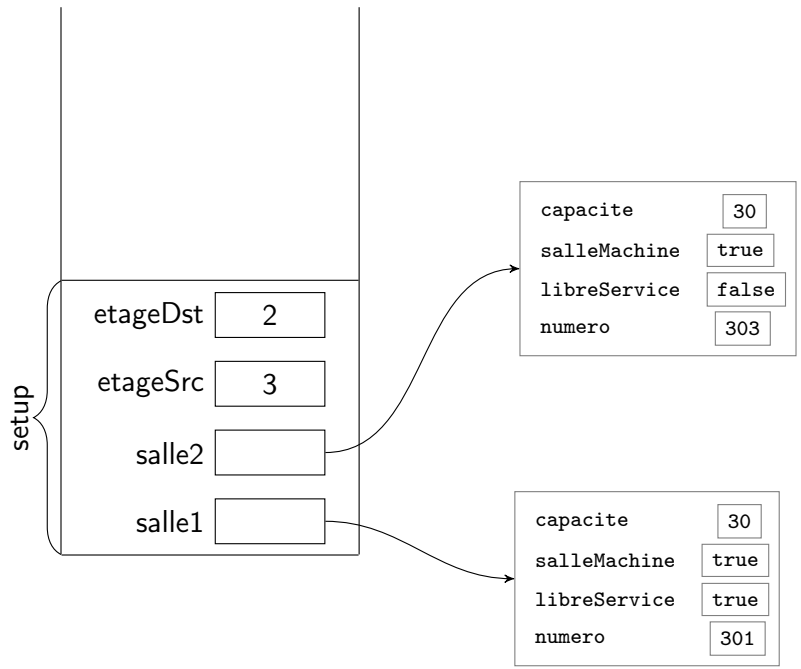
Passage de référence en paramètre

```
void demenage(Salle s, int etageAvant, int etageApres) {
  // si la salle est a l'etage concerne
  if (s.numero / 100 == etageAvant) {
    // on demenage ! 301 devient 201
    s.numero = (s.numero % 100) + (etageApres * 100);
  }
}

void setup() {
  Salle salle1 = new Salle( 30, true, true, 301);
  Salle salle2 = new Salle( 30, true, false, 303);
  int etageSrc = 3;
  int etageDst = 2;
  println("Numero avant :", salle1.numero);
  demenage(salle1, etageSrc, etageDst);
  println("Numero apres :", salle1.numero);
}
```



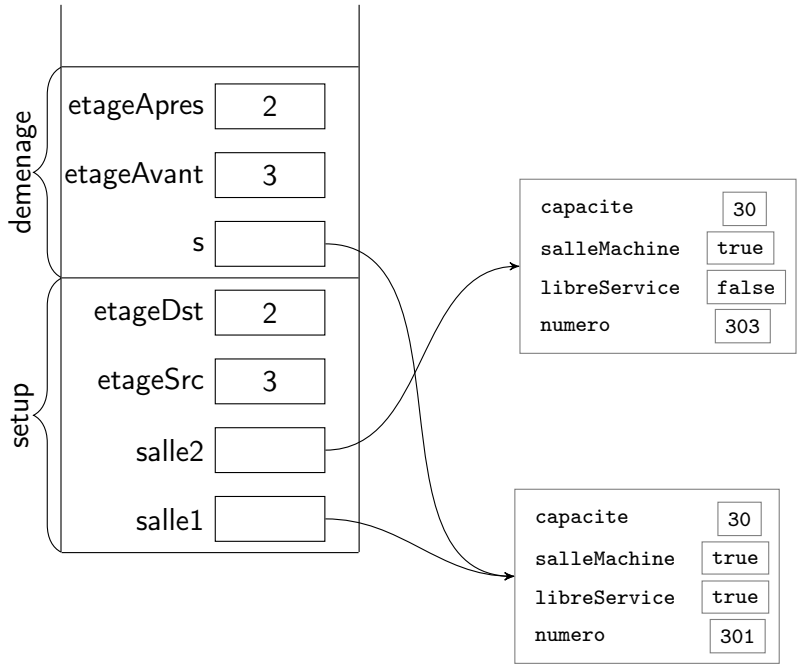
Passage de référence en paramètre

```
void demenage(Salle s, int etageAvant, int etageApres) {
  // si la salle est a l'etage concerne
  if (s.numero / 100 == etageAvant) {
    // on demenage ! 301 devient 201
    s.numero = (s.numero % 100) + (etageApres * 100);
  }
}

void setup() {
  Salle salle1 = new Salle( 30, true, true, 301);
  Salle salle2 = new Salle( 30, true, false, 303);
  int etageSrc = 3;
  int etageDst = 2;
  println("Numero avant :", salle1.numero);
  demenage(salle1, etageSrc, etageDst);
  println("Numero apres :", salle1.numero);
}
```



