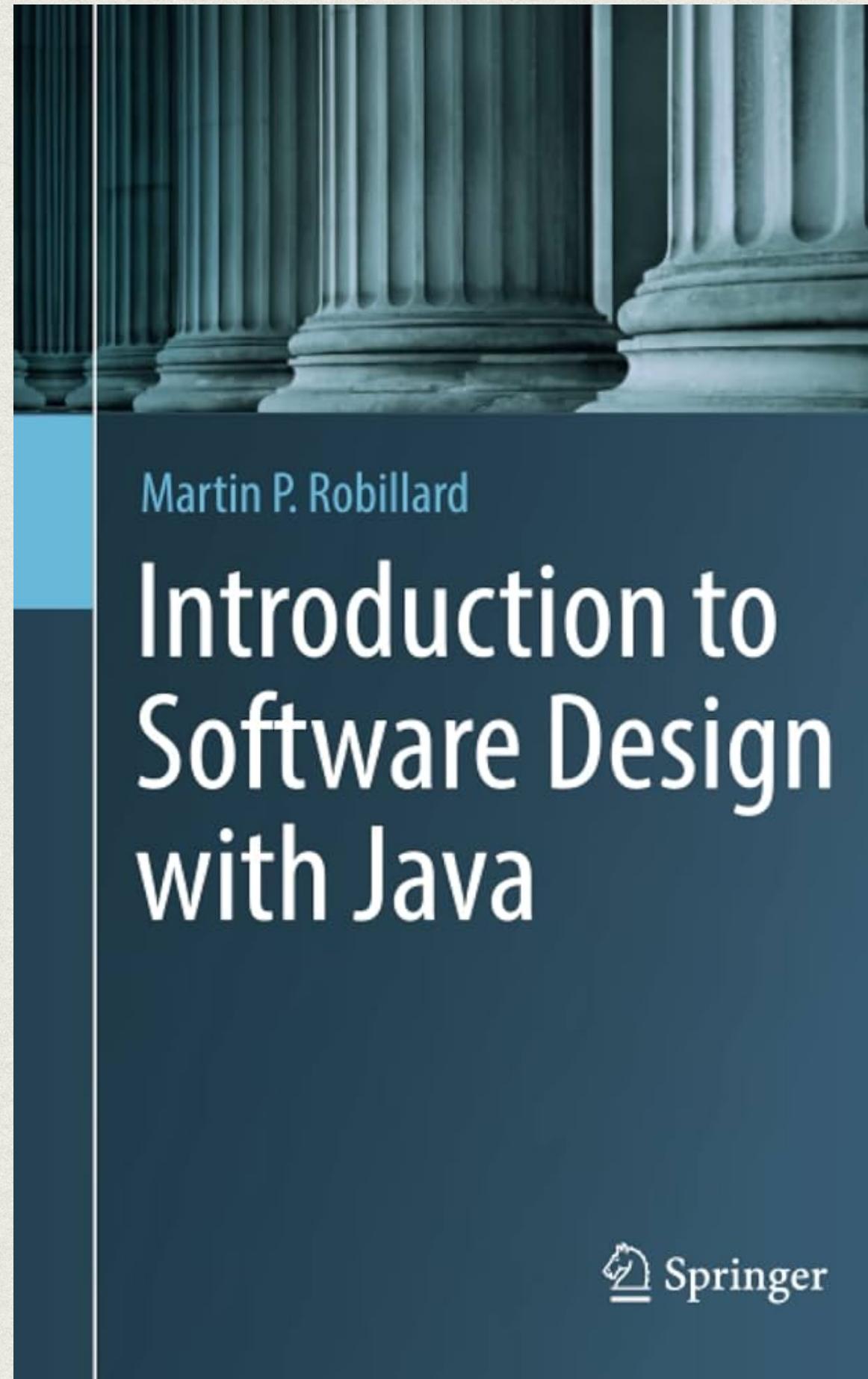




M1 IMAGE / MAPC COMPOSITION

Pascal POIZAT

RÉFÉRENCES



Conception Logicielle
Site web du cours INF5153 de l'UQAM

- Retourner à la page d'accueil

Introduction

Notes de cours

- Introduction au cours
- Écosystème de développement
- Schotten Totten

Capsules Vidéos

Introduction au cours

Introduction du cours de Conception Logicielle [IN...]

Partager

Regarder sur YouTube

Sébastien Mosser - INF5153
Chapitre 0 - Capsule 1
Automne 2020

ACE

CC BY-NC-SA

STRUCTURATION

- système trop large → parties plus petites + assemblage
« diviser pour régner » (*divide-and-conquer*)
- c'est classique, il y a tout un historique
par exemple des modules de Parnas en 1972 aux (micro-)services
- ici via utilisation de relations « verticales » et « horizontales »
spécialisation et composition

PRINCIPES DE BASE

- on part d'une classe qui a trop de responsabilités = **God Class⁺**
! responsabilité \neq méthode
- on distribue les responsabilités
 - utilisation de l'héritage (! L de SOLID)
 - utilisation de la composition ET de la délégation
- exemple de GameModel et de ses « 13 piles de cartes »

PRINCIPES DE BASE

- attention à ne pas arriver à une solution problématique
- pour aider : Composite, Decorator, loi de Demeter

COMPOSITE

- illustration sur CardSource
implémentations Deck, CardSequence et CompositeCardSource
méthodes isEmpty et draw
itération sur les éléments
- diagramme de classe et diagramme d'objets
(problème de cycles, un patron n'est pas tout)
- diagramme de séquence et code
(passer de l'un à l'autre + délégation)
- problématique de l'ajout dans le composite avec add(e) vs this()
(avantages et inconvénients)

LOI DE DEMETER

- illustration sur deux diagrammes de séquence
(générique et/ou GameModel)
- présentation du problème
- le principe de la loi
(double point, le voisin de mon voisin)

DECORATOR

- illustration sur CardSource avec comportement « en plus »
(log ou mémorisation)
- problématiques de certaines solutions
 - classes spécialisées (combinatoire, actions en + vs méthodes en +)
 - implémentation de (multi-)modalité (God Class et Switch Statement)
- solution et fonctionnement du décorateur

DECORATOR

- facilité de combiner les décos
- importance du caractère additif de ces dernières
- suppression de comportement vs L de SOLID
- perte d'identité
- combinaison Composite + Décorateur

CREATION DYNAMIQUE : COPIE PAR CLONAGE

- que créer dynamiquement quand on a besoin ?
- illustration sur CardSourceManager et la copie de ses sources
- checklist pour utilisation de Clonable

CREATION DYNAMIQUE : CREATION PAR PROTOTYPES

- mauvaises solutions
- solution de Prototype

SYNTHÈSE

- attention aux **God Class⁺**, respecter **SOLID**, diviser les choses utiliser la délégation au besoin
- loi de **Demeter**
- patrons **Composite**
groupe (arbres) d'objets
se comportant comme un objet « simple »
- patrons **Decorator**
ajout dynamique de comportement à des objets
- clonage (**Clonable**) et patron **Prototype**
création d'objets à partir d'objet existants sans forcément connaître le type de ces derniers
- **se rappeler** : les patrons ne font pas tout

