1 Les types

Java est un language *typé statiquement*. C'est à dire que toutes les variables, les entrées et les sorties de fonctions ont un type bien défini qui ne pourra pas être changé. Si une variable est déclarée comme un entier, on ne pourra pas la transformer en chaîne de caractères comme en Python.

	En java	valeurs
Vide	`void`	
Booléens	`boolean`	{`true`, `false`}
Très petits Entiers	`byte`	$[-128, \dots, 127]$
Petits entiers	`short`	$[-32768, \dots, 32767]$
Entiers	`int`	$[-2\ 147\ 483\ 648, \dots, 2\ 147\ 483\ 647]$
Grand entiers	`long`	$[-9\ 223\ 372\ 036\ 854\ 775\ 808, \dots, 9\ 223\ 372\ 036\ 854\ 775\ 807]$
Petits nombres à virgule	`float`	$[-1.4 \times 10^{-45}, 3.4 \times 10^{1038}]$
Grand nombre à virgule	`double`	$[-4.9 \times 10^{-324}, 1.7 \times 10^{10308}]$
Caractères	`char`	caractère Unicode (ex : 'v')
Chaînes de caractères	`String`	de 0 à 1 114 112 caractères Unicode (ex : "Hello World")

2 Les classes

Les *classes* sont des structures qu'on peut comparer à des moules. Avec un moule, on va pouvoir faire différents objets qui auront la même structure mais qui n'auront pas forcément les même caractéristiques.

Par exemple, une voiture. Toutes les voitures ont une marque, un modèle, une couleur, etc... On va donc déclarer que notre classe `Voiture` aura une `marque`, un `modele` et une `couleur`. A partir de cette classe, on va pouvoir créer plusieurs voitures avec des marques, modèles et couleurs différents.

En java, la casse de la première lettre d'un nom a un sens important. Si la première lettre est en majuscule, il s'agit TOUJOURS du nom d'une classe. Si c'est en minusucle, il s'agit de n'importe quoi d'autre. Pour le reste du nom, les lettres sont en minuscules sauf quand on change de mot. On appelle cette notation la *lowerCamelCase*. Exemple : `marqueDeVoiture` est une variable ou une fonction, alors que `MarqueDeVoiture` est une classe.

2.1 Les niveaux d'accès

Les niveaux d'accès permettent de bloquer ou ouvrir l'accès aux attributs ou méthodes.

— `public` permet d'ouvrir complètement l'accès aux autres classes.

— `private` bloque l'accès aux autres classes (les attributs et les méthodes en private son quad même accessibles à l'intérieur de la classe à laquelle elle appartient)

2.2 Les attributs

Ce sont les variables/caractéristiques spécifiques à la classe. La marque, le modèle et la couleurs sont des caractéristiques de la voiture. En général, tous les attributs sont en `private` (on protège les données des objets).

On les déclare toujours de la façon suivante :

```
private String marque;
private String modele;
private String couleur;
```

Remarque : On ne fait que déclarer les variable. On ne les initialise pas ici.

2.3 Les méthodes

Les méthodes sont des fonctions spécifiques à la classe. Comme dans toutes les fonctions en programmation, les méthodes sont comparables à des boites noires dans lesquels il y a une ou plusieurs entrées (les paramètres) et une seule sortie. On peut représenter sous la forme de ce type de dessin.

Les informations d'entrée et de sortie vont être déclarées dans la toute première ligne de la méthode, avant même d'entrer dedans. On appelle cette ligne la *signature* de la méthode. Pour le dessin, on aura la signature suivante :

```
public int nomDeMethode (float a, String b, char c)
```

En général, les méthodes sont en public.

Les méthodes s'écrivent de la façon suivante :

```
public int nomDeMethode (float a, String b, char c){
    instruction1;
    instruction2;
}
```

Dans les méthodes, on utilise le type `void` comme type de sortie quand la méthode ne retrourne aucune information. Autrement dit, si la fonction ne contient pas de `return`, le type de sortie est `void`.

