Conception et Programmation Objet - GM3

Introduction au monde des objets

Mathieu Bourgais

2023

Plan

Organisation du cours

- 7 sessions de cours + TD/TP
- Cours et TD/TP sur moodle
- En parallèle avec les cours de modélisation avec Aurore Blot
- Un TP noté à la fin du semestre
- Un DS sur table

- mail : mathieu.bourgais@insa-rouen.fr
- bureau : BO A R1 10

Les notions abordés

- Réification, classes, héritage et polymorphisme
- Interfaces, classes abstraites et encapsulation
- Collections, égalités et tableaux
- Exceptions
- Entrées/Sorties, flux et organisation en paquets
- Design Pattern et du bonus
- FAQ avant l'examen

Objectif généraux du cours

- Découvrir et s'imprégner du paradigme de la **Programmation Orientée Objet**
- Découvrir le langage **Java**
- Vous apprendrez des fonctionnalités plus avancées de Java en GM4
- Une fois que vous maîtriserez la Programmation Orientée Objet, vous n'aurez aucun mal à prendre en main des langages comme **Python** ou **Kotlin** (et pleins d'autres)
- Vous ne serez pas des experts de Java !

Plan

Petit historique

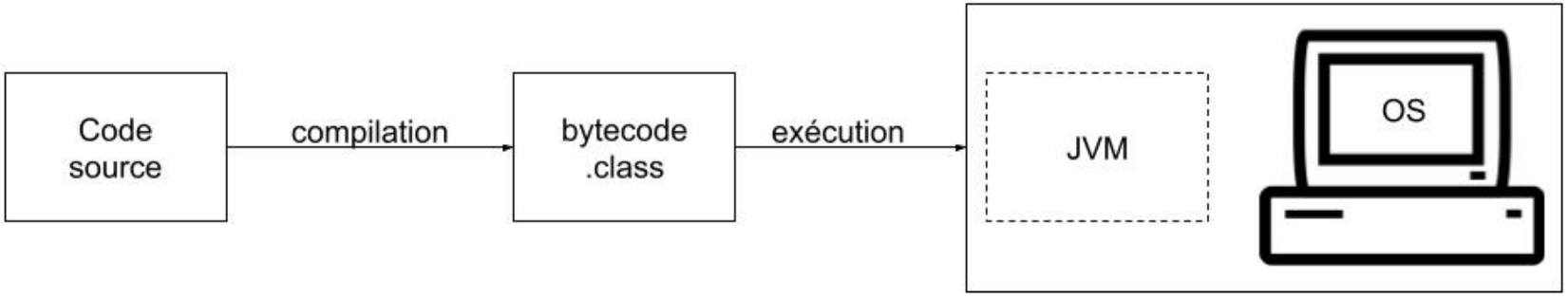
- Années 1960 : Premières discussions et premiers langages objets.
- 1990/1991 : Lancement du *Green Project* chez **Sun Microsystems** pour une technologie portable sur différents appareils
- 1994 : Le langage Java prend son nom et sert de base au navigateur internet *WebRunner*.
- 1996 : Sortie du Java Development Kit 1.0
- 2006 : Le compilateur *javac* et la machine virtuelle *HotSpot* deviennent Open Source sous GNU GPL.
- 2009 : Rachat de Sun par **Oracle**

Caractéristiques de Java

- Langage objet (héritage simple)
- Langage à part entière, pas uniquement web
- Portabilité grâce à la JVM
- Machines virtuelles inclues dans les navigateurs (applet)
- Sécurisation des exécutions grâce à la JVM
- Gestion de la mémoire par ramasse-miette (garbage collector)
- Très utilisé donc de nombreuses bibliothèques sont disponibles

La JVM

- Java s'exécute sur une machine virtuelle dédiée : la **JVM**
- Le code source Java est compilé vers un **bytecode** (.class)
- Ce bytecode est ensuite interprété ou "compilé à la volée" par la machine virtuelle, quelque soit la machine réelle derrière.
- C'est la JVM qui va s'occuper des optimisations potentielles, et pas le compilateur d'origine.



