LIFI-Java 2004

Séance du Jeudi 16 sept.

Cours 2

Programmation Objet

- Qu'est ce que c'est?
  - une "facilité" de programmation?
  - une nouvelle façon de penser?
- But du cours
  - Découvrir les concepts
    - classes, instances
    - interface vs. implémentation
    - héritage, polymorphisme
  - Les appliquer en Java

Pourquoi des objets? (1/3)

- On peut tout faire avec des variables!
- Oui, mais...un exemple:
  - modéliser une course de voitures
    - des variables pour décrire les voitures
      - vitesse `v`
      - position `x`
    - des instructions pour faire évoluer ces variables
      - déplacer une voiture `x=x+v`
      - accélérer `v=v+1`
      - afficher une voiture `drawCarAt(x)`

Pourquoi des objets? (2/3)

- Le code est “lourd”:
  - plusieurs voitures `->` plusieurs variables
    - voiture 1 `v1,x1`
    - voiture 2 `v2,x2`
    - etc...
  - les variables “internes” sont exposées
    - exemple: -vitesse max `vmax1`
      -puissance `puis1`
      -accélérer `v1 = max(v1+puis1,vmax1)`

Pourquoi des objets? (3/3)

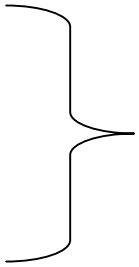
- Le code est difficile à réutiliser
  - notion d'API et de librairie
    - un client veut une fonction
    - il ne veut pas connaître les détails
    - *vous ne voulez pas non plus*!
  - conflits de noms de variables
    - pas de solution simple!
  - toutes les variables sont "globales"
    - problème de contrôle d'accès et de cohérence

Le(s) problème(s)

- Comment “grouper” des données?
  - rattachées à un même “objet”
- Comment “cacher” des données/actions?
  - variables internes à un calcul
  - fonctions intermédiaires
- Comment contrôler l’accès/la modification
  - ex: interdire d’augmenter la puissance

Solution: les objets (1/2)

- *Encapsule* (variables+comportements)
  - classe et instances
  - membres `public`,`private`
- Abstrait les données
  - interface vs. implémentation
- Et plus encore!
  - héritage
  - polymorphisme



prochain cours

Solution: les objets (2/2)

- Un exemple:

```
Vehicule v = new Vehicule[2];
v[0] = new Car();
v[1] = new Moto();

while (true) {
    for (int i=0;i<v.size();++i) {
          if (joystick(i).button().isPressed()) {
              v[i].accelerate();
          }

          v[i].advance();
          v[i].draw();
    }
}
```

Classe *vs.* Instance

- Classe = description d'un modèle
  - variables
  - fonctions

  } membres
- Instance = un exemplaire du modèle

- 1 seule classe, plusieurs instances

Ex: définition d'une classe

CLASSE

```
public class Point {
    public double x;
    public double y;

    Point(double ax,double ay) {
        x = ax;
        y = ay;
    }

    public String toString() {
        return "("+x"+","+y+")";
    }
}
```

Ex: création d'instances

INSTANCES

```
Point p = new Point(1,2);

System.out.println(p.x);
System.out.println(p.y);

p.x = 3;

System.println(p.toString());

Point q = new Point(0,0);

double dx = q.x-p.x;
double dy = q.y-p.y;
double l = Math.sqrtf(dx*dx+dy*dy);
```

Classe

- Variables
- Méthodes
- Constructeurs...
- Destructeurs...

CLASSE

```
public class Point {
    public double x;
    public double y;

    public String toString() {
        return "("+x"+","+y+")";
    }

    Point(double ax,double ay) {
        x = ax;
        y = ay;
    }

    protected void finalize() {
    }
}
```

Constructeurs (1/5)

- Rappel : toute variable doit être initialisée
- Question : comment initialiser les variables d'une instance?
- Réponse : le (ou les) constructeurs!

CLASSE

```
public class Point {
    public double x;
    public double y;

    Point(double ax,double ay) {
        x = ax;
        y = ay;
    }
}
```

INSTANCES

```
Point p = new Point(1,2);
```

Constructeurs (2/5)

- Plusieurs constructeurs sont possibles

CLASSE

```
public class Point {
    // ...
    Point(double ax,double ay) {
        x = ax;
        y = ay;
    }
    Point() {
        x = 0;
        y = 0;
    }
}
```

INSTANCES

```
Point p = new Point(1,2);
`
```

```
Point q = new Point();
```

Constructeurs (3/5)

