

Modèles de conception réutilisables ou « Design Patterns »



Jean-Paul ARCANGELI

Jean-Paul.Arcangeli@irit.fr

D[i] | Département Informatique



UPS – IRIT

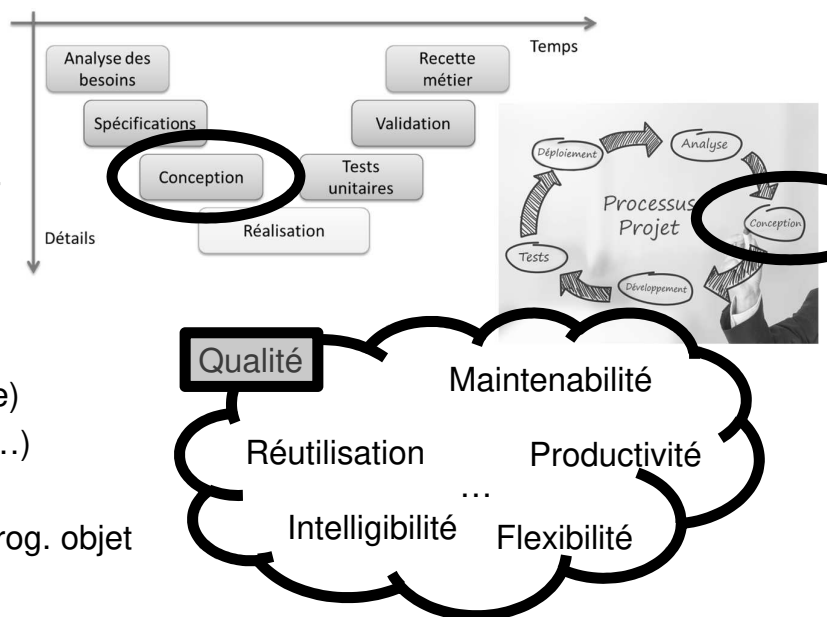


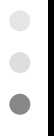
M1 MIAGE FI+FA – Ingénierie Logicielle

2022-2023

UE « Ingénierie Logicielle » (IL)

- 3 ECTS, 30h
- 2 parties
 - VERIFICATION/TEST
 - DESIGN PATTERNS
 - 16 h = 8 hC + 8 hTD
 - Pas de TP
 - Conception (avancée)
 - Objet (même si...)
 - Prérequis
 - Conception et prog. objet
 - UML 
 - Diagrammes de classes
 - Diagrammes d'objets (séquence, communication)
 - Java 
 - Expérience en conception ☺





UE « Ingénierie Logicielle » (IL)

- Modalités de Contrôle des Connaissances (MCC)
 - 50% CC, partie Test
 - 50% CT, partie DP
 - Sur table, 1h30
 - Mercredi 19/10 à 13h30

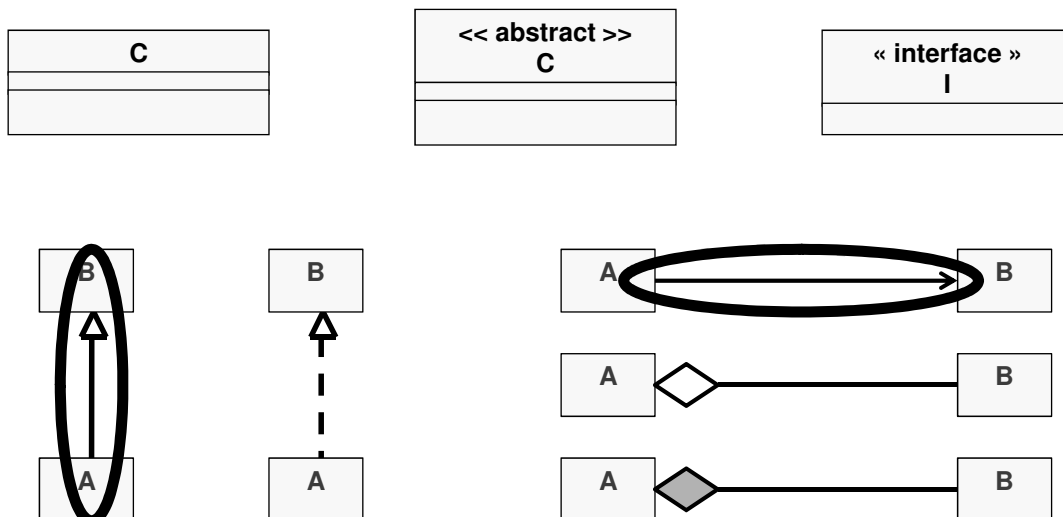
3



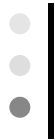
UML (Unified Modeling Language)