Plan

Types de base

Dans Java, on retrouve tous les types de base que vous avez l'habitude de manipuler :

- byte (entier sur 8 bits)
- short (entier sur 16 bits)
- int (entier sur 32 bits)
- long (entier sur 64 bits)
- float (flottant sur 32 bits)
- double (flottant sur 64 bits)
- char (mène au type String)
- boolean (true/false)

Opérations de base

Opérateurs arithmétiques	+ − * / %
Opérateurs d'affectation	= += −= *= /= %= ++ −−
Opérateurs de comparaison	< > <= >= == !=
Opérateurs logique	! && ||

```
int a = 2;
int b = 3;

System.out.println(a+b); //5
System.out.println(a-b); //-1
System.out.println(a*b); //6
System.out.println(b/a); //1.5
```

Opérations de base (suite)

Conditionnel :

```
if (conditionBooleene()){
    ...
    } else {
    ...
    }
```

Choix multiple :

```
int x;
switch(x) {
    case 1 :
        fonction1();
        break;
    default :
        fonctionDefault();
}
```

Boucles

Boucle déterministe :

```
for(int i = 0; i<11; i++){
    System.out.println(i);
}
```

Boucle indéterministe :

```
int i=0;
while (i<11){
    System.out.println(i);
    i++;
}
int j=0;
do{
    System.out.println(j);
    j=j+1;
}while (j<10)
```

La classe un type complexe

Une **classe** est un **type complexe** qui représente un **objet** ; on parle de **réification**.

Une classe est composée de :

- Un nom unique
- Un ensemble d'**attributs** qui représentent des valeurs permettant d'identifier l'objet (qui peuvent être de type simple ou de type complexe)
- Un ensemble de **méthodes** qui indiquent les actions que l'objet peut réaliser et comment ces actions sont réalisées
- Au moins un **constructeur** pour pouvoir créer des **instances** de la classe avec une initialisation stable.

Accesseurs et mutateurs

Une bonne pratique consiste à déclarer les attributs d'une classe comme des éléments "privé" (*private*) et à créer des méthodes d'accession et de modification qui seront accessibles de façon "publique" (public) : les **accesseurs** ("*getter*") et les **mutateurs** ("*setter*").

```
private int vie;

public int getVie(){
    return this.vie;
}

public void setVie(int vieModif){
    this.vie = vieModif;
}
```

