Programmation orientée objets en Java

Jean-Rémy Falleri@Bordeaux INP

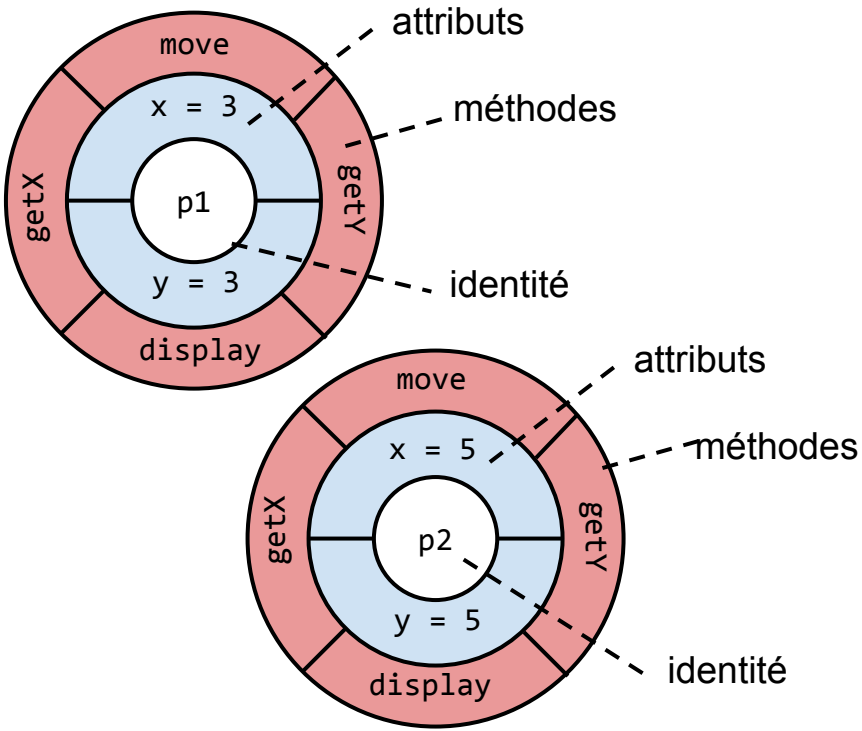
Séance 1

Un point en C

```c
typedef struct {
   int x;
   int y;
} Point;

void move(Point *p, int x, int y) {
   p->x = x;
   p->y = y;
}

void display(Point *p) {
   printf("Point is at (%d,%d).\n", p->x, p->y);
}
```

```c
int main(int argc, char *argv[]) {
   Point *p = malloc(sizeof(Point));
   move(p, 0, 0);
   display(p);
   move(p, 5, 5);
   display(p);
   free(p);
}
```