- Constructeurs par défaut

CLASSE

```
public class Point {
    public double x;
    public double y;
}
```

INSTANCES

```
Point p = new Point();

Point q = new Point(1,2);
```

refusé à la compilation

CLASSE

```
public class Point {
    public double x;
    public double y;
    Point(double ax,double ay) {
        x = ax;
        y = ay;
    }
}
```

INSTANCES

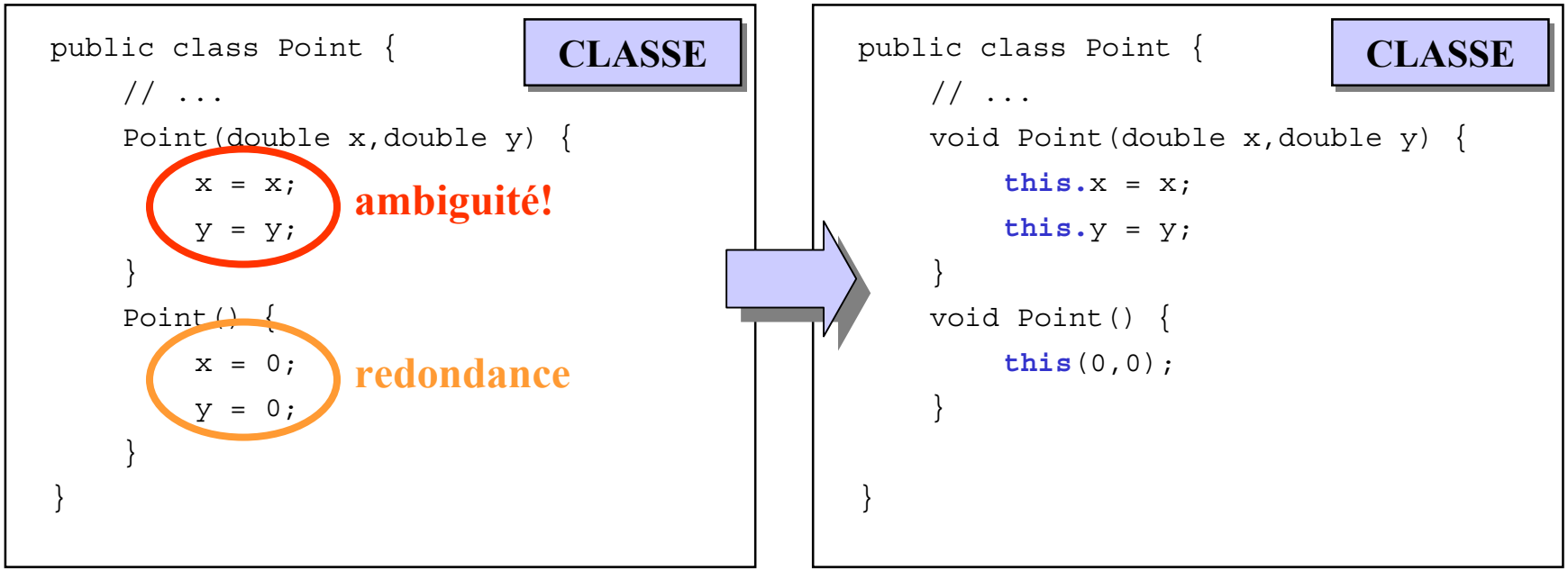
```
Point p = new Point();

Point q = new Point(1,2);
```

refusé à la compilation

Constructeurs (4/5)

- Le mot clé `this` pour:
  - lever les ambiguités
  - réutiliser un constructeur



Constructeurs (5/5)

- Constructeurs *par copie*

CLASSE

```
public class Point {
    public double x;
    public double y;
    Point(double ax,double ay) {
        x = ax;
        y = ay;
    }
    Point(Point p) {
        this(p.x,p.y);
    }
}
```

INSTANCES

```
Point p = new Point(1,2);

Point q = new Point(p);
```

Destructeurs

- Lors de l'instanciation
  - la place nécessaire en mémoire est reservée
  - un compteur de référence est mis en place
- Quand une instance n'est plus référencée
  - elle est détruite (*garbage collect*)
  - la méthode `finalize()` est appellée
- Pas de gestion de la mémoire comme en C!

