Chapitre 8

Les classes et les objets

Introduction

- Java est un langage très fortement objet., basé sur des classes.

- Java possède de très nombreuses classes, comme par exemple la classe **`String`**.

- Nous verrons ensuite comment construire et utiliser nos propres classes.

1. La classe String

Classe pré-définie du langage java

- Permet la gestion des chaînes de caractères.
- Contient un grand nombre d'outils pour faciliter leur utilisation.

Rappel :

Chaîne de caractère = ensemble de caractères.

Ne pas confondre avec char : un seul caractère.

1.1 Manipuler des mots en programmation

L'utilisation des chaînes de caractère est importante.

Ex: Stocker dans un logiciel des noms, messages, …

Les mots nécessitent un type de donnée particulier (un mot possède un nombre quelconque de caractères).

Déclaration d'une chaîne de caractères

– Nous devons déclarer une variable pour mémoriser la suite de caractères du mot.

⇒ Utilisation du type **`String`**

– Exemple de déclaration :

- `String mot = "exemple";`
- La variable **`mot`** est un ensemble de 7 cases mémoire contenant les 7 caractères du mot "exemple".

Remarque :

Les variables de type **`String`** ne contiennent pas directement l'information qui les caractérise mais seulement l'**adresse où trouver cette information.**

⇒ On ne parle plus de variable mais d'**OBJETS**

Les **objets** correspondent à un type qui permet de **regrouper sous une même adresse :**

- **Plusieurs <u>données</u>.**
- **Des <u>méthodes</u> (=fonctions)** permettant d'agir sur ces données.

1.2 Les différentes méthodes de la classe String

Pour réaliser des opérations sur les chaînes de caractères (=objets **`String`**), on utilise des **méthodes** (ou fonctions) prédéfinies.

- `int length()`

Retourne la longueur de la chaîne.

Ex:

```
String mot = "exemple";
int longueur;

longueur = mot.length();
System.out.print(longueur);   // affiche 7
```

- **`char charAt(int index)`**

Retourne le caractère placé à la position spécifiée en paramètre. Le premier caractère occupe la position **`0`** et le dernier la position **`length()-1`**

Ex:

```
String mot = "exemple";
char lettre;
lettre = mot.charAt(1);
System.out.print(lettre);  // affiche 'x'
```

- `String substring(int debut, int fin)`

Extrait une sous-chaîne dans un mot, à partir de « début » jusqu'à « fin » (exclu).

Ex:

```
String mot = "exemple";
System.out.print(mot.substring(2,5));
// Affiche « emp »
```

- **`boolean startsWith(String chaine)`**

Recherche si le mot commence par la chaîne passée en paramètre.

Ex:

```
String mot = "exemple";
if( mot.startsWith("ex") == true )
   System.out.print("mot commence par ex");
```

- **`boolean endsWith(String chaine)`**

Recherche si le mot se termine par la chaîne passée en paramètre.

- `int indexOf(String chaine)`

Localise un caractère ou une sous-chaîne dans un mot, à partir du début du mot.

Renvoie la valeur –1 si le caractère ou la chaîne recherchée ne fait pas partie du mot.

Ex:

```
String mot = "exemple";
System.out.print(mot.indexOf("emp"));
// Affiche 2
```

Méthodes pour comparer des chaînes

- `int compareTo(String autre_mot)`

Compare deux mots et retourne une valeur :

- Nulle si les deux mots sont identiques.
- Positive si le premier mot est plus grand (placé après le deuxième mot dans le dictionnaire).
- Négative si le premier mot est plus petit (placé avant le deuxième mot dans le dictionnaire).

- **`boolean equals(String autre_mot)`**

Compare la valeur de deux mots :

- retourne **`true`** si les deux mots sont identiques.
- retourne **`false`** sinon.

- **`boolean equalsIgnoreCase(String autre_mot)`**

Compare la valeur de deux mots sans différencier les majuscules des minuscules :

- retourne **`true`** si les deux mots sont identiques.
- retourne **`false`** sinon.

Méthodes de transformation de chaînes

- `String toLowerCase()`

Transformation en minuscules.

- `String toUpperCase()`

Transformation en majuscules.

- `String concat(String autre_mot)`

Concatène (accole) deux chaînes.

1.3 Appliquer une méthode à un objet

Ne pas confondre avec un appel classique de fonction.

Exemple :

```
double x=4, y;
y = Math.abs(x);
```

⇒ **`Math.abs()`** s'applique à **`x`** en passant la variable en paramètre.

⇒ Aucun traitement (méthode) n'est associé à cette information.

```
String mot="petit", MOT;
MOT = mot.toUpperCase();
```

⇒ **`toUpperCase()`** est appliquée à l'objet **`mot`** par l'intermédiaire d'un point (**`.`**) placé entre le nom de l'objet et la méthode.

