Object-Oriented Software Analysis and Design

School of Computer Science
University of Windsor

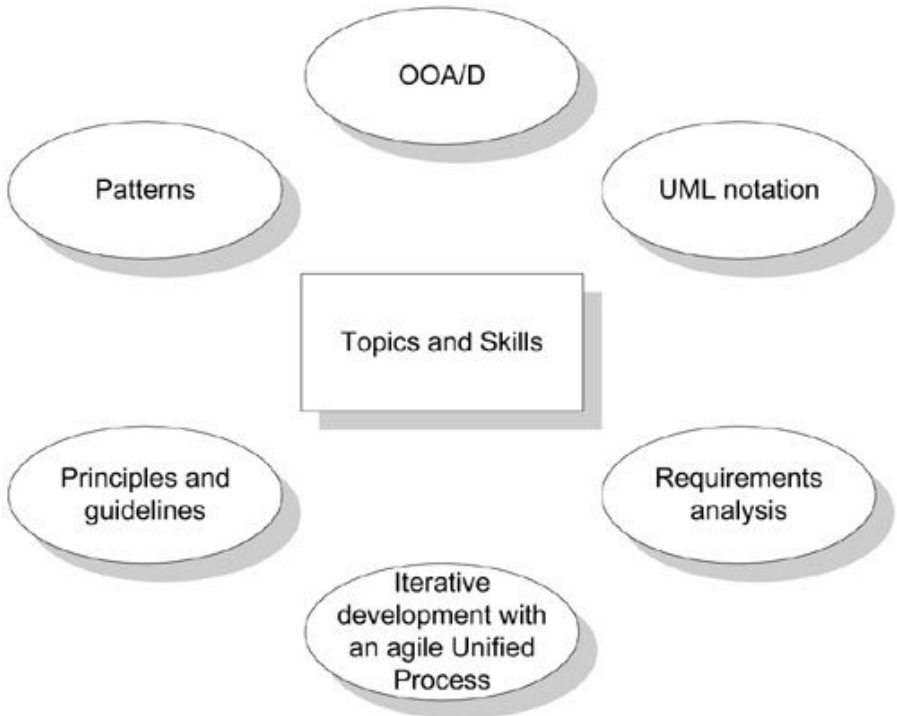
Course Objectives

- to help developers and students learn **core skills** in object-oriented analysis and design (OOA/D).

Course Usefulness

- for the creation of well-designed, robust, and maintainable software using OO technologies and languages such as Java or C#.
- emphasizes mastery of the fundamentals, such as how to assign responsibilities to objects, frequently used UML notation, and common design patterns.

Topics and Skills to be Covered



Other Important Skills

- usability engineering
- user interface design
- database design and so on

What is Analysis and Design?

- **Analysis** emphasizes an investigation of the problem and requirements, rather than a solution.

  For example,

  if a new online trading system is desired, how will it be used? What are its functions?
- "Analysis" is a broad term, best qualified, as in **requirements analysis** (an investigation of the requirements) or **object-oriented analysis** (an investigation of the domain objects).

What is Analysis and Design? (contd.)

- **Design** emphasizes a conceptual solution that fulfills the requirements, rather than its implementation.

  For example,

  a description of a database schema and software objects.
- Ultimately, designs can be implemented, and the implementation (such as code) expresses the true and complete realized design.

- Useful analysis and design have been summarized in the phrase **do the right thing (analysis)**, and **do the thing right (design)**.

What is Object-Oriented Analysis and Design?

- During **object-oriented analysis**, there is an emphasis on finding and describing the objects—or concepts—in the problem domain.

  For example,

  in the case of the flight information system, some of the concepts include `Plane`, `Flight`, and `Pilot`.

What is Object-Oriented Analysis and Design? (contd.)

