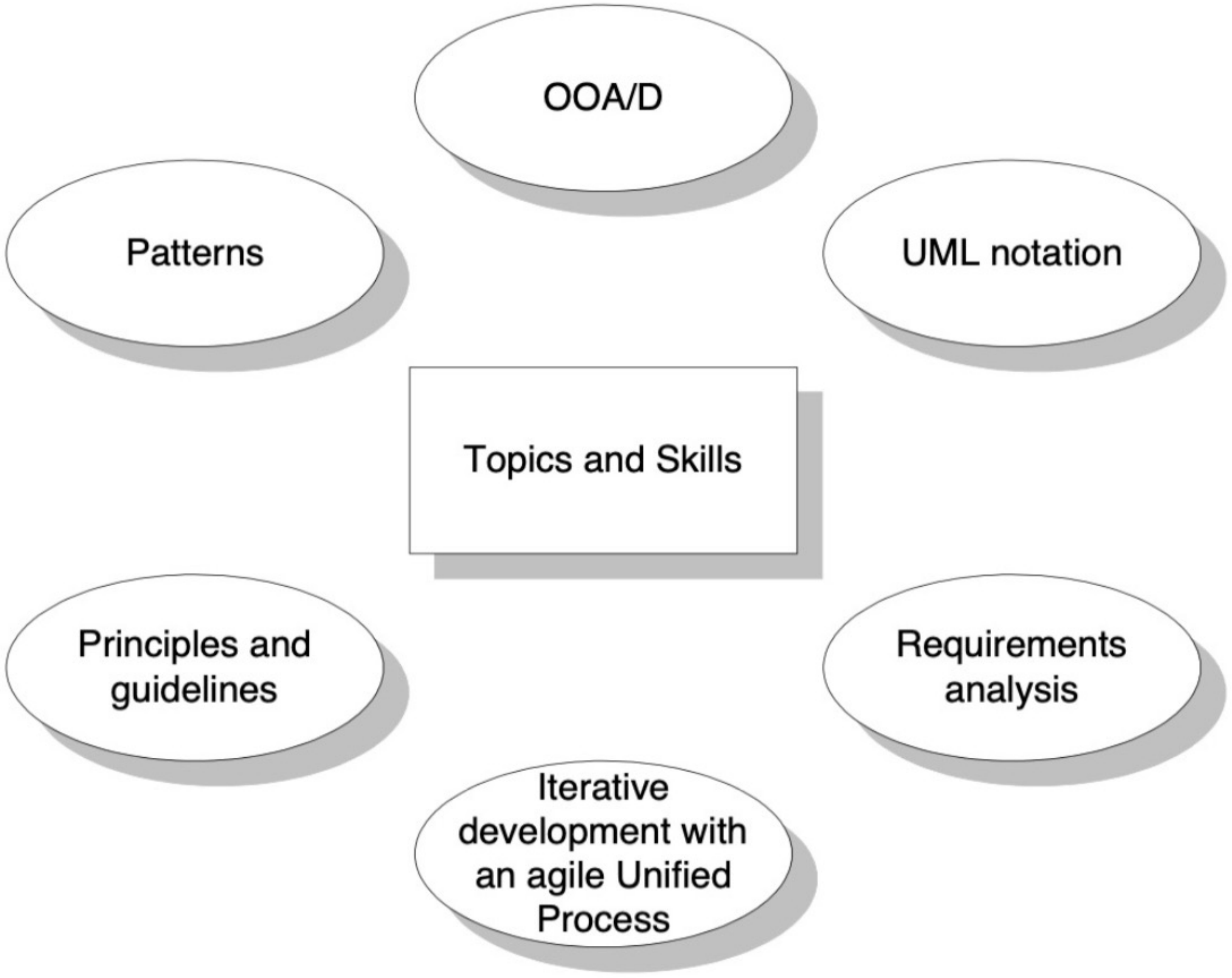
Object-Oriented Analysis and Design

A very brief introduction

Larman Ch1


OOA/D
Patterns
UML notation
Topics and Skills
Principles and guidelines
Requirements analysis
Iterative development with an agile Unified Process

