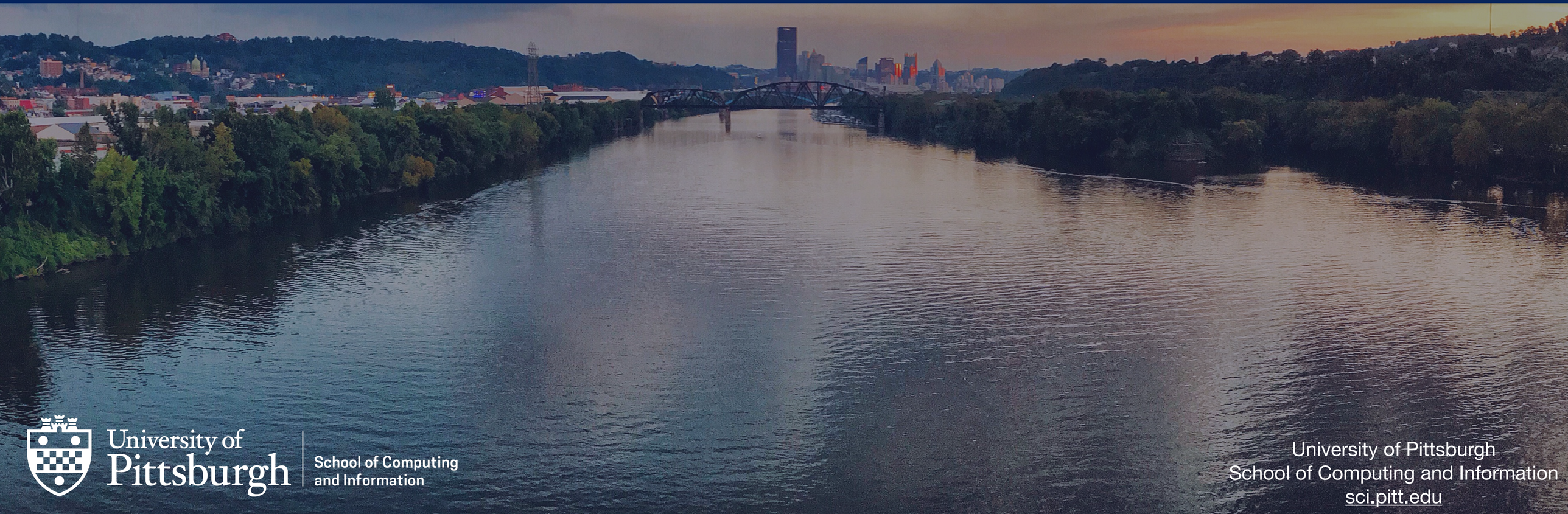
Fall 2025

L11 Requirements Analysis & OOP Recap

CS 1530 Software Engineering

Nadine von Frankenberg



University of Pittsburgh
School of Computing and Information

University of Pittsburgh
School of Computing and Information
sci.pitt.edu

Copyright

- These slides are intended for use by students in CS 1530 at the University of Pittsburgh only and no one else. They are offered free of charge and must not be sold or shared in any manner. Distribution to individuals other than registered students is strictly prohibited, as is their publication on the internet.
  - All materials presented in this course are protected by copyright and have been duplicated solely for the educational purposes of the university in accordance with the granted license. Selling, modifying, reproducing, or sharing any portion of this material with others is prohibited. If you receive these materials in electronic format, you are permitted to print them solely for personal study and research purposes.
  - Please be aware that failure to adhere to these guidelines could result in legal action for copyright infringement and/or disciplinary measures imposed by the university. Your compliance is greatly appreciated.
- Material from these notes is obtained from various sources, including, but not limited to, the following:
  - Bruegge, & Dutoit. Object–oriented software engineering. using UML, patterns, and Java. Pearson, 2009.
  - Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns. Pearson, 1994.
  - Sommerville, Ian. "Software Engineering" Pearson. 2011.
  - http://scrum.org/

Learning goals

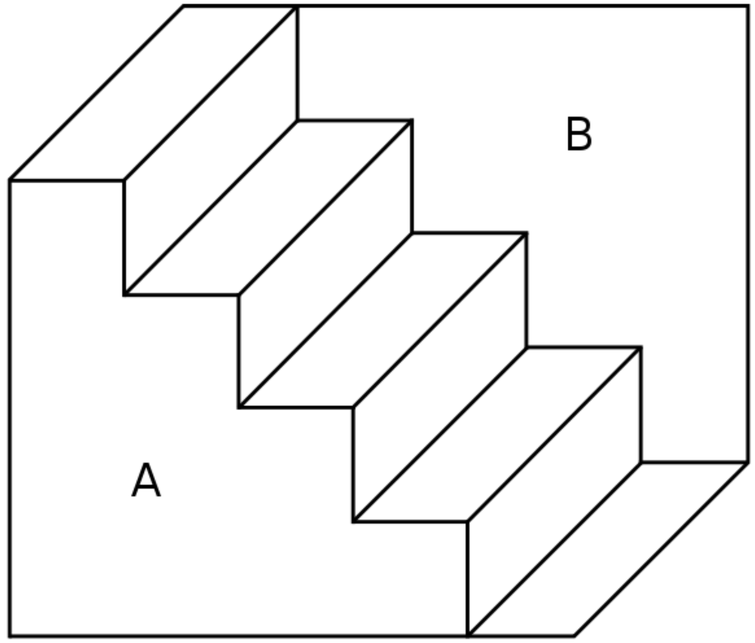
- You understand the purpose of an analysis object model
- You can derive entity, boundary, and control objects from requirements / a problem statement

Today's roadmap

- Intro to Analysis
  - Recap: OOP
- Analysis object model

Requirements specification?

William Ely Hill, 1915, My Wife and My Mother-in-Law



Schröder's stairs, 1858

If these multi-stable images were a requirements specification, which model would you have constructed?
→ Specifications contain **ambiguities**

Status in the SDLC

- Requirements: What the system must do (functional + non-functional)
- **Analysis**: Build models to understand and organize those requirements in the problem domain
- Design: Translate analysis models into solution models (classes, architecture, code)

Analysis — First step towards system architecture

University of Pittsburgh

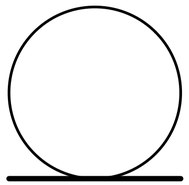
- Refine and validate requirements
  - Resolve ambiguities discovered during requirements stage
- Identify analysis objects
  - From problem domain (e.g., Sinkhole, Report, Resident, City Official)
- Define relationships
  - Associations, aggregations, generalizations among analysis objects
- Model system behavior
  - Scenarios → sequence diagrams, state machines, activity diagrams
- Partition the problem
  - Which responsibilities belong to which objects or subsystems?