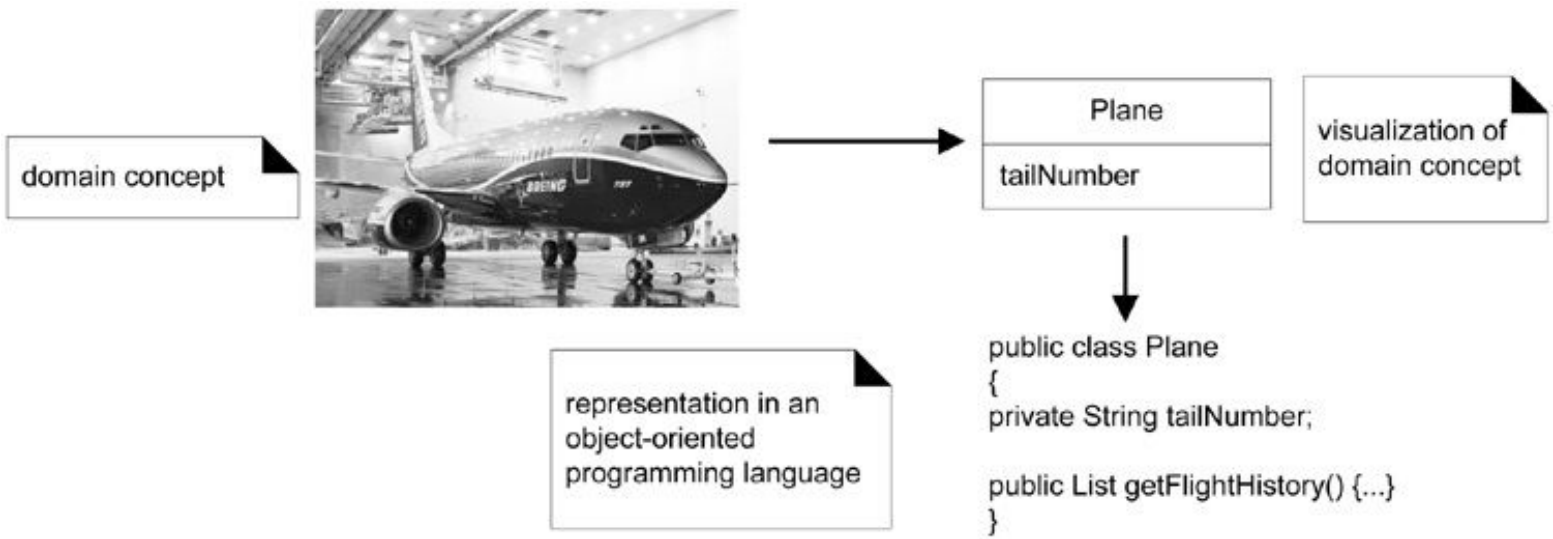
- During **object-oriented design**, there is an emphasis on defining software objects and how they collaborate to fulfill the requirements.

  For example,

  in the flight information system, a `Plane` software object may have a `tailNumber` attribute and a `getFlightHistory` method

What is Object-Oriented Analysis and Design? (contd.)

- Finally, during implementation or object-oriented programming, design objects are implemented, such as a `Plane` class in Java.



Object-Oriented Thinking

- The proverb "*owning a hammer doesn't make one an architect*" is especially true with respect to object technology.
- Knowing an object-oriented language (such as Java) is a necessary but insufficient first step to create object systems.
- Knowing how to "***think in objects***" is critical!

Object-Oriented Thinking (contd.)

- involves examining the problems or concepts at hand, breaking them down into component parts, and thinking of those as objects.

  For example,

  a `tweet` on Twitter or a `product` on an online shopping website could be considered objects.

- When translated to object-oriented modelling, object-oriented thinking involves representing key concepts through objects in your software.

Object-Oriented Thinking (contd.)

- Objects may have specific details associated with them, which are relevant to users.

  For example,

  a `person` object may have details such as `name`, `age`, `gender`, and `occupation`. A `place` object may have a `size`, or a `name`.

Object-Oriented Thinking (contd.)

- Objects might also have behaviors or responsibilities associated with them.

