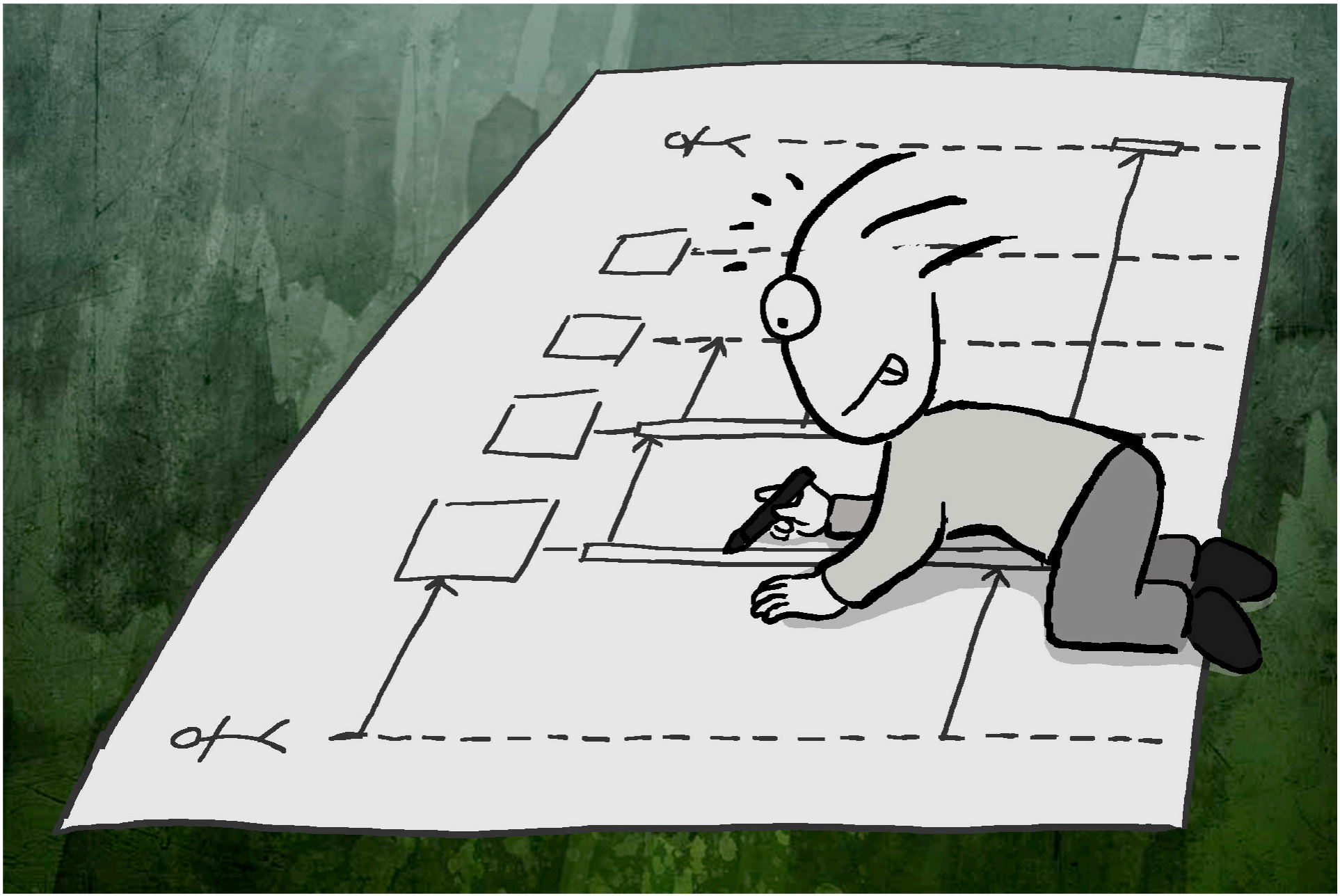


CPEN 321

W4 L1:
UML

Outline

- Logistics
- Processes − recap
- Some UML

Logistics

- Lecture slides are in Piazza, see *Resources* section

- Guest lecture from Microsoft Azure on October 29, 2018

I-Clicker Exercise

A: Talking too fast

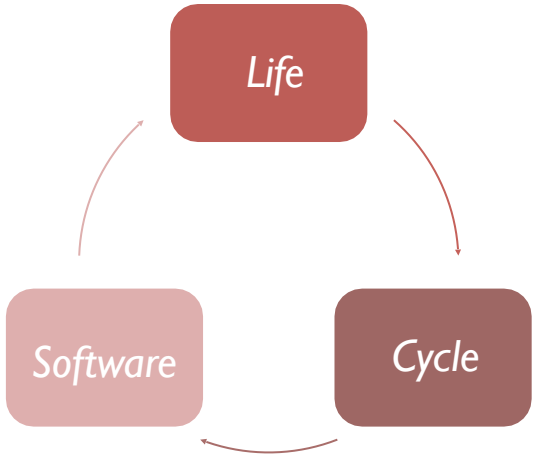
B: Going over the material too fast

C: So far, the pace is fine

Processes – Main Message

- Follow processes, but do not over-emphasize process over product
- Customize the process for your needs!
- No good or bad processes: it depends on the organizational culture, structure, needs

Popular Software Development Process Models



No Plans

- **Code-and-fix**: write code, fix it when it breaks

Plan and Document

- **Waterfall:** perform each phase in order (1970)

Incremental

- **Staged Delivery:** waterfall-like beginnings, then, short release cycle
- **Evolutionary prototyping:** develop a skeleton system and evolve it for delivery
- **Spiral**: triage/figure out riskiest things first (1988)
- **Agile**: *a family of principles* promoting adaptive planning, evolutionary development, early delivery, and continuous improvement (1970-2005+)
  - Most popular: **Scrum** and **Kanban**

Exercise

Your team has to develop **the control software for a car's anti-lock braking system (ABS**). Which process model will you use?

A: Waterfall

B: Staged delivery (waterfall-like beginnings, then, short release cycle)

C: Evolutionary prototyping (evolve a skeleton system until delivery)

D: Spiral (evolve a skeleton system until delivery, address main risks first)

E: Agile (Scrum)

Exercise

Your team has to develop **a hospital accounting system that replaces an existing system**. Which process model will you use?

A: Waterfall

B: Staged delivery (waterfall-like beginnings, then, short release cycle)

C: Evolutionary prototyping (evolve a skeleton system until delivery)

D: Spiral (evolve a skeleton system until delivery, address main risks first)

E: Agile (Scrum)