⇒ Référence (adresse) vers un ensemble d'informations (caractères).

⇒ Utilisation de méthodes propres à l'objet.

2. Construire ses propres classes

Définir une classe, c'est construire un type structuré de données.

Deux étapes:

- Définition des données
  - Type simple: **`char`**, **`int`**, **`double`**, …
  - Type composé: **`String`**, autres objets, …
  - On les appelle des **champs, attributs** ou **membres** de la classe.

- Construction des méthodes
  - Méthodes associées aux données.
  - Elles s'écrivent comme de simples fonctions.

2.1 Construire une classe Cercle

Créer un type de données qui décrive au mieux la représentation d'un cercle quelconque :

- Rechercher les caractéristiques propres à tout cercle :
  - Rayon
  - Position (x,y) du centre

- Définir le comportement de tout cercle :
  - Déplacer()
  - Agrandir()
  - Périmètre()

A écrire dans un fichier **`Cercle.java`** :

```
public class Cercle
{
  // DONNEES DE LA CLASSE Cercle :
  public int x, y;   // position du centre
  public int rayon;  // rayon

  // METHODES DE LA CLASSE Cercle :

  // Affichage des données de la classe
  public void affiche()
  {
     System.out.println("Centre en " + x + "," + y);
     System.out.println("Rayon : " + rayon);
  }
 …
```

```
//Calcul du périmètre d'un cercle
public double périmètre()
{
   return 2*Math.PI*rayon;
}

// Déplace le centre du cercle en (nx, ny)
public void déplacer(int nx, int ny)
{
   x = nx;
   y = ny;
}

// Augmente la valeur courante du rayon
public void agrandir(int r)
{
   rayon = rayon + r;
}
```

```
    // Échange la position d'un cercle avec celle
    // du cercle passé en paramètre
    public void échange(Cercle autre)
    {
        int tmp;

        tmp = x;            // échanger la position en x
        x = autre.x;
        autre.x = tmp;

        tmp = y;            // échanger la position en y
        y = autre.y;
        autre.y = tmp;
    }
} // Fin de la classe Cercle
```

Les données **x**,**y**,**rayon** du type **Cercle** sont déclarées en dehors de toute fonction.

2.2 Quelques observations

- Les données **`x`**,**`y`**,**`rayon`** sont accessibles par toutes les méthodes de la classe.

- Il existe deux types de méthodes :
  - Celles qui permettent d'accéder aux données de la classe.
  - Celles qui permettent de les modifier.

- Une classe est définie pour être exécutée dans un programme exécutable qui contient une fonction **`main()`.**

2.3 Définir un objet

1) Déclarer un objet

```
Ma_classe  un_objet;
```

- **`Ma_classe`** correspond à une classe définie par le programmeur.

- Exemple:
  - **`Cercle  objet_cercle;`**
  - Cette déclaration crée une case mémoire nommée **`objet_cercle`**.
  - Cette case contiendra l'adresse de l'espace mémoire où seront stockées les informations concernant **`objet_cercle`**.

2) Réserver l'espace mémoire à l'aide de l'opérateur **new**

```
// réserver de l'espace mémoire pour l'objet
// un_objet
un_objet = new Ma_classe();
```

– Exemple :

```
Cercle  objet_cercle;
objet_cercle = new Cercle();
```

– Remarque : on peut aussi faire les deux opérations en une seule ligne :

```
Cercle  objet_cercle = new Cercle();
```

Remarques :

- **`new`** réserve suffisamment d'espace mémoire pour stocker les données de la classe et pour copier les méthodes associées.

- **`new`** détermine l'adresse où sera stocké l'ensemble de ces informations.

- Lors de la réservation, les données de la classe sont initialisées.

- L'objet défini est un représentant de la classe :

  ⇒ **`objet_cercle`** est une **instance** de la classe **`Cercle.`**

2.4 Manipuler un objet

- L'objet est entièrement déterminé par ses données et ses méthodes.

- Accéder aux données de la classe :

```
un_objet.nomDeLaDonnée = valeur du bon type;
```

- Accéder aux méthodes de la classe :

```
un_objet.nomDeLaMethode(paramètres éventuels);
```

