SOFTWARE ARCHITECTURE AND DESIGN

Trần Đình Quế

Mail: tdque@yahoo.com

Phone: 0904066883

Các nhóm gửi bài cho nhau chấm: 11->10->9->8->7->6->5->4->3->2->1->11

Chấm điểm BT2

- **Câu 1,4,5,6:** 1.5đ (sơ sài 0.5đ)
- **Câu 2,3,7,8:** 1đ (sơ sài 0.5đ)
- **Đến 3:10PM các nhóm submit kết quả điểm**
- Lớp trưởng tạo file excel để nhóm nhập điểm (danh sách, câu 1…..8 và nhận xét)

Bài tập nhóm: Hệ thống XYZ phân công rõ ai làm gì....

Chương 1: Xác định yêu cầu

1.1 Giới thiệu hệ thống

1.2 Các chức năng của hệ thống

1.3 Biểu đồ use case

1.4 Use stories

Chương 2: Cơ sở 4+1 view cho Thiết kế kiến trúc

//trình bày các biểu đồ tương ứng với các view như trong slide trang 33-34

2.1 Logical view

2.2 Process view

2.3 Development view

2.4 Deployment view

Chương 3: Biểu đồ lớp – Gói và mẫu thiết kế - Cơ sở dữ liệu

Chương 4: Áp dụng một số kỹ thuật thông minh cho hệ thống

Chương 5: Cài đặt

Course Contents

- Architecture design
- Architecture patterns
- Subsystem/detailed design [components/packages /classes]
- **Design patterns**
- Design intelligent systems

Questions???

1. Design pattern =/= architecture pattern?
2. MVC && DAO?
3. Why design pattern?
4. Reusability, maintainability (CODE), upgrade

//upgrade: to improve the quality or usefulness of something,
//or change it for something newer or of a better standard:

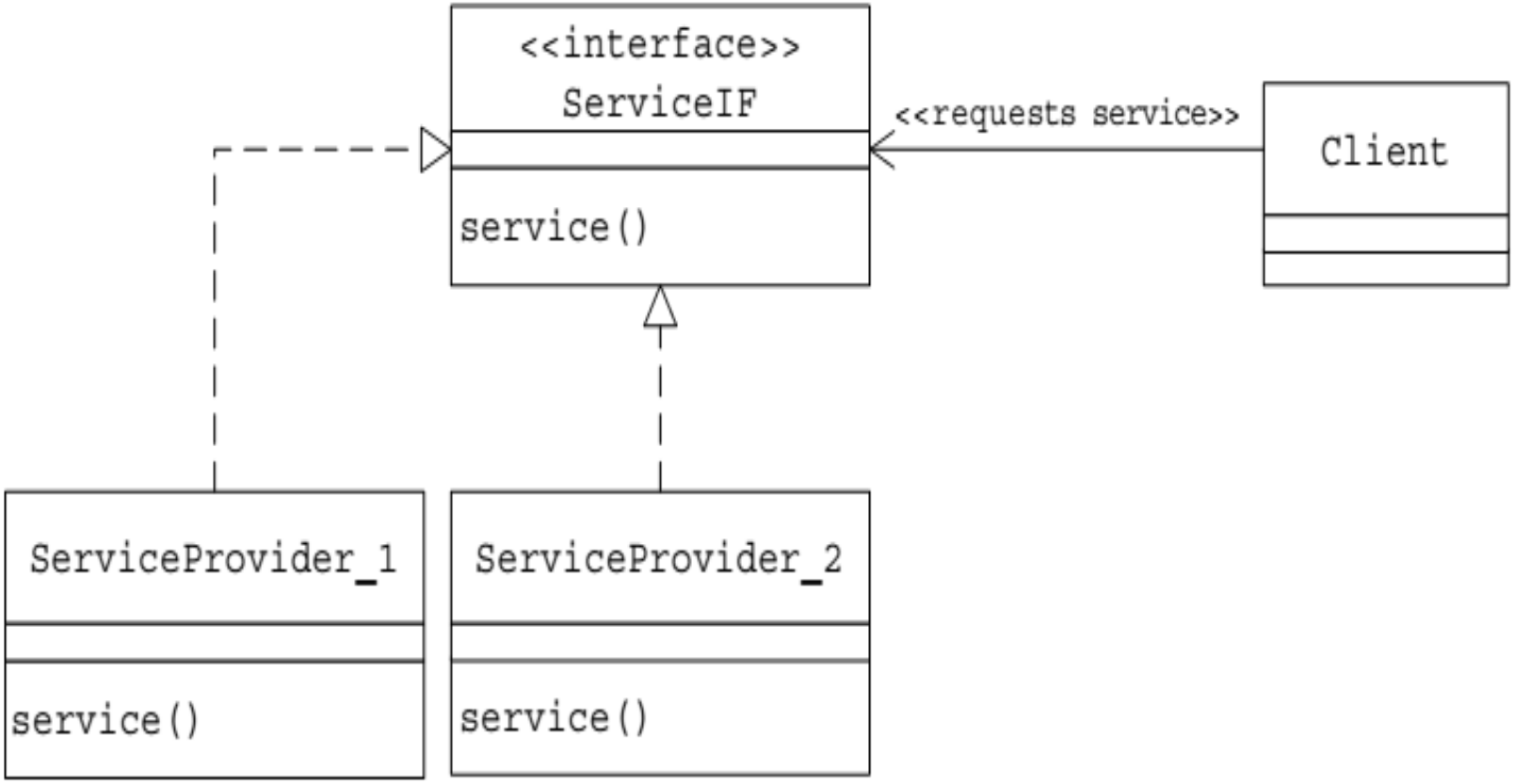
Design Pattern: Introduction

- Classification of design patterns
- Refer to Textbook

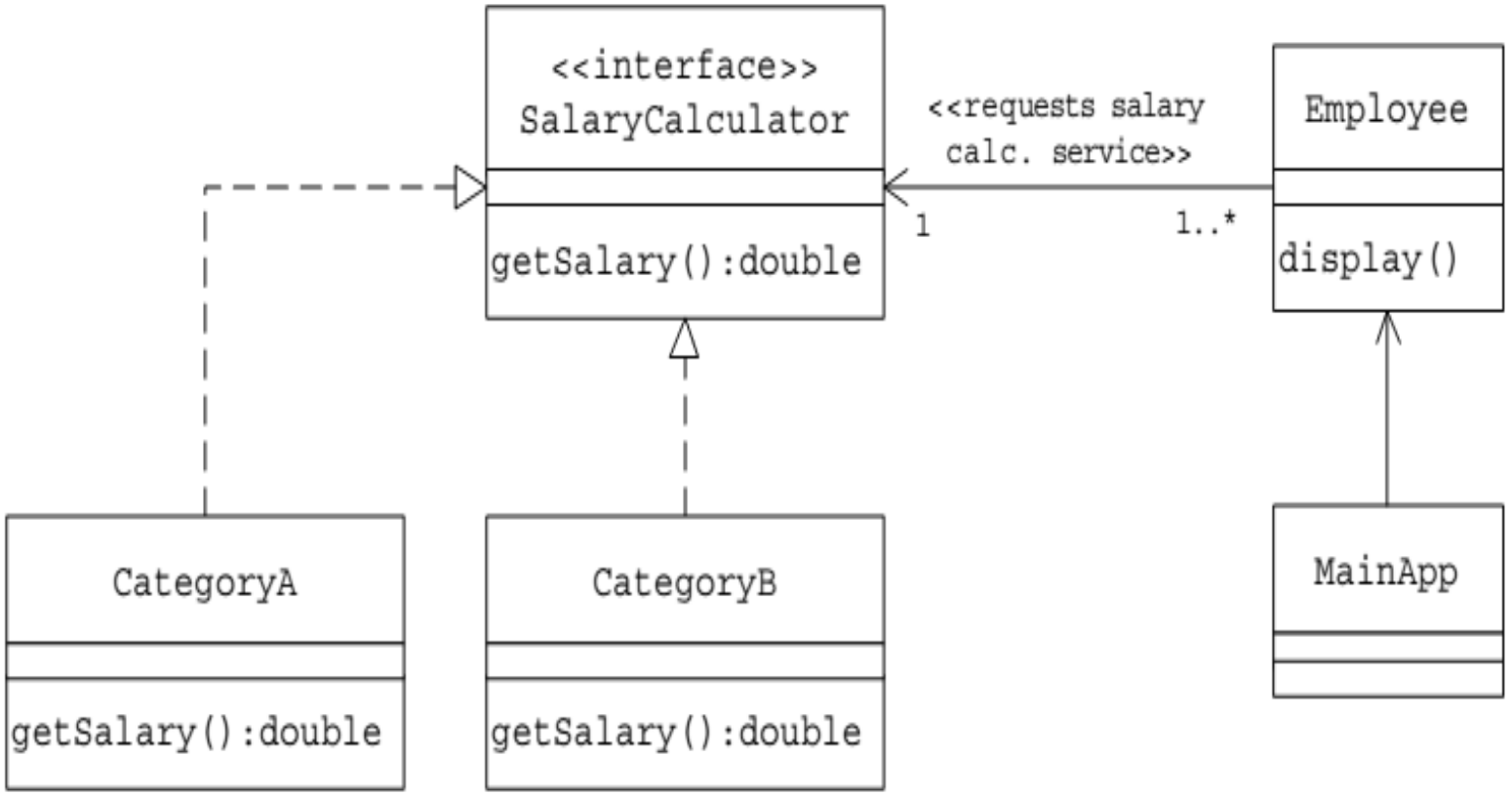
Module 1: Interface & Abstract

- Why using interface?
- Why using abstract?
- **Service** given by class = **method**

Problem 1: Using various services



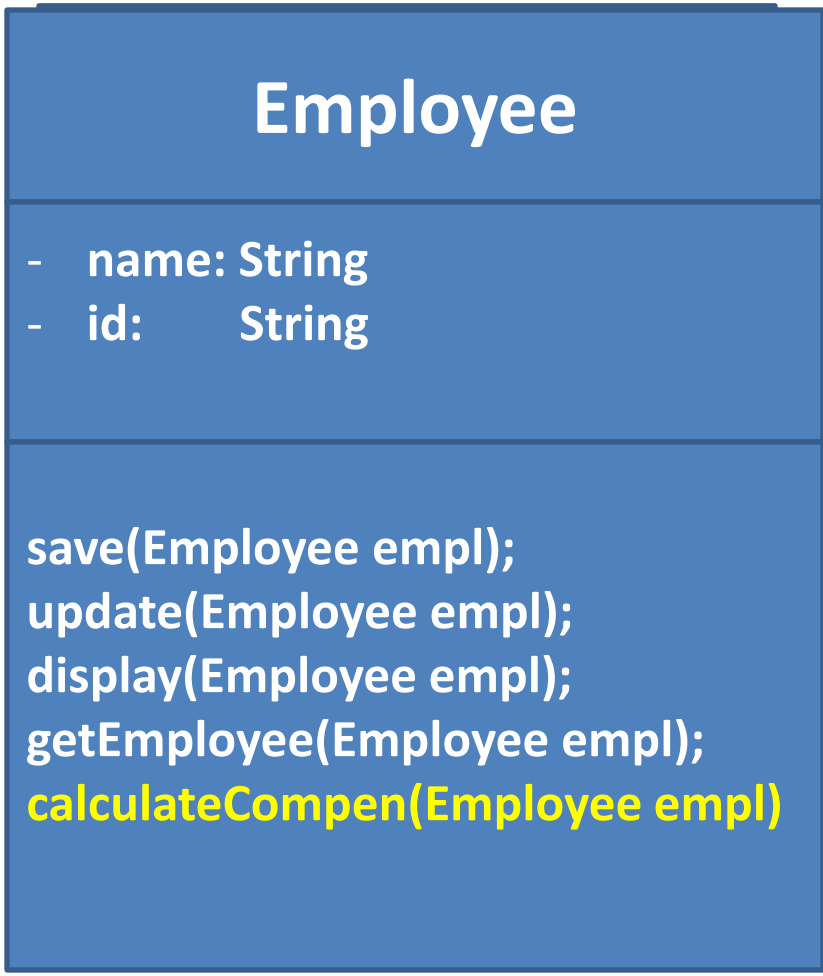
Application: salary calculation



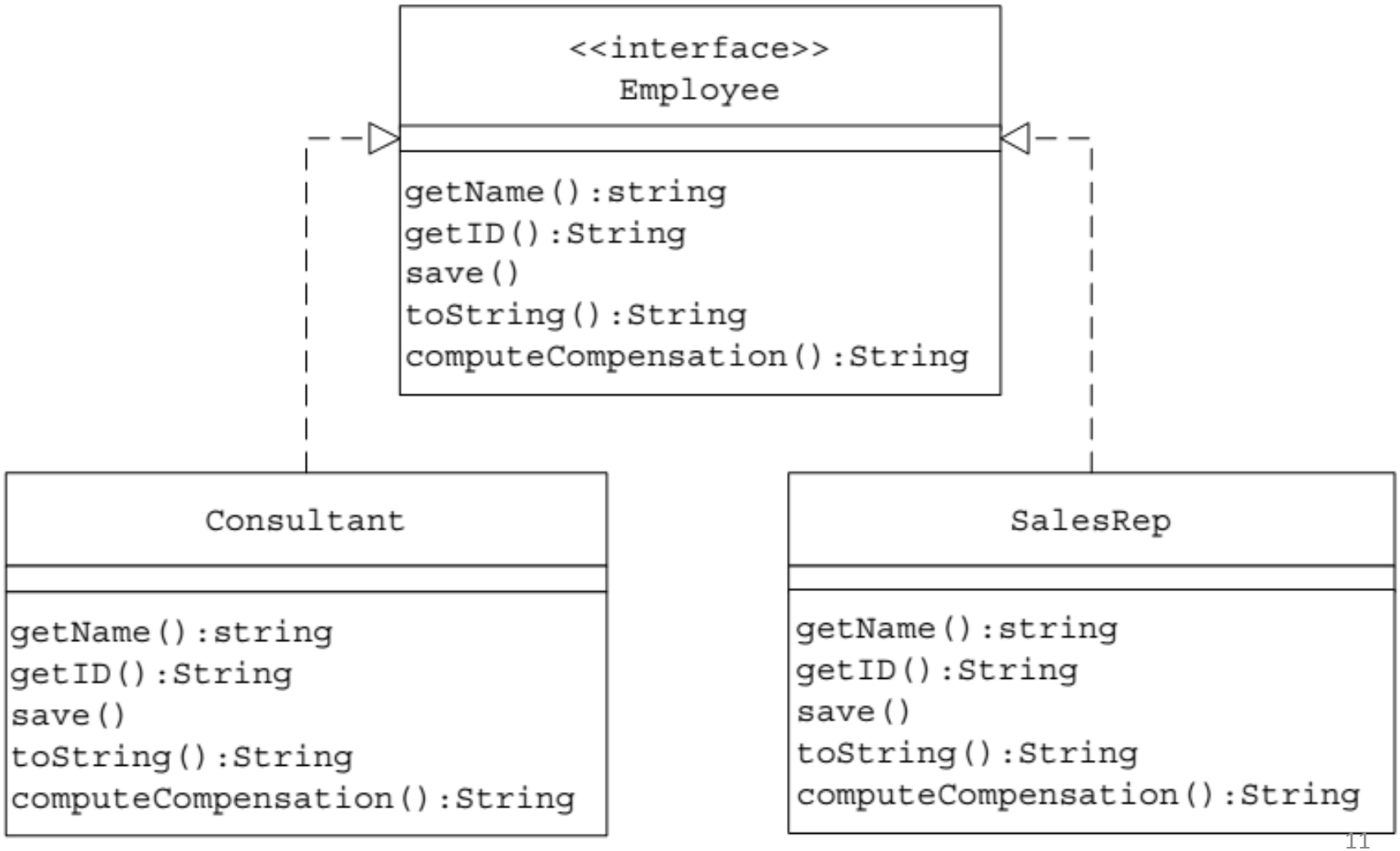
Problem 2: Class design with abstract/interface

