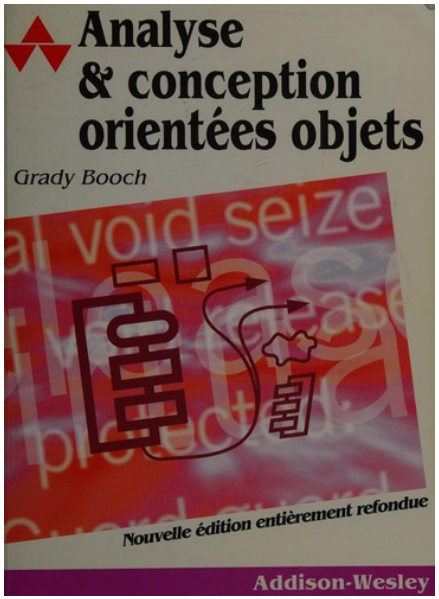
Programmation orientée objet – Application en Kotlin

Classes et objets : notions importantes et définitions

LPRO Dream

Sylvie TROUILHET – www.irit.fr/~Sylvie.Trouilhet

Bibliographie



Diaporama du cours de Nicolas Garric

Grady Booch - Analyse et conception orientées objets 1994
Addison-Wesley France

Site de Kotlin - https://kotlinlang.org/docs/home.html

Programmation orientée objet – Application en Kotlin

PLAN

Partie 2 : classes et objets

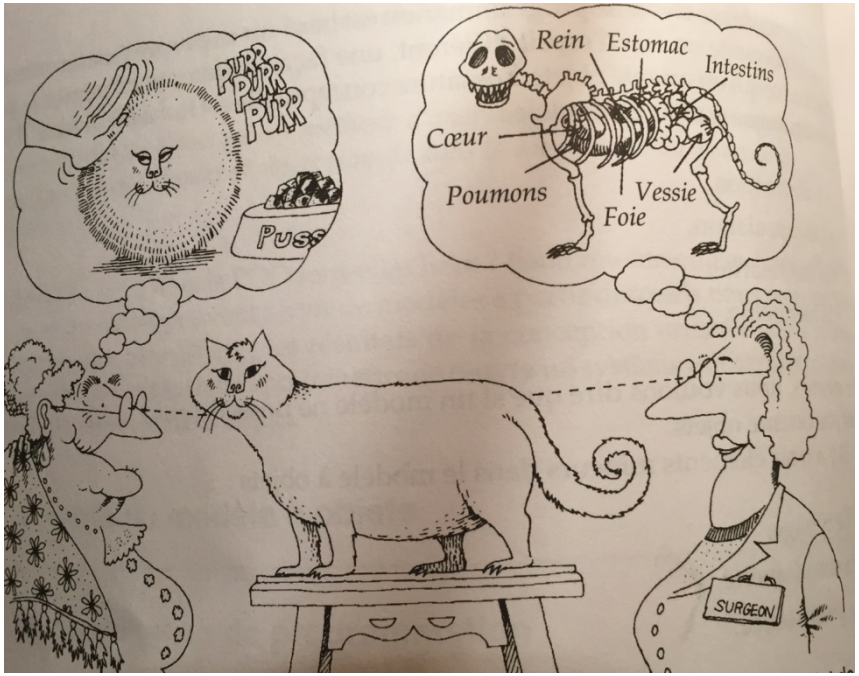
Partie 2 : notions majeures de la programmation orientée objet

4 éléments majeurs dans ce modèle de programmation :

