

Algorithmique « avancée »

CM1 : Algorithmique « avancée » et efficacité

Mickaël Martin Nevot

V1.0.0



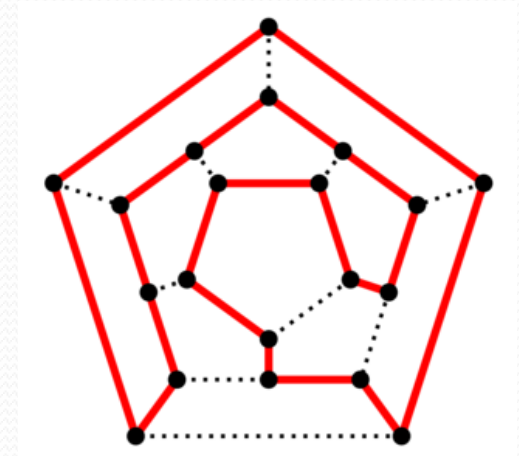
Cette œuvre de [Mickaël Martin Nevot](#) est mise à disposition selon les termes de la [licence Creative Commons Attribution – Pas d'Utilisation Commerciale – Partage à l'Identique 3.0 non transposé](#).

Algorithmique « avancée »

- I. Présentation du cours
- II. Efficacité
- III. Tris
- IV. Algorithmique "avancée"
- V. Présentation de l'APP

Notions de base

- **Terminaison :**
 - Assurance de finir en temps fini
- **Correction :**
 - Garantie d'une solution correcte
 - Solution au problème posé
- **Complétude :**
 - Proposition de solutions sur un espace de problème donné
- **Déterminisme :**
 - Se comporte de façon prévisible (un ensemble de données particulier produira toujours le même résultat)



Programmation modulaire


- Regroupement de codes sources en **unités de travail logiques**
- **Encapsulation**
- **Réutilisabilité** et partage du code facilités
- Facilitation de réalisation de bibliothèques
- **Généricité** possible



Programmation par contrat

- Paradigme
- But principal : **réduire le nombre de bogues**
- Précise les **responsabilités** entre le client et le fournisseur
- **Pas de vérification** de validité des règles
- **Assertions** :
 - Énoncé considéré ou présenté comme vrai
 - Écrites dans le code source en commentaires
 - Donnent la sémantique du programme

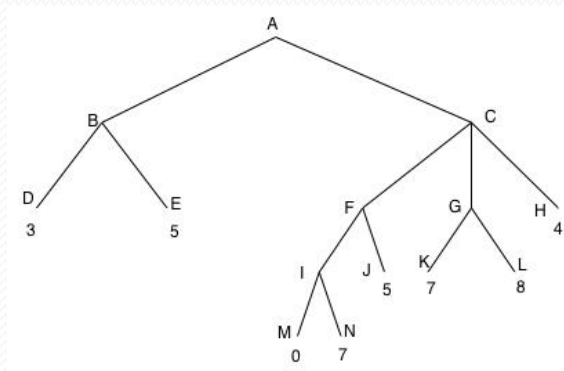
Type d'assertions

- **Précondition** :  Doit garantir un traitement possible et sans erreur
 - Doit être vérifiée **avant** un traitement par le **client**
- **Postcondition** :
 - Doit être garantie **après** un traitement par le **fournisseur**
- **Invariant** :
 - Doit être **toujours vrai**
(durant toute / une partie d'une application)

```
/**  
 * Fonction calculant la racine carrée de la valeur de x.  
 *  
 * Précondition :  $x \geq 0$ .  
 * Postcondition : résultat  $\geq 0$  et si  $x \neq 1$  alors résultat  $\neq x$ .  
 */  
public int sqrt() { ... }
```

Heuristiques

- Cas d'utilisation :
 - **Complexité trop grande**
 - Impossibilité d'obtenir un résultat en temps raisonnable
- Rechercher la solution la plus proche possible d'une solution optimale par essais successifs
- Pas d'exhaustivité possible : il faut faire des **choix** !
- **Choix** généralement très dépendant du problème