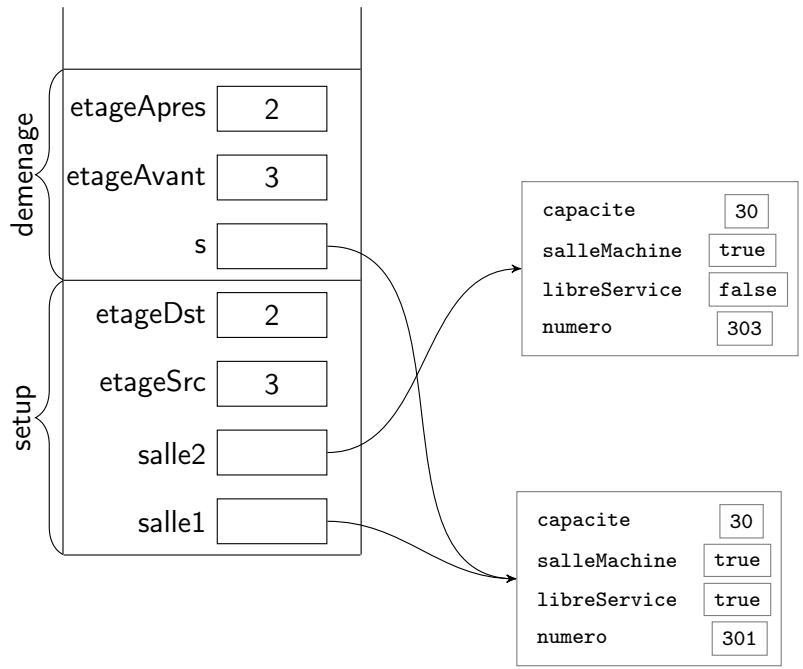
Passage de référence en paramètre

```
void demenage(Salle s, int etageAvant, int etageApres) {
  // si la salle est a l'etage concerne
  if (s.numero / 100 == etageAvant) {
    // on demenage ! 301 devient 201
    s.numero = (s.numero % 100) + (etageApres * 100);
  }
}

void setup() {
  Salle salle1 = new Salle( 30, true, true, 301);
  Salle salle2 = new Salle( 30, true, false, 303);
  int etageSrc = 3;
  int etageDst = 2;
  println("Numero avant :", salle1.numero);
  demenage(salle1, etageSrc, etageDst);
  println("Numero apres :", salle1.numero);
}
```