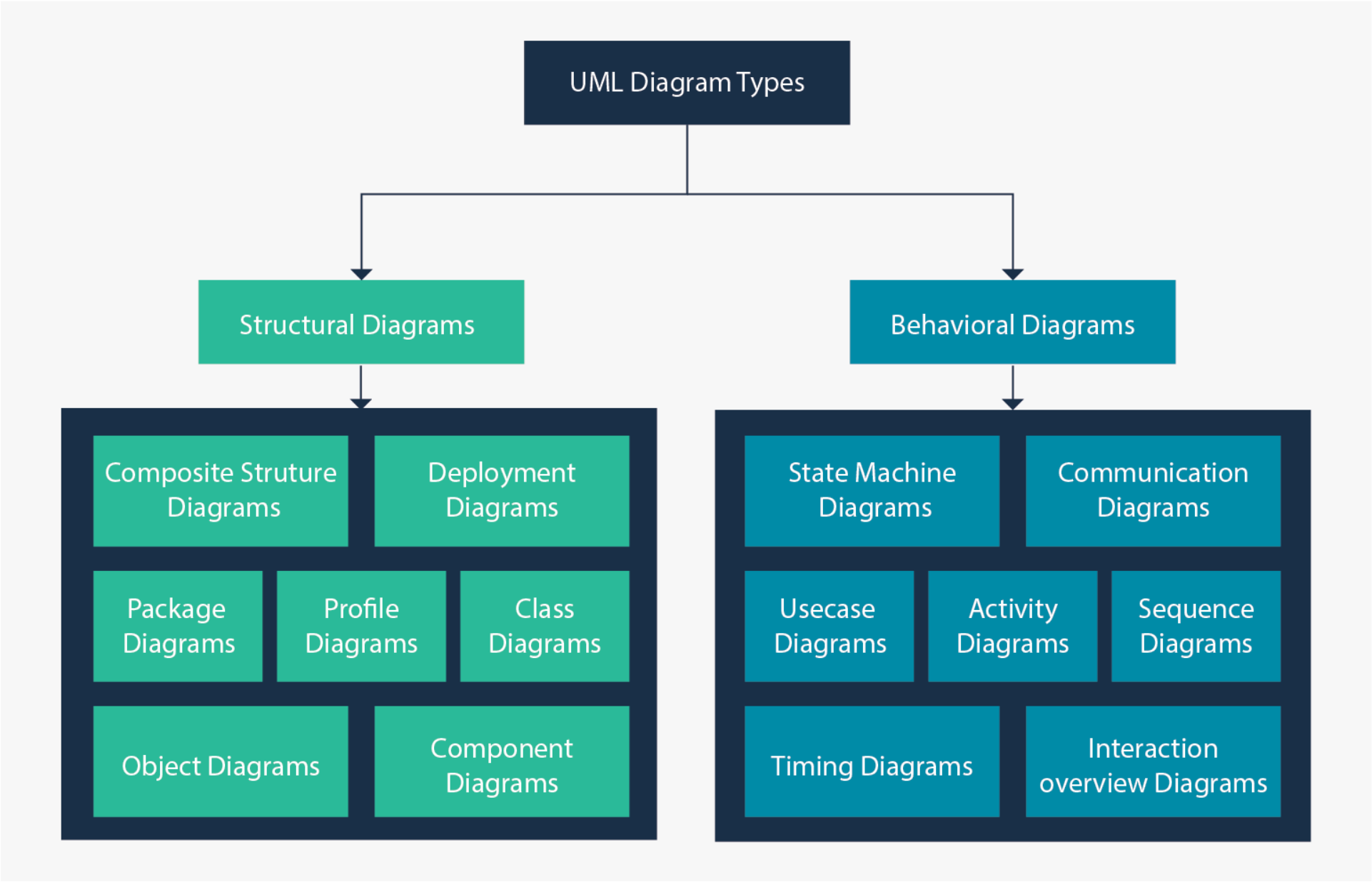
Outline

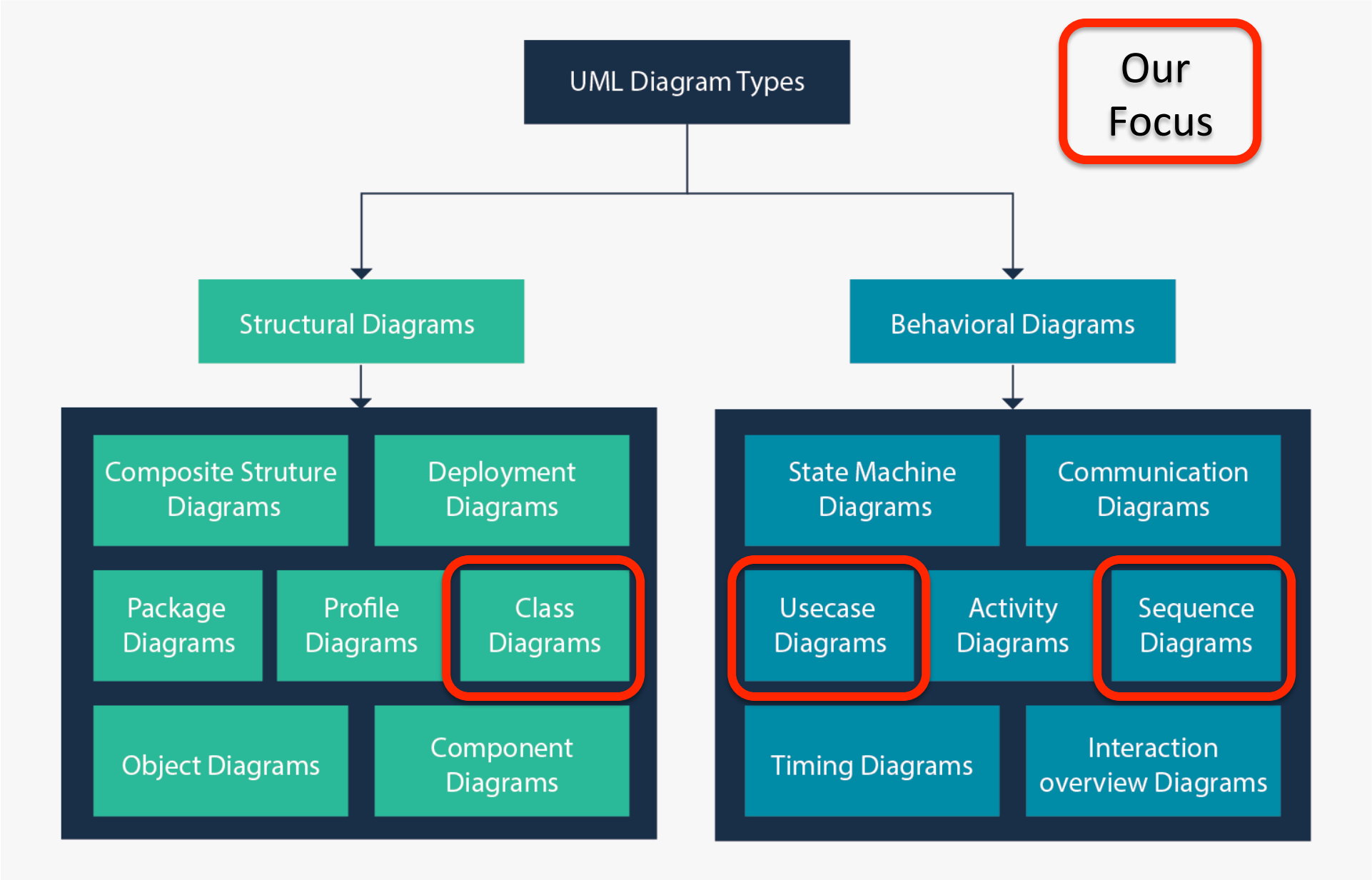
- Logistics
- Processes – recap
- **Some UML**

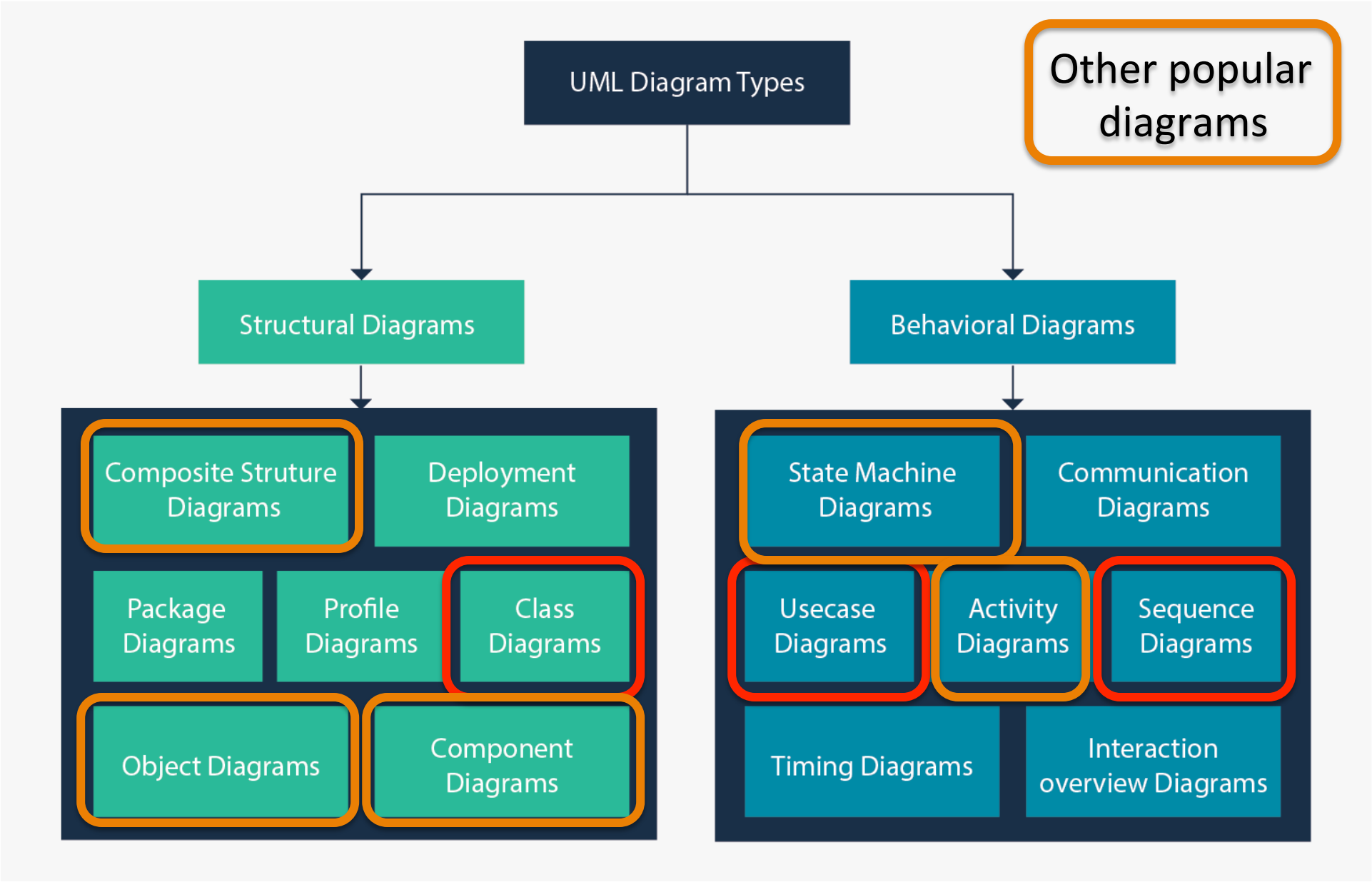
What is UML?

- 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)


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
Our Focus
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
Other popular diagrams
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 Diagrams

- Different UML diagrams are used for capturing different aspects of (structural and behavioral) design
- Used for
  - requirements
  - systems architecture
  - program design
  - etc.

UML Model

- "A model captures a view of a physical system. It is an abstraction of the physical system, with a certain purpose.

- This purpose determines what is to be included in the model and what is irrelevant.

- Thus, the model completely describes those aspects of the physical system that are relevant to the purpose of the model, at the appropriate level of detail."

OMG (www.omg.org)

Outline

- Logistics
- Processes – recap
- **Some UML**
  - **Class diagram**
  - Use case diagram
  - Sequence diagram

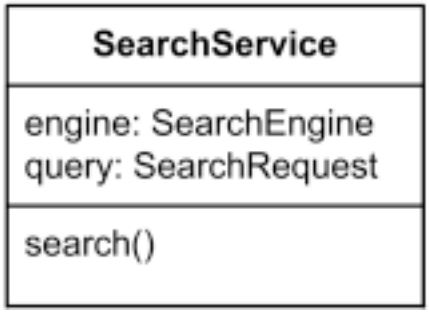
Class Diagram

Shows the classes in a system and the relationships between these classes

Class Diagram – Main Concepts

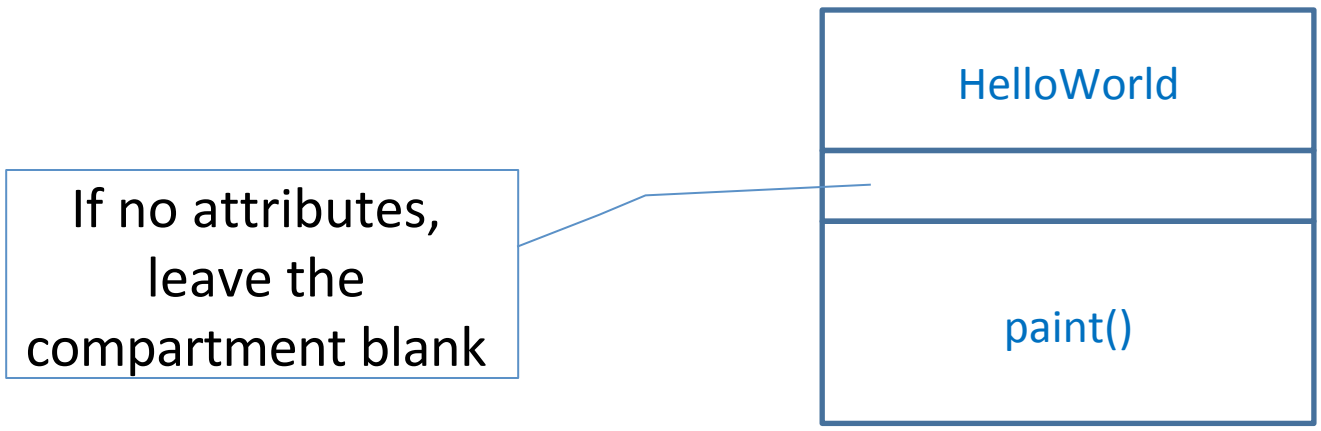
- Class: a rectangle showing the name of the class

