Programmation orientée objets en Java

Jean-Rémy Falleri

Le module PG220

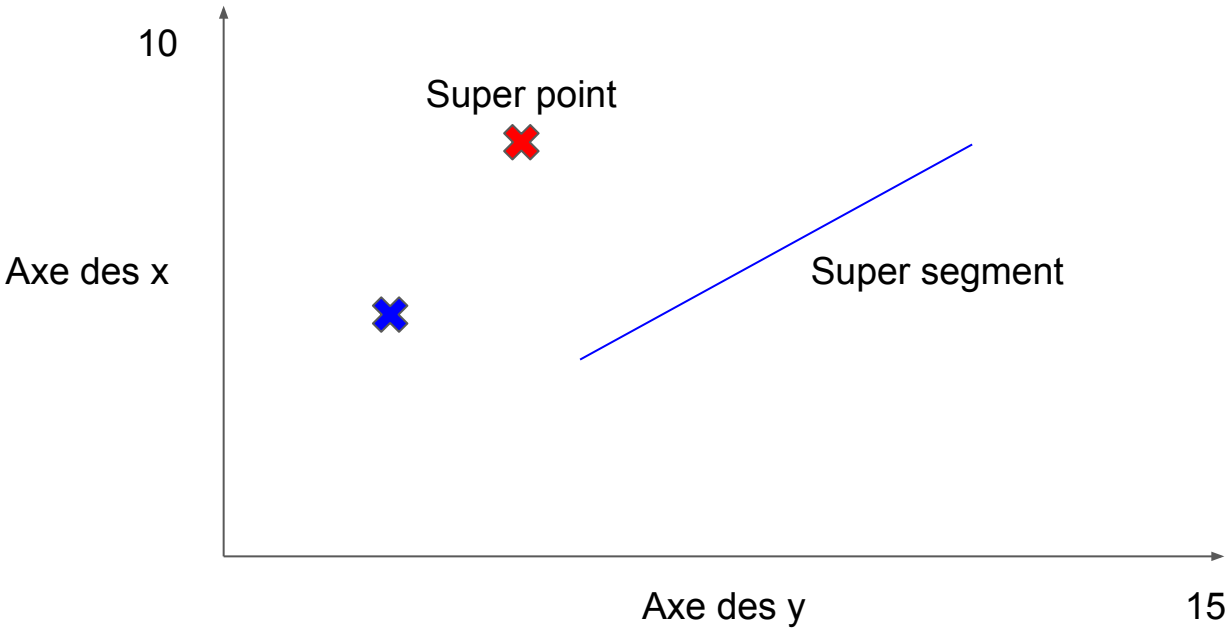
- Pré-requis :
  - Programmation C
  - Algorithmique et structures de données
- On apprend
  - Le paradigme objet
  - Certaines spécificités de la couche objet Java
- On apprend pas
  - Les GUI en Java
  - Les fonctionnalités avancées de Java
  - Toutes les spécificités de tous les paradigmes objets

Déroulé

- 5 séances de cours en EI
  - Manipulation en fin de séances
  - A finir à la maison
- 2 TPs
  - A finir à la maison
  - Tests et corrections disponibles
- Un projet
  - Séances de suivi
  - Binômes
- Note finale : `0.4 * QCM + 0.6 * PROJET`