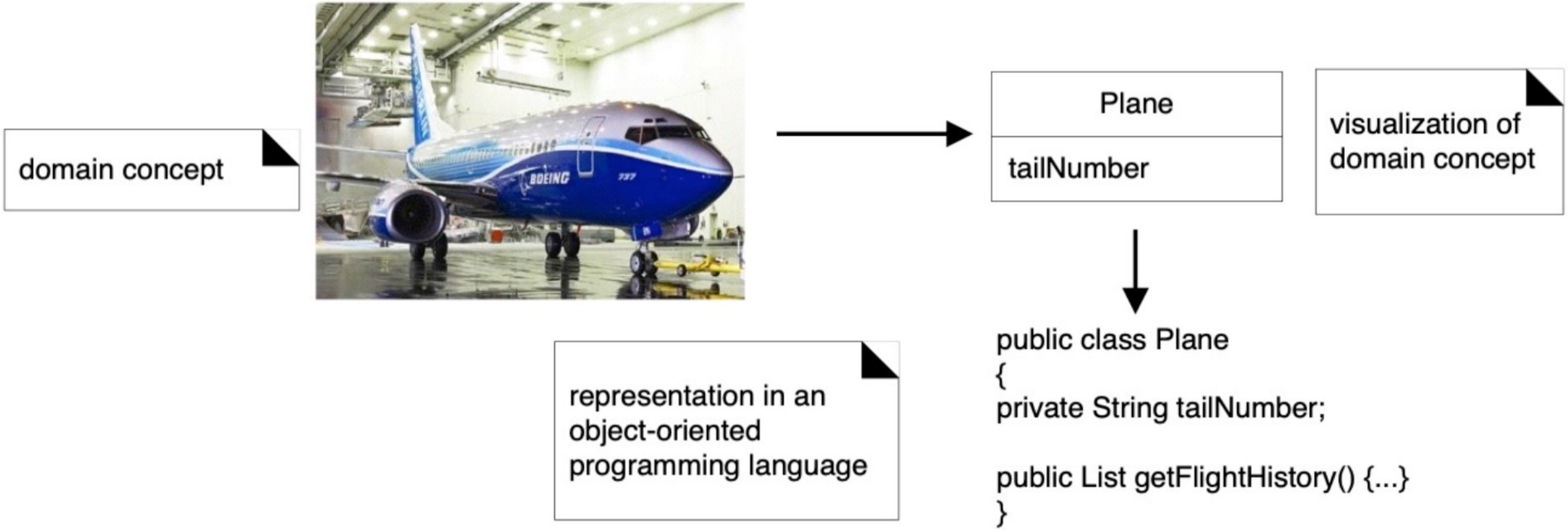
Analysis *investigation* of the problem and requirements, rather than a solution

requirements analysis (an investigation of the requirements)

object- oriented analysis (an investigation of the domain objects).

Design a *conceptual solution* (in software and hardware) that fulfills the requirements, rather than its implementation.

a description of software objects.


domain concept
Plane
tailNumber
visualization of domain concept
representation in an object-oriented programming language

```
public class Plane
{
private String tailNumber;

public List getFlightHistory() {...}
}
```

During **object-oriented analysis** find and describe the objects or concepts in the problem domain.

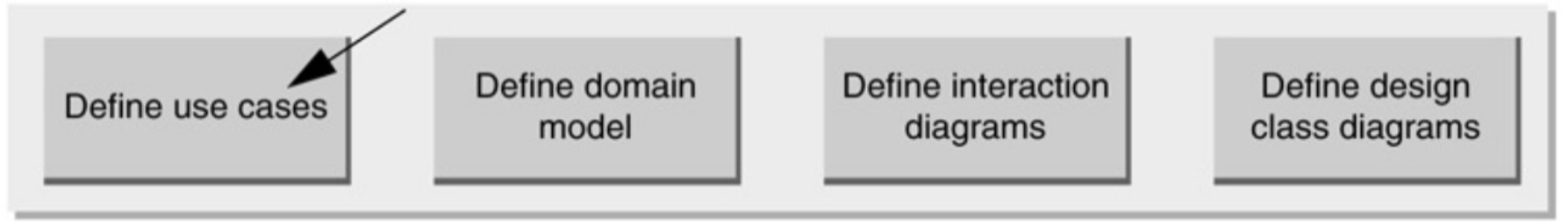
During **object-oriented design** define software objects and how they collaborate to fulfill the requirements.

For example, a *Plane* software object may have a *tailNumber* attribute and a *getFlightHistory* method

A Short Example: Dice Game

A dice game" in which software simulates a player rolling two dice. If the total is seven, they win; otherwise, they lose.

