L'héritage en POO

Classes et héritage

- Classe Rectangle pour manipuler des rectangles simples
- Si nous voulons manipuler des carrés, il faut à nouveau définir une classe Carre

```
public class Rectangle{

   // les attributs

   private double largeur, longueur;


   // les constructeurs

   public Rectangle(){

      largeur = longueur = 1.0;    }

   public Rectangle(double larg, double lon){

      largeur = larg;

      longueur = lon; }

   // les méthodes

   @Override

   public String toString(){

      return "rectangle de longueur " + longueur + " et de largeur " + largeur;    }


   public double perimetre(){

      return 2.0*(largeur + longueur);    }

   public double surface(){

      return largeur*longueur;    }


   public void changerLongueur(double lg){
```

La classe Carre

```
public Class Carre {

    // attribut
    private double côté;

    // constructeur
    Carre (double c){
côté = c;
    }

    // les méthodes
    public double périmètre(){
return 4.0*côté;
    }

    public double surface(){
return côté*côté;
    }
}
```

Carré
- côté: réel
+ Carré(c) + périmètre() : réel + surface() : réel

Remarques

- La classe ressemble beaucoup à la classe Rectangle
- Un carré est un rectangle particulier
- Il est légitime de ne pas réécrire totalement la classe Carré mais de réutiliser une partie de la classe Rectangle
- L'héritage est une relation entre deux classes qui permet à une classe de réutiliser les caractéristiques d'une autre

```
public class Carre extends Rectangle {

    public Carre (double c){
super(c,c);
    }
}
```

Principe de l'héritage

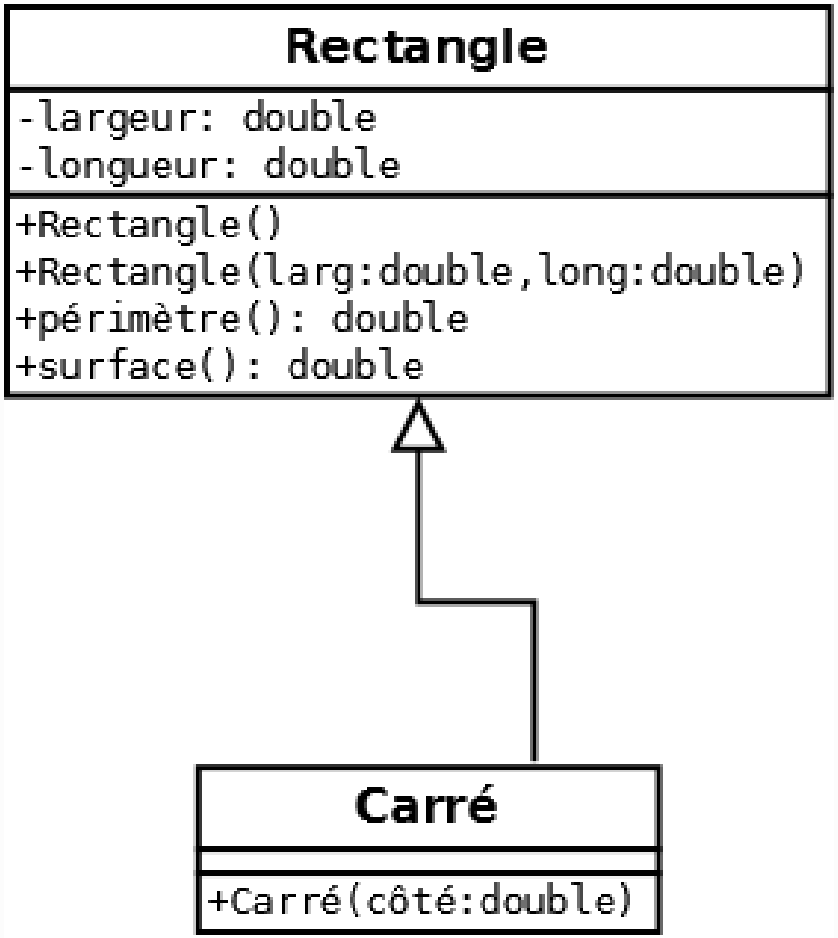
- Une classe (en Java) peut hériter d'une seule autre classe en utilisant le mot-clef extends
- La classe dérivée peut :
  - accéder aux méthodes et attributs de la classe parent (ou mère) (en respectant les règles de visibilité)
  - ajouter des nouveaux attributs à la classe mère
  - redéfinir des méthodes de la classe mère
  - → la classe dérivée étend la classe mère
- L'héritage est un concept **fondamental** de la POO, il permet de réutiliser du code déjà écrit