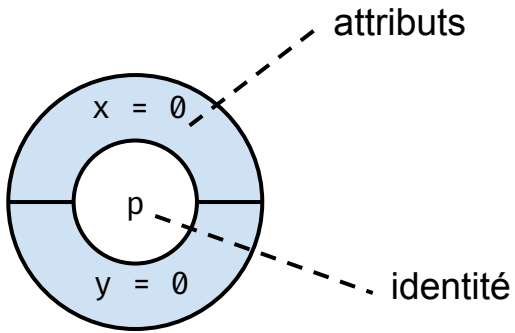
Séance 1

Un canevas

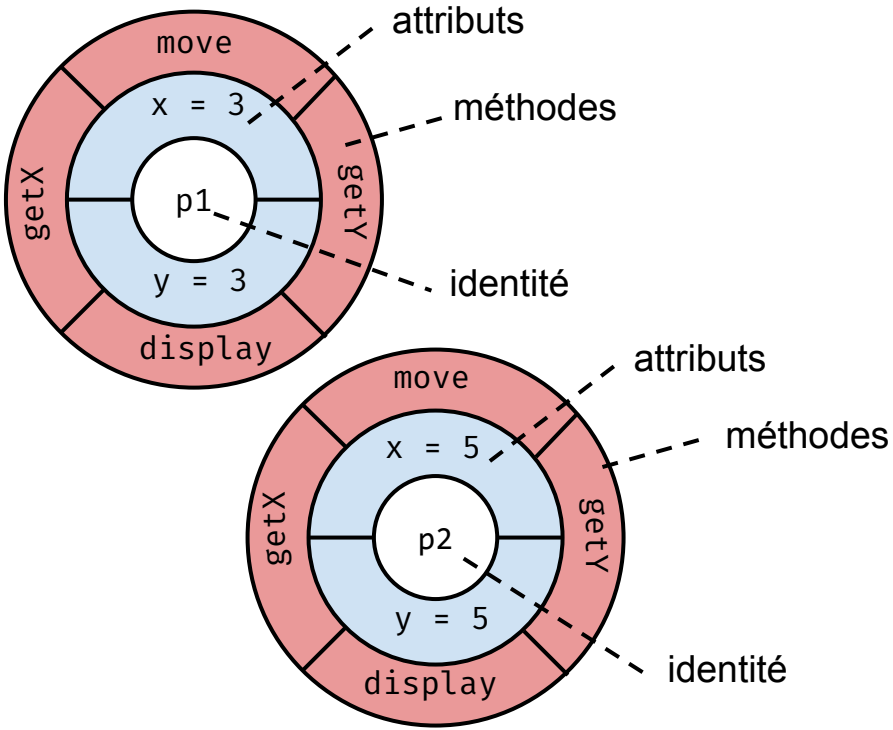


Exemple de point en C

```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 (%d,%d).\n", p->x, p->y);
}

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



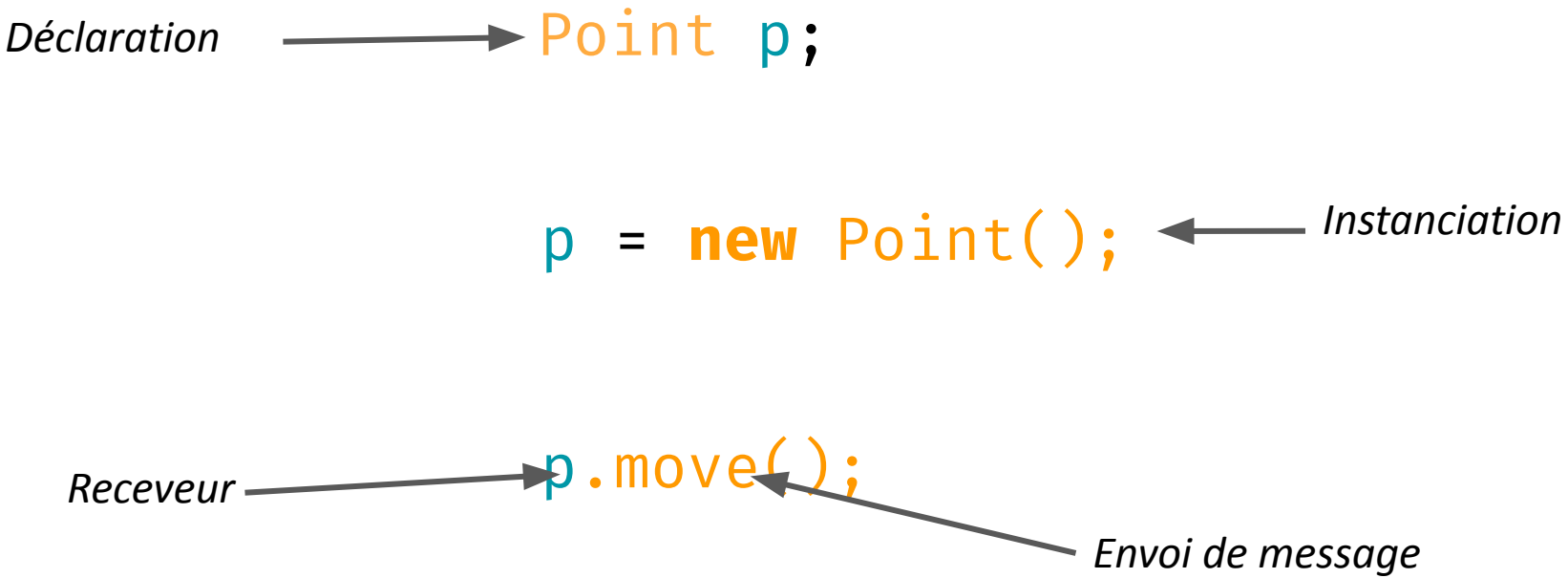
Les objets




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

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

Encapsulation des données

Un peu de vocabulaire



Les classes

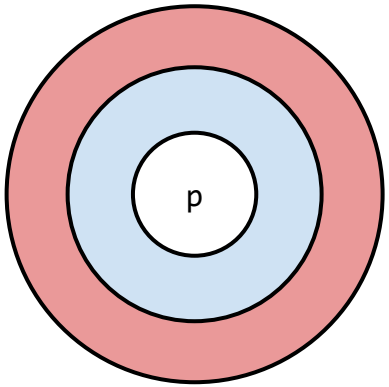
- Unité de programmation en Java (on ne programme pas directement des objets)
  - Un fichier Java → une classe Java
- Déterminent les caractéristiques des objets
  - Liste des attributs
  - Liste des constructeurs → *instanciation*
  - Liste des méthodes → *envoi de message*
- Servent de moules à objets
  - Via les constructeurs et l'utilisation de **`new`** `new Point();`
- Introduisent de nouveaux types pour les variables
  - `Point p;`
- Les objets sont *instances* d'une classe

La classe point

Point.java

```
class Point { // classe
  Point() { // constructeur vide, optionnel
  }
}
```

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

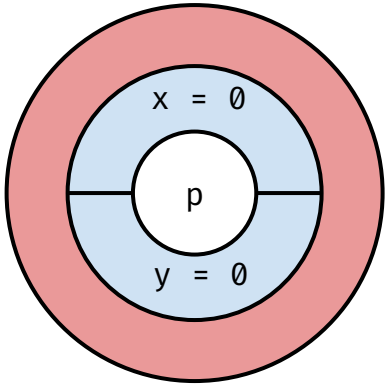


Attributs

Point.java

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

  Point() { // constructeur vide
    this.x = 0;
    this.y = 0;
  }
}
```

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

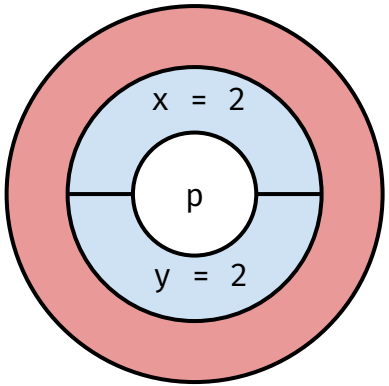


Surcharge de constructeurs

Point.java

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

  Point() { // constructeur vide
    this.x = 0;
    this.y = 0;
                                              }

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

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

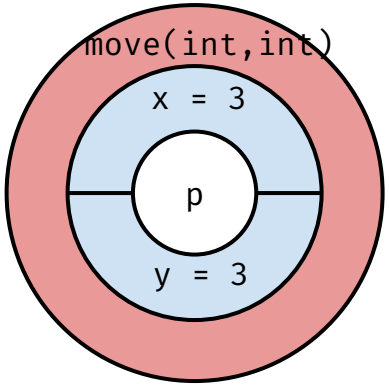


Méthodes

Point.java

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

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

  Point() { // surcharge constructeur
    x = 0;
    y = 0;
  }

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

```
Point p = new Point(2,2);
p.move(3,3);
```

