SWE209
Object Oriented Analysis and Design

Modelling with UML - 1

Note

- This presentation is based on the slides and content of the course main textbook.
- Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014
- https://ase.in.tum.de/lehrstuhl_1/component/content/article/43-books/217

Agenda

Introduction

Software Development and Diagrams

UML Diagrams

Modeling Concepts

Introduction

Notation

- Notations enable us to express complex ideas succinctly and precisely.
- For a notation to enable accurate communication;
  - It must come with a well-defined semantics.
  - It must be well suited for representing a given aspect of a system.
  - It must be well understood among project participants.
- Strength of standards and conventions.

Software Development and Diagrams

UML

- UML will be used as the primary notation in this course. Why?
  - Standard notation in the software industry.
  - Standard notation that can be used by all object-oriented methods.
  - Provides a spectrum of notations for representing different aspects of a system.

Software Development and Diagrams

UML

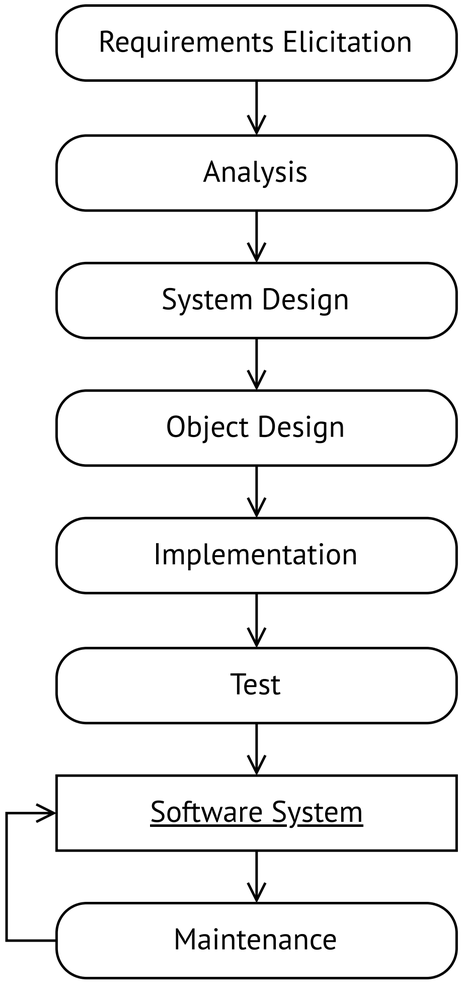
- UML → Unified Modeling Language
  - Developed by the Object Management Group
    - https://www.omg.org/spec/UML/2.5.1/About-UML/
  - Evolves → Has different versions.
  - Focus on ≥ UML 2.2

Software Development and Diagrams

- There are many different types of UML diagrams.
- Focus on five main types:
  - Use case diagrams
  - Class diagrams
  - Interaction diagrams
  - State machine diagrams
  - Activity diagrams

Software Development and Diagrams

Software Engineering Development Activities



Software Development and Diagrams

- Three models of the system:
  - Functional Model
    - Use Case Diagrams
  - Object Model
    - Class Diagrams
  - Dynamic Model
    - Interaction Diagrams, State Machine Diagrams, Activity Diagrams

