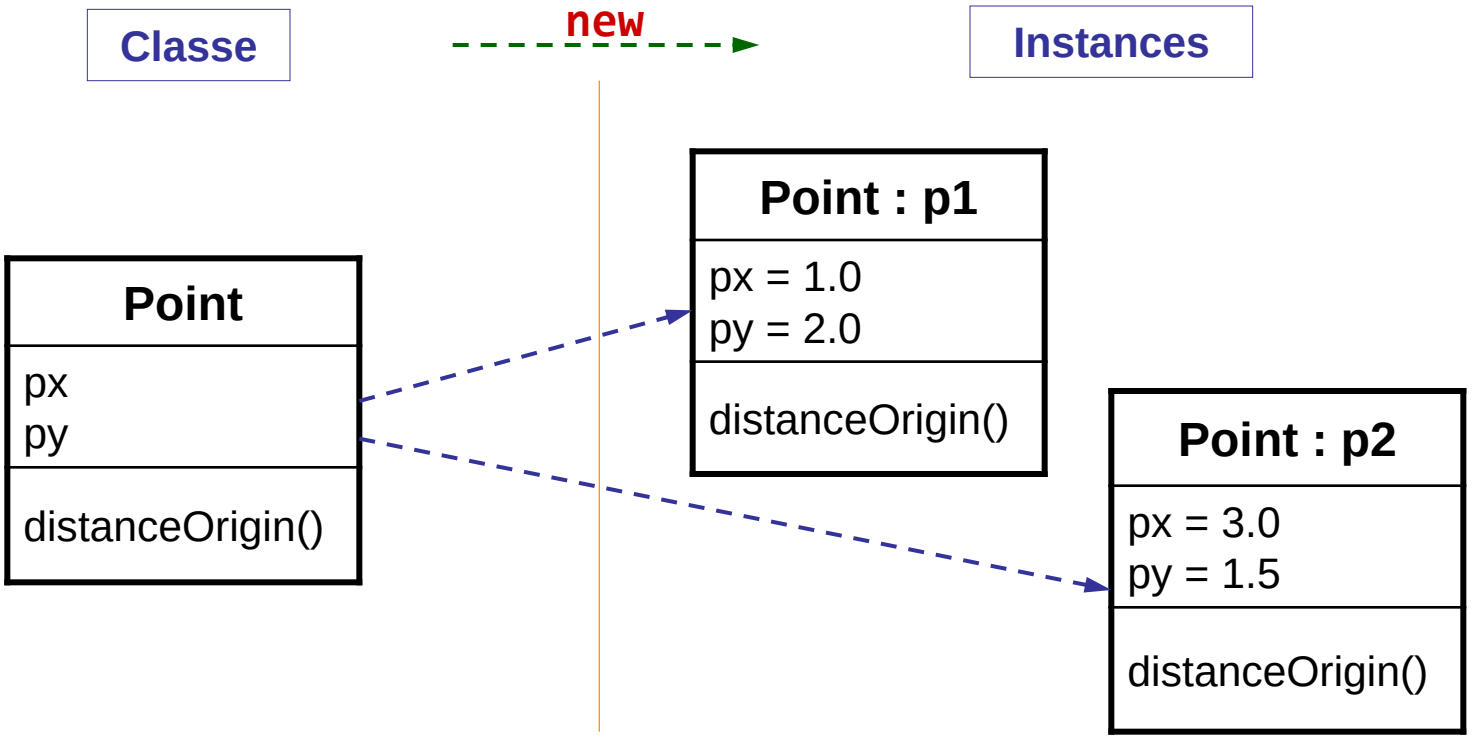
Programmation orientée objet avec Java

Membres statiques, wrappers ...

Membres d'instance

- Par défaut, lors de la création d'un objet (instanciation d'une classe avec l'opérateur **new**), les **membres** (champs et méthodes) sont **associés à chacune des instances** de la classe.
  On les appelle **membres d'instance**.



