

IA – 2014 / 2015

Programmation Orientée Objet

Cours I

Gérald Oster <oster@loria.fr>

Présentation du module

- Objectifs :
 - Connaître les concepts fondamentaux des **langages objets**
 - Etudier et maîtriser un langage objet : Java

Organisation pratique

- 6 cours magistraux
- 9 séances de travaux dirigés
- 7 séances de travaux pratiques
- Evaluation :
 - 1 examen sur table (28/03/2015)
 - 1 projet
 - Quelques travaux « à la maison »
 - Quizz durant quelques séances de TD

Ressources

- Site du cours sur Arche :
<http://arche.univ-lorraine.fr/course/view.php?id=7973>
- Page web institutionnelle :
<http://www.loria.fr/~oster/pmwiki/index.php/Main/POO>
- MOOC en français (Coursera / EPFL)
 - Initiation à la programmation (en Java)
<https://www.coursera.org/course/initprojava>
 - Introduction à la programmation orientée objet (en Java)
<https://www.coursera.org/course/intropoojava>


CAY HORSTMANN

Big Java

COMPATIBLE
WITH
Java
5 & 6

3RD EDITION

ns.

-
- Revised and Updated for Java SE 6
- # Core Java™
- ## Volume I • Fundamentals
- EIGHTH EDITION
-  Sun
- Cay S. Horstmann • Gary Cornell

Plan du cours

- Introduction
- Programmation orientée objet :
 - Classes, objets, encapsulation, composition
 - 1. Utilisation
 - 2. Définition
- Héritage et polymorphisme :
 - Interface, classe abstraite, liaison dynamique
- Exceptions
- Généricité

Acquis de formation

- Utiliser le langage Java pour implémenter et tester des algorithmes pour résoudre des problèmes simples,
- Concevoir et implémenter une classe,
- Utiliser les mécanismes d'encapsulation orientés objet tels que les interfaces et les membres privés,
- Appliquer les techniques de décomposition pour découper un programme complexe en morceaux plus simples et réutilisables,
- Utiliser l'héritage pour concevoir des hiérarchies simples de classes permettant aux sous-classes de réutiliser du code,
- Raisonner correctement sur le flot de contrôle dans un programme faisant intervenir la liaison dynamique,
- Tracer l'exécution de segments de code variés et de résumer leur effets en terme de calcul.

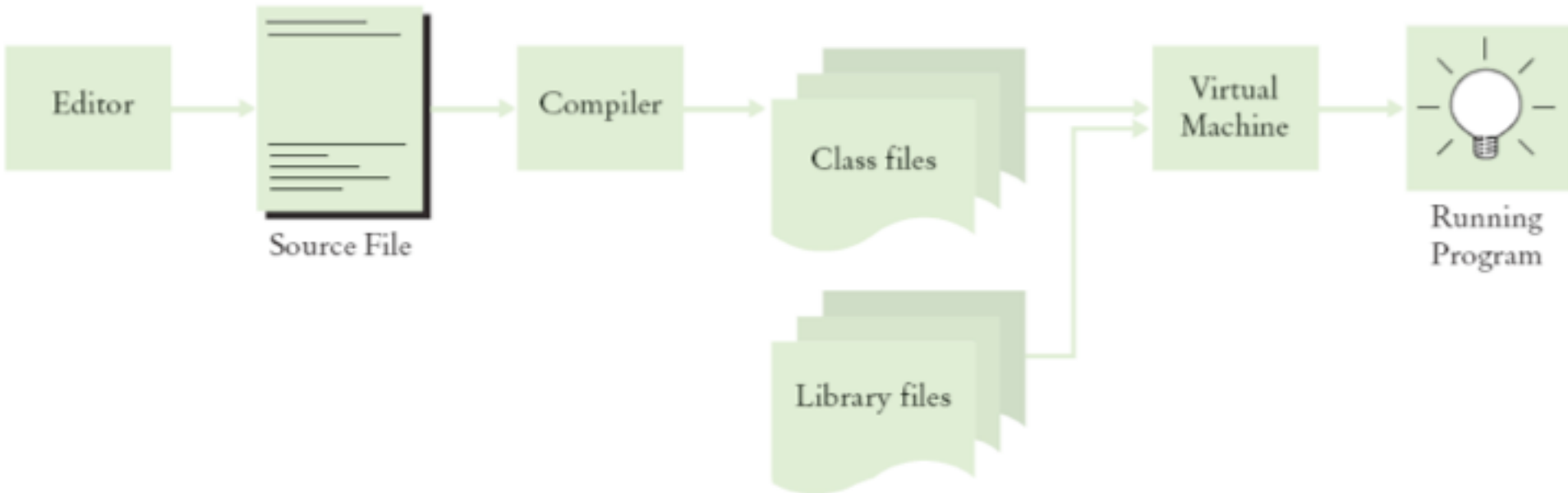
Procédé de programmation



Pourquoi Java ?

- Simplicité
- Correction
- Indépendance vis à vis de la plate-forme
(*write once, run anywhere*)
- Une librairie très riche

Procédé de compilation



Programmation Objet ? Kesako ?