Today’s roadmap

- Intro to Analysis
  - Recap: OOP
- Analysis object model

Object types

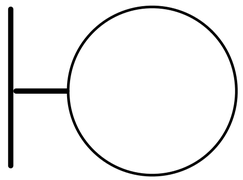
persistent information tracked by the system



Entity

Sinkhole
(has location, status, reportedDate)

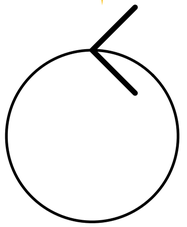
interaction between user and system



Boundary

SinkholeReporter
(e.g., form; validates user input)

control tasks performed by the system



Control

SinkholeReportController
(creates a new Sinkhole, saves it)

Problem Statement

The City of Pittsburgh requires the development of a Sinkhole Monitoring System (SiMCity) to address the issue of sinkholes within the city. The first version of SiMCity should enable residents and city officials to report sinkhole occurrences and track their locations.

The system should be easy to use, accessible through web and mobile platforms, and track sinkholes in real-time.

Residents should be able to view sinkholes on a map, helping them to avoid such areas. It should also allow residents to report new sinkholes by sending in their location and photographic evidence for verification, or call for help. City officials can remove sinkholes from the map after they have been re-filled and repaired.

SiMCity aims to increase public safety, facilitate rapid response to sinkholes, and improve the city's ability to manage and mitigate the impact of these geohazards.

Hints at an entity

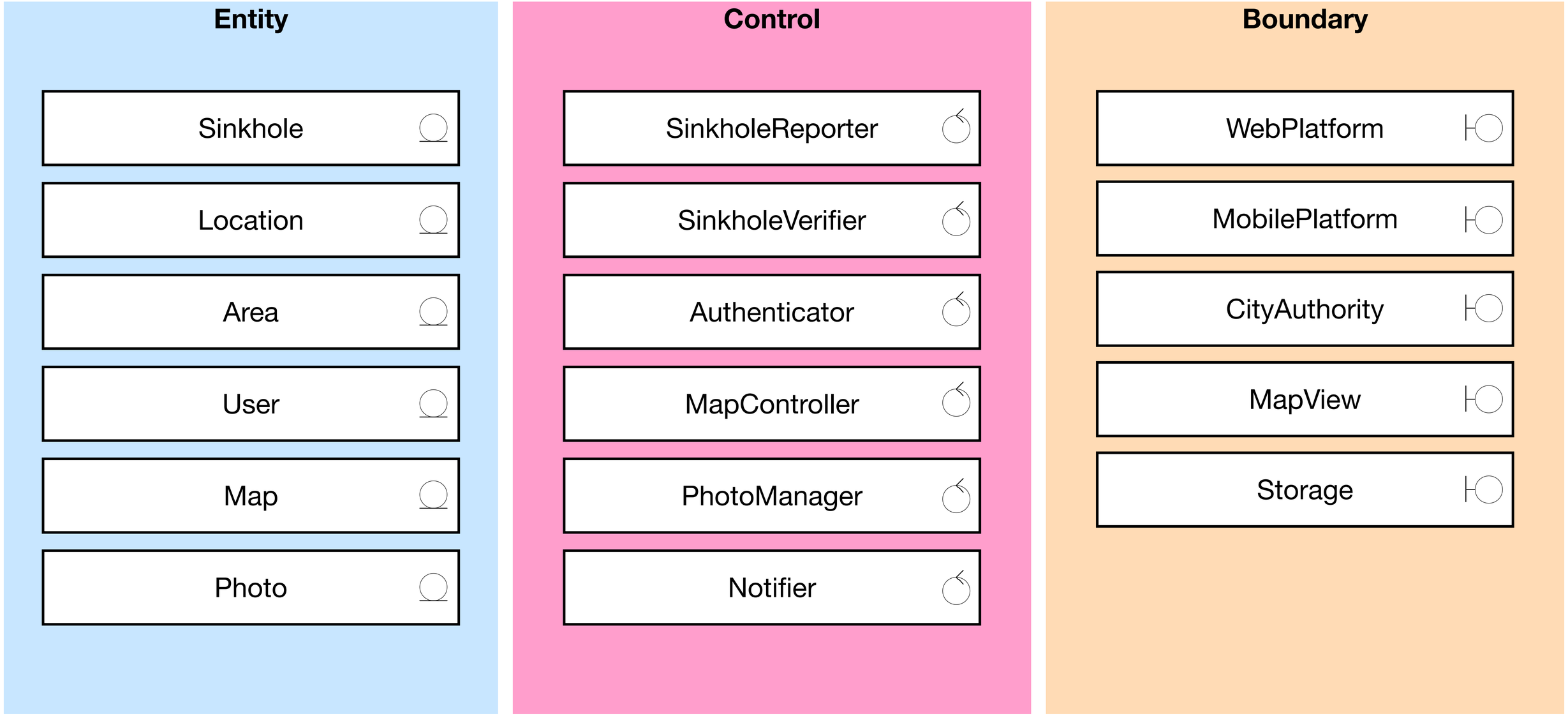
Hints at a control

Hints at boundary

[Example] Abbott's technique

Example	Grammatical construct	UML element
sinkhole	Proper noun	Object
resident	Improper noun	Class
track	Action verb	Operation
is a	Being verb	Inheritance
has a	Having (verb)	Aggregation
must have a	Modal verb	Constraint
fun	Adjective	Attribute
report	Transitive verb	Operation
depends on	Intransitive verb	Constraint, class, association