  For example,

  a `person` may have associated behaviors such as `sitting down` or `typing`. An `electronic device` may be responsible to power `on` or `off`, or to `display` an image.

Object-Oriented Thinking (contd.)

By using objects to represent things in your code, the code stays

Organized

Objects keep code organized by putting related details and specific functions in distinct, easy-to-find places.

Flexible

Objects keep code flexible, so details can be easily changed in a modular way within the object, without affecting the rest of the code.

Reusable

Objects allow code to be reused, as they reduce the amount of code that needs to be created, and keeps programs simple.

A Short Example

- The following few slides present a bird's-eye view of a few key steps and diagrams, using a simple example "dice game" in which software simulates a player rolling two dice.

A Short Example: Define Use Cases

- Requirements analysis may include **stories or scenarios** of how people use the application; these can be written as use cases.

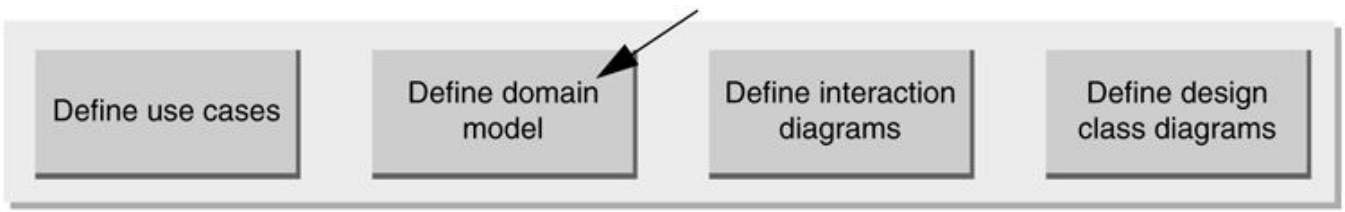
- *Use cases are not an object-oriented artifact they are simply written stories. However, they are a popular tool in requirements analysis.*

  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.

A Short Example: Define a Domain Model

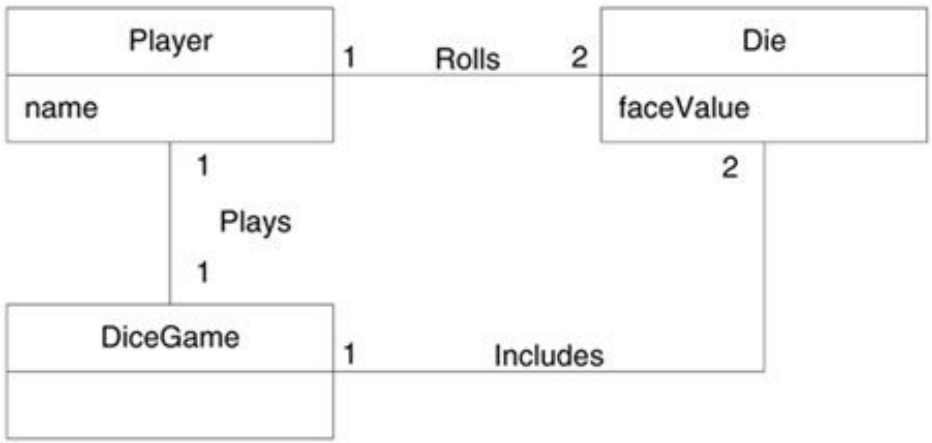
- Object-oriented analysis is concerned with **creating a description of the domain from the perspective of objects**.
- There is an identification of the concepts, attributes, and associations that are considered noteworthy.



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

A Short Example: Define a Domain Model (contd.)

