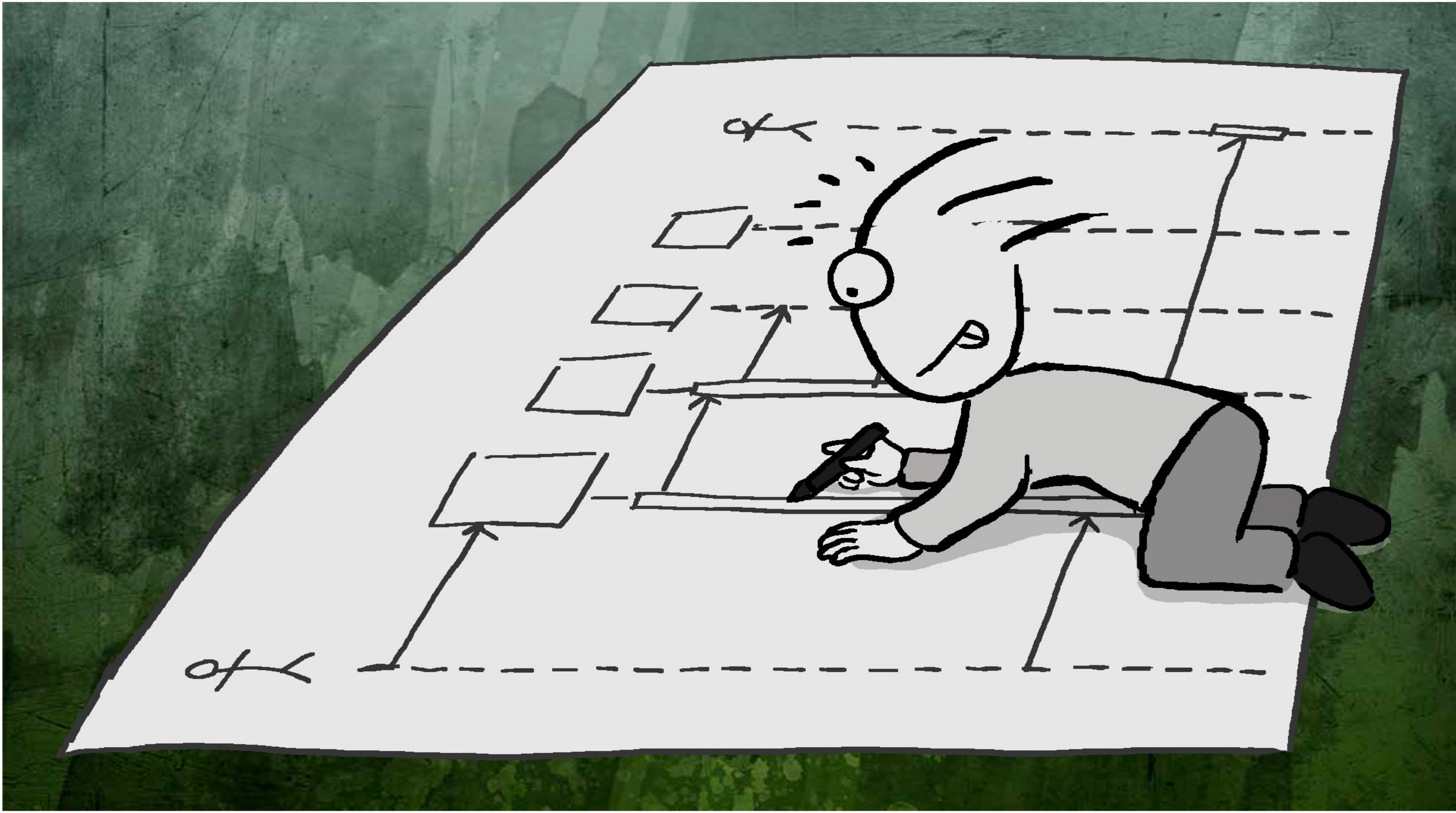


CPEN 321 | Software Engineering | Winter 2023 | UBC

CPEN 321

W2 L2: UML

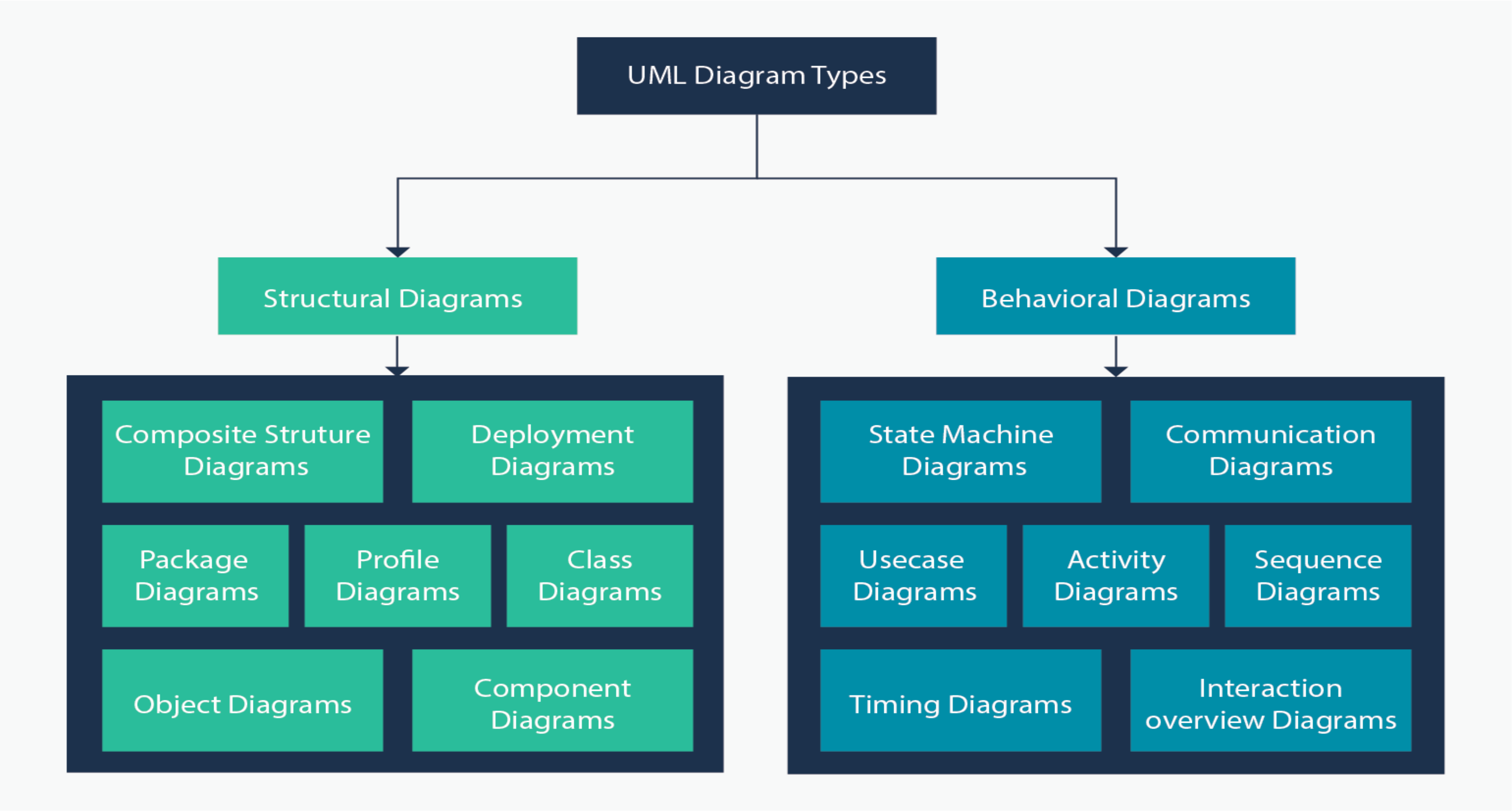
What is UML?

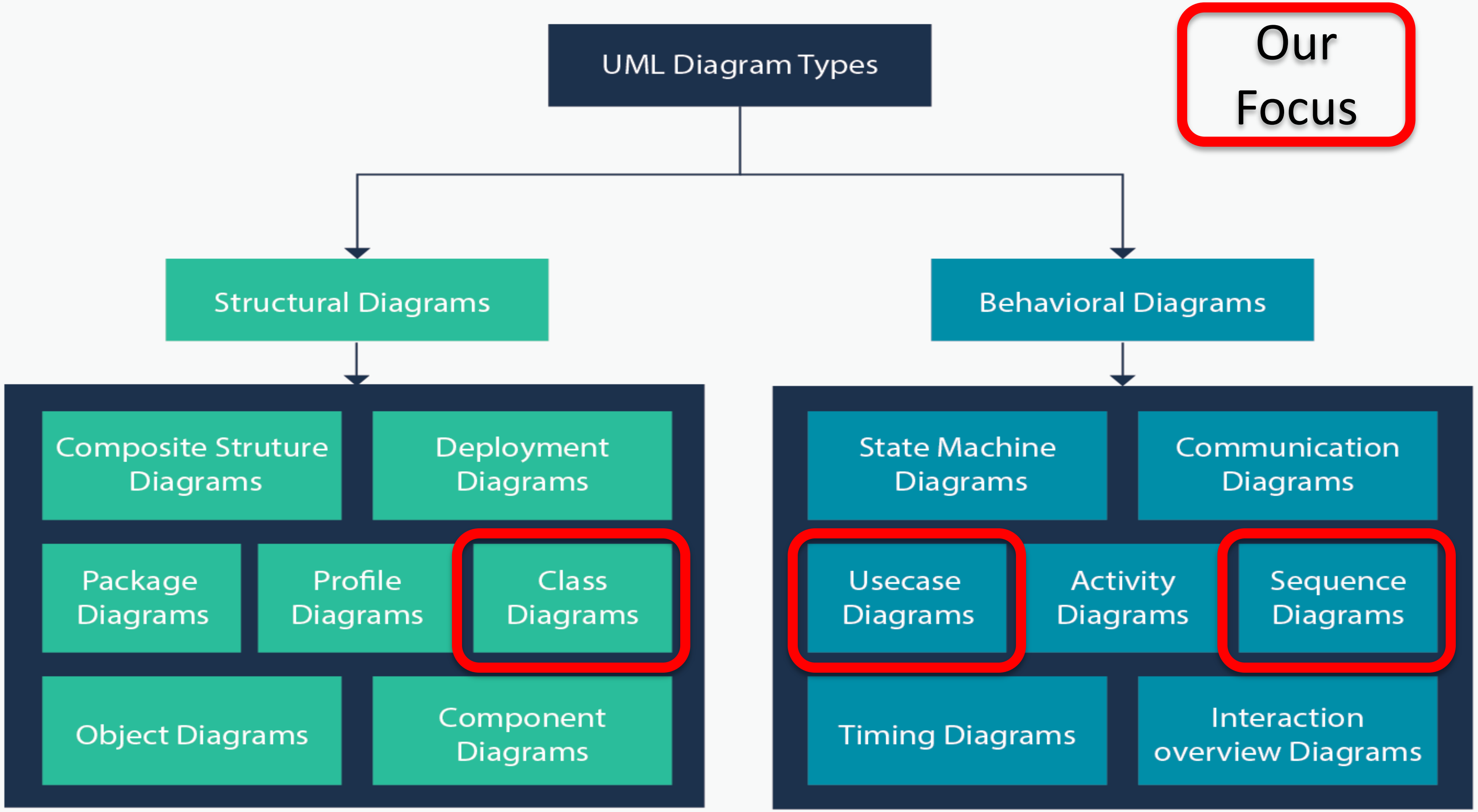
- Unified Modeling Language
- Maintained by Object Management Group (OMG) as a standard
  - www.OMG.org
- Provides a means to specify, model, and document a software system
- Process and programming language independent
- Mostly uses diagrams (visual notations)
- Lingua franca for many software engineers