Représentation UML

- L'héritage est matérialisé par une flèche creuse de la classe dérivée vers la classe mère
- On ne reprend dans la classe dérivée (ou fille) que les attributs et méthodes ajoutés ou modifiés (les attributs et méthodes hérités sont implicites)

Carre c1 = new Carre(5) ;

Sop(c1.surface()) ;



Visibilité

UML	visibilité	mot-clef	classe	package	sous-classe	toute classe
+	publique	public	OUI	OUI	OUI	OUI
#	protégée	protected	OUI	OUI	OUI	NON
~	package		OUI	OUI	NON	NON
-	privée	private	OUI	NON	NON	NON

- En général, les classes dérivées sont dans le même package (répertoire) que leur parent.

La redéfinition des méthodes

- A l'intérieur de la sous-classe, 3 possibilités :
  - La méthode est héritée (elle n'est pas réécrite), elle est disponible
  - La méthode est redéfinie (elle a même signature)
    - Même nom, mêmes arguments
  - Une nouvelle méthode est définie
- On pourrait dans la classe Carre redéfinir la méthode toString() qui affiche que c'est un rectangle
- On peut utiliser le mot-clef @Override qui indique une redéfinition (optionnel en Java mais conseillé)

Redéfinition

- Redéfinir une méthode d'affichage dans la classe Carré

Rectangle r = new Rectangle(2,4) ; S.o.p(r) ; → rectangle de longueur 2 et largeur 4

Carré c = new Carré(5) ; → S.o.p(c) ; → rectangle de longueur 5 et largeur 5

```
public class Carré {
…
@Override
public String toString(){
   return "carré de côté " + largeur ;
}
```

Rectangle r = new Rectangle(2,4) ; → méthode d'affichage de la classe Rectangle

Carré c = new Carré(5) ; → méthode d'affichage de la classe Carré

Le mot-clef super

- Si une classe redéfinit une méthode de la classe mère, elle masque son implémentation dans la classe parent
- L'implémentation dans la classe mère est toutefois accessible par le mot-clef super :
  - super.méthode(arg1, …, argn)
  - Limité à un seul niveau super.super ne fonctionne pas
    - Attention il peut cependant y avoir un super dans le code de la casse mère
  - super( ) sans nom de méthode correspond à l'appel du constructeur
  - super(c,c) dans la classe Carre correspond à l'appel du constructeur de la classe rectangle

Super et constructeur

- Le super( ) utilisé pour appeler un constructeur de classe mère est **obligatoire** si les attributs sont privés (pourquoi ?)
- this(…) permet d'appeler le constructeur de la classe courante (permet de factoriser le code) → PointCartesien
- Les classes dérivées n'héritent **pas** des constructeur de la classe mère (à la différence des autres méthodes)

Héritage et attributs

- Une attribut final est de constant une fois qu'il a été initialisée par le constructeur (en majuscules en Java)
- Un attribut static est partagé par toutes les instances d'une classe

Par conséquent un attribut public static final est disponible dans toutes les classes

  - Par son nom dans les instances de la classe originelle
  - Par NomClasse.Attribut à l'extérieur
  - → la classe carte a créé les constantes Carte.TREFLE, Carte.PIQUE, ...

Recherche d'un attribut ou d'une méthode

- Si on désire accéder à un attribut ou à une méthode d'un objet d'une classe C, il (ou elle) devra être défini(e), soit :
  - Dans la classe C
  - Dans l'un des ancêtres de la classe C
  - Sinon c'est une erreur de programmation
- S'il y a eu des redéfinitions de la méthode, sa première apparition en remontant l'arborescence d'héritage est celle qui sera choisie
- Il faut bien évidemment que cet objet soit visible de la classe appelée

