SWEN30006
Software Design and Modelling

Object-Oriented Design Models

Textbook: Larman Chapter 15, 16, & 17

"A mathematician is a device for turning coffee into theorems."

—Paul Erdös

Learning Objectives

On completion of this topic you should be able to:

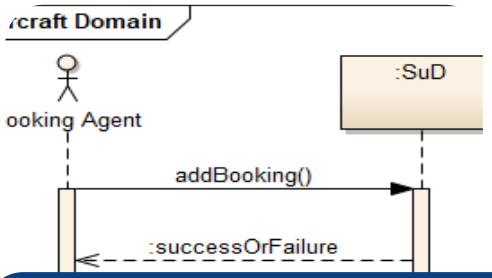
- Be aware of the role of Object-Oriented design
- Be aware of the primary static and dynamic design models
- Understand the underlying relationship between *domain* and *design* models
- Have an awareness of the concepts of responsibility and responsibility-driven design

Revisited: Object-Oriented Analysis, Design, and Development

Problem domain
- Concepts
- Relationships

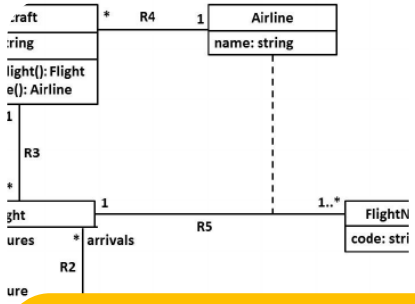
Use Cases

Text stories
Week 1



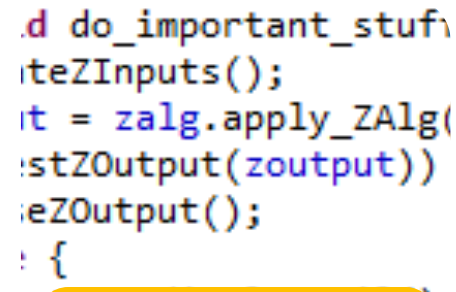
OO Domain Models
- Static: Domain Class Diagram
- Dynamic: System Sequence Diagram

Conceptual Classes
Week 2



OO Design Models
- Static: Design Class Diagram
- Dynamic: Design Sequence Diagram

Software Classes
Week 3



OO Implementation

Classes in Programming
Week 3

Analysis, Design, and Implementation

Object-oriented Domain Models

- ***Analysis***: An investigation of the problem & requirements.
- ***Object-oriented software analysis*** emphases finding and describing objects and concepts in the problem *domain*.

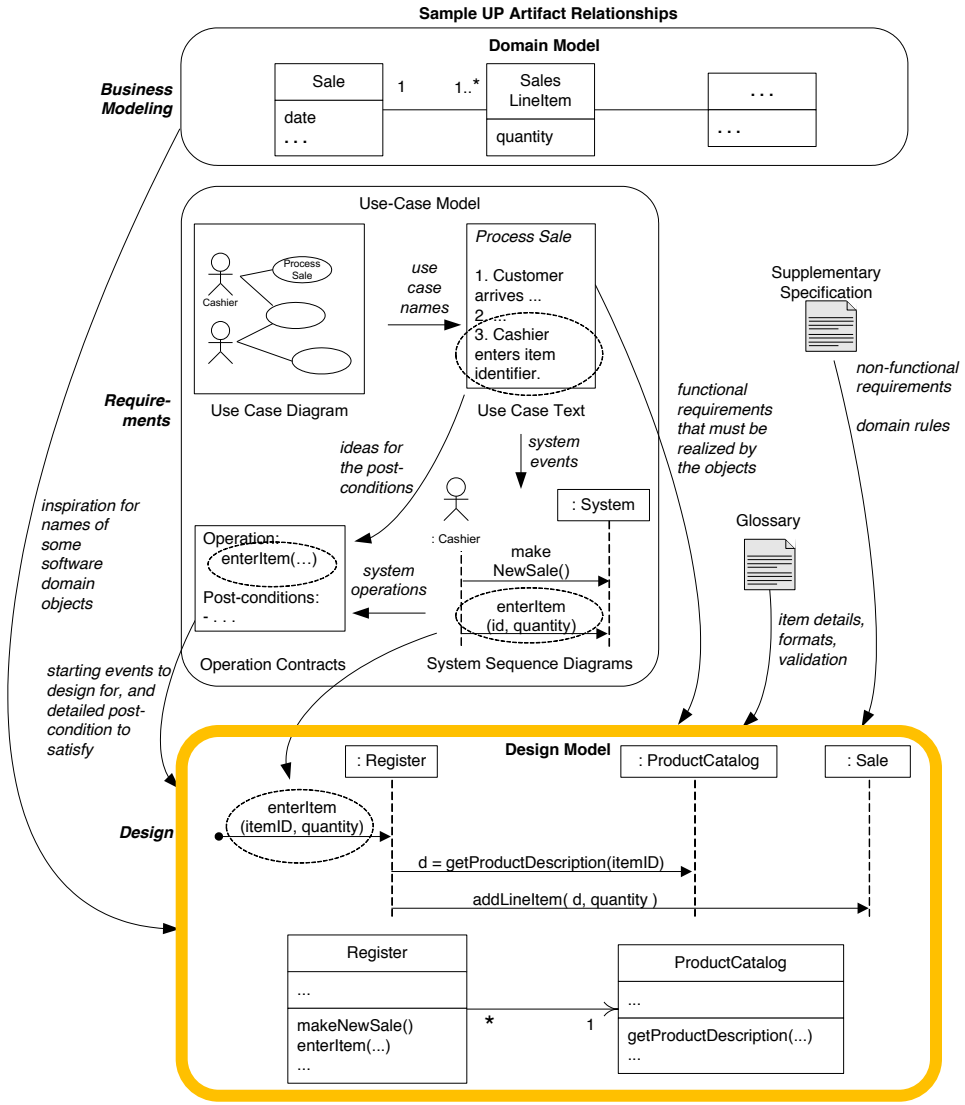
Object-oriented Design Models

- ***Design***: A *conceptual solution* to a problem that meets the requirements.
- ***Object-oriented software design*** emphasises defining *software objects* and their collaboration.

Object-oriented Implementation

- ***Implementation***: A *concrete solution* to a problem that meets the requirements.
- ***Object-oriented software implementation***: implementation in object-oriented languages and technologies.

Relationship of OO Design Models in UP



- UP artifacts influence the OO design
  - Use case text describes functional requirements that must be realized in the design models
  - Domain models provides inspiration for names of software objects in the design models
  - System Sequence Diagram indicates an interaction between users and system, i.e., user inputs and system responses

Object-Oriented Software Design

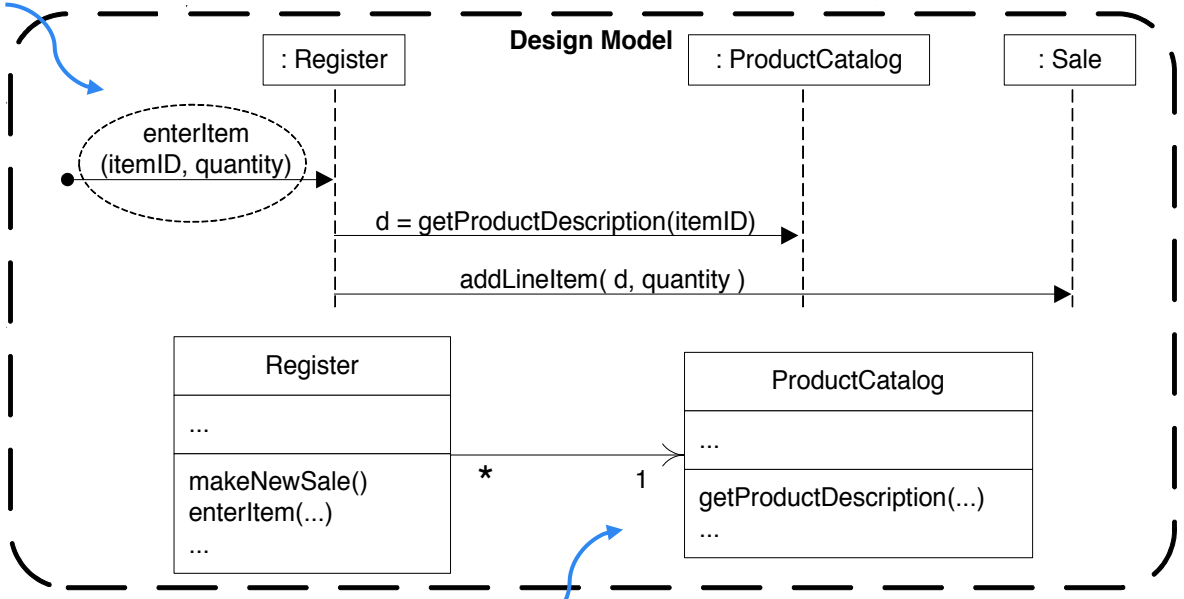
Object-oriented software design is a process of creating a conceptual solution by defining *software objects* and their collaboration.