Instances

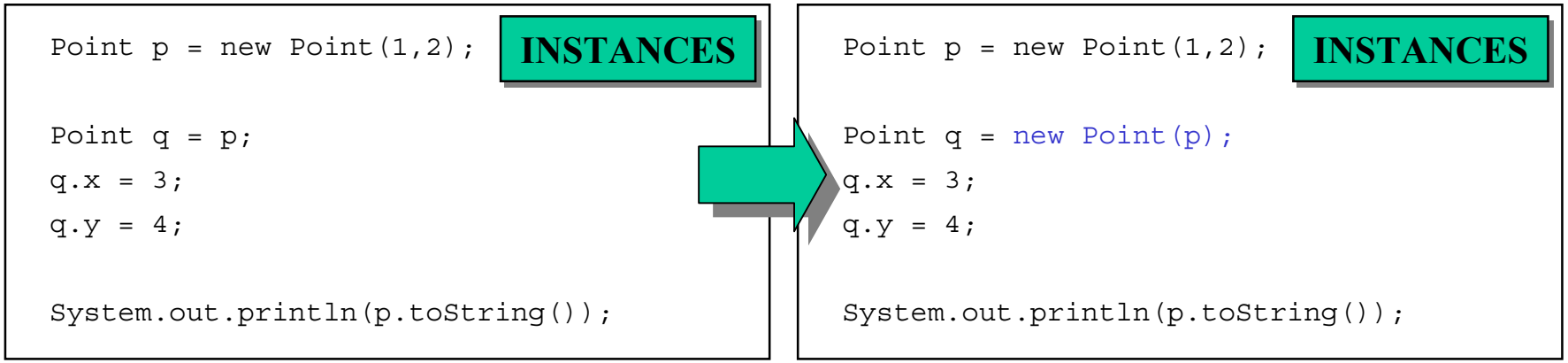
- Création par `new Point()`
- Destruction automatique
  - appel de `finalize()`...

} on vient de le voir

- La subtilité des références...
- Le problème de l'accès...
- Information sur une classe...

Instances (1/3)

- La subtilité des références



affiche (3,4) !!!

affiche (1,2)

- Importance du constructeur par copie!

Instances (2/3)

- Le problème de l'accès

CLASSE

```
public class Point {
    public double x,y;
    public double r,th;
    Point(double ax,double ay) {
        x = ax;
        y = ay;
        r = Math.sqrtf(x*x+y*y);
        th = Math.atan(y/x);
    }
}
```

INSTANCES

```
Point p = new Point(3,4);

p.x = 0;`

System.out.println(p.r);
```

affiche 5: `r` n'a pas été remis à jour en fonction du nouveau `x` !!!

Instances (3/3)

- Information sur une classe
  - toute classe a une méthode `getClass()`
  - `getClass()` renvoie une instance de `java.lang.Class`
    - des `Class` sont comparables entre eux par ==
    - `Class` a une méthode `getName()` qui renvoie un `String`

```
Point p = new Point(1,2);
Point q = new Point(3,4);
String s = new String("salut");

System.out.println(p.getClass() == q.getClass());  // affiche true
System.out.println(p.getClass() == s.getClass());  // affiche false

System.out.println(p.getClass().getName());  // affiche Point
System.out.println(s.getClass().getName());  // affiche String

System.out.println(s.getClass().getClass().getName());  // affiche java.lang.Class
```

Contrôle d'accès (1/4)

- Chaque variable/fonction/constructeur est:
  - `public` : peut être accédé de partout *ou*
  - `private` : peut être accédé de la classe *ou*
  - `protected` : on verra plus tard...

CLASSE

```
public class Point {
    private double x,y;
    private double r,th;
    // ...
}
```

INSTANCES

```
Point p = new Point(3,4);
```

~~`p.x = 0;`~~

refusé à la compilation

Contrôle d'accès (2/4)