Define Use Cases



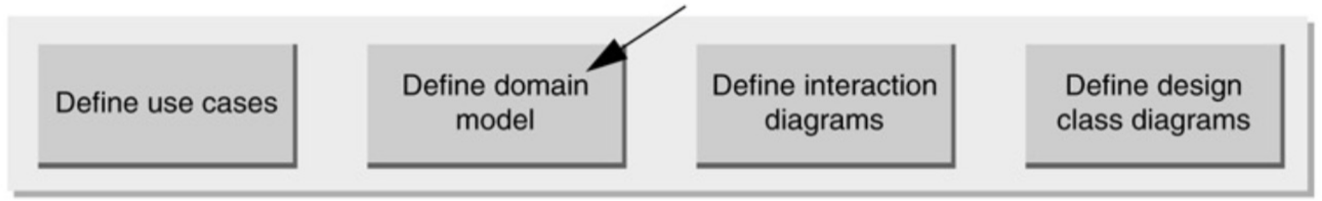
Requirements analysis include stories or scenarios of how people use the application; written as **use cases**.

Use cases are not an object-oriented artifact they are simply written stories, popular tool in requirements analysis.

For example, here is a brief version of the *Play a Dice Game* use case:

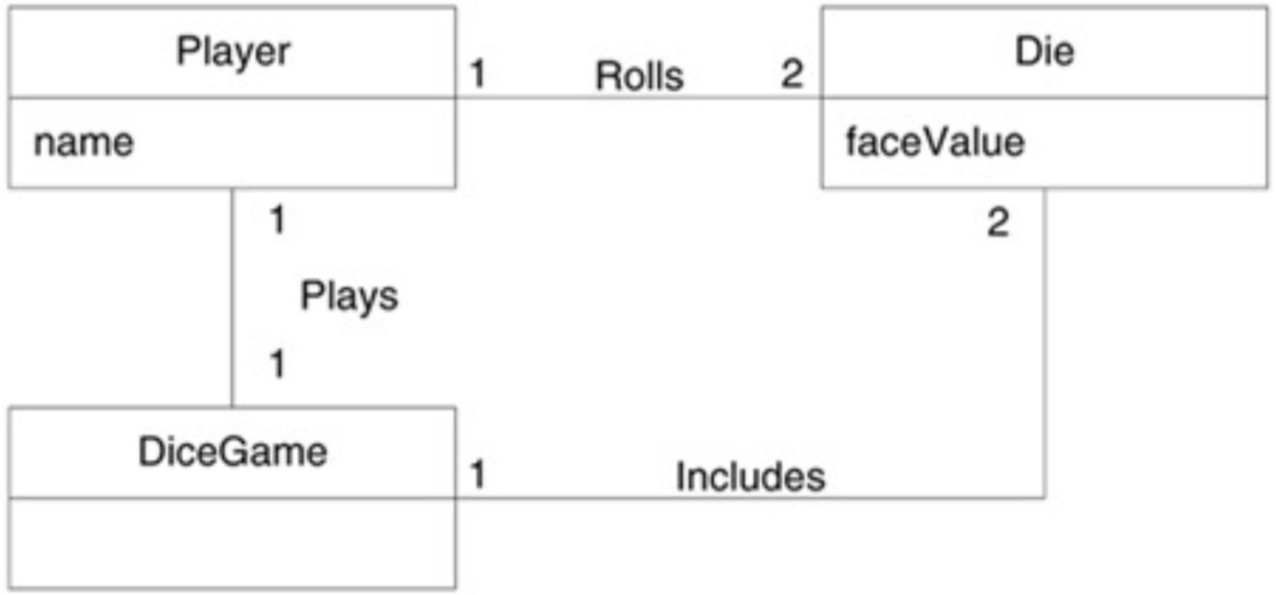
Play a Dice Game: Player requests to roll the dice. System presents results: If the dice face value totals seven, player wins; otherwise, player loses.

Define a Domain Model



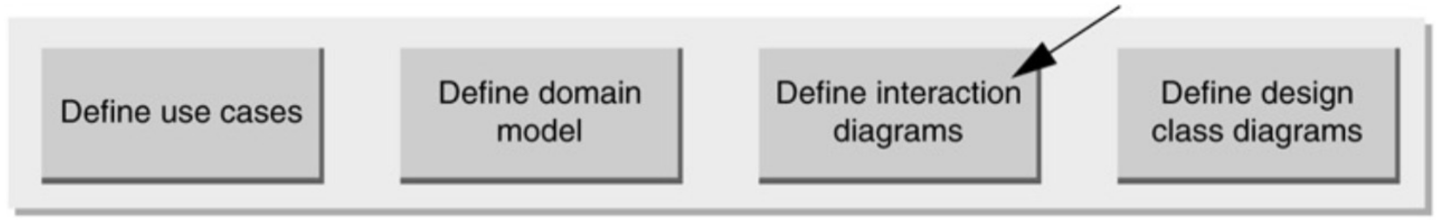
OOA is concerned with creating a description of the domain from the perspective of objects.
i.e., identification of the concepts, attributes, and associations that are considered noteworthy.