- Structure and connectedness
  - Levels, including Architecture
- Interfaces: methods, data types, and protocols
  - External and Internal
- Assignment of responsibilities
  - Principles and Patterns

Modelling the conceptual solution

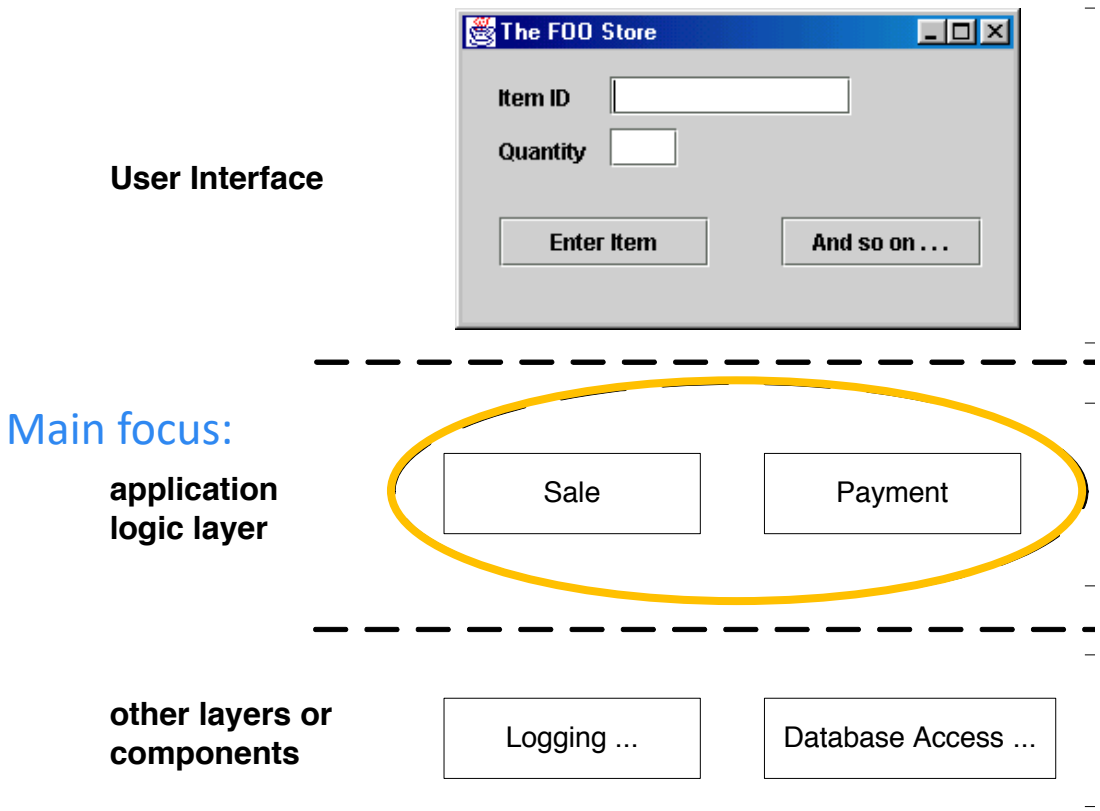
- Static models: Design Class Diagram
- Dynamic models: Design Sequence Diagram
  - Design sequence diagram is one of UML interaction diagrams.
    Another interaction diagram, i.e., communication diagram is not focused in this lecture

Design Sequence Diagram



Design Class Diagram

Layers in OO Design

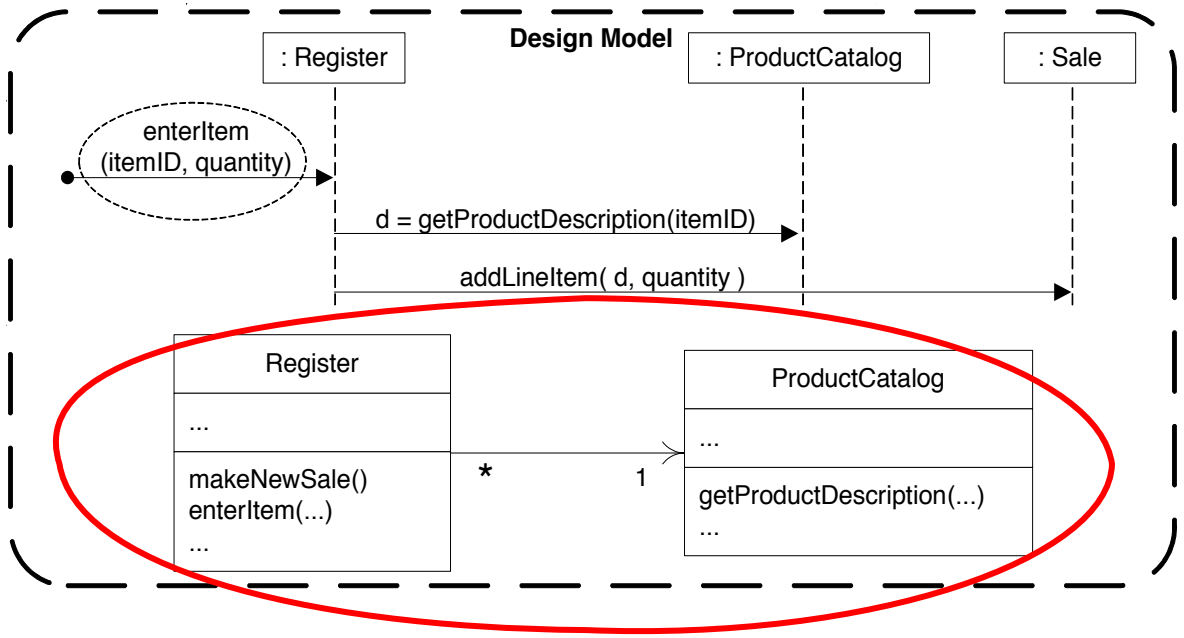


We focus on the core application logic layer

- OO design of core logic layer similar across technologies.
- Essential OO design skills are applicable to other layers.
- Other layers tend to be technology/platform dependent.
- Design approach/patterns for other layers tends to be technology constrained and changeable.

THE UNIVERSITY OF MELBOURNE

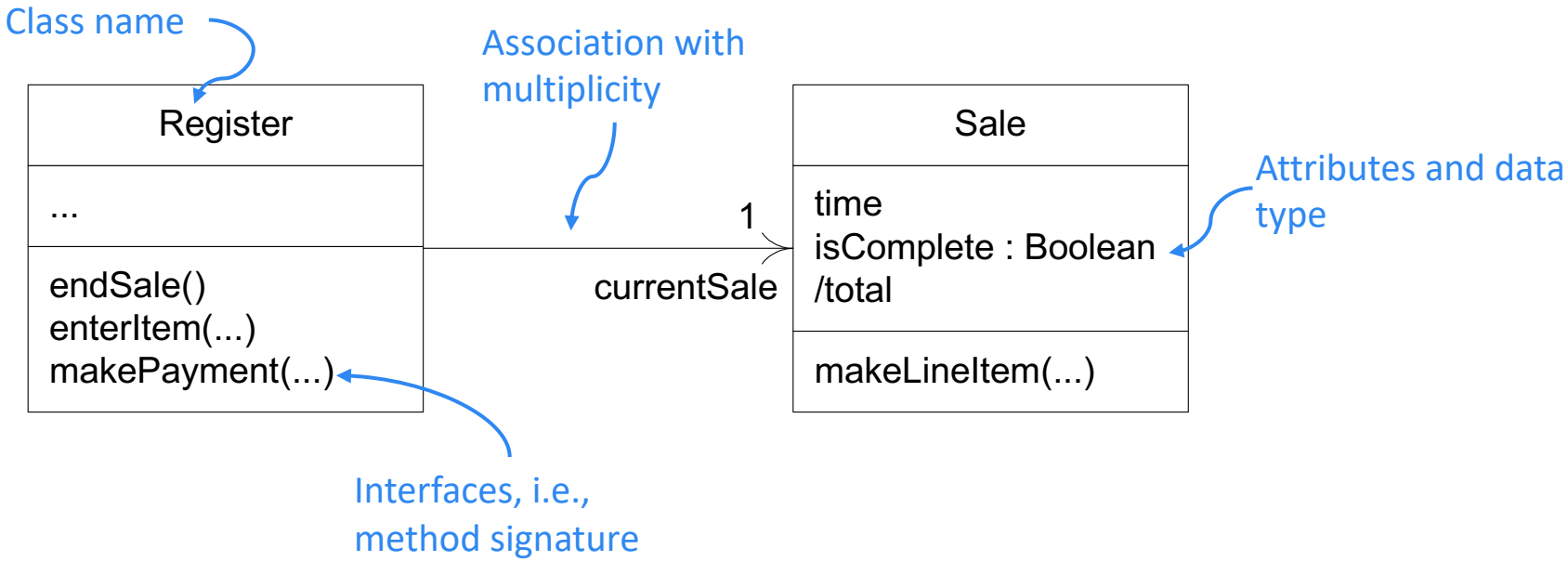
Static OO Design models:
Design Class Diagrams