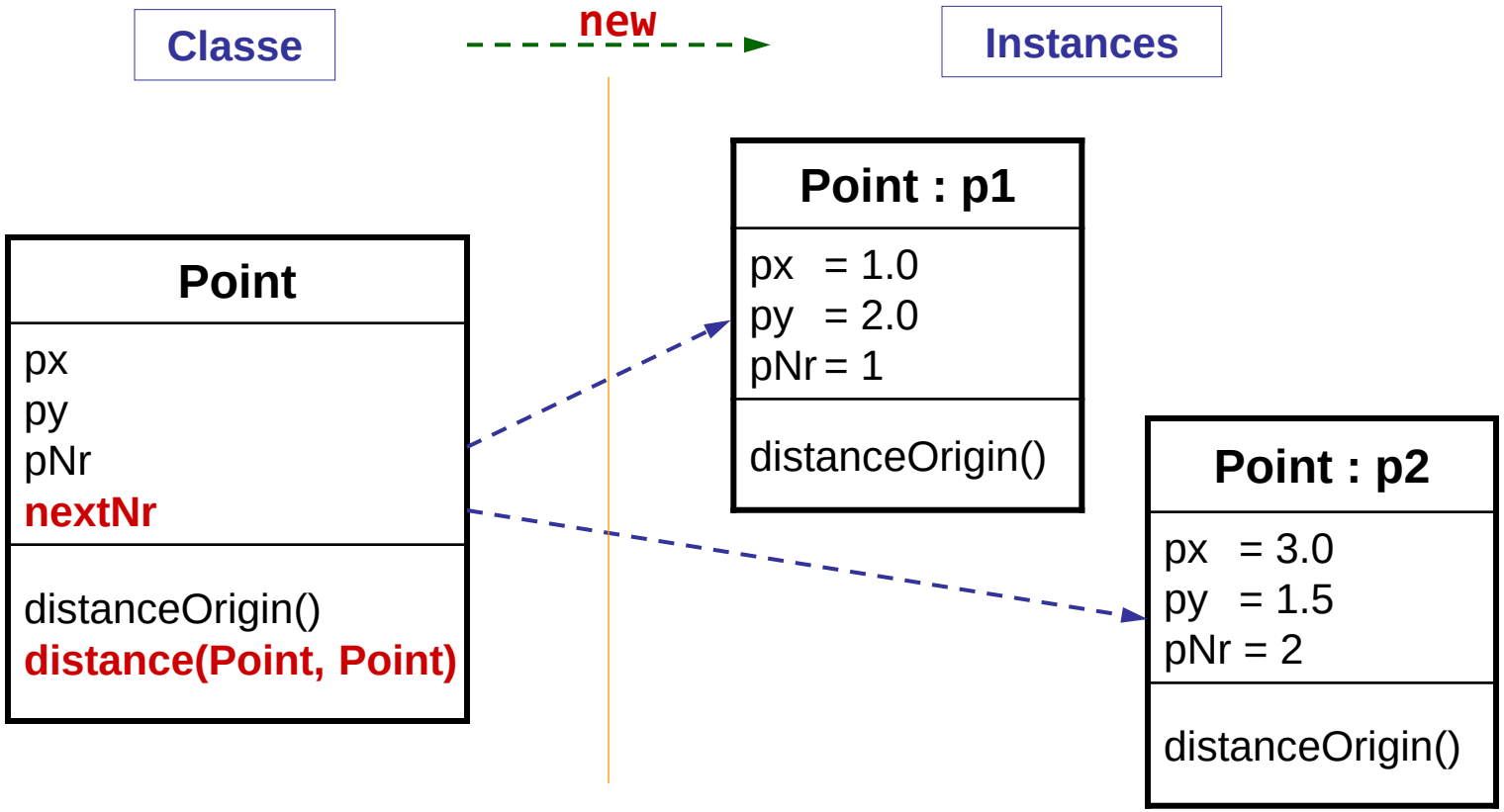
Membres de classe [1]

- On peut également créer des **membres associés à la classe** qui sont appelés **membres de classe** ou **membres statiques**.
- Pour rendre un membre statique, on utilise le mot-clé (modificateur) **`static`** devant la déclaration du champ ou de la méthode :

```
public class Point {

  private       double px;                    // Champ d'instance
  private       double py;                    // Champ d'instance
  private        int    pNr;                  // Champ d'instance
  private static int    nextNr = 1;           // Champ statique

  public Point(double x, double y) {          // Constructeur
    . . .
    pNr = nextNr++;
  }

  public double distanceOrigin() {…}          // Méthode d'instance

                                              // Méthode statique
  public static double distance(Point p1, Point p2) {…}
}
```

Membres de classe [2]

- Les **membres de classe** ne sont pas associés aux instances (objets) mais seulement à la classe. Ils sont accessibles par toutes les instances de la classe (ils sont partagés par toutes les instances).