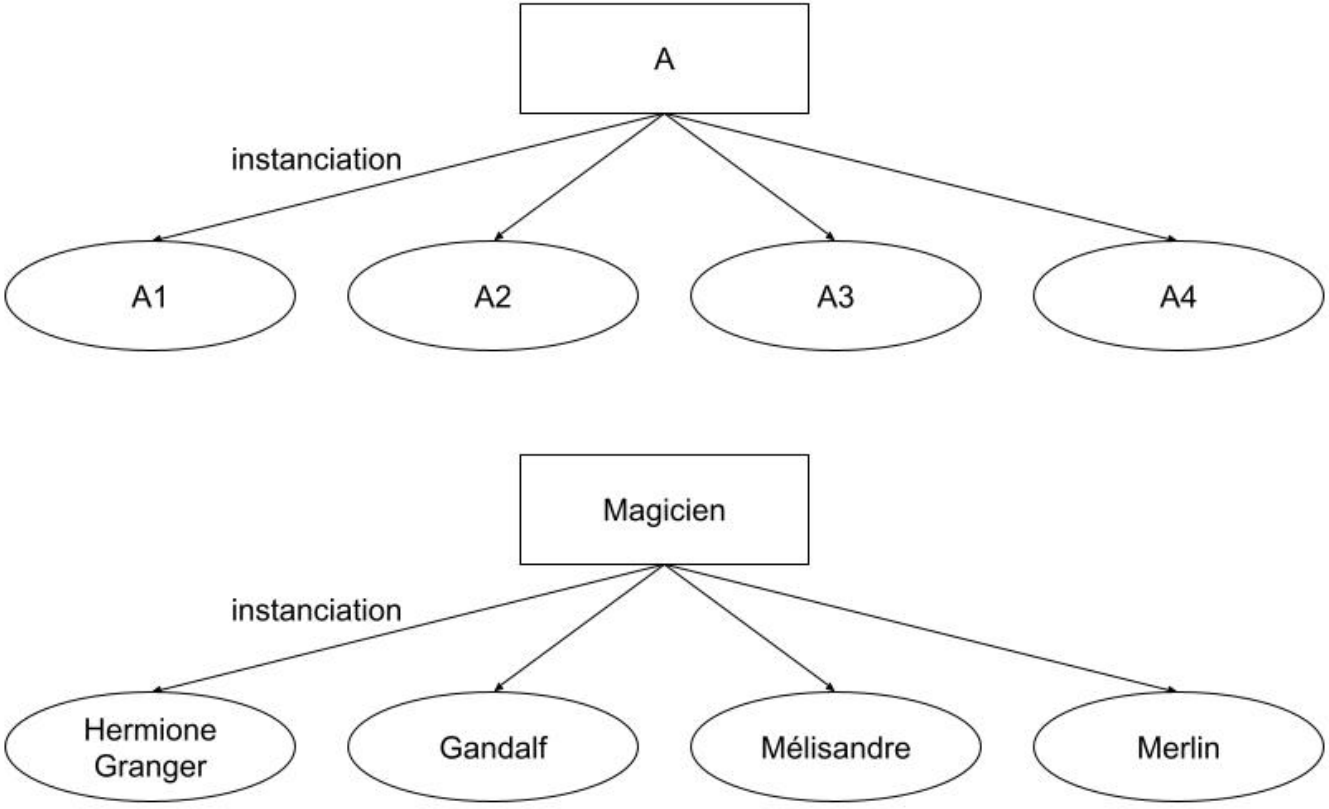
Exemple du magicien

Voici sous la forme d'un tableau la classe représentant un magicien dans un jeu, après réification.

Nom	Magicien
Attributs	String nom int vie int mana
Méthodes	Magicien constructeur() String getNom() Void setNom(String nom) Int getVie() Void setVie(int vie) Int getMana() Void setMana(int entierModificateur) Void lancerSort()

Utiliser les classes en créant des instances

Une classe représente un concept ; pour être utilisée, une classe doit être instantiée dans un contexte précis.



Construire une instance

Pour fabriquer l'instance, nous allons nous appuyer sur la définition d'un constructeur pour chaque classe. Ce constructeur a obligatoirement le même nom que la classe dans laquelle il est et "ne renvoie rien".

```
public Magicien(String nom){
    this.nom = nom;
    this.vie = 10;
    this.mana = 10;
}
```

Par la suite, on déclarera la nouvelle instance à l'aide de l'opérateur **new**.

```
Magicien hermione = new Magicien("Hermione␣Granger
    ");
```

La notion d'héritage

Pour factoriser le code et favoriser sa réutilisation, il y a l'**héritage**.
Les propositions suivantes sont donc équivalentes :

- La classe A hérite de la classe B
- La classe A possède au moins tous les attributs et toutes les méthodes de la classe B
- La classe B est la classe mère de la classe A, qui est une classe fille de B
- La classe A étend la classe B

Il est bien évidemment possible d'avoir un héritage en cascade (A hérite de B qui hérite de C) et plusieurs classes filles peuvent héritées de la même classe mère (A et C héritent de B).
En Java, il n'est pas possible d'hériter de plusieurs classes en même temps. Nous verrons plus tard comment contourner cela.

Hériter des attributs et redéfinir des méthodes

Quelques mots-clés liés à l'héritage :

- **extends** : permet de définir quelle classe est héritée
- **super** : fait référence à la classe mère. Permet notamment d'accéder à des méthodes de la classe mère selon le format super.maMethode() ;
- **super()** : fait référence au constructeur de la classe mère
- **this** : fait référence à l'objet lui-même. Permet de désambiguïser l'accès à des attributs ou à des méthodes selon le format this.attribut ; this.maMethode() ;
- **this()** : fait référence au constructeur de l'objet lui-même

Polymorphisme

Lors de l'héritage, on peut avoir envie de redéfinir une méthode de la classe mère pour la spécifier dans la classe fille. On peut aussi vouloir définir une méthode similaire à une existante en ajoutant ou supprimant quelques paramètres d'entrée.
On parle alors de **polymorphisme**.