- Programmation dirigée par les données et non par les traitements
- Les procédures/fonctions existent toujours, mais on se concentre :
 - d'abord, sur les entités à manipuler
 - ensuite, comment les manipuler
- Notion d'encapsulation :
 - les données et les procédures liées sont regroupées au sein d'une même entité
 - cacher le fonctionnement interne d'une entité

I^{ère} Partie : Concepts et manipulations

Types

- Chaque valeur/expression a un type
- Exemple :
 - "bonjour" : **type** `String` (chaîne de caractères)
 - 27 : **type** `int` (entier)
 - 'x' : **type** `char` (caractère)
 - true : **type** `boolean` (valeur booléenne)

Variables

- Variable :
 - Stocke une valeur
 - Peut être utilisée à la place de la valeur qu'elle stocke

- Définition d'une variable :

nomDuType nomVariable = valeur;

ou

nomDuType nomVariable;

opérateur d'affectation

- Par exemple :

```
String greeting = "Hello, Dave!";
```

Variables : affectation

```
int luckyNumber = 13; ①
```

```
luckyNumber = 12; ②
```

①

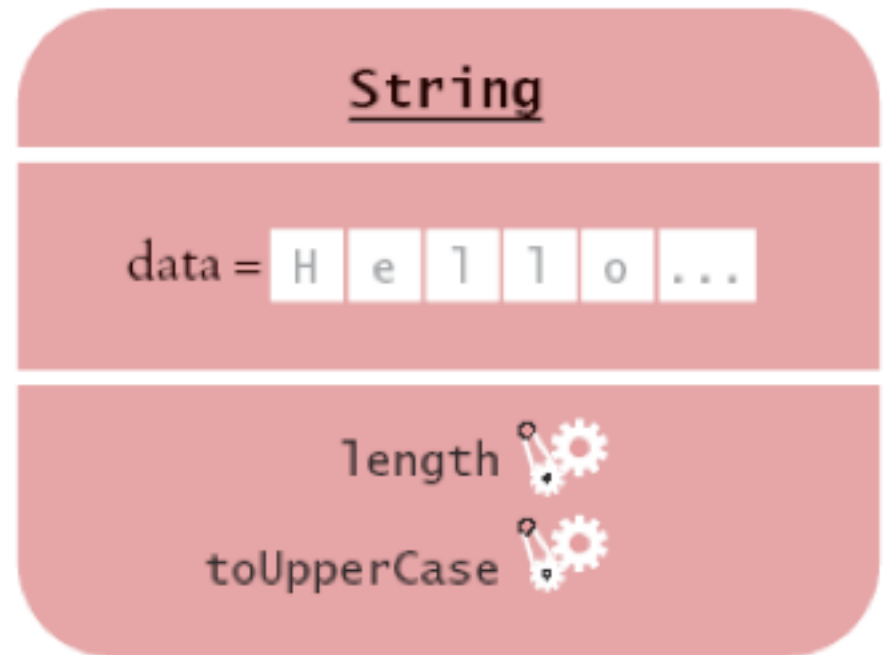
luckyNumber = 13

②

luckyNumber = 12

Objet

- Objet : une entité manipulée dans un programme (en appelant des méthodes)
- Un objet est caractérisé par :
 - Son identité :
 - Unicité
 - Son type
 - Son état :
 - valeurs des attributs à un moment donné
 - Son comportement :
 - ensemble des méthodes (consultation, mise à jour)

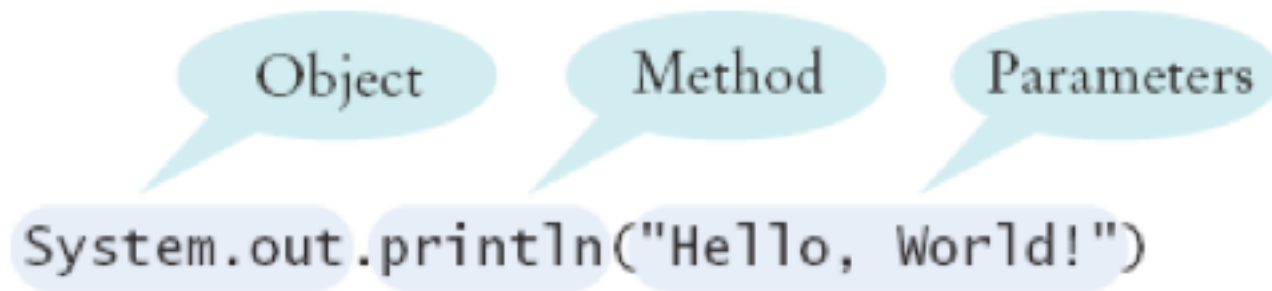


Classe

- Définition d'une famille d'objets ayant une même structure et un même comportement caractérisée par un nom
- Chaque objet appartient à une classe
- Permet d'instancier une multitude d'objets
- Convention d'écriture
 - `NomDeClasse objetA`

Méthode

- Méthode : séquence d'instructions qui accèdent aux données d'un objet
- On manipule des objets par des appels de ses méthodes