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



Class Diagram – Example

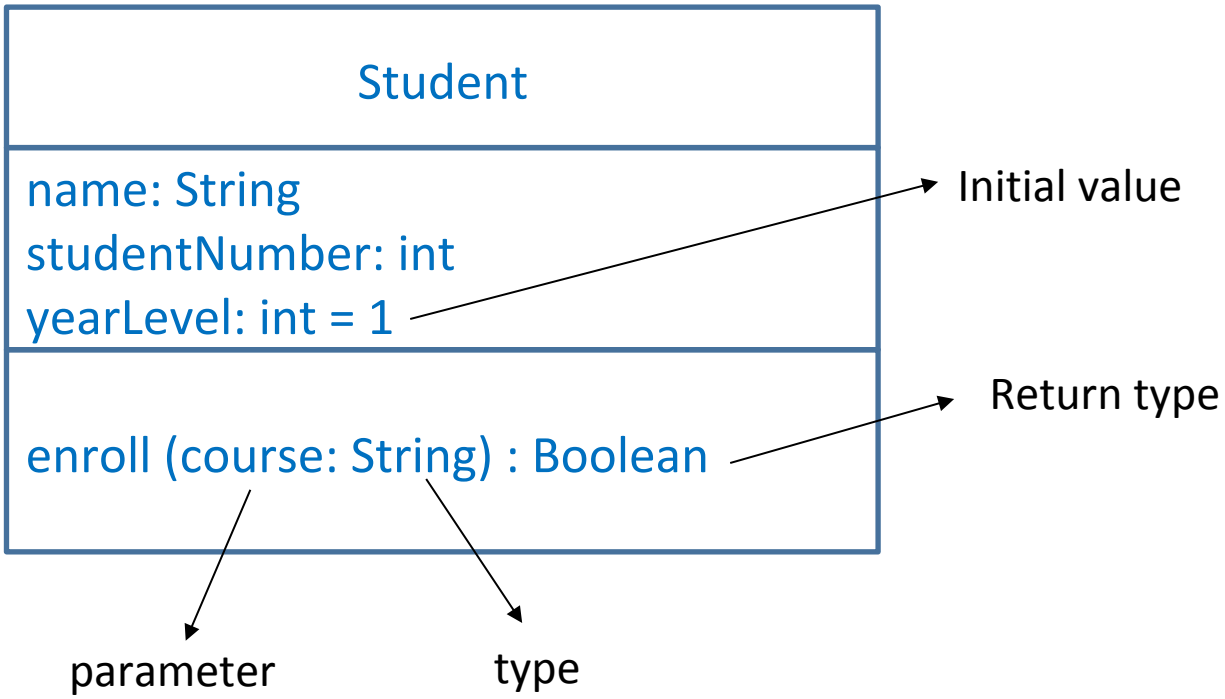
```
import java.awt.Graphics;

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



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

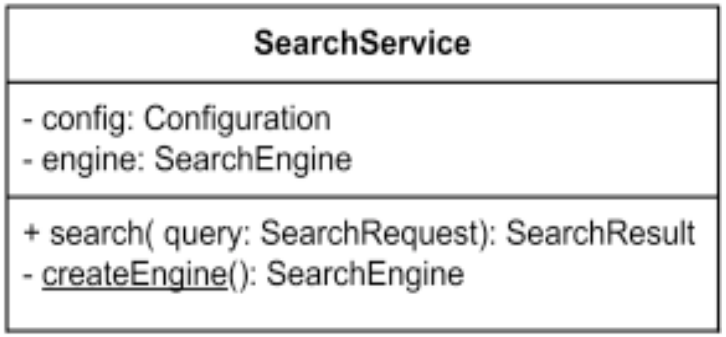
- Example:



Visibility Symbols

- Visibility of attributes/ operations:

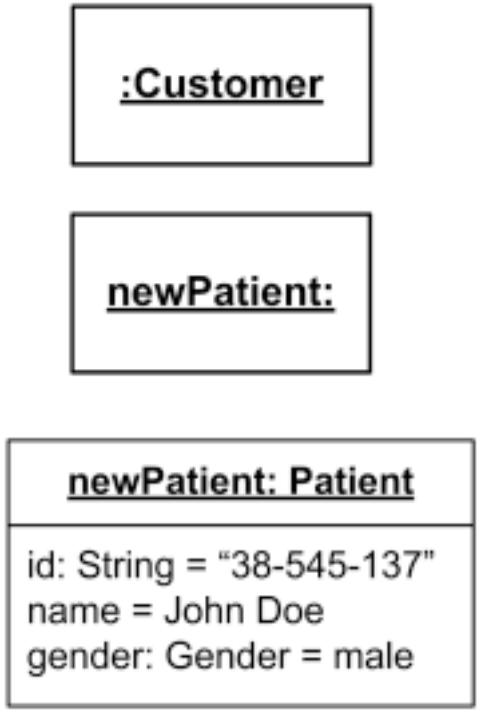
SYMBOL	MEANING	EXPLANATION
+	Public	The member is visible to all code in the application.
–	Private	The member is visible only to code inside the class.
#	Protected	The member is visible only to code inside the class and any derived classes.
~	Package	The member is visible only to code inside the same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



Object

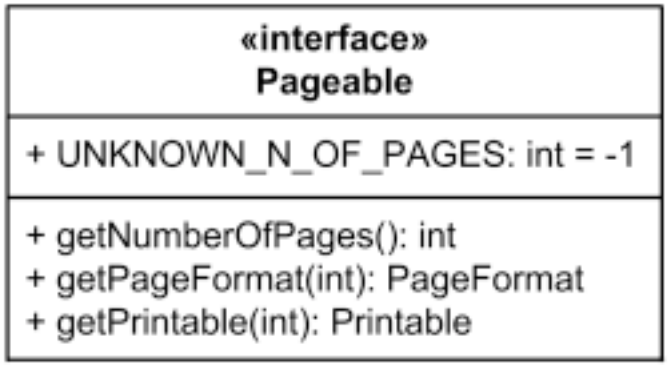
- An instance of a class
- Can optionally contain valuation of fields

- 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

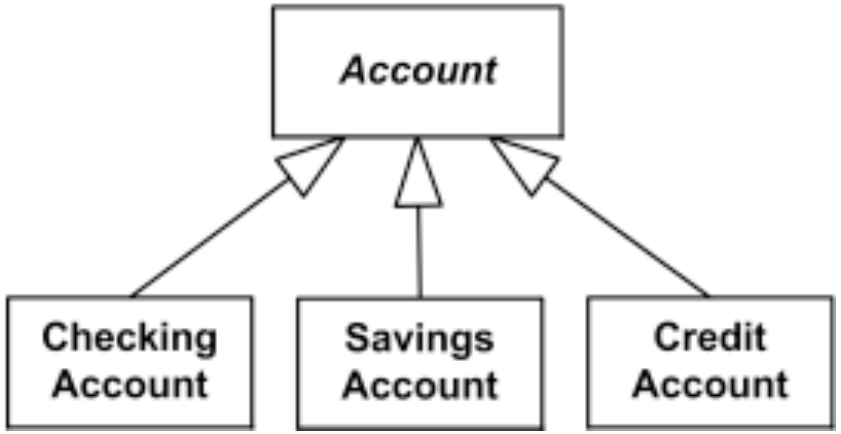


In UML, both Class and Interface are instances of an abstract class called Classifier.

Main Relationships Between Classifiers

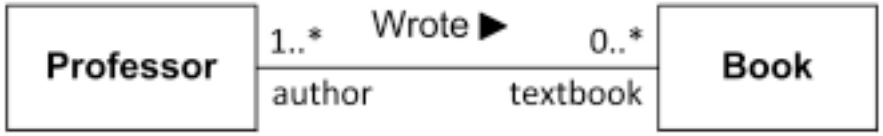