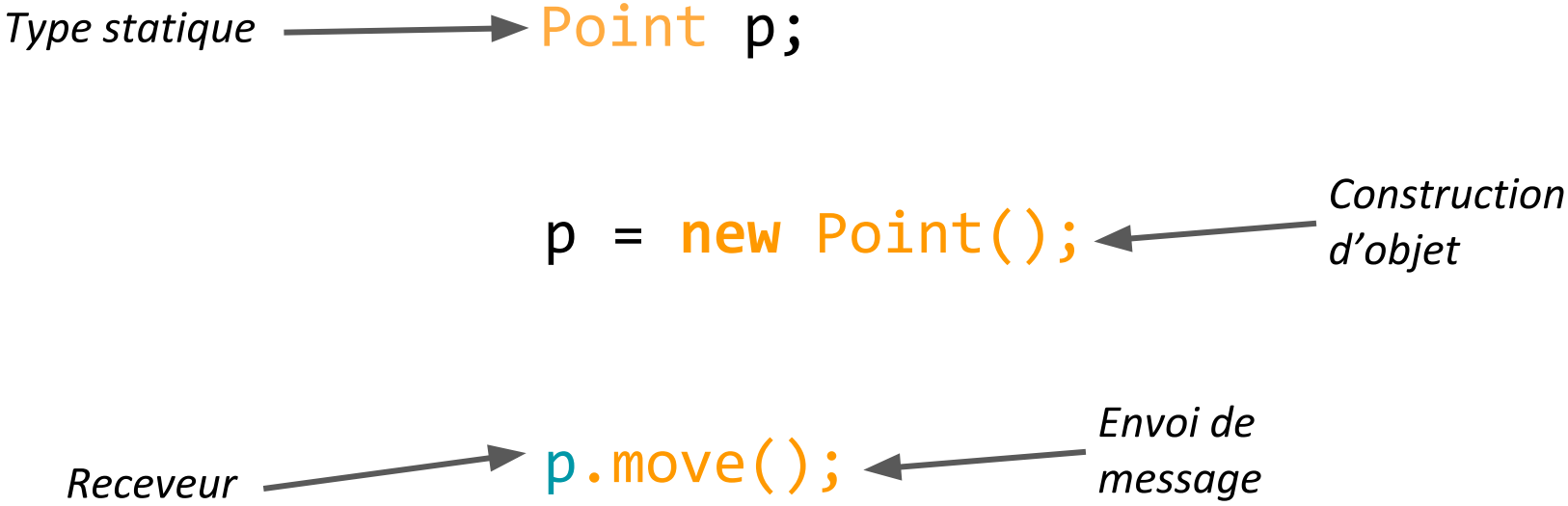
Les objets



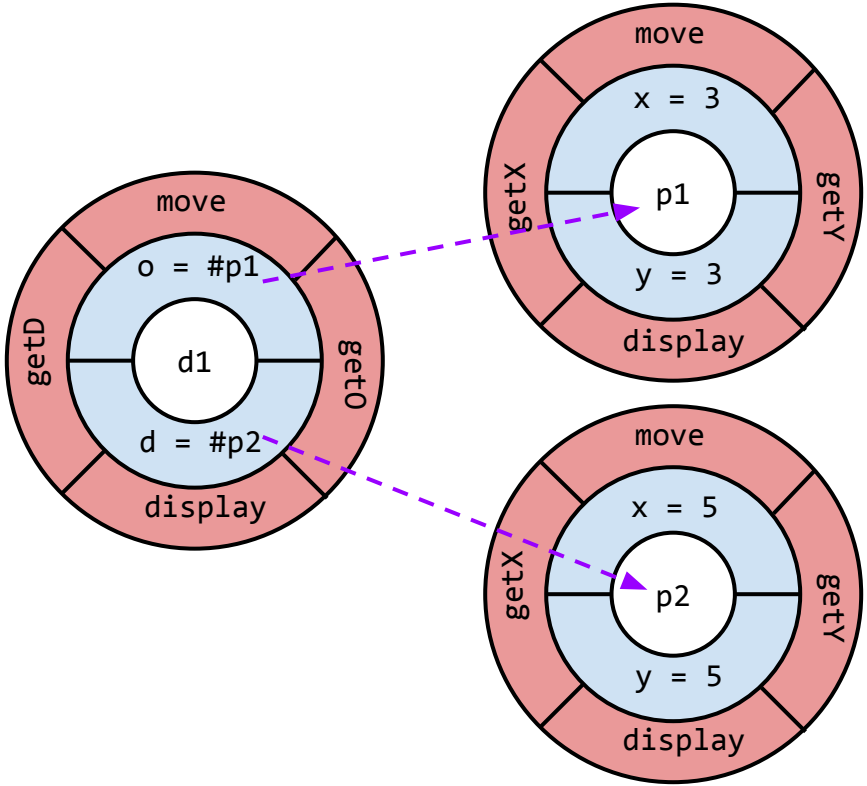

```
Point p1 = new Point();
p1.move(3, 3);
p1.display();

Point p2 = new Point();
p2.move(5, 5);
```

Encapsulation des données

Syntaxe


Type statique
Point p;
p = new Point();
Construction d'objet
Receveur
p.move();
Envoi de message

La délégation



Les classes

- Unité de programmation en Java (on ne programme pas directement des objets)
- Déterminent les caractéristiques des objets
  - Liste des attributs
  - Liste des méthodes
  - Liste des constructeurs
- Servent de moules à objets
  - Via les constructeurs et l'utilisation de new
  - `new Point();`
- Introduisent des types pour les variables
  - `Point p;`

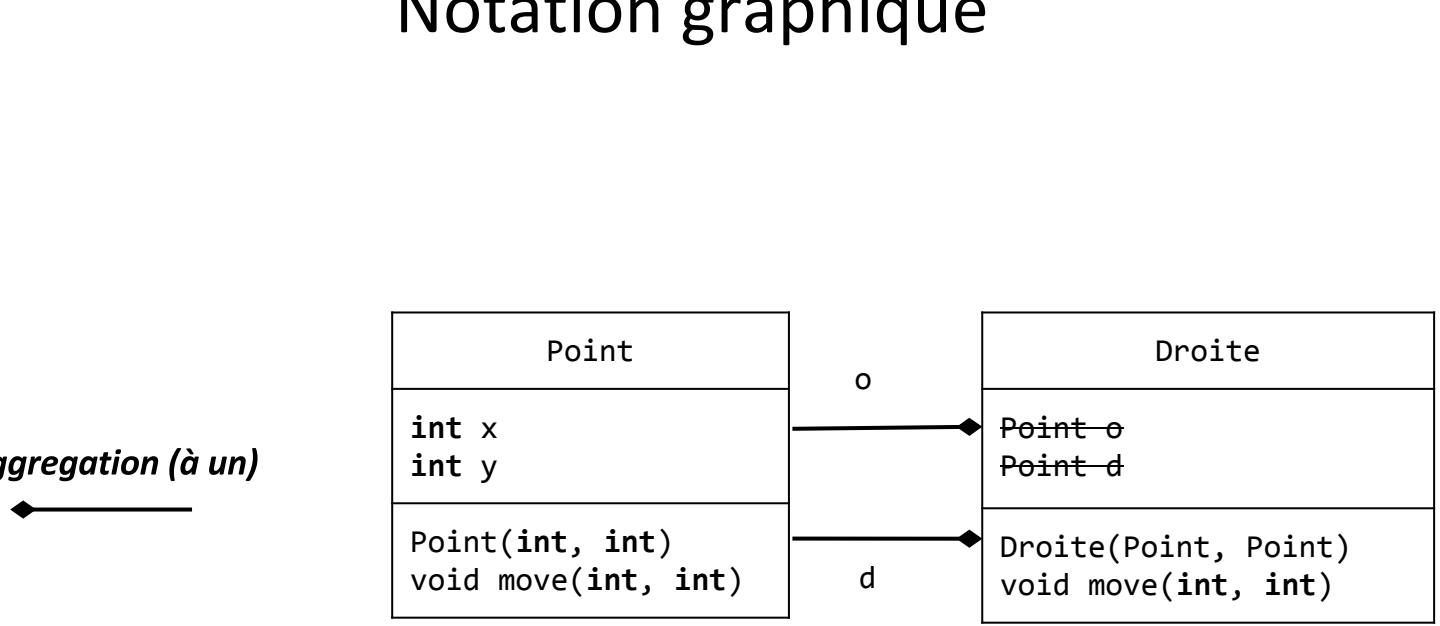
Exemples de classes

```
class Point { // classe
  int x; // attribut
  int y; // attribut

  Point(int x, int y) { // constructeur
    this.x = x;
    this.y = y;
  }

  void move(int x, int y) { // méthode
    this.x += x;
    this.y += y;
  }
}
```