Résumé

- Quand B hérite de la classe A :
  - tout objet de B a toutes les propriétés d'un objet de A (+ d'autres)
  - un objet B peut être considéré comme un objet A
  - Les attributs définis dans un objet de A sont aussi présents dans B (mais ils peuvent être occultés)
  - idem pour les méthodes : les méthodes A sont présentes dans B et la classe B peut définir de nouvelles méthodes
  - B peut redéfinir des méthodes de A

Résumé

Carré hérite de la classe Rectangle

- pour les **attributs**:
  - tous les attributs de Rectangle sont accessibles pour un objet Carré (sauf si ils sont de type private)
  - la classe Carré peut ajouter des attributs (par exemple côté), et si le nom est identique cela occultera la variable de même nom

- pour les **méthodes** :
  - les méthodes de Rectangle sont accessibles dans Carré (sauf si elles sont private)
  - la classe Carré peut ajouter de nouvelles méthodes, par exemple une méthode agrandirCarré
  - Carré peut redéfinir des méthodes (par exemple la méthode toString())

Résumer super()

- super() permet d'appeler le constructeur de la classe mère
- C'est la première chose à faire dans la construction d'une sous-classe
- Un super() en deuxième ligne d'un constructeur ne compile pas
- Appeler le constructeur de la classe mère garantit que l'on peut initialiser les attributs déclarés par la classe mère quand ils sont private
- On passe les paramètres nécessaires

super() exemple

```
public class Superclass {

  public void printMethod() {
    System.out.println("Printed in Superclass.");
  }
}

public class Subclass extends Superclass {

  // overrides printMethod in Superclass
  public void printMethod() {
    super.printMethod();
    System.out.println("Printed in Subclass");
  }
  public static void main(String[] args) {
    Subclass s = new Subclass();
    s.printMethod();
  }
}
```

Résultat

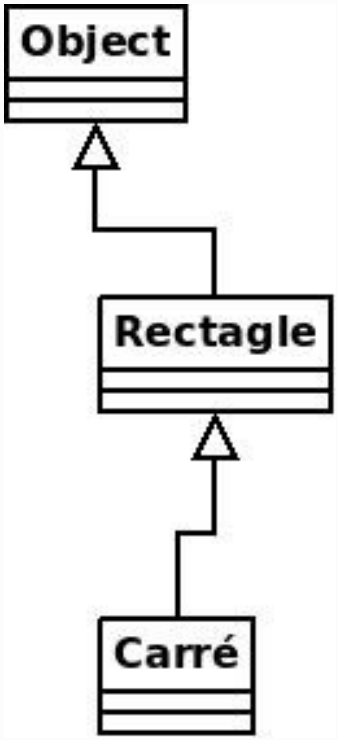
- Printed in Superclass.
- Printed in Subclass

Exercice

- Écrire une classe Animal
  - un attribut nom
  - un constructeur
  - une méthode d'affichage toString()
- Écrire une classe Poisson qui hérite d'Animal
  - deux attributs supplémentaires de type booléen (rivière et mer)
  - deux constructeurs
- Écrire une classe Oiseau qui hérite d'Animal
  - un attribut supplémentaire
  - deux constructeurs
  - une méthode d'affichage toString() (nom + couleur)
- Afficher un animal, un poisson et un oiseau. Quel est le problème ?

Hiérarchie des classes Java

- En Java (et uniquement en Java), toute classe hérite de manière automatique de la classe Object
- Les méthodes toString(), equals() et clone() sont des méthodes qui sont définies dans cette classe Object

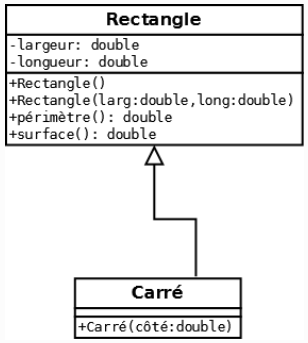


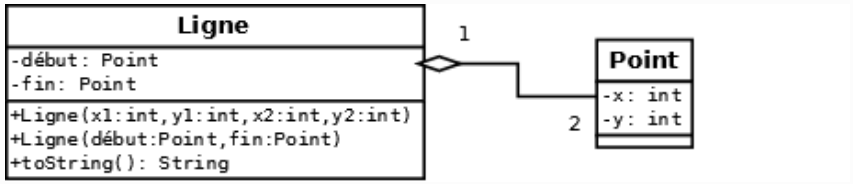
Méthode de la classe Object

- boolean equals(Object o) attention, == teste les références.
- protected Object clone() : crée une copie de l'objet.
- public String toString() renvoie une String décrivant l'objet. Par défaut, renvoie le type et l'adresse de stockage (=> redéfinir @Override)
- public final Class getClass() renvoie dans le type de l'objet sous forme d'un objet de la classe Class (réflexivité)