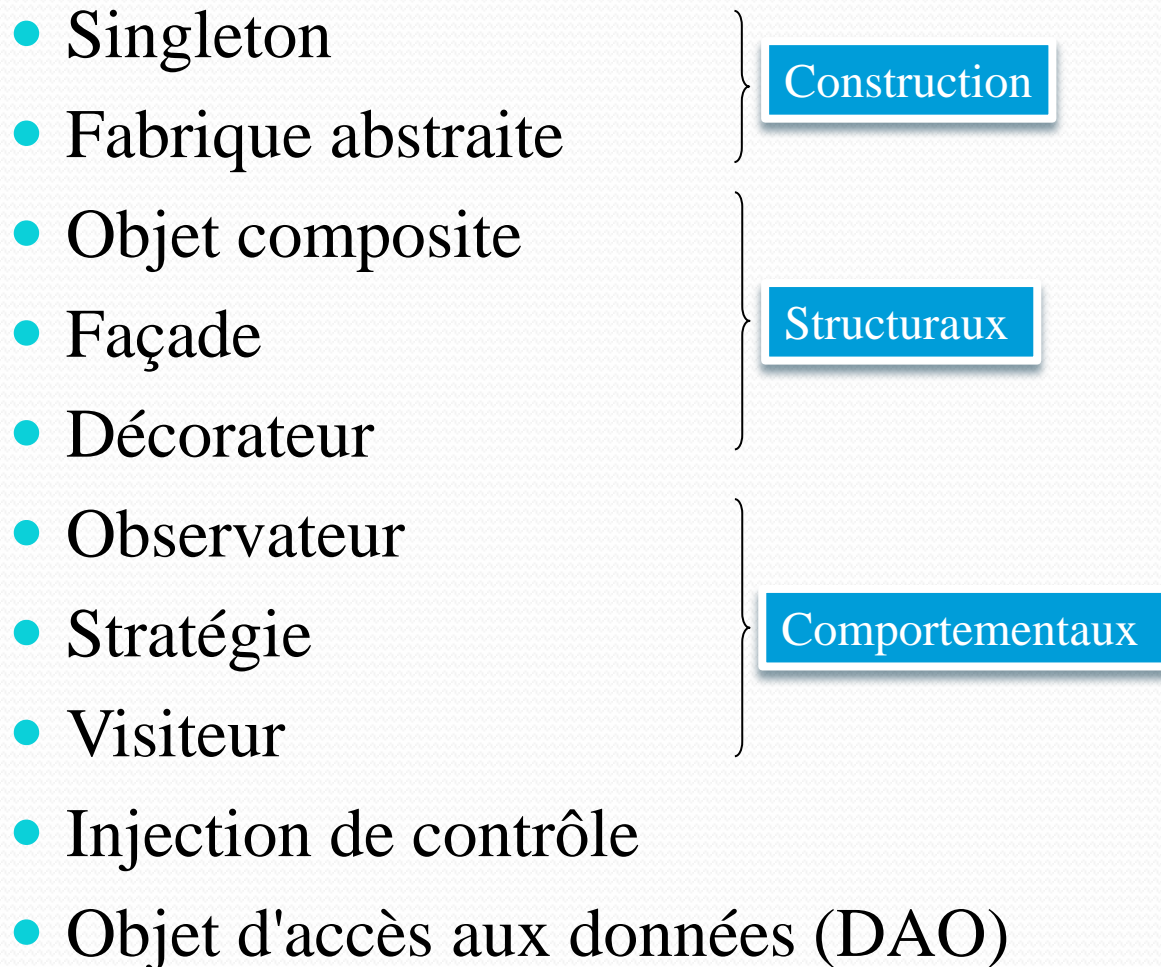


Cela s'appelle une heuristique

Modèles de conception

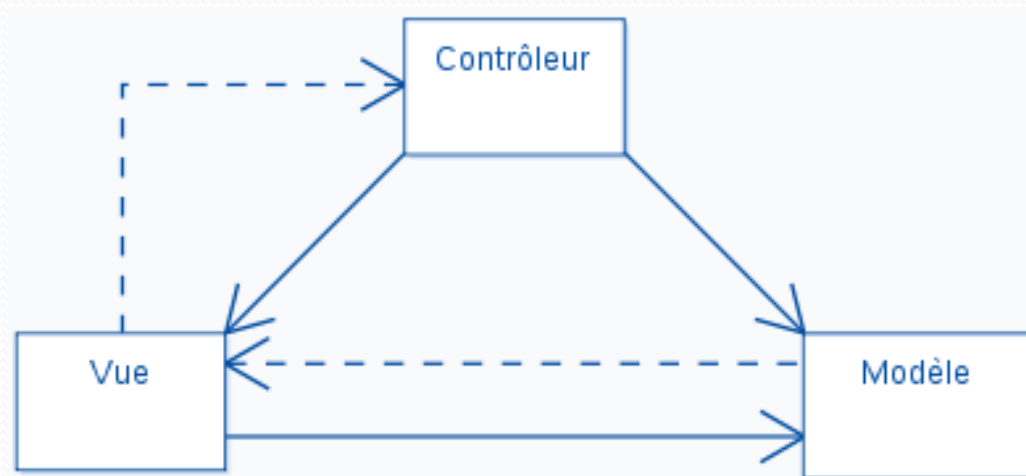
- **Optimiser le temps** de développement
- **Augmenter la qualité** d'une application
- **Minimiser les interactions** entre modules/classes
- Familles :
 - **Construction** :
 - Instanciation/configuration des classes et objets
 - **Structuraux** :
 - Organisation et groupement des classes
 - **Comportementaux** :
 - Collaboration inter-objet et distribution des responsabilités