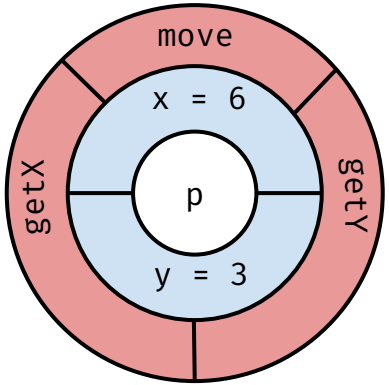


Getters et *setters*

Point.java

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

                                                    ...

            void       setX(int       x)       {
    this.x = x;
  }

  int getX() {
    return this.x;
  }
}
```

```
Point p = new Point(2,2);
p.move(3,3);
int x= p.getX();
p.setX(6);
```



La classe Segment

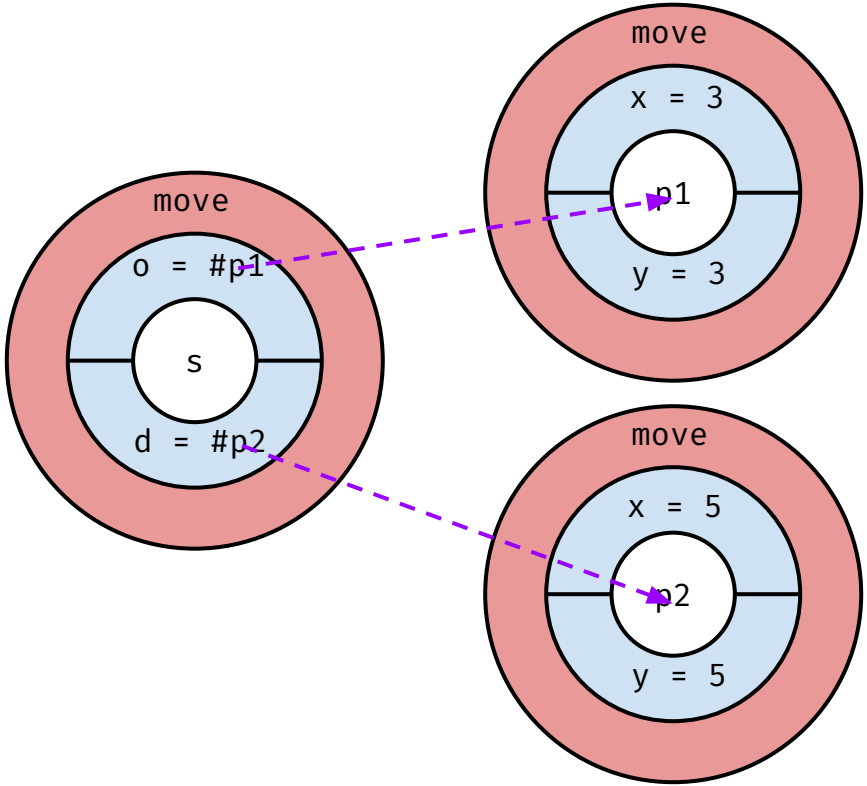
Point.java

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

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

  Point() { // surcharge constructeur
    x = 0;
    y = 0;
  }

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

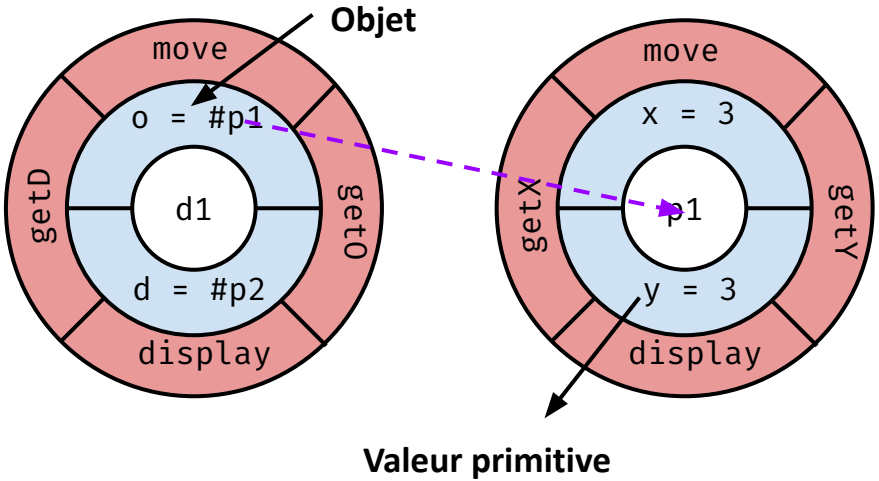
Segment.java

```
class Segment {
  Point o;
  Point d;