UML Diagrams

UML Diagrams

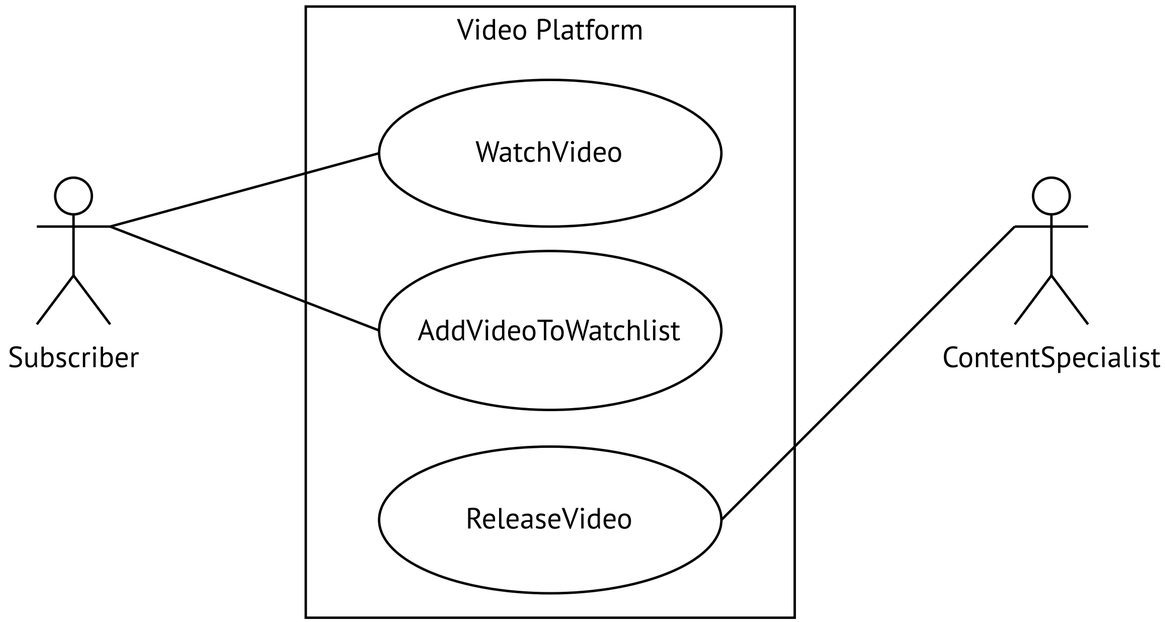
- Overview of five main types of UML diagrams:
  - Use Case Diagrams
  - Class Diagrams
  - Interaction Diagrams
  - State Machine Diagrams
  - Activity Diagrams

Use Case Diagrams

- Used during requirements elicitation and analysis stages.
- Represent the functionality of the system.
- Represent the functional model of the system
- Illustrate the behavior of the system from an external point of view.
- Use case diagram → Use case + Actor
  - Use case → A function provided by the system.
  - Actor → An entity that interacts with the system.

Use Case Diagrams

- Use case diagrams → Boundary of the system
- System vs Environment
  - Use cases → Inside
  - Actors → Outside



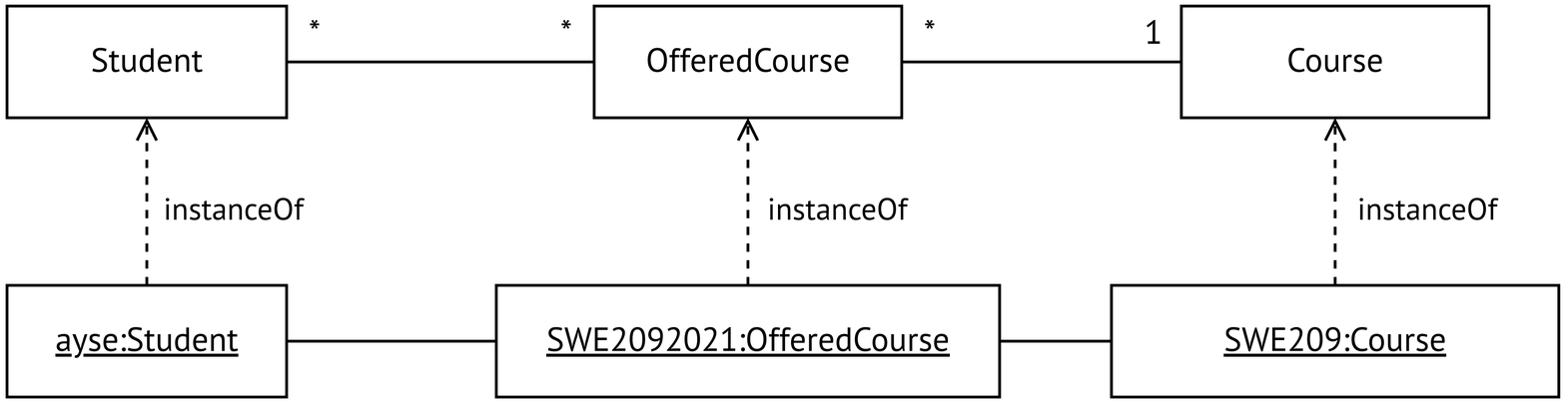
Class Diagrams

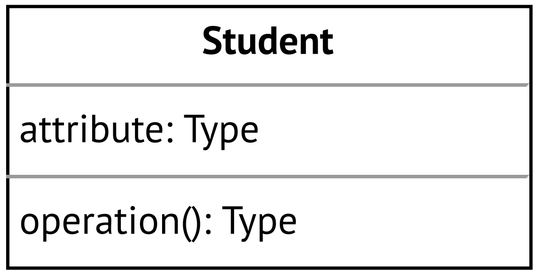
- Used to describe the structure of the system.
- Represent the object model model of the system.
- **Class:** Abstraction that specify the common structure and behavior of a set of objects.
- **Object:** Instance of a class.
- An object has state. The state of the object includes;
  - The values of its attributes.
  - The links with other objects.

