

Programmation 2 : premier cours

Arnaud Labourel arnaud.labourel@univ-amu.fr

19 septembre 2018



Bienvenue dans le cours de programmation 2

Volume horaire

- 10 séances de cours d'1h30 (15h au total)
- 22 séances de travaux pratiques de 2h (44h au total)

Évaluation

- un examen sur machine en janvier (60% de la note)
- un partiel sur papier le 10 décembre (20% de la note)
- des rendus de travaux pratiques en novembre (20% de la note)

Objectifs du cours

- Apprendre à programmer avec des objets
 - ▶ adopter le “penser objet”
 - ▶ connaître et savoir mettre en œuvre les concepts fondamentaux de la programmation objet
- Apprendre à coder proprement
- Connaître le langage Java
- Préparer aux cours de **Projet : initiation génie logiciel** du semestre 4 et de **Programmation et conception** du semestre 5

- Aix :
 - ▶ Arnaud Labourel (CM + TP 1)
 - ▶ Adrien Varet (TP 2)
- Luminy :
 - ▶ Arnaud Labourel (CM)
 - ▶ Pacôme Perrotin (1/2 TP 1)
 - ▶ Manon Scholivet (1/2 TP 1)
 - ▶ Raphaël Sturgis (TP 2)
 - ▶ François Hamonic (TP 3)

Positionnement du cours dans la licence informatique

Semestre 2

Programmation 1



Semestre 3

Programmation 2



Semestre 4

Projet : initiation
génie logiciel



Semestre 5

Programmation et
conception

Contenu du cours

S'initier à toutes les étapes du développement de logiciels

- 1 **Analyser** les besoins
- 2 **Spécifier** les comportements du programme
- 3 **Choisir** et éventuellement **concevoir** les solutions techniques
- 4 **Implémenter** le programme (coder)
- 5 **Vérifier** que le programme a le comportement spécifié (tester)
- 6 **Déployer** le programme dans son environnement, **fournir** une documentation dans le cas de bibliothèque
- 7 **Maintenir** le programme (corriger les bugs, ajouter des fonctionnalités)

Apprendre à programmer proprement

Un programme propre :

- respecte les attentes des utilisateurs
- est fiable
- peut évoluer facilement/rapidement
- est compréhensible par tous

Méthode pour programmer proprement

- nommer correctement les éléments du code
- écrire du code lisible (par un autre humain)
- relire et améliorer le code

Exemple de code mal écrit

```
public class Rec {
    int l, L;

    public Rec(int l, int L) {
        this.l = l;
        this.L = L;
    }

    public static int compte(List<Rec> r){
        for(int i = 0, int s = 0; i < r.size(); i++){
            if(r.get(i).l == r.get(i).L)
                s++;
        }
        return s;
    }
}
```

Après un petit peu de refactoring

```
public class Rectangle {
    int width, height;
    public Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }
    private boolean isSquare(){ return width == height; }
    static int countSquares(List<Rectangle> rectangles){
        int nbSquares = 0;
        for(Rectangle rectangle : rectangles)
            if(rectangle.isSquare())
                nbSquares++;
        return nbSquares;
    }
}
```

Le penser objet

Penser objet : décomposer le programme en objets

- Quels sont les objets nécessaires à la résolution du problème ?
⇒ décomposition du problème en objets
- À quels modèles des objets correspondent-il ?
⇒ Quelles sont les classes ?
- Quels sont les fonctionnalités/opérations dont on doit/veut pouvoir disposer sur ces objets ?
⇒ Quelles sont les méthodes des classes ?
- Quelle est la structure des données de l'objet ?
⇒ Quelles sont les attributs des classes ?

Exemple de problème

- un catalogue regroupe des articles, il permet de trouver un article à partir de sa référence.
- un article est caractérisé par un prix et une référence que l'on peut obtenir. On veut aussi pouvoir déterminer si un article est plus cher qu'un autre
- une commande est créée pour un client et un catalogue donnés, on peut ajouter des articles à une commande, accéder à la liste des articles commandés ainsi que prix total des articles et le montant des frais de port de la commande.
- un client peut créer une commande pour un catalogue et commander dans cette commande des articles à partir de leur références.