- L’abstraction
- L’encapsulation
- La modularité
- La hiérarchie

Abstraction

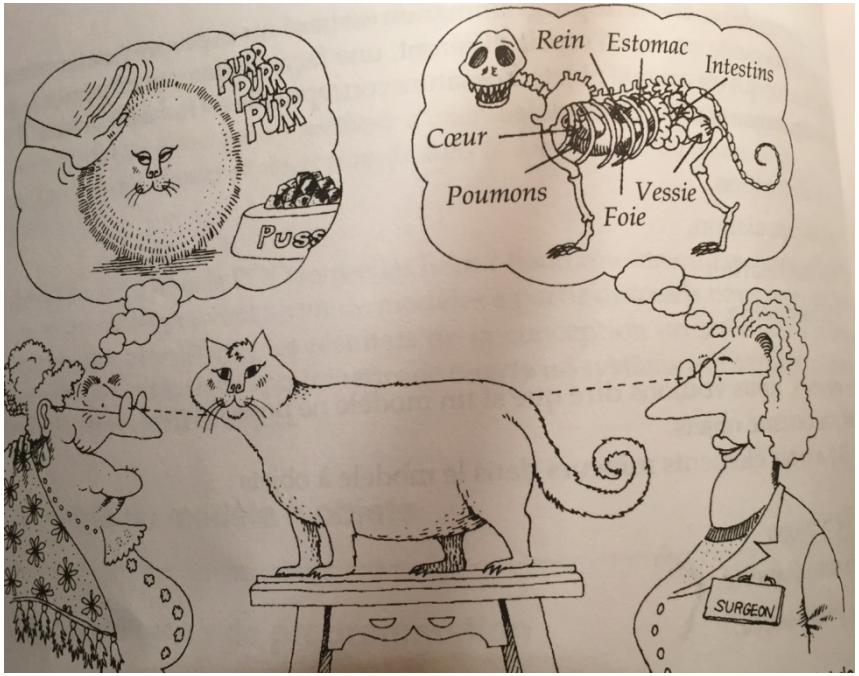
Une abstraction fait ressortir les caractéristiques essentielles d'un objet qui le distinguent de tous les autres.
Elle procure des frontières conceptuelles adaptées au point de vue de l'observateur.



→ l'abstraction se concentre sur la vue externe de l'objet

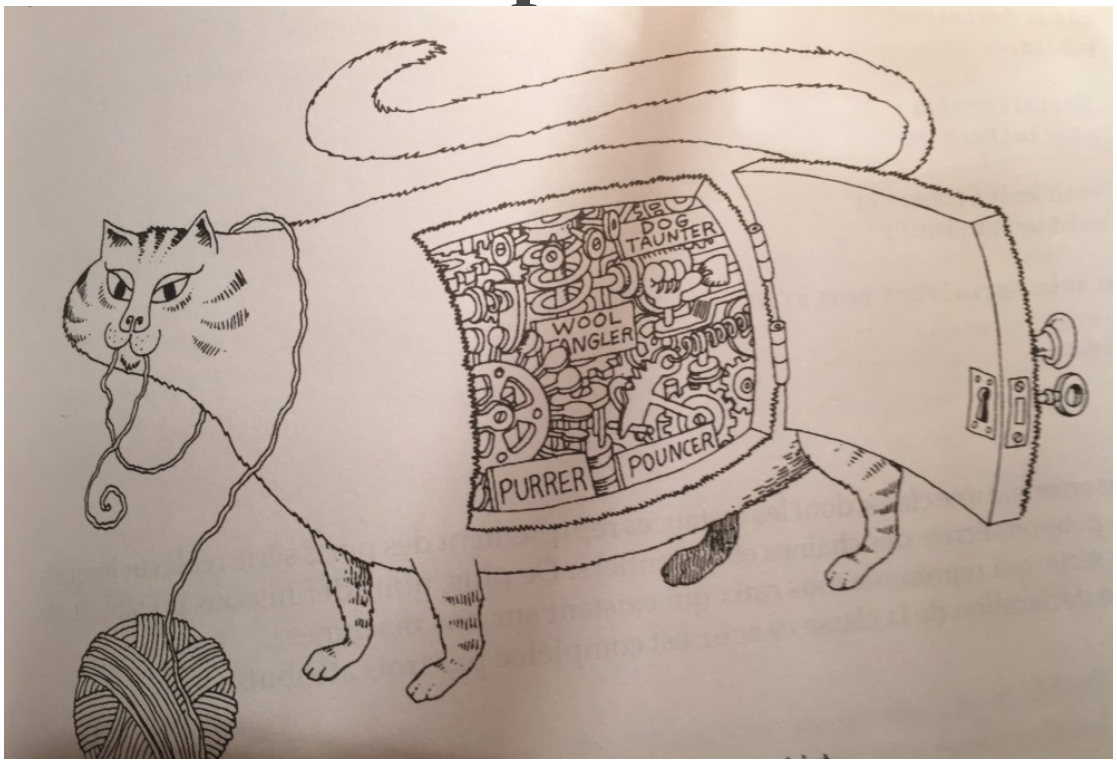
Abstraction

La vue externe de l'objet donne les services qu'il offre à ses clients (objets qui l'utilise).