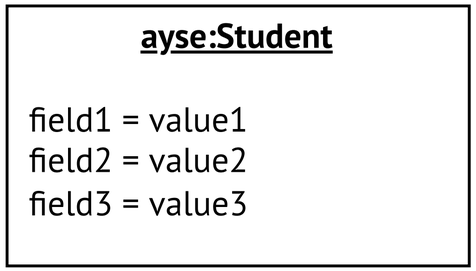
Class Diagrams

- Class diagrams describe the system in terms of;
  - objects
  - classes
  - attributes
  - operations
  - associations

Class Diagrams







Interaction Diagrams

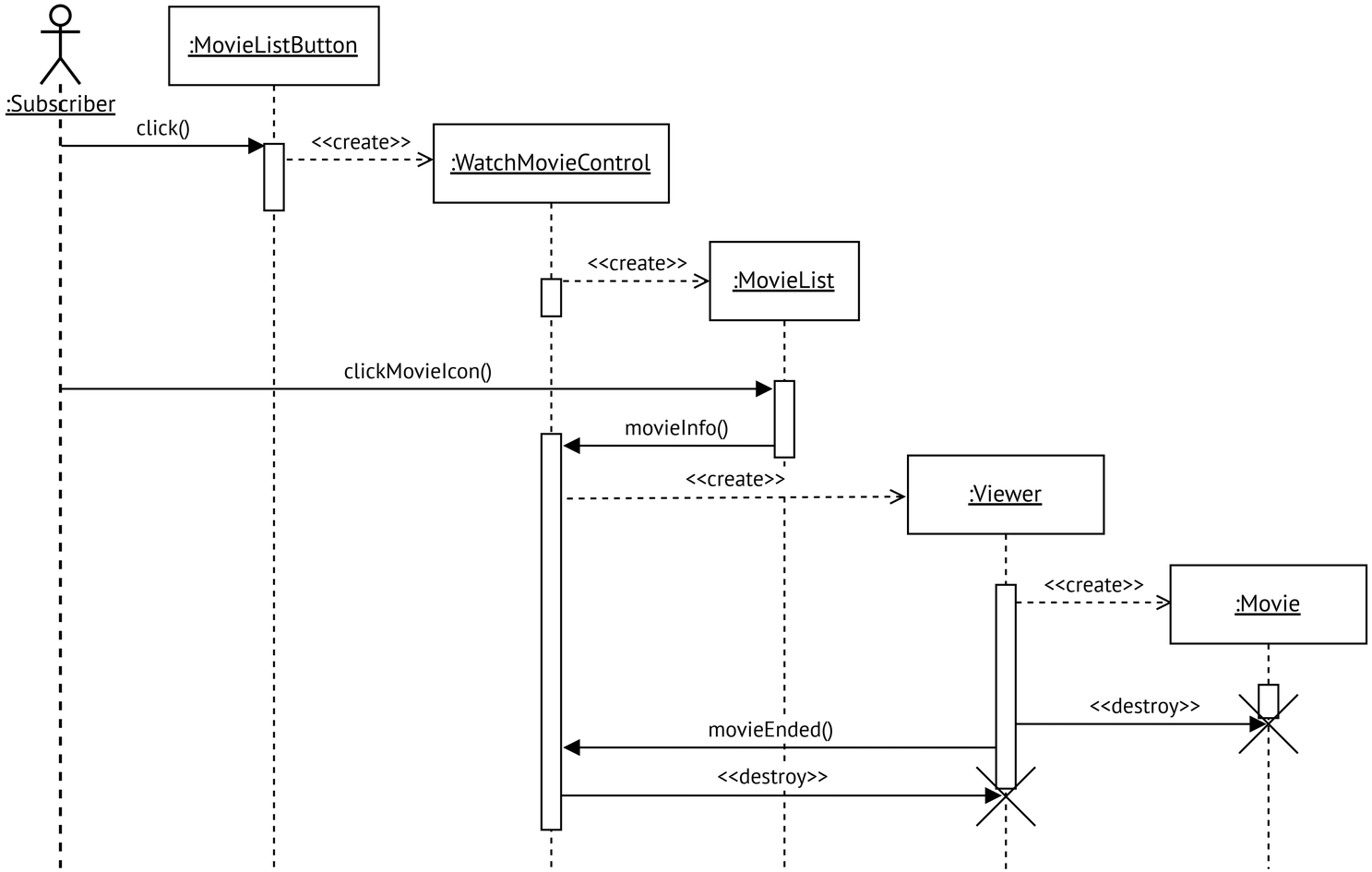
- Used to formalize the dynamic behavior of the system.
- Used to visualize the communication among objects.
- Represent the dynamic model of the system.
- Useful for identifying additional objects that participate in the use cases.
- Represents the interactions that take place among participating objects.
  - **Participating objects:** Objects involved in a use case.