- Interface publique :
 - définit ce l'on peut faire sur un objet

Méthode /2

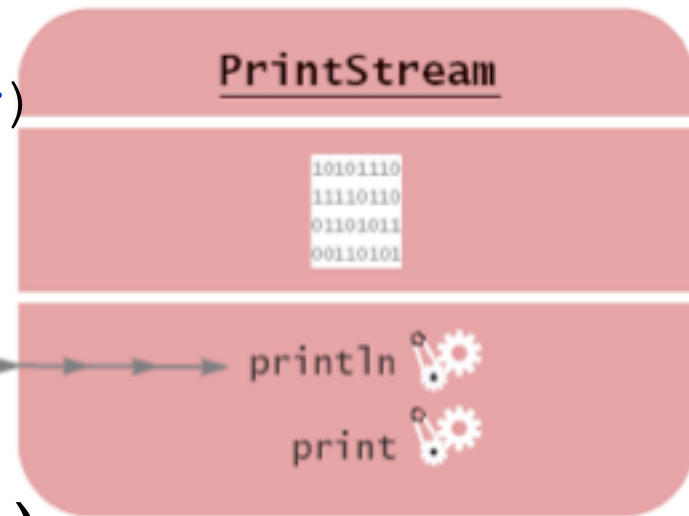
- La classe d'un objet détermine les méthodes que l'on peut appeler sur un objet
- `String greeting = "Hello, World! "`
- `length()` : **compte le nb de caractères**
- `int n = greeting.length(); // affecte 13 à n`
- `toUpperCase()` : **crée un nouvel objet String dont les caractères sont en majuscules**
- `String river = "Mississippi";`
- `String bigRiver = river.toUpperCase(); //"MISSISSIPPI"`
- **Quand on appelle une méthode sur un objet, toujours vérifier que cette méthode est définie dans la classe appropriée**
- `System.out.length(); // This method call is an error`

Paramètres explicites et receveur

- Paramètre (paramètre explicite) :
 - données en entrée d'une méthode
 - certaines méthodes n'ont pas de paramètres explicites

- `System.out.println(greeting)`
`greeting.length()`

"Hello, World" →

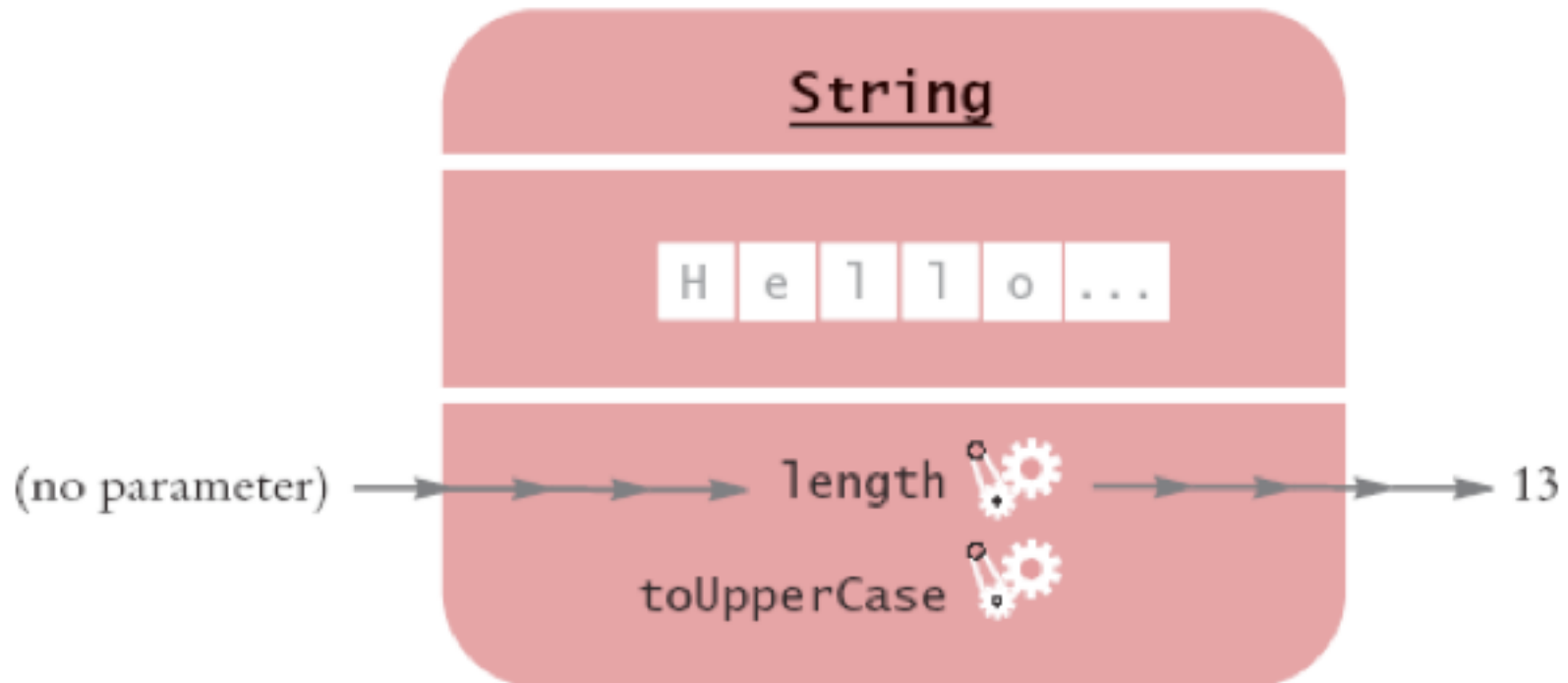


- Receveur (paramètre implicite) :
 - objet sur lequel on invoque la méthode
- `System.out.println(greeting)`

Valeur de retour

- Le résultat calculé par une méthode
- Retournée au code qui a appelé la méthode