Héritage ou Composition

- L'héritage ou la composition (agrégation) sont deux formes de réutilisation de code

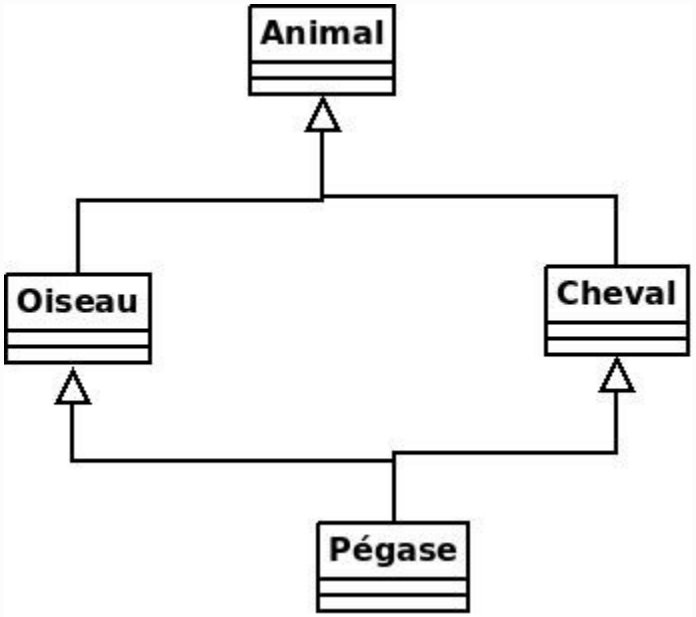




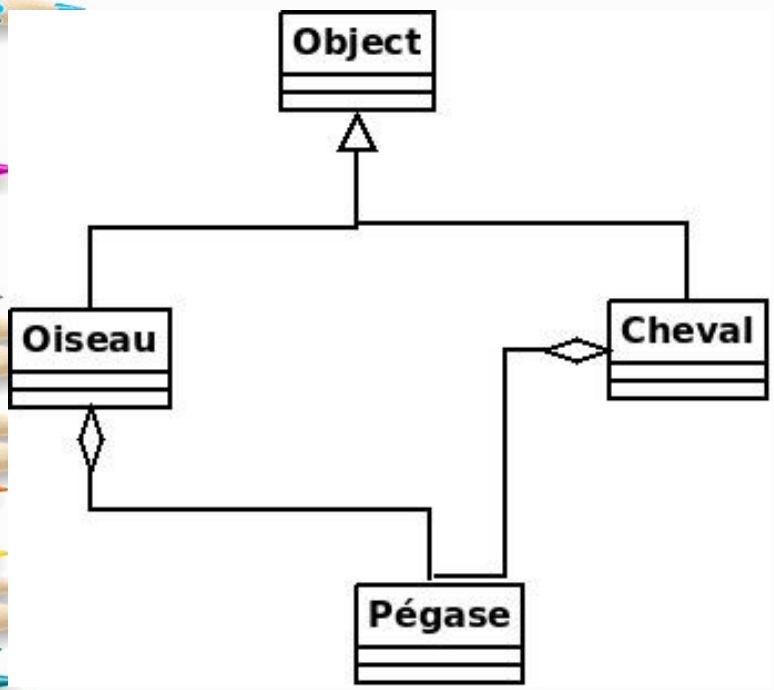
- Héritage
  - extends
  - Délégation implicite à la classe mère
  - Relation « est-un »
  - Un carré est un rectangle
- Composition (à préférer)
  - Référence à une instance
  - Délégation explicite à une autre classe
  - Relation « a-un »
  - Une ligne a un point (ici 2)

Héritage multiple

- Pas d'héritage multiple en Java (une seule classe parent)



Une solution : la composition




```
public class Pegase{

private Cheval cheval;
private Oiseau oiseau;

public Pegase(...){

}

}
```