Generalization

- 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

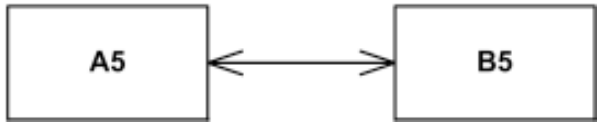
- Describes the presence of a relationship between classes



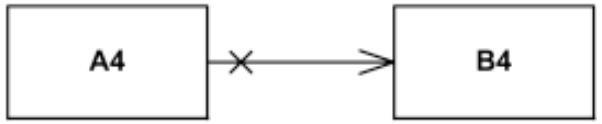
- Name of the association end and multiplicity may be placed near the end of the line
  - The association end name is commonly referred to as **role** (*Professor is an author of a book; A book is used as a textbook by a professor*)
  - Multiplicity − like in use case diagrams (*every Book has at least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

Association – Navigability

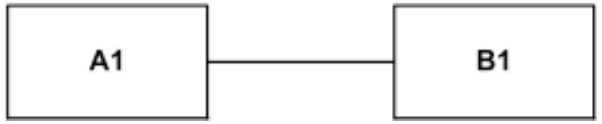
- End property of association is **navigable** from the opposite end if instances of the classes at this end of the link ***can be accessed efficiently at runtime*** from instances at the other ends
- Notations:
  - navigable end: **open arrowhead** (both A5 and B5 here)



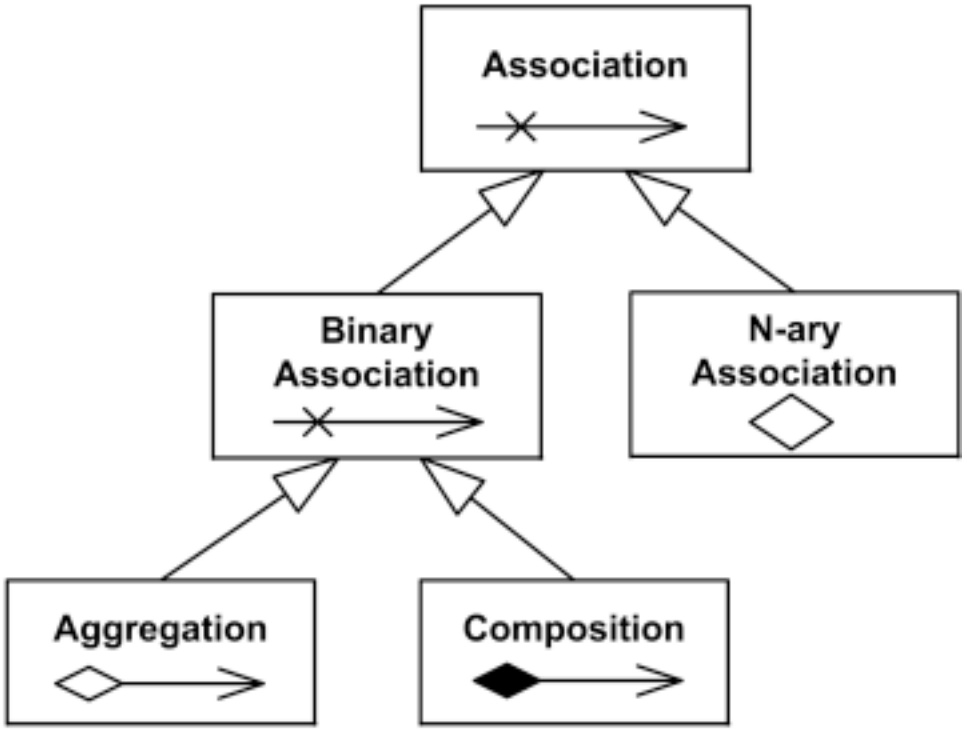
  - not navigable end: a **small x** on the end of an association (A4 cannot be accessed for B4 here)



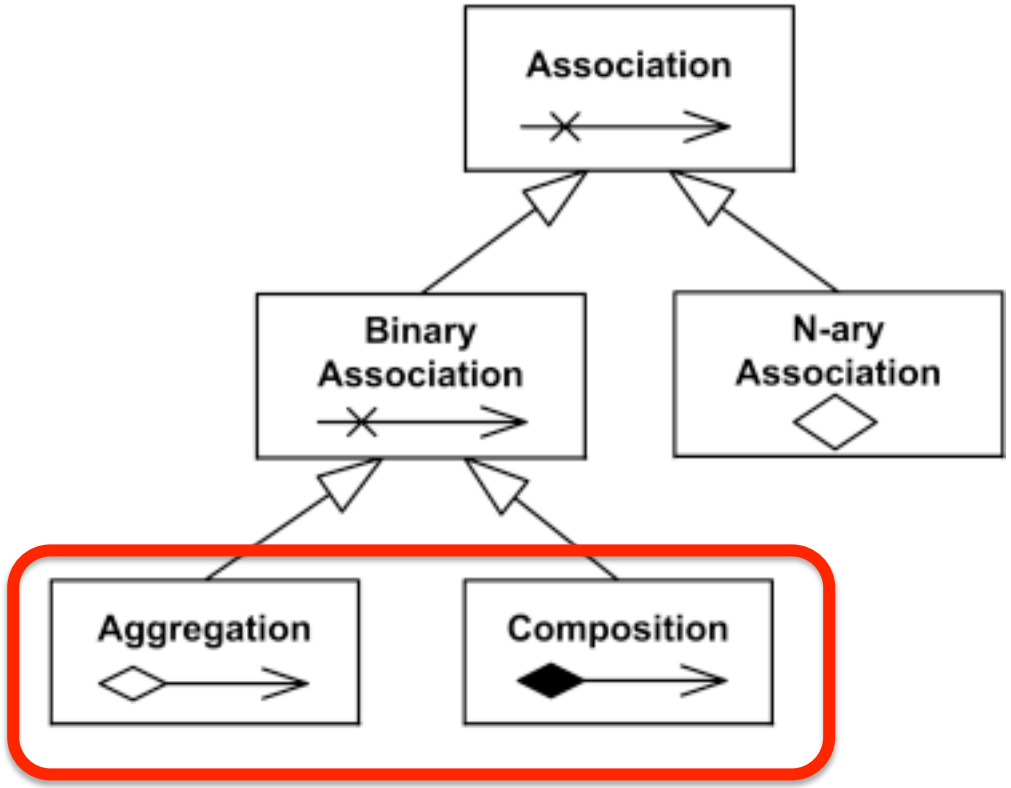