The result can be expressed in a **domain model** that shows the *noteworthy* domain concepts or objects.

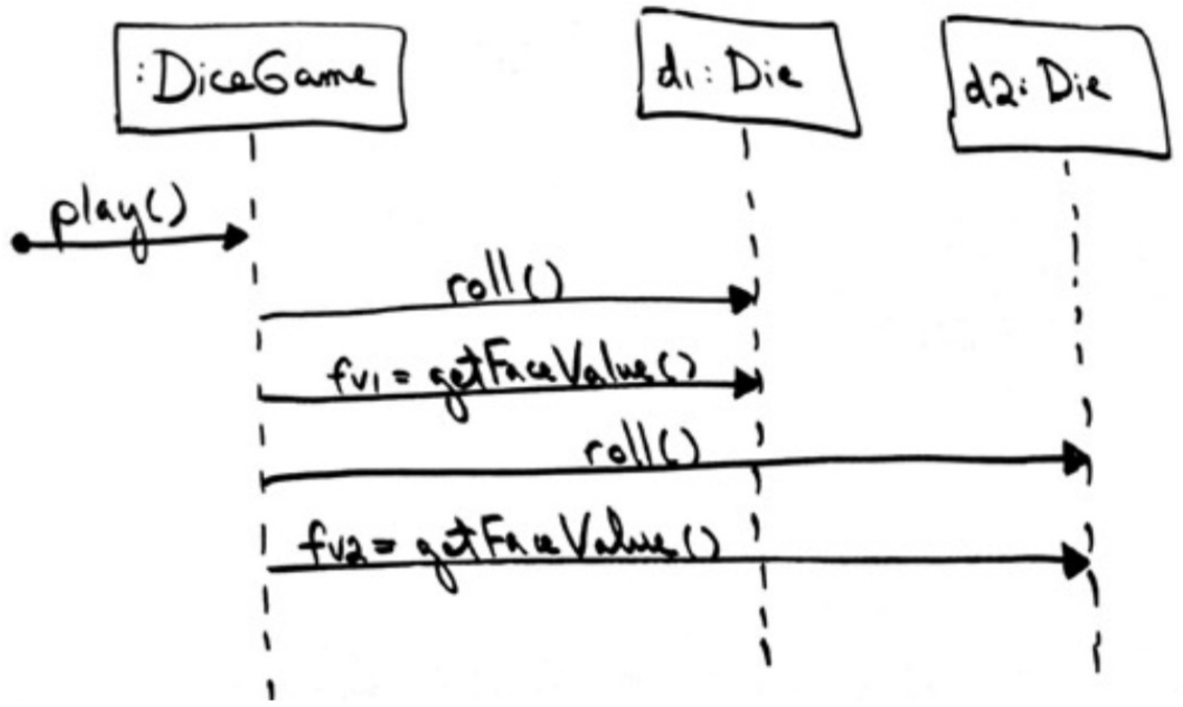


This model illustrates the noteworthy concepts *Player, Die,* and *DiceGame*, with their associations and attributes.
Note that a domain model is not a description of software objects; it is a visualization of the concepts or mental models of a real-world domain. Thus, it has also been called a **conceptual object model**.