[Example] Entity, control, and boundary objects

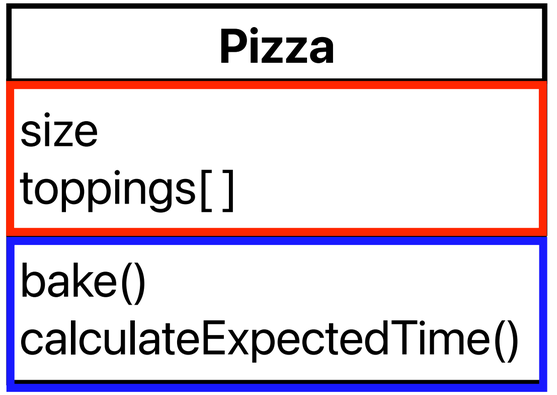


Encapsulation promotes security & modularity

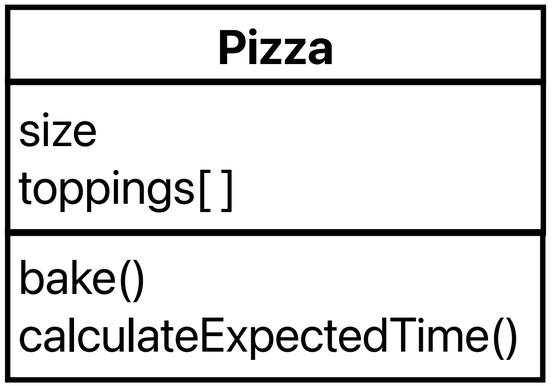
- **Purpose:** Hide internal details of an object and only expose the necessary functionality through public methods
- New objects can be identified based on the system's requirements
- Encapsulation means creating classes for such objects to define the characteristics by defining attributes and behavior by providing methods
- Java supports encapsulation by using classes with attributes for structuring and methods for describing functionality

Problem Statement:

"Pizza lovers can bake pizzas with varying sizes and toppings and check the expected time until the pizza is ready."

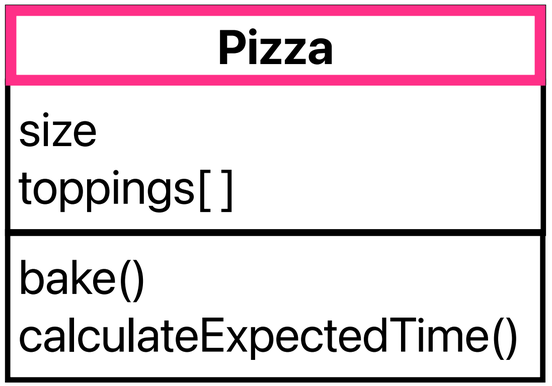


From model to code



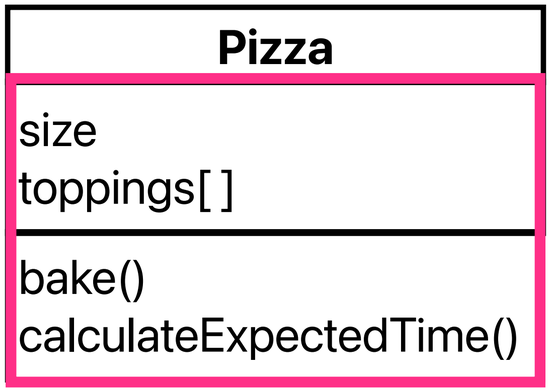

```
public class Pizza {

}
```

From model to code



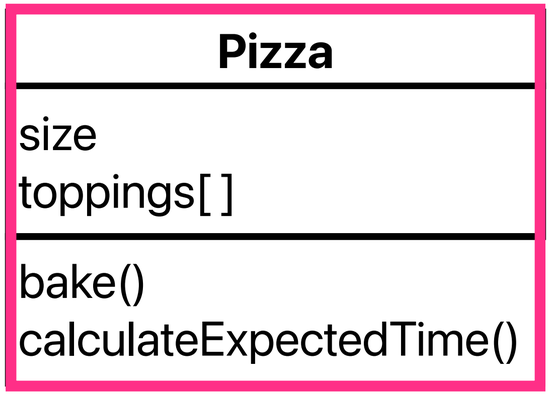

```
public class Pizza {

}
```

From model to code



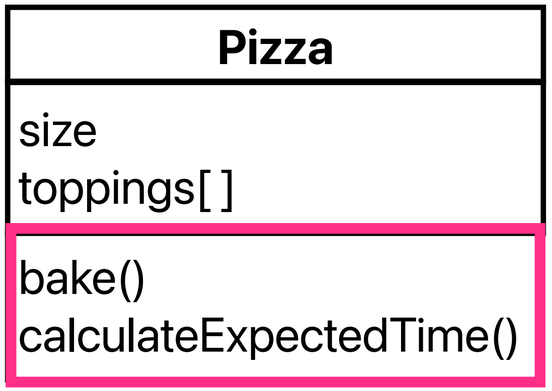

```
public class Pizza {
    private String size;
    private String[] toppings;




    public void bake(){

    }

    public int calculateExpectedTime( ) {




    }
}
```

From model to code




```java
public class Pizza {
    private String size;
    private String[] toppings;

    public Pizza(String size, String[] toppings) {
        this.size = size;
        this.toppings = toppings;
    }

    public void bake(){

    }

    public int calculateExpectedTime( ) {



    }
}
```

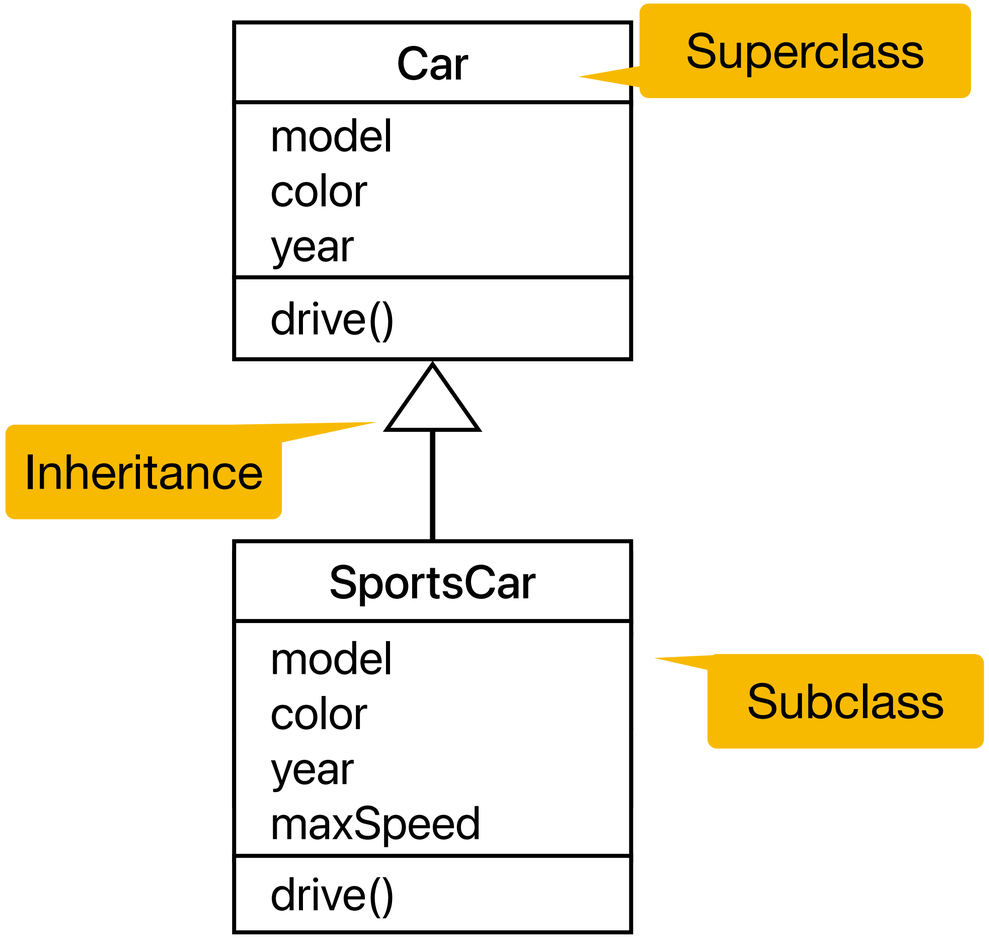
From model to code




```java
public class Pizza {
    private String size;
    private String[] toppings;

    public Pizza(String size, String[] toppings) {
        this.size = size;
        this.toppings = toppings;
    }

    public void bake(){
      System.out.print("🍕🔥");
    }

    public int calculateExpectedTime() {
      int expectedTime = 10; // 10 minutes for baking the crust

      if (size.equals("large")) { expectedTime += 5; }

      return expectedTime;
    }
}
```