Textbook: Larman Chapter 16 & (partial) 17

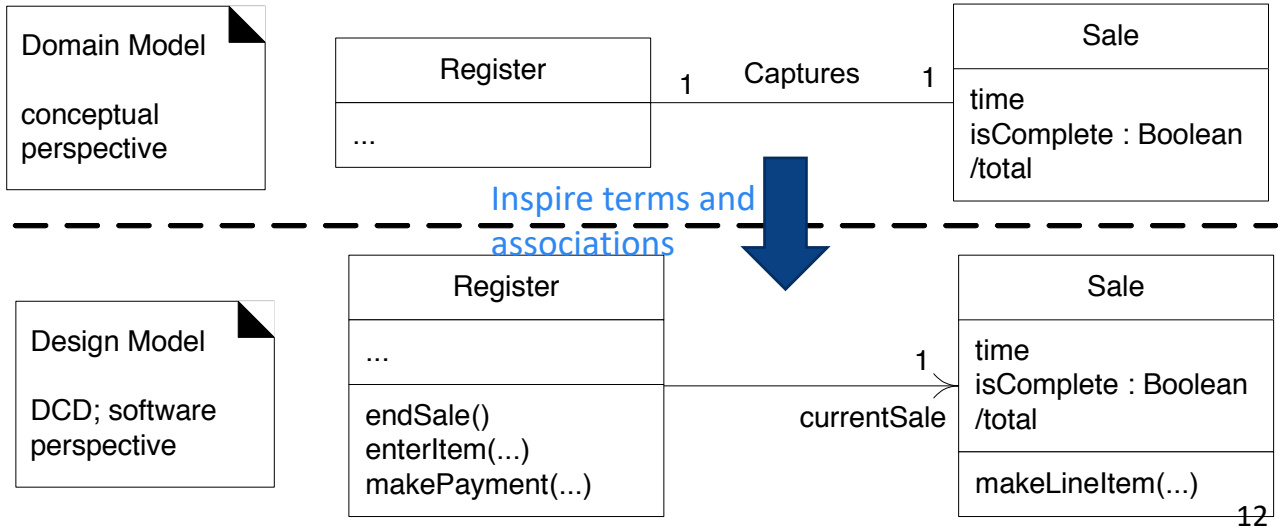
Static Design Models

- **Definition**: A static design model is a representation of software objects which define class names, attributes, and method signatures (but not method bodies)
  - Use UML Class Diagram to visualize the model
- Design Class Diagram
  - illustrates class names, interfaces, and the associations of software objects



Domain Models *vs* Design Models

- Both models use the same UML diagram, **but** their focuses are different
  - Domain model focuses on a *conceptual perspective of the problem domain*
    - What are the nothworthy *concepts*, *attributes*, and *associations* in the problem domain
  - Design model focuses on an *implementation perspective* of software
    - What are the *roles* and *collaborations* of software objects
- Domain models **inspire** the design of software objects
  - To ***reduce the representational gap*** between how stakeholders conceive the domain and its representation in software



Designing Software Objects

- One of popular ways to design software objects is using **Responsibility-Driven Design** or RDD
- RDD focuses on assigning *responsibility* to software objects
  - Responsibility: The obligations or behaviours of an object in terms of its role
- Two types of responsibility:

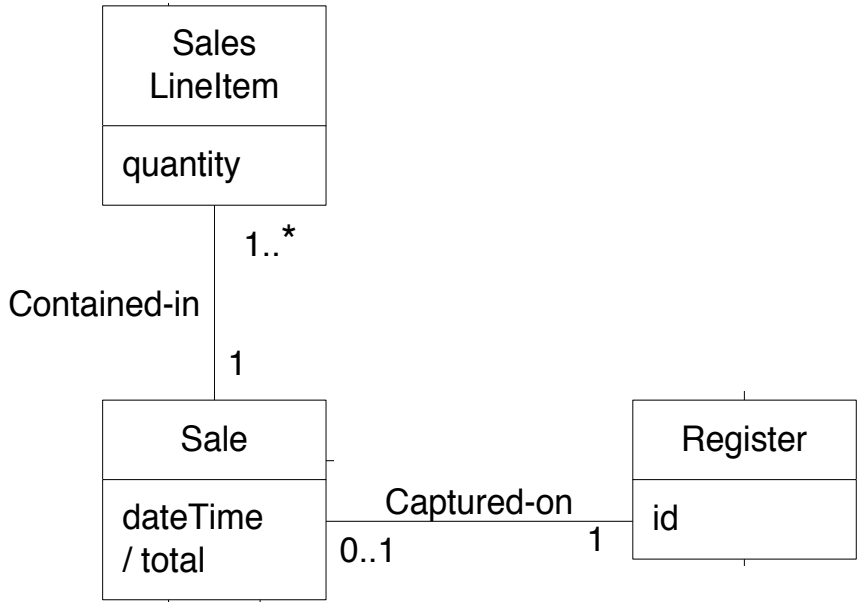
- **Knowing** responsibilities include:
  - knowing about private encapsulated data
  - knowing about related objects
  - knowing about things it can derive or calculate

- **Doing** responsibilities include:
  - doing something itself, such as creating an object or doing a calculation
  - initiating action in other objects
  - controlling and coordinating activities in other objects

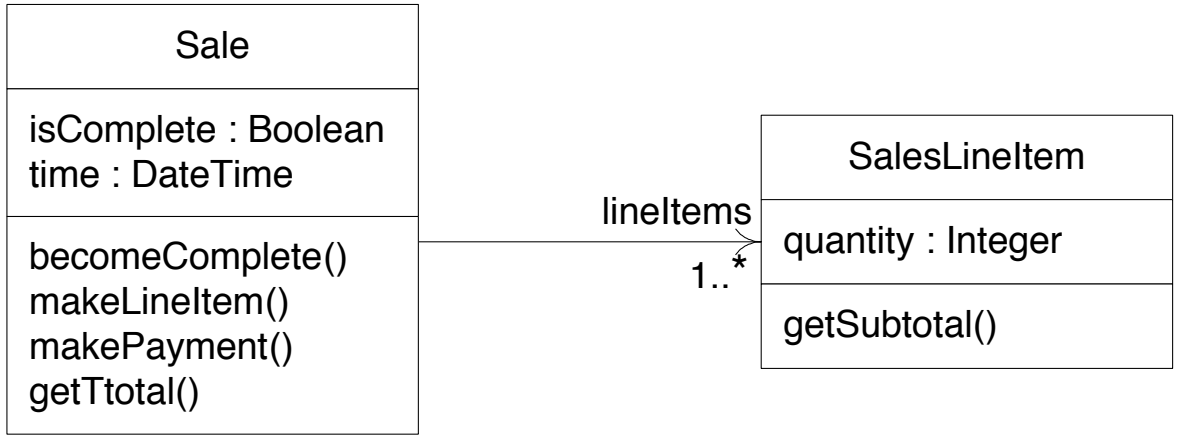
- RDD also includes the idea of *collaboration*
  - Responsibilities can be implemented by means of methods that either act alone or collaborate with other methods and objects

Example: POS

- In the domain model, Sale has the total attribute
  - Let's declare: A sale class is responsible for *knowing* its total (knowing)
    → A Sale class will have 'getTotal' method
- Based on the domain model and use case text, Sales Line Items will be contained in Sale and will be created when a product is entered
  - Let's declare: A sale class is responsible for *creating* a Sale Line Items (doing)
    → A Sale class will have 'makeLineItem' method
- The sale total is derived from the sub total of sale line item
  - 'getTotal' of Sale is *collaborating* with 'getSubtotal' of SaleLineItem