  Segment(Point o, Point d) {
    this.o = o;
    this.d = d;
  }

  void move(int x1, int y1, int x2, int y2) {
    this.o.move(x1,y1);
    this.d.move(x2,y2);
  }
}
```

La délégation



Objets et valeurs primitives




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

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

Passage de paramètres

Les objets sont passés par **référence**

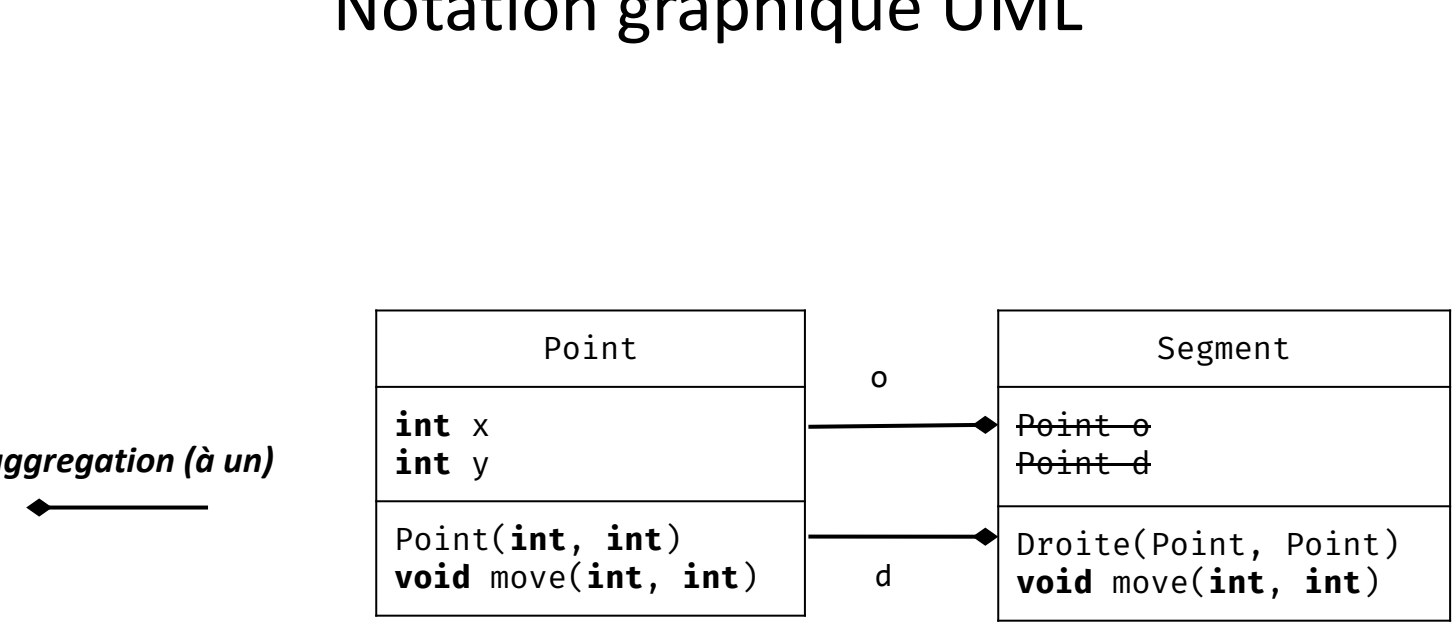
```
void foo(Point p) {
   p.move(0, 0);
}

Point p = new Point(10, 10);
foo(p); // après l'appel, p est à (0, 0)
```

Les valeurs primitives sont passées par **copie**

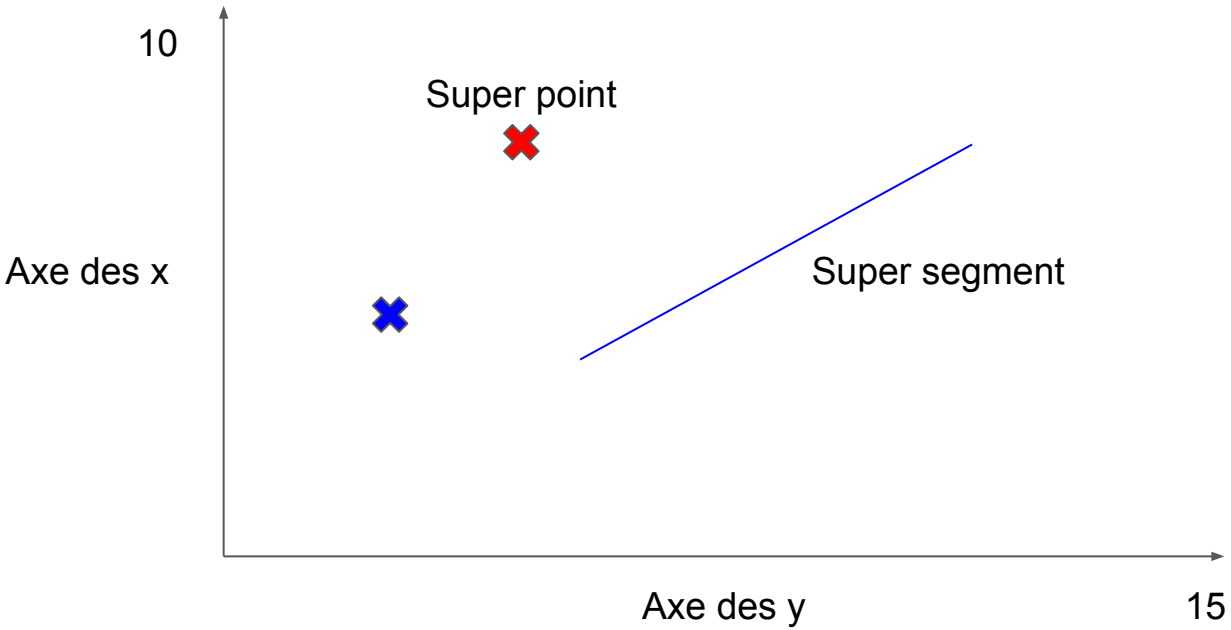
```
void bar(int i) {
   i = 0;
}