UML Diagrams

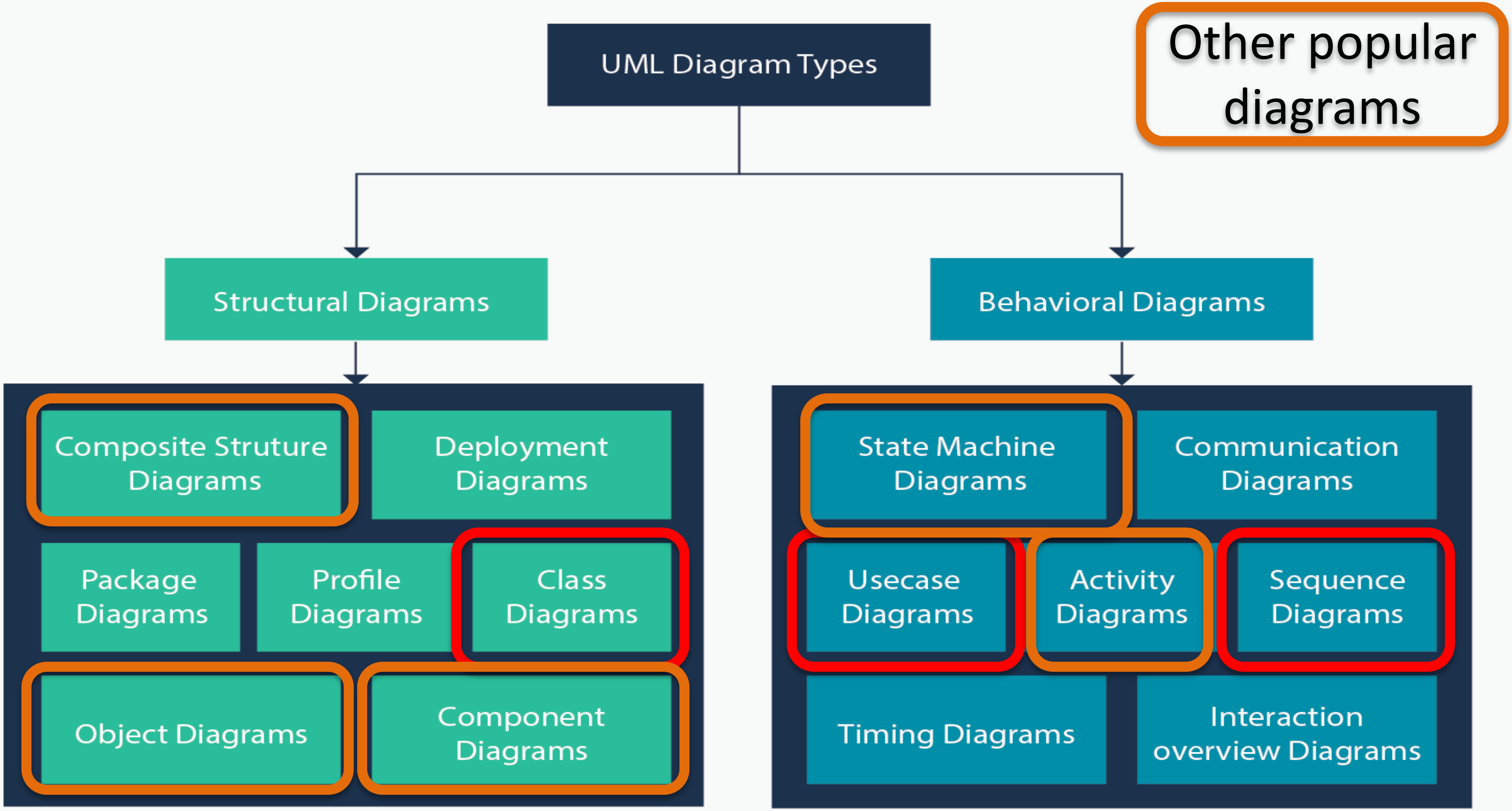
- UML diagrams are used for capturing different aspects of (structural and behavioral) design

- Used for
  - requirements
  - systems architecture
  - program design
  - etc.


UML Diagram Types
Structural Diagrams
Behavioral Diagrams
Composite Struture Diagrams
Deployment Diagrams
Package Diagrams
Profile Diagrams
Class Diagrams
Object Diagrams
Component Diagrams
State Machine Diagrams
Communication Diagrams
Usecase Diagrams
Activity Diagrams
Sequence Diagrams
Timing Diagrams
Interaction overview Diagrams


UML Diagram Types
Structural Diagrams
Behavioral Diagrams
Composite Struture Diagrams
Deployment Diagrams
Package Diagrams
Profile Diagrams
Class Diagrams
Object Diagrams
Component Diagrams
State Machine Diagrams
Communication Diagrams
Usecase Diagrams
Activity Diagrams
Sequence Diagrams
Timing Diagrams
Interaction overview Diagrams

Our Focus



UML Diagram Types

Structural Diagrams

Composite Struture Diagrams

Deployment Diagrams

Package Diagrams

Profile Diagrams

Class Diagrams

Object Diagrams

Component Diagrams

Behavioral Diagrams

State Machine Diagrams

Communication Diagrams

Usecase Diagrams

Activity Diagrams

Sequence Diagrams

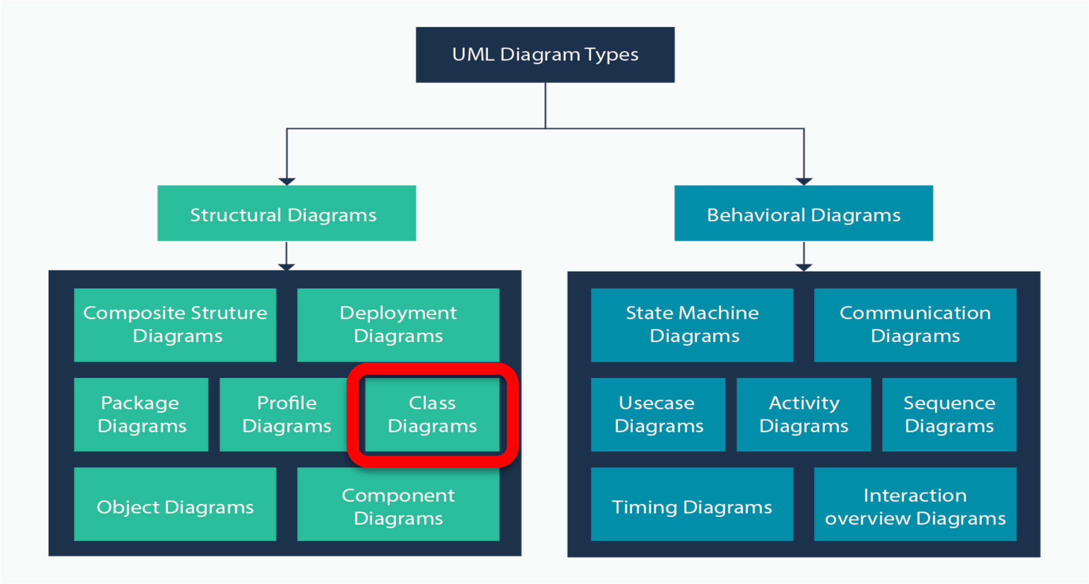
Timing Diagrams

Interaction overview Diagrams

Other popular diagrams

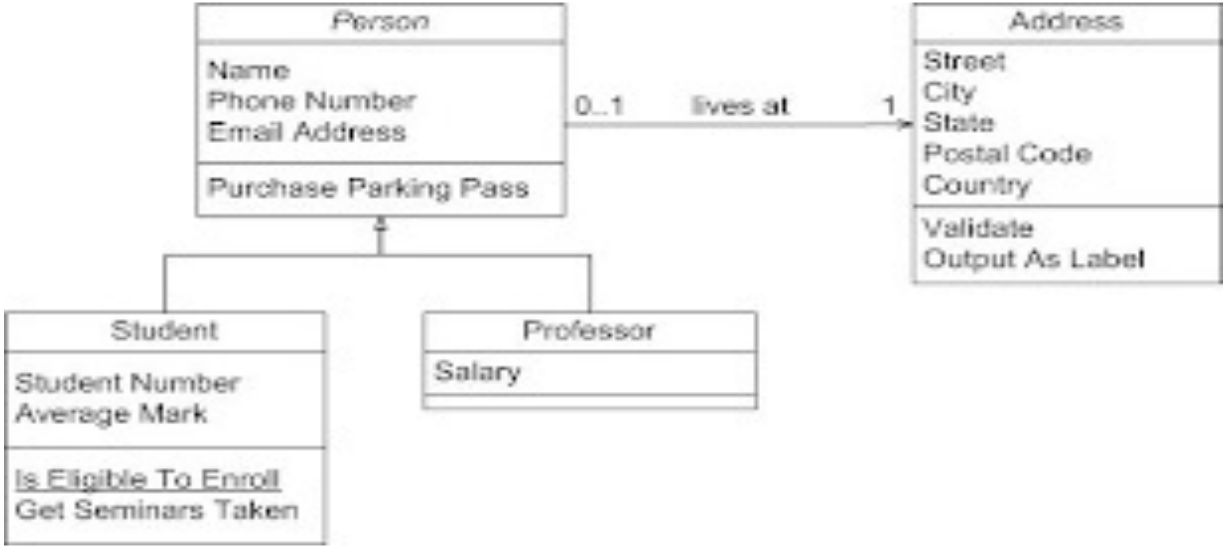
Outline

- **Class diagram**
- Use case diagram
- Sequence diagram



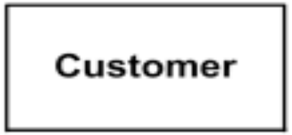
Class Diagram

- Shows the classes in a system and the relationships between these classes
- Particularly useful for OO systems, but can also represent modules and other types of components