For example, a partial domain model is shown in Figure



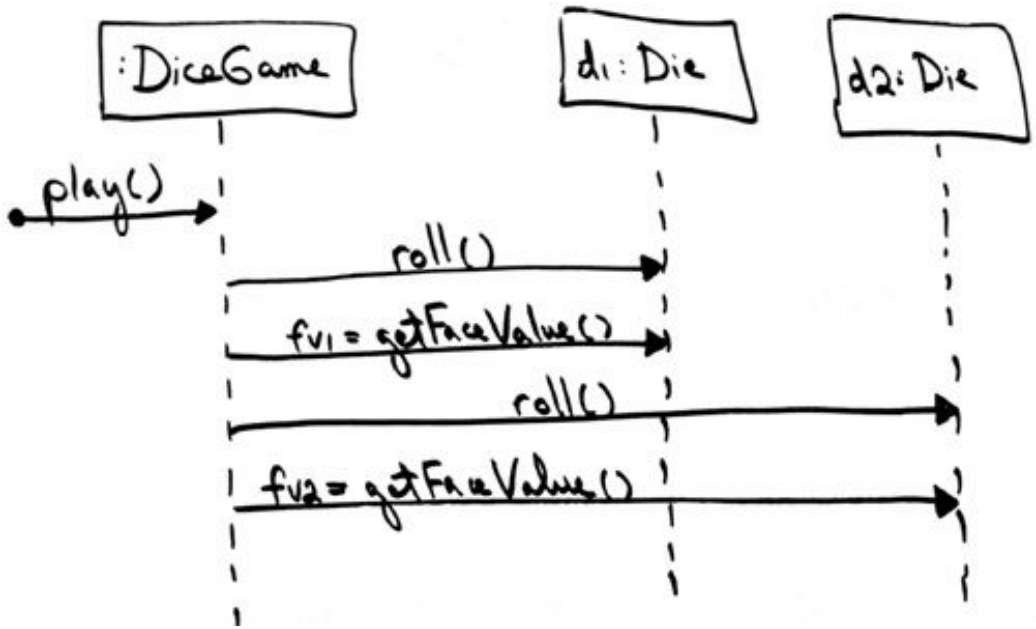
- Note that a domain model is not a description of software objects; it is a visualization of the concepts or mental models of a real-world domain. Thus, it has also been called a **conceptual object model**.

A Short Example: Assign Object Responsibilities and Draw Interaction Diagrams

- Object-oriented design is concerned with **defining software objects their responsibilities and collaborations**.
- A common notation to illustrate these collaborations is the sequence diagram.
- It shows the flow of messages between software objects, and thus the invocation of methods.

A Short Example: Assign Object Responsibilities and Draw Interaction Diagrams (contd.)

For example, the sequence diagram in Figure illustrates an OO software design, by sending messages to instances of the DiceGame and Die classes. Note this illustrates a common real-world way the UML is applied: by sketching on a whiteboard.

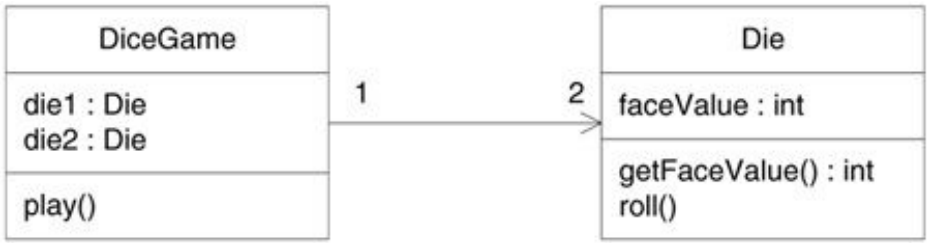


A Short Example: Define Design Class Diagrams

- 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**.

A Short Example: Define Design Class Diagrams (contd.)

For example, in the dice game, an inspection of the sequence diagram leads to the partial design class diagram shown in Figure.



- In contrast to the **domain model showing real-world classes**, **class 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.

What is the UML?

- The Unified Modeling Language is a **visual** language for specifying, constructing and documenting the artifacts of systems.
- The word **visual** in the definition is a key point the UML is the de facto standard diagramming notation for drawing or presenting pictures (with some text) related to software primarily OO software.
- It was developed by Grady Booch, Ivar Jacobson and James Rumbaugh at Rational Software in 1994-1995, with further development led by them through 1996.

Three Ways to Apply UML

1. UML as sketch
2. UML as blueprint
3. UML as programming language

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.