  - no adornment on the end of an association: unspecified navigability (both A1 and B1 here)



Types of Association



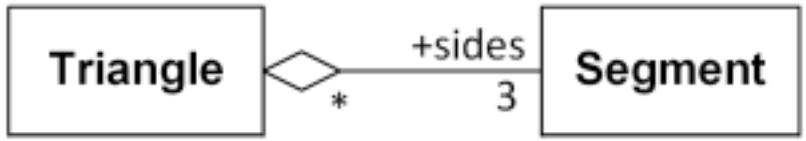
Types of Association



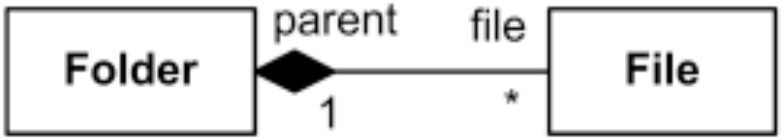
Aggregation and Composition

Whole/part association:

Aggregation:
(a weak form of whole/part)

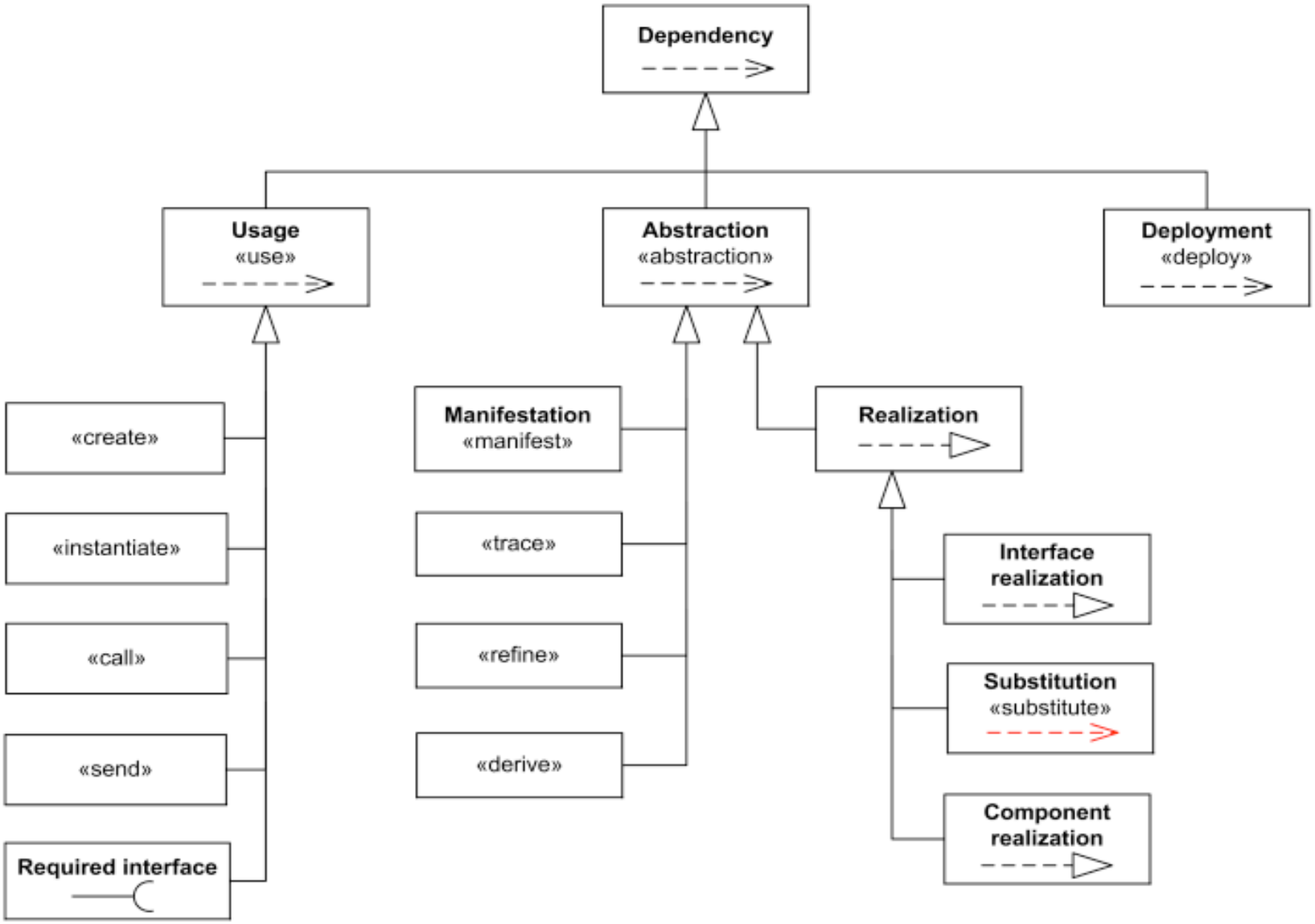


Composition:
(a strong form of whole/part)



- 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



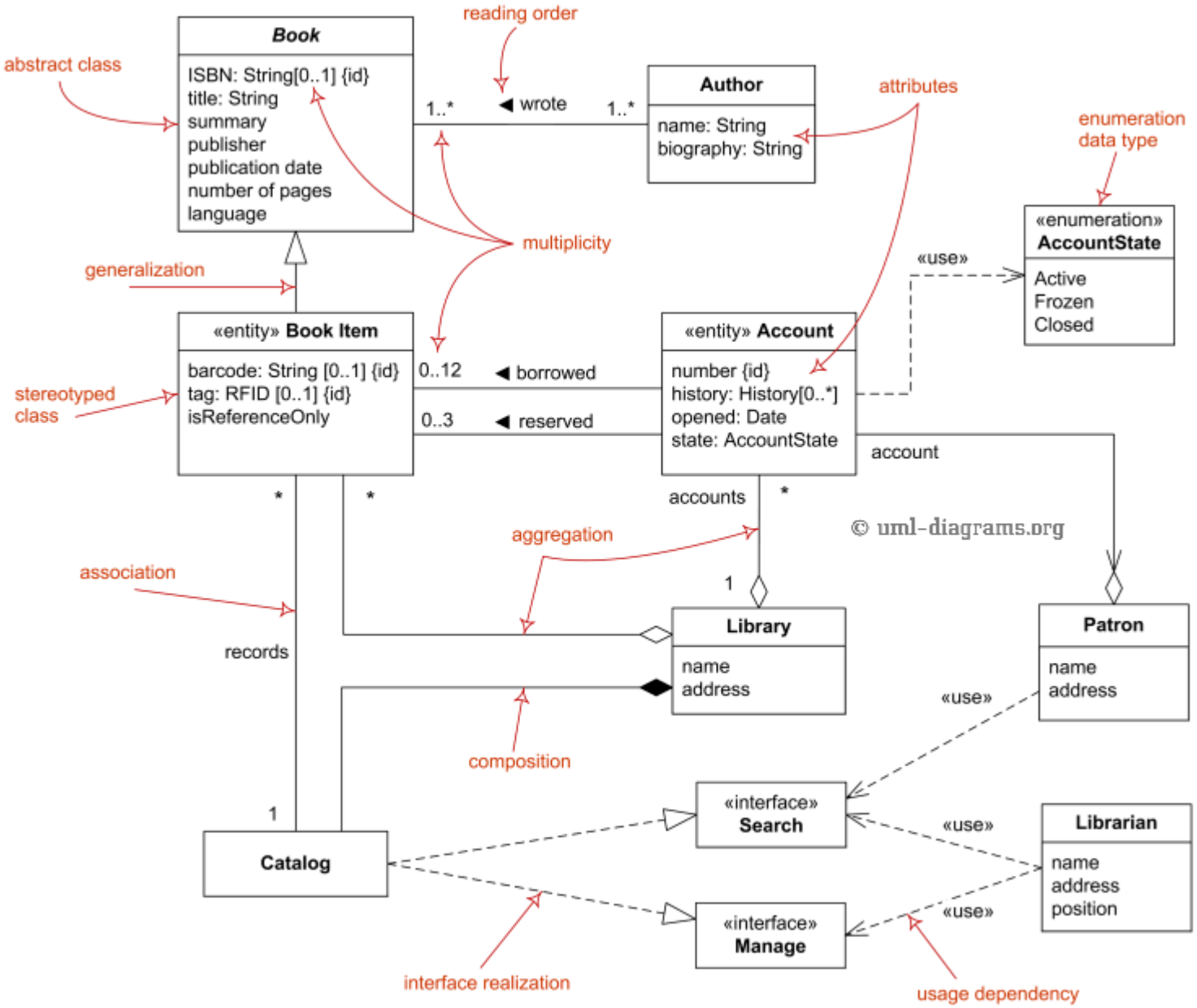


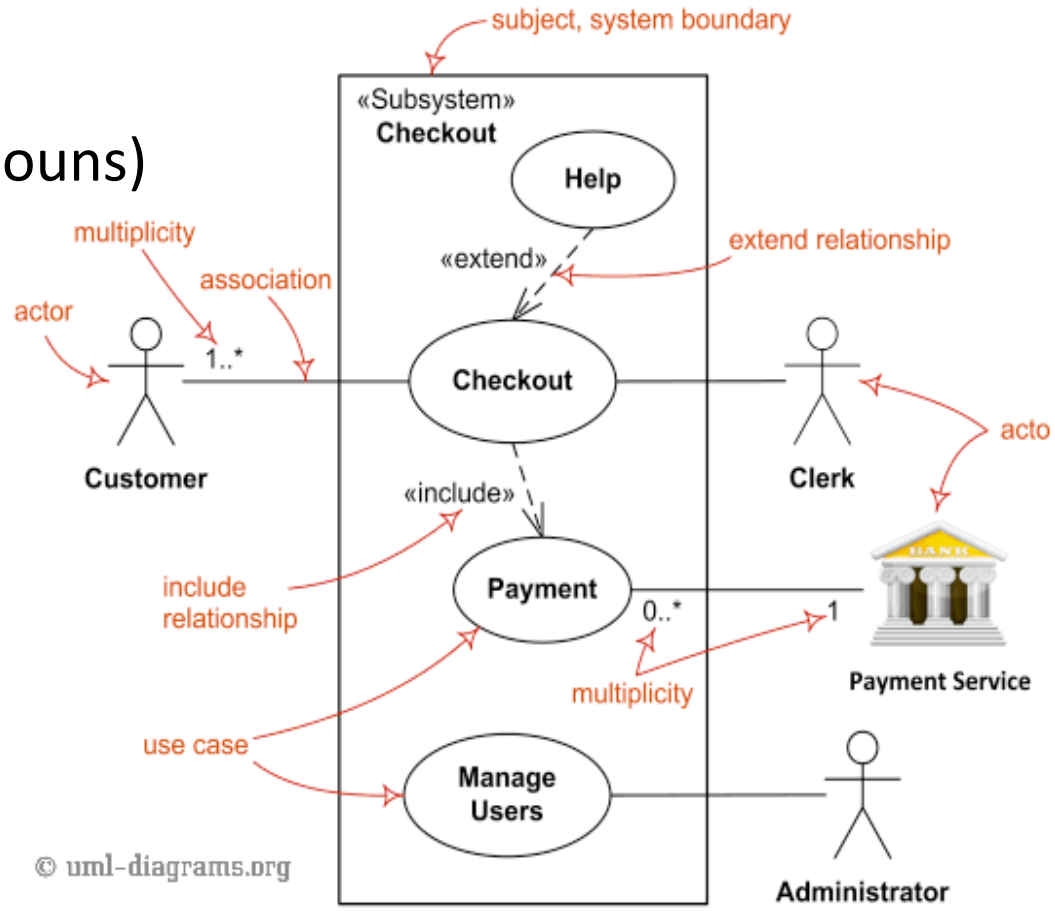
Image source: http://www.uml-diagrams.org/class-diagrams-overview.html

Outline

- Logistics
- Processes – recap
- **Some UML**
  - Class diagram
  - **Use case diagram**
  - Sequence diagram

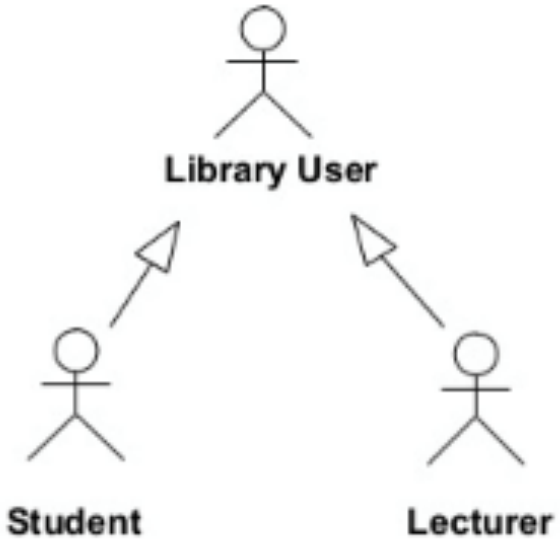
Use Case Diagram – Main Concepts

- 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



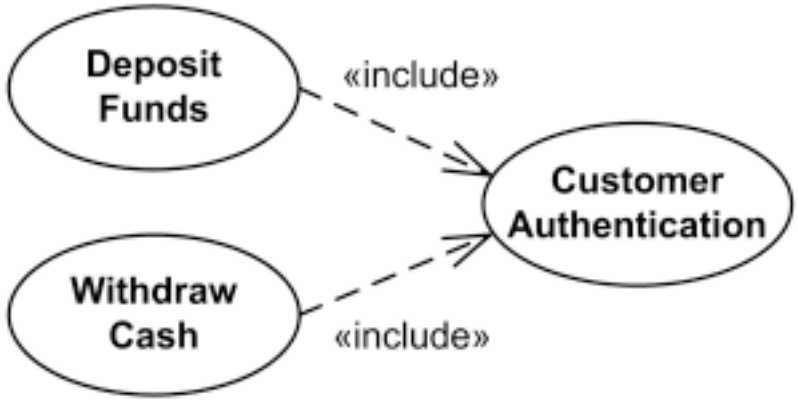
Relationships between actors

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



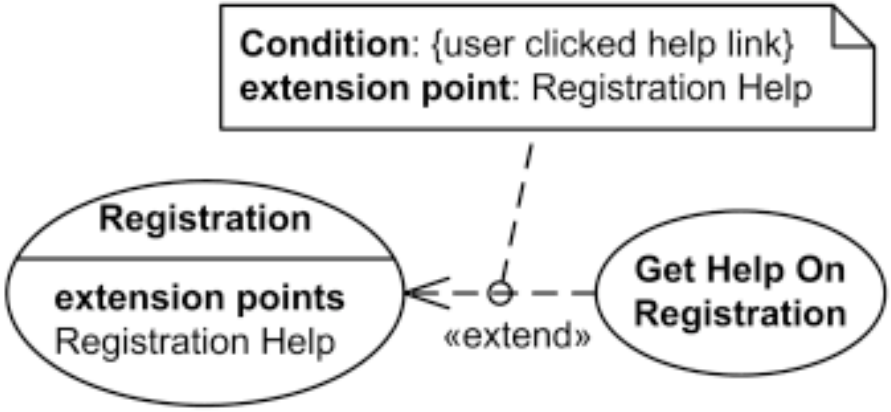
Relationships between use cases

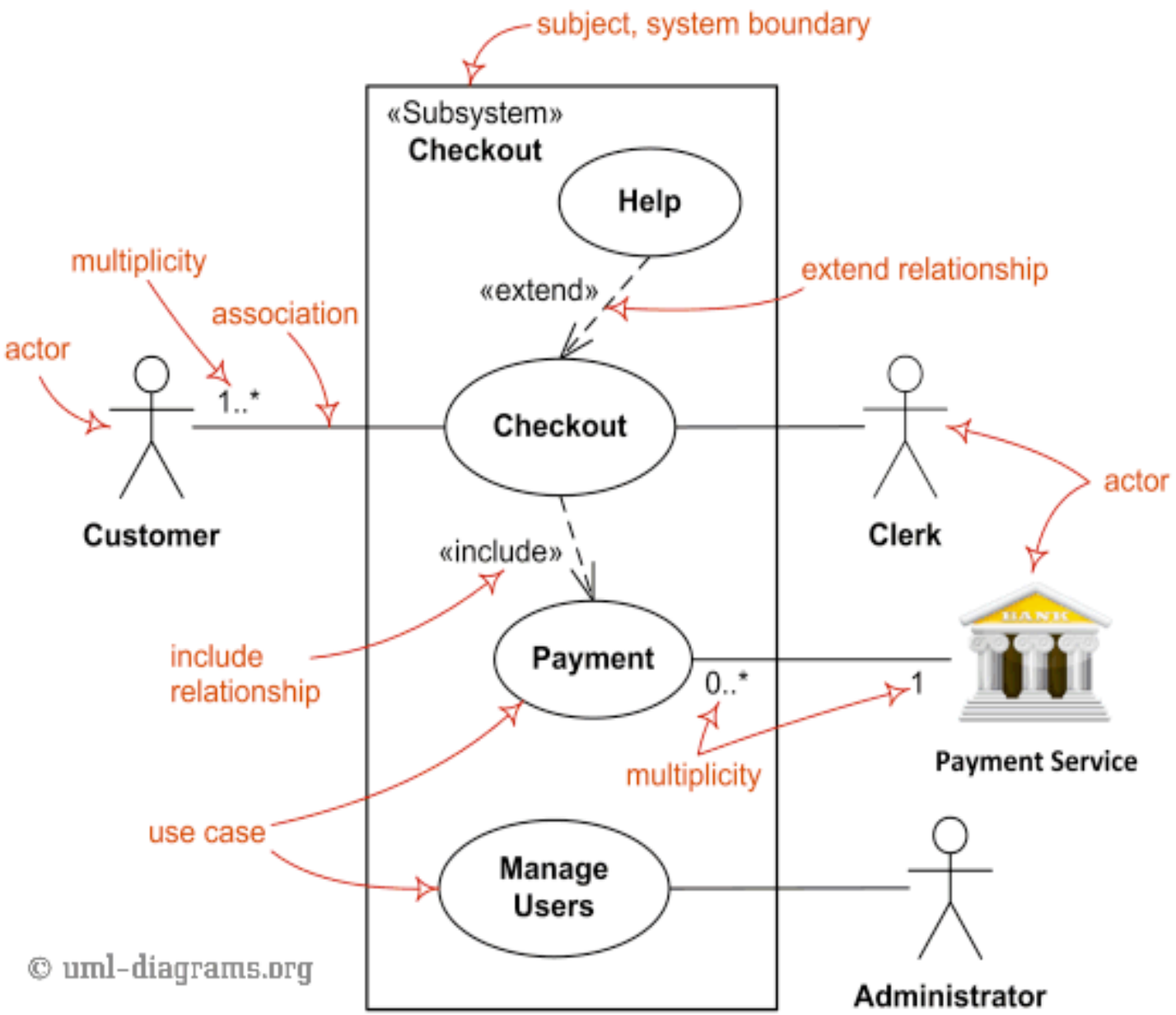