Assign Object Responsibilities and Draw Interaction Diagrams

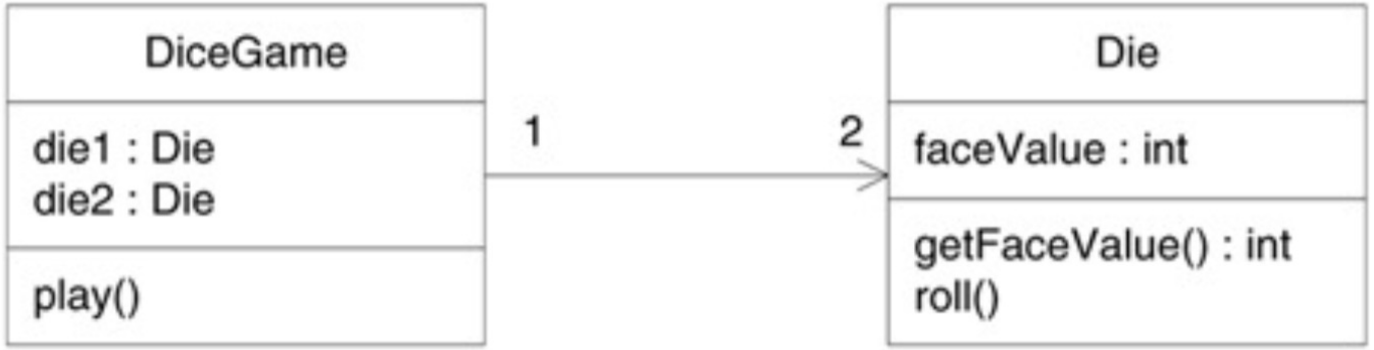


OOD is concerned with defining software objects their responsibilities and collaborations.
A common notation to illustrate these collaborations is the **sequence diagram** (a kind of UML interaction diagram).
It shows the flow of messages between software objects, and thus the invocation of methods.

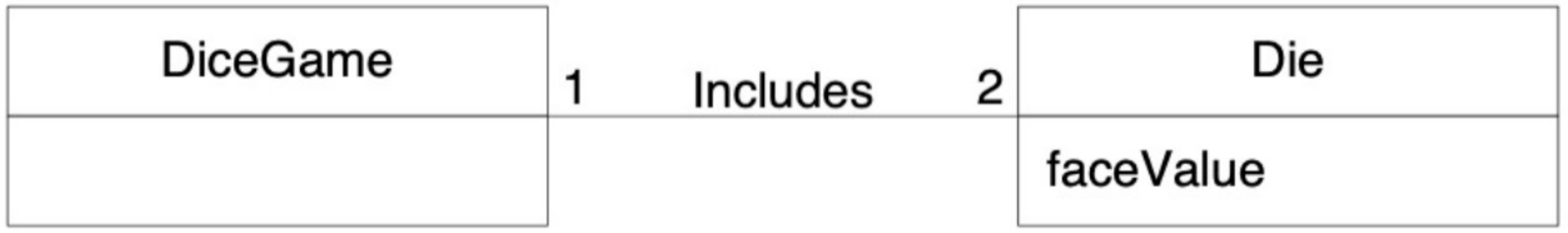


Define Design Class Diagrams

In addition to a *dynamic* view of collaborating objects shown in interaction diagrams, a *static* view of the class definitions is usefully shown with a **design class diagram**. This illustrates the attributes and methods of the classes.
In contrast to the domain model showing real-world classes, this diagram shows software classes.

