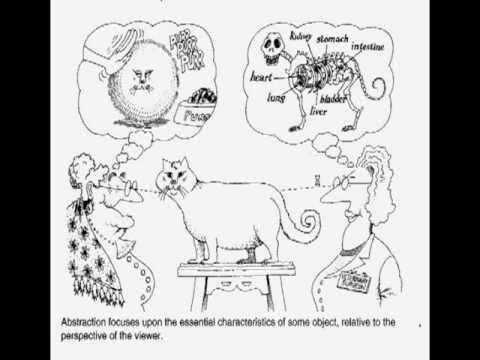
IUT d’Orsay - DUT Informatique - Jean-Claude MARTIN – JEAN-CLAUDE.MARTIN@U-PSUD.FR

Programmation Orientée Objet (POO)  
COURS 5 : Classes abstraites



Rappel sur l’héritage et le polymorphisme

=> à tester sous Eclipse !

**Mere m = new Mere ();**

**Fille f = new Fille ();**

**m = f;**

**m.attributMere = 0 ;**

**=>**

**Ok car attributMere est hérité donc le compilateur**

**a alloué de**

**la place pour cet attribut lors du new Fille()**

**f = m ; // erreur de compilation**

**f.attributFille ;**

**=>**

**ERREUR DE COMPILATION**

**l’attribut attributFille n’a pas été**

**prévu lors de l’allocation du new Mere()**

Contexte : quel est le problème ?

* En entreprise, un projet est généralement développé en équipe et parfois sur plusieurs mois / années avec maintenance.
* Un développeur doit donc prévoir que ses programmes seront utilisés par quelqu'un d'autre que lui-même.
* C'est par exemple le cas du JDK (plus de 1000 classes qui sont utilisées par XXX développeurs Java).
* Besoin d’extensibilité, de flexibilité et de réutilisabilité
* **=> proposition de cadres / frameworks (par exemple les Collections) qui comportent notamment des classes abstraites.**

|  |
| --- |
| *En* [*programmation informatique*](https://fr.wikipedia.org/wiki/Programmation_informatique)*, un framework ou structure logicielle est un ensemble cohérent de* [*composants logiciels*](https://fr.wikipedia.org/wiki/Composant_logiciel) *structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d’une partie d'un* [*logiciel*](https://fr.wikipedia.org/wiki/Logiciel) *(*[*architecture*](https://fr.wikipedia.org/wiki/Architecture_informatique)*). Un framework se distingue d'une simple* [*bibliothèque logicielle*](https://fr.wikipedia.org/wiki/Biblioth%C3%A8que_logicielle) *principalement par :*   * *son caractère générique, faiblement spécialisé, contrairement à certaines bibliothèques ; un framework peut à ce titre être constitué de plusieurs bibliothèques chacune spécialisée dans un domaine. Un framework peut néanmoins être spécialisé, sur un* [*langage*](https://fr.wikipedia.org/wiki/Langage_informatique) *particulier, une* [*plateforme*](https://fr.wikipedia.org/wiki/Plate-forme_%28informatique%29) *spécifique, un domaine particulier :* [*reporting*](https://fr.wikipedia.org/wiki/Reporting)*,* [*mapping*](https://fr.wikipedia.org/wiki/Data_mapping)*, etc. ;* * *le cadre de travail (traduction littérale de l'*[*anglais*](https://fr.wikipedia.org/wiki/Anglais) *: framework) qu'il impose de par sa construction même, guidant l'*[*architecture logicielle*](https://fr.wikipedia.org/wiki/Architecture_logicielle) *voire conduisant le développeur à respecter certains* [*patterns*](https://fr.wikipedia.org/wiki/Pattern#Informatique) *; les bibliothèques le constituant sont alors organisées selon le même* [*paradigme*](https://fr.wikipedia.org/wiki/Paradigme)*.* |

* Principe
  + Encapsulation = séparation interface et implémentation
  + Analogie avec un iceberg

Schéma

* + Interface = spécification = partie visible par les développeurs qui vont utiliser la classe
  + Implémentation = choix de programmation (attributs, corps et instructions des méthodes)
  + Pour une même spécification, on peut avoir plusieurs implémentations possibles
  + La séparation entre interface et implémentation permet de s’abstraire des détails de fonctionnement de la classe
  + Le client (utilisateur de la classe encapsulée) ne manipule que L’INTERFACE il n’a pas accès aux IMPLÉMENTATIONS ; il peut se focaliser sur une programmation de plus haut niveau (meilleure lisibilité)
  + Interface = contrat entre le codeur de la classe et les développeurs
  + Interface = ce que fait la classe ≠ comment elle le fait

Classes abstraites

Permet d’aller progressivement vers les interfaces / spécifications

Définition

Une classe abstraite est une classe qui ne peut pas être instanciée

Utilité

* Dans une hiérarchie de classes, on veut parfois pouvoir définir un comportement général qui sera suivi ou redéfini par les sous-classes sans que ce comportement général donne lieu à des instances
* **=> il y a des classes que l’on ne veut pas qu’un programmeur puisse implémenter**
* Exemple :
  + Une classe ………………………………………………………….
  + Une classe ………………………………………………………….
  + Une classe ………………………………………………………….
  + => ………………………………………………………….………………………………………………………….