int i = 20;
bar(i); // après l'appel, i vaut 20
```

Notation graphique UML

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

Segment

- 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)

EnsembleDeSegments

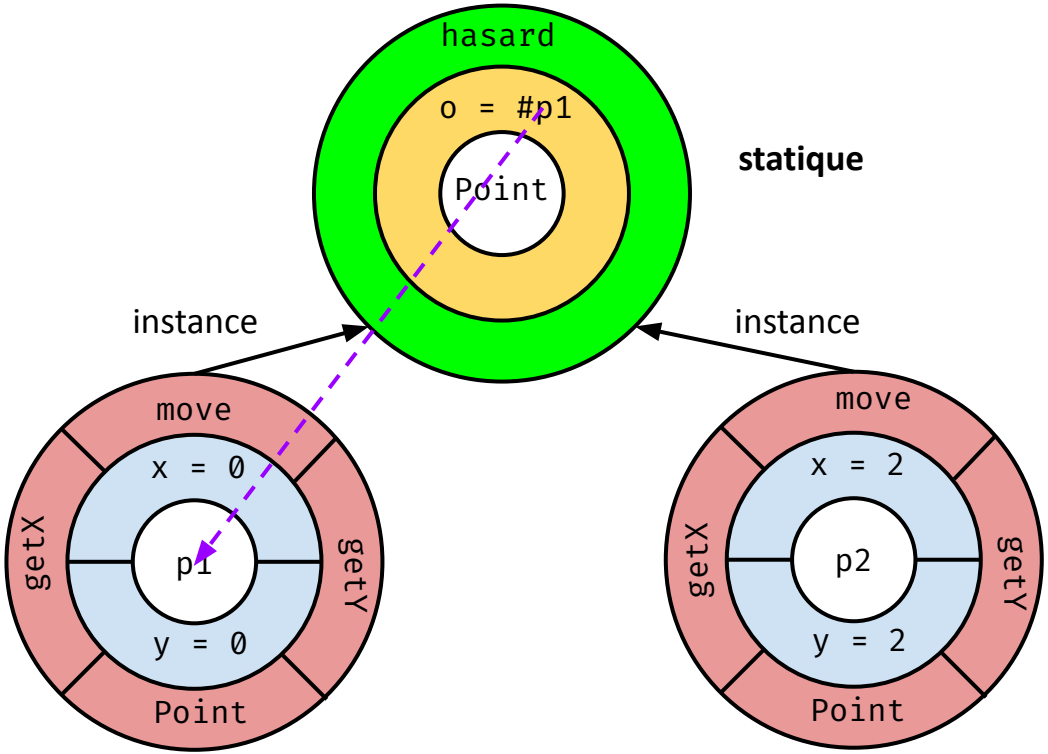
- Attributs
  - segments: Segment[]
- Méthodes
  - ajouter droite(Droite d)
  - enlever droite(Droite d)

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 xMax, int yMax) {
    int x = (int)
Math.floor(Math.random()*xMax);
    int y = (int)
Math.floor(Math.random()*yMax);
    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 aux membres statiques
// directement depuis la classe!
Point p2 = Point.origine;
Point p3 = Point.origine;
// p2 et p3 référencent le même objet

Point p4 = Point.auHasard(15,10);

// on peut aussi appeler une méthode statique
// depuis une instance (idem attribut)
Point p5 = p2.auHasard();
```

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

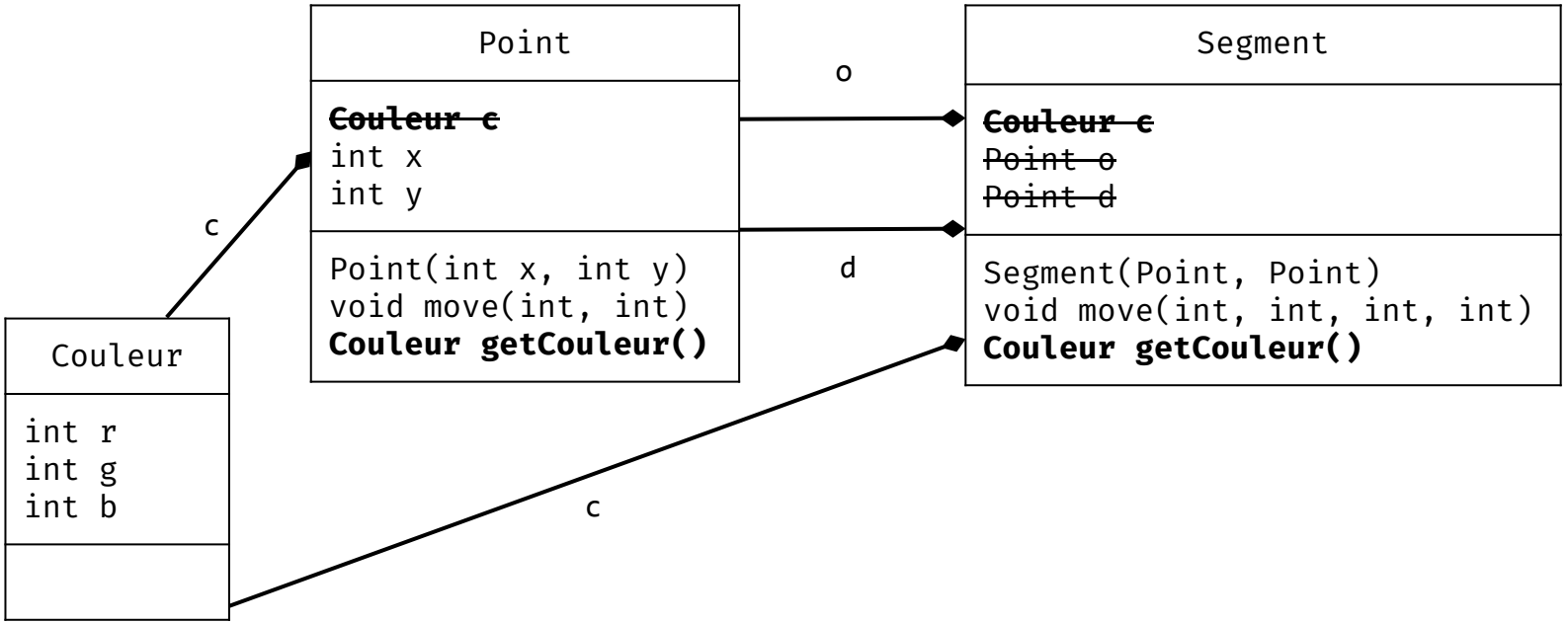
Rendez-vous sur :
http://www.labri.fr/perso/falleri/perso/ens/pg220/