Elle définit un **contrat** : ce sont les responsabilités de l'objet, le comportement que l'on peut attendre de lui.
Chaque opération associée à un objet possède des préconditions et des postconditions qui ne doivent pas être violées

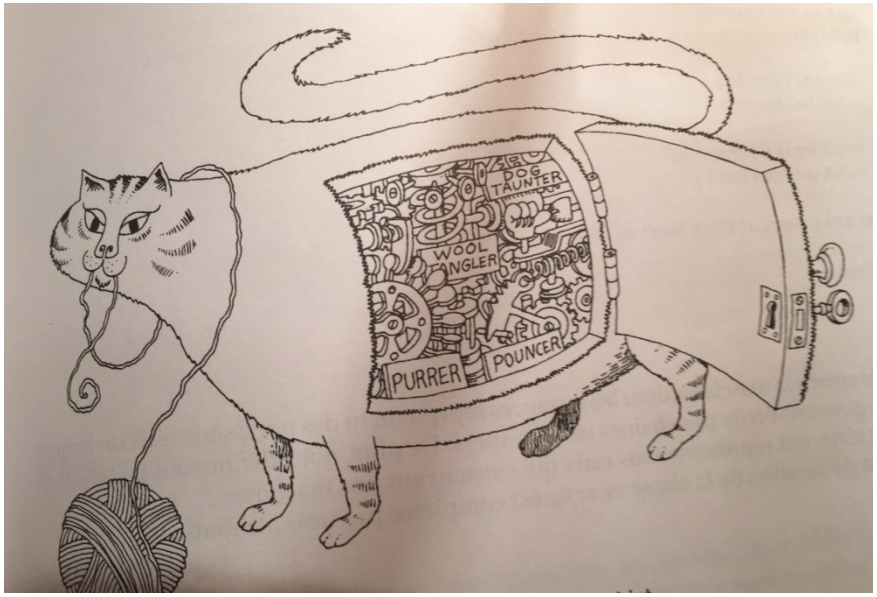
Encapsulation



→ L'encapsulation s'attache à la mise en œuvre du comportement de l'objet.
Elle permet de séparer l'interface contractuelle de la mise en œuvre de l'objet.

Encapsulation

La mise en œuvre renferme la représentation de l'abstraction ainsi que les mécanismes qui réalisent le comportement désiré.



L'encapsulation permet d'effectuer des modifications du programme en toute sécurité et avec un effort moindre.

Encapsulation

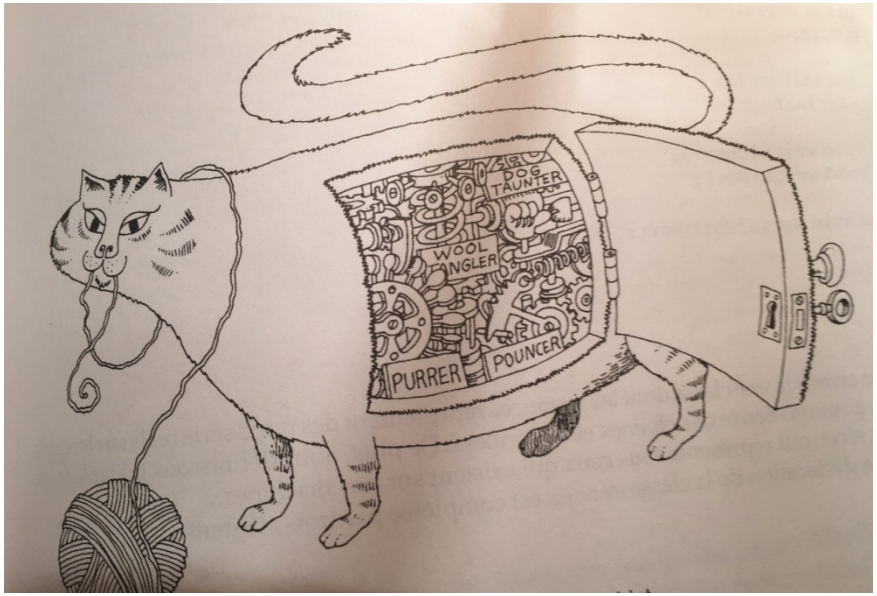
Exemple :
Soit l’objet angle ayant comme valeur 90 degrés (mise en œuvre de l’objet).