Interaction Diagrams

- Two types of interaction diagrams:
  - Sequence diagrams
  - Communication diagrams

Interaction Diagrams

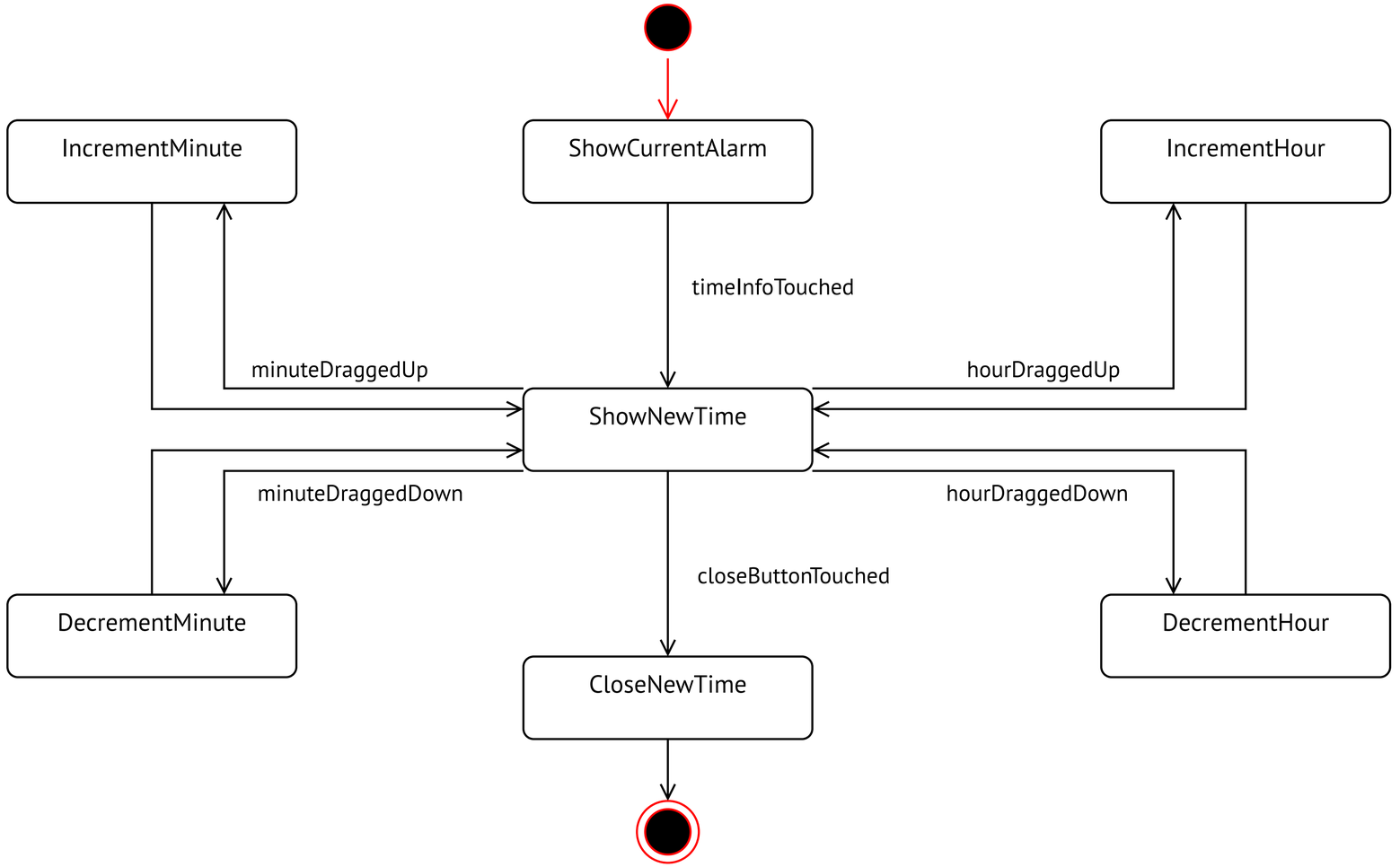
Sample Sequence Diagram



State Machine Diagrams

- Describe the dynamic behavior of an individual object.
  - A number of states + Transitions between the states
- Represent the dynamic model of the system.
- State → Particular set of values for an object
- Transition →
  - A future state the object can move to
  - The conditions associated with the change of state

