Modèles de conception réutilisables ou « Design Patterns »

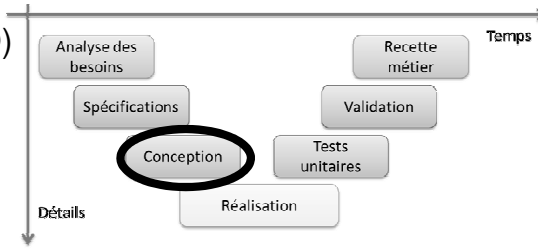
UNIVERSITE TOULOUSE III PAUL SABATIER
Faculté Sciences et Ingénierie
D[i] Département Informatique
IRIT Institut de Recherche en Informatique de Toulouse
CNRS - INP - UT3 - UT1 - UT2J

Jean-Paul ARCANGELI
Jean-Paul.Arcangeli@irit.fr
UPS – IRIT

UE MCO - M1 INFO & RT
S7 2020-2021

UE MCO – Partie « Design Patterns » (DP)

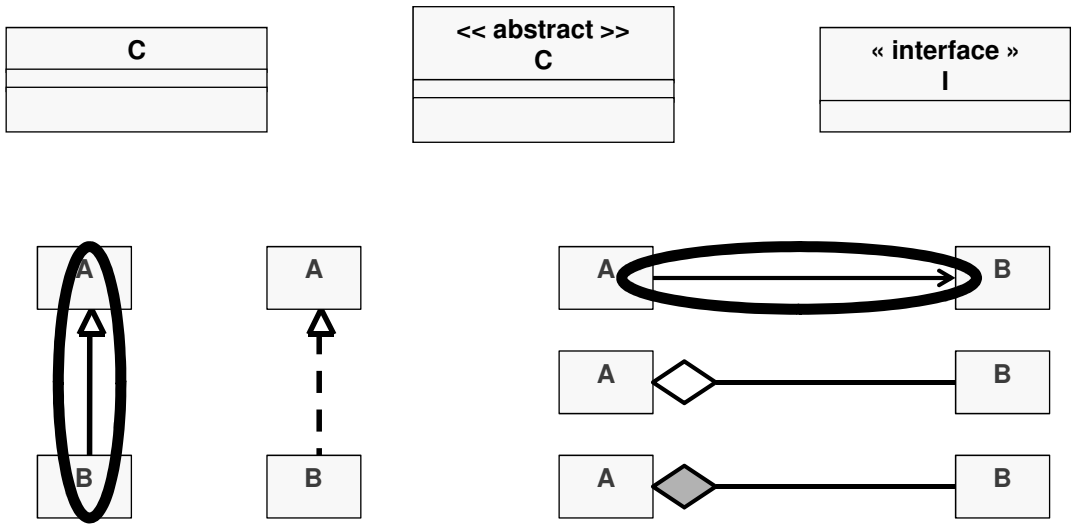
- Conception « avancée »
  - Choix de solution(s) afin de satisfaire les exigences (fonctionnelles et extrafonctionnelles)
  - Objet (même si…)
    - Mise en œuvre en Java (dans l'UE MCO)



- Prérequis
  - Conception et programmation objet
    - Expérience en conception ☹ ?
  - UML
    - Diagrammes de classes (+ de communication et de séquence)
  - Java
- 3x2h de cours + 3x2h de TD ☹
  - Pas de TP
  - À compléter par du travail personnel