On l’encapsule en ajoutant une interface : obtenirValeurAngle() qui retourne la valeur de l’angle (ici 90°).

Cette séparation permet de changer la mise en œuvre (passer en radians) sans impact sur les clients de l’objet : la valeur de l’angle est PI/2 radians et obtenirValeurAngle() retourne 90 * PI/180

Masquage des données

L’encapsulation est obtenue grâce au masquage des données



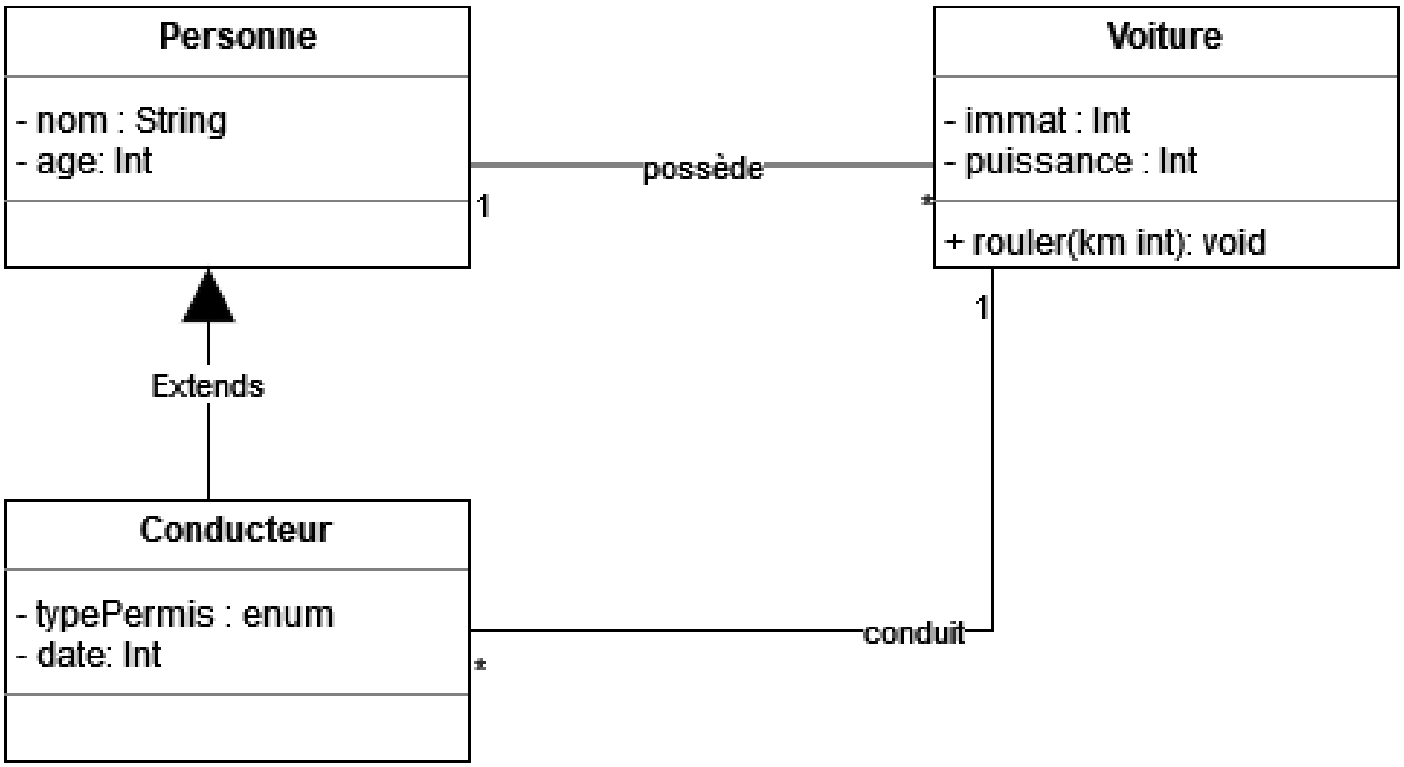
Le masquage des données consiste à **cacher** tous les secrets d’un objet lorsqu’ils ne contribuent pas à ses caractéristiques essentielles.