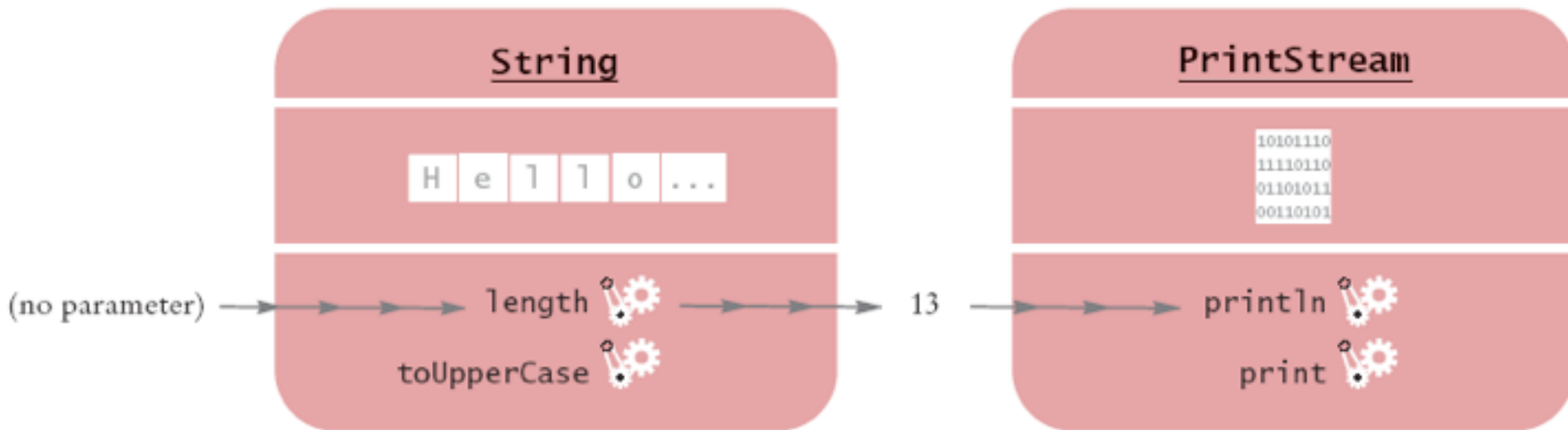
```
int n = greeting.length();  
// retourne une valeur stockée dans n
```



Utilisation d'une valeur de retour

- Une valeur de retour peut être passée en paramètre d'une autre méthode :

```
System.out.println(greeting.length());
```



- Certaines méthodes n'ont pas de valeur de retour : (`println`)

Méthode : définition

- Spécification :
 - nom de la méthode
 - type de la valeur de retour
 - types des paramètres explicites
- Remarque :
 - le type du receveur n'est pas précisé ; classe courante

- Exemple :

```
public int length()  
public String replace(String target,  
                      String replacement)
```

Méthode : définition /2

- Si une méthode ne retourne pas de valeur :
 - type de retour est déclaré comme `void`

```
public void println(String output)  
    //dans la classe PrintStream
```

- Type d'un paramètre explicite ne peut pas être `void`

Méthode : définition : surcharge

- Plusieurs méthodes (ou constructeurs) avec le même nom («*overloading*»)

```
class Point {  
    private int x;  
    private int y;  
    public void translate(int xp,int yp) {  
        x= x+xp; y= y+yp;  
    }  
    public void translate (Point p) {  
        x += p.x; y += p.y;  
    }  
}
```

- En Java, une méthode est identifiée par :
 - nom, nombre de paramètres et types des paramètres
 - Mais, ne prend pas en compte : type de retour

Appel de méthode

objet.nomDeLaMéthode (paramètres)

- **Exemple :**

```
System.out.println("Hello, Dave!");
```

- **Effets :**

- Invoquer une méthode d'un objet et lui passer des paramètres additionnels

Types primitifs : nombre

- Valeur entière :
 - `short, int, long`
 - `13`
- Valeur réelle :
 - `float, double`
 - `1.3`
 - `0.00013`
- Attention : ce ne sont pas des objets
- Classe « *Wrapper* » :
 - `Short, Integer, Long, Float, Double`

Types primitifs : nombre /2

- Opérateurs arithmétiques :
 - $+$, $-$, $*$, $/$, $\%$
 - $($, $)$
- Exemples :
 - $10 + n$
 - $n - 1$
 - $10 * n$
- Ce ne sont pas des méthodes !

Exemple d'objet : Rectangle

- Cette classe représente un `Rectangle` et non pas la figure Rectangle

<u>Rectangle</u>	<u>Rectangle</u>	<u>Rectangle</u>
x = 5	x = 35	x = 45
y = 10	y = 30	y = 0
width = 20	width = 20	width = 30
height = 30	height = 20	height = 20

- 3 objets = 3 instances de la classe `Rectangle`

Constructeurs

- Utilisation :

```
new Rectangle(5, 10, 20, 30)
```

- L'opérateur new :

- construit l'objet de classe Rectangle
- utilise les paramètres pour initialiser les attributs de l'objet
- Retourne le nouvel objet

- Généralement, « l'objet » est conservé dans une variable :

```
Rectangle box = new Rectangle(5, 10, 20, 30);
```

Constructeurs /2

- Créer un objet c'est :
 - Instancier une classe par l'appel d'un constructeur
- Les quatres valeurs 5, 10, 20, et 30 sont les paramètres de construction
- Certaines classes offrent plusieurs constructeurs (*surchage*)

```
new Rectangle()
```

```
new Rectangle(5,10,20,30)
```