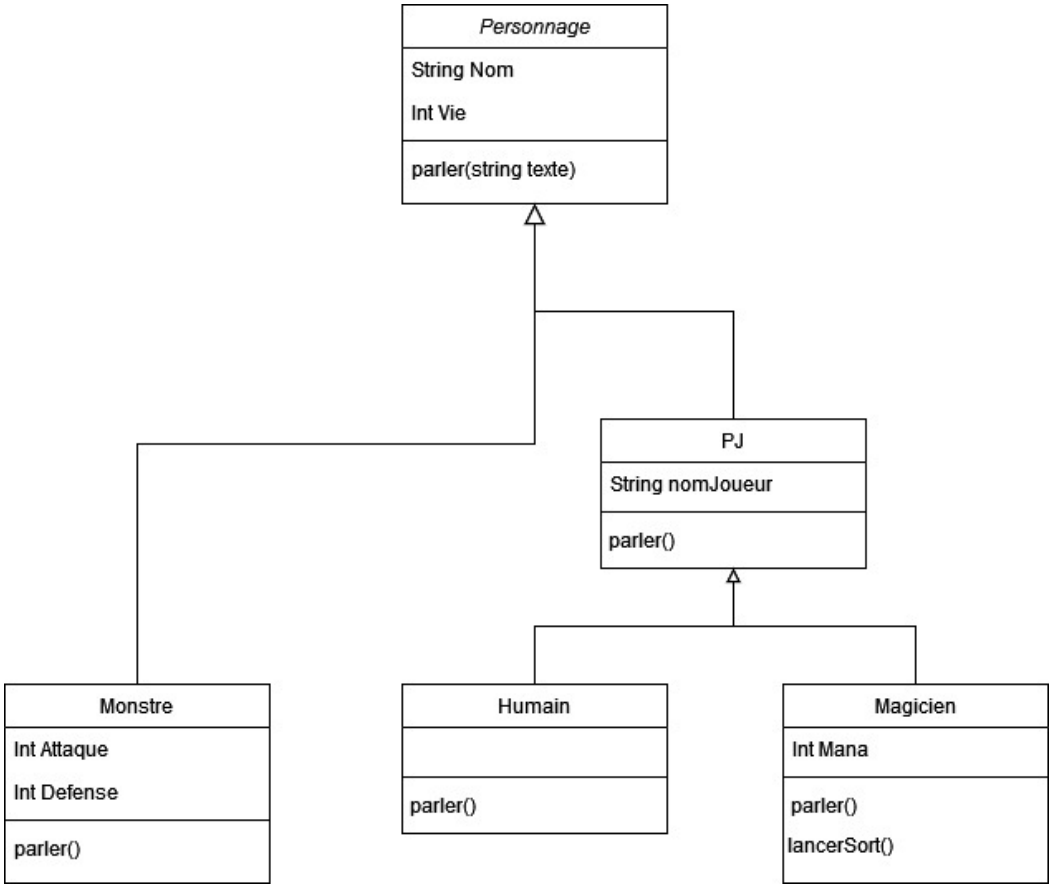
```
public String parler(){
    return super.parler() + ", mon nom est " +
        this.nom;
}

public String parler(String texte){
    return this.nom + " dis : " + texte ;
}
```

Suite de l'exemple du jeu de rôle

Pour notre jeu de rôle exemple, nous ajoutons la possibilité pour les joueurs de sélectionner entre un magicien et un humain (sans magie) et de mettre des monstres à affronter. Pour ne pas copier-coller de code, nous allons utiliser l'héritage :

- Une classe **Personnage** contiendra toutes les informations communes à tous les types de personnages.
- Une classe **PJ** (personnage joueur), fille de *Personnage*, contiendra les informations supplémentaires communes à *Magicien* et *Humain* (le nom du joueur et la méthode parler()).
- Une classe **Monstre**, fille de *Personnage*, contiendra les informations spécifiques aux monstres (attaque, défense et parler())

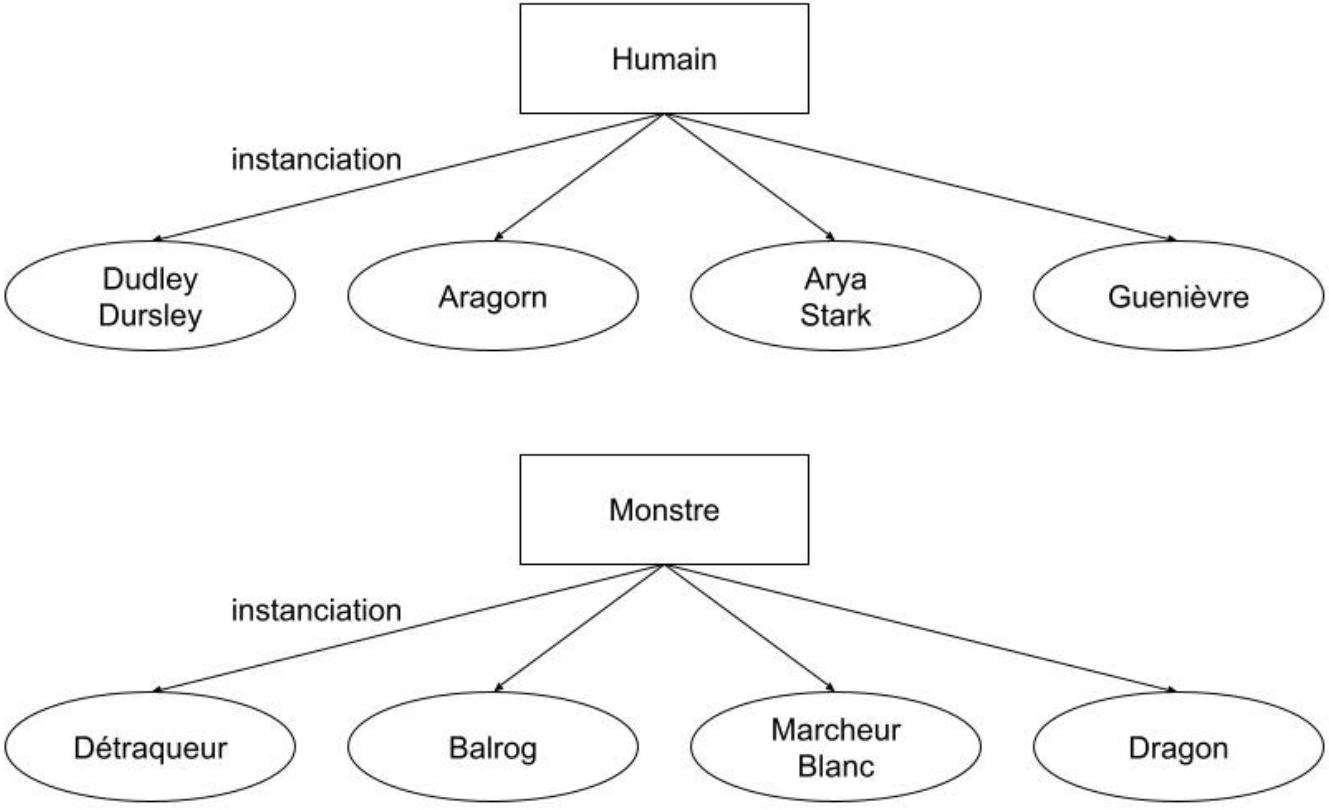
Représentation graphique de l'héritage dans l'exemple