Inheritance creates common structure & behavior

- **Purpose:** Create new classes by inheriting attributes & behaviors from existing classes
- Establishes an inheritance hierarchy (also called a "**taxonomy**")

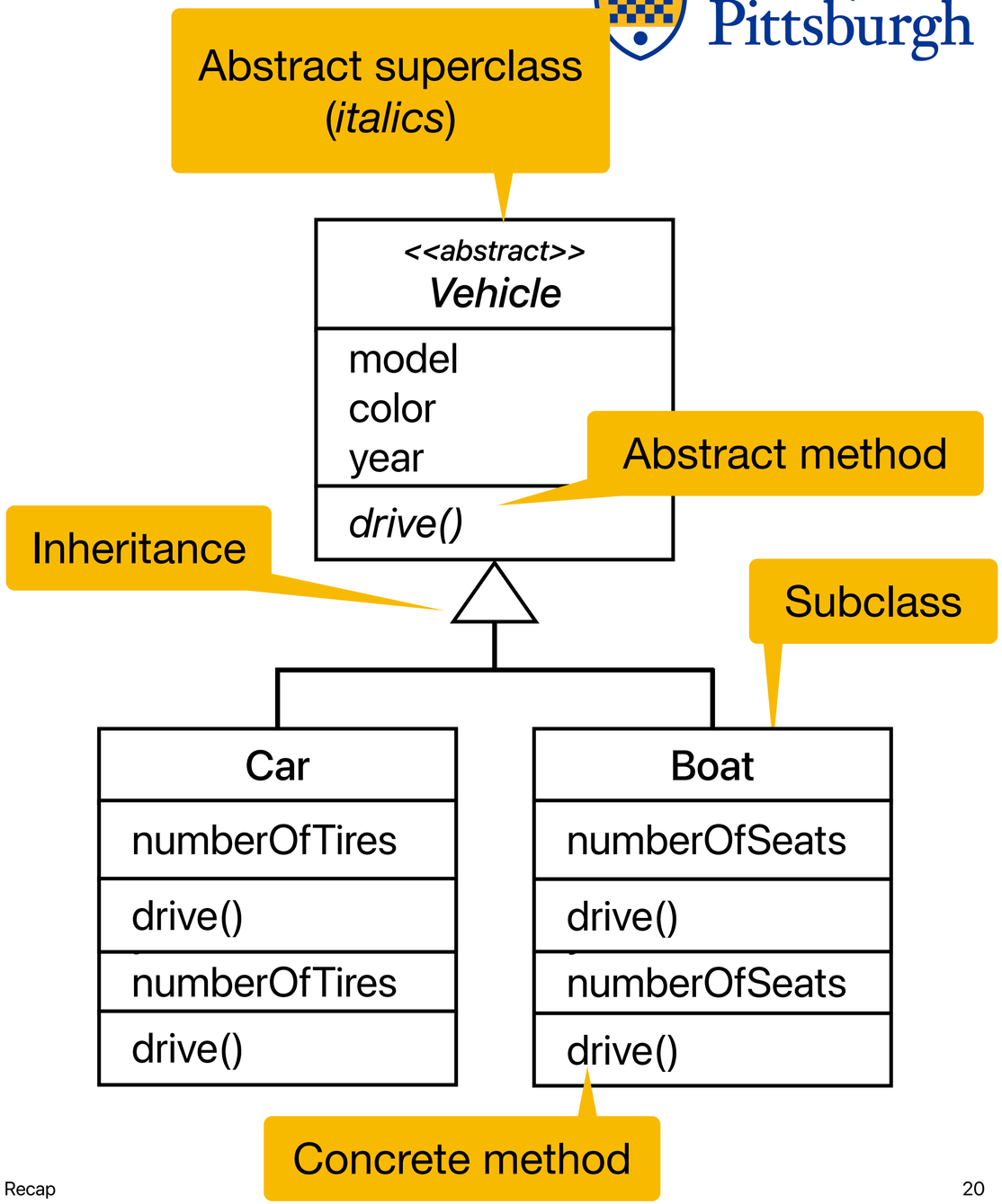



```java
class Car {
      String model;
      String color;
      int year;

      void drive() {
         System.out.print("vroom");
      }
}

class SportsCar extends Car {
      int maxSpeed;

      @Override
      void drive() {
         System.out.print("VROOM 💨");
      }
}
```

Inheritance

Classes vs. abstract classes

- **Class**
  - An abstraction in the context of object oriented programming
  - A class encapsulates state and behavior
- **Abstract class**
  - Cannot be instantiated
  - Define an abstract structure which holds common structure (state) or functionality (behavior)
  - Allows to create a blueprint for other classes without implementing all of the concrete methods and properties themselves
    → Greater code reusability and modularity



Generalization vs Specialization Inheritance

- **Generalization**: A superclass defines common features
- **Specialization**: Subclasses refine or extend the superclass with more specific features

- An **abstract class** represents generalization:
  - There is shared data (attributes) and shared logic.
  - Subclasses are variations of a "is-a" relationship

↔️ An **interface** defines shared functionality, role, or a "contract" across unrelated hierarchies

[Example - SiMCity] Is a CityOfficial a special kind of Resident (specialization)? Or are Resident and CityOfficial both kinds of User (generalization)?