Partial domain model



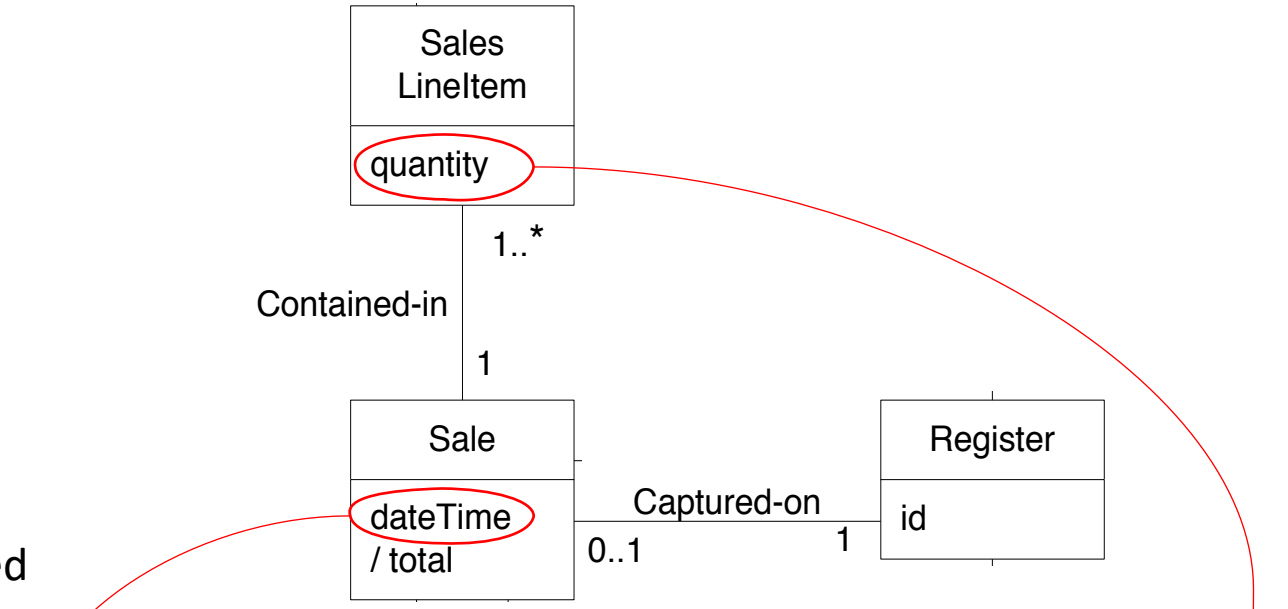
Partial design model

Example: POS

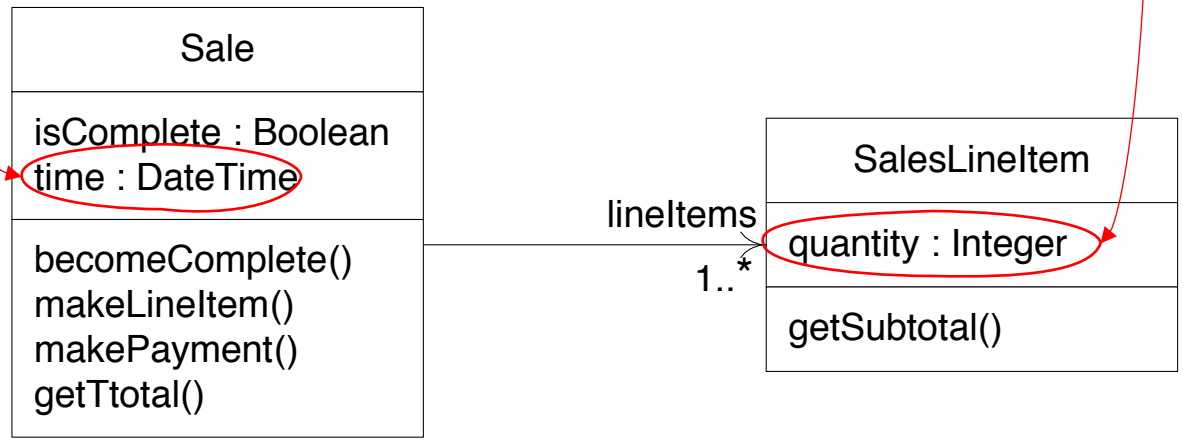
Some Remarks:

- A responsibility in RDD is not the same thing as a method—it's an abstraction—but methods fulfill responsibilities

- The domain model often inspires responsibilities related to “knowing”, achieving ***low representational gap***
  - Ex: the domain model Sale class has a *time* attribute
    → a software Sale class knows *time*
  - Ex: the domain SaleLineItem class has a *quantity* attribute
    → a software SaleLineItem class knows *quantity*



Partial domain model

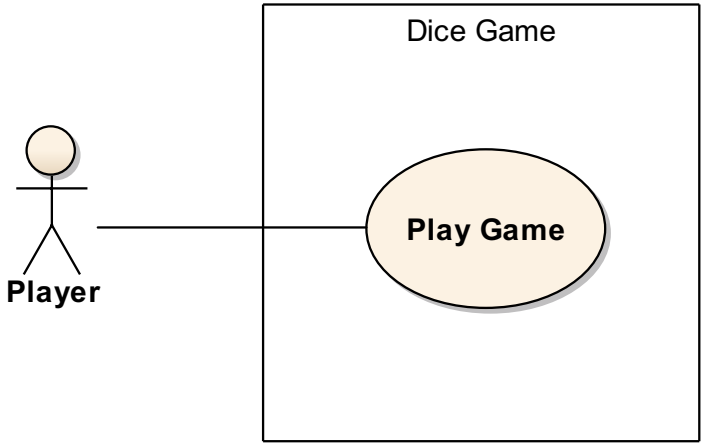


Partial design model

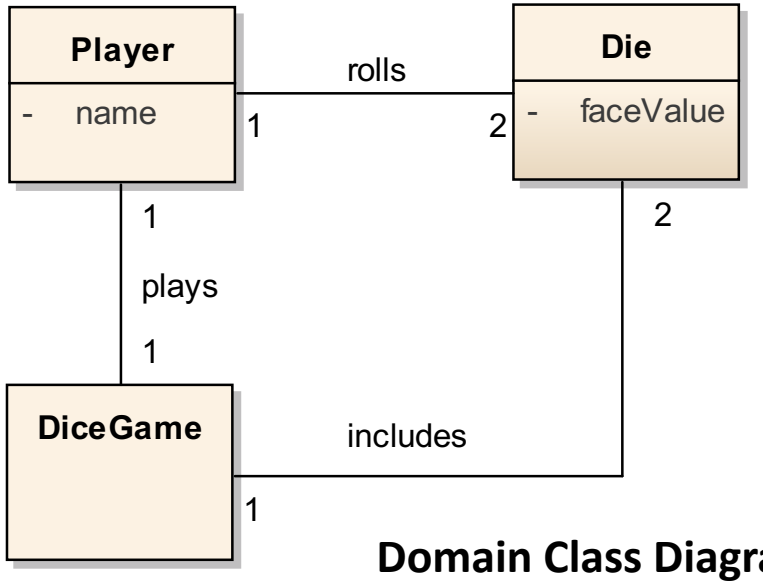
Exercise: Lucky 7

Use Case:

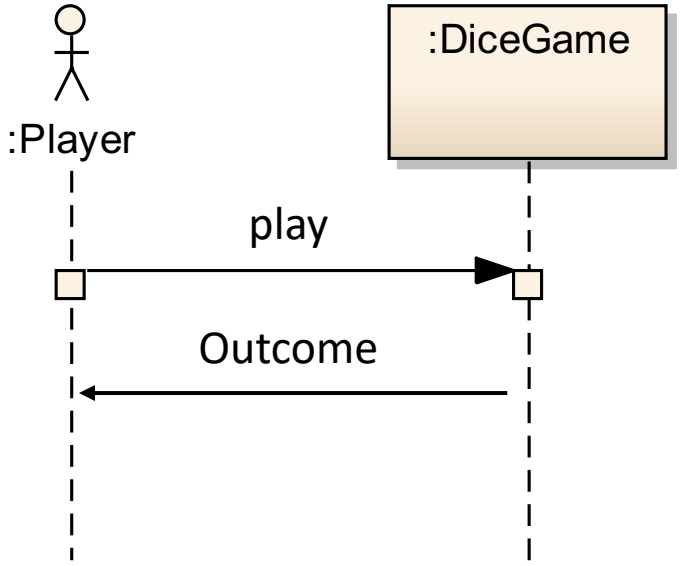
"Player play a game by rolling 2 dice. System presents the outcome (Win if the dice face value totals 7; Otherwise lose)."