Designing **Employee** class with methods

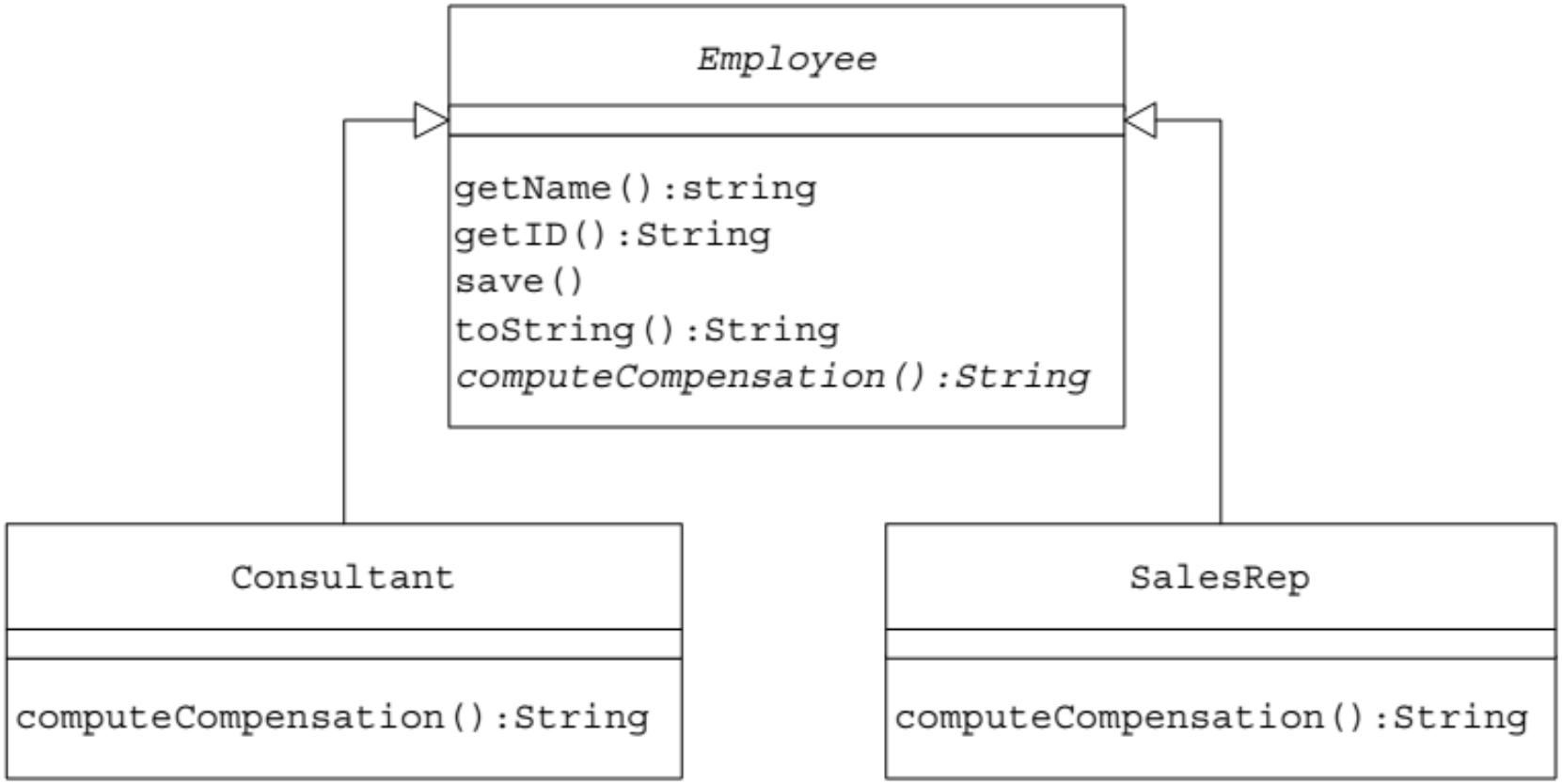
- Save employee data
- Display employee data
- Update
- Get employee attributes such as name and ID
- **Calculate compensation/salary**



Using interface



Using abstract



Design Pattern: introduction

https://www.coursera.org/lecture/design-patterns/2-1-1-what-is-a-design-pattern-C7wF7

https://www.tutorialspoint.com/design_pattern/index.htm

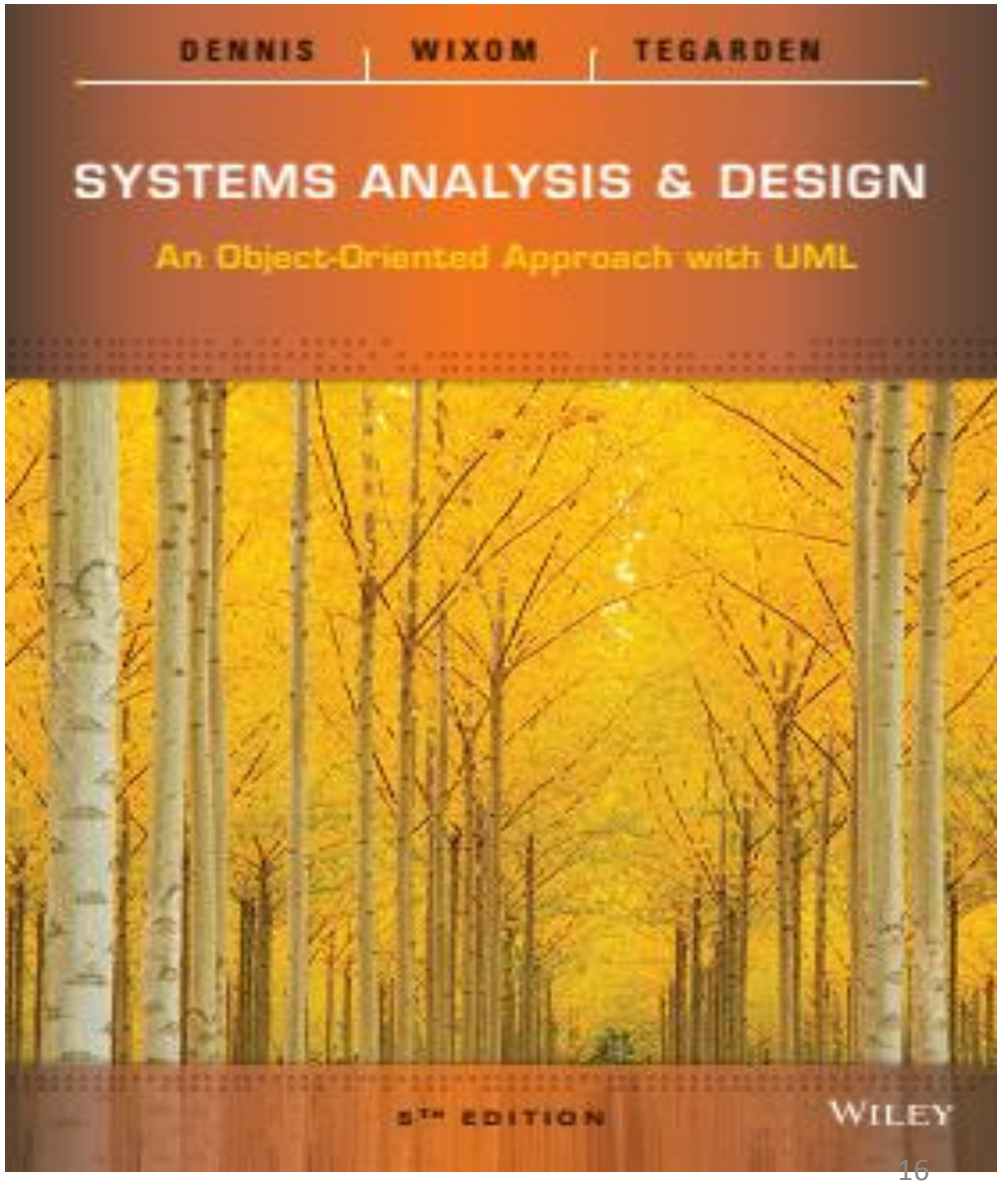
Design pattern: factory method

- https://www.youtube.com/watch?v=EcFVTgRHJLM
- https://www.youtube.com/watch?v=v-GiuMmsXj4
- Linux https://www.youtube.com/watch?v=o8NPllzkFhE

CRC card: what? How?

Class Responsibility Collaborator (CRC)

This book, page 172



CRC card: what? How?

