.Adri.

Zeste de savoir

Les objets et la mémoire (débutant)

23 mai 2019

Table des matières

Le but de cet article est de mieux comprendre les objets et d'avoir une représentation visuelle des objets en mémoire.

1. La classe

1.0.1. Définition de la classe Cercle

Avant de pouvoir créer des objets, nous devons définir le type de ces objets. Ici, nous allons décider de créer des objets de type cercle. Par conséquent, nous devons créer une classe Cercle.

Une classe est un **type** (comme int, char, ...)

Cercle
int x ; int y ; int r ;
Cercle() agrandir() ; perimetre() ; afficher() ;

Notre classe Cercle comporte trois variables d'instances `x`, `y` (qui représentent les coordonnées du centre O du cercle dans l'espace) et la variable d'instance `r` qui définit le rayon du cercle. Ces variables s'appellent des **variables d'instances**, car elles appartiennent à un objet précis. On dit également qu'un objet de type cercle et une instance de la classe Cercle.

Variables d'instances

Les variables d'un objet s'appellent des variables d'instances.
On dit également qu'un objet est une instance de classe.

La classe Cercle possède également des fonctions qu'on appelle des **méthodes** en Orienté Objet.

Ces méthodes vont nous permettre de modifier ou récupérer des données sur les objets qui seront créés.
En Orienté Objet, il existe des méthodes "spécifiques" qui :

— permettent d'accéder[1] aux données comme `perimetre()`
— permettent de modifier les données comme `agrandir()`

Mais bien-sûr, il existe des méthodes qui ont d'autres rôles tels que `afficher()`.

2. Les objets

2.0.1. Créer un premier objet de type cercle

Après avoir créé une classe, nous pouvons créer différents **objets de type Cercle** c'est-à-dire qui sont **des instances de la classe Cercle**.

Création d'un objet instance de Cercle :

```
1 public static void main(String[] args) {
2     Cercle monCercle = new Cercle()
3 }
```

La création de l'objet précédent peut être découpée en 3 étapes :

1. Création d'une case mémoire dans **la pile** (Stack) où sera stockée **la référence**[2] `(0x11158920)` de l'objet.
   `monCercle` est donc une variable de référence.

   ```
   1     Cercle monCercle
   ```

2. Création d'un espace mémoire dans **le tas** (Heap) pour stocker les données et les méthodes de l'objet (par copie)

   ```
   1 new Cercle();
   ```

3. Affectation de l'objet qui est dans le tas à la case mémoire qui est dans la pile

   ```
   1 Cercle monCercle = new Cercle();
   ```

1. Par accéder, on entend une méthode qui va permettre de consulter les données. C'est une méthode qui renvoie un type en retour. Par exemple perimetre() renverra le périmètre du cercle.

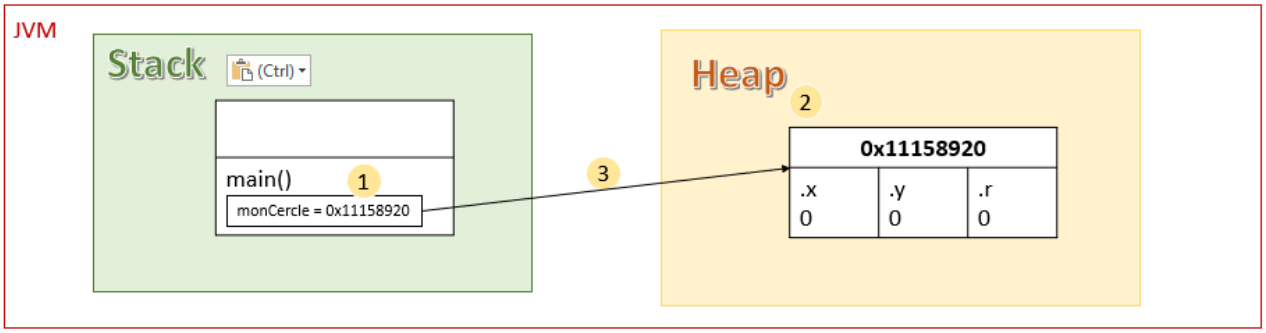

FIGURE 2. – Création de l'objet monCercle

La variable `monCercle` se trouve dans la pile et stocke l'adresse de l'objet qui lui se trouve dans le tas.

