Programmation Objet – Initiation à Java

Letícia SEIXAS PEREIRA

Cours 07 – Concepts de base de l'OO

Concepts de base de l'OO

1. **Encapsulation**
2. **Héritage**
3. Polymorphisme
4. Abstraction

Concepts de base de l'OO

Encapsulation

- Un objet porte en lui des données -> ces données sont *en principe* à l’usage exclusif de l’objet:

Concepts de base de l'OO

Encapsulation

- Un objet porte en lui des données -> ces données sont *en principe* à l'usage exclusif de l'objet:
  - La visibilité des membres d'une classe: toutes les variables d'instance (attributs) d'une classe doivent être déclarés **private**

```
public class Rectangle {

    private int longueur ;

    private int largeur ;

    // ...

}
```

Visibilité	public	private
Dans la même classe	oui	oui
Dans une classe du même *package*	oui	**non**

Concepts de base de l'OO

Encapsulation

- Un objet porte en lui des données -> ces données sont *en principe* à l'usage exclusif de l'objet;
- Il est possible de *consulter* ou de *mettre à jour* ces données seulement si l'objet accepte de le faire:
  - Il faut *demander* à l'objet en lui envoyant un message de communiquer les données qu'il conserve, ou de les mettre à jour.

Il faut que l'objet soit conçu pour répondre à ces requêtes

Concepts de base de l'OO

Encapsulation

- Un objet porte en lui des données -> ces données sont *en principe* à l'usage exclusif de l'objet;
- Il est possible de *consulter* ou de *mettre à jour* ces données seulement si l'objet accepte de le faire:
  - Il faut *demander* à l'objet en lui envoyant un message de communiquer les données qu'il conserve, ou de les mettre à jour.

```
public class Rectangle {

    private int longueur ;

    private int largeur ;

    // ...

}
```

```
    public void setLongueur(int l){
        this.longueur = l;
    }
```

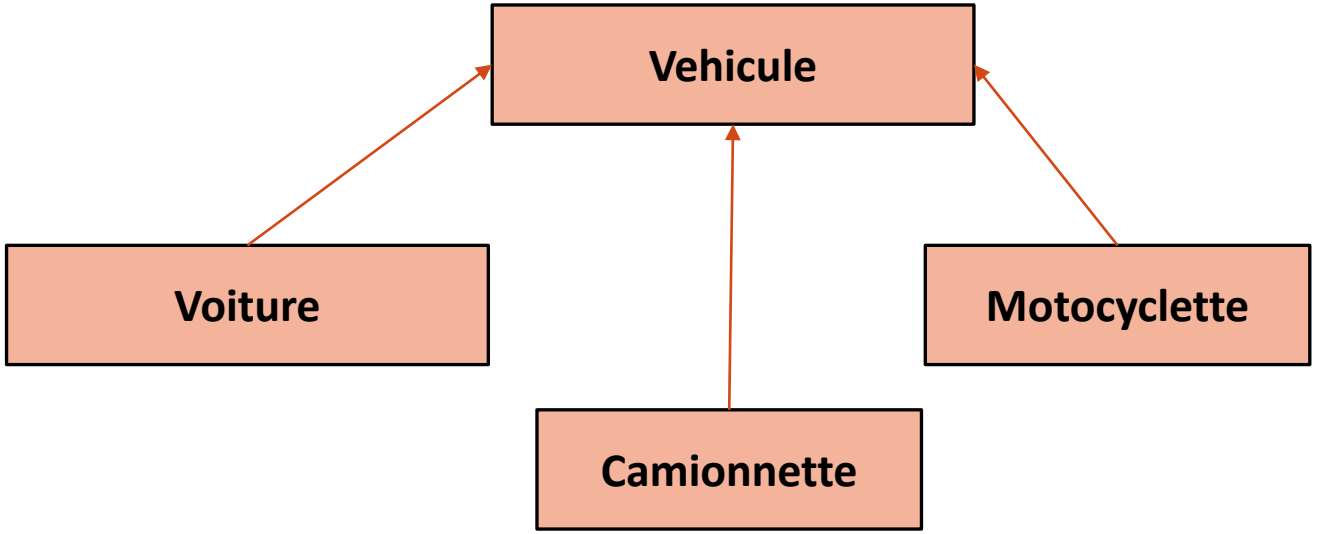
```
    public int getLongueur(){
        return this.longueur;
    }
```

Concepts de base de l'OO

1. **Encapsulation**
2. **Héritage**
3. Polymorphisme
4. Abstraction

Héritage

- La notion d'héritage est l'un des fondements de la programmation orientée objet;
- Une classe B **hérite** d'une classe A s'il existe une relation **« est un »** entre B et A.