Accesseur / Modificateur

- **Accesseur** : ne change pas l'état interne d'un objet (paramètre implicite)

```
double width = box.getWidth();
```

- **Modificateur** : change l'état interne

```
box.translate(15, 25);
```


Exemple

- Ecrire une classe représentant un Rectangle (`Rectangle`)
- Ecrire une classe de test (`MoveTester`)
- Fournir une méthode `main`
- Dans cette méthode, construire plusieurs objets
- Appeler des méthodes sur ces objets
- Afficher les résultats que vous escomptez

Example : solution

```
01: public class Rectangle {
02:     private int x;
03:     private int y;
04:     private int width;
05:     private int height;
06:
07:     public Rectangle(int x, int y, int w, int h) {
08:         this.x = x;
09:         this.y = y;
10:         this.width = w;
11:         this.height = h;
12:     }
13:     public int getX() {
14:         return this.x;
15:     }
16:     public int getY() {
17:         return this.y;
18:     }
19:     public void translate(int dx, int dy) {
20:         this.x = this.x + dx;
20:         this.y = this.y + dy;
21:     }
22: }
```

Exemple : solution

```
01:
02:
03: public class MoveTester
04: {
05:     public static void main(String[] args)
06:     {
07:         Rectangle box = new Rectangle(5, 10, 20, 30);
08:
09:         // Déplacer le rectangle
10:         box.translate(15, 25);
11:
12:         // Afficher les informations concernant le rectangle
13:         System.out.print("x: ");
14:         System.out.println(box.getX());
15:         System.out.println("Expected: 20");
16:
17:         System.out.print("y: ");
18:         System.out.println(box.getY());
19:         System.out.println("Expected: 35");
20:     }
21: }
```

Output:

```
x: 20
Expected: 20
y: 35
Expected: 35
```

Référence (d'un objet)

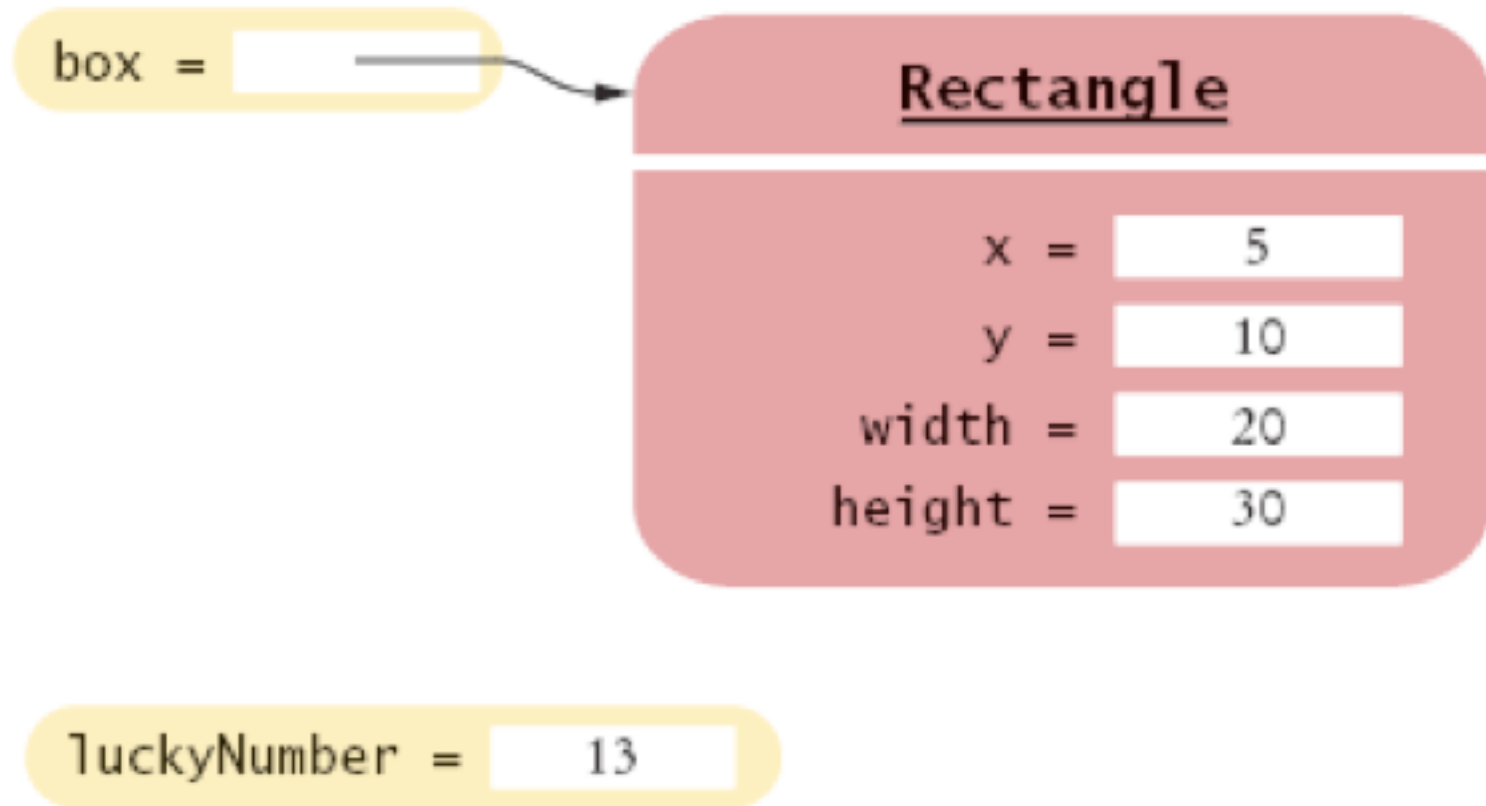
- Référence : décrit la localisation d'un objet
- Opérateur `new` retourne une référence vers un nouvel objet

```
Rectangle box = new Rectangle();
```