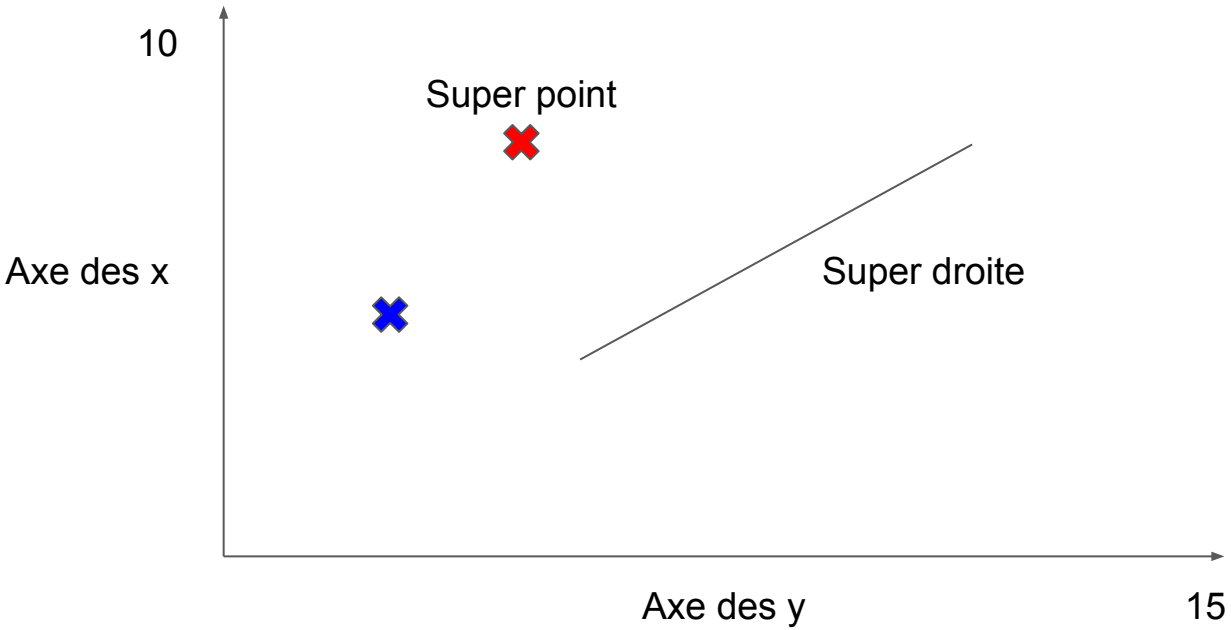
```
class Droite { // classe
  Point o; // attribut
  Point d; // attribut

  Droite(Point o, Point d) { // constructeur
    this.o = o;
    this.d = d;
  }

  void move(int x, int y) { // méthode
    this.o.move(x,y);
    this.d.move(x,y);
  }
}
```

Notation graphique

aggregation (à un)



Un canevas



Exemple de conception objet

Repère

- Attributs
  - titre : Chaîne de caractères
  - points : Ensemble de points
  - droites : Ensemble de droites
  - axe des X : Axe
  - axe des Y : Axe
- Méthodes
  - ajouter point (Point p)
  - ajouter droite(Droite d)
  - titre(Chaîne c)

Axe

- Attributs
  - taille : Entier
  - titre : Chaîne de caractères
- Méthodes
  - tailleMax(Entier e)
  - titre(Chaîne c)

Point

- Attributs
  - titre : Chaîne de caractères
  - couleur : Couleur
  - abscisse : Entier
  - ordonnée : Entier

Droite

- Attributs
  - origine: Point
  - arrivée : Point
  - titre : Chaîne de caractères
  - couleur : Couleur

Couleur

- Attributs
  - r : Entier
  - g : Entier
  - b : Entier

Exemple de conception objet

EnsembleDePoints

- Attributs
  - points : Point[]
- Méthodes
  - ajouter point (Point p)
  - enlever point(Point p)

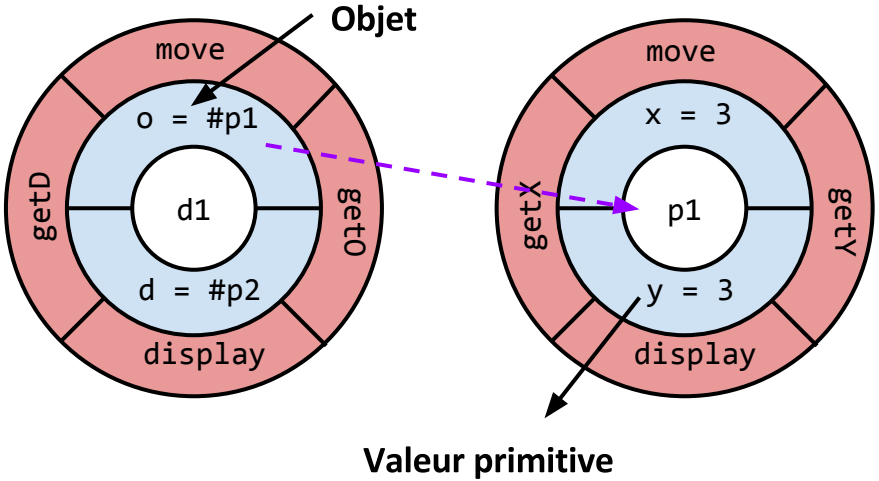
Couleur

- Attributs
  - r : Entier
  - g : Entier
  - b : Entier
- Méthodes
  - r(Entier e)
  - g(Entier e)
  - b(Entier e)

EnsembleDeDroites

- Attributs
  - droites: Droite[]
- Méthodes
  - ajouter droite(Droite d)
  - enlever droite(Droite d)

Objets et valeurs primitives




```
Point p = new Point(0,0);
p.move(5, 5); // p est un objet

int i = 0; // i est une valeur primitive
int j = i + 5
```