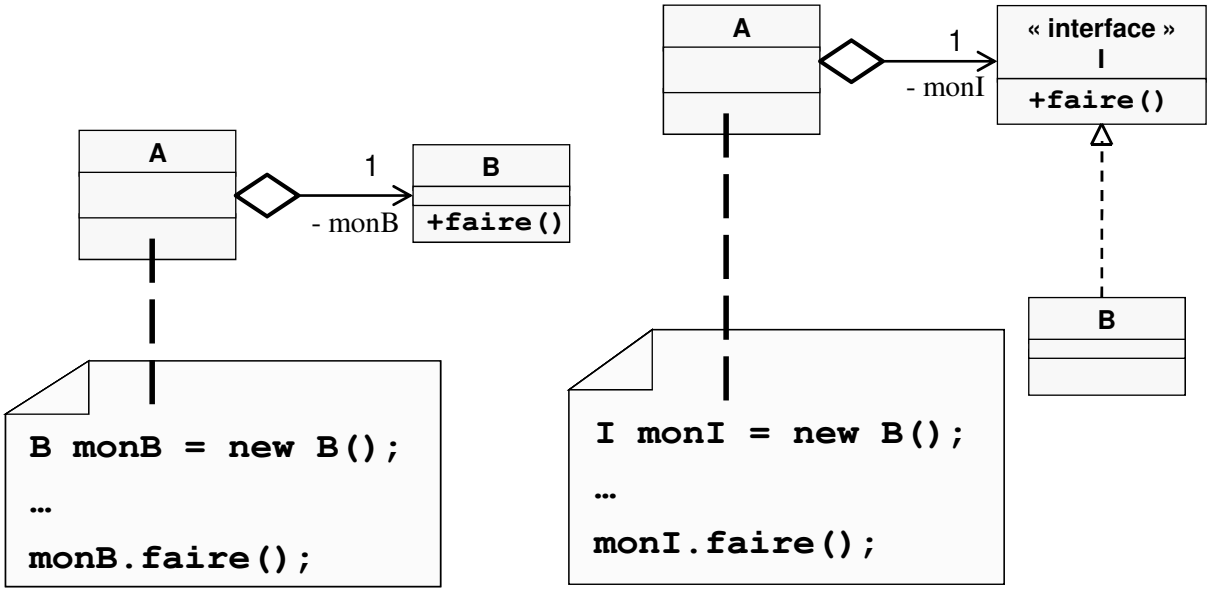
UML (Unified Modeling Language)

- Notations (rappels)

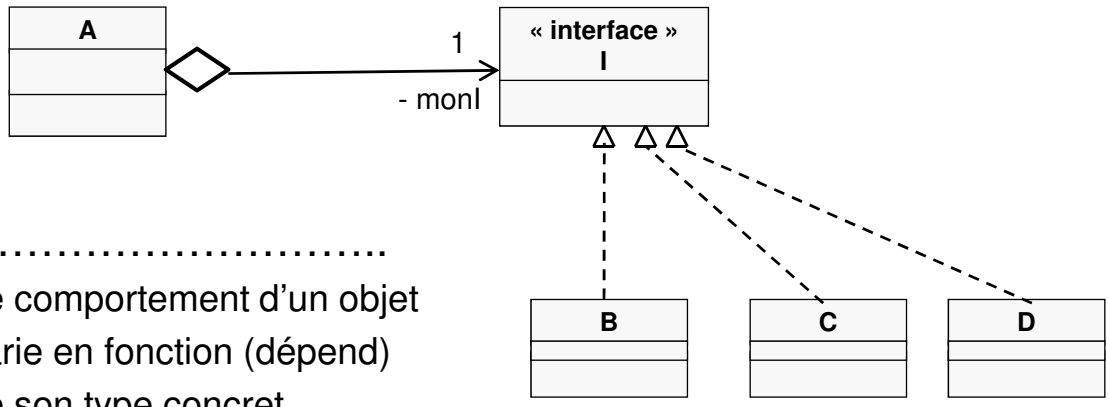


Conception et diagrammes de classes

- Quelle construction préférer ?

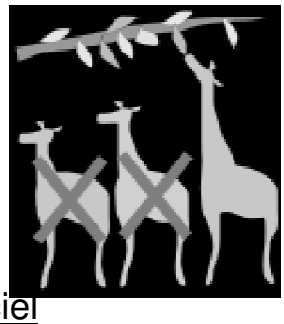


Bonnes pratiques et bons principes de conception



- Le ……..
  - Le comportement d'un objet varie en fonction (dépend) de son type concret

Bonnes pratiques et bons principes de conception



- Penser aux évolutions futures
  - L'évolution est un « invariant » dans la vie du logiciel
  - Anticiper pour éviter ou limiter les reprises de conception
  - Bien identifier les aspects susceptibles de varier
    - Séparer ce qui peut varier de ce qui ne varie pas
    - Autant que possible !