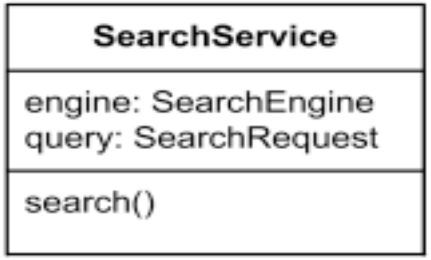


Class Diagram – Main Concepts

- Class: a rectangle showing the name of the class



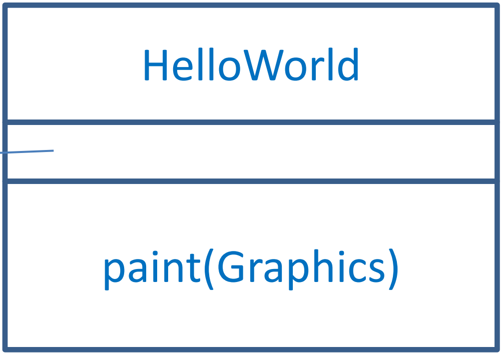
- Can contain two additional compartments:
  - Attributes (local variables)
  - Operations (methods)



Class Diagram – Example

```
import java.awt.Graphics;

class HelloWorld extends java.applet.Applet
{
   public void paint(Graphics g) {
      g.drawString("Hello, World!", 10, 10);
      // ...
   }
}
```

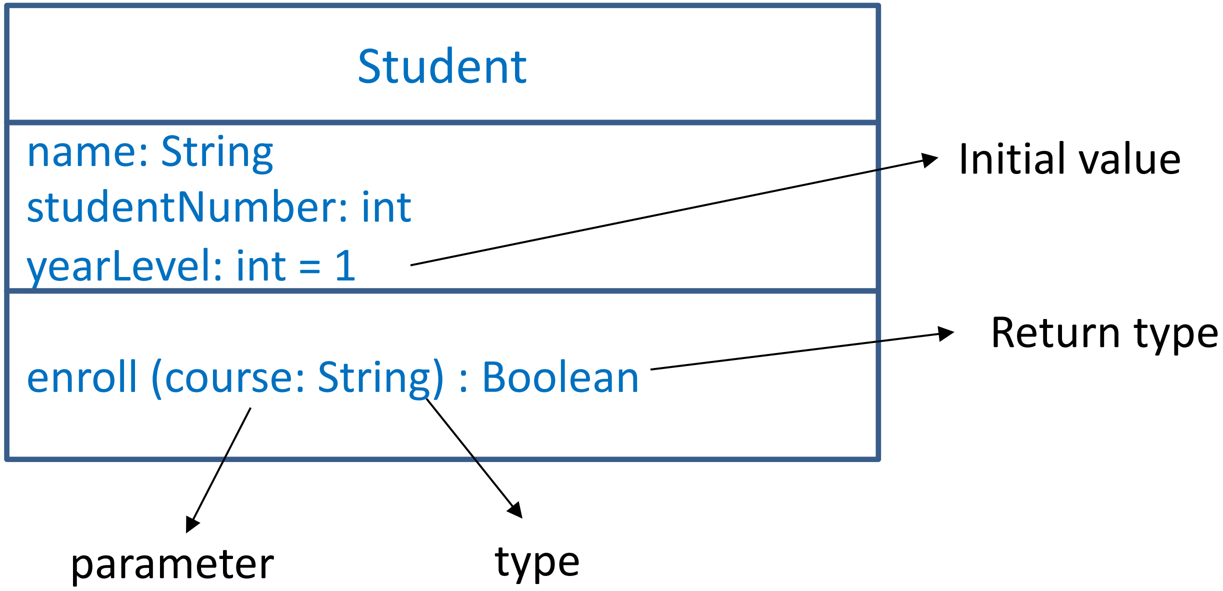


If no attributes, leave the compartment blank

Class – more info

- A more complete class diagram may also include:
  - the type of the variables (attributes) and initial values
  - the parameters, types, and the return type of a method

- Example:



Student

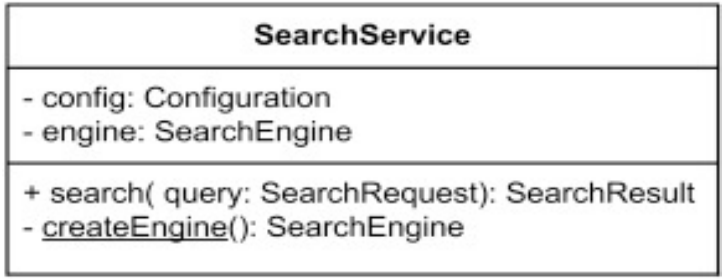
name: String
studentNumber: int
yearLevel: int = 1

enroll (course: String) : Boolean

Visibility Symbols

- Visibility of attributes/ operations:

SYMBOL	MEANING	EXPLANATION
+	Public	The member is visible to all code in the application.
–	Private	The member is visible only to code inside the class.
#	Protected	The member is visible only to code inside the class and any derived classes.
~	Package	The member is visible only to code inside the same package.



SearchService

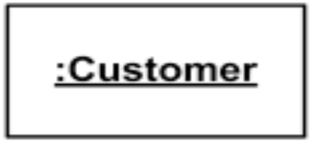
- config: Configuration
- engine: SearchEngine

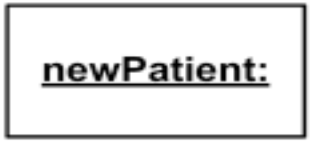
+ search(query: SearchRequest): SearchResult
- createEngine(): SearchEngine

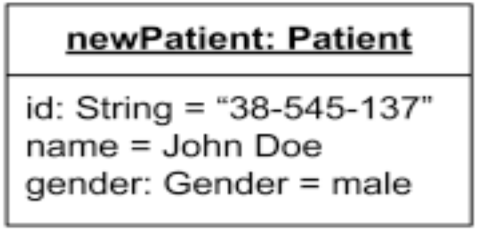
Object

- An instance of a class
- Can optionally contain valuation of fields

- Examples:
  - An unnamed instance of the customer class
  - An instance named *newPatient* of some unnamed or unknown class
  - Instance *newPatient* of the *Patient* class with values specified

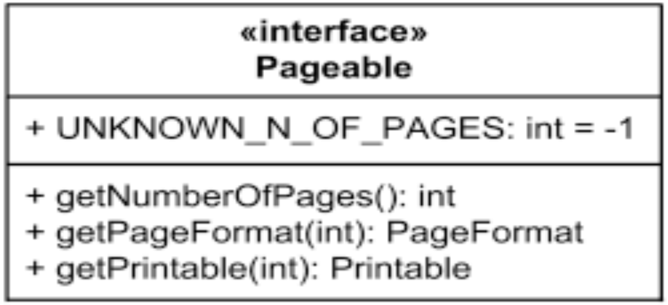






Interface

- Specifies a contract
- Any instance of a classifier that realizes (implements) the interface must fulfill that contract and thus provides services described by contract



«interface»
Pageable

+ UNKNOWN_N_OF_PAGES: int = -1

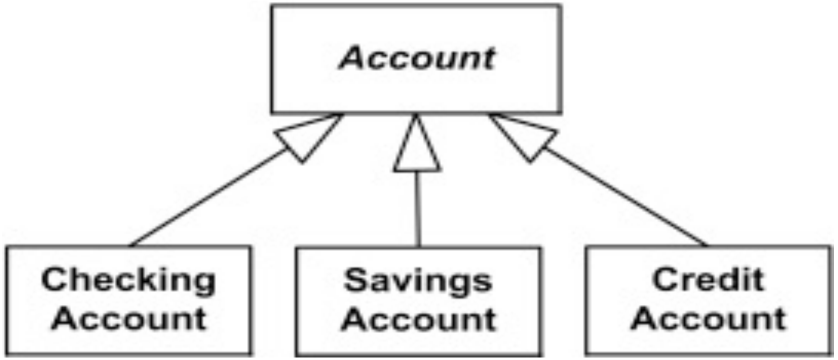
+ getNumberOfPages(): int
+ getPageFormat(int): PageFormat
+ getPrintable(int): Printable

In UML, both Class and Interface are instances of an abstract class called Classifier.

Main Relationships Between Classifiers

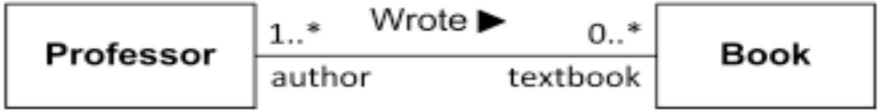
Generalization

- Informally called “inheritance” or “**is a**” relationship (as in “a Duck is a Bird”)
- Generalization is a directed relationship between a more general classifier (superclass, parent) and a more specific classifier (subclass, child).
- Note: Multiple inheritance is allowed in UML (but not in Java)