Passage de paramètres

Les objets sont passés par **référence**

```
void foo(Point p) {
    p.move(0, 0);
}

Point p = new Point(10, 10);
foo(p); // p est à (0, 0)
```

Les valeurs primitives sont passées par **copie**

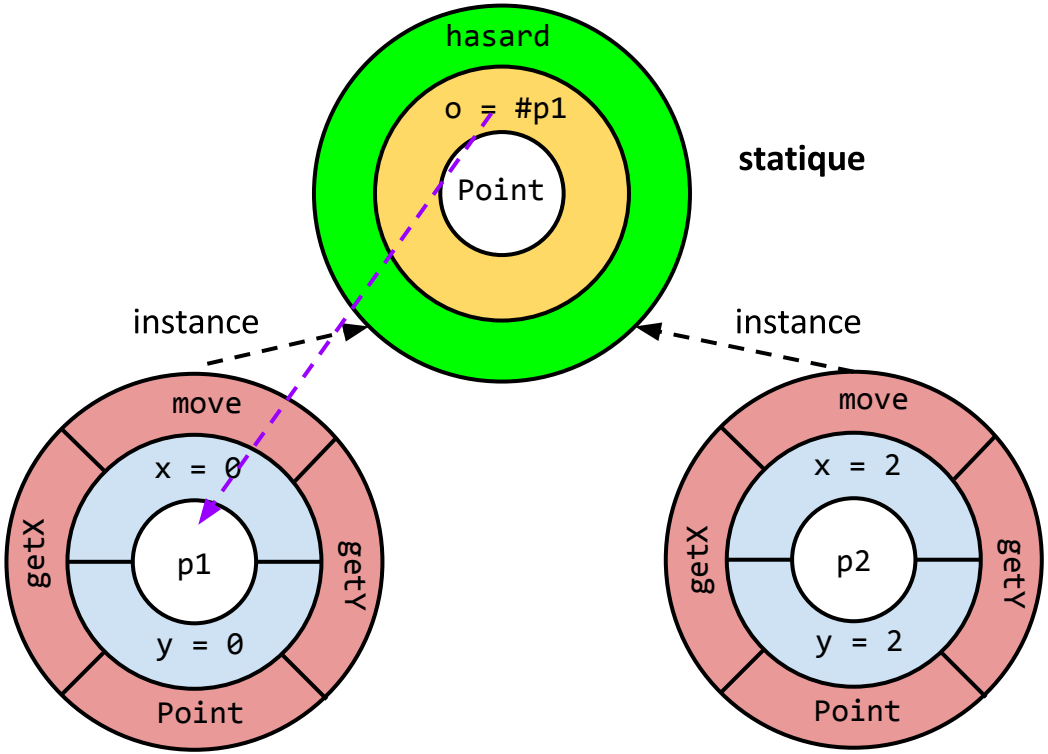
```
void bar(int i) {
    i = 0;
}

int i = 20;
bar(i); // i vaut 20
```

Attributs et méthodes statiques

```
class Point {
  static Point origine = new Point(0, 0);
  int x;
  int y;

  Point(int x, int y) {...}
  void move(int x, int y) {...}
  static Point auHasard(int maxX, int maxY) {
    int x = (int) Math.floor(Math.random()*maxX);
    int y = (int) Math.floor(Math.random()*maxY);
    return new Point(x,y);
  }
}
```

```
Point p1 = new Point(2, 2);
// on ne peut appeler move que sur un objet
p1.move(3, 3);

// on peut accéder à origine ou hasard()
// directement depuis la classe!
Point p2 = Point.origine;
Point p3 = Point.auHasard(15,10);
```

Classes et objets



Point d’entrée

```
class Main {

  public static void main(String[] args) {
    Point p = new Point(0, 0);
    p.move(2, 2);
  }

}
```

Manipulation

Rendez-vous sur :
http://www.labri.fr/perso/falleri/perso/ens/pg220/

Pour compiler votre fichier source java, rendez-vous dans le bon répertoire et lancez la commande suivante dans votre terminal :