- Include: the behavior of the **included** use case (Customer Authentication) is inserted into the behavior of the **including** use case (Deposit Funds)
  - Including use case cannot be complete without the included one
  - Commonly used to
    - simplify large use case by splitting it into several 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



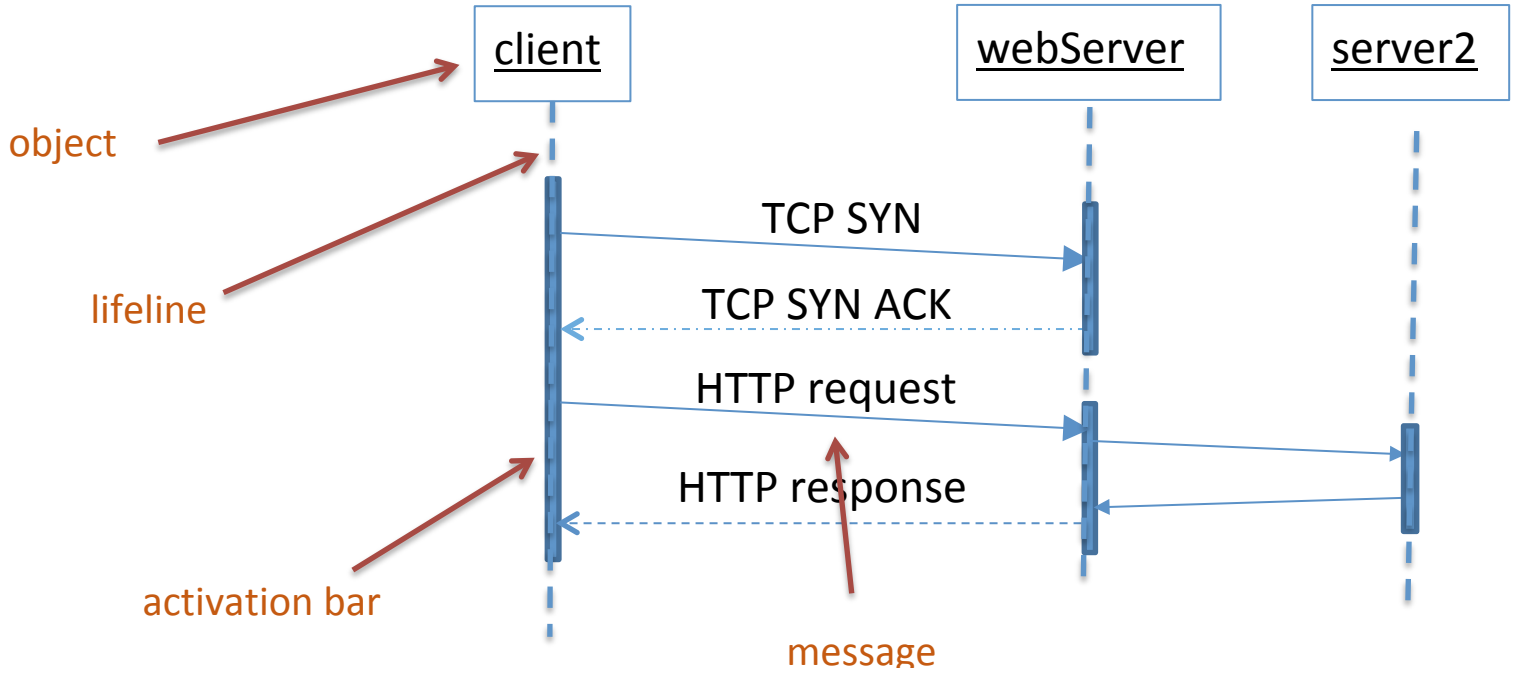

subject, system boundary
«Subsystem»
Checkout
Help
«extend»
extend relationship
multiplicity
association
actor
1..*
Checkout
Customer
Clerk
actor
«include»
include
relationship
Payment
0..*
1
multiplicity
Payment Service
use case
Manage
Users
Administrator
© uml-diagrams.org

Outline

- Logistics
- Processes – recap
- **Some UML**
  - Use case diagram
  - Class diagram
  - **Sequence diagram**

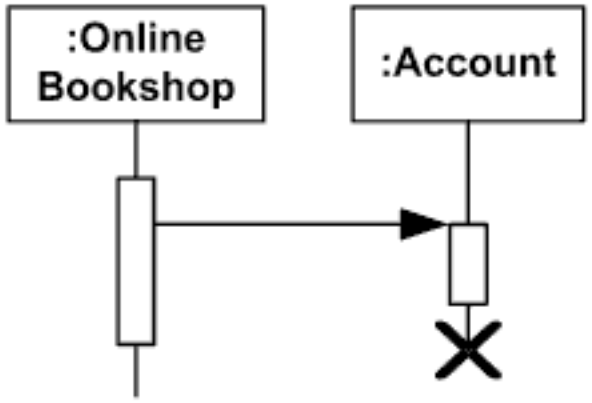
Sequence Diagram

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



Sequence Diagram – Main Concepts

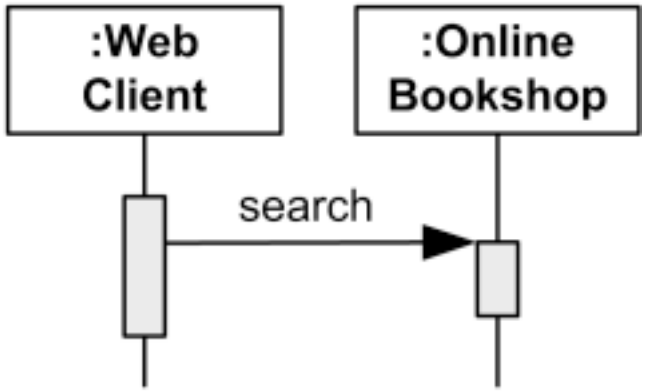
- An interaction: a set of messages exchanged by a set of objects to accomplish a specific purpose
  - A sequence diagram describes an interaction
- A lifeline represents an object involved in the interaction