- http://agilemodeling.com/artifacts/crcModel.htm
- https://www.youtube.com/watch?v=59tkQ-FwcpA

User stories

https://www.pinterest.com/pin/807833251882080010/visual-search/

CRC Card

- **Class-Responsibility-Collaborator cards** https://www.cs.uct.ac.za/mit_notes/software/htmls/ch05s19.html
- Class-responsibility-collaborator cards (CRC cards) are not a part of the UML specification, but they are a useful tool for organizing classes during analysis and design.
- A CRC card is a physical card representing a single class. Each card lists the class's name, attributes and methods (its responsibilities), and class associations (collaborations). The collection of these CRC cards is the CRC model.
- Using CRC cards is a straightforward addition to object-oriented analysis and design:
  - Identify the classes.
  - List responsibilities.
  - List collaborators.
- CRC cards can be used during analysis and design while classes are being discovered in order to keep track of them.
- CRC cards have various benefits, which you might notice makes them very amenable to iterative and incremental process models, especially agile ones:

- They are **portable**: because CRC cards are physical objects, they can be used anywhere. Importantly, they can easily be used during group meetings.

- They are **tangible**: the participants in a meeting can all easily examine the cards, and hence examine the system.

- They have a **limited size**: because of their physicality, CRC cards can only hold a limited amount of information. This makes them useful to restricting object-oriented analysis and design from becoming too low-level.

- Class responsibilities are the class's attributes and methods. Clearly, they represent the class's state and behaviour. Collaborators represent the associations the class has with other classes.
- CRC cards are useful when the development of classes need to be divided between software engineers, as the cards can be physically handed over to them. A useful time to do this is when classes are being reviewed, for, say, determining whether they are appropriate in a design.

VP for CRC card??

- https://www.visual-paradigm.com/support/documents/vpusergui de/94/1289/6518_drawingcrcca.html

2. Class Relationships

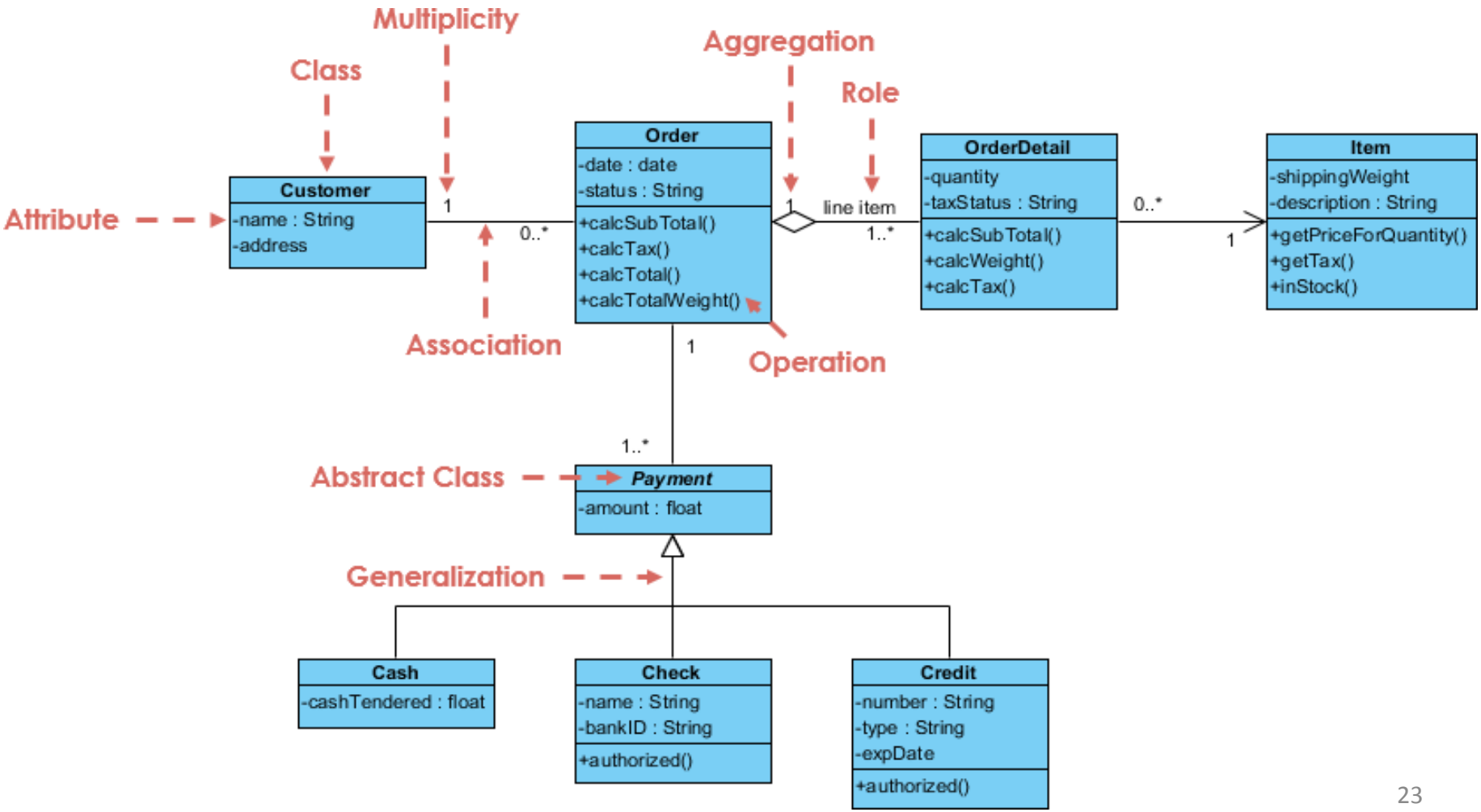
- Understand relationships??
- Discover relationships? Interesting problem?
- Use this relationship but not that ones??
- Correct or incorrect??

Class Relationships

- https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/

Relationship?

Why using association Between Customer and Order??