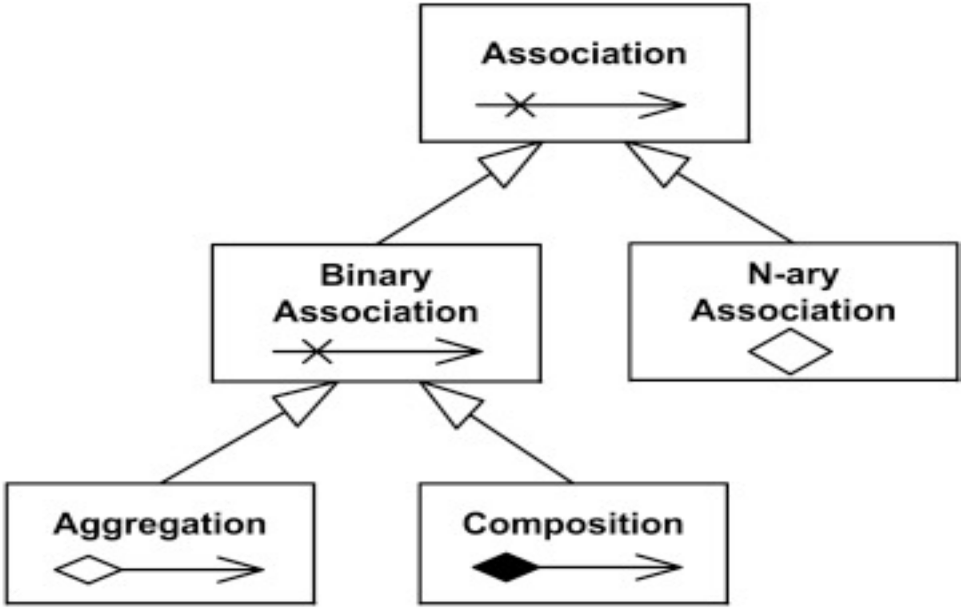
Association

- Describes the presence of a relationship between classes

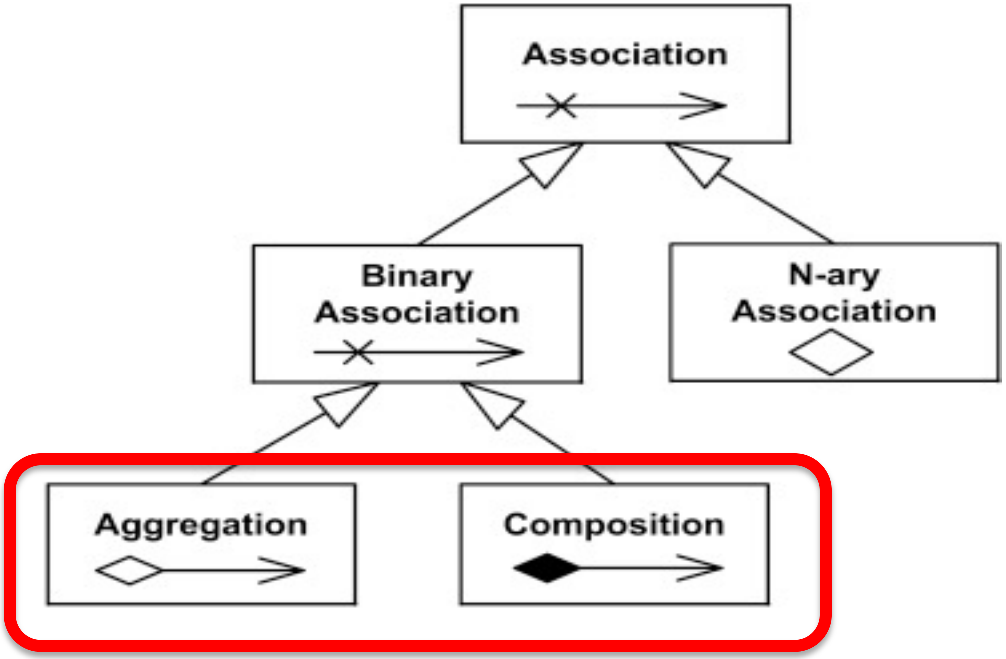


- Name of the association end and multiplicity may be placed near the end of the line
  - The association end name is commonly referred to as **role** (*Professor is an author of a book; A book is used as a textbook by a professor*)
  - Multiplicity (*every Book has at least one author; A professor can write any number of books, including none*)

Types of Association



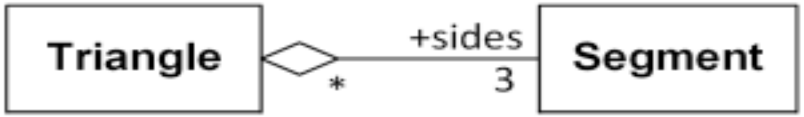
Types of Association



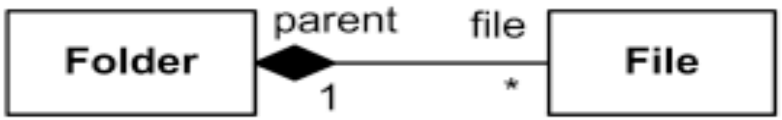
Aggregation and Composition

Whole/part association:

Aggregation:
(a weak form of whole/part)

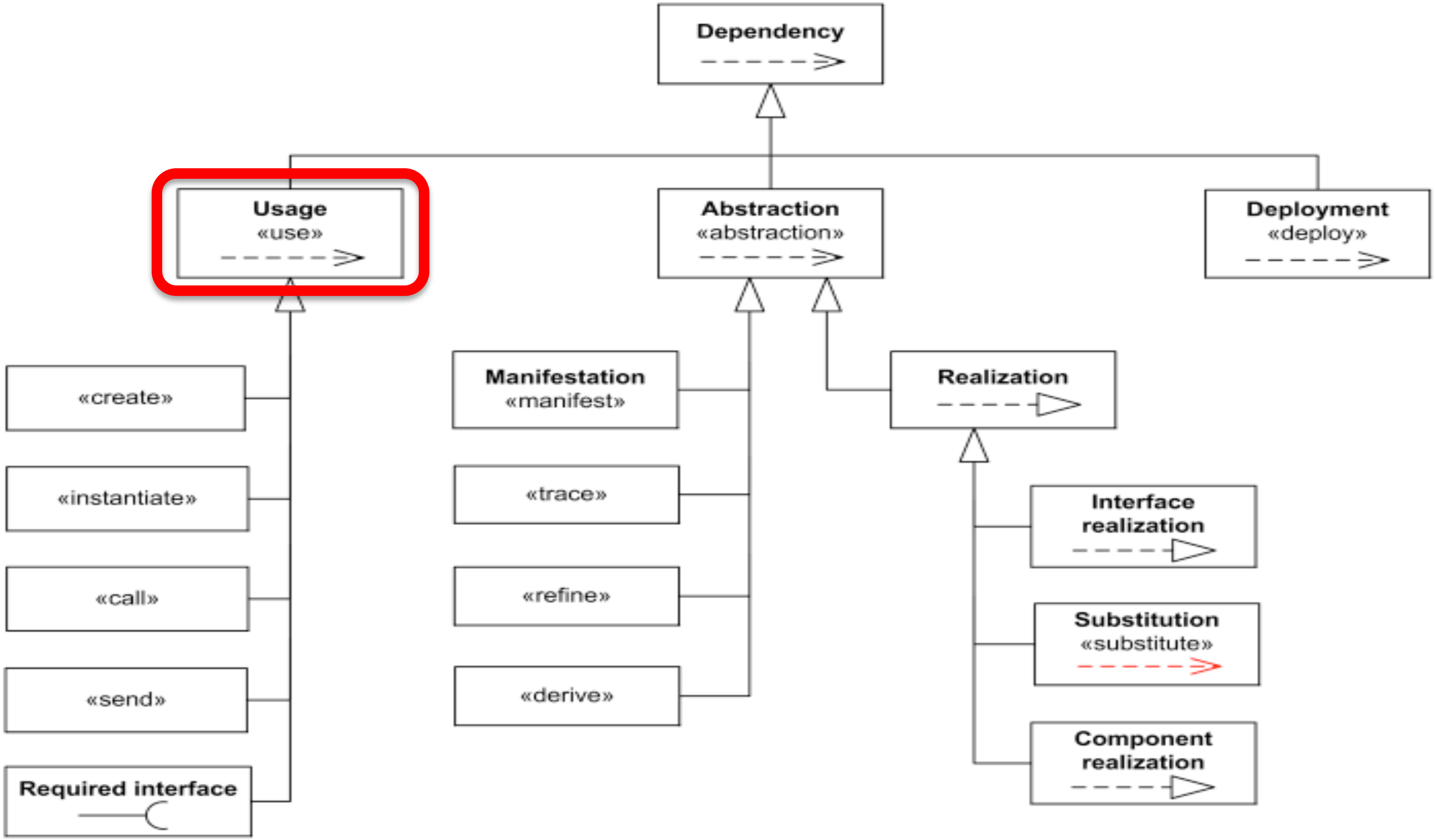


Composition:
(a strong form of whole/part)



- Only one end of association can be marked as aggregation or composition
- Aggregation / composition links should form a directed, acyclic graph, so that no instance are direct or indirect part of itself.

Dependencies



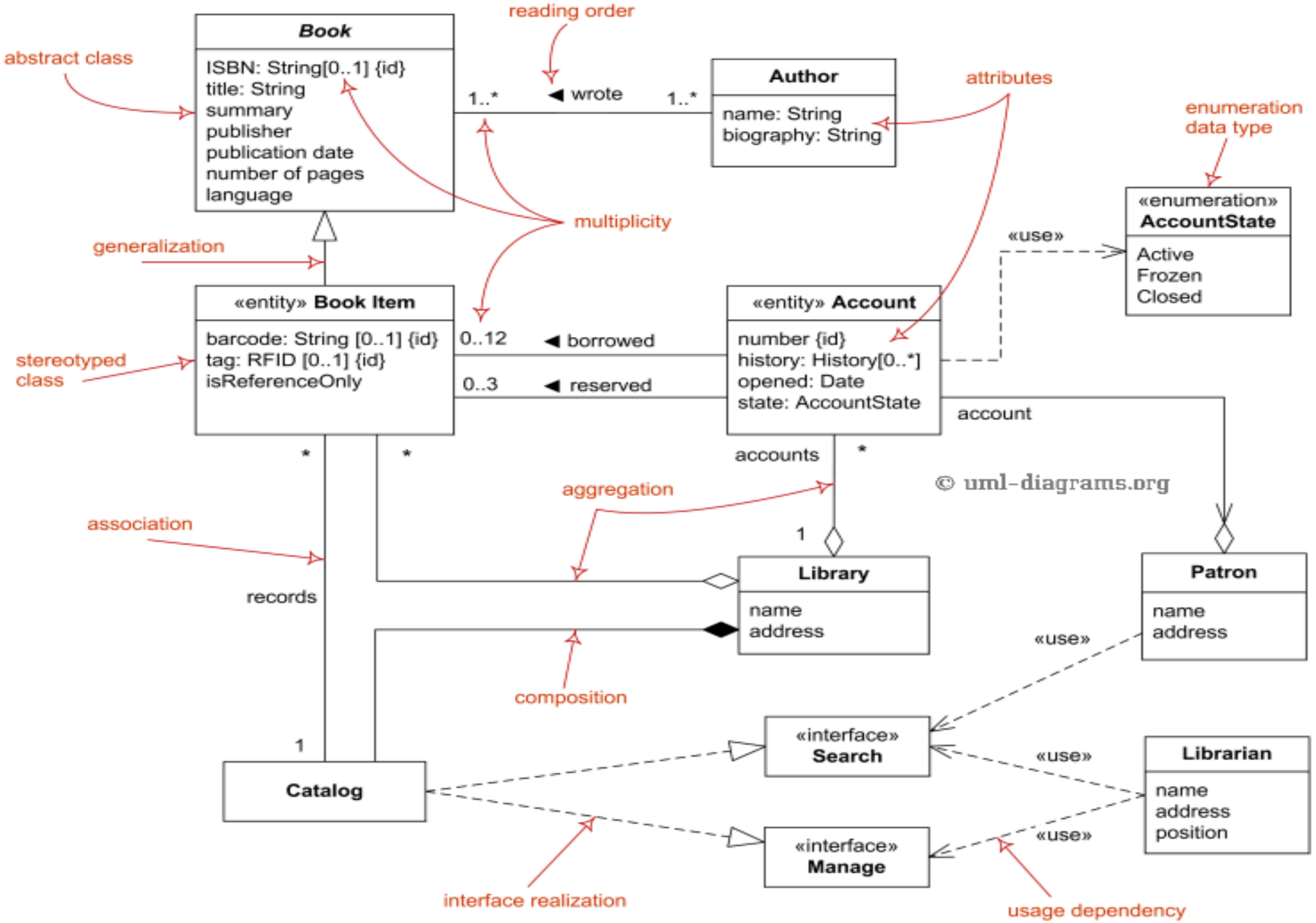
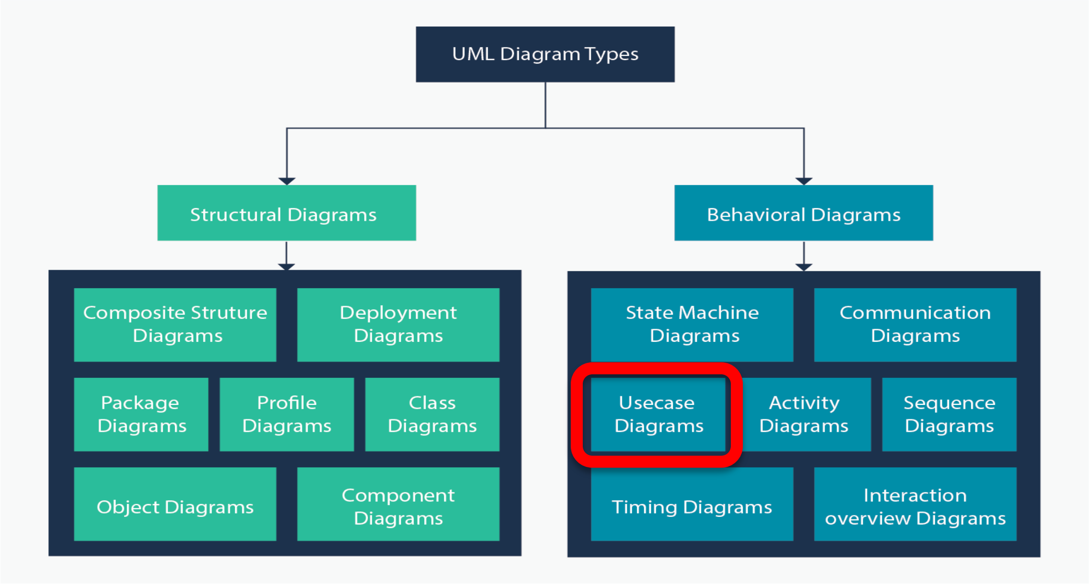