- A message is represented by an arrow.
  - A call message uses a solid line.
  - An (optional)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



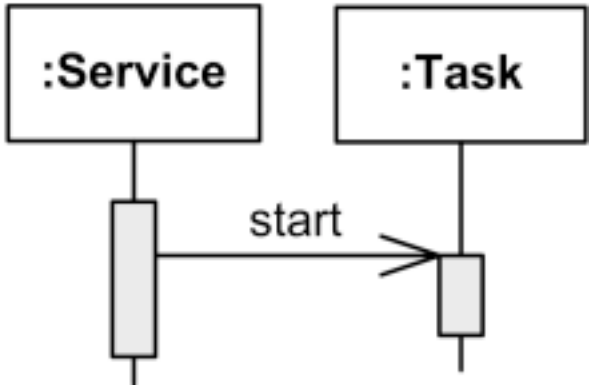
- An execution specification (a.k.a. activation bar) shows when the object is active

Synchronous vs. Asynchronous Messages

- **Synchronous call** represents operation call - send message and suspend execution while waiting for response

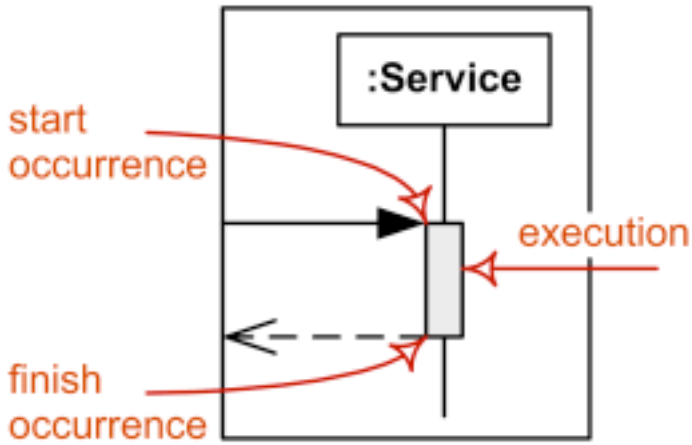


- **Asynchronous call** sends message and proceed immediately without waiting for return value



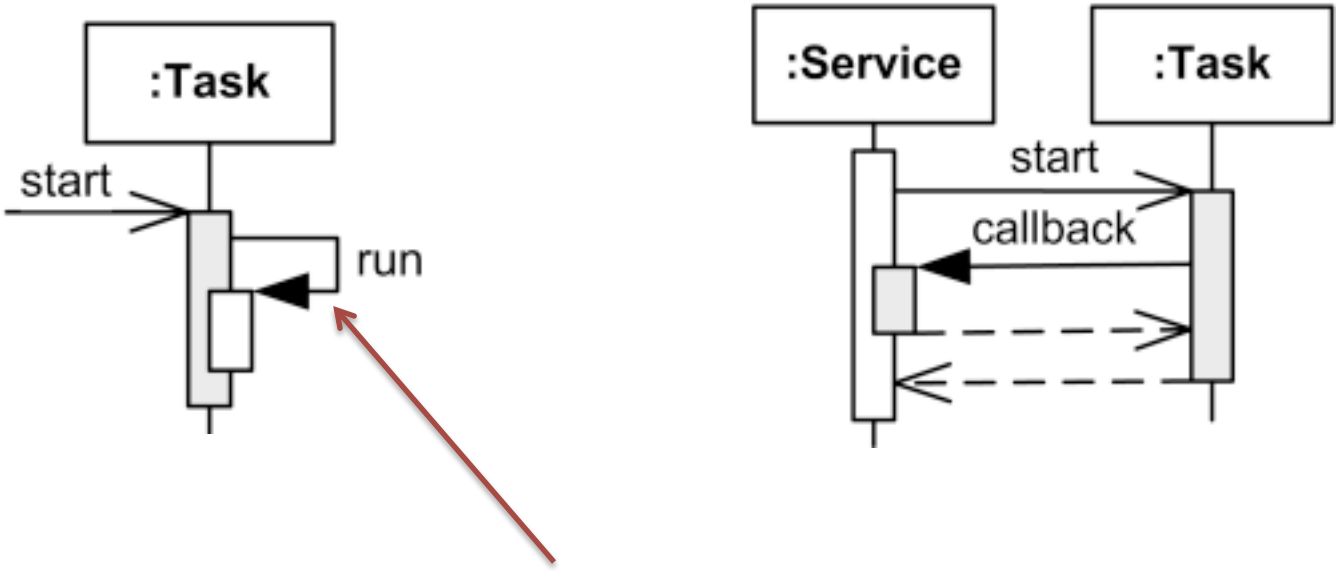
Execution Specification

- 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



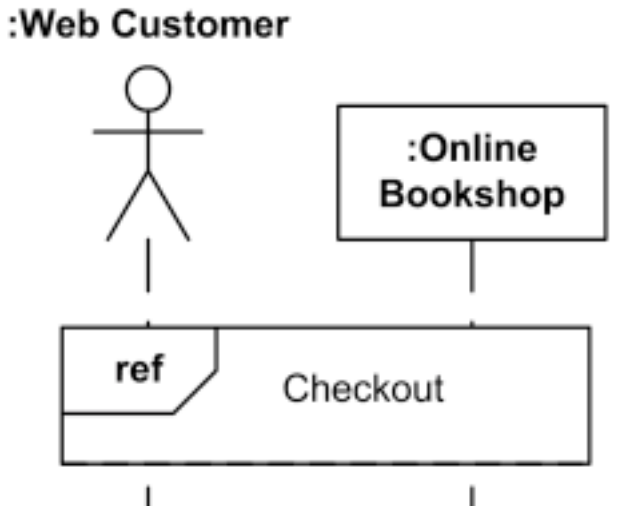
Overlaps

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



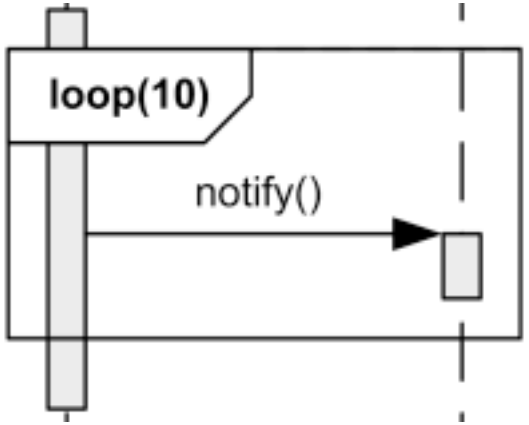
Interaction Fragments

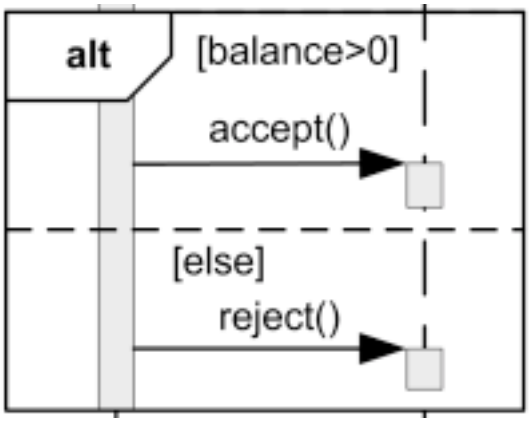
- **Interaction use:** an interaction fragment which allows to call another interaction
- Good for:
  - Simplifying large and complex sequence diagrams
  - Reusing some interaction between several other interactions

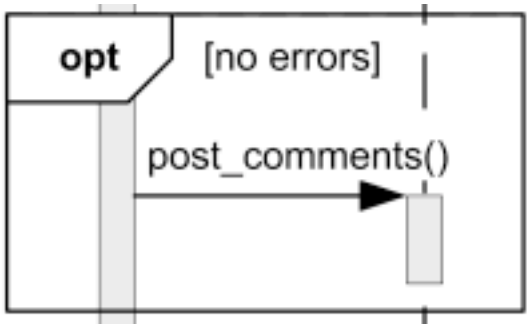