Classes du problèmes

- un **catalogue** regroupe des **articles**, il permet de trouver un article à partir de sa **référence**.
- un **article** est caractérisé par un prix et une référence que l'on peut obtenir. On veut aussi pouvoir déterminer si un article est plus cher qu'un autre
- une **commande** est créée pour un **client** et un **catalogue** donnés, on peut ajouter des **articles** à une commande, accéder à la liste des articles commandés ainsi que prix total des articles et le montant des frais de port de la commande.
- un **client** peut créer une **commande** pour un catalogue et commander dans cette commande des **articles** à partir de leur références.

- un catalogue regroupe des articles, il permet de **trouver** un article à partir de sa référence.
- un article est caractérisé par un prix et une référence que l'on **peut obtenir**. On veut aussi pouvoir **déterminer** si un article est plus cher qu'un autre
- une commande est créée pour un client et un catalogue donnés, on peut **ajouter** des articles à une commande, **accéder** à la liste des articles commandés ainsi que prix total des articles et le montant des frais de port de la commande.
- un client peut **créer** une commande pour un catalogue et **commander** dans cette commande des articles à partir de leur références.

Description d'un catalogue

un catalogue regroupe des articles, il permet de **trouver** un article à partir de sa référence.

Méthodes :

- Item getItem(String reference)

Description d'un article

un article est caractérisé par un prix et une référence que l'on **peut obtenir**. On veut aussi pouvoir **déterminer** si un article est plus cher qu'un autre

Méthodes :

- double getPrice()
- String getReference()
- boolean isMoreExpensiveThan(Item other)

Constructeurs et méthodes de la classe Order

Description d'une commande

une commande est **créée** pour un client et un catalogue donnés, on peut **ajouter** des articles à une commande, **accéder** à la liste des articles commandés ainsi que prix total des articles et le montant des frais de port de la commande.

Méthodes et constructeurs :

- `Order(Client client, Catalog catalog)` (constructeur)
- `void addItem(Item item)`
- `List<Item> allItems()`
- `double getTotalPriceOfItems()`
- `double getShippingCost()`
- `Client getClient()`
- `Catalog getCatalog()`

Description d'un client

un client peut **créer** une commande pour un catalogue et **commander** dans cette commande des articles à partir de leur références.

Méthodes :

- `Order createOrder(Catalog catalog)`
- `void orderItem(Order order, String reference)`

Vocabulaire programmation objet

Un **objet** :

- peut être **construit**
- est **structuré** : il est constitué d'un ensemble d'**attributs** (données de l'objet)
- possède un **état** : la valeur de ses attributs
- possède une **interface** : les opérations applicables appelées **méthodes**

Objet et référence

- Chaque objet est identifié par une adresse mémoire appelée **référence**
- C'est cette référence qui permet d'accéder à l'objet

Construire un objet

```
Classe objet = new Classe(arguments);
```

- Après l'appel au constructeur et l'affectation, la variable contient la référence de l'objet
- Sans appel au constructeur, la variable contient null (adresse pointant vers rien)

Accéder à un attribut d'un objet

```
objet.attribut = ...;
```

Appeler une méthode d'un objet

```
objet.méthode(arguments)
```

Attention au `null`

Si vous n'avez pas construit l'objet, tout accès à :

- une de ses méthodes
- un de ses attributs

générera une `NullPointerException` (davantage de détails sur les exceptions dans un cours ultérieur).

Règles d'or

- Toujours penser à construire les objets qu'on a besoin.
- Éviter si possible d'utiliser `null` dans son code.