Image source: http://www.uml-diagrams.org/class-diagrams-overview.html

Outline

- Class diagram
- **Use case diagram**
- Sequence diagram



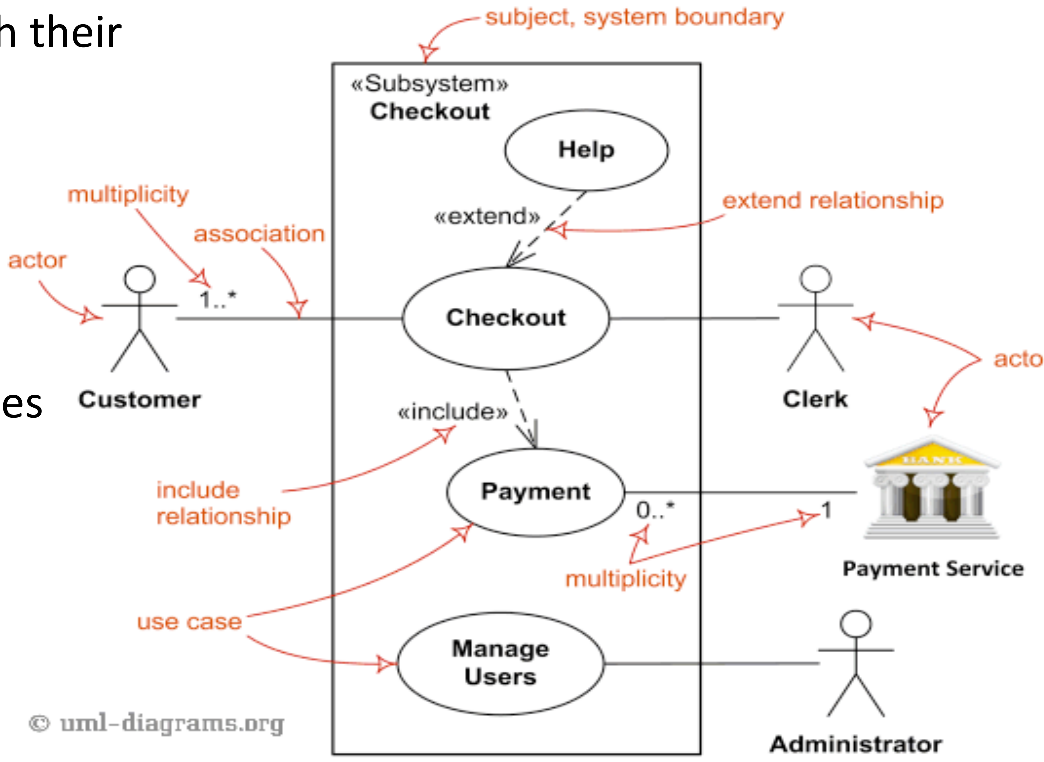
Use Case Diagram

- A representation of a user's interaction with the system

- Shows the relationship between the user and the different use cases **in which the user is involved**

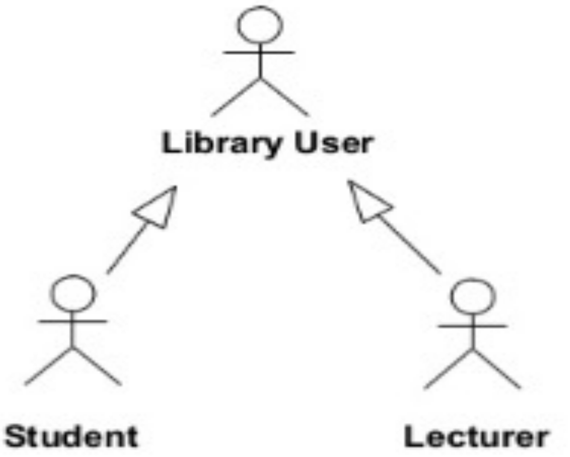
Use Case Diagram – Main Concepts

- Subject: describes the boundaries of the system
- Actors: stick-men (or other shapes), with their names (nouns)
- Use cases: ellipses, with their names (verbs)
- Line associations: connect an actor to a use case in which that actor participates
  - multiplicity



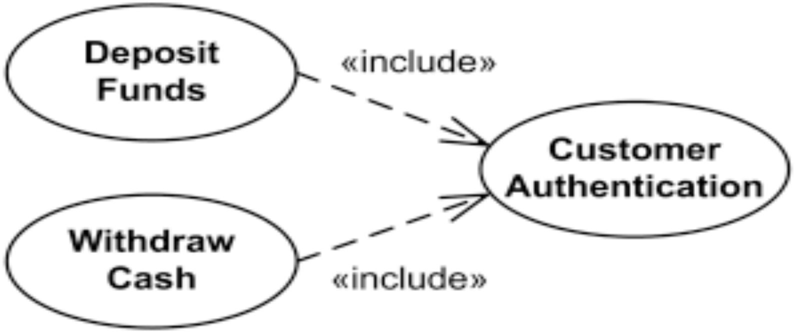
Relationships between actors

- Generalization: all use cases of the superclass actor are applicable to the subclass actor



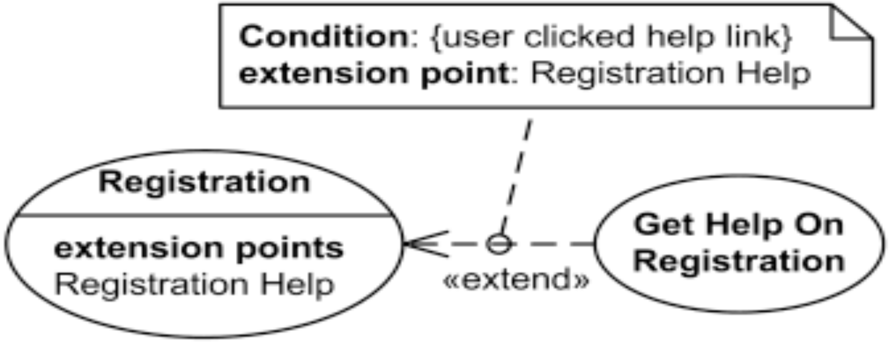
Relationships between use cases: Include

- The behavior of the **included** use case (Customer Authentication) is inserted into the behavior of the **including** use case (Deposit Funds)
  - Including use case cannot be complete without the included one
  - Commonly used to
    - simplify large use case by splitting it into several use cases
    - to extract common parts of the behaviors of two or more use cases.