On remarque que l'objet possède :

— une **adresse** qui permet d'identifier et de retrouver l'objet dans le tas
— les **variables d'instances** de la classe Cercle (par copie)

En effet, chaque objet possède sa propre copie de sa classe. C'est-à-dire que chaque objet de type cercle possède une copie des variables d'instances.

Et, de plus, **les variables d'instances** possèdent une valeur par défaut : ici `0` car le type de donnée est int. En effet, lors de la réservation de l'espace mémoire, l'interpréteur initialise les données de la classe avec leurs valeurs par défaut .

Un objet

Chaque objet possède sa propre copie de la classe d'où il est issu. C'est-à-dire la copie des variables d'instances

Pourquoi posséder une copie des variables d'instances ?
Chaque objet pouvant avoir des valeurs de variable d'instance différentes, il ne faudrait que l'exécuteur fasse la confusion quand il affecte des valeurs aux objets.

2.0.2. Créer un second objet de type cercle

Maintenant que nous avons vu comment créer un objet cercle, nous décidons d'en créer un deuxième

```
1 Cercle autreCercle = new Cercle();
```

2. La référence est l'adresse de l'objet. Dans l'objet monCercle, la référence de l'objet est `0x11158920`

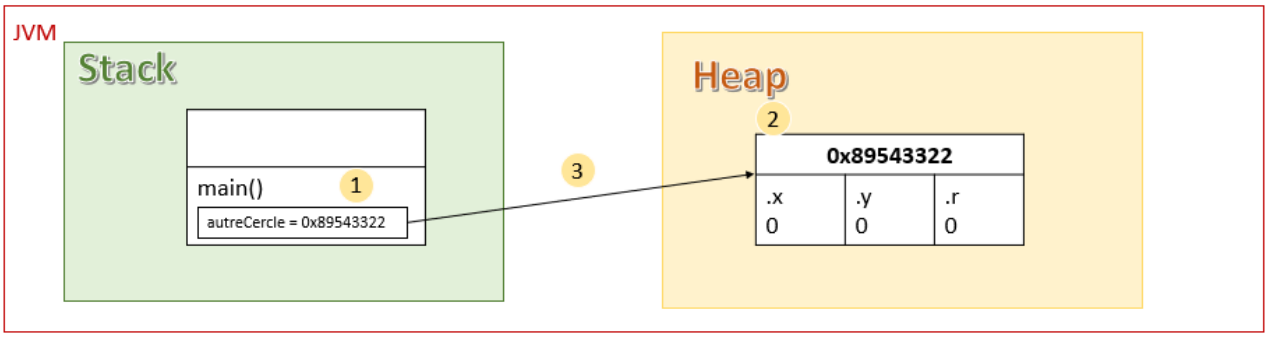

FIGURE 2. – Création d'un deuxième objet cercle : autreCercle

On remarque que la référence (= adresse) de l'objet autreCercle est différente que celle de l'objet monCercle. En effet, deux objets de même type ne partagent pas la même référence (de même pour des objets de type différent). C'est comme dans un village plusieurs personnes ont une boîte aux lettres mais personne n'a la même adresse. Sinon comment le facteur distribuerait son courrier ?
Les objets fonctionnent également pareil.

Même type mais adresses différentes

L'objet autreCercle possède sa propre copie de la classe Cercle. On a donc deux objets qui ont le même type, mais qui ont une adresse différente.

2.0.3. Mais où sont stockées les méthodes d'instance de la classe ?

Dans le premier schéma on voit bien que les variables d'instances ont été copiées. Mais qu'en est-il des méthodes de la classe Cercle ?

Les méthodes `Cercle()` `afficher()` `perimetre` et `agrandir` se trouvent dans un autre espace mémoire appelé **zone des methodes** (methods area).