Polymorphism

- *Poly* (= many) + *morph* (= forms) → "many forms"
- Ability of an object reference to take on many forms
- A super class reference (compile time) is used to refer to any specific subclass object (runtime)
- Polymorphism allows us to mix methods and objects of different types in a consistent way
- The polymorphism the object undergoes depends on the *when* & *what* part of the object is transforming
  - **When**: Compile-time vs dynamic
    - Compile-time polymorphism
    - Runtime polymorphism
  - **What**: Method vs object

https://static.tvtropes.org/pmwiki/pub/images/cinderella_restored_transformation.png

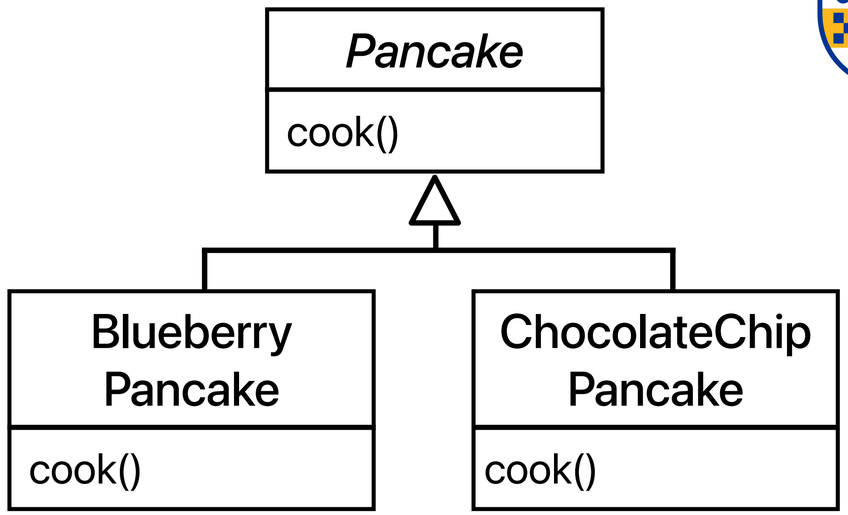
[Example]

```
1 if (type.equals("Cat")) {
2   Cat cat = new Cat();
3   cat.makeSound();
4   cat.describe();
5   cat.showAnimation();
6 } else if (type.equals("Dog")) {
7   Dog dog = new Dog();
8   dog.makeSound();
9   dog.describe();
10  dog.showAnimation();
11 } else {
12  Cow cow = new Cow();
13  cow.makeSound();
14  cow.describe();
15  cow.showAnimation();
16 }
```

"Polymorphism"

```
1 Animal animal;
2 if (type.equals("Cat")) {
3   animal = new Cat();
4 } else if (type.equals("Dog")) {
5   animal = new Dog();
6 } else {
7   animal = new Cow();
8 }
9 animal.makeSound();
10 animal.describe();
11 animal.showAnimation();
```

[Example] Polymorphism




```java
public abstract class Pancake {
    public abstract void cook();
}

public class BlueberryPancake extends Pancake {
    @Override
    public void cook() {
        System.out.println("🫐🥞");
    }
}

public class ChocolateChipPancake extends Pancake {
    @Override
    public void cook() {
        System.out.println("🍫🥞");
    }
}
```

```java
public class PancakeMaker {
  public static void  main(String[] args) {
      Pancake myPancake;

      myPancake = receiveOrder();
      myPancake.cook();
  }

  public static Pancake receiveOrder(Order order) {
      // Order type is based on user input
      // defined somewhere else
      if (order.getType().equals("Blueberry Pancake")) {
          return new BlueBerryPancake();
       } else {
          return new ChocolateChipPancake();
      }
  }
```

Compile time type

Runtime type

Runtime type

OOP principles are tools that help us write better code

University of Pittsburgh

- **Encapsulation/Reusability:** Encapsulating data and behavior allows for efficient code reuse across the application
- **Maintainability:** OOP makes code modification and maintenance easier over time
- **Scalability:** OOP allows code to scale with increasing complexity and new features can be added without breaking existing ones
- **Abstraction:** OOP abstracts away complexity and allows developers to focus on essential aspects of the system
- **Communication & Collaboration:** OOP enables easier collaboration