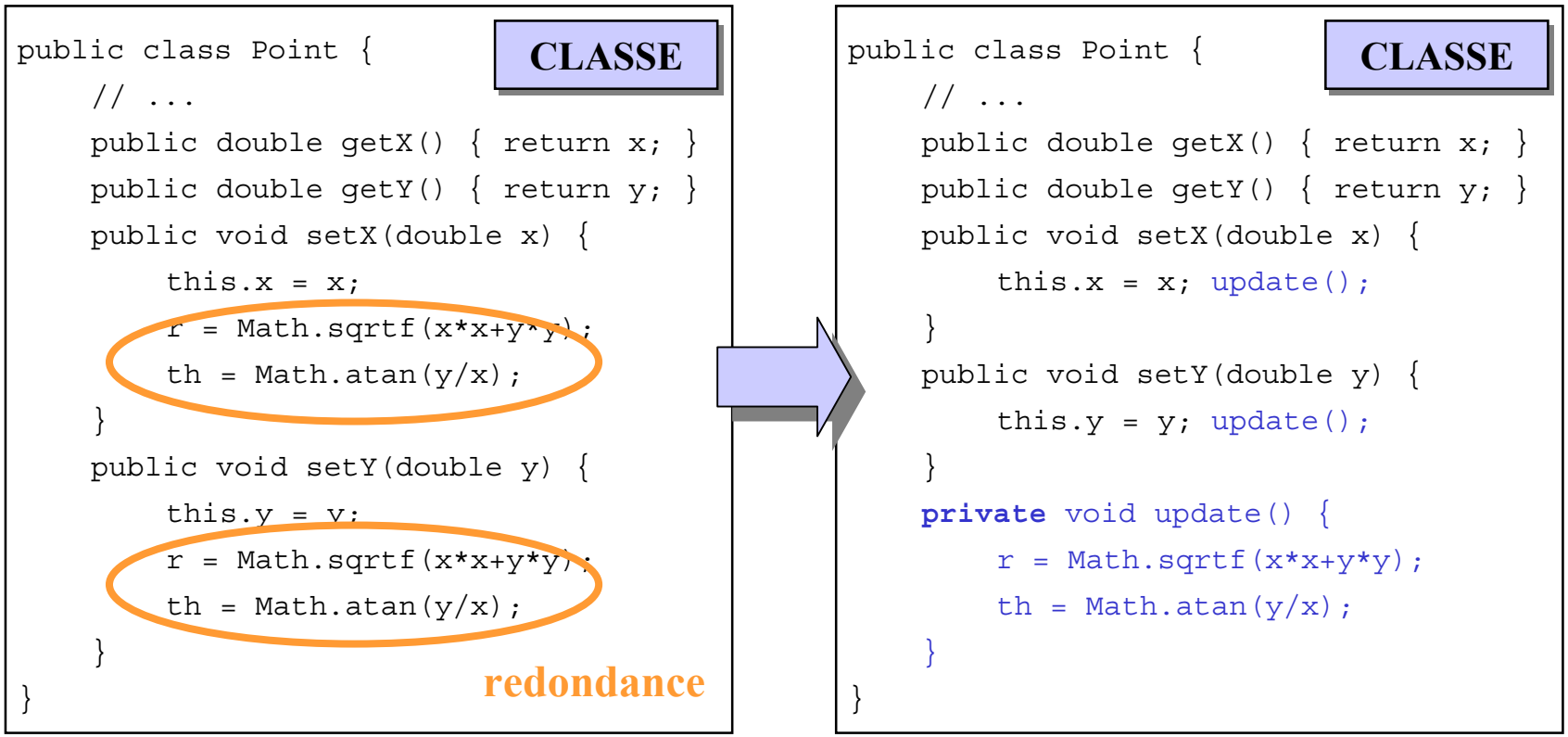
- Règles de "bon design"
  - les variables sont `private`
  - on fournit des *accesseurs*

CLASSE

```
public class Point {
    // ...
    public double getX() { return x; }
    public double getY() { return y; }
    public void setX(double x) {
        this.x = x;
        r = Math.sqrtf(x*x+y*y);
        th = Math.atan(y/x);
    }
    public void setY(double y) {
        this.y = y;
        r = Math.sqrtf(x*x+y*y);
        th = Math.atan(y/x);
    }
}
```

Contrôle d'accès (3/4)

- Règles de "bon design"
  - les fonctions "intermédiaires" sont `private`



Contrôle d'accès (4/4)

- Remarques diverses
  - Si le contrôle d'accès n'est pas précisé:
    - par défaut, il est `private`
    - Il vaut mieux le préciser
  - Le destructeur `finalize()` doit être `protected`

Un exemple: `StringBuffer`

- La classe `String` n'est pas modifiable
  - partage automatique d'instance par Java
  - subtilité avec les tests `==`
- La classe `StringBuffer` est modifiable!
  - `append()`
  - subtilité avec les tests : `equals()`

Conclusion

- Pas de fonctions, que des objets!
  - ex: la fonction `main(String args[])`!
- Difficulté de bien “penser objet”
  - comment encapsuler?
  - quels sont les objets de base à définir?
  - quelles fonctions ces objets doivent proposer?

Exercice

- Définir la classe `Vehicle` du début
  - fichier `Vehicle.java`:
    - définir la classe `Vehicle`
  - fichier `Course.java`:
    - définir une classe `Course` avec une méthode `main(String args[])`
      - qui instancie 3 voitures avec des vitesses différentes
      - les fait avancer aléatoirement
      - arrête quand l'une a parcouru 10km
  - tester le tout

Peut-on aller plus loin?

- Définir des “sous-classes” de `Vehicle`:
  - voitures, camions, motos, ...
- Comment manipuler un ensemble d’instances *hétérogène* de `Vehicle`
- Prochain cours!
  - Héritage
  - Polymorphisme
  - Interface *vs.* implémentation