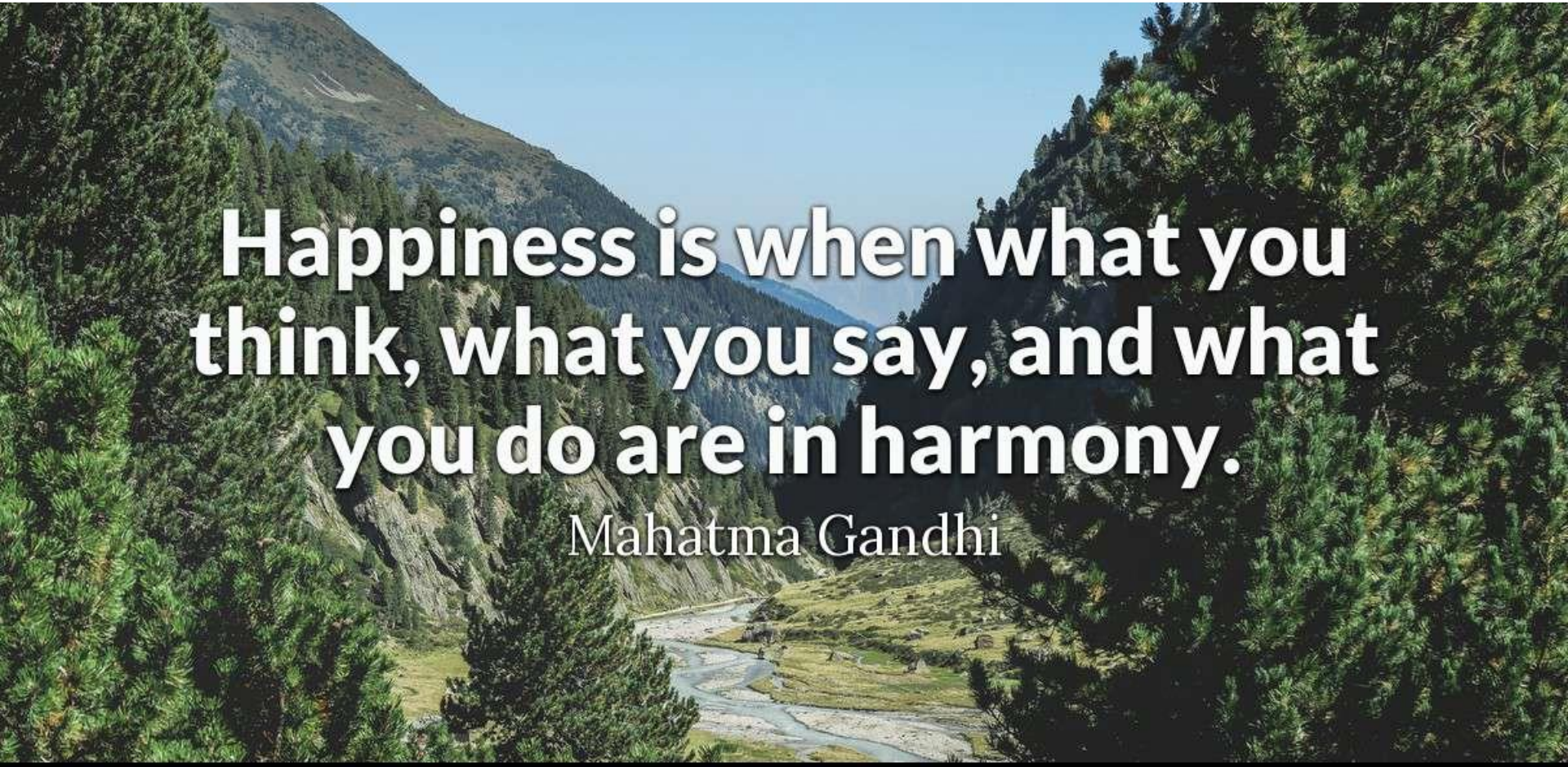


Class diagram

Student analyzes the following class diagrams

- https://www.uml-diagrams.org/class-diagrams-examples.html
- http://agilemodeling.com/artifacts/classDiagram.htm
- https://www.uml-diagrams.org/library-domain-uml-class-diagram-example.html
- https://www.educative.io/courses/grokking-the-object-oriented-design-interview/RMlM3NgjAyR
- https://circle.visual-paradigm.com/on-demand-model-etl-example-lms/

“The more I learn, the more I realize how much I don't know” – Albert Einstein



Questions?????

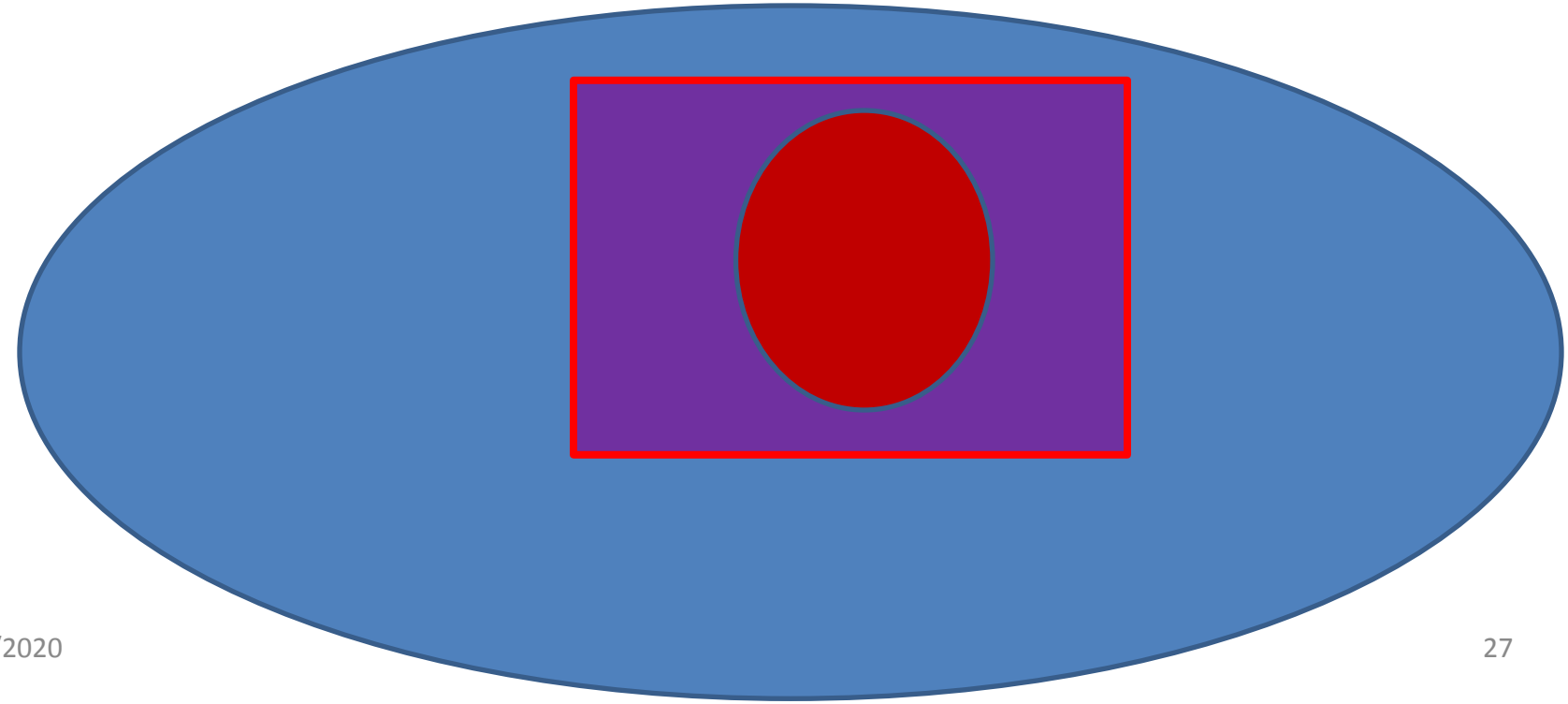
Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

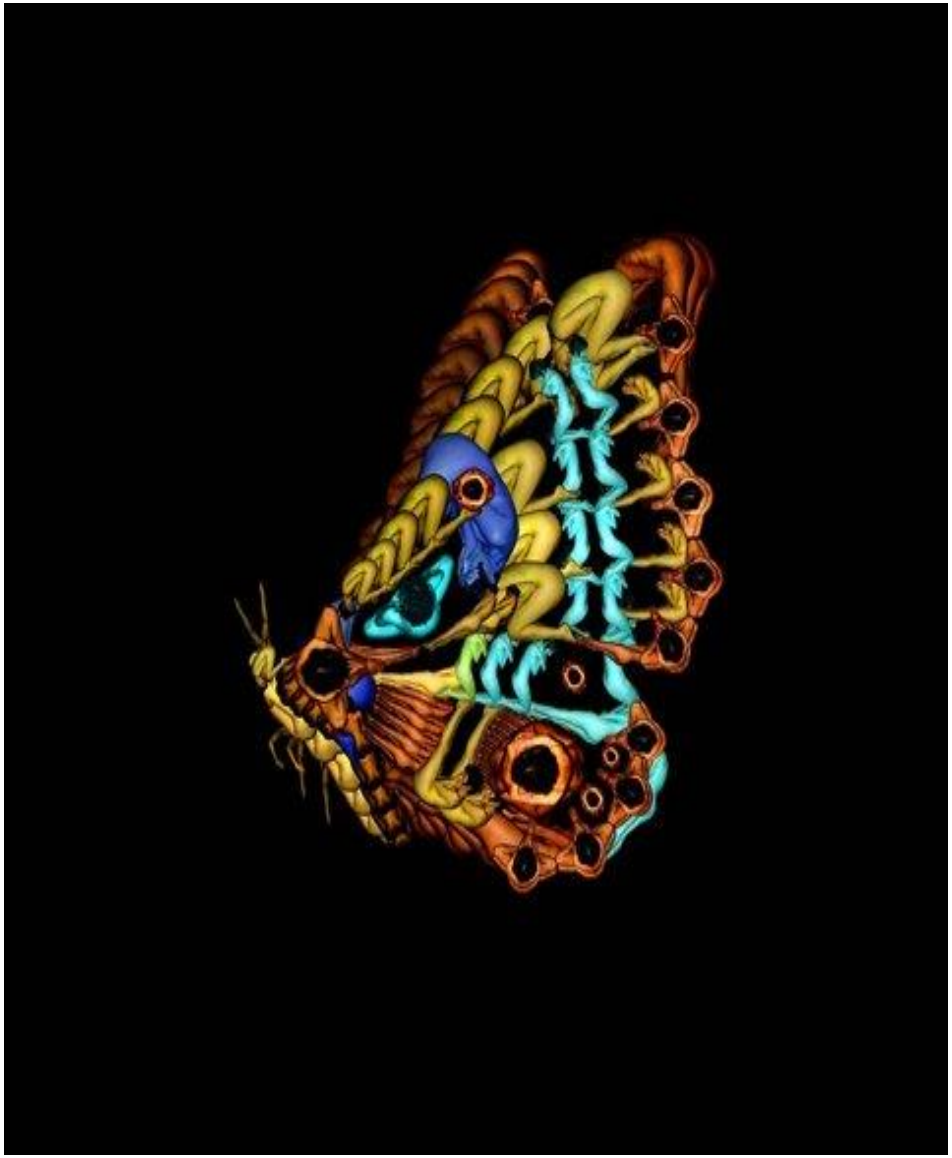
- Philosophy of architecture design: what is basis to define, determine software architecture?
- View?? Software-intensive systems
- Static/dynamic, inside/outside?
- Stakeholders?