- 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 (which is typically short).
- UML tools can be useful, but I encourage people to also consider an agile modeling approach to applying UML.

Three Ways to Apply UML: UML as blueprint

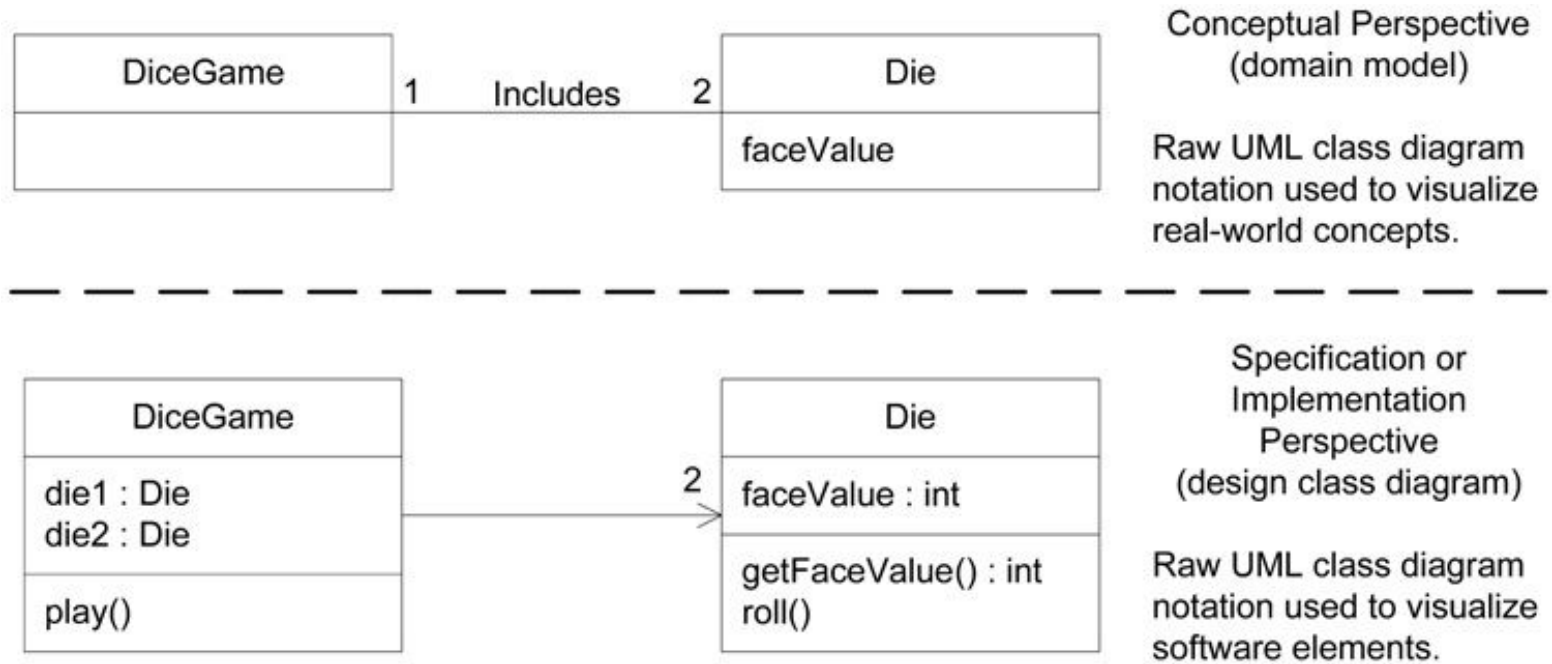
- Relatively detailed design diagrams used either for
  - **reverse engineering** to visualize and better understanding existing code in UML diagrams, or for
  - code generation (**forward engineering**).

Three Ways to Apply UML: 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."
- This use of UML requires a practical way to diagram all behavior or logic (probably using interaction or state diagrams), and is still under development in terms of theory, tool robustness and usability.