Référence / valeur

```
int luckyNumber = 13;
```

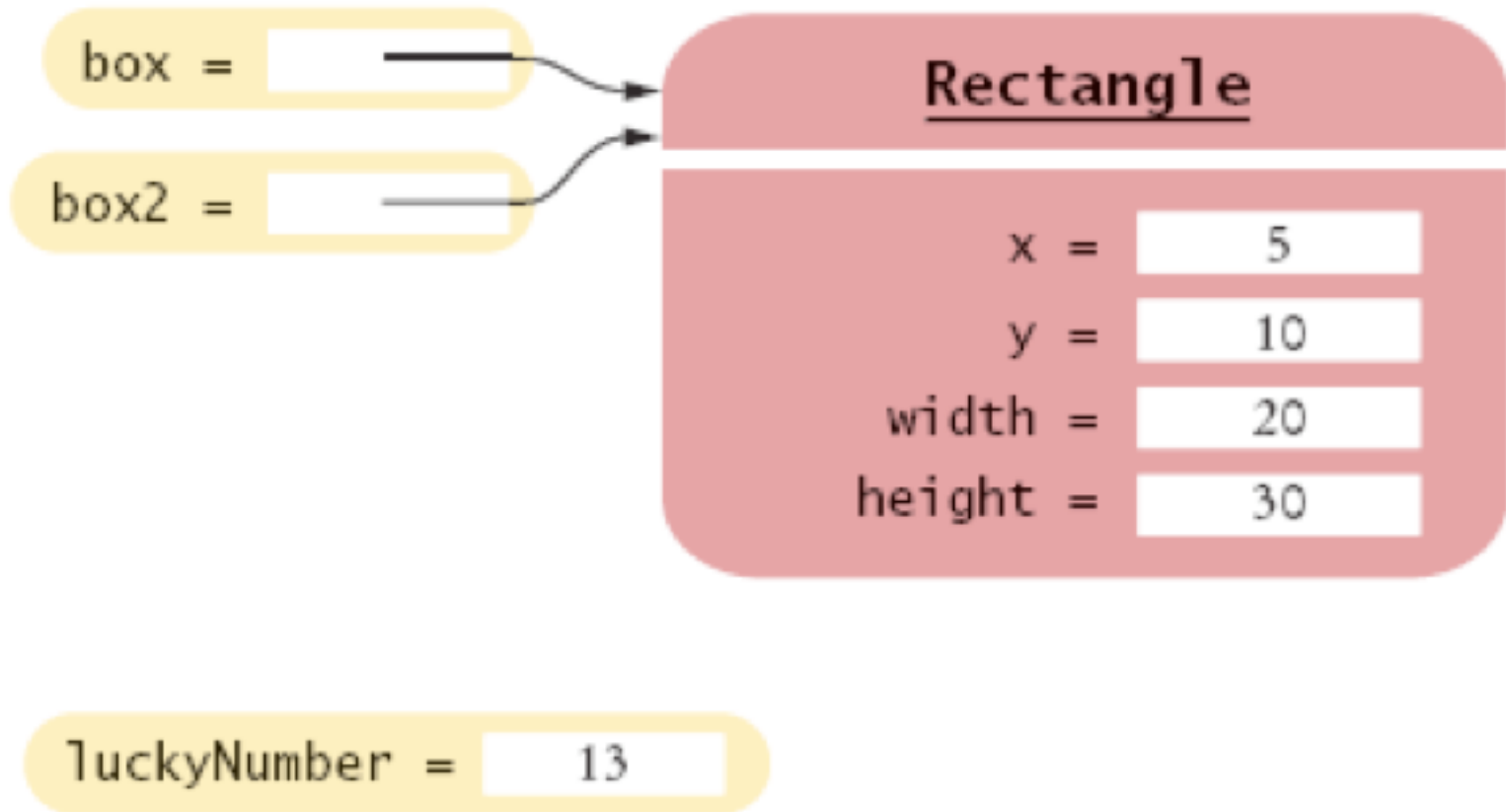
```
Rectangle box = new Rectangle(5, 10, 20, 30);
```



Plusieurs variables peuvent référencer un même objet

```
Rectangle box = new Rectangle(5, 10, 20, 30);
```

```
Rectangle box2 = box;
```