State Machine Diagrams



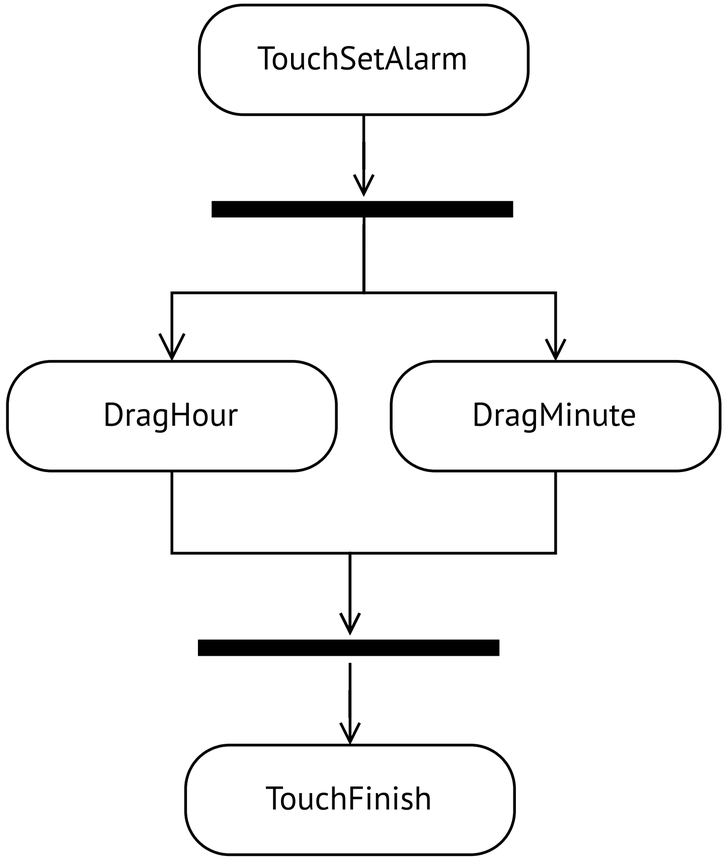
Activity Diagrams

- Describe the behavior of a system in terms of activities.
- Represent the dynamic model of the system.
- Activities → Modeling elements that represent the execution of a set of operations.
- The execution of an activity can be triggered by;
  - The completion of other activities
  - The availability of objects
  - External events

Activity Diagrams

- Similar to flowchart diagrams
  - Used to represent;
    - Control flow
    - Data flow

Activity Diagrams



Modeling Concepts

System, Model, View

- System
  - Organized set of communicating parts.
  - Engineered systems vs natural systems
  - Examples: car, watch, payroll system

System, Model, View

- Subsystem
  - Parts of a system.
  - Can be considered as simpler systems.
  - Examples: The engine of a car, integrated circuit of watch, mainframe computer of payroll system

System, Model, View

- Subsystem decomposition
  - Decomposing systems into its subsystems.
  - Decomposition is recursive.
    - Applied until a subsystem can not be decomposed.
  - End of recursion of subsystem decomposition → Object

System, Model, View

- Modeling
  - Helps dealing with complex systems.
  - Constructing an abstraction of a system.
  - Focuses on issues relevant to the problem and ignores irrelevant details.
  - Focuses on building a model that is simple enough for a person to grasp completely.
    - A rule of thumb is that each entity should contain at most 7 ± 2 parts.

