

# La destruction d'objets

- Dans Java c'est le ramasse-miettes (ou **Garbage Collector** - GC en anglais) qui s'occupe de collecter automatiquement les objets qui ne sont plus référencés.
- Le ramasse-miettes fonctionne en permanence dans un thread de faible priorité (en « *tâche de fond* »). Il est basé sur le principe du compteur de références.

# La comparaison d'objet (1/2)

- ❑ L'opérateur “**==**” compare les références. Si 2 variables v1 et v2 pointent vers le même objet, v1 “v2 retourne true, et false sinon.
- ❑ La méthode equals() héritée de la classe Object compare les états (valeurs d'attributs) de 2 objets (de type String).


- **Exemple :**

```
Livre Livre1 = new Livre(123, "titre1", 54.67);
```

```
Livre Livre2 = Livre1;
```

```
Livre Livre3 = new Livre(123, "titre1", 54.67);
```

```
Livre1.equals(Livre3)  false
```

```
Livre2 == Livre1  true
```

```
Livre3 == Livre1  false
```

## La comparaison d'objet (2/2)

- Dans le cas des chaînes de caractères :

```
String str1 = new String("Hello World");
```

```
String str2 = new String("Hello World");
```

```
System.out.println(str1 == str2);    ➡ false
```

```
System.out.println(str1.equals(str2)); ➡ true
```

# L'autoréférence : this (1/2)

- Le mot réservé **this** , utilisé dans **une méthode**, sert à référencer l'instance de l'objet en cours d'utilisation
- Il peut être utile lorsqu'une variable locale ou un paramètre d'une méthode porte le même nom qu'un attribut.
- **this** est aussi utilisé quand l'objet doit appeler une méthode en se passant lui-même en paramètre de l'appel.
- Son rôle principale :
  1. **lever** une **ambiguïté sur les attributs**
  2. Dans un constructeur, pour **appeler** un **autre constructeur** de la même classe.

# L'autoréférence : this (2/2)

```
class Livre
{
    public int id;
    public String
titre; Livre(int
id,String titre)
    {
this.id=id;
this.titre=titre;
    }
}
```

Pour lever l'ambiguïté sur les mots  
« id » et « titre » et déterminer si c'est le  
« id/ titre » du paramètre ou de l'attribut

# Les packages

- ❑ Toutes ces classes sont organisées en packages (ou bibliothèques) dédiés à un thème précis.
- ❑ Parmi les packages les plus utilisés, on peut citer les suivants :

<code>java.applet</code>	Classes de base pour les applets
<code>java.awt</code>	Classes d'interface graphique AWT
<code>java.io</code>	Classes d'entrées/sorties (flux, fichiers)
<code>java.lang</code>	Classes de support du langage
<code>java.math</code>	Classes permettant la gestion de grands nombres.
<code>java.net</code>	Classes de support réseau (URL, sockets)
<code>java.rmi</code>	Classes pour les méthodes invoquées à partir de machines virtuelles non locales.
<code>java.security</code>	Classes et interfaces pour la gestion de la sécurité.
<code>java.sql</code>	Classes pour l'utilisation de JDBC.
<code>java.text</code>	Classes pour la manipulation de textes, de dates et de nombres dans plusieurs langages.
<code>java.util</code>	Classes d'utilitaires (vecteurs, hashtable)
<code>javax.swing</code>	Classes d'interface graphique

# Les chaines de caractères(1/4)

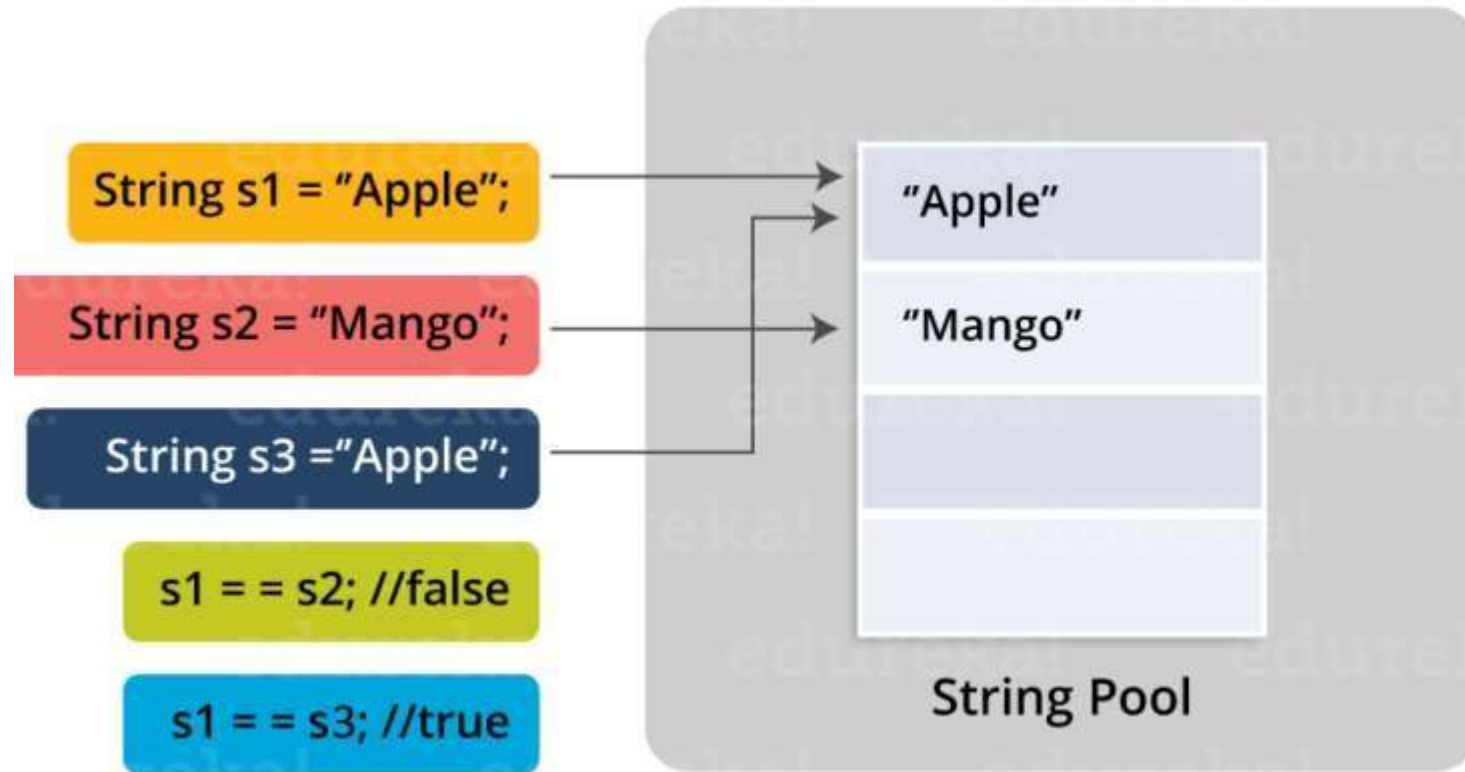
- ❑ En java, String est un type complexe , c'est-à-dire une classe mais particulière.
- ❑ Nous trouvons 4 manières de **déclaration** de chaine de caractères :
  - `String phrase; phrase = "Bonjour tous le monde "; //1ère méthode`
  - `String str = new String(); str = " Annaba ";//2ème méthode`
  - `String chaine1 = " chaine "; //3ème méthode`
  - `String chaine = new String("Algérie ");//4ème méthode.`
- ❑ La concaténation : l'opérateur + entre les deux chaines de caractère