→ Cacher la structure de l’objet ainsi que le codage de ses méthodes

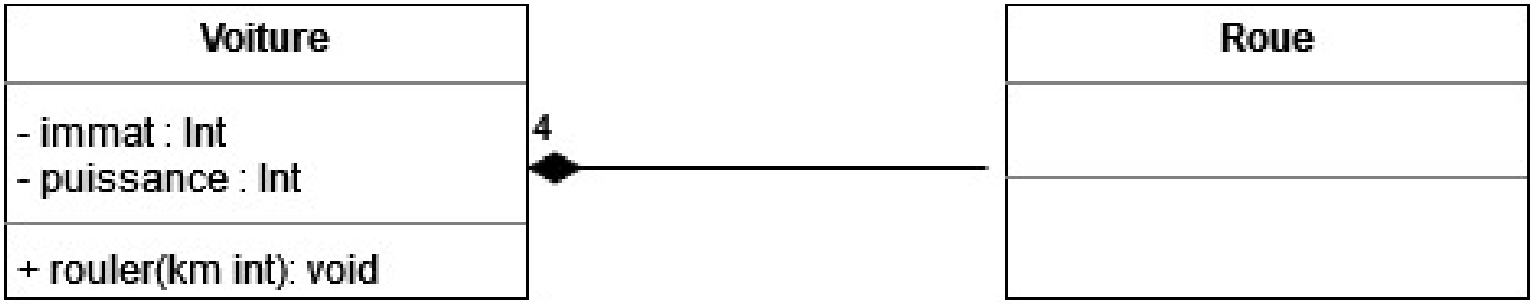
Modularité et hiérarchie

La modularité est la propriété d’un système qui a été décomposé en un ensemble de modules cohérents et **faiblement couplés**.
La hiérarchie est un **ordonnancement des abstractions**.

Modularité et hiérarchie



Modularité et hiérarchie



composition

Partie 2 : objet et classe

Objet et classe : définitions
Création d’un objet
Composition d’une classe

Objet

- Un objet a un **état**, il présente un **comportement** bien défini, et il a une **identité** unique. La structure et le comportement d'objets similaires sont définis dans leur classe commune.
  L'état englobe toutes les propriétés et les valeurs courantes de ces propriétés.
  Le comportement est la façon dont l'objet agit et réagit, en terme de changement d'état et de transmission de messages.

- Les termes instance et objet sont interchangeables.

Objet

- Un objet est composé :

  - d'**attributs** : ce sont des variables qui décrivent les propriétés de l'objet, et dont la durée de vie est égale à celle de l'objet
  - de **méthodes** : ce sont des fonctions associées à l'objet et qui utilisent et/ou modifient les attributs de l'objet

- Les méthodes publiques définissent l'interface de l'objet

- L'accès aux attributs de l'objet ne peut être fait qu'au travers de méthodes.

Classe

- Une classe est un ensemble d'objets qui partagent une structure commune et un comportement commun

- C'est le modèle qui permet de créer des objets. À partir d'une classe on peut créer autant d'objets que nécessaires.

Écriture d’une classe

Mot-clé class
Corps de la classe : contient les données et la définition des comportements
Kotlin : par défaut, toute méthode ou attribut sans modificateur d’accès est public c’est-à-dire qu’il est accessible à partir de n’importe quel fichier.

Exemple :

Kotlin

```
public class Chat {
    public val nom: String ="Garfield"
    public fun miauler() : Unit {
        println("miaou")
    }
}
```

Écriture d'une classe

Mot-clé class
Corps de la classe : contient les données et la définition des comportements
Kotlin : par défaut, toute méthode ou attribut sans modificateur d'accès est public c'est-à-dire qu'il est accessible à partir de n'importe quel fichier.