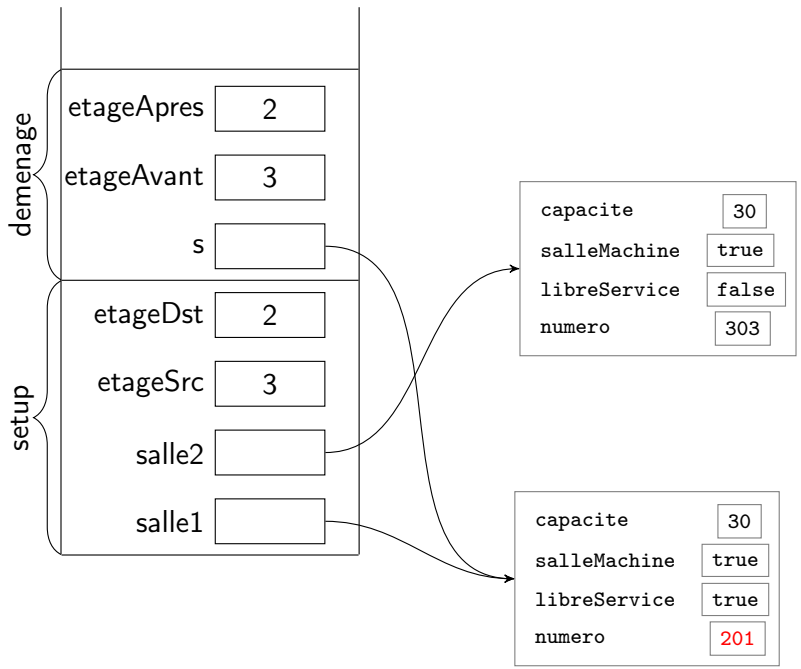
Passage de référence en paramètre

```
void demenage(Salle s, int etageAvant, int etageApres) {
  // si la salle est a l'etage concerne
  if (s.numero / 100 == etageAvant) {
    // on demenage ! 301 devient 201
    s.numero = (s.numero % 100) + (etageApres * 100);
  }
}

void setup() {
  Salle salle1 = new Salle( 30, true, true, 301);
  Salle salle2 = new Salle( 30, true, false, 303);
  int etageSrc = 3;
  int etageDst = 2;
  println("Numero avant :", salle1.numero);
  demenage(salle1, etageSrc, etageDst);
  println("Numero apres :", salle1.numero);
}
```



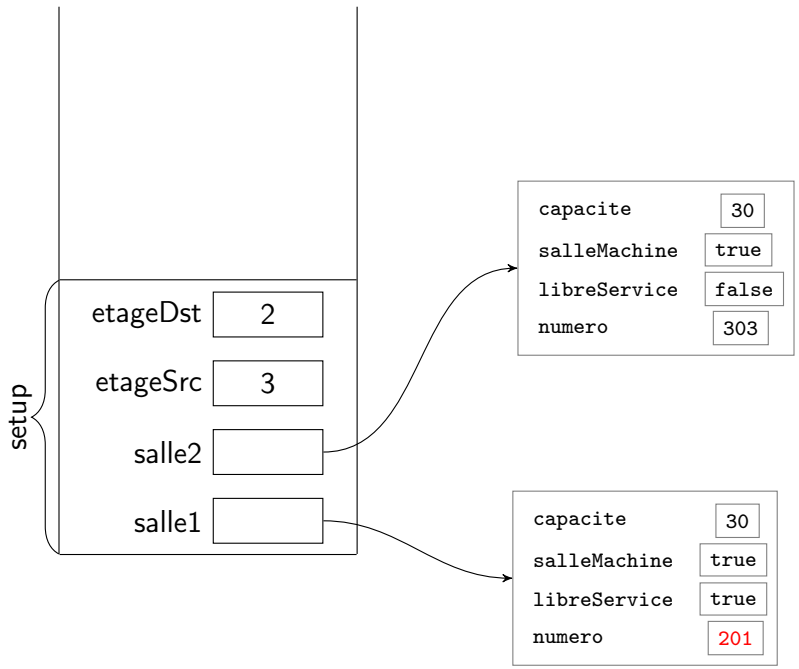
Passage de référence en paramètre

```
void demenage(Salle s, int etageAvant, int etageApres) {
  // si la salle est a l'etage concerne
  if (s.numero / 100 == etageAvant) {
    // on demenage ! 301 devient 201
    s.numero = (s.numero % 100) + (etageApres * 100);
  }
}

void setup() {
  Salle salle1 = new Salle( 30, true, true, 301);
  Salle salle2 = new Salle( 30, true, false, 303);
  int etageSrc = 3;
  int etageDst = 2;
  println("Numero avant :", salle1.numero);
  demenage(salle1, etageSrc, etageDst);
  println("Numero apres :", salle1.numero);
}
```