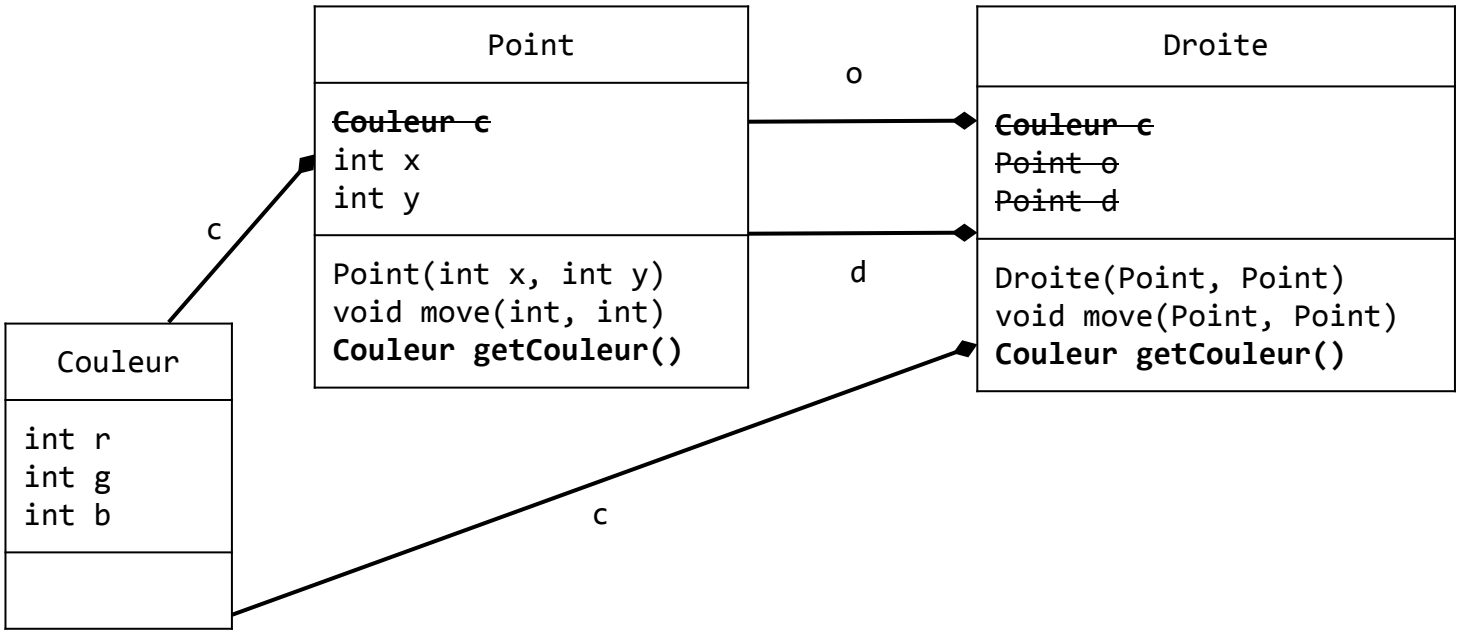
```
javac Main.java
```

Un fichier `Main.class` aura été généré, pour l'exécuter utilisez la commande :

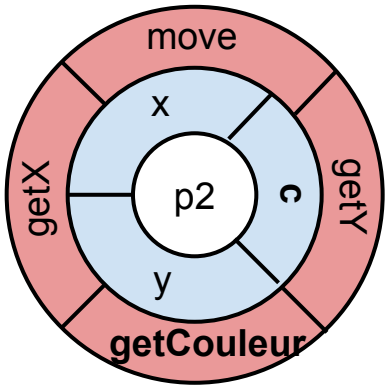
```
java Main
```

Séance 2

Héritage


Point
~~Couleur c~~
int x
int y
Point(int x, int y)
void move(int, int)
Couleur getCouleur()
Droite
~~Couleur c~~
~~Point o~~
~~Point d~~
Droite(Point, Point)
void move(Point, Point)
Couleur getCouleur()
Couleur
int r
int g
int b
o
d
c
c

Héritage



Couleur
int r int g int b

c

ElementRepere
~~Couleur c~~
Couleur getCouleur()

super classe

héritage (est un) →

extends

Point
int x int y
Point(int, int) void move(int, int)

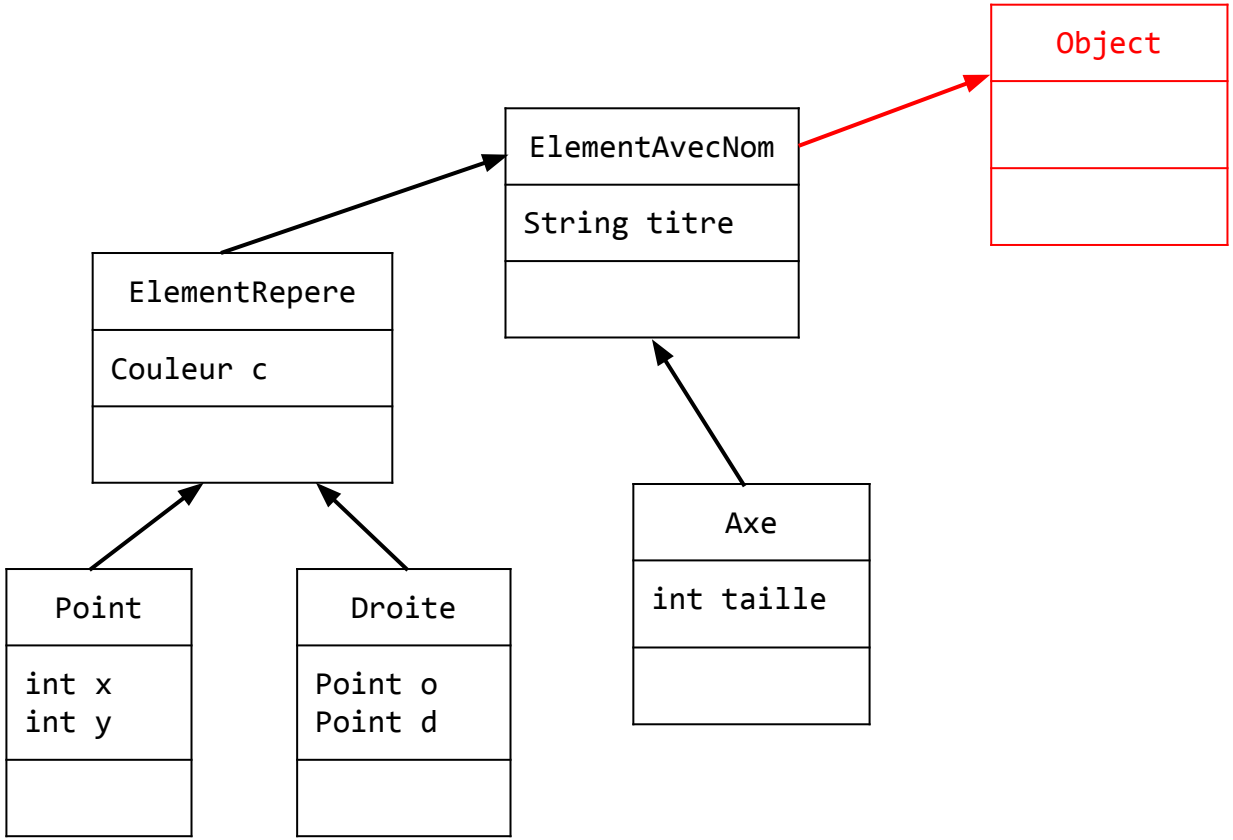
o

d

Droite
~~Point o~~ ~~Point d~~
Droite(Point, Point) void move(Point, Point)