Bonnes pratiques et bons principes de conception

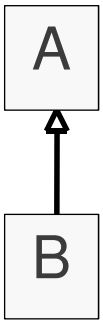
- Réduire les couplages (pour augmenter la flexibilité)
  - Minimiser les dépendances entre classes (séparation, abstraction)
    - ☛ « Programmer une interface (c*omprenez un « supertype »*), pas une implémentation »
      - C'est-à-dire manipuler les objets à travers leurs interfaces (leurs supertypes), pas leurs implantations : polymorphisme !
  - ☛ Attention au *new* !

Couplage = Degré d'interdépendance entre A et B en terme d'évolution

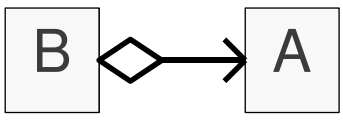
- Couplage faible : un changement dans A (ou B) a peu d'impact sur B (ou A)
- Couplage fort : un changement dans A (ou B) impose des changements importants dans B (ou A)

Bonnes pratiques et bons principes de conception

- Se méfier de l'héritage (relation « est un »)
  - L'héritage est la technique de base pour la réutilisation
    - Permet l'extension de fonctionnalité
  - Problèmes de maintenance
    - Couplage fort
    - Modification => effet de bord sur les sous-classes
    - Via les interfaces => duplication des codes d'implantation
  - L'héritage est un mécanisme simple (en phase de conception)
    - D'où une architecture simple du produit
    - Mais l'héritage est (trop ?) statique



- Préférer l'association (relation « a un »)
  - Un objet agit pour un autre par délégation
  - 1 rôle => 1 interface + 1 objet « délégué »
  - Meilleure flexibilité statique et dynamique



Bonnes pratiques et bons principes de conception

- Exemple
  - Soit une classe abstraite Animal, avec plusieurs implémentations concrètes, dont Chien et Chat

  - Version 1 (programmer une implémentation)

```
Chien c = new Chien ();
c.aboyer();
```

☹

  - Version 2 (programmer une interface / supertype)

```
Animal animal = new Chien ();
animal.emettreSon();
```

☺

  - Version 3 (sans le new)

```
animal = getAnimal();
animal.emettreSon();
```

☺☺

Concevoir (bien) est un « art » difficile !

- Même avec les technologies « objet » !
- Déterminant pour la « qualité » du produit
  - *Qualité… c'est-à-dire ?*
  - Satisfaire les exigences relatives au produit
    - Exigences fonctionnelles
    - Exigences extrafonctionnelles
      - Performance, sécurité, sûreté…
      - Maintenabilité, flexibilité, évolutivité, intelligibilité
  - Satisfaire les exigences relatives au projet
    - P. ex. productivité
      - Développer spécifiquement vs réutiliser des solutions existantes ?
      - Éventuellement développer pour réutiliser ultérieurement

Concevoir (bien) est un « art » difficile !

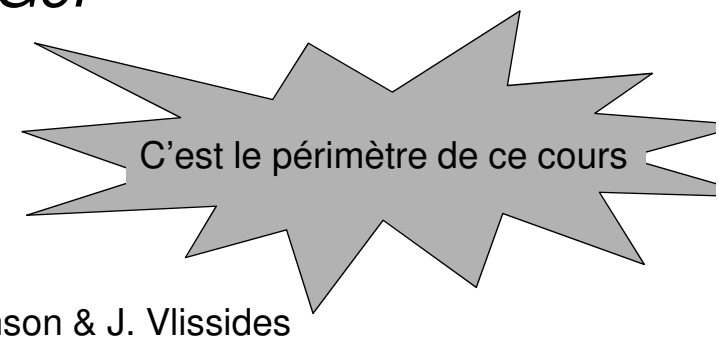
- L'expérience et l'habileté du concepteur sont primordiales
  - Ne pas « réinventer la roue » !
    - Il existe des problèmes de conception récurrents
      - ☛ Savoir les identifier
    - Pour lesquels il y a des solutions connues et éprouvées
      - ☛ Savoir les (ré)utiliser
  - Réutiliser l'expérience de conception (*vs* réutiliser le code)
    - Savoir-faire en matière de structuration et de composition
      - ☛ Solutions de conception
        - Définissent l'organisation et les relations entre classes
        - Réutilisation d'une application à une autre
  - Besoin de documentation pour capitaliser l'expérience