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

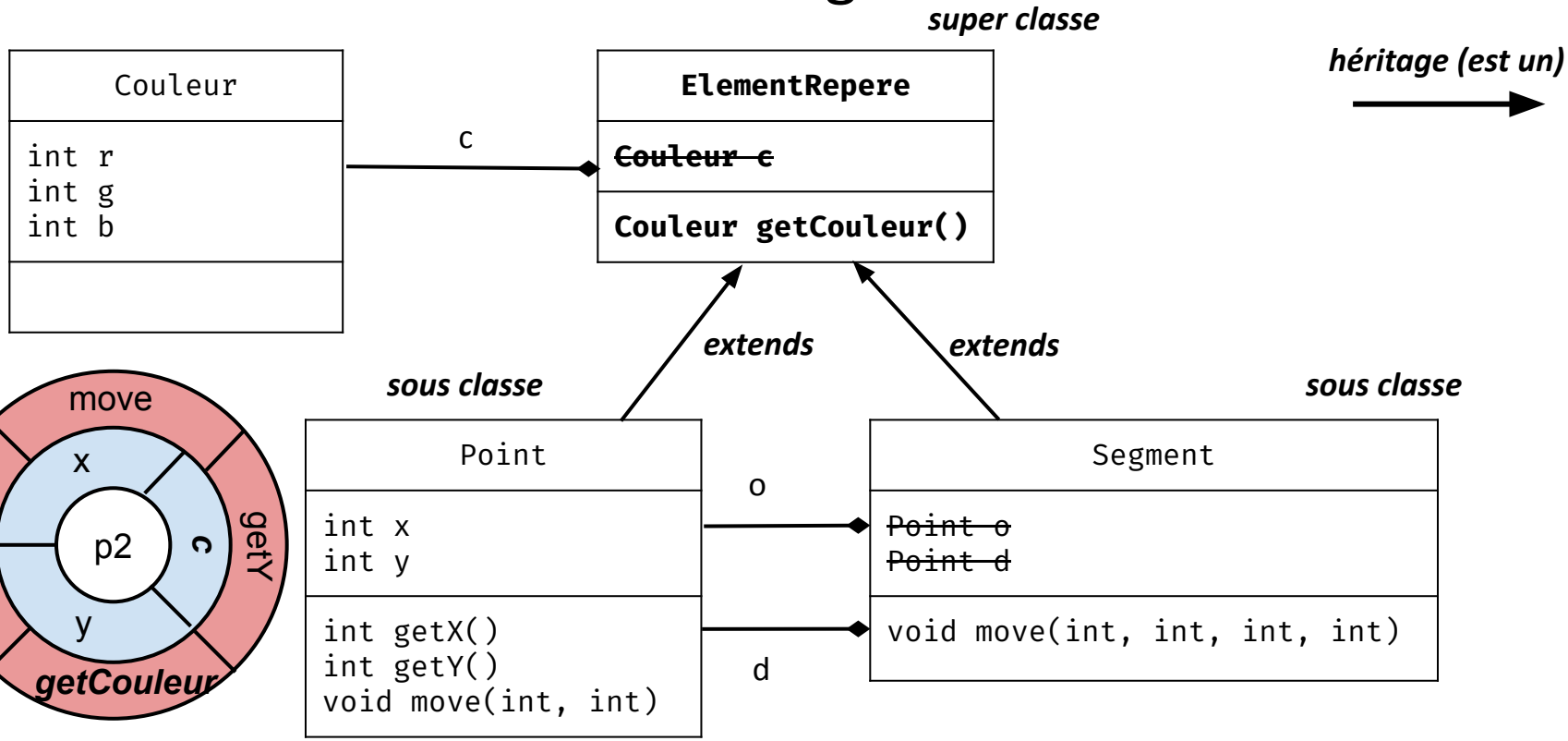
Un fichier `Main.class` aura été généré, pour l'exécuter utilisez la commande :
`java Main`

Séance 2

Héritage

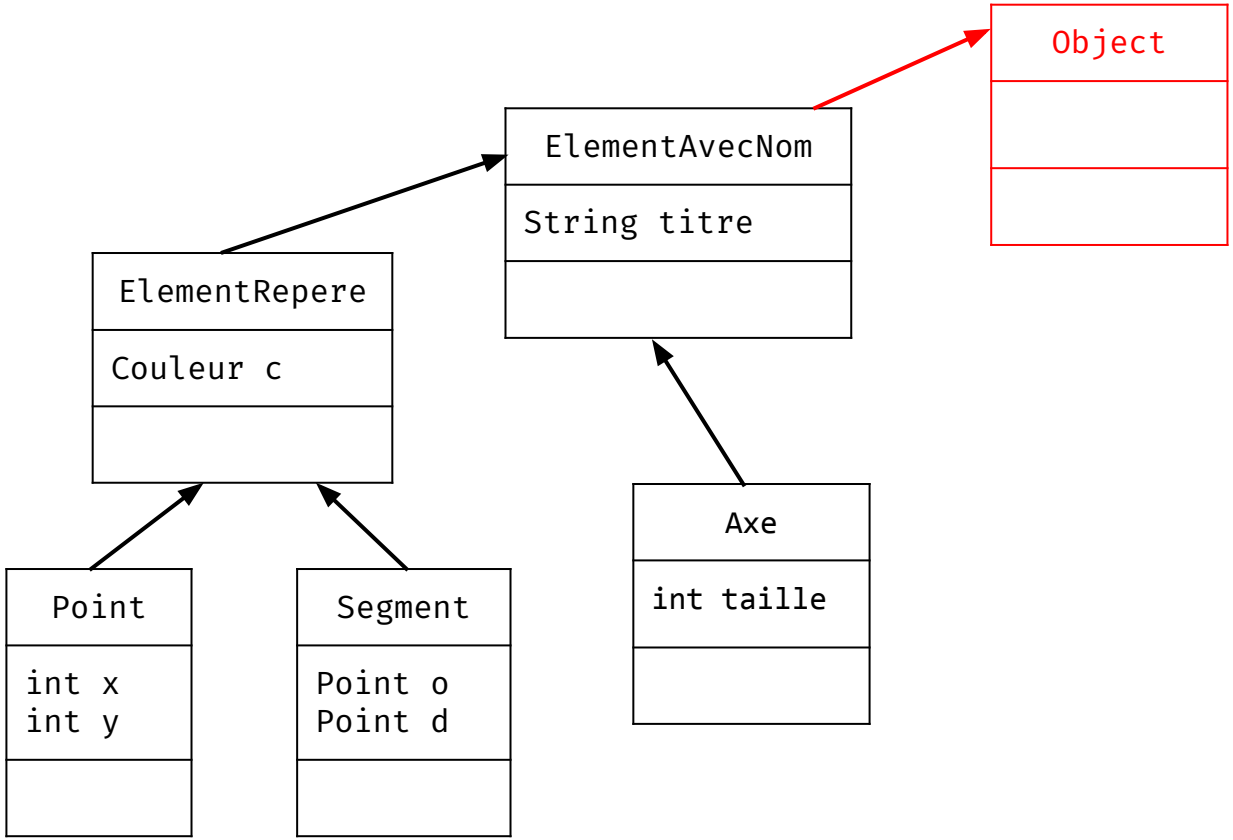

Point
~~Couleur c~~
int x
int y
Point(int x, int y)
void move(int, int)
Couleur getCouleur()
Segment
~~Couleur c~~
~~Point o~~
~~Point d~~
Segment(Point, Point)
void move(int, int, int, int)
Couleur getCouleur()
Couleur
int r
int g
int b
o
d
c
c

Héritage



On n'hérite pas des constructeurs!