sous classes

Hiérarchie de classes

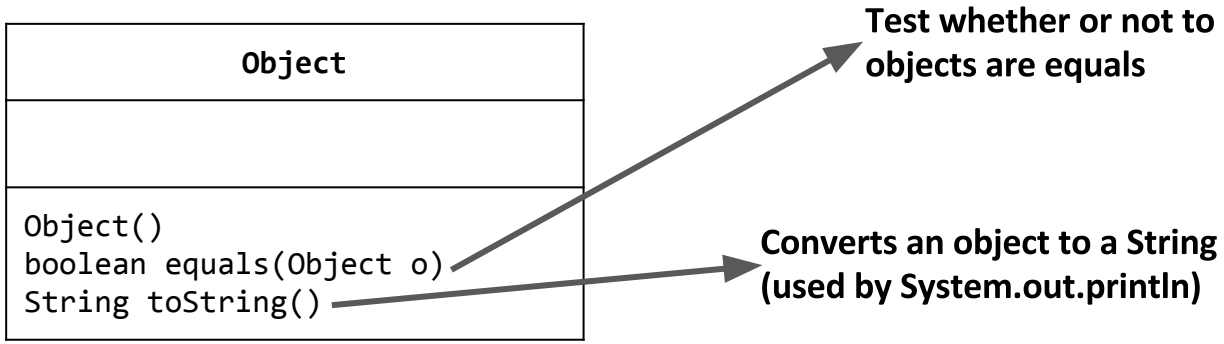


Extends

```
class ElementAvecCouleur {
  Couleur c;

  ElementAvecCouleur(Couleur c) {
    this.c = c;
  }

}
```

```
class Point extends ElementAvecCouleur {
  int x;
  int y;

  Point(int x, int y, Couleur c) {
    this.c = c;
    this.x = x;
    this.y = y;
  }

}
```

La classe Object



Redéfinition de méthodes

```
class Object {

  ...
  String toString() {...}

}
```

Utilisée par
System.out.println :
Point@e12ffc2

```
class ElementAvecCouleur {
  Couleur c;

  ElementAvecCouleur(Couleur c) {
    this.c = c;
  }

  String toString() {
    return "De couleur " + c;
  }
}
```

```
class Point extends
ElementAvecCouleur {
  int x;
  int y;

  Point(int x, int y, Couleur c) {
    super(c);
    this.x = x;
    this.y = y;
  }

  String toString() {
    return "Un point. " +
super.toString();
  }
}
```

Polymorphisme

Upcast : Je veux un ElementRepere, j'ai un Point

```
ElementRepere e = new Point(0,0);
```

Downcast : Je veux un Point, j'ai un ElementRepere

```
ElementRepere e = new Point(0,0);
Point p1 = (Point) e;
Droite d1 = (Droite) e;
```

Test de sous-typage

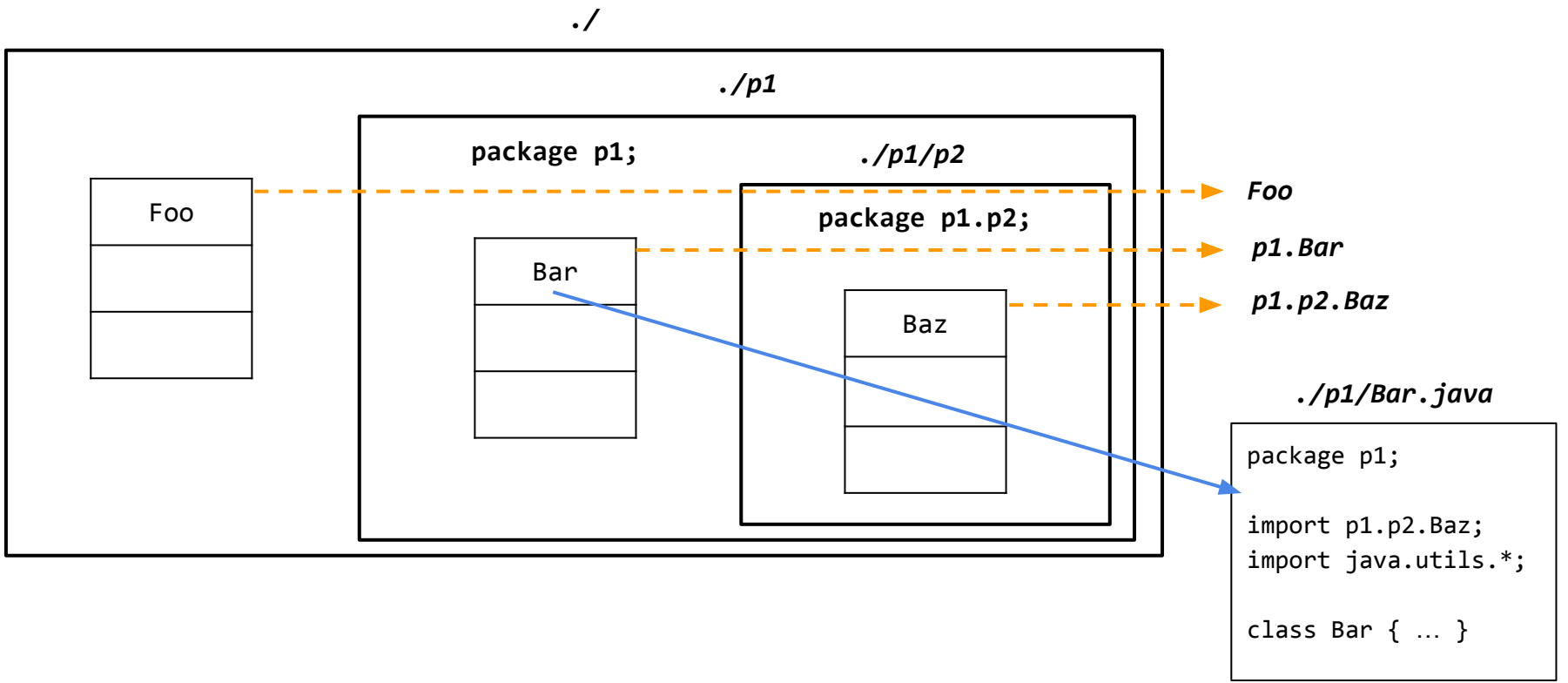
```
ElementRepere e = new Point(0,0);

System.out.println(e instanceof ElementRepere); // true
System.out.println(o instanceof Point); // true : un même objet peut être vu comme ayant différents types
System.out.println(o instanceof Droite);  // false

if (e instanceof Point) {
      Point p = (Point) e;
      p.move(1,1);
}
```

