Fiche de Révision POO

VAN DE MERGHEL Robin

Semestre 4

Table des matières

Introduction

Cette fiche de révision est maintenue au fur et à mesure du semestre, pour permettre de rester à jour sur le cours. Il peut y avoir de gaffes ! N'hésitez pas à me le signaler sur *Discord* : **UnSavantFou#2534**.

De plus, n'hésitez pas à me DM si jamais il y a des thématiques où j'ai mal formulé / que vous ne comprenez pas !

Programmation et Paradigmes

Chaque langage de programmation a une façon de modéliser et de résoudre un problème.

Par exemple un langage fonctionnel, comme `Prolog`, va modéliser un problème en utilisant des fonctions, alors qu'un langage orienté objet, comme `Java`, va modéliser un problème en utilisant des objets.

C'est **ça**, le paradigme : **la façon de modéliser un problème**.

Des exemples de Paradigmes

Pour de la culture générale (mais il est très conseillé de le savoir), voici quelques exemples de paradigmes :

À chaque fois, on va proposer une modélisation d'un problème simple : un chat qui mange de la nourriture.

Notez que le code n'est pas important, je veux juste montrer qu'on peut facilement modéliser le même problème dans chaque langage.

Fonctionnel

On n'utilise pas de variables, mais des fonctions. On ne modifie pas les données, mais on les transforme. On ne fait pas de boucles, mais on utilise des fonctions récursives.

Exemple : `Prolog`

```
% On définit un chat
chat(miaou, 5, 10).

% On définit une nourriture
nourriture(poisson, 5).

% On définit une fonction qui dit si un chat peut manger une
    nourriture
peutManger(Chat, Nourriture) :-
    chat(_, _, PoidsChat),
```

```
    nourriture(_, PoidsNourriture),
    PoidsChat > PoidsNourriture.
```

Impératif

On utilise des variables, on modifie les données, on utilise des boucles.

Exemple : `python`

```python
# On définit un chat
chat = {
    "nom": "miaou",
    "age": 5,
    "poids": 10
}

# On définit une nourriture
nourriture = {
    "nom": "poisson",
    "poids": 5
}

# On définit une fonction qui dit si un chat peut manger une
    nourriture
def peutManger(chat, nourriture):
    if chat["poids"] > nourriture["poids"]:
        return True
    else:
        return False
```

Orienté Objet

On utilise des objets, qui sont des variables qui contiennent des fonctions. On modifie les données, on utilise des boucles.

Exemple : `Java`

```java
// On définit un chat
class Chat {
    String nom;
    int age;
    int poids;

    public Chat(String nom, int age, int poids) {
        this.nom = nom;
        this.age = age;
        this.poids = poids;
```

```
    }
}

// On définit une nourriture
class Nourriture {
    String nom;
    int poids;

    public Nourriture(String nom, int poids) {
        this.nom = nom;
        this.poids = poids;
    }
}

// On définit une fonction qui dit si un chat peut manger une
    nourriture
public boolean peutManger(Chat chat, Nourriture nourriture) {
    if (chat.poids > nourriture.poids) {
        return true;
    } else {
        return false;
    }
}
```

Conclusion sur les exemples

On a à chaque fois le même problème : un chat qui mange de la nourriture. Mais on a utilisé des paradigmes différents. Si je reformule, des façons différentes de modéliser le problème.

Chaque paradigme a un avantage (pour ça qu'il existe).

- Le fonctionnel par exemple est très bon pour pouvoir trouver des bugs, on a une équation mathématique, on peut donc prouver facilement qu'un programme est correct.
- L'impératif est très proche de la machine, et très naturel pour les humains.
- L'orienté objet est très bon pour la modélisation de problèmes complexes, on peut diviser le problème en plusieurs objets.

Java, une brève introduction

`Java` est un langage crée à la base pour pouvoir faire de la programmation orientée objet.

Rapides spécifications

- Un langage de programmation orienté objet.
- Un langage compilé.
  - Il est d'abord précompilé, puis compilé.
- Il est exécuté sur une machine virtuelle.
  - C'est une machine virtuelle qui est installée sur votre ordinateur, et qui va exécuter le code compilé.
  - Cela permet d'être utilisable sur téléphones, tablettes, ordinateurs, etc.
- Il vient avec le JRE (Java Runtime Environment).
  - C'est un ensemble de bibliothèques qui permettent de faire des choses plus complexes.
  - C'est un peu comme le `stdlib` de `C`.

Un langage typé

Caractéristiques

`Java` est un langage typé. Cela signifie que chaque variable a un type, et que le compilateur va vérifier que vous n'utilisez pas une variable de type `int` comme une variable de type `String`.

