OOAD-Lecture 03
By

Dr. Taimoor Khan

Software Engineering vs OOAD:

- Software Engineering
  - What needs to be done?
  - Who does it?
  - When is it done?

- OOAD
  - What needs to be done?
  - What classes are responsible for doing it?
  - What classes have the job of seeing it gets done?

UML vs Thinking in Objects:

- Most of our UML will be pencil-and-paper

- This course is about OOAD –
  Object-Oriented Analysis & Design not UML

- **What classes/objects?**
- **What responsibilities to what classes?**

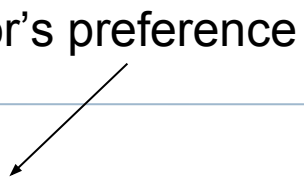
  "Responsibility-driven design"
- *Patterns* as guidance. These are *named* solution formulas

What is UML?

- The *Unified Modeling Language* is a visual language for *specifying*, *construction* and *documenting* the **artifacts** of systems.

- Has a lot more than software design tools
- Lots of software for drawing pictures

How to Apply UML?



- **As a sketch**: informal, hand-drawn documents, used for exploration
- **As a blueprint**: detailed design documents, developed by tools that either *forward-* or *reverse engineer code.*
  - *forward:* tool takes a picture and produces executable code (mostly stubs).
  - *reverse*: tool takes executable code and draws a picture.
- **As a Programming Language**: Tools produce complete executables.
- **As an Agile Programming**: UML as a sketch.

