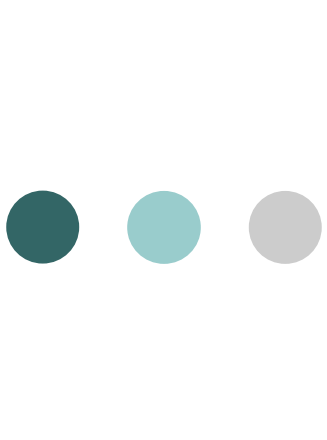


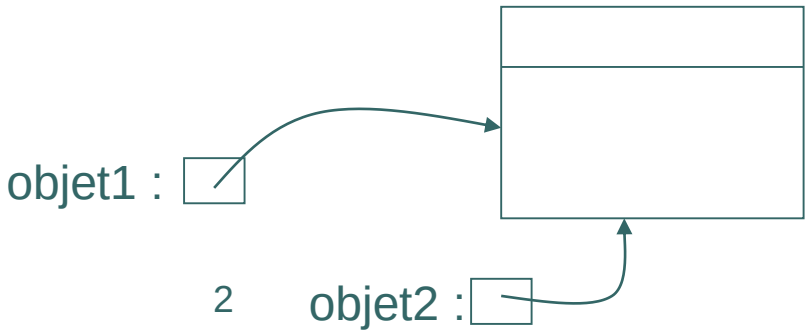
APOO

Égalité des objets

Egalité des objets

if(objet1 == objet2)

- objet1 et objet2 sont égaux (==) s'ils ont la même référence en mémoire.
- Il s'agit de comparer les références, c.-à-d. les adresses mémoires!



Mais …

- Supposons :
  - `Cercle cercle1 = new Cercle(5);`
  - `Cercle cercle2 = new Cercle(5);`

`Cercle1 != cercle2`

Pourtant, `cercle1` et `cercle2` sont identiques puisqu'ils ont le même rayon!

Egalité structurelle

- On va regarder la structure de l'objet pour définir l'égalité.
- Afin de réaliser cela en java, il faut définir deux méthodes :
  - equals
  - hashCode

equals : comment faire ?

Il faut respecter l'en-tête car la JVM appelle cette méthode de façon transparente pour l'utilisateur dans les collections comme **ArrayList**

```
public class Cercle{
   private double rayon;
   …
   public boolean equals(Object obj) {
        //s'ils ont la même adresse mémoire, ils sont
        //égaux.
        if (this == obj) return true;
        //si l'objet en paramètre est null, ils ne sont pas
        //égaux.
        if (obj == null) return false;
        //Si les objets ne sont pas de la même classe, ils
        //ne sont pas égaux.
        if (getClass() != obj.getClass()) return false;
        //On "transforme" le paramètre en un objet de type
        //Cercle et on compare les rayon.
        Cercle cercle = (Cercle) obj;
        return this.rayon == cercle.rayon;
   }
}
```

hashCode

- Quand on écrit la méthode equals, on est obligé d'écrire la méthode hashCode.

- Deux objets égaux doivent avoir le même hashCode.

```
public int hashCode() {
   return ((Double)rayon).hashCode();
}
```

Il faut respecter l'en-tête car la JVM utilise cette méthode dans certaines collections pour classer les objets et optimiser des méthodes Par exemple, la méthode contains de certaines collections utilise **`hashCode`** et **`equals`** !

Test de l'égalité des objets

Le test

```
if (objet1==objet2)
```

doit être remplacé par

```
if (objet1.equals(objet2))
```

Attention : Il faut être certain qu'objet1 n'est pas `null` avant d'appeler la méthode equals.

Remarques :

- Lorsqu'on ne définit pas les méthodes `equals` et `hashCode` dans une classe, Java en définit une par défaut en se basant uniquement sur la « référence » de l'objet.
- Comparaison du contenu de deux chaînes de caractères :

```
String chaine1 = …
String chaine2 = …

if (chaine1 == chaine2) …

//Il faut être certain que chaine1 ne soit pas null.
if (chaine1.equals(chaine2)) …
```

Classe ListeDeCercles en java (1)


```
import java.util.ArrayList;
public class ListeDeCercles{
   private ArrayList<Cercle> cercles;
   public ListeDeCercles(){
      cercles = new ArrayList<Cercle>();
   }
   public boolean ajouter(Cercle cercle){
      if (this.contient(cercle)) return false;
      return cercles.add(cercle);
   }
   public boolean supprimer (Cercle cercle){
      return cercles.remove(cercle);
   }
   public boolean contient(Cercle cercle){
      return cercles.contains(cercle);
   }
```

Classe ListeDeCercles en java (2)


```
    public int nombreDeCercles(){
        return cercles.size();
    }
    public String toString(){
        String texte = "Nombre de cercles : " +
                        cercles.size();
        for (Cercle c : cercles){
            texte += "\n" + c.toString();
        }
        return texte;
    }
}
```

ArrayList et égalité

- Pour voir si un objet est présent dans une ArrayList, java se base sur les méthodes `equals` et `hashCode`.
- Que va afficher ce programme ?

```
public class TestListeDeCercles {
   public static void main(String[] args) {
      ListeDeCercles liste = new ListeDeCercles();
      Cercle c1 = new Cercle(6.0);
      Cercle c2 = new Cercle(3.0);
      Cercle c3 = new Cercle(6.0);
      liste.ajouter(c1);
      liste.ajouter(c2);
      liste.ajouter(c3);
      liste.ajouter(c1);
      System.out.println(liste);
   }
}
```