- Classes et relations entre classes (rappels)

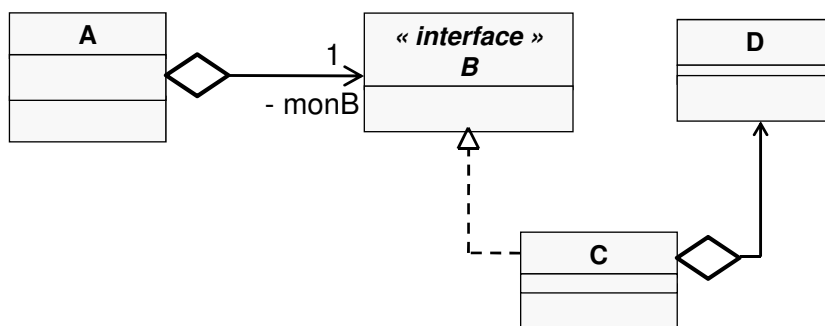
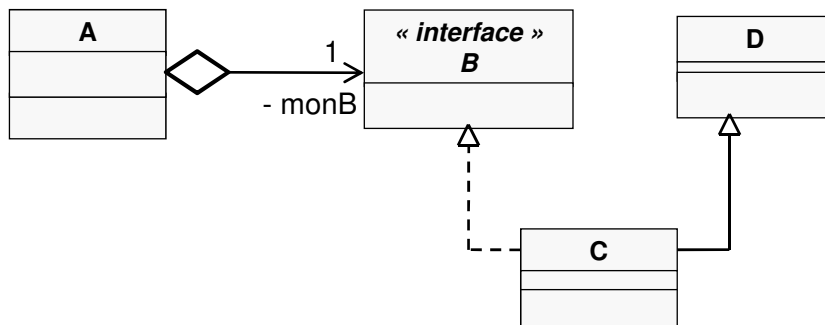


4



Conception et diagrammes de classes

- Exemples



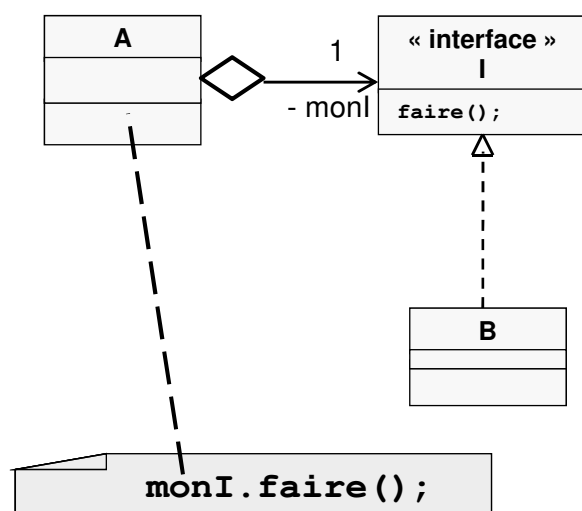
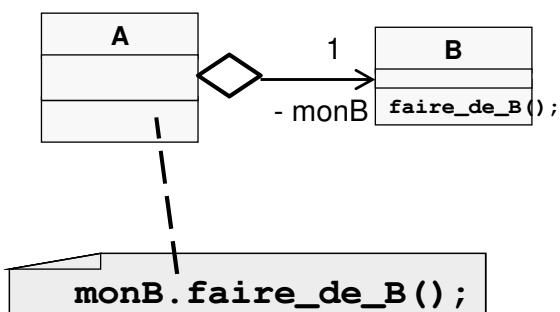
Au déploiement,
combien d'objets
sont créés ?

5

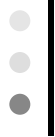


Conception et diagrammes de classes

- Quelle construction préférer ?