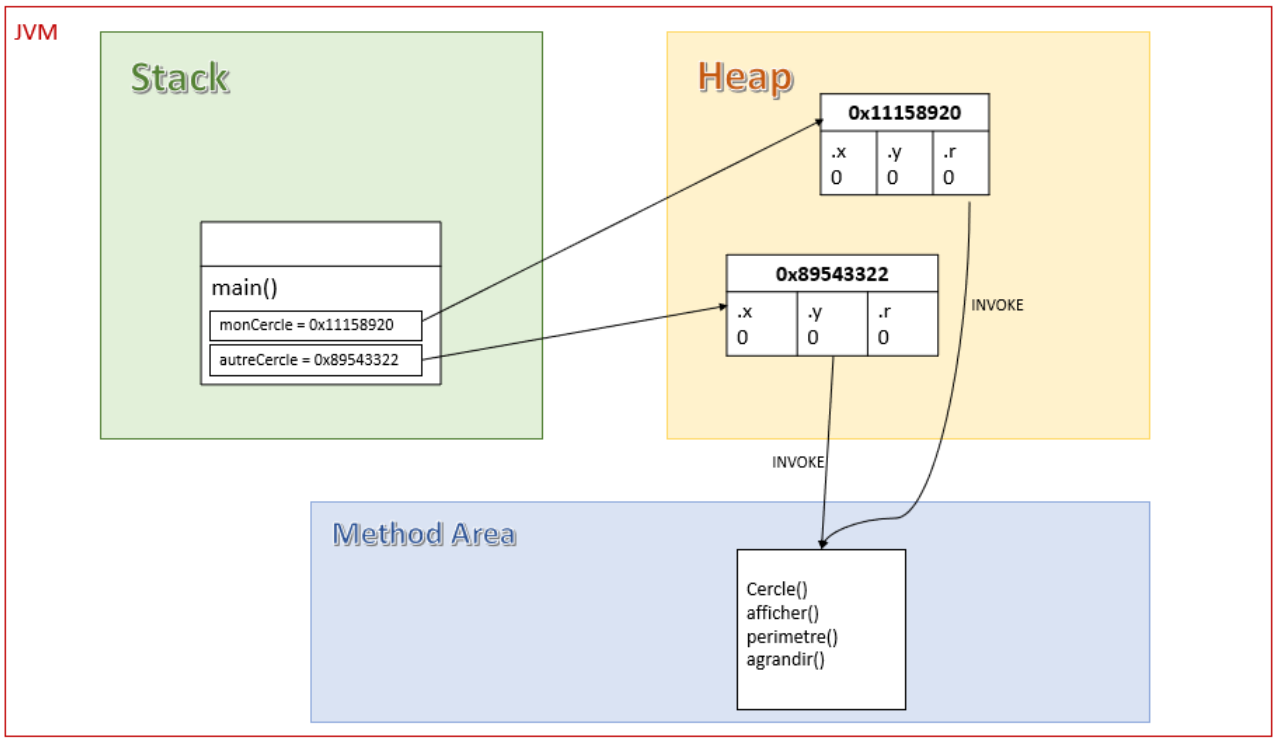

FIGURE 2. – La "methods area" se trouve dans la JVM est a pour but de stocker les méthodes de la classe

Lorsque la classe est chargée pour la première fois, cette dernière est compilée en Bytecode pour être ensuite exécutée par la JVM (et comme dans la classe on retrouve les méthodes d'instance elles seront donc compilées en Bytecode).
La méthode d'instance sera ensuite accessible grâce à une instruction en ByteCode : `INVOKE`.

Par exemple je souhaite appliquer la méthode `afficher()` à `monCercle()`.
J'écris donc

```
1 monCercle.afficher();
```

La JVM va maintenant passer à l'instruction `INVOKE` la référence de l'objet `monCercle` qui est dans le tas et la référence de la méthode `afficher()` qui est la la zone des méthodes.

3. Manipuler les objets

Maintenant que nous avons créé un objet, il serait intéressant de pouvoir le modifier et le consulter. Pour cela nous allons utiliser **les méthodes** que possède l'objet. Pour rappel un objet de type cercle possède comme méthode : `afficher()`, `perimetre()` et `agrandir()`

3.0.1. Accéder aux variables d'instances de la classe

Pour modifier les variables d'instance `x`, `y` ou `z` vous devez utiliser la notation suivante.

Syntaxe : `objet.variableDinstance = valeur;`

Par exemple si je veux définir les coordonnées dans l'espace de `monCercle` et également son rayon :

```
1 monCercle.x = 4;
2 monCercle.y = 6;
3 monCercle.r = 3;
```

L'objet monCercle aura pour coordonnée x = 4, y = 6 et comme rayon 3.

?