Use Case Diagram



Domain Class Diagram

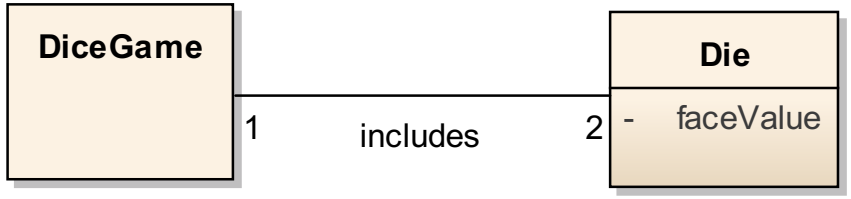


System Sequence Diagram

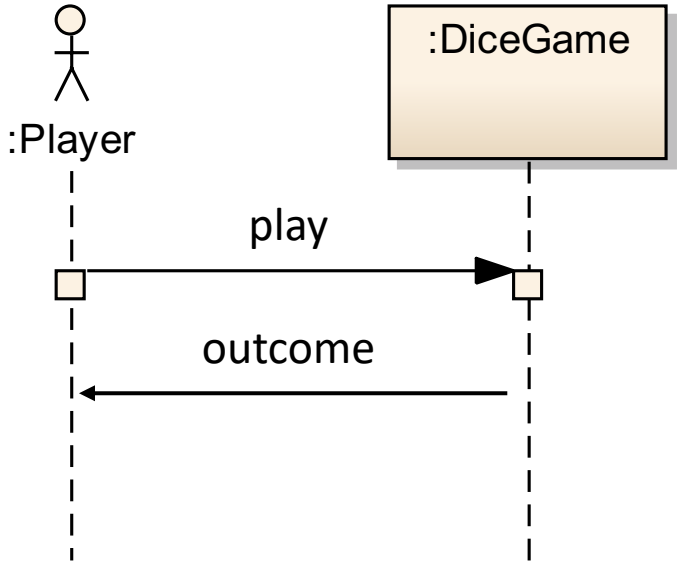
Exercise: Lucky 7

Create a design class diagram

Partial Domain Class Diagram



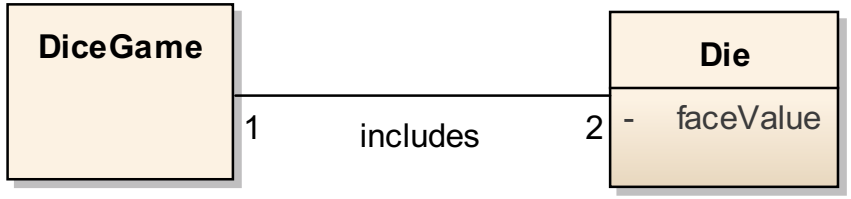
System Sequence Diagram



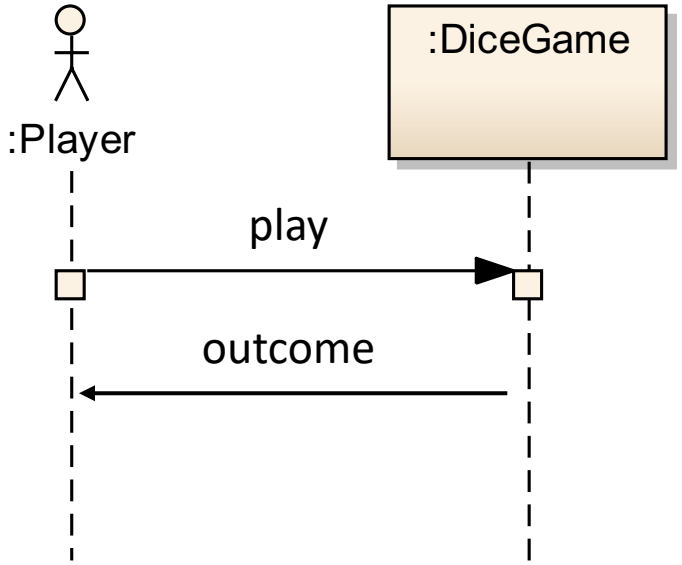
Exercise: Lucky 7

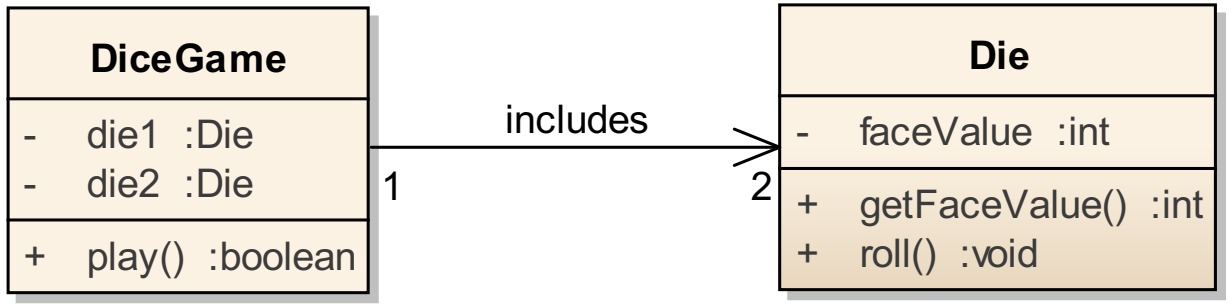
Create a design class diagram

Partial Domain Class Diagram



System Sequence Diagram

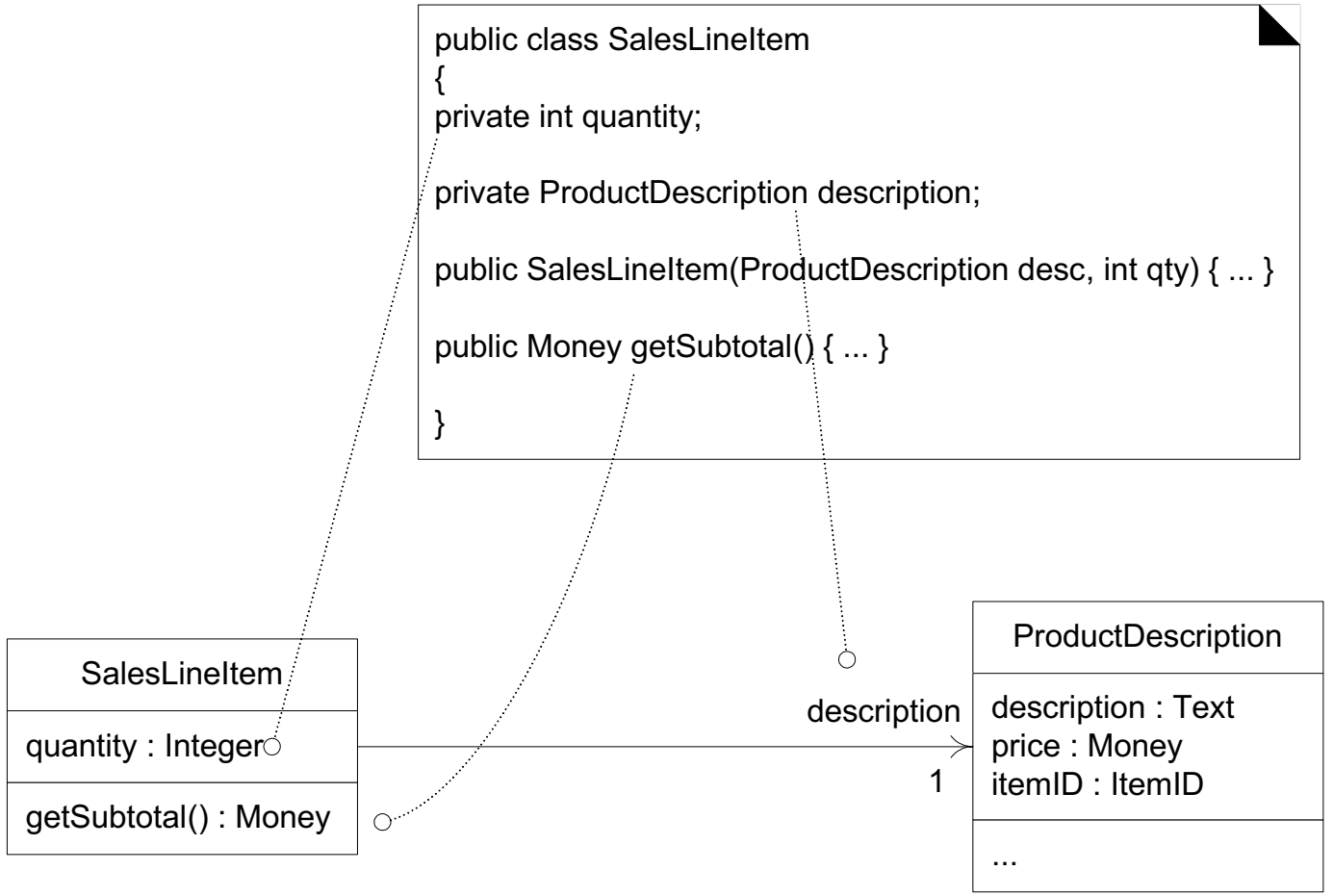




Design Class Diagram

Design Class Diagram To Code

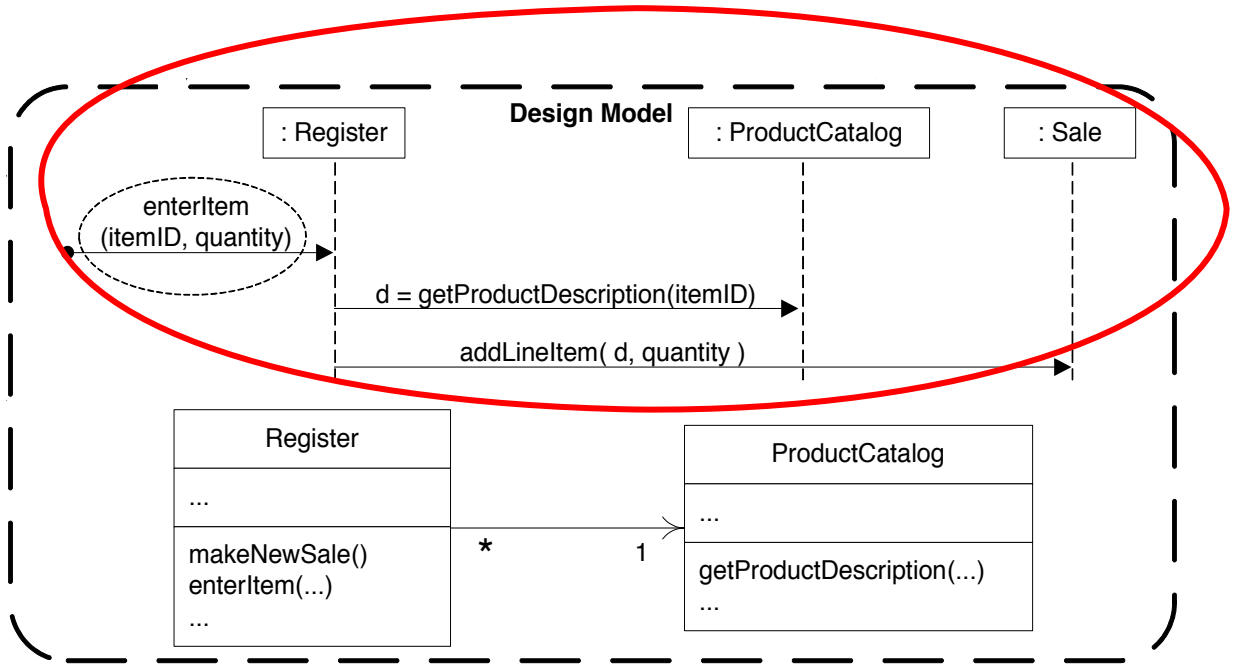
- OO Designs provide information necessary to generate the code
- OO Design can be mapped to OO Programming