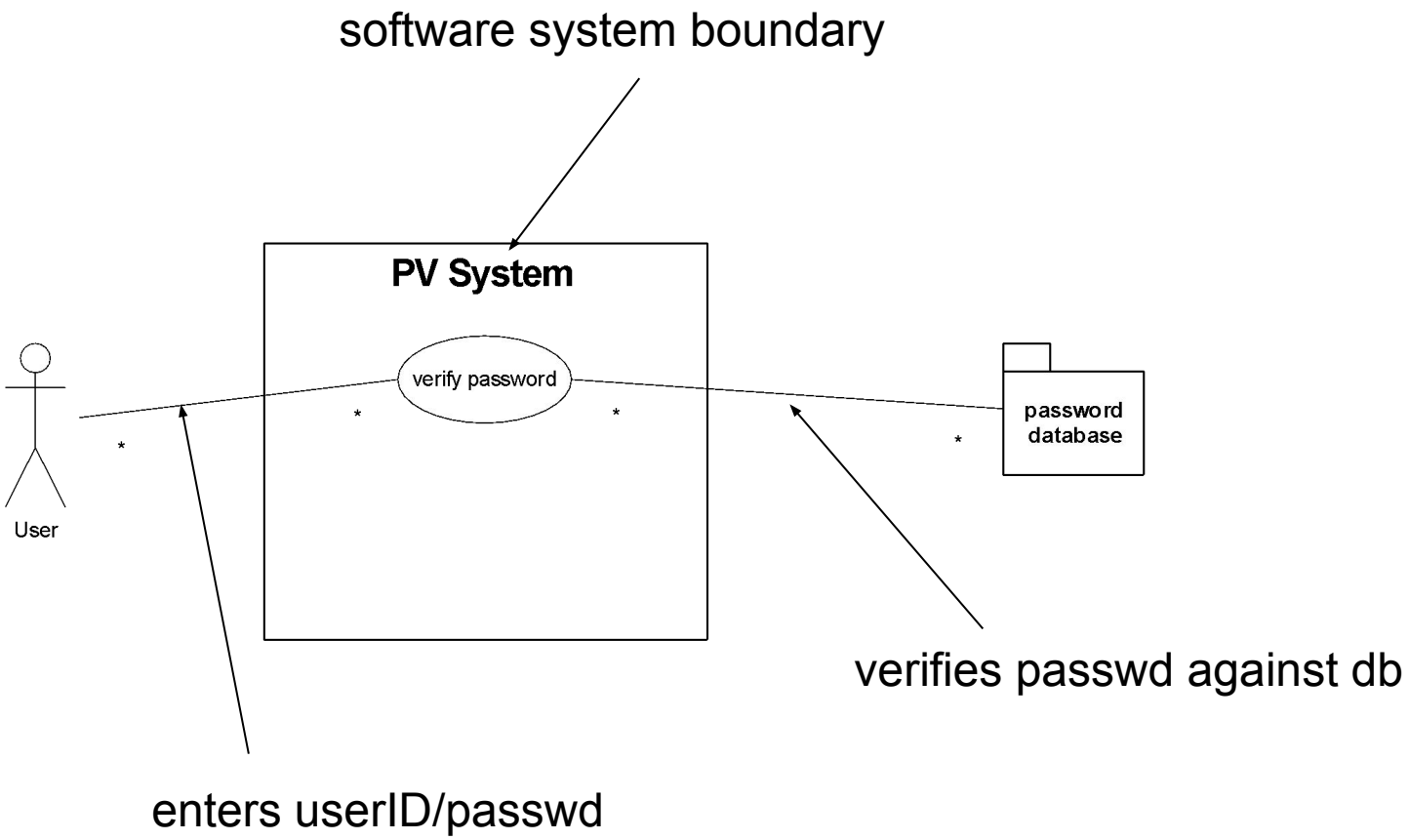
Use Cases

- Where ever you find OOD, you also find *Requirements Analysis*
  - What does the customer want?
  - How to capture the customer needs?
- UML offers *Use Cases* (we will see in Chapter 6) to capture customer requirements
  - Use cases expressed in pictures and text.
  - Use case pictures use stick figures for human agents, boxes for software agents, circles for processes
  - Use case text is a line-by-line description of the user activity.

Example: Password Verification (Use Case Text)

- Scenario 1 (Success):
  - user enters userID
  - user enters password
  - password checked against password database
  - password valid, user set to go
- Scenario 2 (Failure):
  - user enters userID
  - user enters password
  - password fails to check out against password database
  - password invalid, user prompted to reenter password
  - After third successive failure, program aborts.

Example: Password Verification (Use Case Diagram)



What is an Agile Method?

- Agile proponents believe
  - Current software development processes are too heavyweight or cumbersome
    - Too many things are done that are not directly related to software product being produced
  - Current software development is too rigid
    - Difficulty with incomplete or changing requirements
    - Short development cycles (Internet applications)

What is an Agile Method?

- Agile methods are considered
  - Lightweight
  - People-based rather than Plan-based
- Several agile methods
  - No single agile method
  - XP most popular
- No single definition
- Agile Manifesto closest to a definition
  - Set of principles
  - Developed by Agile Alliance

Agile Manifesto

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software

2. Welcome changing requirements, even late in development. Agile process harness change for the customer´s competitive advantage

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale

What is Analysis and Design

- **Analysis**:
  - investigate problem and its requirements; solution comes later
  - ask and answer questions
  - Example: Dice Game. Some questions e.g. How the user will interact?
  - *Requirements Analysis*
    - investigate requirements
  - *Object-oriented Analysis*
    - investigate objects used in and by domain

What is Analysis and Design

- **Design**:
  - a *conceptual* solution that fulfills requirements
  - exclude implementation detail; not an implementation
  - Example: sequence diagram of given requirements.
  - *How object Collaborate to fulfill the Requirements?*
    - Sequence Diagrams
    - Class Diagram (Static view)

Case Studies:

- There are two followed in this book
  - NextGen POS (Point-of-Sale)
  - Monopoly

A Short Example

- **Dicey**: In which a program simulates a player tossing two dice.
- Define Use Cases
  - these are user scenarios, stories, goals