Exemple :

Kotlin

```
class Chat {
  val nom="Garfield"
  fun miauler() {
        println("miaou")
    }
}
```

Création d'un objet

On créé un objet à partir d'une classe en appelant son constructeur

Exemple :

Kotlin

```
val grosMinet:Chat = Chat()
```

grosMinet est une référence (adresse) vers l'objet de type Chat

Méthode constructeur

- Quand on instancie une classe, l'objet créé doit être initialisé, c'est-à-dire que ses attributs doivent avoir des valeurs

- Un constructeur est une méthode qui crée un objet et initiialise son état

  → décrit les opérations à effectuer à l'instanciation de l'objet

Exemple de constructeur

Kotlin

```
class Chat  public constructor(_nom: String) {
 val nom: String = _nom
}
```

→ Définition du constructeur principal (dans l'entête, après **class Nom -** ne contient pas de code)

→ Initialise l'attribut nom avec la valeur passée en paramètre

→ Convention : _nom = paramètre/variable à usage unique

Exemple d'écriture d'une classe

Kotlin

```
class Chat public constructor(_nom:String){
    val nom=_nom
    fun miauler() {
        println("miaou")
    }
}
```

Ou :

Kotlin

```
class Chat(_nom:String){
    val nom=_nom
    fun miauler() {
        println("miaou")
    }
}
```

Exemple d'écriture d'une classe

Kotlin

```
class Chat(_nom:String){
    val nom=_nom
    fun miauler() {
        println("miaou")
    }
}
```

Kotlin

```
fun main() {
    val c = Chat("Garfield")
    c.miauler()
    println(c.nom)
}
```

Partie 2 : modificateurs d’accès

private / public
Getter et setter personnalisés

Modificateurs public , private

- Un modificateur d'accès contrôle la manipulation d'un élément de classe

Kotlin
par défaut : public

- Une méthode ou un attribut public est accessible partout où la classe est visible

  Une méthode ou un attribut private n'est pas accessible de l'extérieur.

  Mise en œuvre du masquage : les méthodes internes non utiles à l'utilisateur de la classe sont private - le s attributs ne sont pas modifiés de façon incontrôlée

Getter personnalisé

Kotlin

```
class Chat constructor(_nom:String){
    val nom=_nom
    get() = field.toLowerCase()
}
```

field dénote le champ de sauvegarde créé par Kotlin pour l'attribut.
Il n'est accessible qu'à l'intérieur du getter.
L'appelant ne voit jamais ce champ directement mais seulement la donnée retournée par le getter.

NB : setter inutile pour une variable non mutable (val).

Setter personnalisé

Kotlin

```
class Chat constructor(_nom:String){
    val nom=_nom
    get() = field.replaceFirstChar()

    var age=0
    get() = field
    set(_age) {if(_age>field) field=_age}

}
```

Exercice

Écrire la classe Personne avec :

- 2 attributs nom et âge
- Un constructeur qui initialise le nom avec la chaîne passée en paramètre et l'âge à 0
- la méthode presenteToi() qui affiche le nom et l'âge de la personne

Dans une fonction main(), créer une personne avec votre nom et votre âge. Écrire la fonction fêterAnniversaire(p: Personne) qui ajoute 1 an à la personne.
L'appeler dans le main() pour la personne que vous avez créée.

Partie 2 : compléments

this
Polymorphisme paramétrique : surcharge de méthodes

Mot-clé this

this désigne l'objet dans lequel on se trouve

Attributs vs variables locales

durée de vie de l'attribut = durée de vie de l'objet

durée de vie de la variable locale = temps de l'exécution de la méthode qui la contient

Constructeur secondaire

Possibilité d'avoir des constructeurs secondaires. Un constructeur secondaire doit déléguer au constructeur principal avec **:this()**

Kotlin

```
class Chat constructor(_nom:String){
…

    constructor(_nom:String, _poids:
Double): this(_nom) {
        poids=_poids;
    }
…
}
```

Surcharge de méthodes

Possibilité d'avoir des méthodes de même nom si elles ont des paramètres différents en nombre et/ou en type

Kotlin

```
class Chat constructor(_nom:String){
    …
    fun miauler() : Unit {
        println("miaou")
    }
    fun miauler(cri:String) : Unit {
        println("miaou $cri")
    }
}
```

Exercice

Ajouter un constructeur secondaire à la classe Personne