Une **classe** (d'objet) définit des :

- **constructeurs** : des façons de construire/instancier les objets (**instances** de la classe)
- **attributs** (champs, propriétés ou données membres) : la structure des objets de la classe
- **méthodes** : le comportement des objets de la classe

Exemple de syntaxe de définition d'une classe

```
public NameOfTheClass{
    public final Type1 attribute1 = new Type1(arguments);
    private Type2 attribute2;

    public NameOfTheClass(arguments){
        // constructor code
    }

    public Type2 getAttribute2(){
        return this.attribute2;
    }
}
```

Déclaration d'attributs

Exemples **sans** initialisation

```
private int count;  
private final Color backgroundColor;  
private final String name;
```

Exemples **avec** initialisation

```
private int count = 0;  
private final Color backColor = new Color(0.5,0.2,0);  
private final String name = "John Doe";  
⇒ objets créés avec des valeurs de bases pour les attributs.
```

Mot-clés

- `private` : accessible que dans la classe
- `public` : accessible partout
- `final` : une seule affectation avant fin construction

Déclaration d'une méthode

```
visibilité + type de retour + nom +  
    (type paramètre 1 + nom paramètre 1, ...) + {  
    instructions  
}
```

Exemples

```
public int getCount() { ... }  
public void setCount(int newCount) { ... }  
public void launchMissile(GPSCoordinate coord) { ... }  
private Vector3D getAcceleration() { ... }
```

Le corps d'une méthode ou d'une classe est composé :

- d'*instructions*,
- de *structures de contrôle*, permettant de décider quelles instructions exécuter.

Remarques

- Sans structure de contrôles, les instructions sont exécutées une par une, dans le sens de lecture.
- On écrit généralement une instruction ou une structure de contrôle par ligne.
- On utilise l'*indentation* pour mettre en valeur les structures de contrôle.

Déclaration d'une variable

Une *variable* est un espace mémoire nommé pour stocker une valeur *momentanément*.

Portée : une variable (qui n'est pas un attribut) existe :

- depuis le moment où elle est déclarée
- jusqu'à la sortie du bloc { ... } où elle est définie.

Remarques

- chaque variable possède :
 - ▶ un type (qui peut correspondre à un type primitif, une classe ou une interface), et ne peut contenir que des valeurs de ce type.
 - ▶ une valeur (les variables non-affectées ont une valeur par défaut : 0, null, ...)

```
int count;  
Vector2D position;
```

L'affectation

L'*affectation* est l'opération consistant à mettre une valeur dans un espace mémoire défini par une variable (**l-value**) :

L'affectation est soumise aux contraintes de types: la valeur stockée doit être d'un type correspondant à celui déclaré.

```
count = 12;  
count = count + 4;  
position = new Vector(3, -2.5);
```

La création d'un nouvel objet se fait avec le mot-clé `new`, qui déclenche :

- 1 l'allocation d'un espace mémoire pour contenir le nouvel objet,
- 2 l'initialisation des propriétés de cet objet avec les valeur de bases,
- 3 l'appel au constructeur de l'objet,
- 4 le retour de la *référence* (adresse mémoire) de l'objet par l'instruction

```
new Vector(3,-2.5);  
new Spaceship();  
new ArrayList<Integer>();
```


Le mot-clé return

Une méthode peut retourner une valeur lorsqu'elle a terminé son travail, en utilisant le mot-clé `return`.

Lors de l'évaluation de l'instruction `return`, l'expression après le `return` (le résultat) est *d'abord* évaluée (si elle existe), *ensuite* l'exécution de la méthode est interrompue.

Le résultat est retourné au niveau de l'instruction ayant fait l'appel de méthode.

Exemples d'utilisation de return

```
return;  
return 42;  
return new Vector(-5, 2.333);
```

Les valeurs sont définies par des *expressions* :

- des littéraux : 0.01, -42, "toto", ...
- combinaison d'expressions par des opérateurs +, *, /, -
- variables et attributs,
- appels de méthode (ayant un type de retour autre que void) ou de constructeurs

Les expressions apparaissent :

- en membre droit d'une affectation,
- en paramètre d'un appel de méthode, d'un constructeur,
- après le mot-clé return

À retenir absolument

Une **classe** (d'objet) définit des :

- **constructeurs** : des façons de construire/instancier les objets (**instances** de la classe)
- **attributs** (champs, propriétés ou données membres) : la structure des objets de la classe
- **méthodes** : le comportement des objets de la classe

Syntaxe de définition d'une classe

```
public NameOfTheClass{
    public final Type1 attribute1;
    private Type2 attribute2;

    public NameOfTheClass(...){
        // constructor code
    }

    public getAttribute2(){
        return this.attribute2;
    }
}
```