Hiérarchie de classes

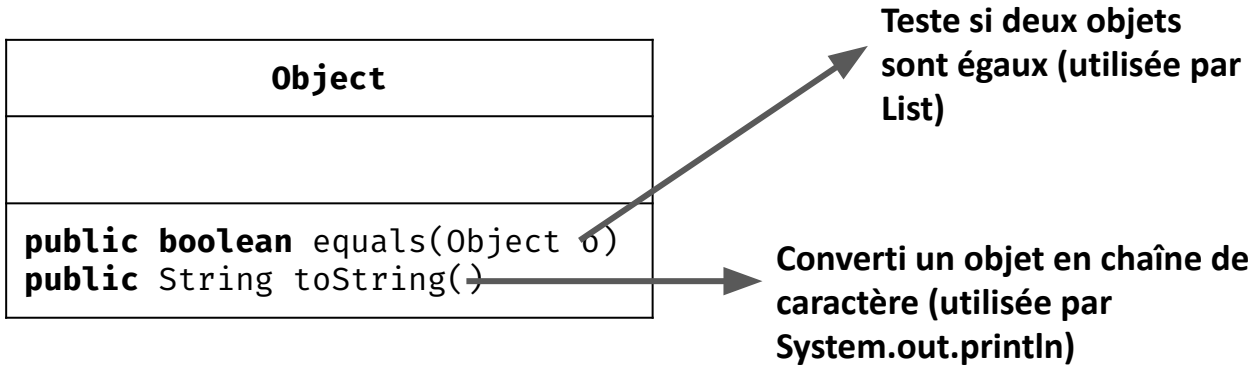


Héritage en Java

```
class ElementRepere {
  Couleur c;

  Couleur getCouleur() {
    return this.c;
  }
}
```

```
class Point extends ElementRepere {
  Couleur c;
  int x;
  int y;

  void move(int x, int y) {
    this.x = x;
    this.y = y;
  }

  Couleur getCouleur() {
    return this.c;
  }
}
```

La classe Object



Redéfinition de méthodes

```
class Object {

   public  String  toString()
{ … }

}
```

```
class ElementRepere {
  Couleur c;

  public String toString() {
    return “De couleur ” + c;
  }
}
```

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

  public String toString() {
    return “Point (” + x + “,” +
y + “) ” + super.toString();
  }
}
```

Un problème de types

```
void println(Object o} {
   ...
  o.toString();
   ...
}

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

Peut on passer le point à la méthode println?

Liaison tardive

```
class Main {
  public static void main(String[] args) {
    Object o = new Object();
    Object e = new ElementRepere();
    Object p = new Point(0,0);
    Object d = new Segment(new Point(0,0), new Point(10,10));

    System.out.println(o.toString()); // Object@12f3ac
    System.out.println(e.toString()); // De couleur (0,0,0).
    System.out.println(p.toString()); // Un point. De couleur (0,0,0).
    System.out.println(d.toString()); // Une droite. De couleur (0,0,0).
  }
}
```

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 Object); // true Point <= Object

System.out.println(e instanceof ElementRepere); // true Point <= ElementRepere

System.out.println(e instanceof Point); // true Point <= Point

System.out.println(e instanceof Droite); // false Droite <= Point

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

Le downcast est “sûr”

Héritage et constructeurs

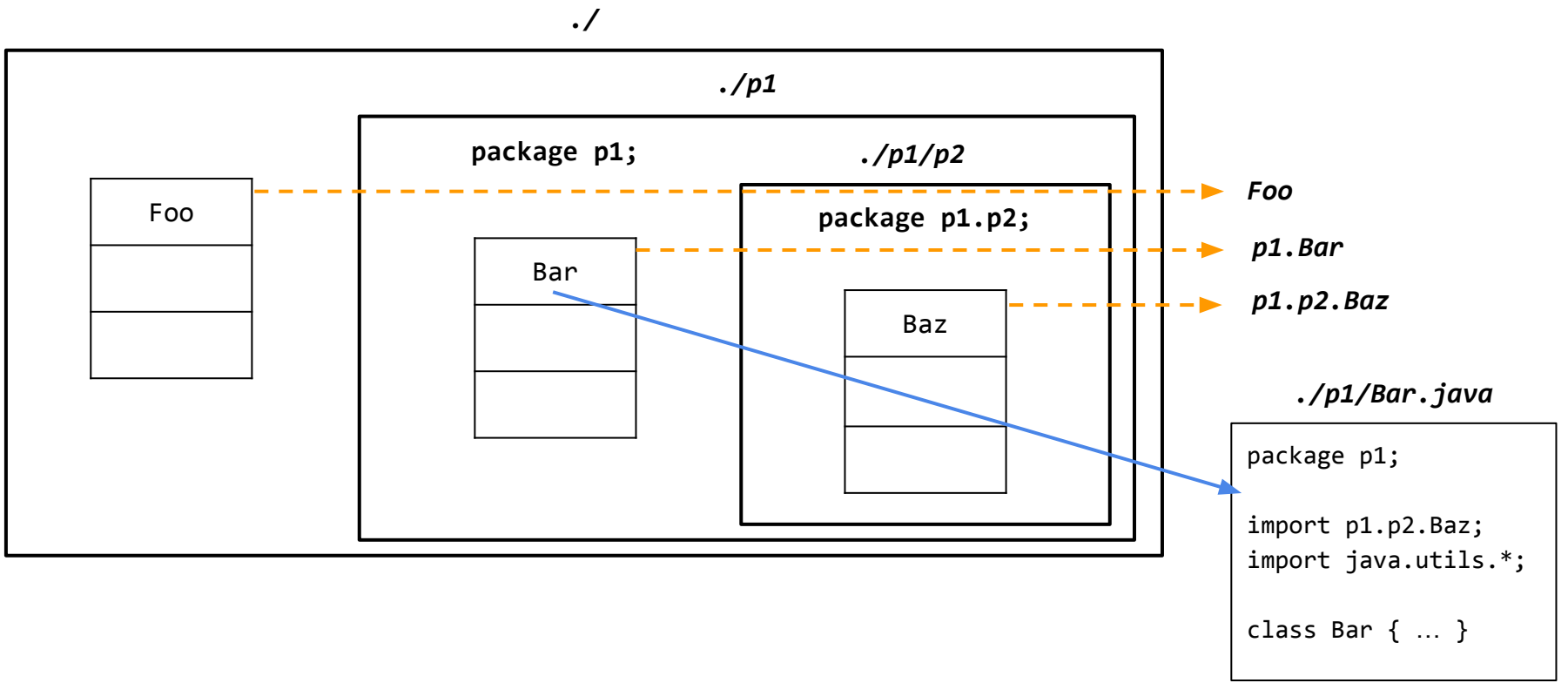
```
class ElementRepere {
  Couleur c;

  ElementRepere(Couleur c) {
    super();
    this.c = c;
  }
}
```