Copie d'une valeur

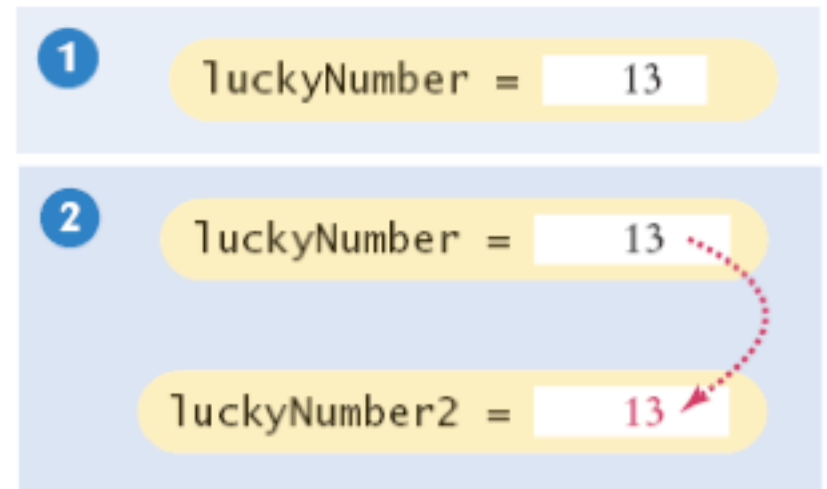
- `int luckyNumber = 13;` ①

①

`luckyNumber = 13`

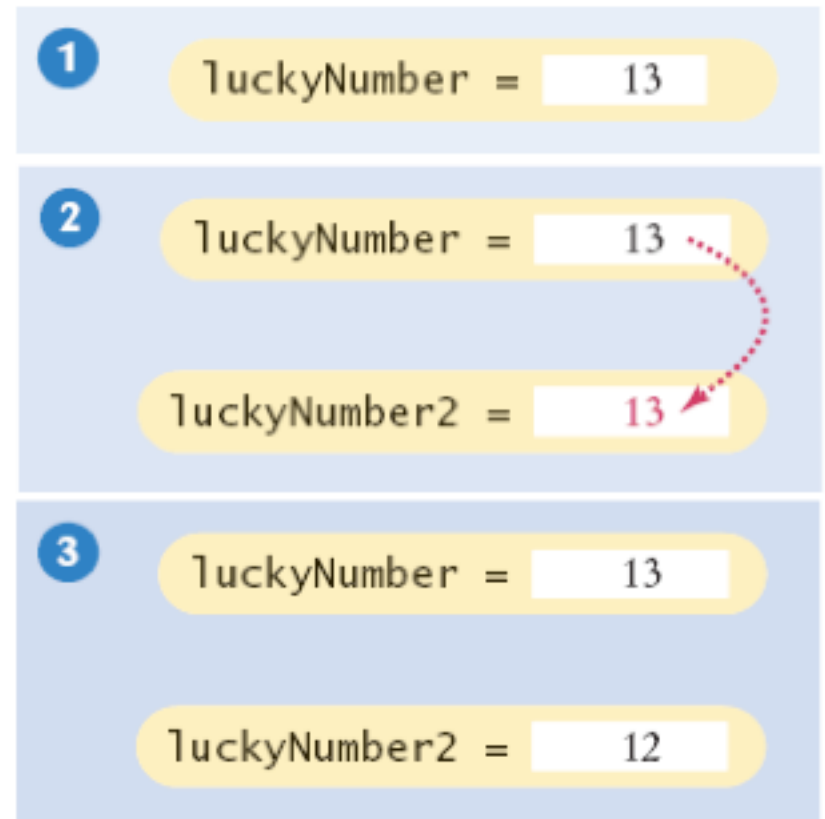
Copie d'une valeur

- `int luckyNumber = 13;` ①
- `int luckyNumber2 = luckyNumber;` ②



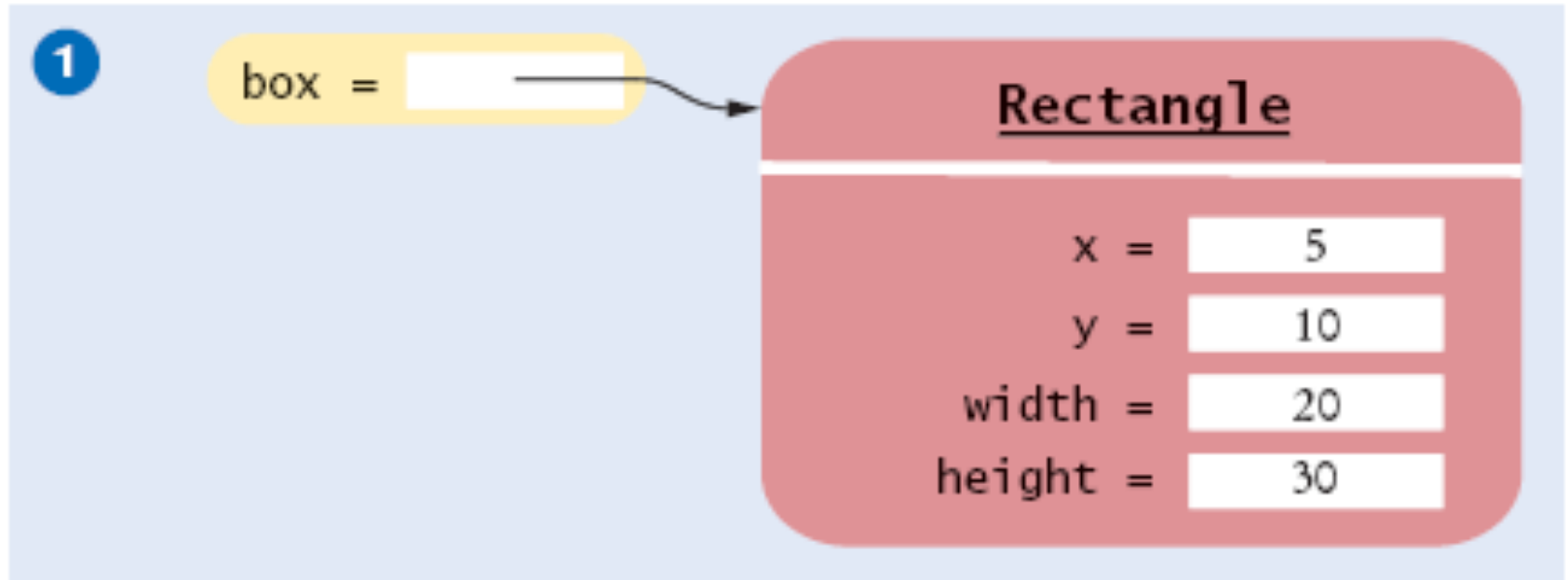
Copie d'une valeur

- `int luckyNumber = 13;` ①
- `int luckyNumber2 = luckyNumber;` ②
- `luckyNumber2 = 12;` ③