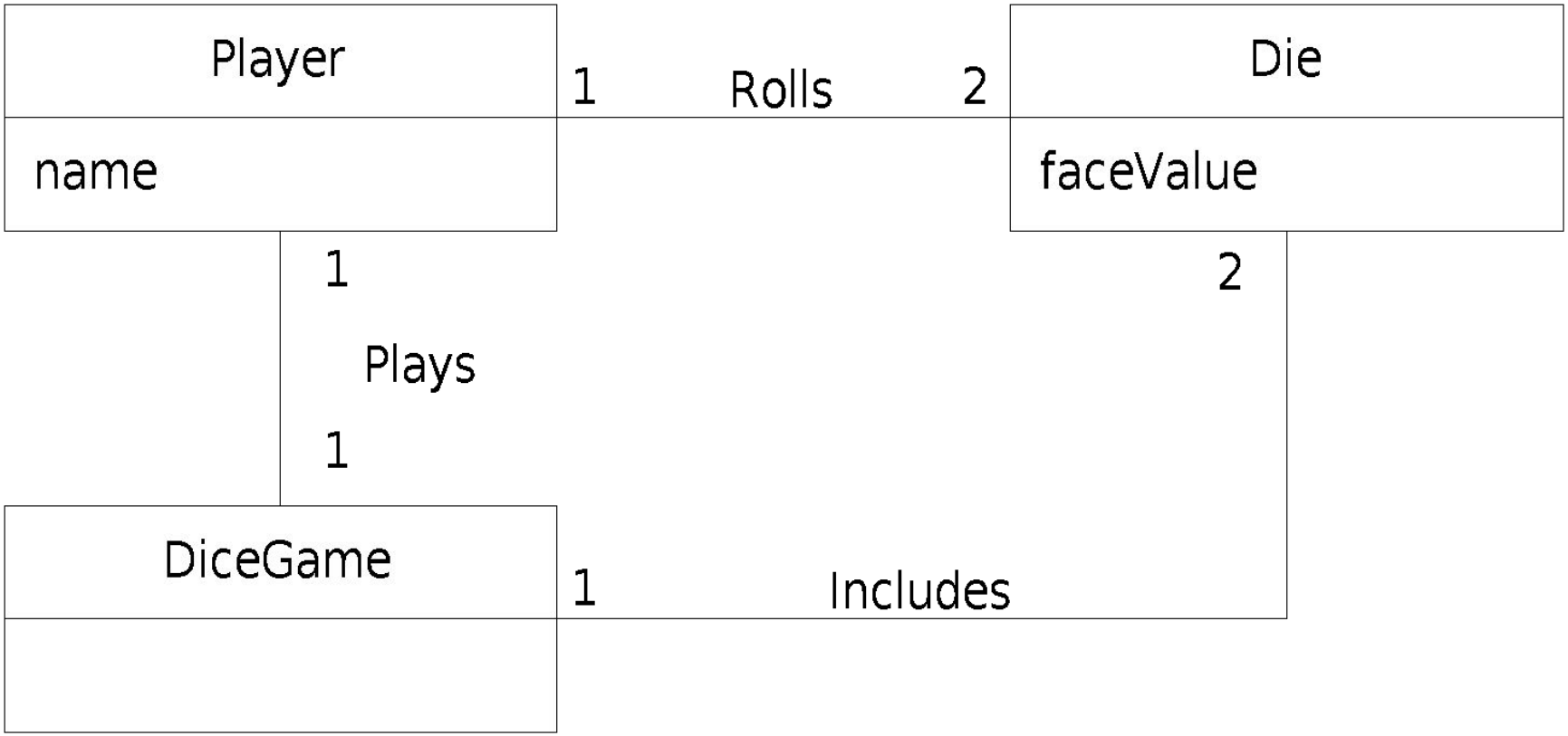
Play a Dice Game Use Case:

Player requests a roll of the two dices . System presents results. If dice show 7 player wins; otherwise player loses