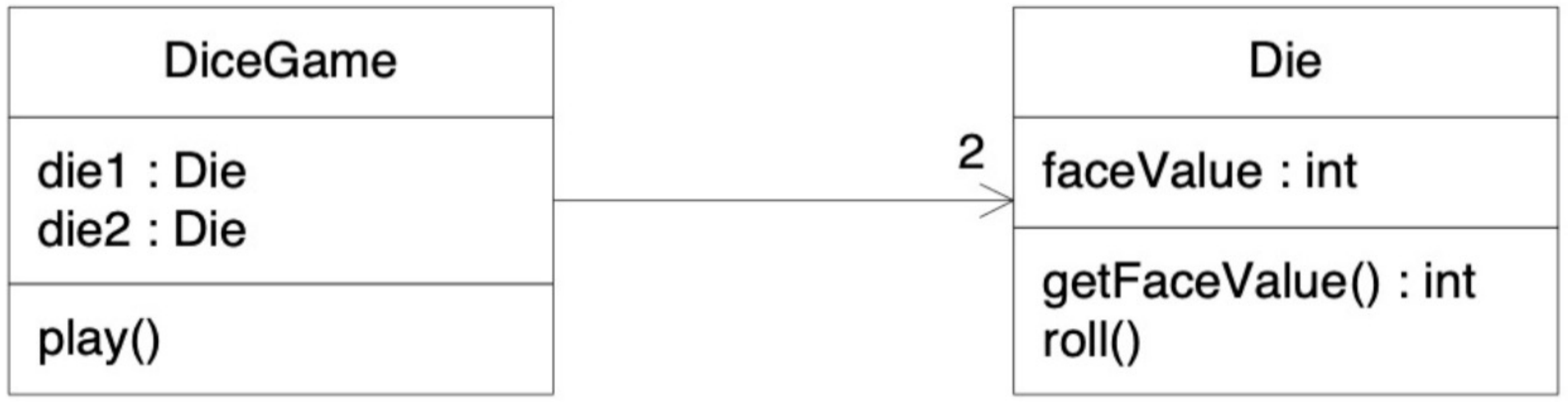


Notice that although this design class diagram is not the same as the domain model, some class names and content are similar. In this way, OO designs and languages can support a **lower representational gap** between the software components and our mental models of a domain. That improves comprehension.



Conceptual Perspective (domain model)

Raw UML class diagram notation used to visualize real-world concepts.



Specification or Implementation Perspective (design class diagram)

Raw UML class diagram notation used to visualize software elements.

Conceptual class real-world concept or thing.

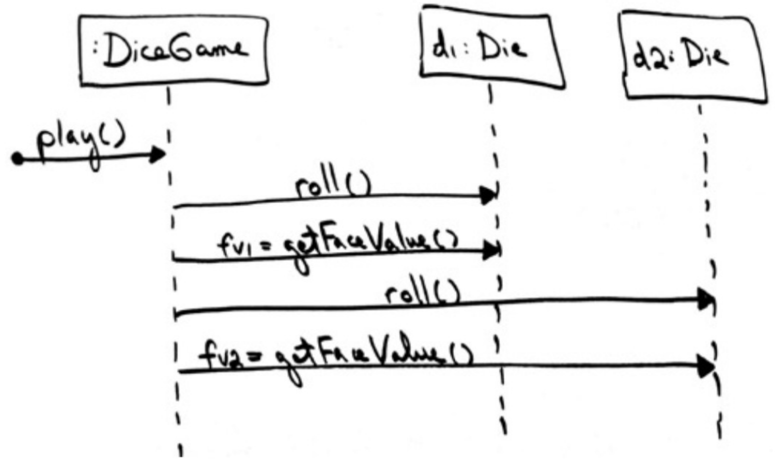
Software class a class representing a specification or implementation perspective of a software component,
Implementation class a class implemented in a specific OO language such as Java.

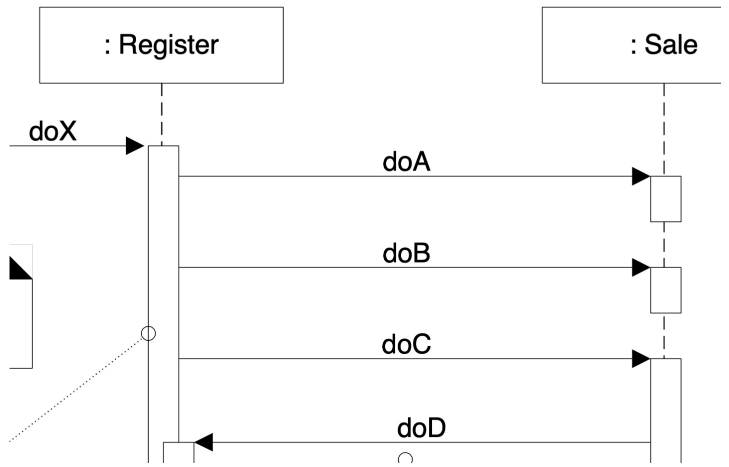
What is the UML?
The Unified Modeling Language is a visual language for specifying, constructing and documenting the artifacts of systems

Three Ways to Apply UML
UML as sketch Informal and incomplete diagrams (often hand sketched on whiteboards) created to explore difficult parts of the problem or solution space, exploiting the power of visual languages.

UML as blueprint Relatively detailed design diagrams used either for 1) reverse engineering to visualize and better understanding existing code in UML diagrams, or for 2) code generation (forward engineering).

UML as programming language Complete executable specification of a software system in UML. Executable code will be automatically generated, but is not normally seen or modified by developers; one works only in the UML "programming language."





"Silver Bullet" Thinking

it's a fundamental mistake to believe there is some special tool or technique in software that will make a dramatic order- of-magnitude difference in productivity, defect reduction, reliability, or simplicity. *And tools don't compensate for design ignorance*.

A person not having good OO design and programming skills who draws UML is just drawing bad designs.

Agile modeling emphasizes *UML as sketch*; this is a common way to apply the UML, often with a high return on the investment of time.