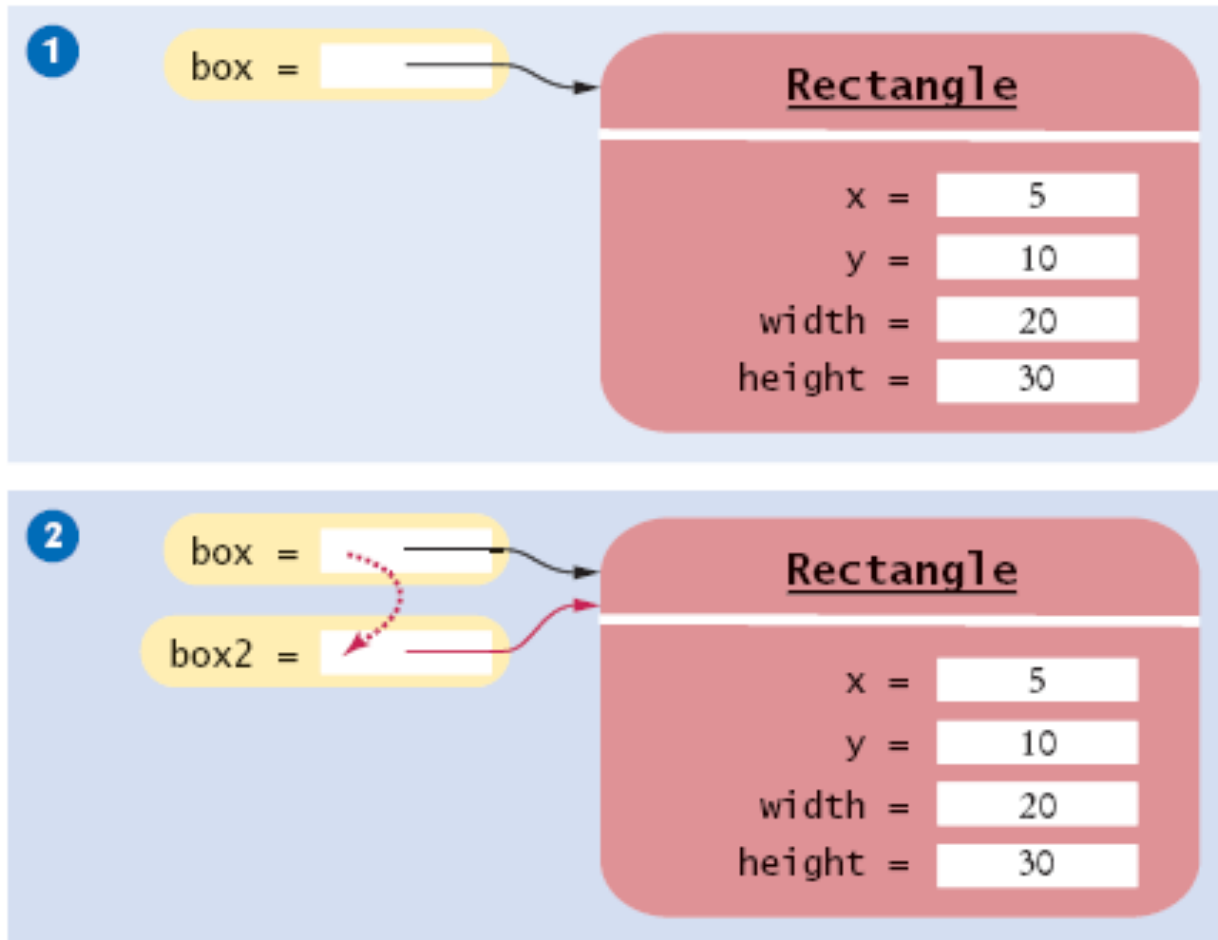
Copie d'une référence

```
Rectangle box = new Rectangle(5, 10, 20, 30);
```



Copie d'une référence

```
Rectangle box = new Rectangle(5, 10, 20, 30);  
Rectangle box2 = box;
```



Modification de l'état d'un objet

```
Rectangle box = new Rectangle(5, 10, 20, 30); ①
```

```
Rectangle box2 = box; ②
```

```
box2.translate(15, 25); ③
```