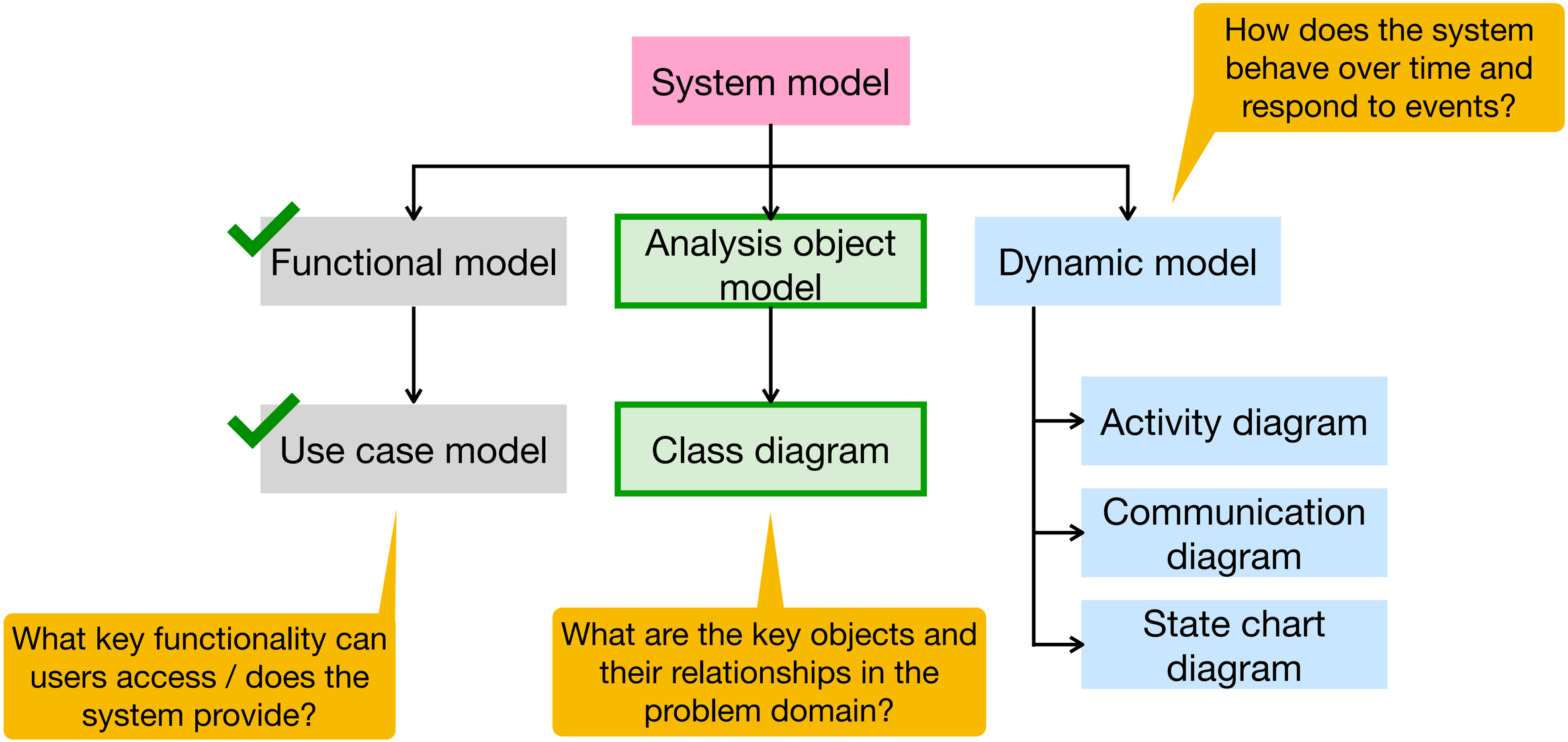
Best Practices

- **Aim for high cohesion**: Ensure that each identified object has a well-defined purpose and responsibilities, promoting high cohesion within the classes
- **Low Coupling:** Minimize dependencies and coupling between objects, aiming for a modular and maintainable system design
- **Iterative Process**: The process may require multiple iterations to refine and validate the identified objects and their relationships as the requirements evolve
- **Documentation**: Document the derived objects, their attributes, behaviors, and relationships in a structured manner, e.g., using class diagrams or entity-relationship diagrams
- **Review and Validation**: Collaborate with stakeholders to review and validate the derived objects, ensuring they accurately represent the requirements

Today's roadmap

- Intro to Analysis
  - Recap: OOP
- Analysis object model

System model — overview



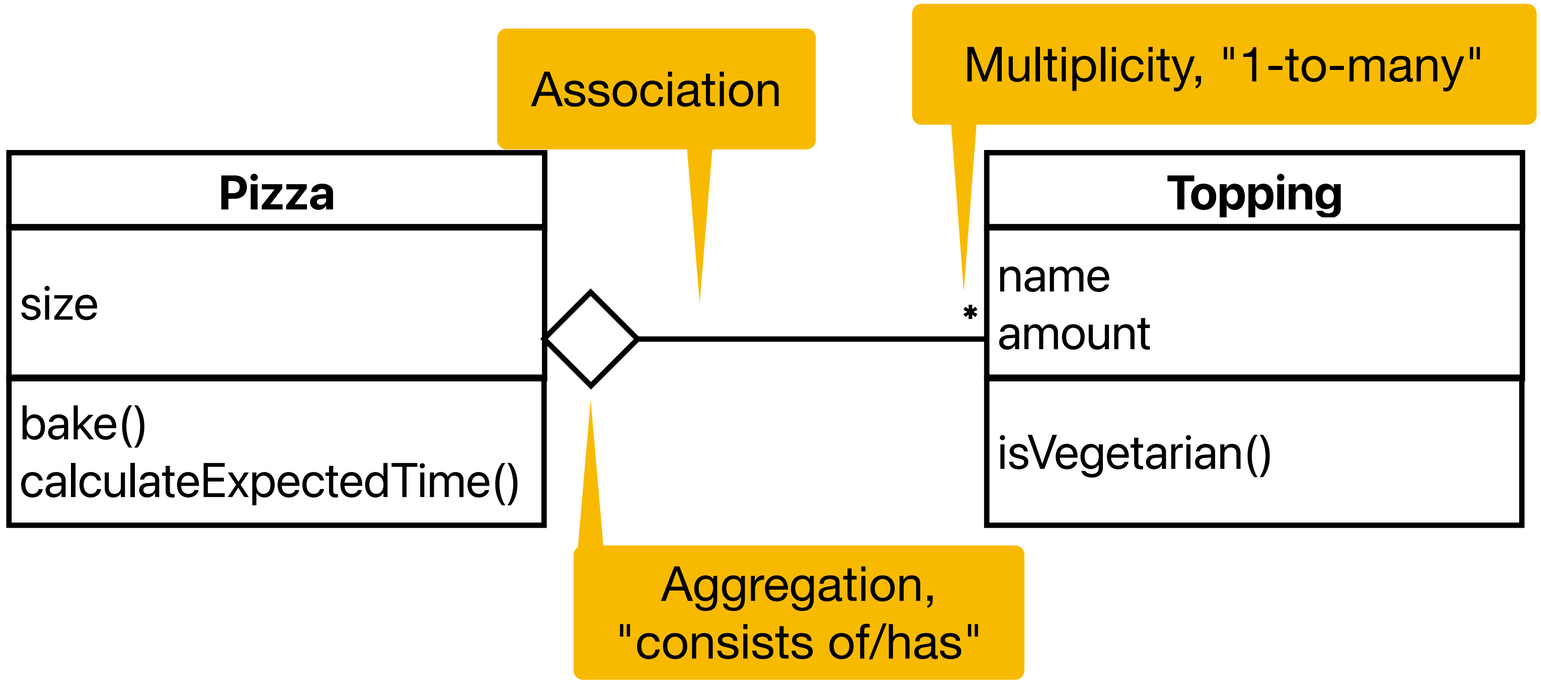
The analysis object model (AOM)

University of Pittsburgh

- Helps in **structuring the problem domain**
  - Ensures that the system accurately represents real-world concepts
  - Identify and structure important concepts, attributes, and associations (relevant to the system's requirements)
  - Bridges the gap between requirements and design
  - Reduces errors
  - Improves communication
  - Facilitates code structure and reusability
- Often depicted using a **UML class diagram**
  - Typically, visibility is not displayed
  - Focus on the most important elements (7+-2!)