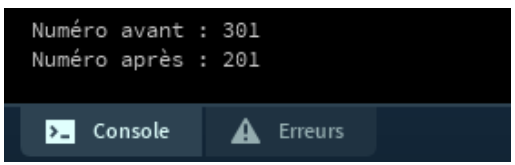
Passage de référence en paramètre

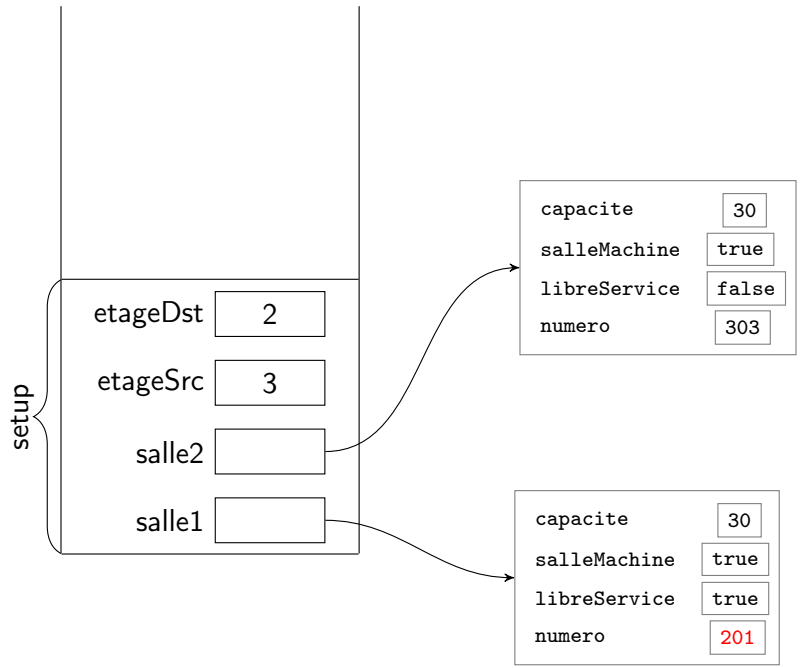
```
void demenage(Salle s, int etageAvant, int etageApres) {
  // si la salle est a l'etage concerne
  if (s.numero / 100 == etageAvant) {
    // on demenage ! 301 devient 201
    s.numero = (s.numero % 100) + (etageApres * 100);
  }
}

void setup() {
  Salle salle1 = new Salle( 30, true, true, 301);
  Salle salle2 = new Salle( 30, true, false, 303);
  int etageSrc = 3;
  int etageDst = 2;
  println("Numero avant :", salle1.numero);
  demenage(salle1, etageSrc, etageDst);
  println("Numero apres :", salle1.numero);
}
```



Passage de référence en paramètre

```
void demenage(Salle s, int etageAvant, int etageApres) {
  // si la salle est a l'etage concerne
  if (s.numero / 100 == etageAvant) {
    // on demenage ! 301 devient 201
    s.numero = (s.numero % 100) + (etageApres * 100);
  }
}

void setup() {
  Salle salle1 = new Salle( 30, true, true, 301);
  Salle salle2 = new Salle( 30, true, false, 303);
  int etageSrc = 3;
  int etageDst = 2;
  println("Numero avant :", salle1.numero);
  demenage(salle1, etageSrc, etageDst);
  println("Numero apres :", salle1.numero);
}
```





Partie 1 : Bases de l'objet

Mini-projet : v1 $\rightarrow$ v2

Niveau 1

1. Tests unitaires (JUnit) sur :
   - la saisie d'une action
   - la mise à jour du plateau

Niveau 2

1. Afficher des séparateurs de ligne/colonne (option via paramètre)
2. Tester (en interactif) avec/sans séparateur