Pour plus de clarté, les constructeurs, accesseurs et mutateurs n'ont pas été indiqués sur le schéma.

Différentes instances des nouvelles classes

Quelques exemples d'instances des nouvelles classes :



Plan

Rédaction d'une classe en Java

```
public class Exemple extends ClasseMere {
    //on ecrit les attributs
    visibilite Type1 nomAttribut1 =
        valeurInitiale1;
    /*  a noter
    que la valeur initiale
    est optionnelle */
    visibilite Type2 attribut2;

    //on ecrit les methodes
    visibilite typeRenvoye nomMethode(Type
        parametreEntree){
     ...
     return variableDuTypeARenvoyer;
    }
}
```

Exemple avec la classe magicien

```
public class Magicien extends PJ{
    private int mana;

    public Magicien(){
        super();
        this.mana = 10;
    }

    public Magicien(String nom, int vie, String
        nomJoueur, int mana){
        super(nom,vie,nomJoueur);
        this.mana = mana;
    }
    ...
}
```

Tous les exemples du cours sont à retrouver sur Moodle.

Rédaction du main

```
public class Test {
public static void main(String[] args) {
    Magicien hermione = new Magicien("Hermione␣
        Granger", 10, "Mathieu", 10);
    Monstre detraqueur = new Monstre();

    detraqueur.setNom("Philippe");
    detraqueur.setVie(2);

    System.out.println(detraqueur.sePresenter());
    System.out.println(detraqueur.parler());
    System.out.println(hermione.sePresenter());
    System.out.println(hermione.parler());
    System.out.println(hermione.parler("Voila␣un␣
        terrible␣detraqueur"));
}
}
```

Compilation et exécution

- Lors de l'exécution, la JVM va chercher la méthode **public static void main(String[] args)**. Cette méthode se trouve dans une classe à part qui peut avoir le nom que l'on souhaite.
- Avant d'exécuter, il faut préalablement compiler tous les fichiers .java : **javac monFichier.java**
- Pour compiler d'un coup tous les fichiers d'un dossier, on peut écrire : **javac *.java**
- Une fois tous les fichiers compilés, on peut exécuter notre main (disons qu'il se trouve dans Test.java) avec la commande : **java Test**
- Si tous les fichiers sont dans le même dossier, les classes seront directement reconnues d'un fichier à l'autre. Sinon, il faut passer par l'importation et l'organisation en package (nous verrons cela plus tard)

Résultat de l'exécution de l'exemple

```
Bonjour, je m'appelle Philippe
2
GROAR !
Bonjour, je m'appelle Hermione Granger et je suis joué par Mathieu
Bonjour, je suis un magicien
Hermione Granger dit : Voilà un terrible détraqueur
Bonjour, je m'appelle Dudley Dursley et je suis joué par Aurore
Dudley Dursley dit : oh, j'ai peur !
```

Le code complet menant à ce résultat est à retrouver sur moodle.