De plus, comme le `C` ont doit explicitement déclarer le type de chaque variable.

```
int age = 10;
String nom = "Jean";
```

Les types

`Java` a plusieurs types de base :

- `int` : un nombre entier.
- `long` : un nombre entier sur 64 bits.
- `short` : un nombre entier sur 16 bits.
- `float` : un nombre décimal.
- `double` : un nombre décimal sur 64 bits.
- `boolean` : un booléen.
- `String` : une chaîne de caractères.
- `char` : un caractère (sur 16 bits, donc on a des caractères spéciaux comme `é`, `è`, etc.)
- `void` : rien.

On retrouve globalement les mêmes types que le `C`, mais on a aussi le type `String` qui est une chaîne de caractères.

Conventions de nommage

Il existe des conventions de nommage pour les variables, les fonctions, les classes, etc.

- Les variables et les fonctions sont en `camelCase`. → onEcritEnEscalier
- Les constantes sont en `UPPERCASE`. → ON_ECRIT_EN_MAJUSCULE
  → On a juste à rajouter le mot `final` devant la déclaration :

```
final int AGE = 10;
```

- Les booléens commencent par `is` ou `has`.
  - `isAlive`, `hasEaten`, etc.

Les tableaux

Comme le `C`, on peut déclarer des tableaux. On peut aussi déclarer des tableaux de tableaux.

Ils ont comme le `C` une taille fixe.

```
int[] tableau = new int[10];
int[][] tableauDeTableaux = new int[10][10];
```

Comme pour le `C`, on peut accéder aux éléments d'un tableau avec des indices.

```
tableau[0] = 10;
tableauDeTableaux[0][0] = 10;
```

Points communs avec le `C`

- Les boucles `for`, `while`, `do while`.
- Les conditions `if`, `else if`, `else`.
- Les opérateurs arithmétiques, logiques, etc.
- Les fonctions.

Globalement, on retrouve les mêmes choses que le `C`, mais avec des noms différents.

Ajouts par rapport au `C`

- On peut récupérer la taille d'un tableau avec la fonction `length`.

```
int[] tableau = new int[10];
int taille = tableau.length;
```

- On peut faire une boucle `for` sur une collection.

```java
for (int i : tableau) {
    // On fait quelque chose avec i
    // i est un élément du tableau
}
```

La compilation

Précompilation

La précompilation est une étape qui permet de générer des fichiers `.class` à partir de fichiers `.java`. Ces fichiers `.class` sont des fichiers binaires, et sont donc plus faciles à lire par la machine virtuelle.

Pour compiler un fichier `.java`, on utilise la commande `javac`.

```
javac MonFichier.java
```

Cela va générer un fichier `.class` qui contient le code compilé.

Cela permet d'avoir un code universel, qui peut être exécuté sur n'importe quel ordinateur.

Cela fait un énorme avantage de la **JVM** (Java Virtual Machine) car plusieurs langages peuvent être compilés pour être exécutés sur la JVM. Tous seront précompilés en `bytecode` (le code compilé pour la JVM).

Optimisations

La compilation permet aussi d'optimiser le code. Par exemple, si on a une boucle `for` qui ne sert à rien, le compilateur va l'optimiser et la supprimer.

Cet outil s'appelle **JIT** (Just In Time).

C'est une optimisation qui est faite au moment de l'exécution du programme (et non au moment de la compilation comme `gcc`).

Les objets en JAVA

Définition

Une classe

Une classe est un ensemble de variables et de fonctions qui permettent de définir un objet.

```java
public class Animal {
    private String nom;
    private int age;
```

```
    public Animal(String nom, int age) {
        this.nom = nom;
        this.age = age;
    }

    public void manger() {
        System.out.println("Je mange");
    }

    public void dormir() {
        System.out.println("Je dors");
    }
}
```

Ici, on définit une classe `Animal` qui a deux variables : `nom` et `age`. On a aussi deux fonctions : `manger` et `dormir`.

Une interface

Une interface est une classe abstraite qui permet de dire ce qu'on a le droit de faire avec un objet.

```
public interface IAnimal {
    public void manger();
    public void dormir();
}
```

Ici, on définit une interface `IAnimal` qui permet de dire que tout objet qui implémente cette interface doit avoir les méthodes `manger` et `dormir`.

Implémentations

Une classe peut implémenter une interface.

```
public class Animal implements IAnimal {
    private String nom;
    private int age;

    public Animal(String nom, int age) {
        this.nom = nom;
        this.age = age;
    }

    public void manger() {
        System.out.println("Je mange");
    }
```

```
    public void dormir() {
        System.out.println("Je dors");
    }
}
```

Ici, on définit une classe `Animal` qui implémente l'interface `IAnimal`. On a donc les mêmes méthodes que dans l'interface.

Instanciation

On peut instancier une classe avec le mot clé `new`. Autrement dit, on peut créer un objet à partir d'une classe.

```
Animal animal = new Animal("Toto", 10);
```

Ici, on crée un objet `animal` de type `Animal` avec le nom `Toto` et l'âge `10`.

Utilisation

On peut utiliser les méthodes d'un objet avec le point.

```
animal.manger();
animal.dormir();
```

Ici, on appelle les méthodes `manger` et `dormir` de l'objet `animal`.