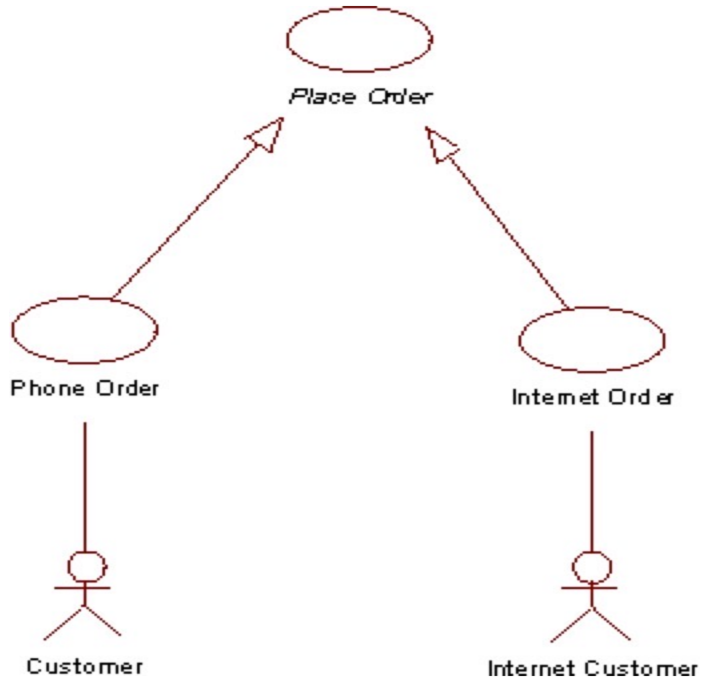
Relationships between use cases: Extend

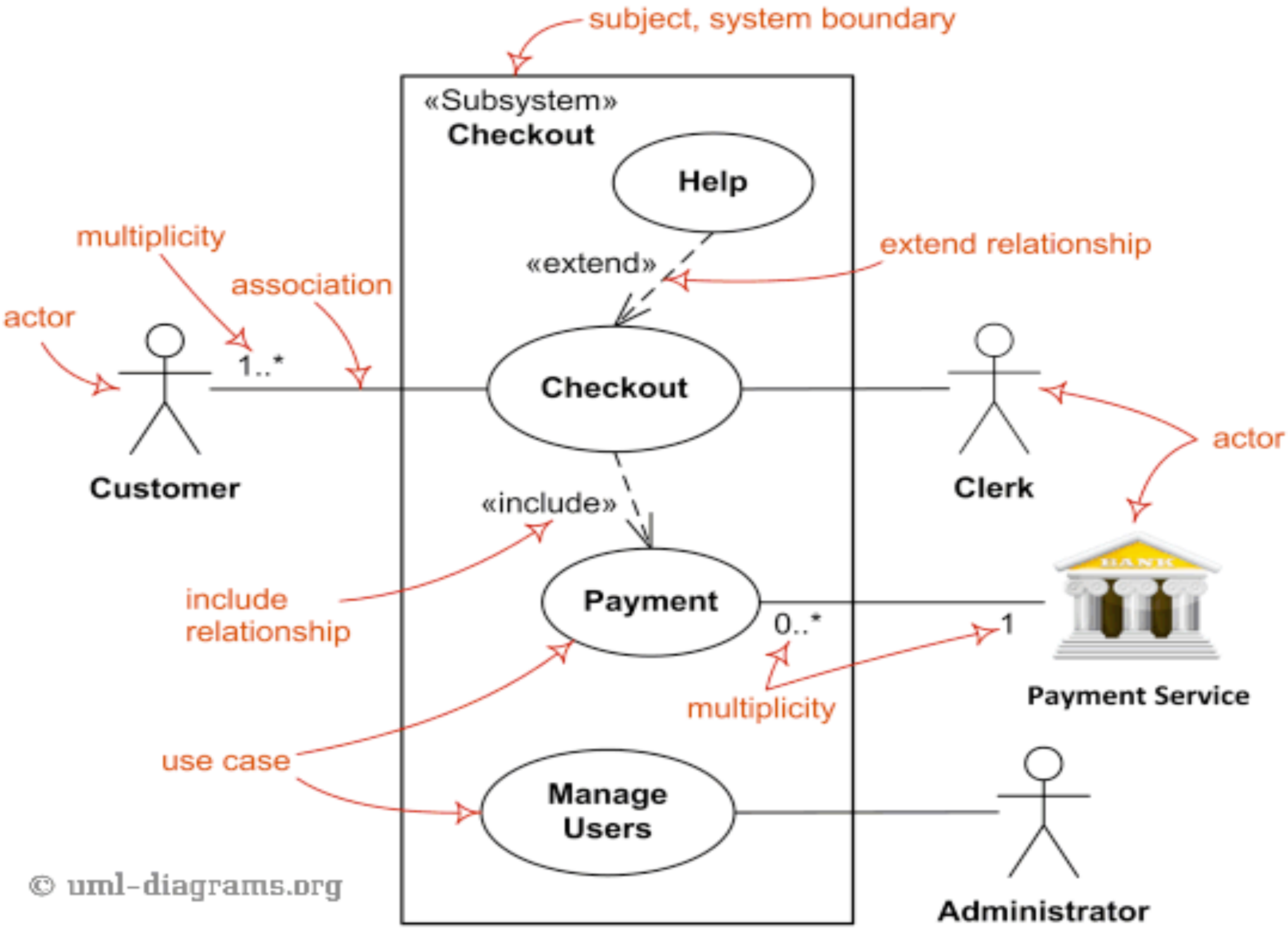
- The behavior of the **extending** use case can (optionally) be inserted into the behavior defined in the **extended** use case
  - Extended use cases can execute on its own
  - Insertion condition can be given
  - Commonly used to specify error handling and exceptional paths


Condition: {user clicked help link}
extension point: Registration Help
Registration
extension points
Registration Help
«extend»
Get Help On Registration

Relationships between use cases: Generalization

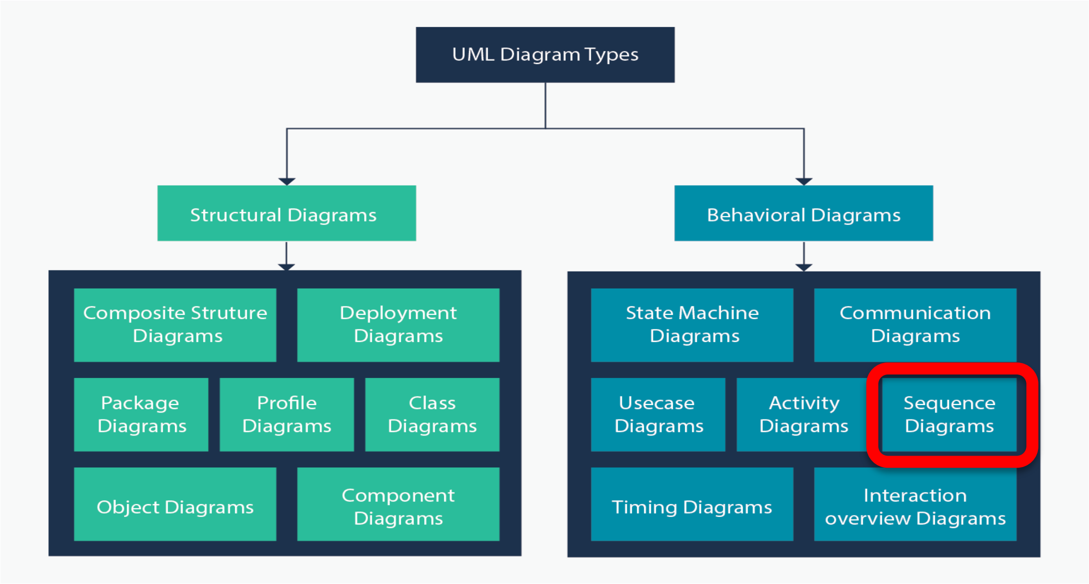
- Similar to the generalization of an actor.
- The behavior of the ancestor is inherited by the descendant.
    - Used when there is common behavior between two use cases and also specialized behavior specific to each use




subject, system boundary
«Subsystem»
Checkout
Help
multiplicity
extend relationship
«extend»
association
actor
1..*
Checkout
Customer
Clerk
actor
«include»
include relationship
Payment
0..*
1
multiplicity
Payment Service
use case
Manage Users
Administrator
© uml-diagrams.org

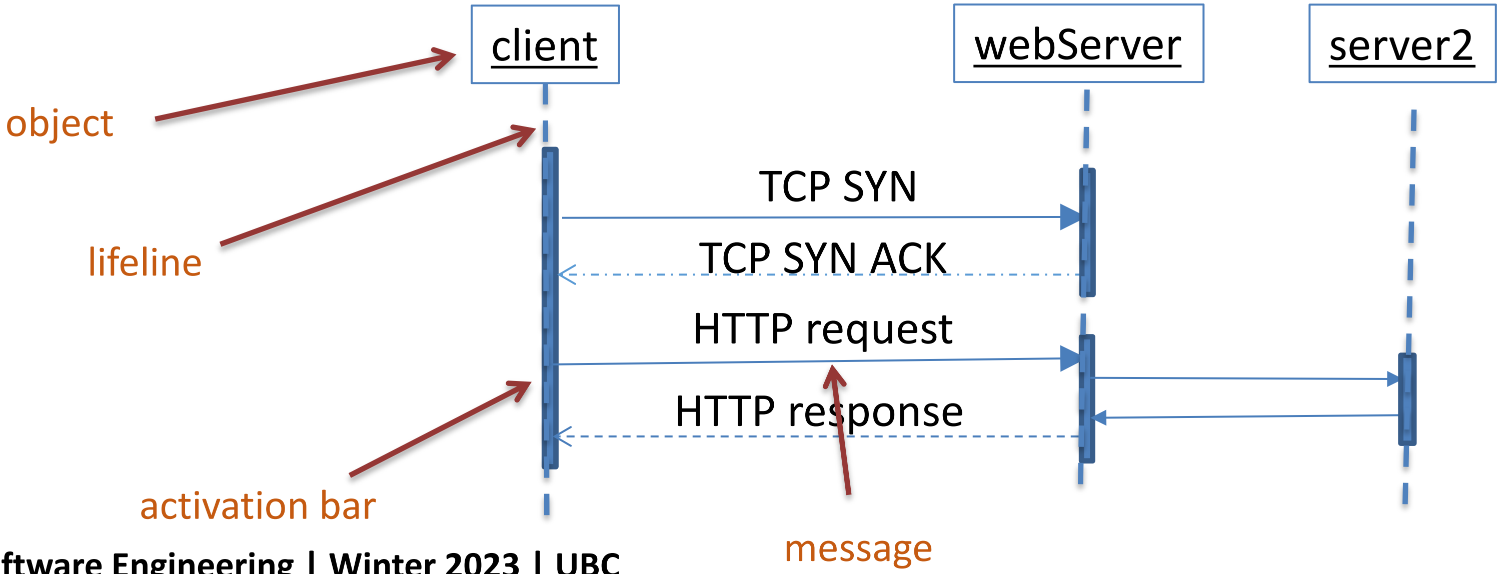
Outline

- Use case diagram
- Class diagram
- **Sequence diagram**



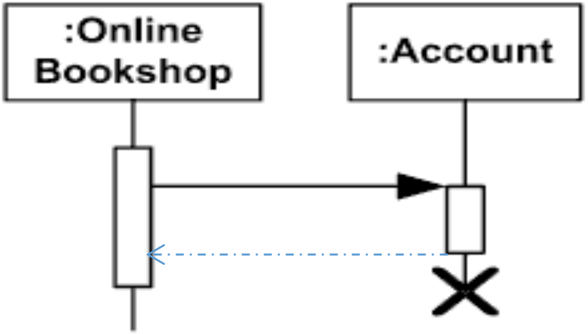
Sequence Diagram

- Represents the interactions of the objects in a system
- Usually considers small, discrete pieces of the system, e.g., individual scenarios or operations
- Time runs downward
- Example: a simplified sequence diagram of web browsing