Membres de classe [3]

- On peut donc déclarer :
  - des **champs statiques** (ou **champs de classe**)
  - des **méthodes statiques** (ou **méthodes de classe**)
- Les **champs statiques** sont **enregistrés au niveau de la classe** et ils peuvent être accédés et manipulés par toutes les instances (objets) de cette classe.
- Indépendamment du nombre d'objets créés, il n'existe **qu'un seul exemplaire** des données associées aux champs statiques (contrairement aux champs d'instance dont il existe un exemplaire pour chaque objet créé).
- On peut considérer les **champs statiques** comme étant des **variables globales partagées par toutes les instances**.
  Il faut donc les utiliser avec précaution et parcimonie afin d'éviter des couplages inutiles (ou même dangereux) entre objets.

Membres de classe [4]

- Les **méthodes statiques** sont liées à une classe et non pas à une instance (objet) de la classe.
- Dans une **méthode statique**, on ne peut pas faire référence à une méthode d'instance sans créer d'objet, car les méthodes statiques ne s'exécutent pas dans le contexte d'un objet (autrement dit, pour les méthodes statiques, il n'existe pas de référence `this`).
- Pour accéder à un membre statique d'une classe en dehors de cette classe, il faut **préfixer le nom du membre** (champ ou méthode) **avec le nom de la classe** (ou utiliser `import static`…).

```
Point  p1 = new Point(1.0, 2.0);
Point  p2 = new Point(2.0, 3.0);
int     n = Point.nextNr;              // Champ statique
double d1 = Point.distance(p1, p2);    // Méthode statique
double d2 = p1.distanceOrigine();      // Méthode d'instance
```

Importation statique

- Il est possible d'**importer** sélectivement ou globalement les **membres statiques** d'une classe en ajoutant le mot-clé `static` à l'instruction `import`.
- Cette importation statique permet, dans certains cas, d'alléger l'écriture en évitant de devoir préfixer les champs et les méthodes statiques avec le nom du package et/ou de la classe.
- Par exemple :

```
import static java.lang.Math.max;
```

  importe la méthode statique **`max()`** qui pourra donc être invoquée sans préfixe : `r = max(v1, v2);`

- Pour importer tous les membres statiques de la classe **`Math`** :

```
import static java.lang.Math.*;

a = PI * pow(r, 2);
```