- Define a Domain Model
  - this is a description of the domain from the point of view of the objects involved
  - identify the concepts, attributes and associations
  - result is called the *domain model*

Fig. 1.3

Partial (Conceptual) Domain Model

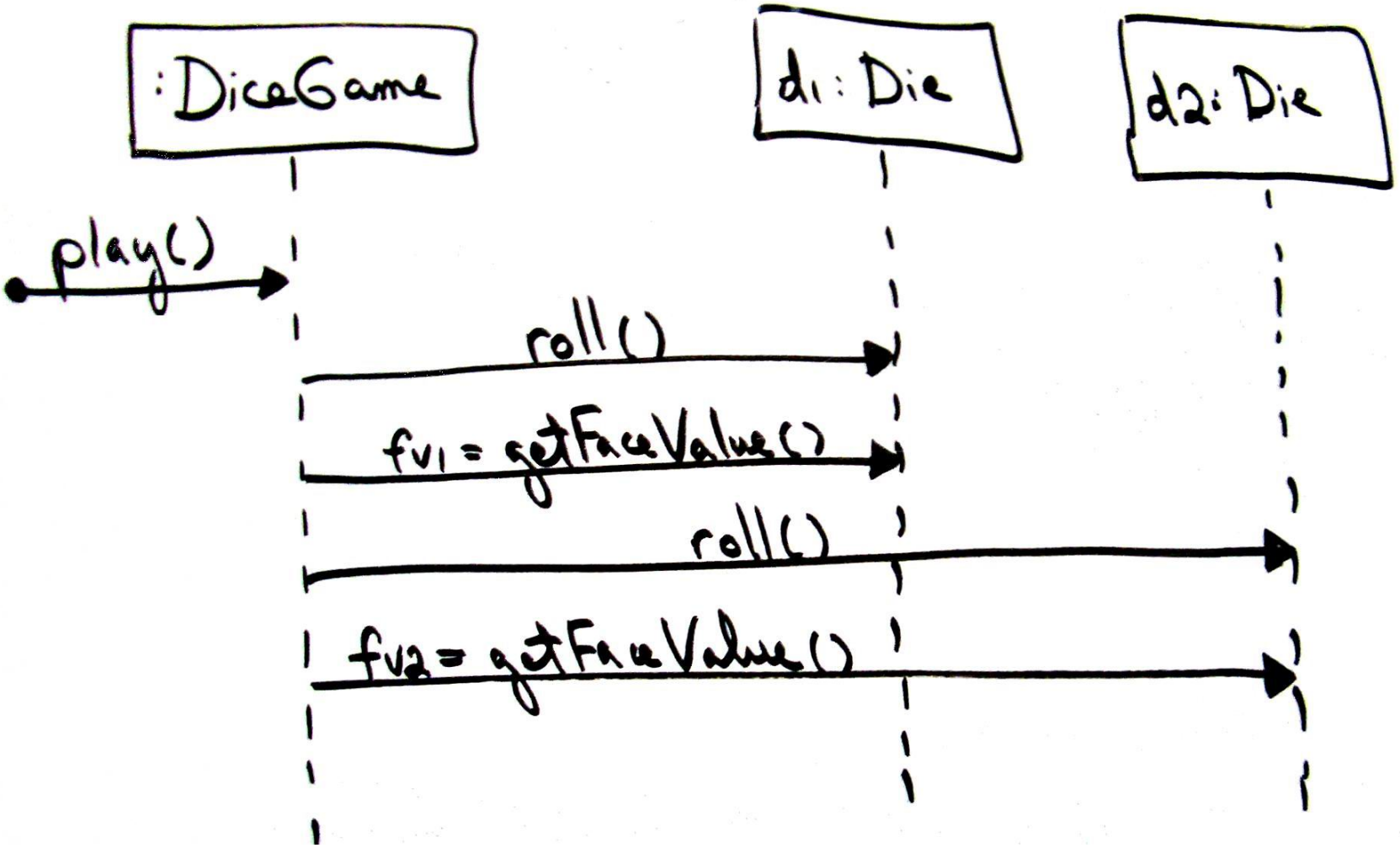


not descriptive of the software objects but rather of the domain

Assigning Responsibilities

- In a program, objects collaborate – pass each other messages, make data available to one another
- Common notation is called a ***sequence diagram***.
- A sequence diagram shows the *flow of messages* between objects
- If we know where the message goes we know who is "**responsible**" for "responding to" the message.
- Answering the message involves knowing "how" to answer the message
- How close is "flow of message" to the concept of "function call"?

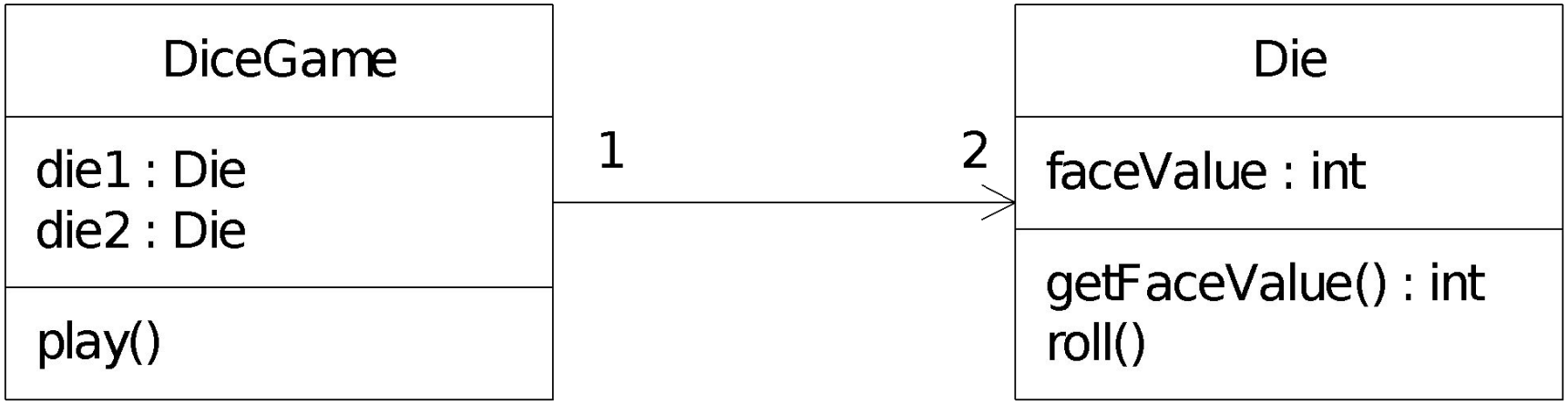
Fig. 1.4


:DiceGame
d1 : Die
d2 : Die
play()
roll()
fv1 = getFaceValue()
roll()
fv2 = getFaceValue()

Defining Classes

- The preceding is a *dynamic* view of collaborating classes
- The following is a *static* view of collaborating classes; more often called a **class diagram**.
- Differs from the domain model, with similarities.
- Follows the sequence diagram in the order of things.

Fig. 1.5


DiceGame
die1 : Die
die2 : Die
play()
1
2
Die
faceValue : int
getFaceValue() : int
roll()

Different Perspectives of "Class".

- Rectangular boxes are classes. But they can be physical things, abstract concepts, software things, events, …
- A software design methods or methodologies superimposes structure/naming conventions on the various UML objects.
- For example, in the Unified Process (UP), Domain Model, a box represents a conceptual class. In the Design Model they are called Design Classes.
- Our book follows UP:
  - conceptual class, software class, implementation class.

Visual Modeling is a Good Thing

- Diagrams help our understanding of things.
- Text is not easy to learn from, it is so linear.
- A picture is worth a thousand words.