SCHEMA « Niveaux d’abstraction »

Programmation et utilisation

Vous aurez à

* Programmer vous même vos classes abstraites
* Utiliser les classes abstraites du JDK

Exemple du JDK

* Classe abstraite Component
  + superclasse des objets qui ont une représentation graphique : Button, Scrollbar, Window (Cf Javadoc)
  + méthodes : getSize, getX, hasFocus, isVisible, paint, setBackground, setForeground, set Font
* Classe abstraite AbstractSet
  + sous-classes concrètes : TreeSet, HashSet, …

Déclaration

Mot clé abstract devant l’entête de la classe

Compilation

Le compilateur n'accepte pas de créer des objets instance d'une classe abstraite.

**public** **abstract** **class** Vehicule {

**int** annee ;

}

**class** Circuit {

**public** **static** **void** main (String args[]) {

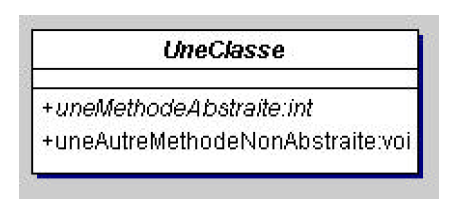
Vehicule v ; // ………………………………………………………….!

v = **new** Vehicule(); //………………………………………………………….………………………………………………………….………………………………………………………….………………………………………………………….………………………………………………………….

}

}

* Une classe abstraite peut avoir des attributs et des méthodes   
  (qui seront héritées par ses sous-classes)
* Une classe abstraite comporte **souvent** 1 ou pls méthodes abstraites (mais ce n’est pas obligatoire).
* Une classe abstraite peut avoir des constructeurs   
  (qui seront appelés par les constructeurs des sous-classes concrètes).
* UML : Dans les diagrammes de classe, les classes et méthodes abstraites sont en italique.



Méthodes abstraites

Définitions

* Une méthode abstraite est définie uniquement par son ENTÊTE
* Elle ne comporte pas de corps / d’instruction.

Déclaration

* mot-clé abstract devant la méthode

Définitions

* Une classe abstraite comporte généralement au moins une méthode abstract.
* Une classe abstraite peut comporter aussi des méthodes implémentées (concrètes) représentant une partie de l’implémentation.
* Une classe qui comporte une méthode **abstraite** doit être **abstraite**.
* Implémenter une méthode : donner un corps / des instructions à la méthode

Utilisation

* Une classe abstraite peut être dérivée en une (ou plusieurs) sous-classe(s) :
* concrète (non abstraite) si toutes les méthodes abstraites sont implémentées dans la sous-classe
* abstraite si une ou plusieurs méthodes ne sont pas implémentées par la sous-classe

Utilité

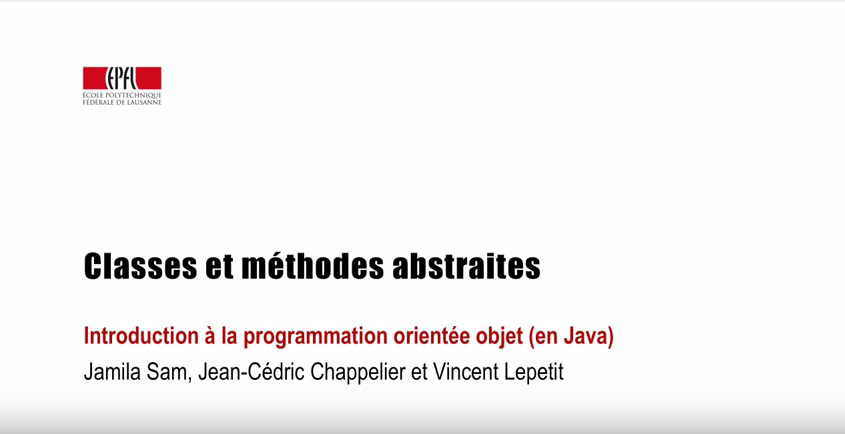
* Une classe abstraite C permet à son développeur :
  + de fournir à d'autres développeurs une partie de l'implémentation de la classe C (ATTRIBUTS, CORPS DE CERTAINES MÉTHODES)
  + d'imposer aux autres développeurs d'implémenter certaines méthodes s'ils veulent pouvoir utiliser la classe C (les méthodes de C qui sont ABSTRAITES) **pour créer une classe concrète**
  + de laisser aux autres développeurs la manière d'implémenter le reste de la classe (nouvelles méthodes définies et implémentées dans LES SOUS CLASSES DE LA CLASSE C)
* On parle parfois de méthode *différée* ou *retardée*