System, Model, View

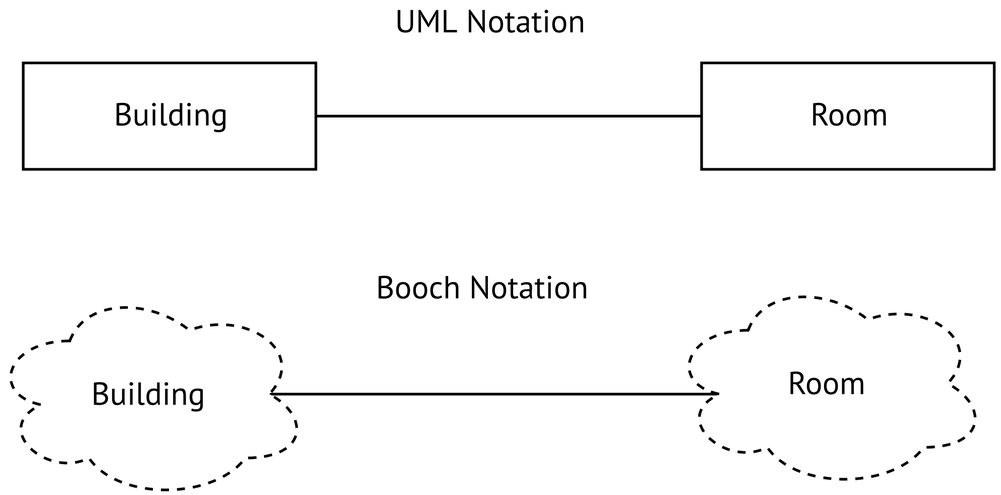
- System model
  - The set of all models built during development is called the system model.
  - The model usually precedes the system.
  - Coding before modeling may result inefficient development procedure.

System, Model, View

- View
  - A model may become so complex that it is not easily understandable.
  - A view focuses on a subset of a model to make it understandable.

System, Model, View

- Notation
  - Graphical or textual rules for representing views.
  - Class diagram → A graphical view of the object model



Data Type, Abstract Data Type, Instance

- Data type
  - Abstraction of data.
  - Has a unique name that distinguishes it from other data types.
  - Denotes a set of values that are members of the data type. → **Instances**
  - Defines the structure and the operations valid in all instances of the data type.
  - Used in typed languages to ensure that only valid operations are applied to specific instances.

Data Type, Abstract Data Type, Instance

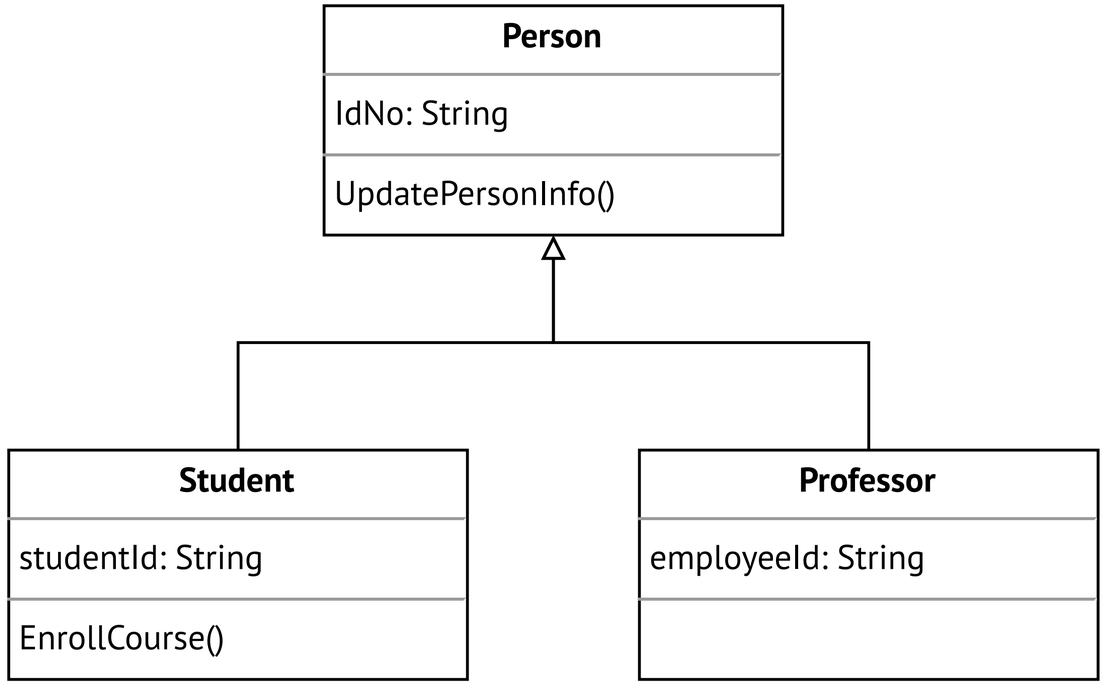
- Abstract data type
  - A data type defined by an implementation-independent specification.
  - Enables developers to reason about a set of instances without looking at a specific implementation of the abstract data type.
  - Examples: set, sequence

Class, Abstract Class, Object

- Class
  - Abstraction in object-oriented programming languages.
  - Encapsulates both structure and behavior.
  - Can be defined in terms of other classes by using **inheritance**.