Questions ?????

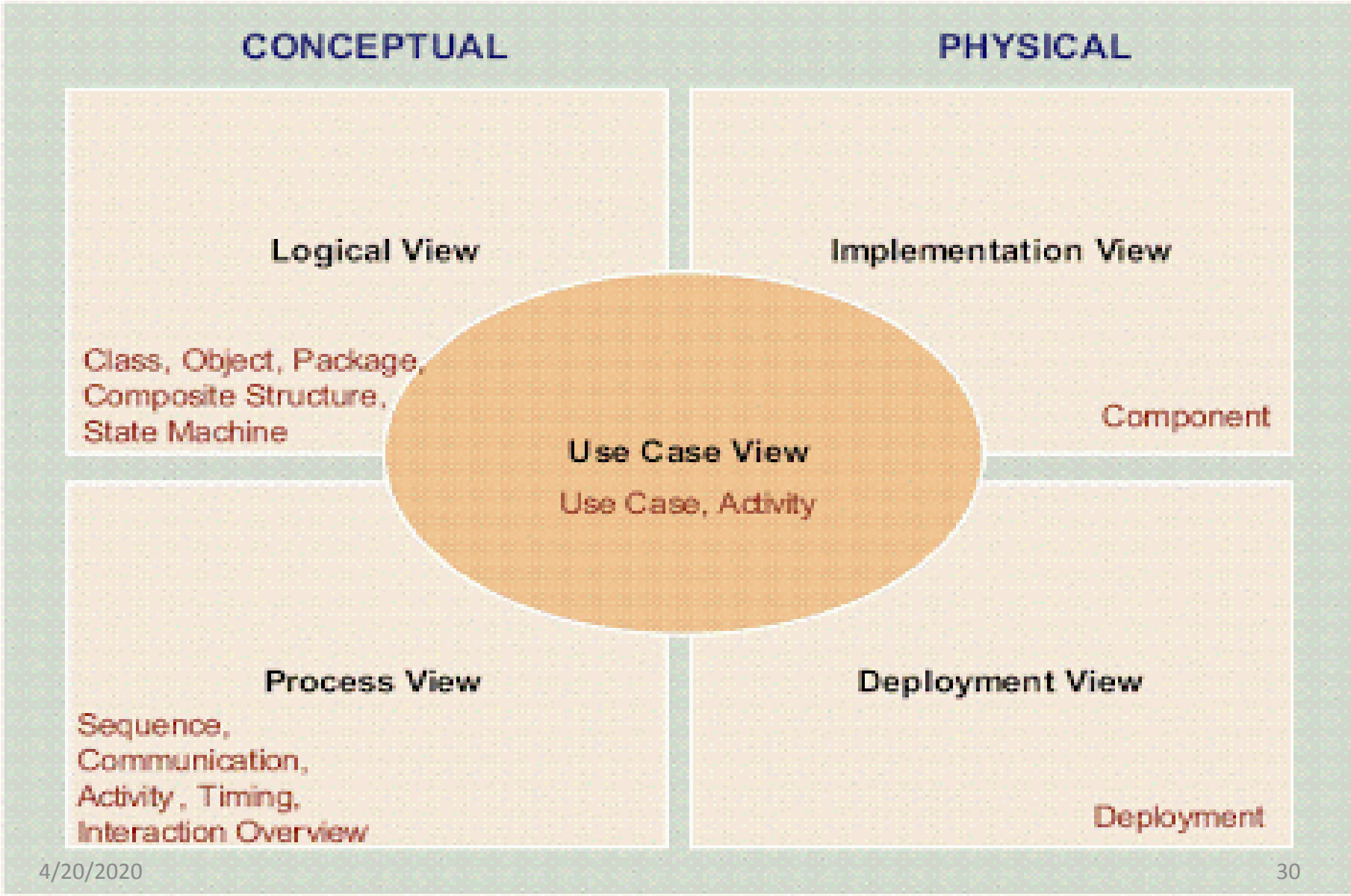
- The more you know, the more you realize you don’t know

//the more you learn, the more you are stupid





4+1 View


CONCEPTUAL
PHYSICAL
Logical View
Class, Object, Package,
Composite Structure,
State Machine
Implementation View
Component
Use Case View
Use Case, Activity
Process View
Sequence,
Communication,
Activity , Timing,
Interaction Overview
Deployment View
Deployment

4+1 view

- **4+1** is a view model used to describe the architecture of software-intensive systems, based on the use of multiple, concurrent views.
- The views are used to describe the system from the viewpoint of different **stakeholders**, such as end-users, developers, system engineer, and project managers.

4+1 view

- The four views of the model are logical, development, process and physical view, scenario/use case.
- In addition, selected use cases or scenarios are used to illustrate the architecture serving as the 'plus one' view. Hence, the model contains 4+1 views

4+1 view

- ***Logical view*:** The logical view is concerned with the functionality that the system provides to end-users. UML diagrams are used to represent the logical view, and include class diagrams, and state diagrams.
- ***Process view*:** The process view deals with the **dynamic aspects** of the system, explains the system processes and how they communicate, and focuses on the run time behavior of the system. The process view addresses concurrency, distribution, integrator, performance, and **scale + ability**, etc. UML diagrams to represent process view include the sequence diagram, communication diagram, activity diagram.

4+1 view

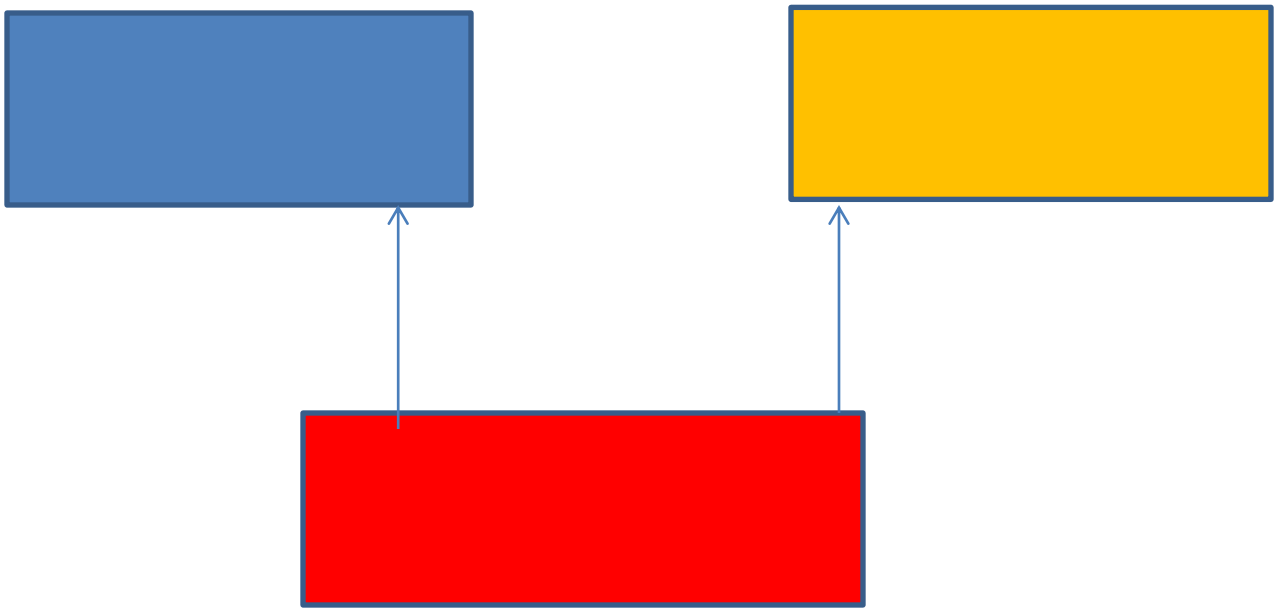
- ***Development* /implementation *view*:** This view illustrates a system from a programmer's perspective and is concerned with software management. It also is named view. It uses the Component diagram to describe system components and the Package diagram to represent the development view.
- ***Physical /Deployment view***: The physical view depicts the system from a system engineer's point of view. It is concerned with the topology of software components on the physical layer as well as the physical connections between these components. UML diagrams used to represent this view include the deployment diagram.