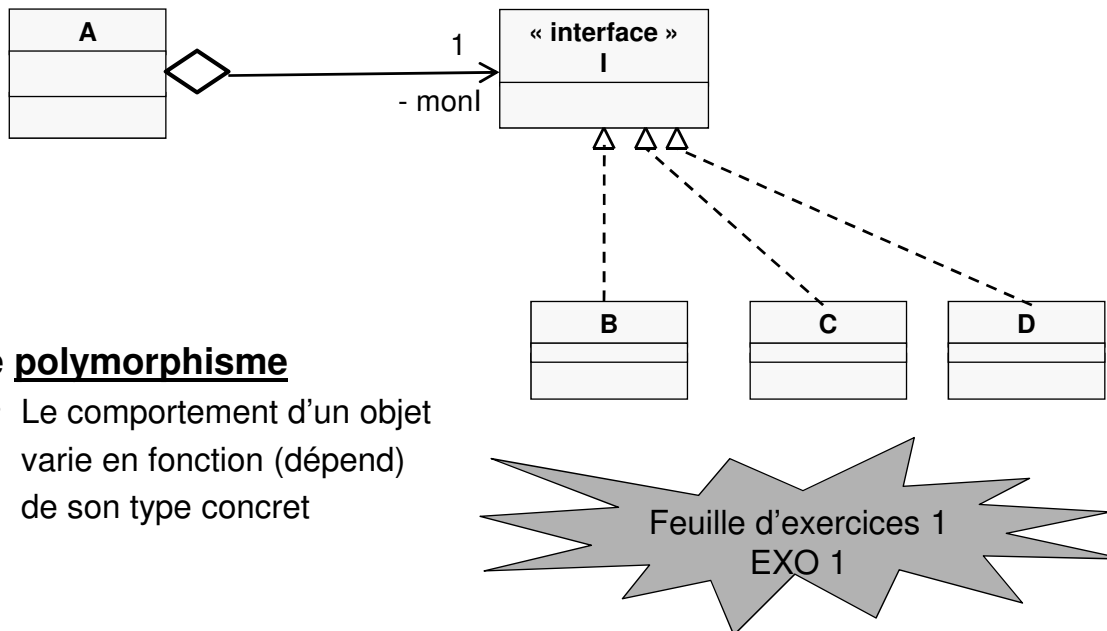


6



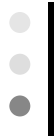
Conception et diagrammes de classes

- Une bonne pratique de conception



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

7



Bonnes pratiques

- Quelques 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

- Quelques bons principes de conception
 - Réduire les couplages (pour augmenter la flexibilité)
 - Minimiser les dépendances entre classes (séparation, abstraction)
 - « Programmer une interface (*comprenez 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)


9




Bonnes pratiques

- 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)

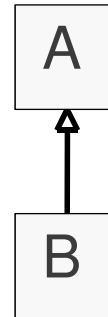
```
Animal animal = getAnimal();  
animal.emettreSon();
```



10

Bonnes pratiques

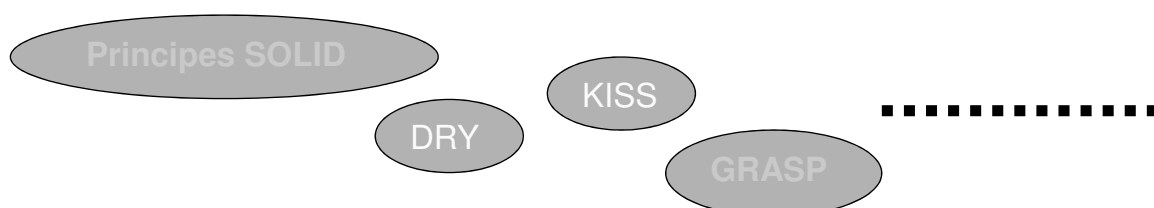
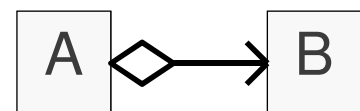
- Quelques bons principes de conception (suite)
 - Se méfier de l'héritage
 - L'héritage est la technique de base pour la réutilisation
 - Permet l'extension de fonctionnalité
 - Problèmes de maintenance
 - Couplage fort
 - Modification => effet de bord sur les sous-classes
 - *Via* les interfaces => 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 (relation « est un »)



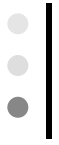
11

Bonnes pratiques

- Quelques bons principes de conception (suite)
 - Préférer l'association à l'héritage
 - Relation « a un » vs relation « est un »
 - Un objet agit pour un autre par délégation
 - 1 rôle => 1 interface + 1 objet « délégué »
 - Meilleure flexibilité statique et dynamique



12



Plan du cours

1) Introduction

Qu'est-ce qu'un *design pattern* (« modèle » ou « patron » de conception) ?

2) Description et classification des modèles de conception

3) Catalogue : quelques modèles de conception

4) Conclusion