Liaison tardive

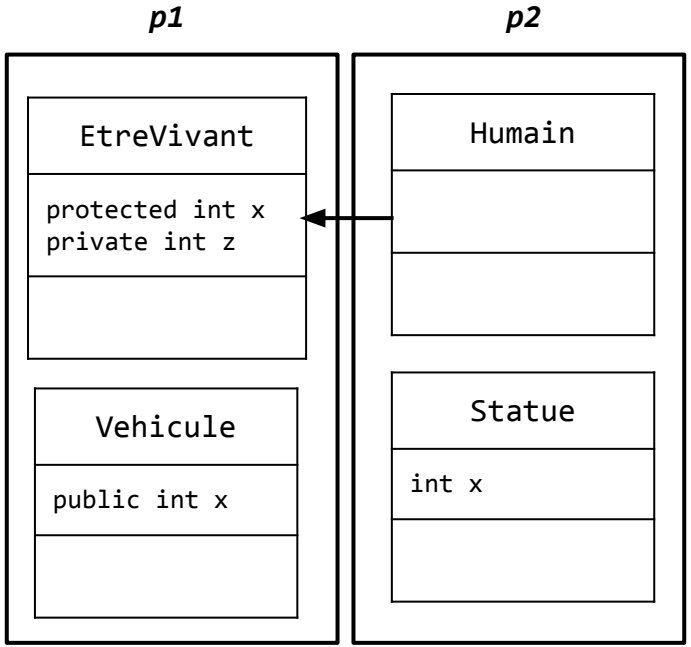
```
class Main {
  public static void main(String[] args) {
    ElementRepere e1 = new ElementRepere();
    System.out.println(e1.toString()); // De couleur noir.
    ElementRepere e2 = new Point(0,0);
    System.out.println(e2.toString()); // Un point. De couleur noir.
  }
}
```

Séance 3

Packages et noms qualifiés



Visibilités



	class	subclass	package	project
private	yes	no	no	no
protected	yes	yes	yes	no
public	yes	yes	yes	yes
package	yes	no	yes	no

Les erreurs en C

```
void bar() {
  ...
  int err = foo();
  if (err != 0)
    // Code en cas d'erreur
  else
    // Code normal
}
```

```
int foo()  {
  ...
  if (erreur)
    return 1;
  else
    return 0;
}
```

Les exceptions

```
public class Point {
  private int x;
  private int y;

  public Point(int x, int y) throws
PointInvalide {
    if ((x<0) || (y<0))
      throw new PointInvalide(this);
    this.x = x;
    this.y = y;
  }
}
```

```
public class PointInvalide extends Exception
{
  private Point p;

  PointInvalide(Point p) {
    super("Point Invalide :" + p);
    this.p = p;
  }

  boolean isXValid() {
    return p.getX() >= 0;
  }

  ...
}
```

Gestion d'une exception

Sur place

```
public class Droite {
  public Point origine;
  public Point destination;

  public Droite() {
    try {
      this.origine = new Point(-1, -1);
      this.destination = new Point(-1, -1);
    } catch (PointInvalide e) {
      System.out.println("erreur");
    }
  }

}
```

Renvoyer

```
public class Droite {
  public Point origine;
  public Point destination;

  public Droite() throws PointInvalide {
    this.origine = new Point(-1, -1);
    this.destination = new Point(-1, -1);
  }

}
```

Choix du gestionnaire d'exception

```
try {
   new Point(-1,-1);
} catch (Exception e) {...}
} catch (PointInvalide e) {...} // jamais exécuté!
```

```
try {
   new Point(-1,-1);
} catch (PointInvalide e) { // exécuté pour les erreurs de points }
} catch (Exception e) { // exécuté pour les autres erreurs }
```

Le bloc `finally`

```
try {
   …
} catch (Exception e) {
   …
} finally {
   System.out.println(“Instruction toujours exécutée.”);
}
```

Exceptions à traitement optionnel

```
public class PointInvalide extends
RuntimeException {

}
```

- `NullPointerException`
  - Point p = null
  - p.move(0,0);
- `ArithmeticException`
- `IndexOutOfBoundException`
- `ClassCastException`
- `...`

Séance 4

Classes et méthodes abstraites

```
public abstract class ElementRepere extends
ElementAvecNom {

  public ElementRepere(String nom) {
    super(nom);
  }

  public abstract void dessiner();

}

public class Droite extends ElementRepere {
  ...

  // Redéfinition obligatoire
  public void dessiner(){
    System.out.println("Je suis une droite");
  }

}
```

```
public class Main {
  public static void main(String[] args) {
    ElementRepere element;

    //ElementRepere est une classe abstraite
    //Elle ne peut pas être instanciée
    element = new ElementRepere("d1");

    //Droite est une classe concrète
    //Elle peut donc être instanciée
    element = new Droite("d1");
    element.dessiner();
  }
}
```

Attributs, méthodes et classes finales

```
public final class Droite extends ElementRepere {
// On ne peut pas hériter de droite

  // On ne peut affecter qu'une fois o
  public final Point o;

  // On ne peut pas redéfinir dessiner()
  public final void dessiner(){
    System.out.println("Je suis une droite");
  }

}
```