2.4 Le constructeur

Le constructeur est une méthode spéciale. C'est la méthode qui va permettre de fabriquer les objets (instances) de la classe. C'est donc dans cette méthode qu'on va initialiser les attributs.

```
public Voiture(String marque, String modele, String couleur){
    this.marque = marque;
    this.modele = modele;
    this.couleur = couleur;
}
```

Remarque 1 : Elle n'a pas de type de sortie (même pas `void`).

Remarque 2 : Dans l'exemple, il y a le mot `this`. Ce mot permet d'indiquer à java qu'on utilise les éléments specifiques à la classe dans laquelle on est. Autrement dit, "`this.marque" signifie l'attribut marque` permet d'avoir à la fois l'attribut `marque` de la classe et une variable locale `marque` sans que l'une soit écrasée par l'autre.

TRES IMPORTANT : Le nom du constructeur est la copie exacte du nom de la classe (avec la majuscule). Le contructeur de la classe `Voiture` se définit `public Voiture(args)` et pas autrement.

Généralement, en java, un fichier contient une seule classe. Le nom de fichier doit TOUJOURS être exactement le nom de la classe (avec la majuscule). S'ils n'ont pas les mêmes noms,

2.5 Les accesseurs

Les accesseurs sont des méthodes qui permettent de contrôler l'accès aux attributs de l'extérieur de la classe (rappel : les attributs sont en private donc pas accessibles de l'extérieur de la classe). La signature contient toujours

- L'accès `public`
- Le type de sortie (qui dépend du type de l'attribut)
- Le nom, composé de "get" suivit du nom de l'attribut
- Aucun argument

Par exemple, l'accesseur de l'attribut `marque` :

```
public String getMarque(){
    return(this.marque);
}
```

2.6 Les mutateurs

Les mutateurs sont des méthodes qui permettent la modification d'un attribut de l'extérieur de la classe (rappel : Les attributs sont en private). La signature des mutateurs contient toujours

- l'accès
- `void` comme type de sortie (les mutateurs ne font que modifier l'attribut, ils ne rendent rien)
- Le nom, composé de "set" suivit du nom de l'attribut.
- Un argument contenant la nouvelle valeur à attribuer à l'attribut

Par exemple, le mutateur de l'attribut `marque` :

```
public void setMarque(String marque){
    this.marque = marque
}
```

2.7 L'instanciation

En java, TOUT est classe, les entiers, les chaînes de caractère, les nombres décimaux ou les classes que vous créez sont toutes des classes. Pour créer une variable, il faut la déclarer (afin que java attribut un espace mémoire pour elle) :

```
int variable;
```

On peut aussi l'initialiser, autrement dit, lui donner une "valeur" :

```
int variable = 3;
```

On appelle cette initialisation, l'instantiation d'une classe. Les classes qui permettent d'instancier les types sont un peu spéciales et les initialisations sont intuitives (= 3 pour donner la valeur 3). Pour les classes que l'on fabrique nous même, c'est un peu différent :

```
Voiture voiture1 = new Voiture("Renault", "Clio", "rouge");
```

On utilise le mot clé `new` qui permet d'appeller le constructeur de la classe, situé juste à coté (dans l'exemple, `Voiture(String marque, String modele, String couleur)`). Quand on a créer le constructeur, on a fait en sorte de "demander" à l'utilisateur certaines informations d'initialisation (ici, les valeurs de attributs). Comme c'est ici qu'on va vraiment créer à l'objet, c'est ici qu'on va mettre les valeurs de celui-ci.

Une fois notre objet instancié, on va pouvoir l'utiliser. Sachant que les attributs sont en `private`, ils ne sont pas accessibles ici. Par contre, les méthodes sont généralement en public. C'est donc avec elles qu'on va pouvoir jouer.

```
voiture1.getMarque();
```

Dans cet exemple, on appelle la méthode `getMarque()` de l'objet `voiture1` (qui est une instance de la classe `Voiture` dans lequel on a défini `getMarque`). Les parentèses à la fin de la méthode sont très importantes, elles indiquent qu'on appelle une méthode.

Sans l'utilisation des parentèses, java comprend qu'on appelle un attribut. On peut écrire `voiture1.marque` pour appeller l'attribut marque mais java refusera de compiler parce que les attributs sont en `private`.