  - Attributes in Design models = Java fields
  - Method signatures in Design models = Java methods
  - The Java constructor is derived from the create responsibility



**More detail is available in Workshop 3 supplementary materials*

THE UNIVERSITY OF MELBOURNE

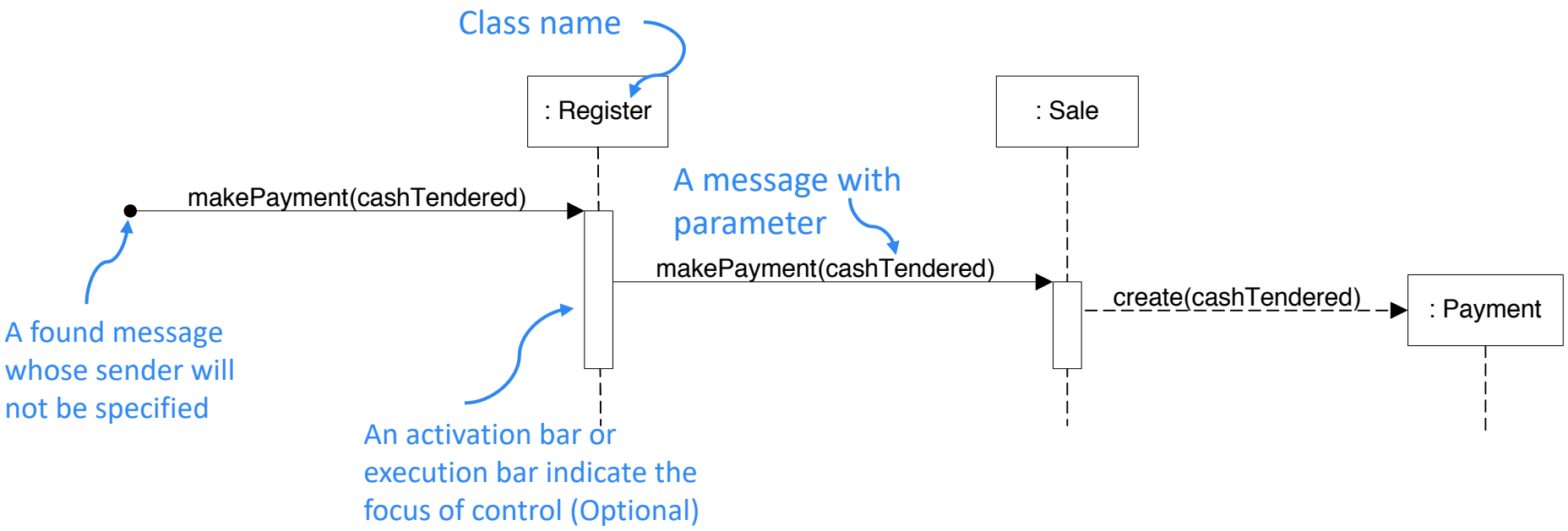
Dynamic OO Design models:
Design Sequence Diagrams



Textbook: Larman Chapter 15

Dynamic Design Models

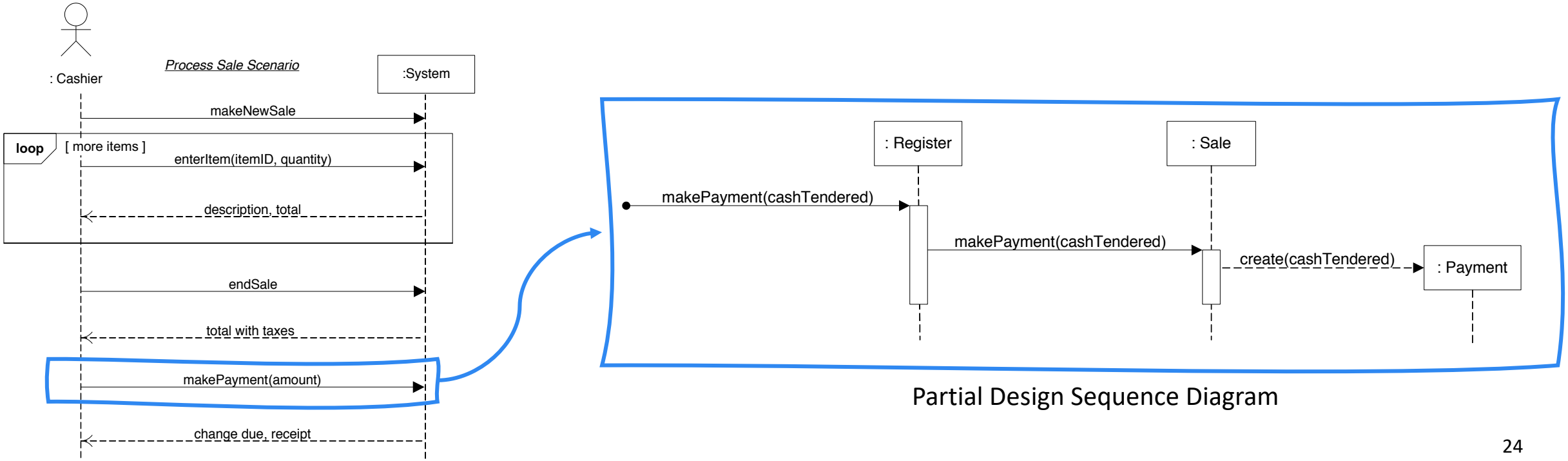
- **Definition**: A dynamic design model is a representation of how software objects interact via messages
  - UML Sequence and communication* diagrams are commonly used to visualize the models
- Design Sequence diagram
  - Illustrates sequence or time ordering of messages sent between software objects