Three Perspectives to Apply UML

1. Conceptual perspective
2. Specification (software) perspective
3. Implementation (software) perspective



Three Perspectives to Apply UML: Conceptual Perspective

- the diagrams are interpreted as describing things in a situation of the real world or domain of interest.

Three Perspectives to Apply UML: Specification (Software) Perspective

- the diagrams (using the same notation as in the conceptual perspective) describe software abstractions or components with specifications and interfaces, but no commitment to a particular implementation (for example, not specifically a class in C# or Java).

Three Perspectives to Apply UML: Implementation (Software) Perspective

- the diagrams describe software implementations in a particular technology (such as Java).

The Meaning of "Class" in Different Perspectives

- **Conceptual class** real-world concept or thing. A conceptual or essential perspective. The UP Domain Model contains conceptual classes.
- **Software class** a class representing a specification or implementation perspective of a software component, regardless of the process or method.
- **Implementation class** a class implemented in a specific OO language such as Java.

UML Diagram Taxonomy

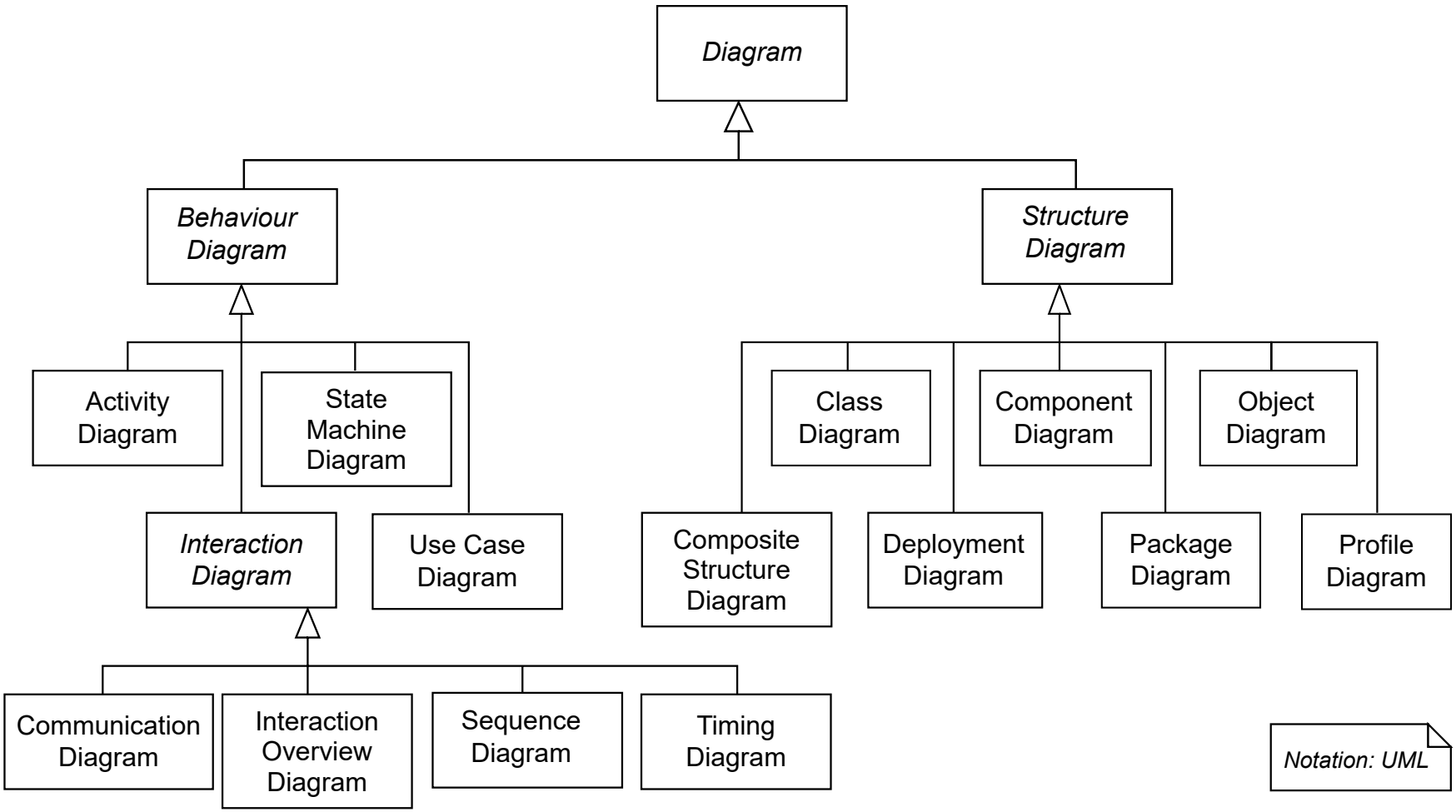


Figure: UML diagram taxonomy

Structure Diagrams

- used to show the static structure of elements in the system.