« Voiture **est un** véhicule »

« Camionnette **est un** véhicule »

« Motocyclette **est un** véhicule »

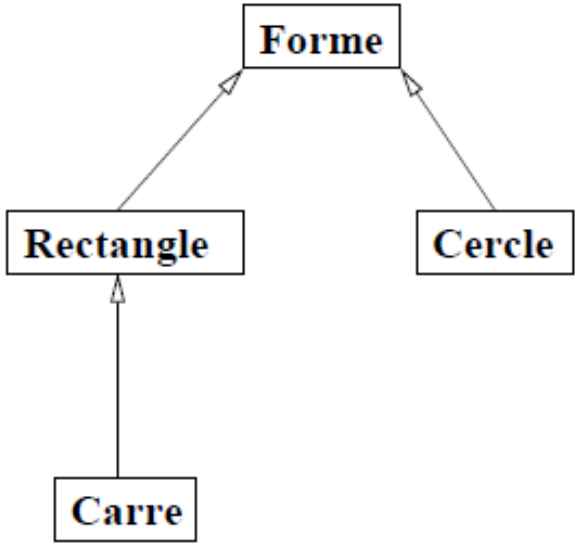
Héritage

- C’est un mécanisme qui permet de définir une nouvelle classe à partir d’une classe existante;
- Une classe fille hérite de tous les composants de sa classe mère (attributs et méthodes);
- Cela permet :
  - d’étendre une classe en lui ajoutant des composants (attributs et/ou méthodes);
  - de modifier le comportement d’une classe sans modifier la classe de base (= **redéfinir** les méthodes héritées).

Héritage

- **Déclaration:**

```
class SousClasse extends SuperClass
```




```
public class Carre extends Rectangle {

        //...

}
```

Polymorphisme

« Le polymorphisme désigne la capacité d'un objet à être conforme à plusieurs classes, selon une certaine hiérarchie d'héritage »

Polymorphisme

« Le polymorphisme désigne la capacité d'un objet à être conforme à plusieurs classes, selon une certaine hiérarchie d'héritage »

- Les instances d'une classe B, sous-classe d'une classe A, sont de type B, mais aussi de type A;

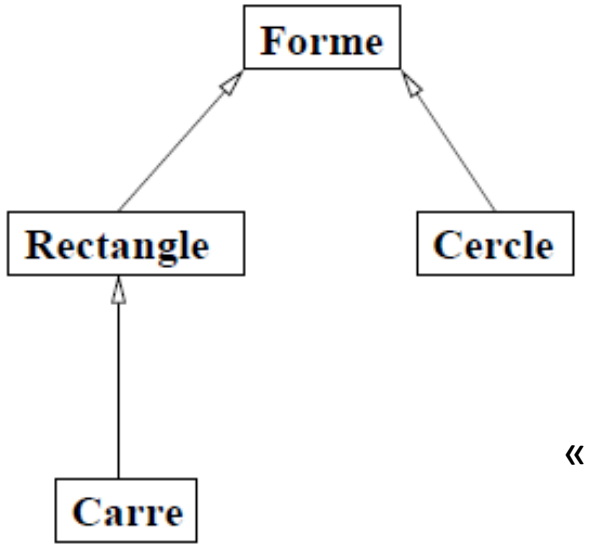
Polymorphisme

« Le polymorphisme désigne la capacité d'un objet à être conforme à plusieurs classes, selon une certaine hiérarchie d'héritage »

- Les instances d'une classe B, sous-classe d'une classe A, sont de type B, mais aussi de type A;
- Un objet qui selon une certaine hiérarchie d'héritage a plusieurs types est dit ***polymorphe***.