Class, Abstract Class, Object

- Class: Inheritance example
  - Person → Student, Professor
  - Generalization class
    - **superclass** → Person
  - Specialized class
    - **subclass** → Student, Professor



Class, Abstract Class, Object

- Abstract class
  - A class that is not meant to be instantiated.
  - Represent generalized concepts in the application domain.
  - Their names are italicized.
  - Not all generalizations are abstract classes.
  - In Java, Collection is an abstract class.

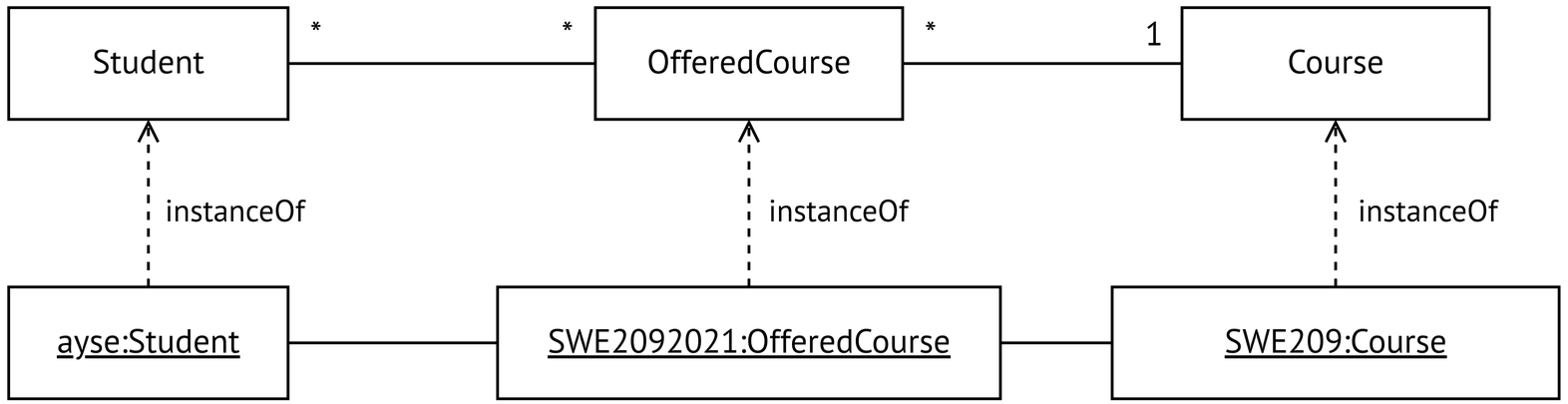
Class, Abstract Class, Object

- Operation
  - Applied to the instances of classes.
  - Defines the functionality of the instances.
  - Operations of a superclass can be inherited.
- Attribute
  - Applied to the instances of classes.
  - A named slot in the instance where a value is stored.

Class, Abstract Class, Object

- Object
  - An instance of a class.
  - Has an identity and stores attribute values.
  - Each object belongs to exactly one class.

Class, Abstract Class, Object



Event Class, Event, Message

- Event class
  - Abstraction representing a kind of event for which the system has a common response.
- Event
  - An instance of an event class
  - A relevant occurrence in the system.