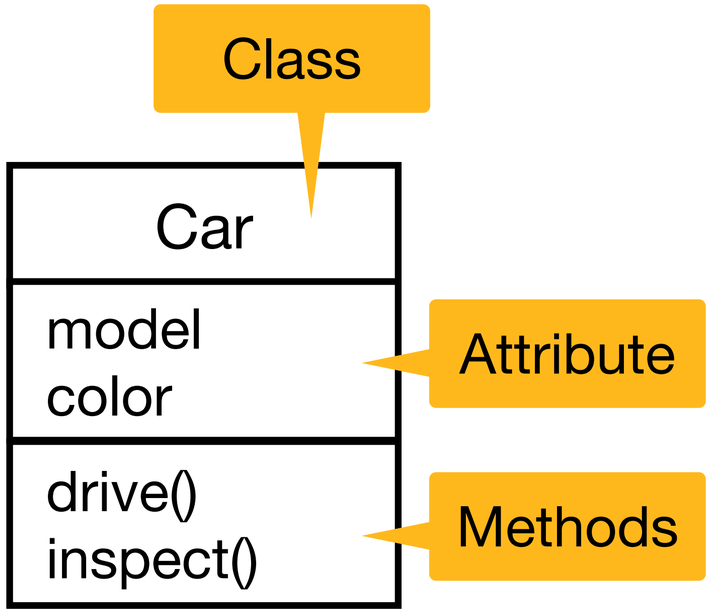
How to think in an abstract way?

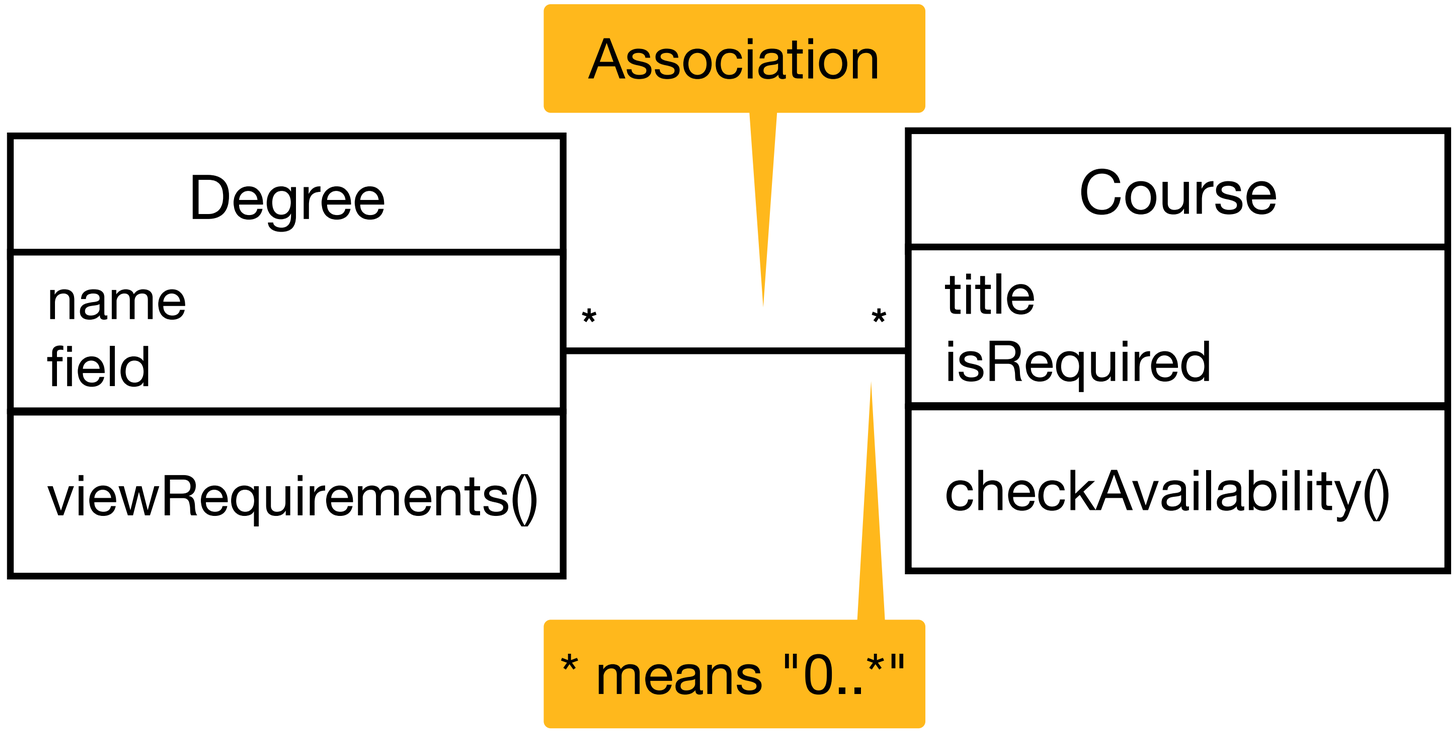
1. What are the main **entities**? → **Objects**
2. What **characteristics/properties** does the entity have? → **Attributes**
3. What **functionality/behavior** does the entity provide/need? → **Methods**
4. How do the entities **communicate** with each other? → **Associations**