Video

<https://www.youtube.com/watch?v=sZ3UecFEi8c&list=PLZbs1ERZ-TGXyZw164Y_Bye3SudSZB_Ry>

Début : rappels sur le polymorphisme

Classes abstraites à 3:28



Exemple

La classe abstraite Animal (dans un jeu de simulation de clinique vétérinaire par exemple).

Méthode crier () qui affiche un message indiquant comment l’animal crie

public abstract class Animal {

// PARTIE IMPLEMENTEE

double poids ;

boolean estDeshydrate ;

void boire () {

estDeshydrate = false ;

}

// PARTIE QUI RESTE A IMPLEMENTER

abstract void crier()

}

Pourquoi une classe abstraite ?

..................................................................................

..................................................................................

Sous-classe concrètes / non abstraites : Chat, Chien

**class** Chat **extends** Animal {

// Si on n'implémente pas la méthode abstraite :

……………………………………………………….………………………………………………………….………………………………………………………….………………………………………………………….………………………………………………………….

// Si on l'implémente :

..................................................................................

..................................................................................

**void** crier() {

System.*out*.println ("Miaou");

}

// La classe peut contenir des méthodes spécifiques

**void** ronronner () {

System.*out*.println ("Ron ron ron ron");

}

}

Si les sous-classes concrètes ajoutent des méthodes implémentées, il peut y avoir besoin de faire un cast pour les appeler :

**class** Veterinaire {

**public** **static** **void** main (String args []) {

Animal a ;

a = **new** Chat (); // …Permis grâce ………………………………………………... ………….

a.crier(); // crier est une méthode ……………………………………………..

// la résolution d'appel de méthode se fait …………………………………………....

// C'est la méthode crier() de la classe …………………………………………...qui est appelée

a.ronronner(); // ERREUR

..................................................................................

..................................................................................

((Chat) a).ronronner(); // Ok grace au .................................

}

}

SCHEMA POLYMORPHISME

Exemple de répartition du travail : spécification / implémentation grâce aux classes abstraites

Intérêt des classes abstraites

abstract class X {

public abstract void f () ; // f n’est pas encore implémentée

}

Dans une classe quelconque :

void algo (X x) {

x.f() ; // Appel correct accepté à la compilation car

// on est sûr que tout objet instance d’une classe dérivée de X

// implémentera une méthode f() qui pourra être appelée

}

Exemple (cf feuille d’exercice) :

* M. Pasquier est chef de projet et développeur.
* Le but du projet est de développer un éditeur de dessin sur SmartPhone en Java.
* Il ne peut pas tout développer tout seul et doit collaborer avec M. Dupont.

Répartition des tâches

* M. Pasquier va prendre en charge la partie affichage graphique et stockage en mémoire de toutes les formes graphiques dans un tableau. Il devra notamment écrire un programme affichant des statistiques avec toutes les formes crées classées par surface décroissante.
* M. Dupont va prendre en charge la partie implémentation de différentes formes graphiques : des cercles, des rectangles ... Cette liste pourra évoluer au fur et à mesure du projet.

Exemple avec un tableau de références à une classe abstraite

**abstract** **class** Affichable {

**abstract** **public** **void** affiche() ;

}

**class** Entier **extends** Affichable {

**private** **int** valeur ;

**public** Entier (**int** n) {

valeur = n ;

}

**public** **void** affiche() {

System.*out*.println ("Je suis un entier de valeur " + valeur) ;

}

}

**class** Flottant **extends** Affichable {

**private** **float** valeur ;

**public** Flottant (**float** x) {

valeur = x ;

}

**public** **void** affiche() {

System.*out*.println ("Je suis un flottant de valeur " + valeur) ;

}

}

**public** **class** Tabhet3 {

**public** **static** **void** main (String[] args) {

Affichable [] tab ;

tab = **new** Affichable [3] ;

tab [0] = **new** Entier (25) ;

tab [1] = **new** Flottant (1.25f);

tab [2] = **new** Entier (42) ;

**int** i ;

**for** (i=0 ; i<3 ; i++)

tab[i].affiche() ; // Appel d'une méthode ……………………………………………………

}

}