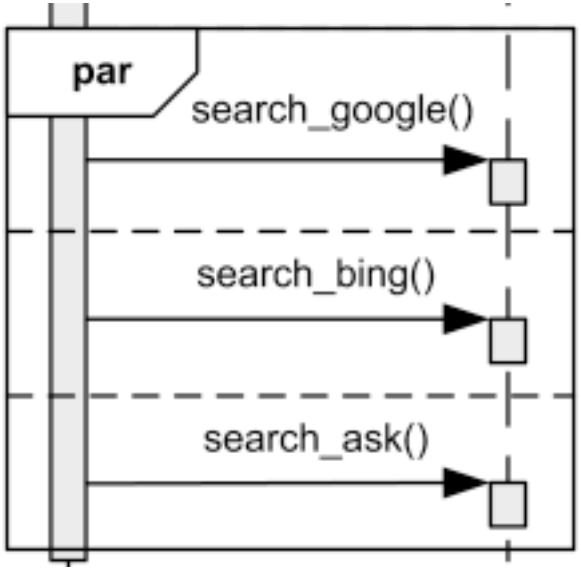
Additional Fragment Operators

- **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



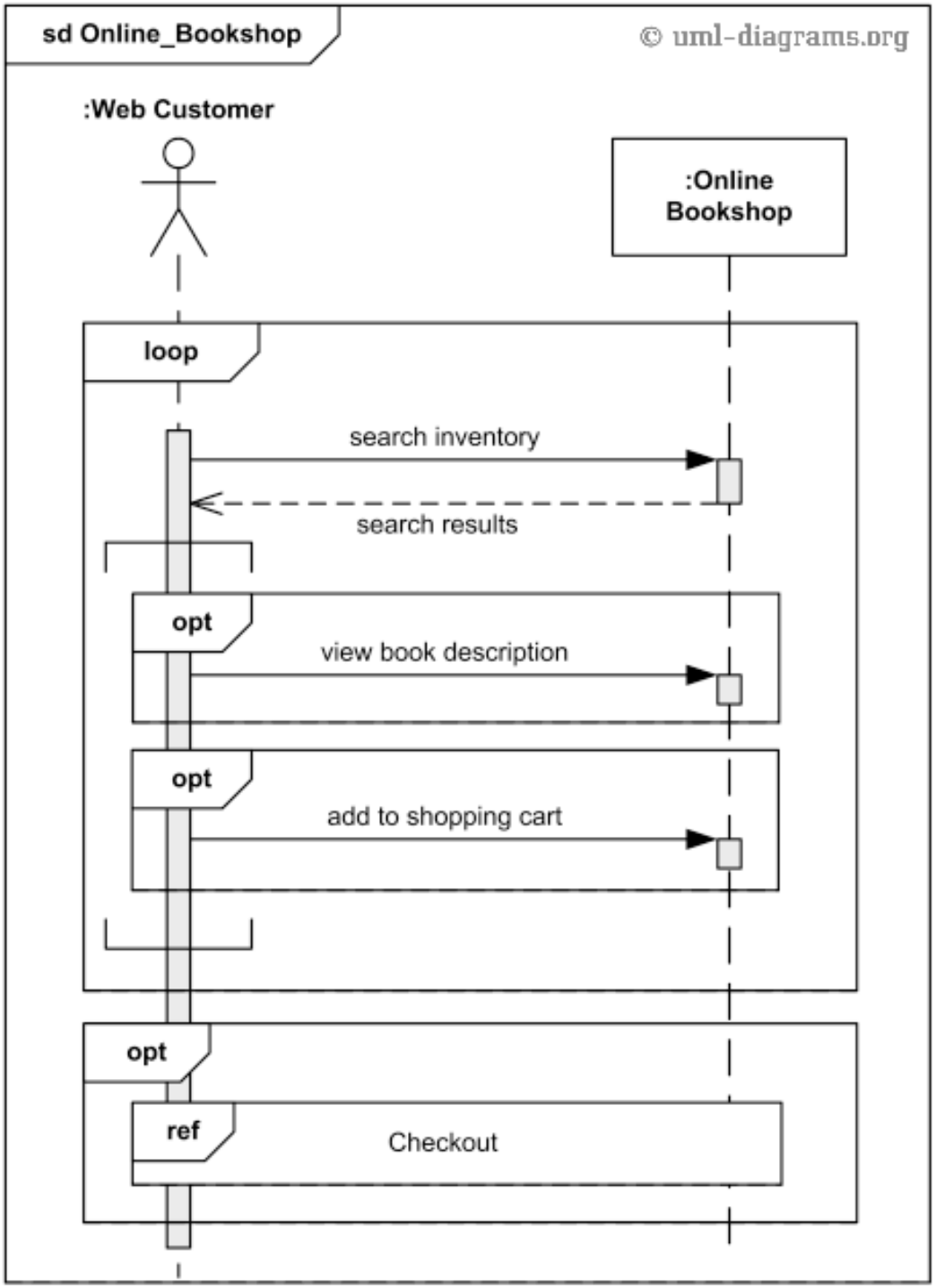






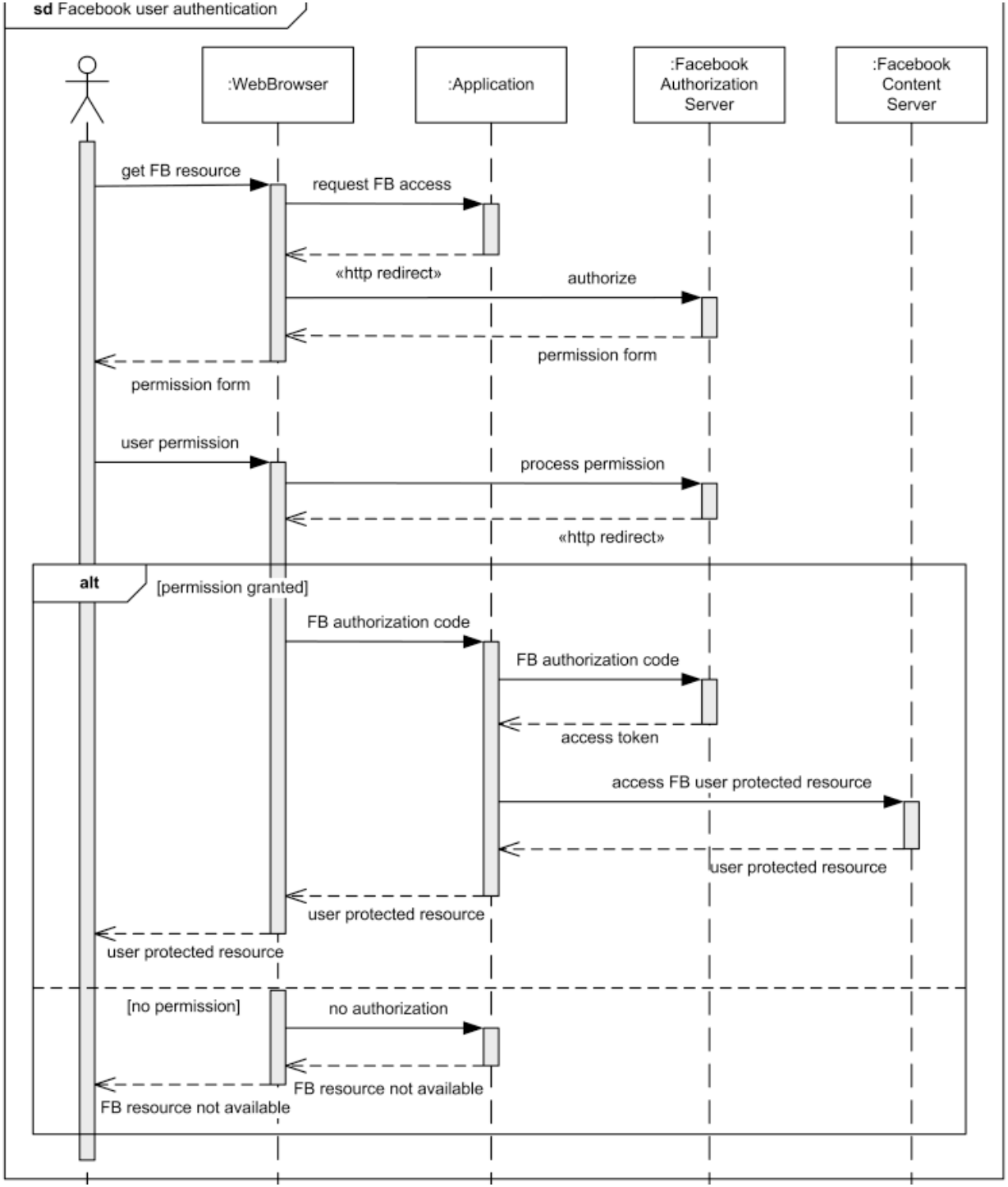
Example

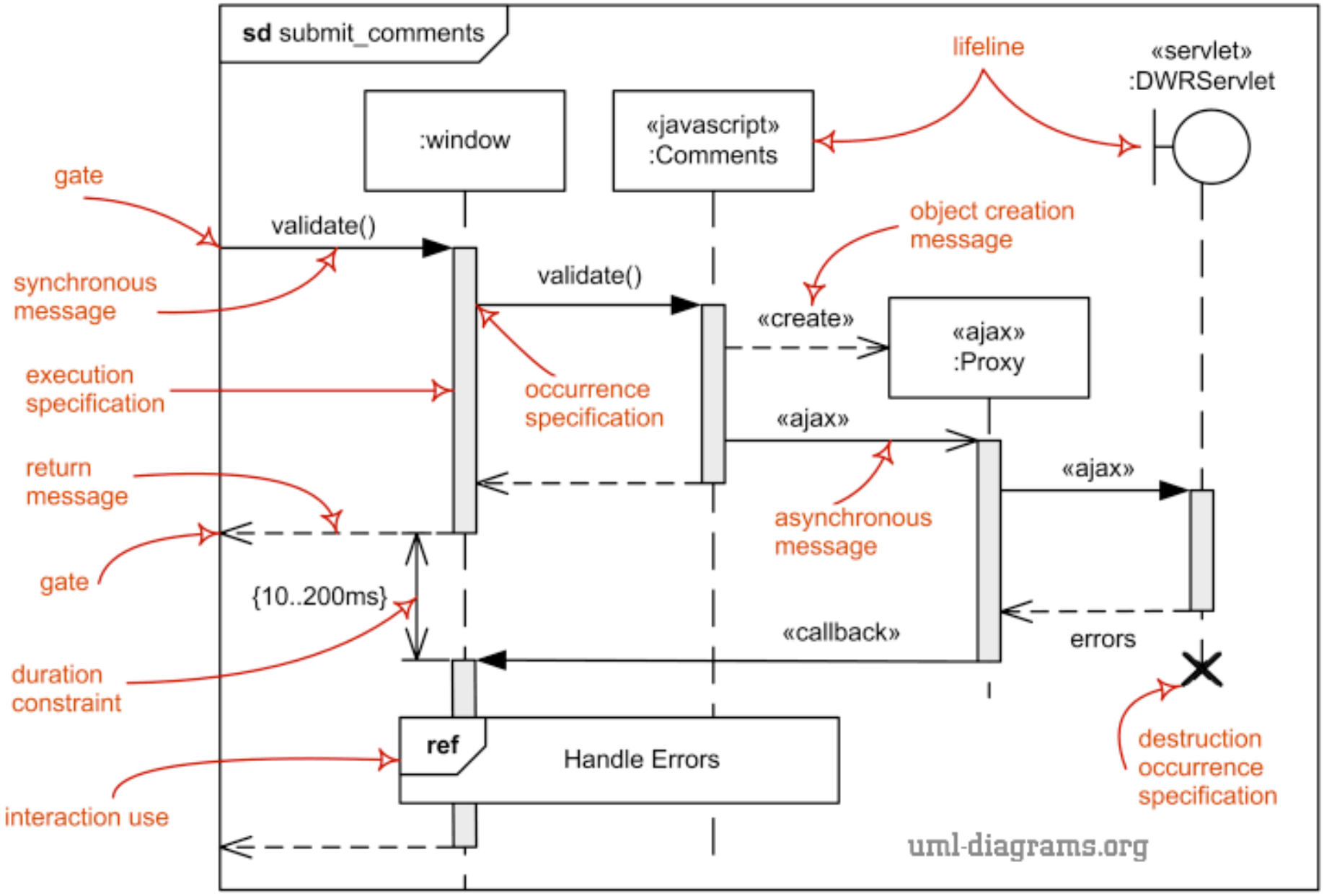
Sequence of interactions between a web customer and an online book shop



Example

Facebook user authentication




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

Assigned Reading for Next Week

- Component diagram: https://www.uml-diagrams.org/component-diagrams.html

- Activity diagram: https://www.uml-diagrams.org/activity-diagrams.html

For More Info

- Online Resources
  - UML standard (http://www.omg.org/spec/UML/2.5/)
  - Wikipedia
  - http://www.sparxsystems.com/resources/uml2_tutorial/
  - UML-diagrams.org

- *UML Weekend Crash Course*, T.A. Pender, Wiley 2002

- …

Summary

- Many diagrams
- One might debate on how often UML is used in practice
  - Answer: Some diagrams are used more widely than others:
    - Simplified class diagrams
    - Activity diagrams (flowcharts)
    - Sequence diagrams
    - 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