Polymorphisme

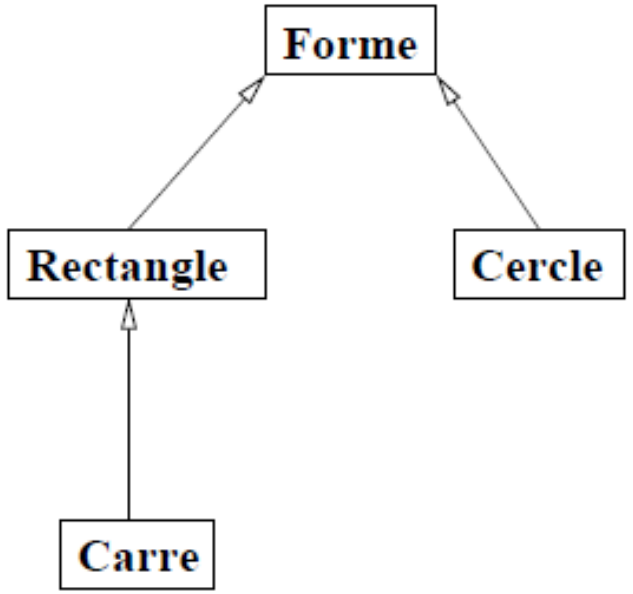
« Le polymorphisme désigne la capacité d'un objet à être conforme à plusieurs classes, selon une certaine hiérarchie d'héritage »



« Carre **est un** Rectangle »

« Carre **est une** Forme »

Polymorphisme




```
public class Forme {
    // ...
}
```

```
public class Rectangle extends Forme {
    // ...
}
```

```
public class Cercle extends Forme {
    // ...
}
```

```
public class Carre extends Rectangle {
    // ...
}
```

Polymorphisme

- Il a une seule classe "réelle" qui est celle dont le constructeur a été appelé en premier (c'est-à-dire la classe figurant après le **new**);
- Mais il peut aussi être déclaré avec une classe supérieure à sa classe réelle;

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



    }
}
```

Polymorphisme

- Il a une seule classe "réelle" qui est celle dont le constructeur a été appelé en premier (c'est-à-dire la classe figurant après le **new**);
- Mais il peut aussi être déclaré avec une classe supérieure à sa classe réelle;

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

        Forme r = new Rectangle(10,20);
        Forme c = new Carre(10);
    }
}
```

« Carre **est un** Rectangle »

« Carre **est une** Forme »

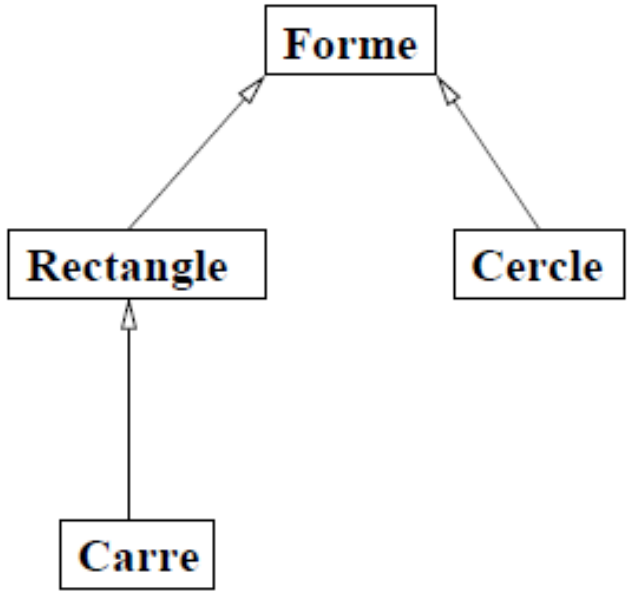
Polymorphisme

- Il a une seule classe "réelle" qui est celle dont le constructeur a été appelé en premier (c'est-à-dire la classe figurant après le **new**);
- Mais il peut aussi être déclaré avec une classe supérieure à sa classe réelle;
- Cette propriété est très utile pour la création d'ensembles regroupant des objets de classes différentes comme dans l'exemple suivant :

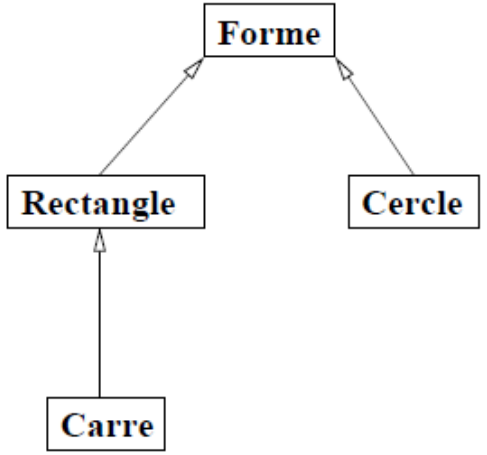
```
public class MainFormes{
    public static void main(String[] args){
        Forme[] tableau = new Forme[4];
        tableau[0] = new Rectangle(10,20);
        tableau[1] = new Rectangle(5,30);
        tableau[2] = new Carre(10);
        tableau[3] = new Forme();
    }
}
```

Polymorphisme

- Opérateur **instanceof**
  - Il permet de savoir à quelle classe appartient une instance d'un objet.