*Communication diagram is not focused in this lecture

System Sequence Diagram *vs* Design Sequence Diagram

- System Sequence Diagram treats the system as a black box, focusing on the interaction between actors and the system
- Design Sequence Diagram illustrates the behaviors within the system, focusing on the interaction between software objects

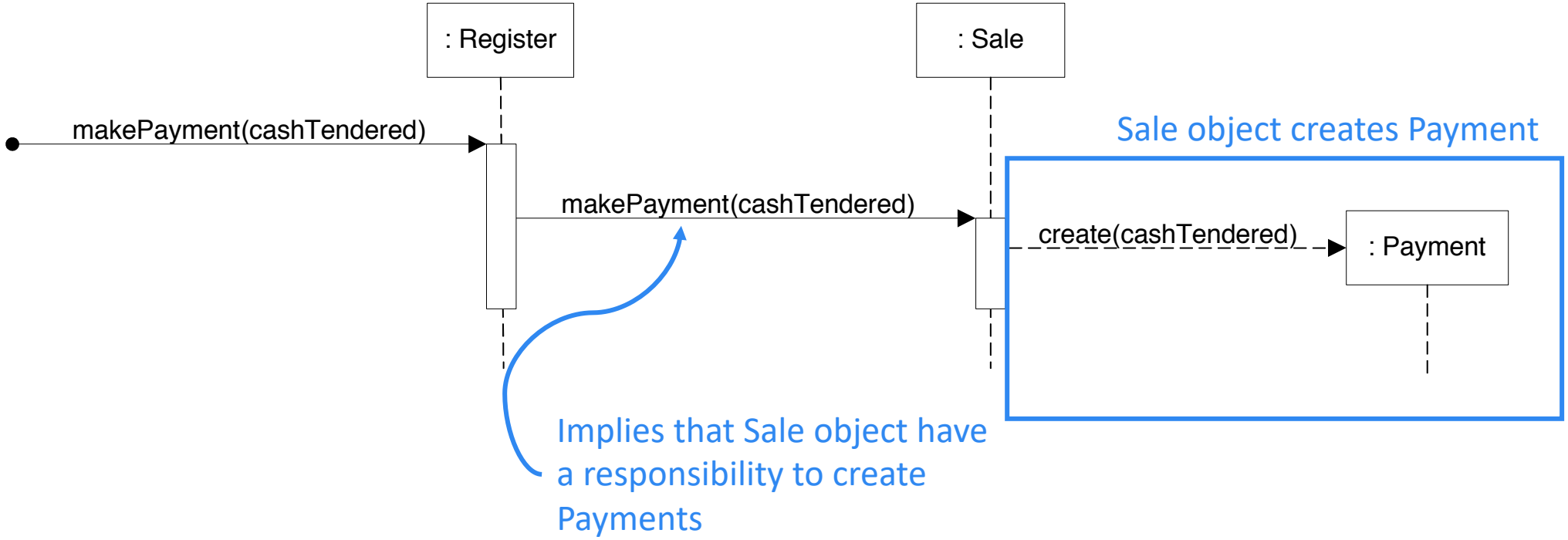


System Sequence Diagram

Partial Design Sequence Diagram

RDD & Design Sequence Diagram

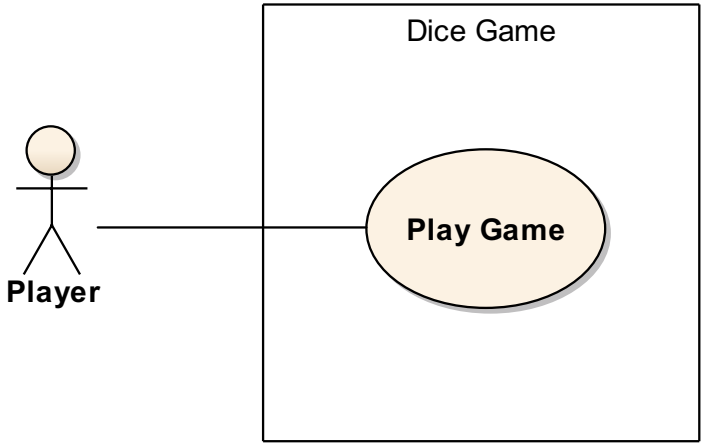
- Design Sequence Diagram helps in better realizing responsibilities of software objects
  - Concerning the responsibility assignment when drawing the diagram



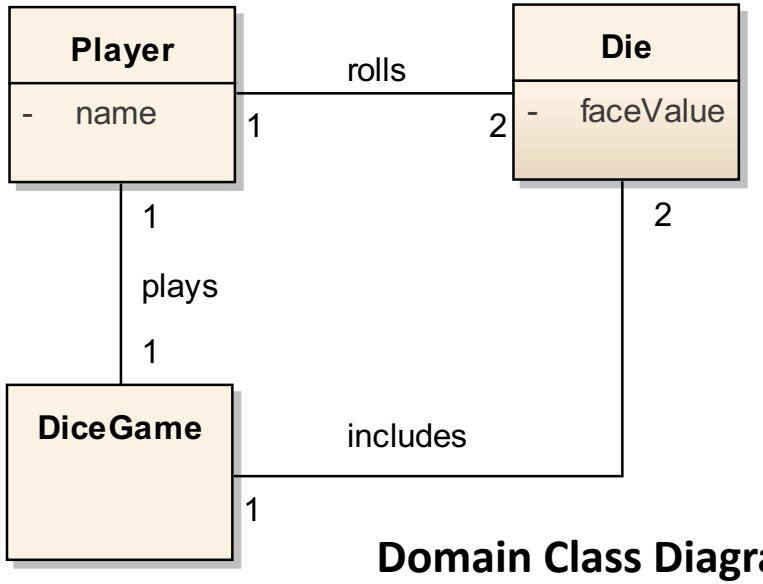
Exercise: Lucky 7

Use Case:

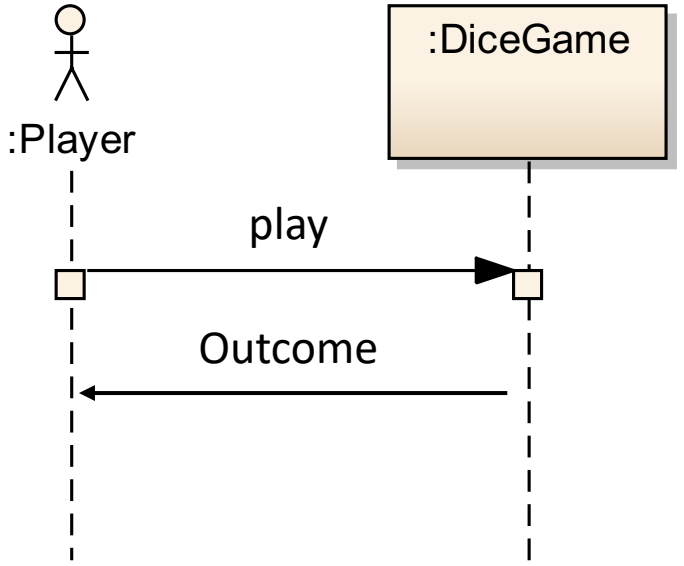
"Player play a game by rolling 2 dice. System presents the outcome (Win if the dice face value totals 7; Lose otherwise)."