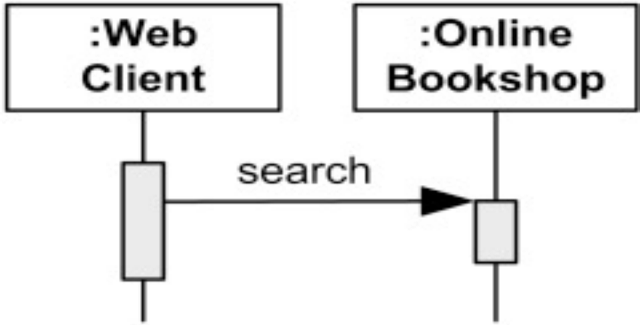
Sequence Diagram – Main Concepts

- An interaction: a set of messages exchanged by a set of objects to accomplish a specific purpose
  - A sequence diagram describes an interaction
- A lifeline represents an object involved in the interaction
- A message is represented by an arrow.
  - A call message uses a solid line.
  - An (optional) response message uses a dashed line
- An execution specification (a.k.a. activation bar) shows when the object is a

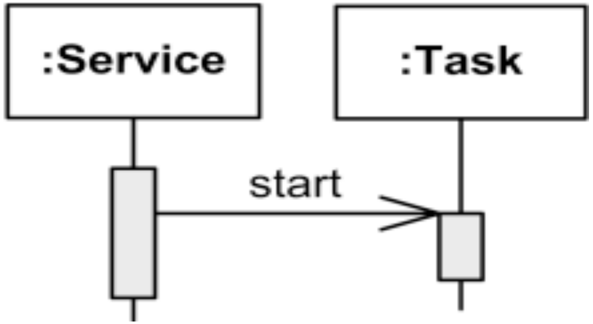


Synchronous vs. Asynchronous Messages

- A **synchronous call** represents an operation call – sends a message and suspends execution while waiting for response

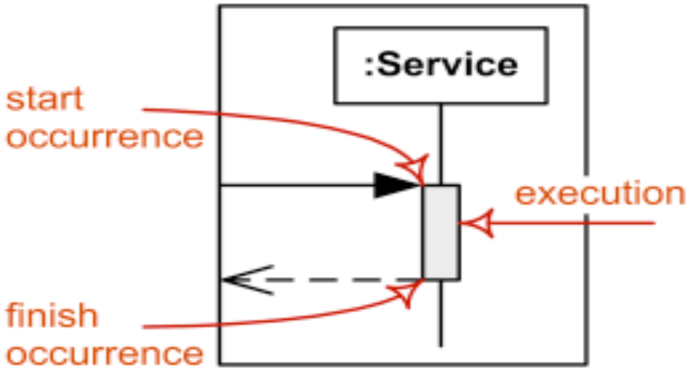


- An **asynchronous call** sends a message and proceeds immediately without waiting for return value



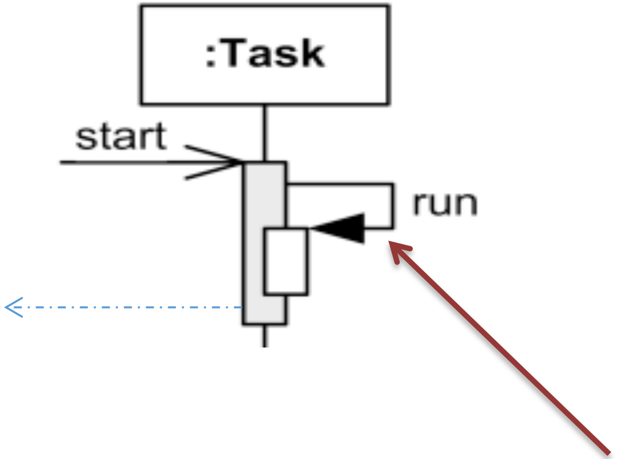
Execution Specification

- Represents a period in the participant's lifetime
  - when it is executing a unit of behavior or action within the lifeline
  - sending a signal to another participant
  - waiting for a reply message from another participant

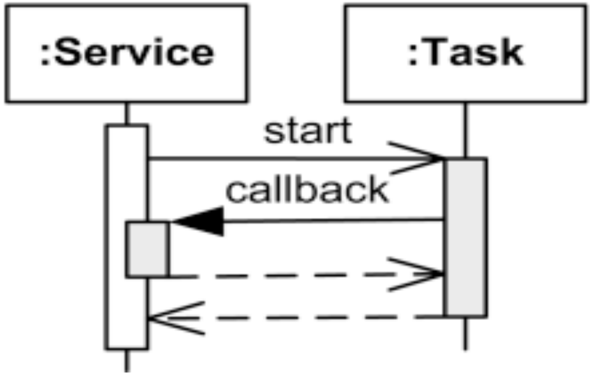


Overlaps

- Overlapping **execution specifications** on the same lifeline are represented by overlapping rectangles.

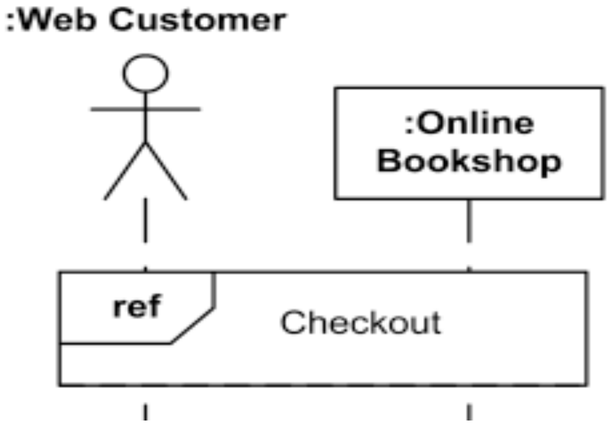


Message to self



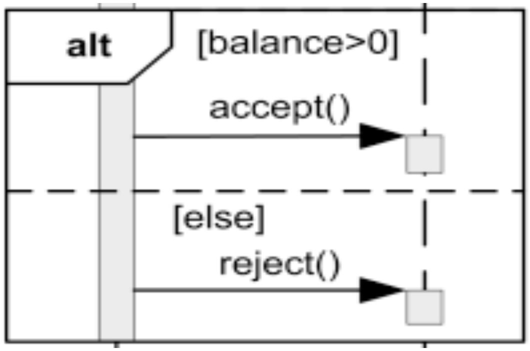
Interaction Fragments

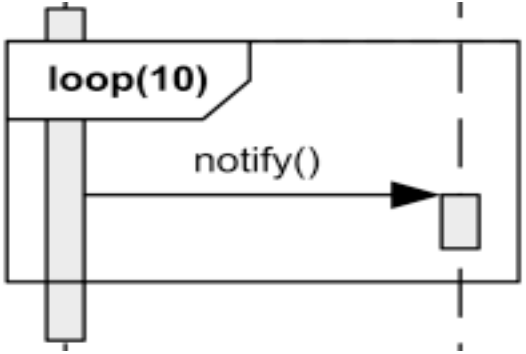
- **Interaction use:** an interaction fragment which allows to call another interaction
- Good for:
  - Simplifying large and complex sequence diagrams
  - Reusing some interaction between several other interactions

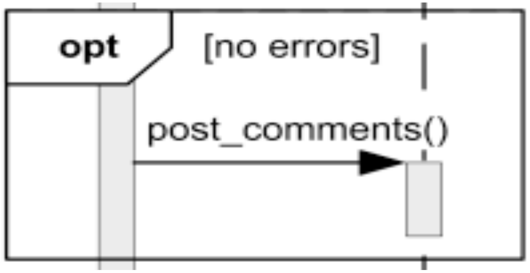


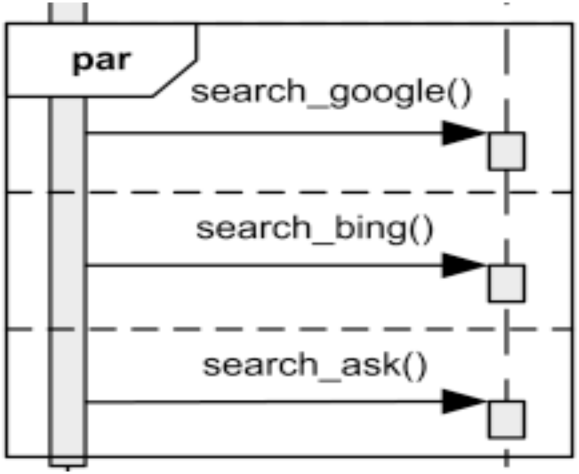
Additional Fragment Operators

- **alt** - alternatives
- **opt** - option
- **loop** - iteration
- **break** - break
- **par** - parallel
- **strict** - strict sequencing
- **seq** - weak sequencing
- **critical** - critical region
- **ignore** - ignore
- **consider** - consider
- **assert** - assertion
- **neg** - negative



