La classe Object

La classe Object

- En Java**, toutes les classes héritent de la classe** `Object`. La classe `Object` est donc la classe parente de toutes les classes. Les deux déclarations ci-dessous sont équivalentes.

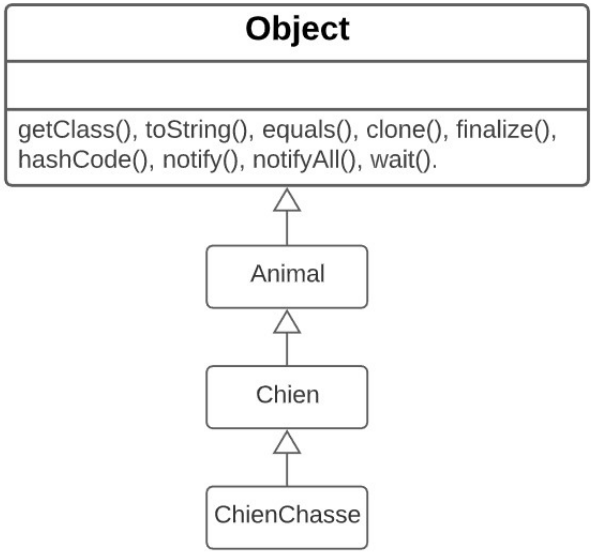
```
public class Rectangle{
}
```

```
public class Rectangle extends Object{
}
```

- La classe `Object` permet donc de présenter le comportement minimal de tout objet ainsi que le code par défaut pour ce comportement minimal.
- Les méthodes de la classe Object sont: **getClass(), toString(), equals(**), clone(), finalize(), hashCode(), notify(), notifyAll() et wait().
- Dans ce cours, on s'intéresse aux méthodes getClass(), toString() et equals().

La classe Object

- Toutes **les méthodes contenues dans la classe `Object` sont accessibles à partir de n'importe** quelle classe, car par héritage successif, toutes les classes héritent de la classe Object.




```
ChienChasse chien = new ChienChasse("Pitou");
chien.éléments accessibles de la classe Object
chien.getClass(), chien.toString(), chien.equals(),…
```

La classe Object

- On peut déclarer une variable de type Object et lui affecter un objet de n'importe quelle classe puisque toutes les classes héritent de la classe `Object`.

```
Object obj1, obj2;
obj1 = new Produit( ... );
obj2 = new chien( ... );
obj1.éléments accessibles de Object
obj2.éléments accessibles de Object
```

- On peut également convertir une variable de type `Object` en une variable d'une autre classe.

```
( (Produit) obj1 ).éléments accessibles de Produit et de Object
( (Chien) obj2 ).éléments accessibles de chien et de Object
```

La méthode equals() de la classe Object

- La méthode `equals()` de la classe `Object` permet de comparer deux objets. Elle retourne `true` si les deux objets pointent vers la même zone mémoire, et `false` dans le cas contraire.
- On doit redéfinir (remplacer) cette méthode dans nos propres classes afin d'effectuer une comparaison plus appropriée.
- Pour savoir si deux objets sont égaux, on utilise la méthode `equals()` et non pas l'opérateur `==`

```
if ( obj1.equals( obj2 ) ) {
   // obj1 est égal à obj2.
}
```

```
if ( obj1== obj2 ) ) {
   // obj1 et obj2 pointent vers la même
   // zone mémoire.
}
```

La méthode equals() de la classe Object

- **Exemple:** on redéfinit la méthode `equals()` dans la classe `Produit`. On suppose que deux objets de cette classe sont égaux (équivalents) s'ils ont le même numéro produit.

```
@Override
public boolean equals(Object autreObjet) {
   boolean egalite = false;
   // Si l'objet courant et l'autre objet pointent sur la même zone mémoire.
   if (this == autreObjet) {
      egalite = true;
   } else if (autreObjet != null && autreObjet instanceof Produit) {
       // Convertir le type de l'autre objet en type Produit.
       Produit autreProduit = (Produit) autreObjet;
       // L'objet courant et l'autre objet sont identiques si les numéros
       // des produits sont identiques.
          if (this.getNumero() == autreProduit.getNumero()) {
             egalite = true;
          }
   }
   return egalite;
}
```

La méthode equals() de la classe Object

- On peut alors comparer des objets de la classe Produit

```
Produit produit1 = new Produit(10, "Chaise", 132.56);
Produit produit2 = new Produit(20, "Table", 98.50);
Produit produit3 = new Produit(10, "Bureau", 120.00);

if (produit1.equals(produit3)) {
     System.out.println("Le produit " + produit1 + " est  identiques au produit" +
produit3);
}
```

- Affichage à la console:

```
Le produit  10: Chaise: 132,56$ est  identique au produit 10: Bureau: 120,00$
```

La méthode toString de la classe Object

- La méthode `toString()` définie dans la classe `Object` renvoi **le nom de la classe** suivi du **caractère** `@` et de **l'adresse mémoire** de cet objet en hexadécimale .

  Exemple: supposons que `toString()` ne soit pas redéfinie dans la classe `Rectangle`

```
Rectangle rect1 = new Rectangle(4,5);
System.out.println(rect); // affiche Rectangle@6d06d69c
```

- Lorsqu'on définit une classe, il peut être très utile de redéfinir la méthode `toString()` afin de donner une **description satisfaisante des objets de cette classe.**
- Lorsque la méthode `System.out.print()` reçoit en paramètre un objet de n'importe quel type, celle-ci fait appel à la méthode `toString()` de l'objet pour savoir comment l'afficher. Si cette méthode n'est pas redéfinie, c'est la méthode `toString()` de la classe `Object` qui est appelée.
- Exemple de redéfinition de méthode `toString()`.

```
public class Rectangle {
   private int longueur;
   private int largeur;
   …
   @Override
   public String toString() {
      return this.getClass().getName()+  "[" +
         this.getLargeur() + "," +
         this.getLongueur()+ "]";
}
```

```
public static void main(String[] args) {
      Rectangle rect = new Rectangle(45, 12);
      System.out.println(rect);
   }
```

- Affiche à la console

```
Rectangle[12,45]
```

La méthode getClass() de la classe Object

- La méthode `getClass()` retourne la classe d'un objet de type Class . Il existe donc une classe `Class` qui modélise les classes en Java.
- On peut connaître les caractéristiques de la classe en utilisant les méthodes de la classe `Class` :
  - `getName()` : retourne le nom de la classe incluant le package.
  - `isInterface()` : retourne `true` si la classe est une interface, `false` si non.
  - `IsArray()` : retourne `true` si la classe est un tableau.
  - `getSuperClass()` : retourne les informations sur la classe mère de la classe (de type `class`).
- **Exemple:**

```
Produit produit1 = new Produit(10, "Chaise", 132.56);
Produit produit2 = new Produit(20, "Table", 98.50);
System.out.println(produit1.getClass()); // class Produit
System.out.println(produit1.getClass().getName()); // Produit
System.out.println(produit1.getClass().isInterface()); // false
System.out.println(produit1.getClass().isArray()); // false
System.out.println(produit1.getClass().getSuperclass()); // class java.lang.Object
System.out.println((produit1.getClass() == produit2.getClass())); // true
```

La méthode getClass() de la classe Object

- Il est possible d'utiliser `getClass()` pour **vérifier si la classe d'un objet est la même que celle d'un autre objet**. Si les deux classes ne sont pas dans la même arborescence d'héritage (hiérarchie), cela provoque une erreur à la compilation.

```
if (objet1.getClass()==objet2.getClass()) {
      //traitement quand la classe de obj1 = classe de obj2
}
```