## Les chaines de caractères(2/4)

- ❑ Comparaison de deux chaines de caractère :
- ❑ La méthode equals vérifie qu'elles possèdent exactement le même contenu
- ❑ L'opérateur == permet de comparer les références des deux chaines de caractères
- ❑ Deux chaînes strictement identiques sont placées à la même adresse dans le pool, et sont donc déclarées égales par l'opérateur "==".



# Les chaines de caractères(3/4)



# Les chaines de caractères(4/4)

- Exemple :

- ☐ String s1 = "Hello World";

- ☐ String s2 = "Hello World";

/\* La référence s3 pointe vers une instance de String, dont la chaîne n'est pas poolée. \*/

- ☐ String s3 = new String("Hello World");

/\* Vérifions que s1 et s2 ont la même adresse, mais que s3 pointe sur une adresse différente. \*/

- ☐ System.out.println(s1 == s2); /\* true \*/

- ☐ System.out.println(s1 == s3); /\* false \*/

- ☐ ici s1 et s3 ne sont pas sur le même string pool .

## ❑ Déclaration :

❑ Pour déclarer un tableau en Java, **on doit**:

- Spécifier un type des éléments du tableau (primitif ou objet), suivi de la notation [] (crochet ouvrant suivi de crochet fermant).

- **Exemple :**

- `int[] tableauEntier; // un tableau d'entiers.`
- `String[] tableauChaine; // un tableau de chaînes de caractères.`
- `Voiture[] tableauVoiture; // un tableau d'instances de la classe Voiture.`
- `int tab [] []; // tableau à deux dimensions`

## □ Initialisation :

□ Il existe deux méthodes pour initialiser un tableau vide :

```
int tableauEntier[] = new int[6];
```

```
String tableauChaine[] = {"chaine1", "chaine2", "chaine3",  
"chaine4"};
```

```
int[] tableauEntier = new int[] {1, 2, 3, 4};
```

```
int[] tableauEntier = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }; // Le  
tableau contiendra 10 éléments.
```

```
int[] tableauEntier = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }; //  
Le tableau contiendra 10 éléments.
```

```
int[] tableau = {1, 2, 3};
```

- ❑ Taille du tableau
- ❑ On utilise l'attribut **length**
- ❑ `int[] tableau = {1, 2, 3}; System.out.println(tableau.length);` // Imprime 3 sur la console.

❑ **Parcourir un tableau** par exemple :

```
int[] tab = new int[50] ;  
for (int index = 0; index < tab.length; index++) {  
    System.out.println(tab[index]);  
}
```

# Exercice 1

```
class A {  
    public static void main (String [] args){  
        A a,b,c ;  
        a=new A() ;  
        b=new A() ;  
        c=b ; a=b ;  
    }  
}
```

- ☐ Combien d'instances de la classe A sont créées pendant l'exécution du code suivant ?
- ☐ Solution : deux instances

## Exercice 2

```
class D {  
    public static int x ;  
    public int y ;  
    public static travailler() {  
        x++;  
    }  
    public D() {  
        x++; y-- ;  
    }  
}
```

- ☐ Qu'affichera le code suivant ?
- ☐ D.travailler() ; D a=new D() ; D b=new D() ; a.travailler() ;
- ☐ System.out.println(b.x + " et " + b.y) ;

## Exercice 2

- ☐ Solution :
- ☐ D.travailler() ; //incrémente x de 0 à 1
- ☐ D a=new D() ; //incrémente x de 1 à 2 et décrémente y de a de 0 à -1
- ☐ D b=new D() ; //incrémente x de 2 à 3 et décrémente y de b de 0 à -1
- ☐ a.travailler() ; // incrémente x de 3 à 4 et -1





- **Fin partie 3**