Polymorphisme

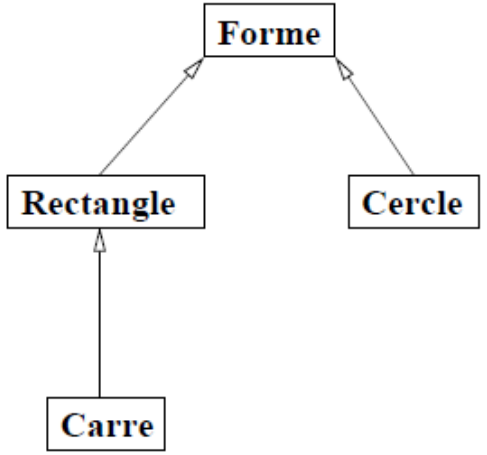


- Opérateur **instanceof**
  - Il permet de savoir à quelle classe appartient une instance d'un objet.

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

        Forme r = new Rectangle(10,20);
        Forme c = new Carre(10);

    }
}
```

c `instanceof Forme` → true

Polymorphisme

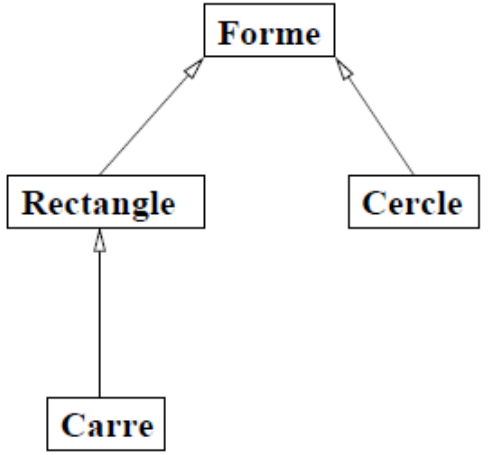


- Opérateur **instanceof**
  - Il permet de savoir à quelle classe appartient une instance d'un objet.

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

        Forme r = new Rectangle(10,20);
        Forme c = new Carre(10);

    }
}
```

```
c  instanceof Forme        true
c  instanceof Rectangle    true
```

Polymorphisme

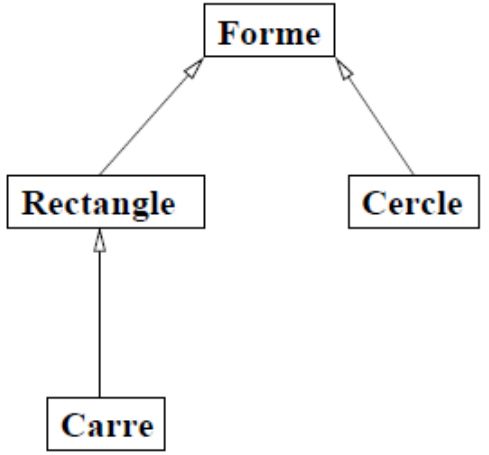


- Opérateur **instanceof**
  - Il permet de savoir à quelle classe appartient une instance d'un objet.

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

        Forme r = new Rectangle(10,20);
        Forme c = new Carre(10);

    }
}
```

```
c instanceof Forme        true
c instanceof Rectangle    true
c instanceof Carre        true
```

Polymorphisme



- Opérateur **instanceof**
  - Il permet de savoir à quelle classe appartient une instance d'un objet.

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

        Forme r = new Rectangle(10,20);
        Forme c = new Carre(10);

    }
}
```

```
c  instanceof Forme       true
c  instanceof Rectangle   true
c  instanceof Carre       true
```

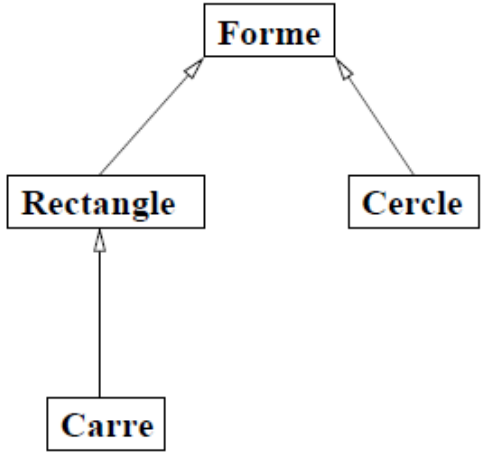
Donc,

un Carre est un Rectangle;

un Rectangle est une Forme;

un Carre est une Forme.