Pourquoi mettre `monCercle` devant les variables d'instances ?
Rappelez-vous l'histoire du facteur. Dans un village plusieurs personnes possèdent une boîte aux lettres.
Comment le facteur livre le courrier à vous et non à votre voisin ?
=> grâce à **l'adresse** On précise donc à quelle adresse on affecte les valeurs pour `x`, `y` et `r`. Ici c'est à l'adresse qui est stockée par la variable `monCercle` dans la pile.

Si on reprend le premier schéma on remarque que monCercle contient une adresse `0x11158920`. C'est grâce à cette adresse que l'exécuteur place les valeurs `4`, `6` et `3` à l'objet monCercle et non à l'objet autreCercle.

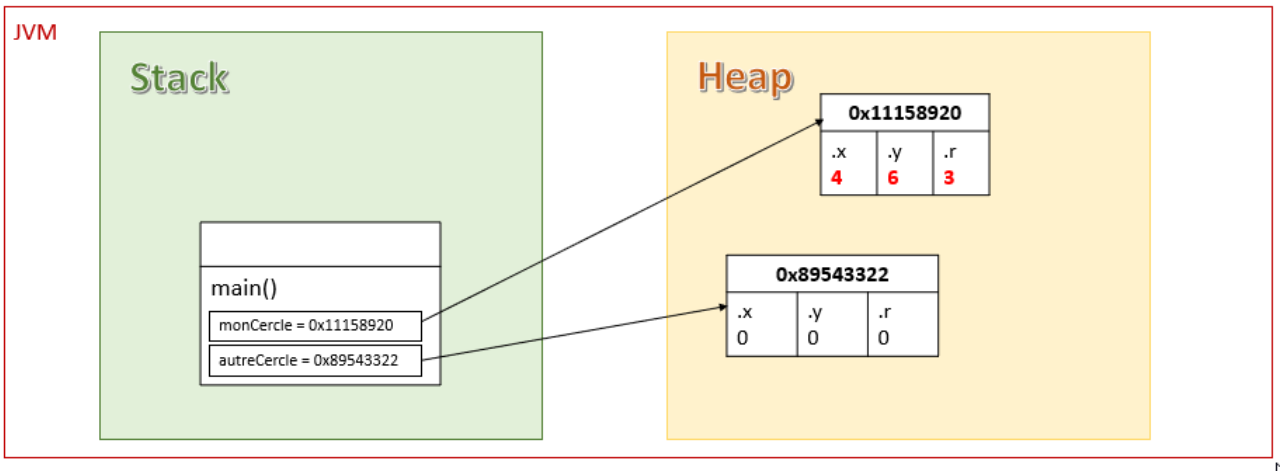

FIGURE 3. – Représentation des 2 objets créés, la fonction main() qui est dans la pile contient donc les deux variables locales où sont stockées les références de nos objets qui eux sont dans le tas. Ici, nous avons décidé de ne pas représenter la zone des méthodes mais elle y est toujours

Les valeurs des variables d'instances de monCercle ont été modifiées tandis que celle de autreCercle non. Rappelez-vous chaque nouvel objet est une copie de sa classe.

3.0.2. Accéder aux méthodes de la classe

Maintenant que nous avons modifié les valeurs de nos objets, nous allons voir comment utiliser les méthodes sur ces objets.

Syntaxe : `objet.methode(parametreEventuel);`

Si je veux afficher le cercle `monCercle`.
Je vais utiliser la méthode `afficher()` que je vais appliquer à l'objet `monCercle`.

```
1 monCercle.afficher();
```

Mais si maintenant je veux agrandir le rayon de mon cercle, le passer de 3 à 10, soit un agrandissement de 7. Je vais devoir utiliser la méthode `agrandir` et lui passer en paramètre la valeur de l'agrandissement : `7`. Et comme pour afficher(), je vais devoir appliquer cette méthode à l'objet monCercle :

```
1 monCercle.agrandir(7);
```

Comme pour les objets, on précise l'adresse ou on doit effectuer la méthode

---

Pour conclure, voici quelques points importants :

- Une classe est égale à un type de donnée
- Chaque objet possède sa propre copie de la classe
- Deux objets de même type n'ont jamais la même adresse
- Les objets sont accessibles grâce à leur adresse

Merci,et continuez d'apprendre