Use Case Diagram



Domain Class Diagram

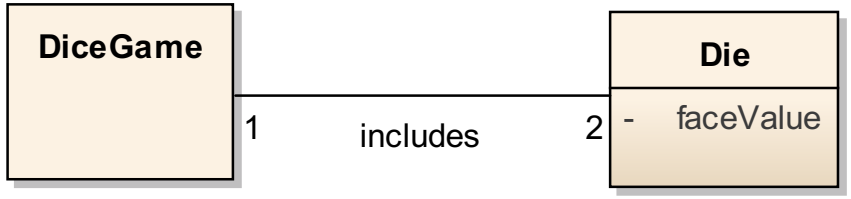


System Sequence Diagram

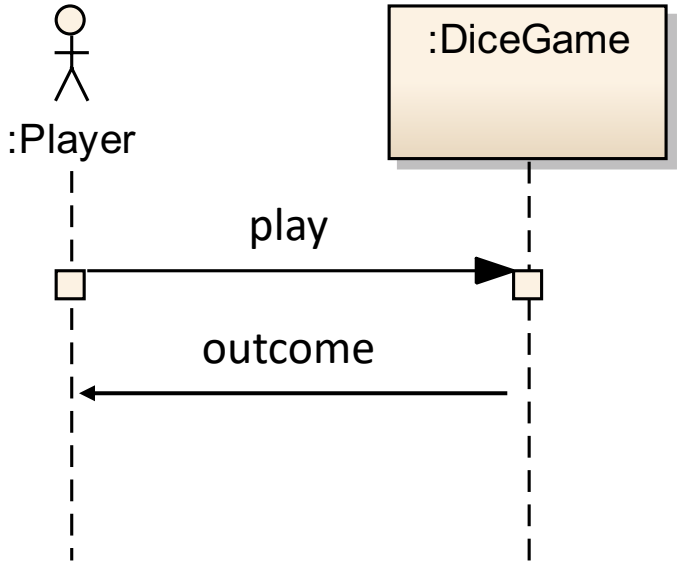
Exercise: Lucky 7

Create a design sequence diagram

Partial Domain Class Diagram



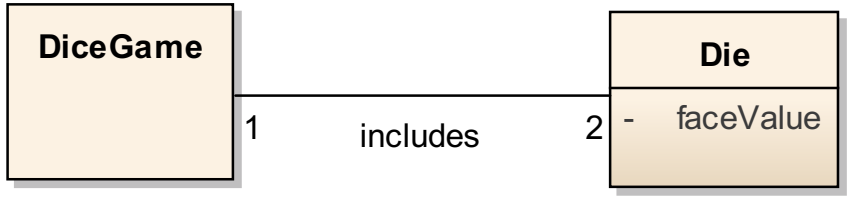
System Sequence Diagram



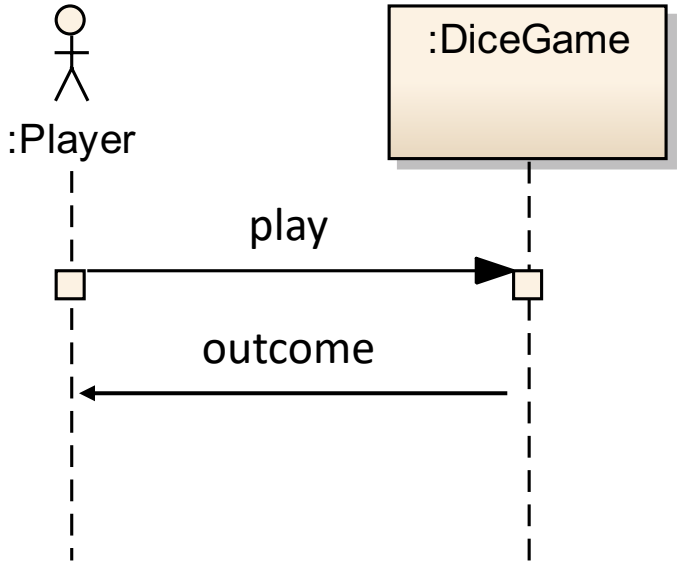
Exercise: Lucky 7

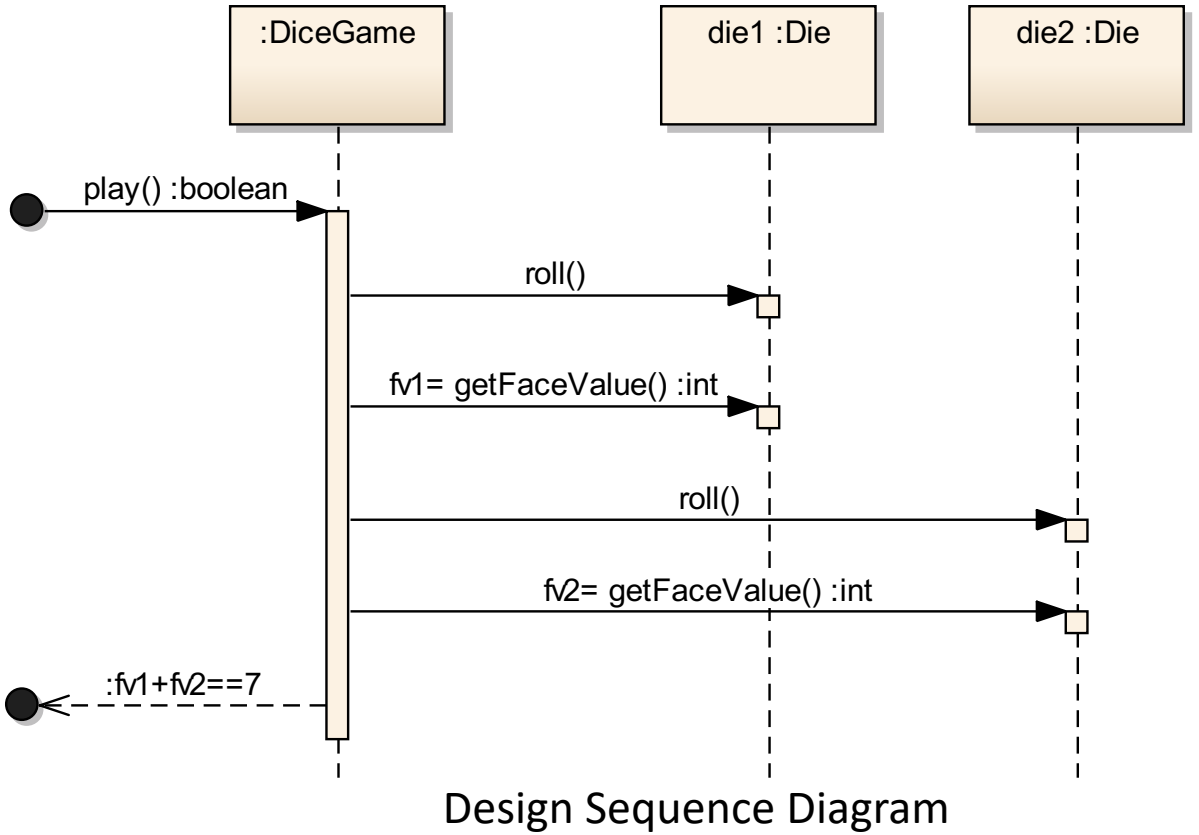
Create a design class diagram

Partial Domain Class Diagram



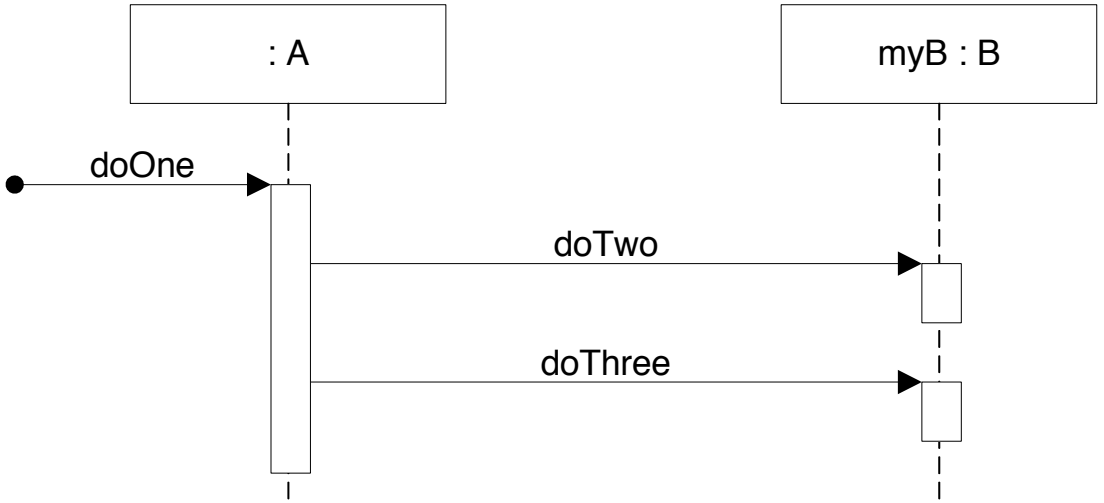
System Sequence Diagram





Design Sequence Diagram

Design Sequence Diagram To Code




```
public class A {
    private B myB = new B();
    public void doOne()
    {
        myB.doTwo();
        myB.doThree();
    }
    // ...
}
```

Summary & Remarks

- **Object-Oriented Design** aims to create a conceptual solution by defining *software objects* and their collaboration.
- **Static Design Model (Design Class Diagram)** illustrates software objects which define class names, attributes, and method signatures (interfaces)
- **Dynamic Design Model (Design Sequence Diagram)** illustrates how software objects interact via messages
- **Responsibility-Driven Design** is a metaphor of designing software objects
  - By seeing software objects as similar to people with responsibilities who collaborate with other people to get work done
- To achieve **low representational gap**, domain models often inspires the class names, attributes, and responsibilities of software objects

Lecture Identification

Lecturer: Patanamon Thongtanunam

Semester 2, 2020

These slides include materials from:

Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third Edition, by Craig Larman, Pearson Education Inc., 2005.