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

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

  Point() {
    this(0, 0, new Couleur(0, 0, 0));
  }
}
```

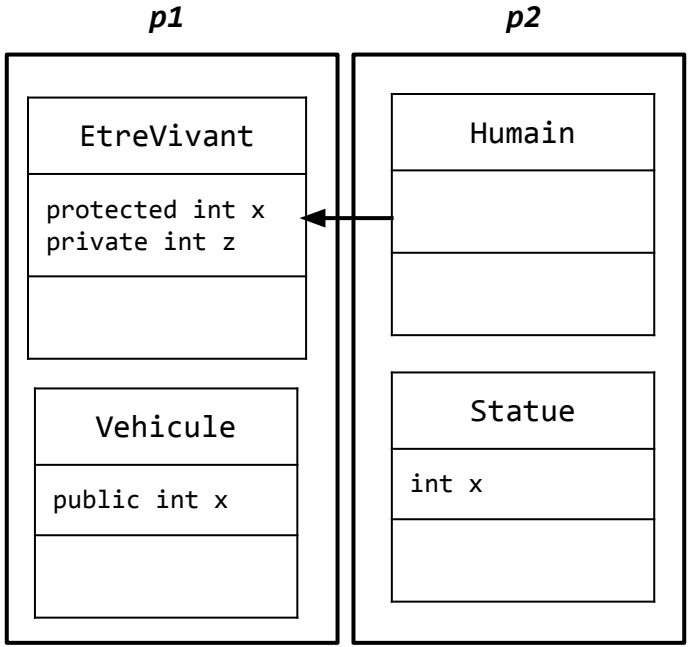
Appel à un super-constructeur obligatoire!

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
  - int i = 3/0;
- IndexOutOfBoundException
  - int[] tab = new int[3];
    tab[3] = 12;
- 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();

}
```

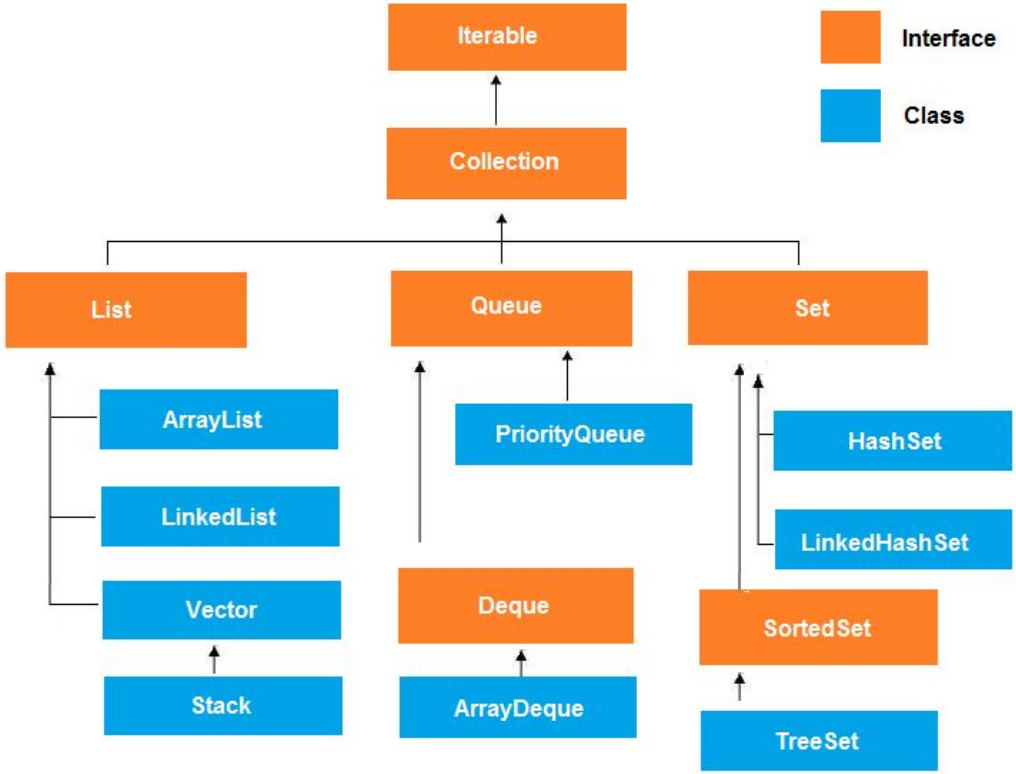
```
public class Main {
  public static void main(String[] args) {
    EnsembleElementRepere<ElementRepere> ens1 =
        new EnsembleElementTableau<>();

    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