DESIGN PATTERNS

Patterns GRASP

- GRASP = General Responsibility Assignment Software Patterns
  - Principes pour l'assignation de responsabilités aux classes et objets
    - Patterns de conception de base, assez simples et intuitifs
    - « Bonnes pratiques » de conception
  - Exemples : expert en information, faible couplage, forte cohésion, création, délégation, contrôleur, polymorphisme...
  - *Applying UML and Patterns*, Craig Larman, Prentice Hall, 2004

Design patterns du GoF



- GoF = « Gang of Four »
  - E. Gamma, R. Helm, R. Johnson & J. Vlissides
  - Proposition d'un catalogue de design patterns (1995)

- Organisation du code dans un cadre objet
  - Expression en termes de classes, d'interfaces, d'objets… et de relations entre ces éléments
  - Le concept « objet » donne des mécanismes pour construire des logiciels flexibles, maintenables, évolutifs
    - Abstraction, encapsulation, héritage et délégation, polymorphisme…

Patterns d'architecture

- Structuration à grande échelle du système logiciel
  - Forme générique d'architecture (« pattern » ou « style ») en termes d'éléments et de propriétés, de contraintes et de relations entre ces éléments
  - Exemples
    - Client-serveur, architectures n-tier…
    - MVC

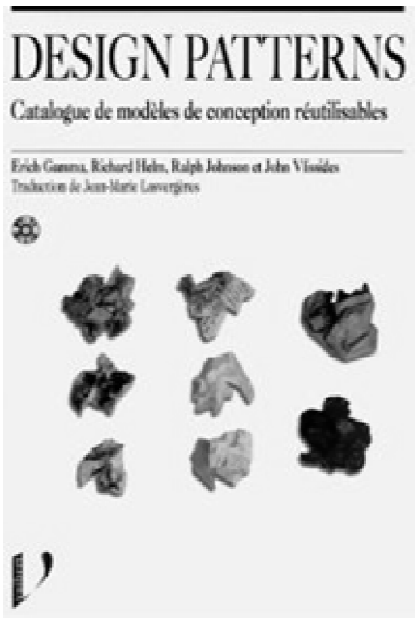
Objectifs du cours

- Connaître le concept de design pattern
  - Savoir ce qu'est un design pattern (et ce que ce n'est pas)
- Connaître les (quelques) principaux design patterns (GoF)
  - Connaître leurs avantages et leurs limites
- Savoir manipuler les design patterns (compétences)
  - Savoir exploiter la description d'un design pattern en vue de son utilisation
  - Savoir utiliser un design pattern (ou ne pas l'utiliser…) dans le cadre d'un problème de conception
  - Savoir mettre en œuvre (p. ex. en Java) le design pattern choisi

Plan du cours (MCO – Partie DP)

1. Introduction : qu'est-ce qu'un design pattern ?
   - « Design pattern » = « modèle de conception » = « patron de conception »
2. Description et classification des design patterns
3. Catalogue (quelques design patterns)
4. Conclusion (remarques et compléments)

Quelques ressources



- *Design Patterns, Elements of Reusable Object-Oriented Software*
  - E. Gamma, R. Helm, R. Johnson & J. Vlissides (« the Gang of Four » ou GoF), 1995
  - En français : Design Patterns, Catalogue de modèles de conception réutilisables, Vuibert Informatique, 1999

Quelques ressources

- *Head First - Design Patterns*
  - E. Freeman & E. Freeman, O'Reilly, 2005
  - En français : Design Patterns - Tête la première, Digit Books 2009

Quelques ressources

- *Design Patterns pour Java*
  - L. Debrauwer, ENI, seconde édition, 2009
- *Design Patterns pour C#*
  - L. Debrauwer, ENI, 2009

Quelques ressources

- Descriptions (trop ?) synthétique des design patterns du GoF
  - http://www.mcdonaldland.info/files/designpatterns/designpatternscard.pdf
  - http://www.blackwasp.co.uk/GangOfFour.aspx
- https://refactoring.guru/
- https://sourcemaking.com/design_patterns
- …
- Les Design Patterns en Java
  - S.J. Metsker & W.C. Wake, Campus Press, 2006
- The « Hillside Group » home page
  - http://hillside.net/