4+1 view

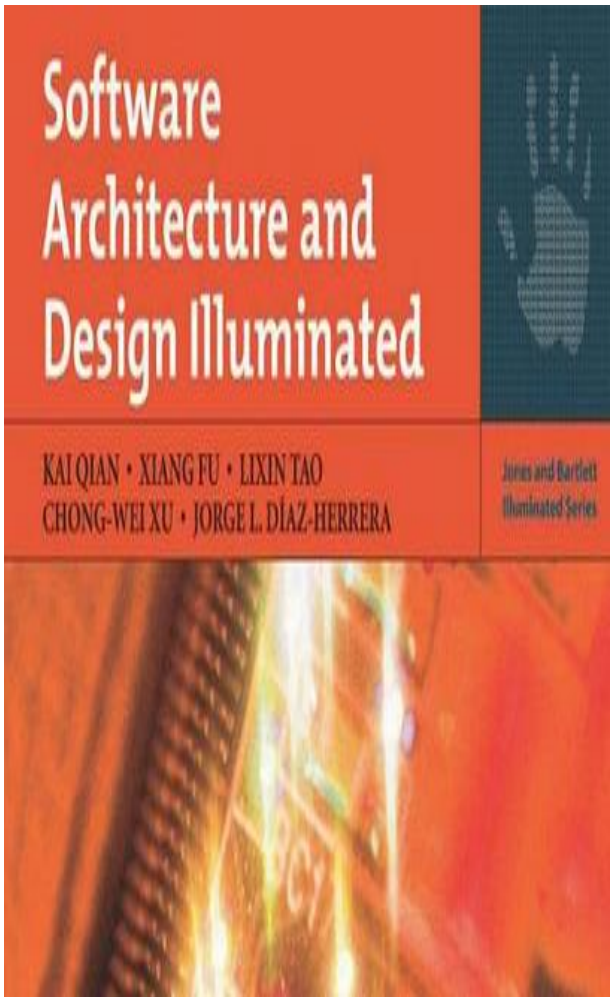
- ***Scenarios/Use cases***: a small set of use cases or scenarios, which become a fifth view, is used to illustrate the description of an architecture .
- The scenarios describe sequences of interactions between objects and between processes. They are used to identify architectural elements and to illustrate and validate the architecture design. They also serve as a starting point for tests of an architecture prototype. This view is also known as the **use case view**.