- depicts such things as
  - the architectural organization of the system
  - the physical elements of the system
  - its runtime configuration and
  - domain-specific elements of your business

For example,

- A **class diagram** is used to show the existence of classes and their relationships in the logical view of a system.
- A single class diagram represents a view of the class structure of a system.
- During analysis, we use class diagrams to indicate the common roles and responsibilities of the entities that provide the system's behavior.
- During design, we use class diagrams to capture the structure of the classes that form the system's architecture.

Structure Diagrams (contd.)

example (contd.)

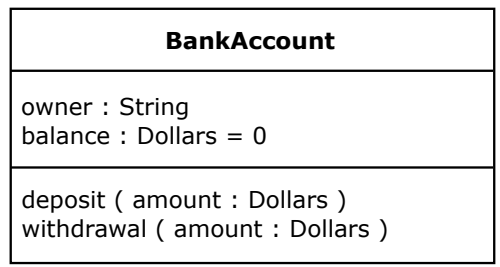


Figure: *Class diagram*

Behavior Diagrams

- express the dynamic behavioral semantics of a problem or its implementation

  For example,

  - **Use case diagram**s are used to depict the context of the system to be built and the functionality provided by that system.
  - They depict who (or what) interacts with the system.
  - They show what the outside world wants the system to do.

Behavior Diagrams (contd.)

example (contd.)

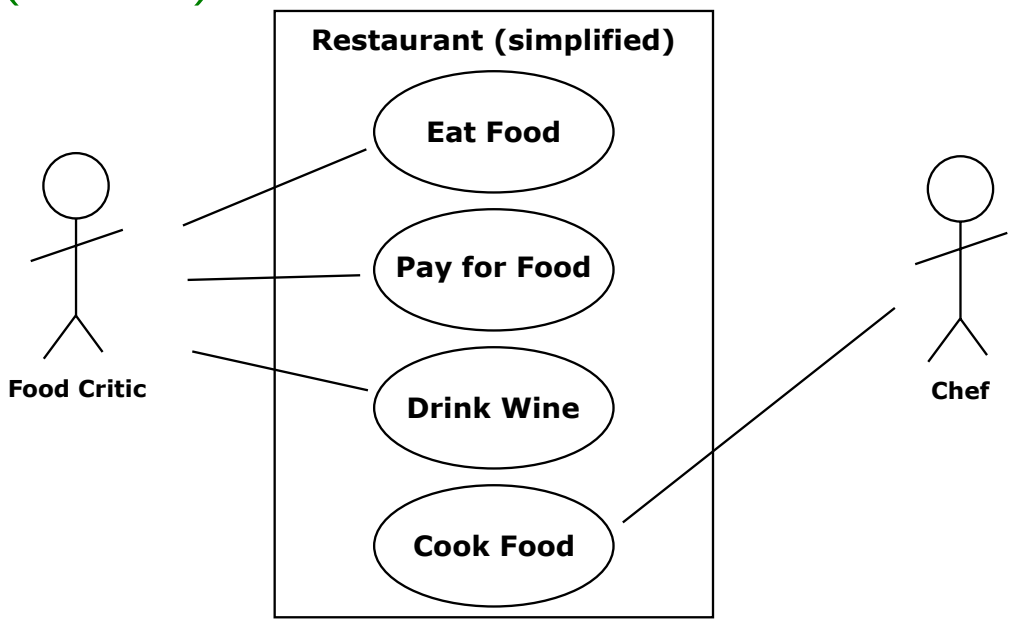


Figure: *Use Case diagram*

Interaction Diagrams

- a subset of behavior diagrams, intended to provide an overview of the flow of control and data among the elements in the system being modeled.