Quelques modèles de conception

- Singleton
 - Fabrique abstraite
 - Objet composite
 - Façade
 - Décorateur
 - Observateur
 - Stratégie
 - Visiteur
 - Injection de contrôle
 - Objet d'accès aux données (DAO)
- Construction
- Structuraux
- Comportementaux
- 

Rappel : modèle-vue-contrôleur

- **Architecture et méthode de conception** d'IHM
- **Modèle** : données et leur manipulation
- **Vue** : élément de l'interface graphique
- **Contrôleur** : orchestre les actions, synchronise
- **MVC 2** : un seul contrôleur



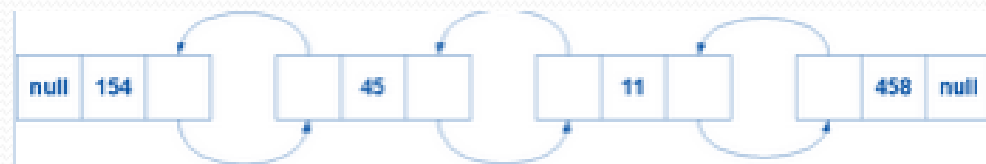
Récurtivité structurelle

- **Liste chaînée** (collection ordonnée) :

- Simplement chaînée



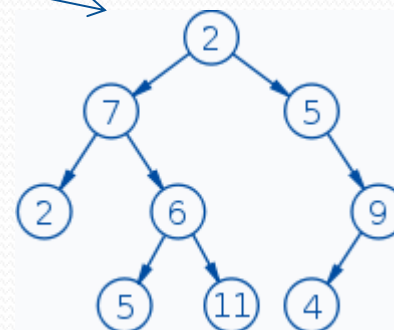
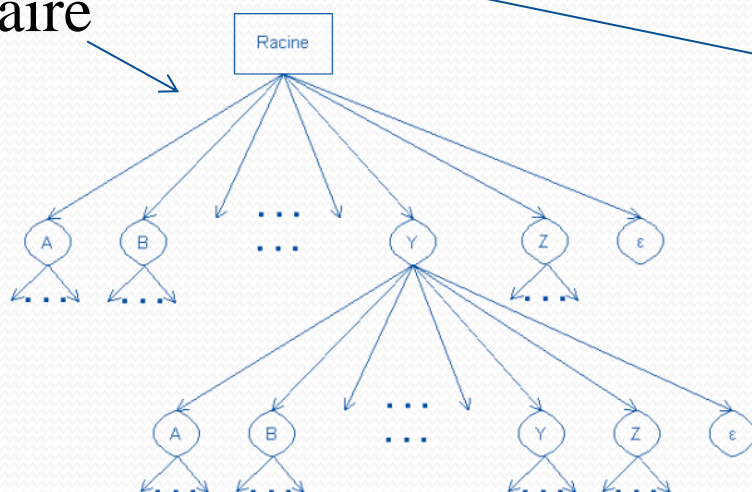
- Doublement chaînée



- **Arbre :**

- Binaire

- N-aire



Optimisation de code

- Ne faire qu'**une fois que le programme fonctionne** et répond aux spécifications fonctionnelles
- Choisir un algorithme de **complexité inférieure**
- Choisir des **structures de données** adaptées
- Bien ordonner les instructions
- Bien utiliser le langage de programmation choisi
- Bien utiliser les bibliothèques du langage
- Penser à l'optimisation automatique (compilateur)
- Utiliser un langage de bas niveau pour les points critiques

Aller plus loin

- Classe de complexité
- Tri utilisant la structure des données, volumineux, bande, Introsort
- Dérécursivité
- Récursivité croisée
- Co-variance/contra-variance
- Théorie des graphes
- Algorithmes évolutionnistes
- Réseaux de neurones

Liens

- Documents classiques :
 - Livres :
 - N.H. Xuong. *Mathématiques discrètes et informatique*.
 - Gamma, Helm, Johnson, Vlissides. *Design Patterns*.

Crédits

Auteur

Mickaël Martin Nevot

mmartin.nevot@gmail.com



Carte de visite électronique

Relecteurs

Cours en ligne sur : www.mickaël-martin-nevot.com