~~`public class SuperDroite extends Droite {}`~~

Interfaces

```
public interface EnsembleElementRepere {
  // Les méthodes d'une interface sont publiques
  void ajoute(ElementRepere element);
  int taille();
  ElementRepere recupere(int index);
}

public class EnsembleElementRepereTableau
implements EnsembleElementRepere {

  void add(ElementRepere element) { … }
  int taille() { … }
  ElementRepere recupere(int index) { … }
}

public class EnsembleElementRepereChaine
implements EnsembleElementRepere {

  void add(ElementRepere element) { … }
  int taille() { … }
  ElementRepere recupere(int index) { … }
}
```

```
public class Main {

  public static void main(String[] args) {
    // Version tableaux
    EnsembleElementRepere ens =
        new EnsembleElementRepereTableau();
    ens.add(new Point());
    for (int i = 0; i < ens.taille(); i++)
      System.out.println(ens.recupere(i));

    // Version liste chaînée
    ens = new EnsembleElementRepereChaine();
    ens.add(new Point());
    for (int i = 0; i < ens.taille(); i++)
      System.out.println(ens.recupere(i));
  }

}
```

Types génériques

```
public interface EnsembleElementRepere {

  void ajoute(ElementRepere element);
  int taille();
  ElementRepere recupere(int index);

}

public interface EnsembleElement<E> {

  void ajoute(E element);
  int taille();
  E recupere(int index);

}
```

```
public class Main {

  public static void main(String[] args) {
    EnsembleElement<ElementRepere> ens1 =
        new EnsembleElementTableau<>();
    ens1.ajoute("toto");
    ens1.ajoute(new Point());

    EnsembleElement<String> ens2 =
        new EnsembleElementTableau<>();
    ens2.ajoute("toto");
    ens2.ajoute(new Point());
  }

}
```

Types génériques bornés

```
public interface EnsembleElementRepere<E extends
ElementRepere> {

  void ajoute(E element);
  int taille();
  E recupere(int index);
  void dessiner();

}
```

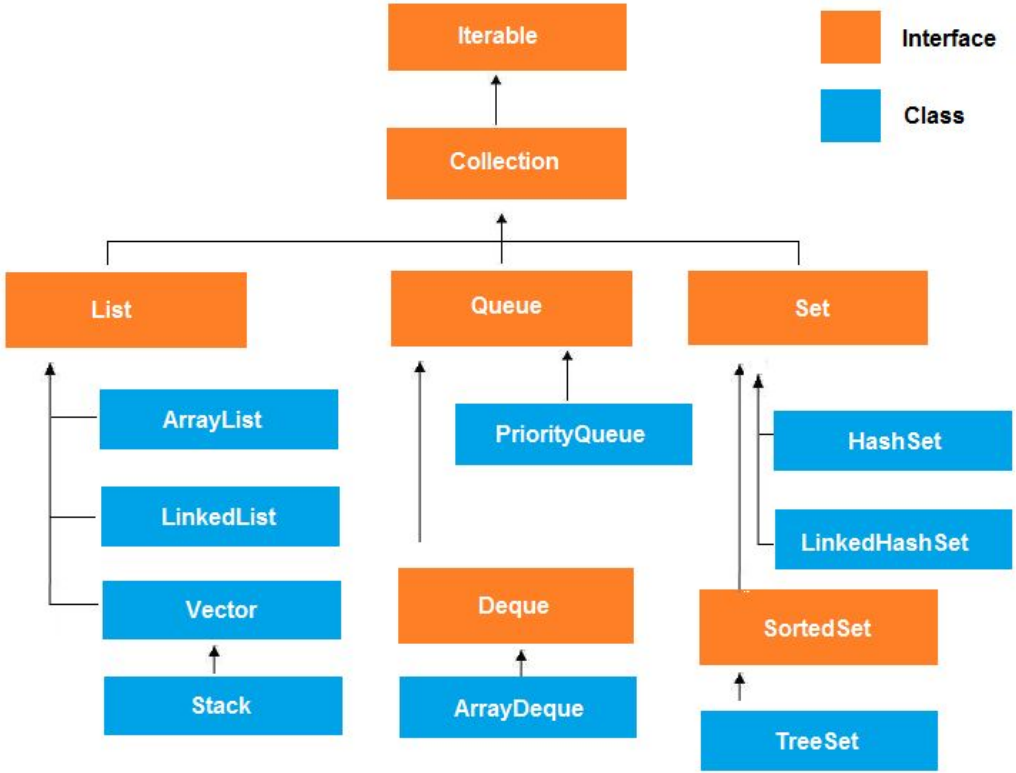
```
public class Main {
  public static void main(String[] args) {
    EnsembleElementRepere<ElementRepere> ens1 =
        new EnsembleElementTableau<>();

    EnsembleElementRepere<Point> ens2 =
        new EnsembleElementTableau<>();

    // String n'hérite pas de ElementRepere
    EnsembleElementRepere<String> ens3 =
        new EnsembleElementTableau<>();
  }
}
```

Types génériques - Wildcard (Joker)

```java
public interface EnsembleElement<E> {

  void ajoute(E element);
  int taille();
  E recupere(int index);

}
```

```java
public class Main {
  public static void main(String[] args) {

    EnsembleElement<?> ens;

    ens = new
      EnsembleElementRepereTableau<ElementRepere>();
    ens = new
      EnsembleElementRepereTableau<String>();

    ens.ajoute("test");
    // Ne peut pas marcher car on ne connaît pas le
    // type

    Object o = ens.recupere(0);
    // Fonctionne car Object est supertype de tous
    // les types
  }
}
```

Les collections



Auteurs

- Jean-Rémy Falleri
- Cédric Teyton
- Alan Charpentier
- Mohamed Ameziane Oumaziz