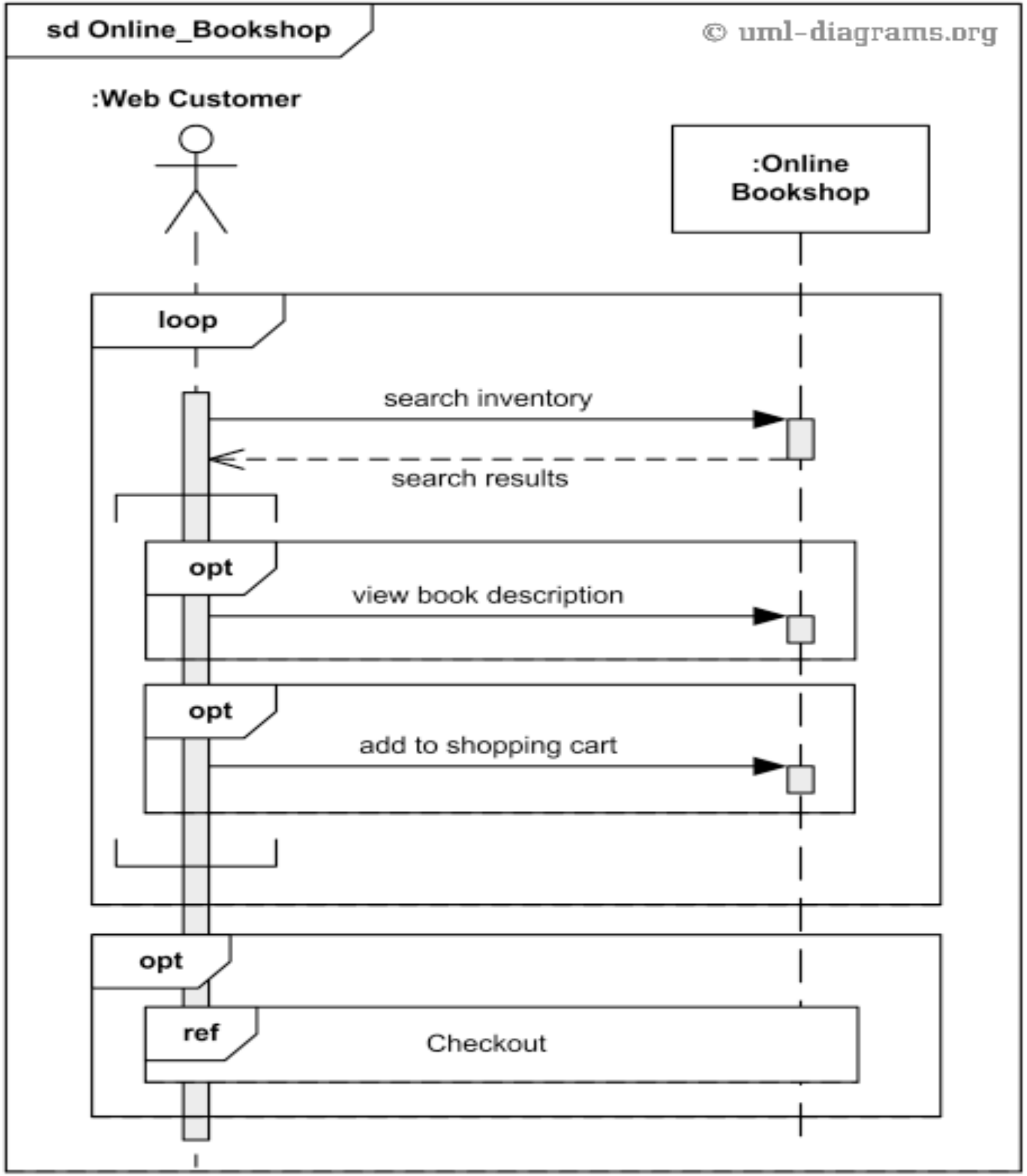






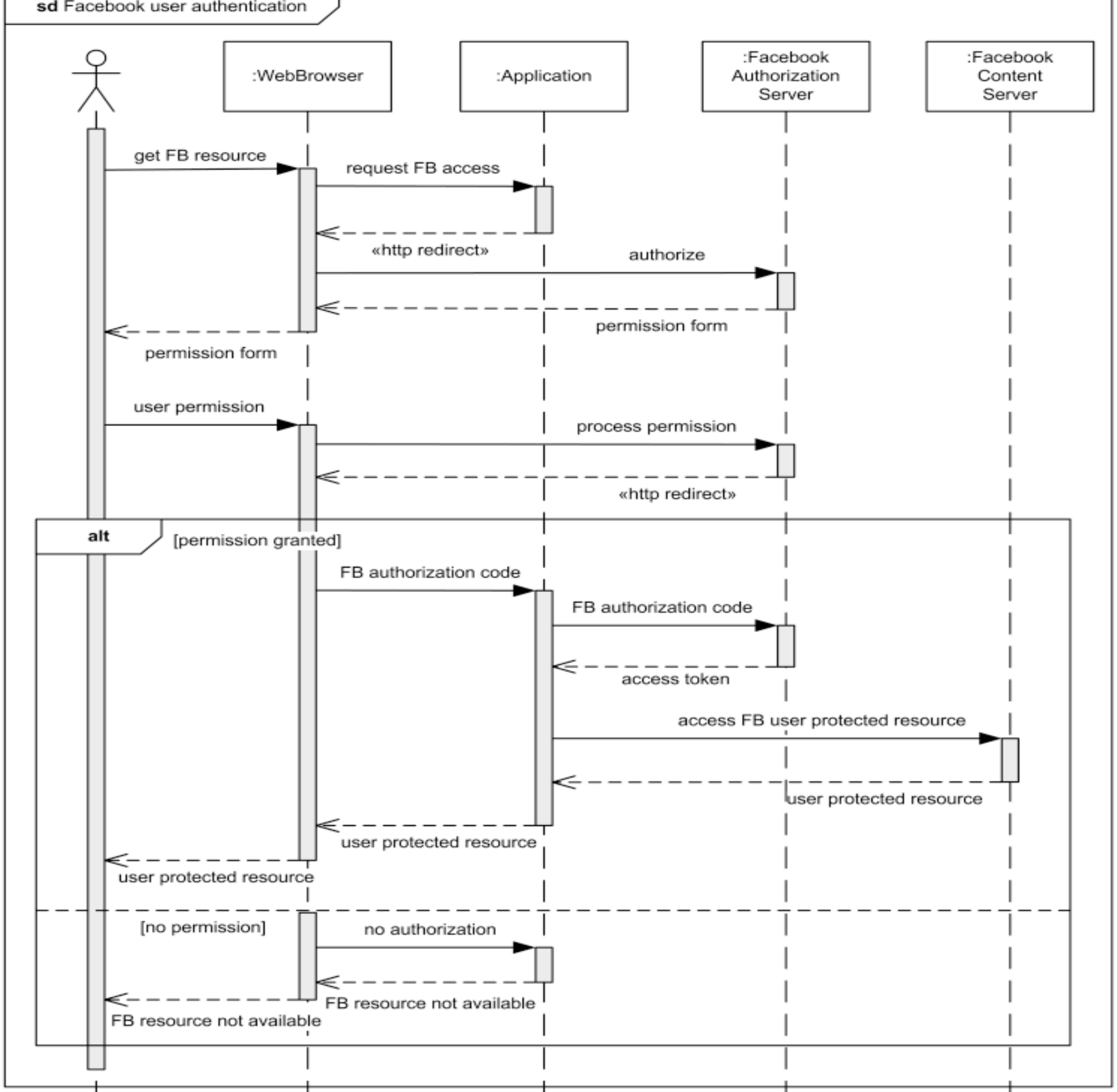
Example

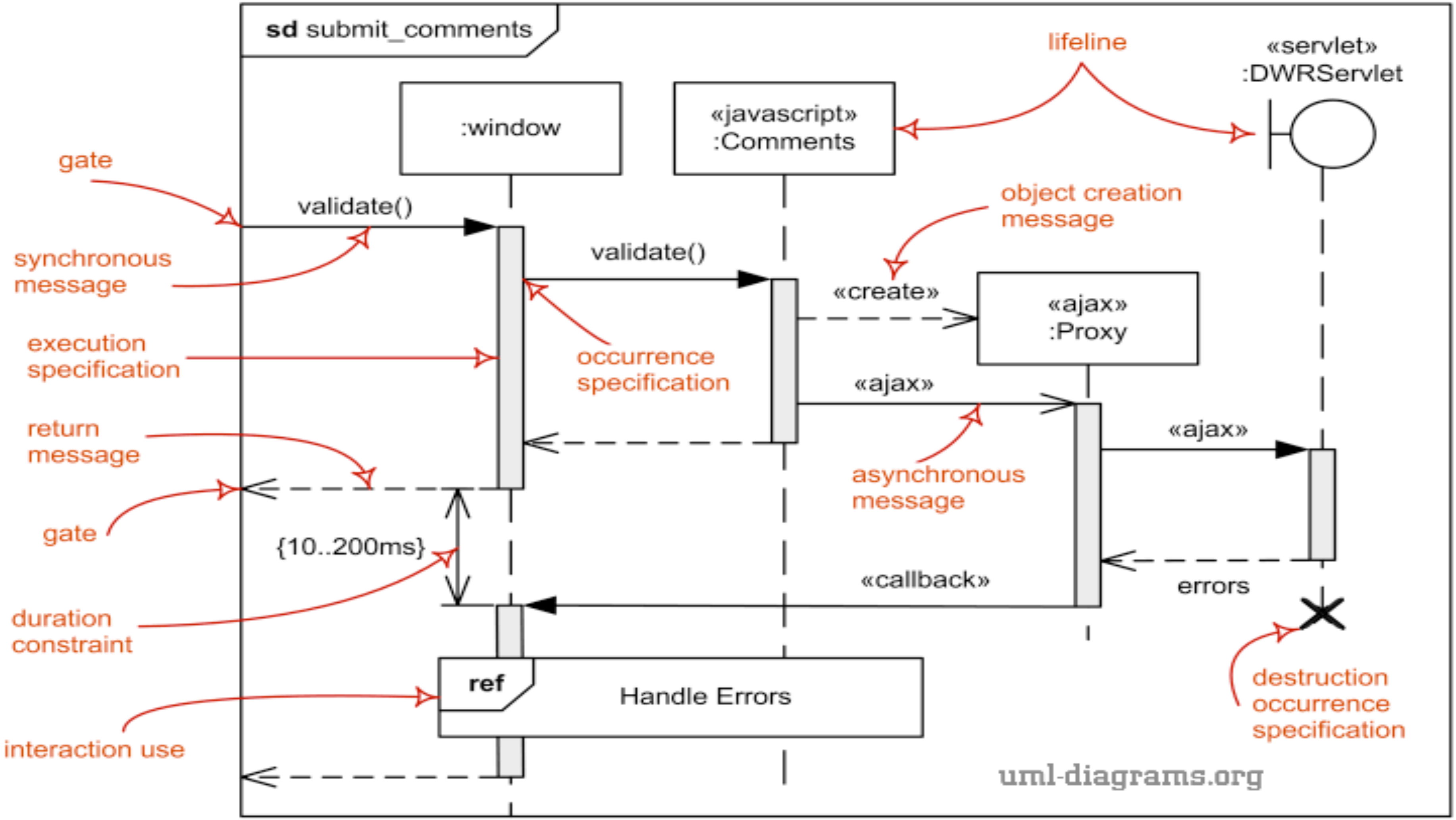
Sequence of interactions between a web customer and an online book shop



Example

Facebook user authentication




sd submit_comments
:window
«javascript»
:Comments
«servlet»
:DWRServlet
lifeline
gate
validate()
synchronous
message
validate()
object creation
message
«create»
«ajax»
:Proxy
execution
specification
occurrence
specification
«ajax»
«ajax»
return
message
asynchronous
message
gate
{10..200ms}
«callback»
errors
duration
constraint
ref
Handle Errors
destruction
occurrence
specification
interaction use
uml-diagrams.org

More Information

- https://sparxsystems.com/platforms/uml.html
- https://www.uml-diagrams.org/
- Wikipedia
- …

Please familiarize yourself with the syntax,
we will be using these diagrams

Summary

- Language to express system requirements and design
- Some diagrams are used more widely than others:
    - Simplified class diagrams
    - Use case diagrams
    - Sequence diagrams
    - Activity diagrams (flowcharts)
    - State machines (for full code generation, e.g., with IBM Rhapsody)
    - …
- Main benefits
    - Accurately specify design aspects to consider
    - Provide a standard language of communication