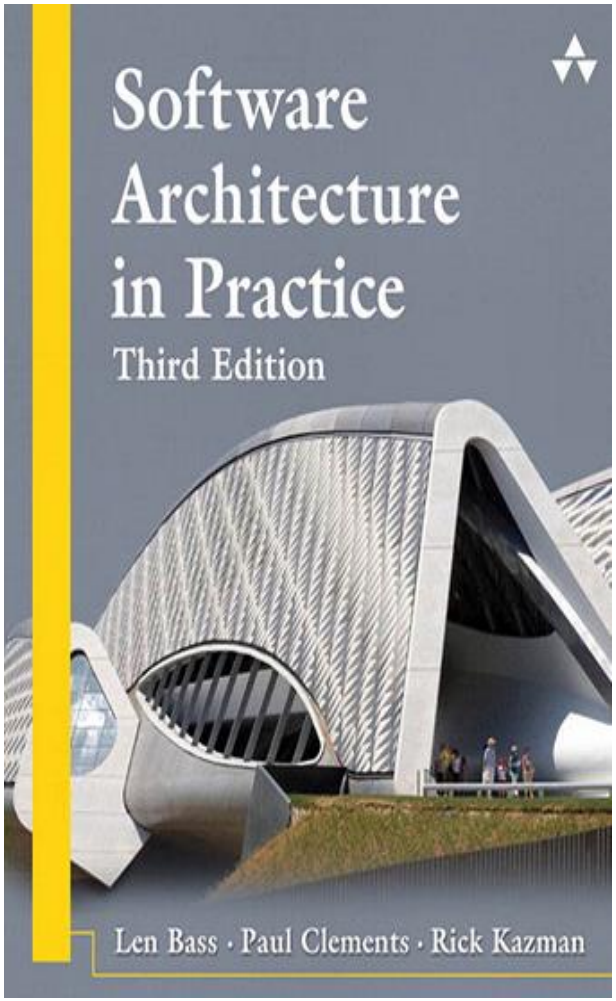
Many in 1

https://www.visual-paradigm.com/guide/



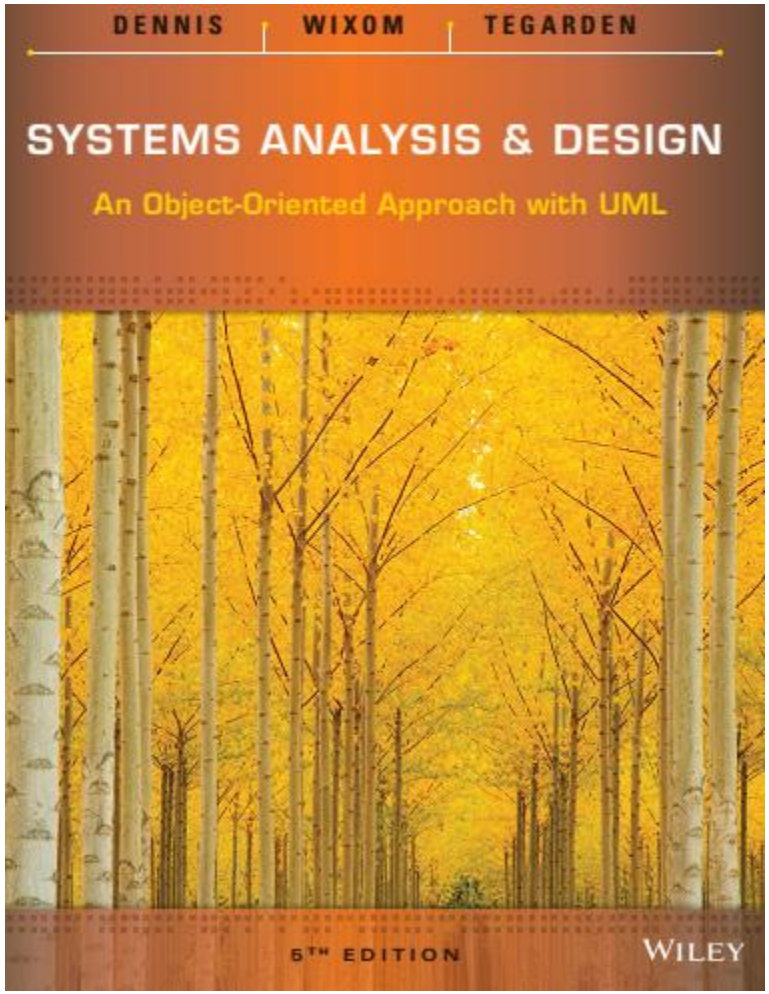
Part I: Design architecture



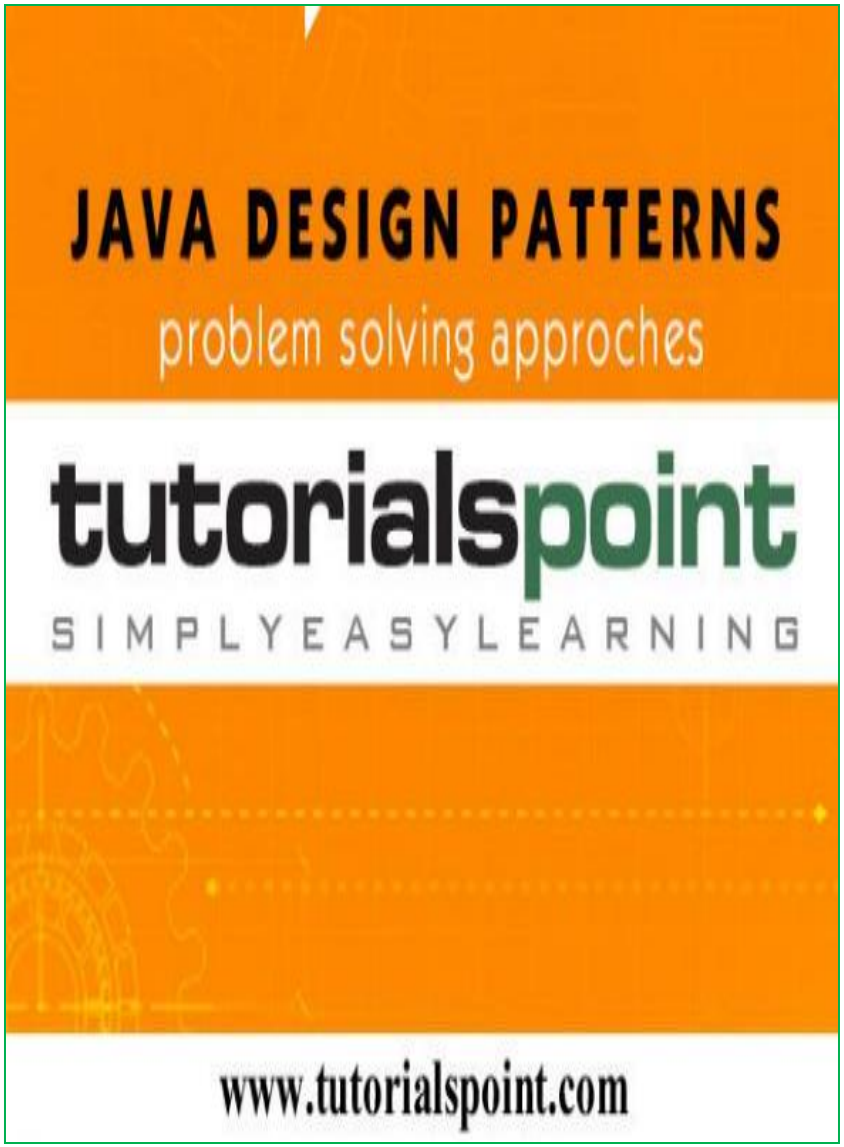


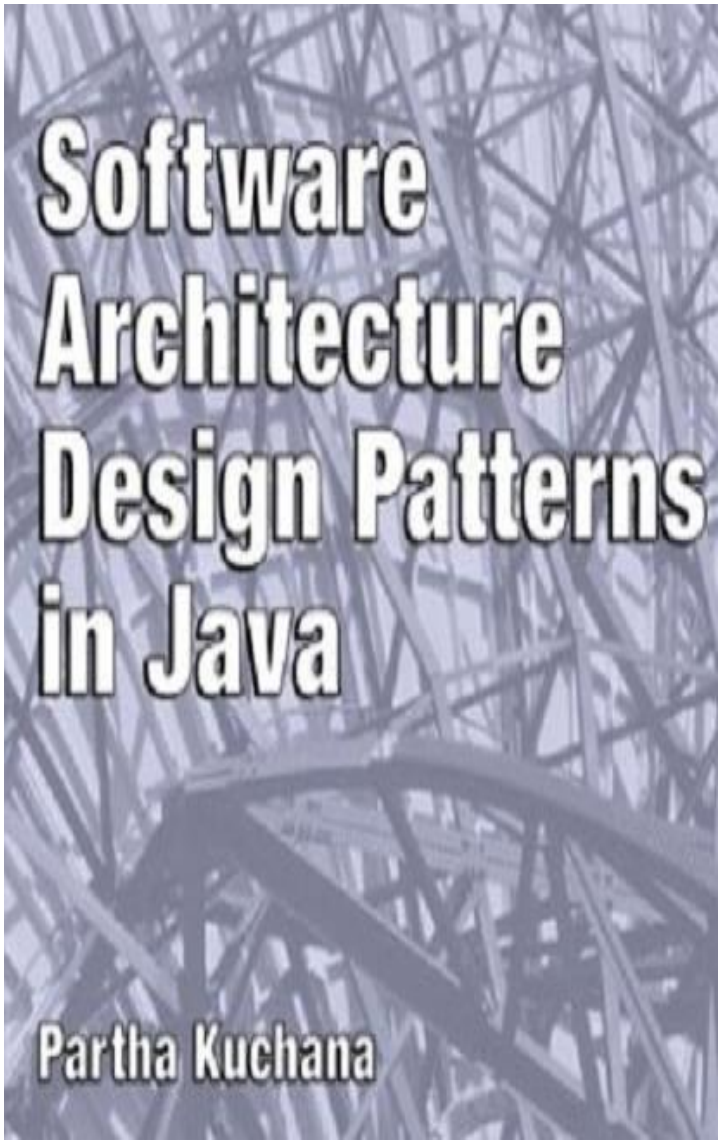
Part II: Design Layers

- Design layers
- Design package
- Design class
- Design method



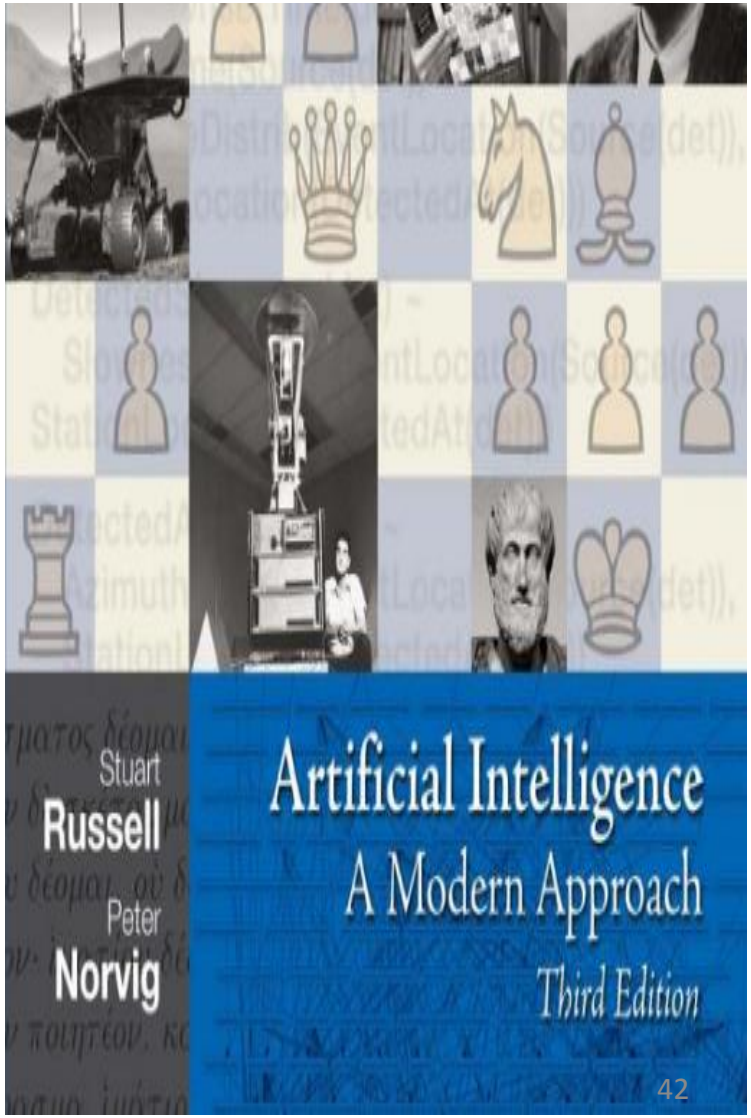
Part III: Design patterns





Part IV: Intelligent systems

- Artificial intelligence
- Machine learning
- Design intelligent systems



Assessment

- 20%: 4 Assignments
- 20%: Big Project (group working)
- 60%: Final examination (writing)