  For example,

  - A **sequence diagram** is used to trace the execution of a scenario in the same context.
  - They capture the interaction between objects in the context of a collaboration.
  - Sequence Diagrams are time focused and they show the order of the interaction visually by using the vertical axis of the diagram to represent time what messages are sent and when.

Interaction Diagrams (contd.)

example (contd.)

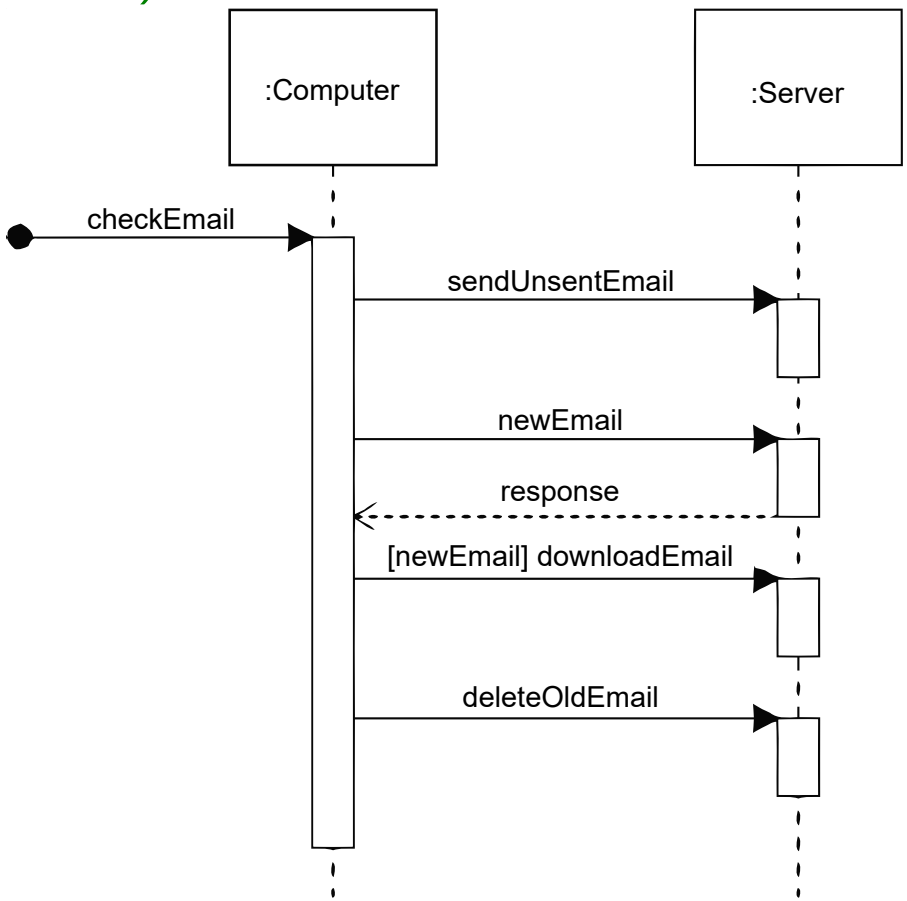


Figure: *Sequence diagram*

It's Quiz Time

1. Use cases are not an object-oriented artifact. (True or False)
2. A dynamic view of the class definitions is usefully shown with a design class diagram. (True or False)
3. In ..... perspective, the diagrams are interpreted as describing things in a situation of the real world or domain of interest.