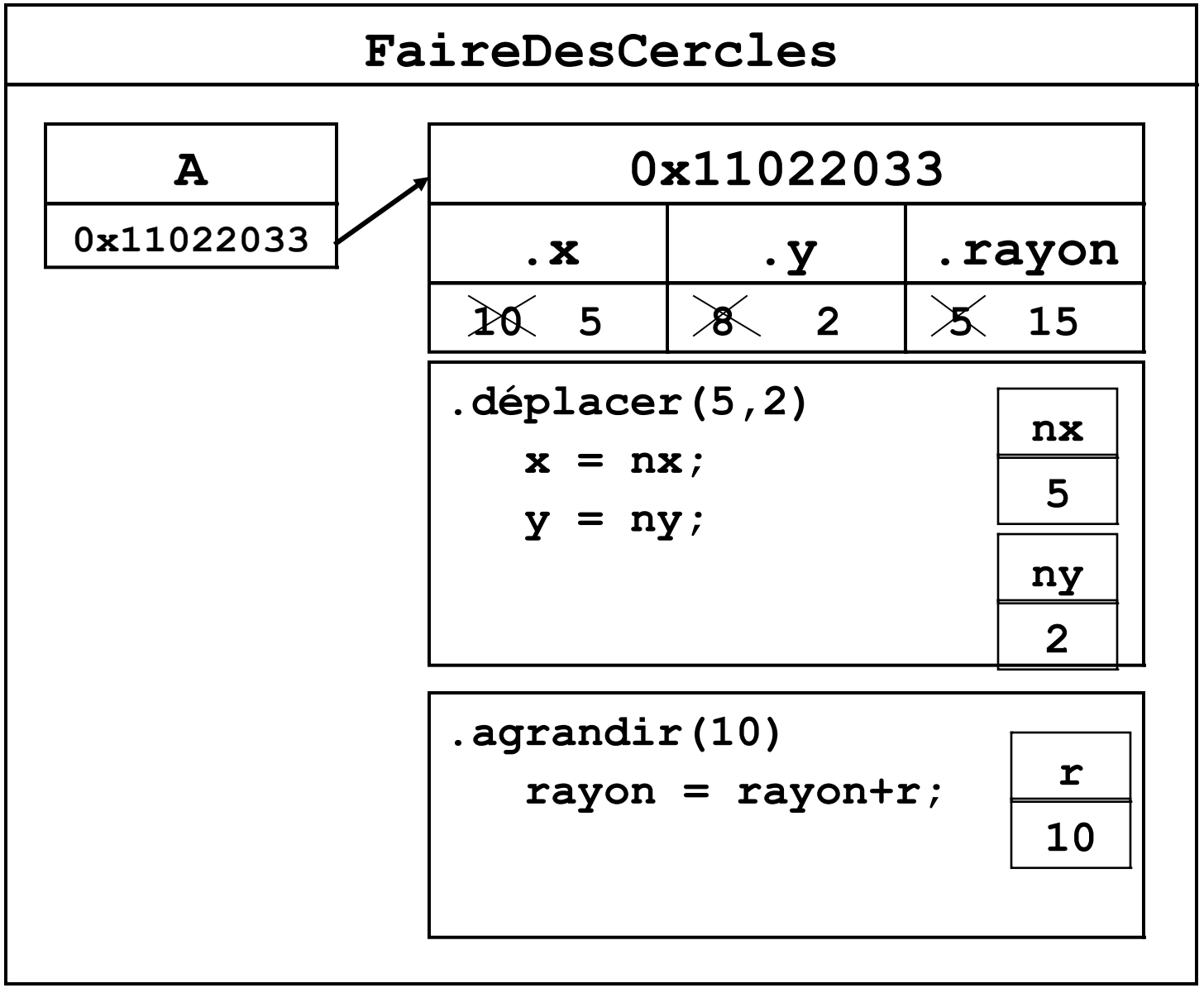
- Voir exemple Classe FaireDesCercles

Écrire la classe suivante dans un fichier **FaireDesCercles.java**, et le placer dans le même répertoire que **Cercle.java**.

```
public class FaireDesCercles
{
   public static void main(String [] arg)
   {
      Scanner clavier = new Scanner(System.in);
      Cercle A  = new Cercle();
      A.affiche();

      System.out.print("Entrez la position en x:");
      A.x = clavier.nextInt() ;
      System.out.print("Entrez la position en y:");
      A.y = clavier.nextInt() ;
      System.out.print("Entrez le rayon:");
      A.rayon = clavier.nextInt();
      A.affiche();
```

```
...

double p = A.périmètre();
System.out.print("Perimetre du cercle:" + p);

A.déplacer(5, 2);
System.out.println("Apres deplacement:");
A.affiche();

A.agrandir(10);
System.out.println("Apres agrandissement:");
A.affiche();

...
```

```
        ...

        Cercle B = new Cercle();
        B.x = 50;
        B.y = 50;
        B.rayon = 100;
        System.out.println("Le cercle B:");
        B.affiche();

        B.échange(A);

        System.out.println("Après echange le cercle A:");
        A.affiche();
        System.out.println("et le cercle B:");
        B.affiche();
    }
}
```


FaireDesCercles
A
0x11022033
0x11022033
.x
.y
.rayon
10 5
8 2
5 15
.déplacer(5,2)
x = nx;
y = ny;
nx
5
ny
2
.agrandir(10)
rayon = rayon+r;
r
10