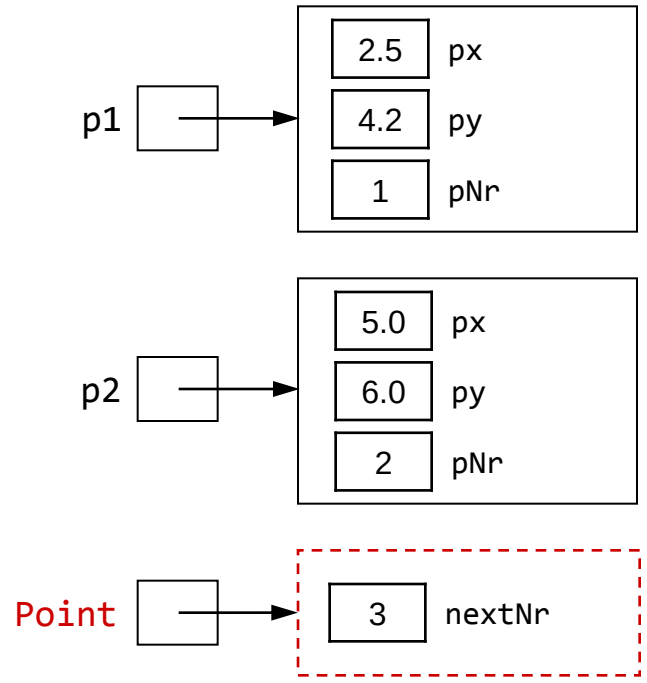
Représentation en mémoire

- Les **champs statiques** existent et sont accessibles même si l'on n'a pas créé d'objets.
- On les représentera donc dans une zone mémoire liée à la classe et commune à toutes les instances de cette classe.

```
p1 = new Point(2.5, 4.2);
p2 = new Point(5.0, 6.0);
n  = Point.nextNr;
n  = p1.nextNr;
```

Bien que déconseillée, cette notation est acceptée par le langage.
Le compilateur la remplacera par
`n = Point.nextNr;`

Le champ statique `nextNr` est associé à la classe **`Point`** et est commun à toutes ses instances



Modificateur `final`

- Le modificateur **`final`** peut être utilisé lors de la déclaration de variables locales, de paramètres de méthodes et de champs.
- Il indique que la valeur de la variable ou du champ ne peut plus être modifiée après l'affectation initiale.
  Cela revient donc à **déclarer des constantes**.
- Il est très fréquent de déclarer les constantes générales comme **champs statiques** au niveau de la classe et de les déclarer `public`.
  Exemple : `public static final double PI = 3.141592653589793;`
- Les champs déclarés constants (`static final`) sont habituellement écrits en majuscules et utilisent le caractère sous-ligné '_' comme séparateur.
  Exemple : `private static final double FREQUENCY_MAX = 3.5E9;`
- Pour les classes et les méthodes, une autre utilisation du mot-clé `final` sera vue dans le cadre de l'héritage.

Wrappers [1]

- En *Java*, pour des raisons d'efficacité, les types primitifs ne sont pas des objets.
- Dans certaines circonstances, il peut être utile de pouvoir traiter les types primitifs comme des objets (par exemple pour pouvoir les enregistrer dans des structures de données abstraites de type *liste*, *pile*, *arbre*, … ne manipulant généralement que des objets).
- Ainsi, il existe pour chaque type primitif, une classe ***Wrapper*** (classe d'emballage, classe enveloppe) qui permet de convertir (d'emballer) une variable (ou une valeur littérale) de type primitif en un objet correspondant (une instance d'une des classes d'emballage).
- Les classes *Wrapper* sont déclarées dans le paquetage `java.lang` (et sont donc accessibles sans importation explicite).
- Les classes *Wrapper* disposent d'un certain nombre de constantes et de méthodes (statiques et non-statiques) permettant d'effectuer diverses conversions.

Wrappers [2]

- Liste des classes d'emballage *(Wrappers) :*

Type primitif	Classe Wrapper	
`byte`	`Byte`	
`short`	`Short`	
`int`	`Integer`	*(Et non pas* `Int` *!)*
`long`	`Long`	
`float`	`Float`	
`double`	`Double`	
`char`	`Character`	*(Et non pas* `Char` *!)*
`boolean`	`Boolean`	

- A deux exceptions près, le nom de la classe *Wrapper* correspond à celui du type primitif avec une majuscule initiale.
- Les classes *Wrapper* créent des **objets immuables** !

Wrappers [3]

- Exemples d'utilisation des classes *Wrapper* :

```
String str   = "1.23";
float  primF = 3.456F;
Float  objF;

primF = Float.MAX_VALUE;          // Plus grande valeur positive de type float
primF = Float.MIN_VALUE;          // Plus petite valeur positive de type float

objF  = new Float(primF);         // Conversion float  -> Float
primF = objF.floatValue();        // Conversion Float  -> float
objF  = new Float("2.34");        // Conversion String -> Float
objF  = Float.valueOf(str);       // Conversion String -> Float
str   = objF.toString();          // Conversion Float  -> String
str   = Float.toString(primF);    // Conversion float  -> String
primF = Float.parseFloat(str);    // Conversion String -> float
```

Ces exemples, donnés pour les types `float` et `Float`, s'appliquent par analogie aux autres types primitifs (avec quelques légères différences pour les types `boolean` et `char`).