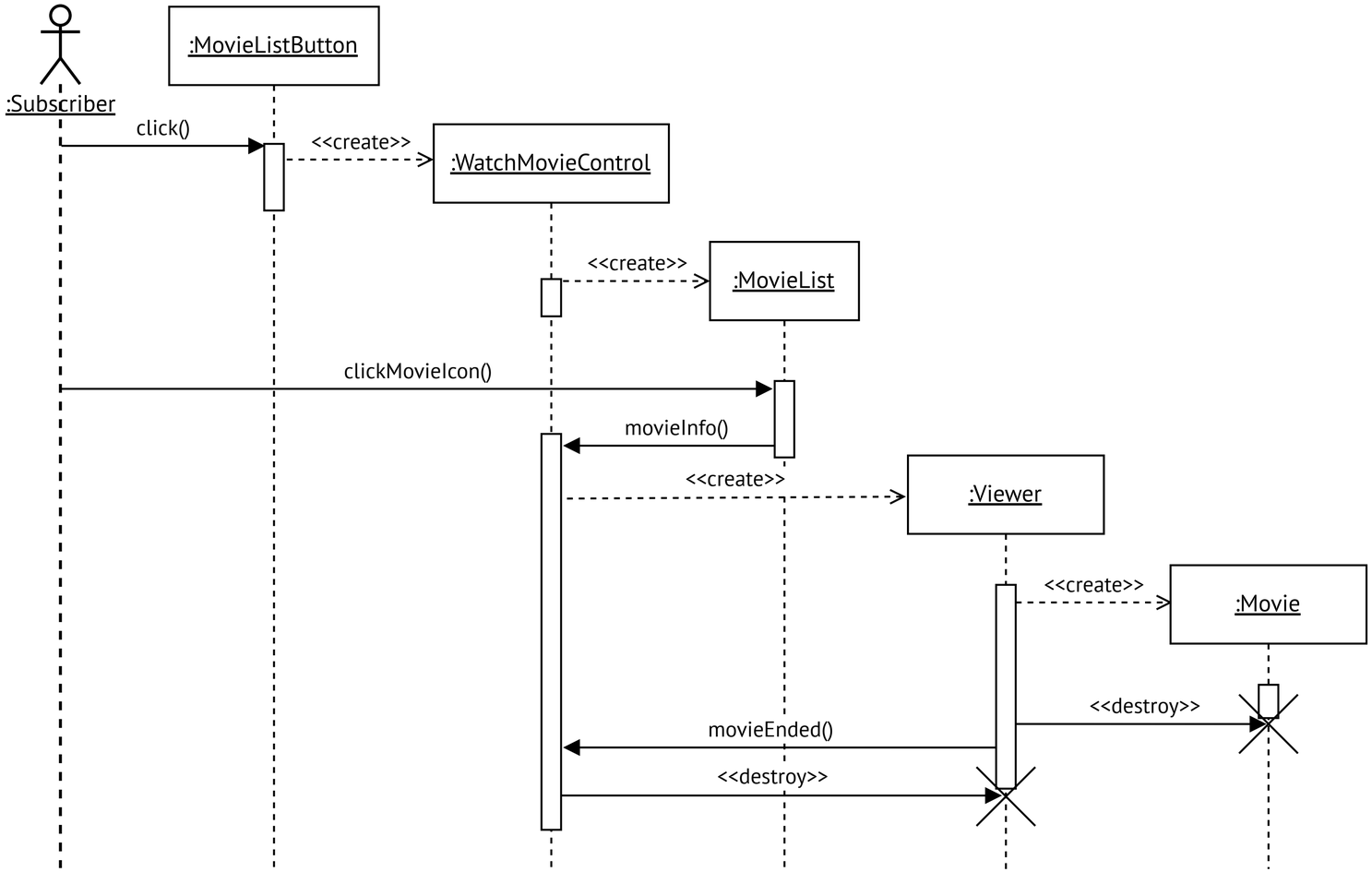
Event Class, Event, Message

- Event
  - An event can be;
    - A stimuli from an actor
    - A time-out
    - Sending of a message between two objects.

Event Class, Event, Message

- Sending a **message** → Mechanism
  - Requesting the execution of an operation in the receiving object.
  - Message
    - Name
    - Argument

Event Class, Event, Message



Object-Oriented Modelling

- Application domain
  - Represents all aspects of the user’s problem.
    - Physical environment
    - The users and other people
    - Work processes, etc.
  - Changes over time.

Object-Oriented Modelling

- Solution domain
  - The modeling space of all possible systems.
  - Modeling in the solution domain represents the system design and object design activities.
  - The solution domain model is much richer and more volatile than the application domain model.
  - The deployment of the system can change the application domain.

Object-Oriented Modelling

- **Object-oriented analysis** → Modeling the application domain
- **Object-oriented design** → Modeling the solution domain
- Object-oriented analysis and design → OOAD
- Object-oriented software engineering → OOSE
- In OOAD/OOSE, the application domain model is also part of the system model.

Falsification, Prototyping

- Falsification
  - The process of demonstrating that;
    - Relevant details have been incorrectly represented or not represented in the model.
    - The model does not correspond to the reality it is supposed to represent.

Falsification, Prototyping

- Prototype
  - Used to help applying falsification in software systems.
  - Construct a prototype that only simulates a part of the system.
  - Present to potential users for evaluation (falsification).
  - Modify subsequently.

References

- Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.
- Object Management Group, OMG Unified Modeling Language Superstructure, Version 2.2., http://www.omg.org/2009.

Thank you.