Polymorphisme



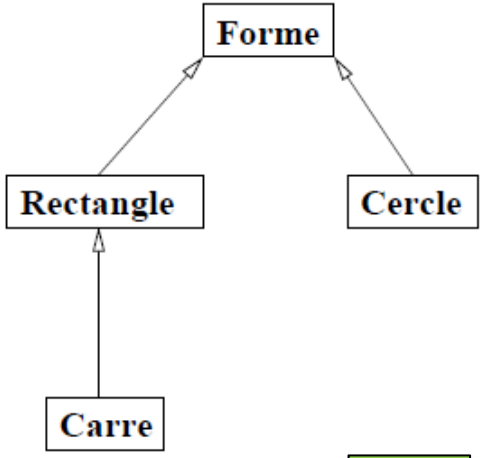
- Opérateur **instanceof**
  - Il permet de savoir à quelle classe appartient une instance d'un objet.

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

        Forme r = new Rectangle(10,20);
        Forme c = new Carre(10);

    }
}
```

```
r  instanceof Forme

r  instanceof Rectangle

r  instanceof Carre
```

Polymorphisme

- Opérateur **instanceof**
  - Il permet de savoir à quelle classe appartient une instance d'un objet.

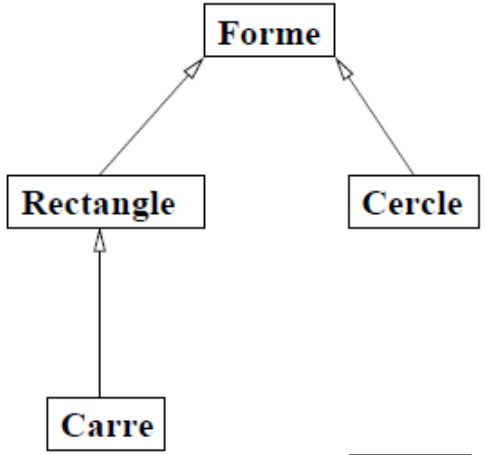



```
public class MainFormes{
    public static void main(String[] args){

        Forme r = new Rectangle(10,20);
        Forme c = new Carre(10);

    }
}
```

```
r  instanceof Forme        true
r  instanceof Rectangle    true
r  instanceof Carre        false
```

Polymorphisme

- Opérateur **instanceof**
  - Il permet de savoir à quelle classe appartient une instance d'un objet.




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

        Forme r = new Rectangle(10,20);
        Forme c = new Carre(10);

    }
}
```

```
r  instanceof Forme
r  instanceof Rectangle
r  instanceof Carre
```

true

true

false

Donc,

un Rectangle est une Forme.

Polymorphisme – Exercice d'application

- En utilisant la classe MainFormes suivante, pour chaque élément du tableau, affichez afficher à quelle classe il appartient:

```
public class MainFormes{
    public static void main(String[] args){
        Forme[] tableau = new Forme[4];
        tableau[0] = new Rectangle(10,20);
        tableau[1] = new Rectangle(5,30);
        tableau[2] = new Carre(10);
        tableau[3] = new Forme();
    }
}
```

```
element [0] est une forme
element [0] est un rectangle
element [1] est une forme
element [1] est un rectangle
element [2] est une forme
element [2] est un rectangle
element [2] est un carre
element [3] est une forme
```

```java
public class MainFormes {
    public static void main(String[] args) {
        Forme[] tableau = new Forme[4];
        tableau[0] = new Rectangle(10,20);
        tableau[1] = new Rectangle(5,30);
        tableau[2] = new Carre(10);
        tableau[3] = new Forme();

        for (int i = 0 ; i < tableau.length ; i++) {
            if (tableau[i] instanceof Forme) {
                System.out.println("element ["+i+"] est une forme");
            }
            if (tableau[i] instanceof Rectangle) {
                System.out.println("element ["+i+"] est un rectangle");
            }
            if (tableau[i] instanceof Carre) {
                System.out.println("element ["+i+"] est un carre");
            }
        }
    }
}
```

```
element [0] est une forme
element [0] est un rectangle
element [1] est une forme
element [1] est un rectangle
element [2] est une forme
element [2] est un rectangle
element [2] est un carre
element [3] est une forme
```