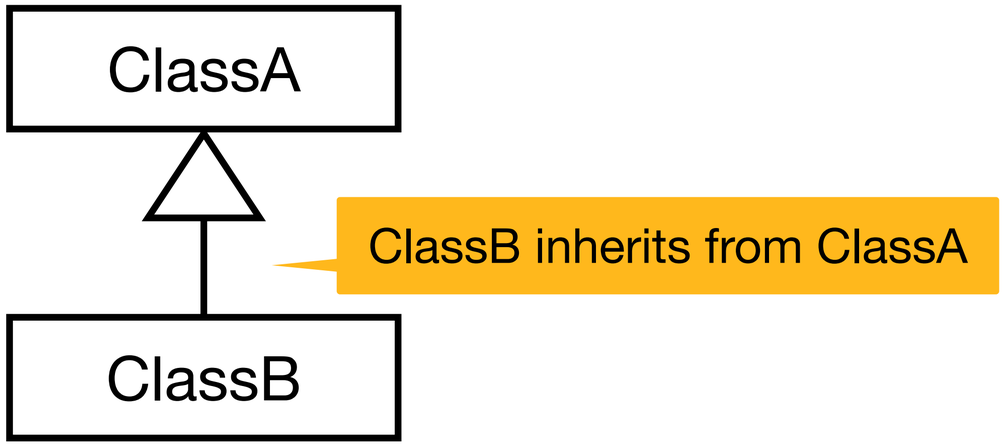
UML class diagram — Elements I

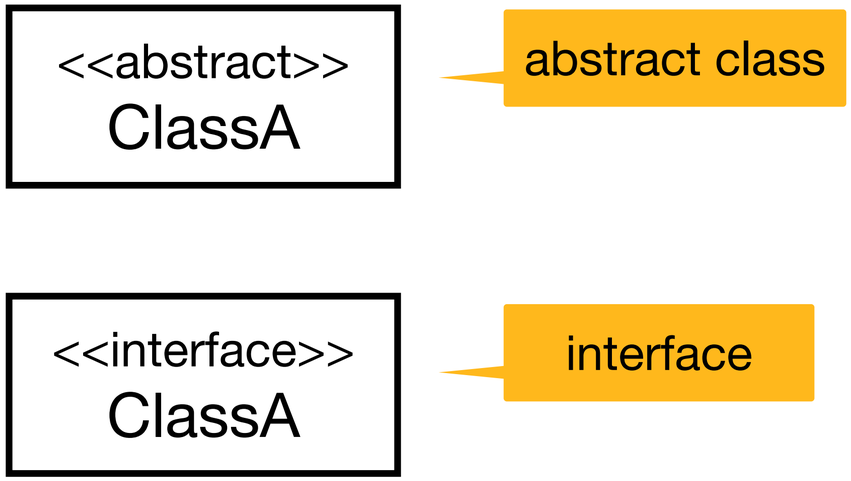
- UML class diagrams are structural models
- Focus on the most important entities, their attributes and methods, and their relationships among each other

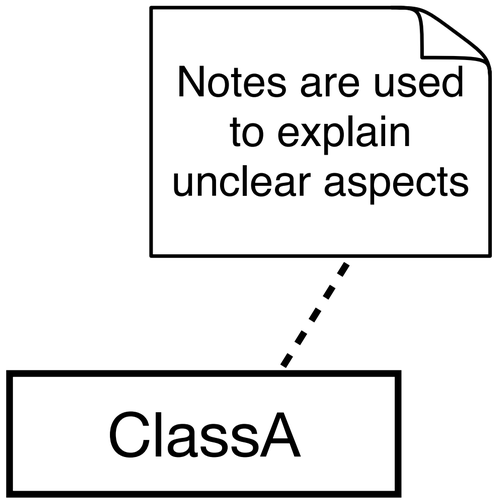




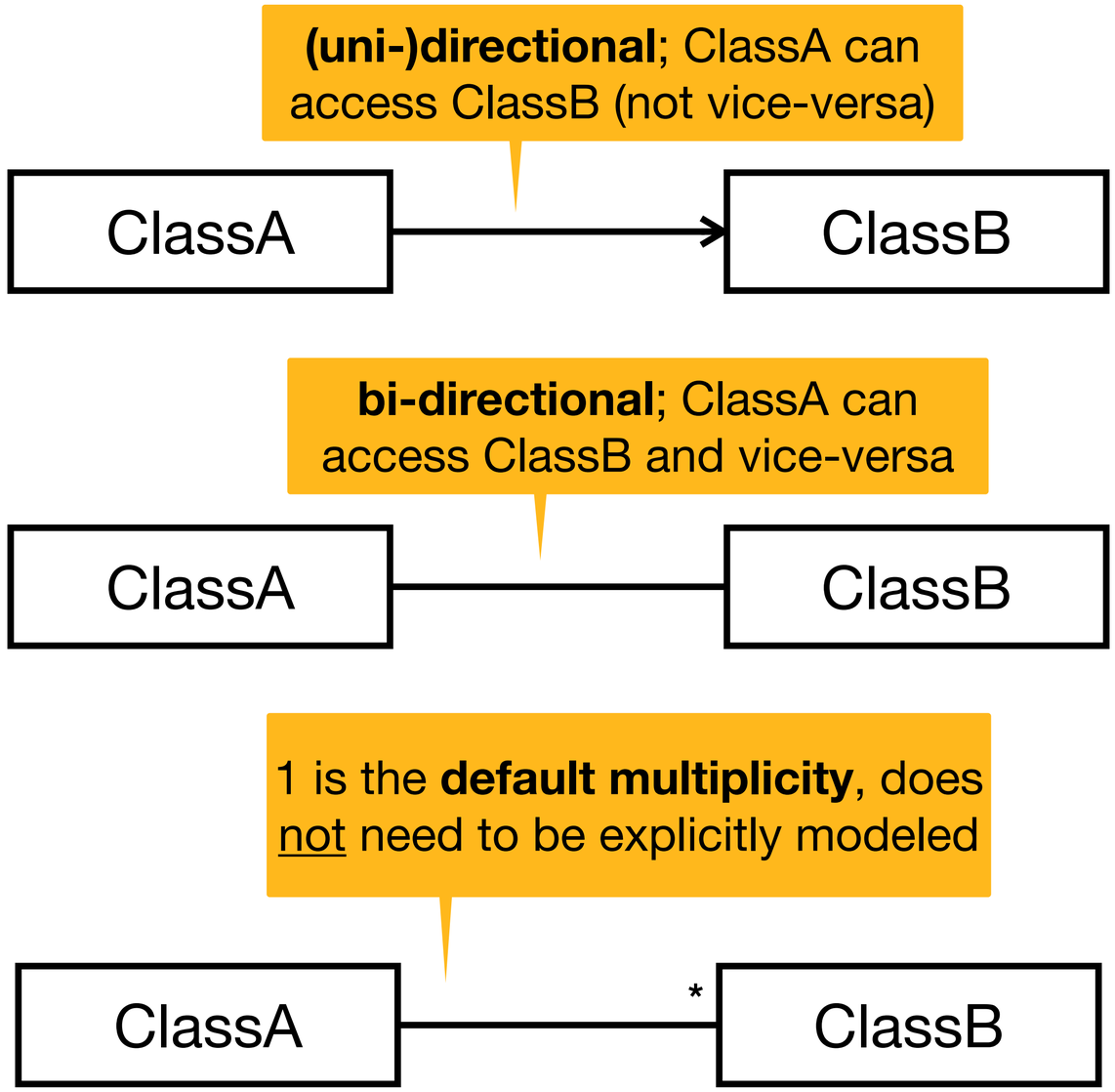
UML class diagram — Elements II







UML class diagram — Basic Associations

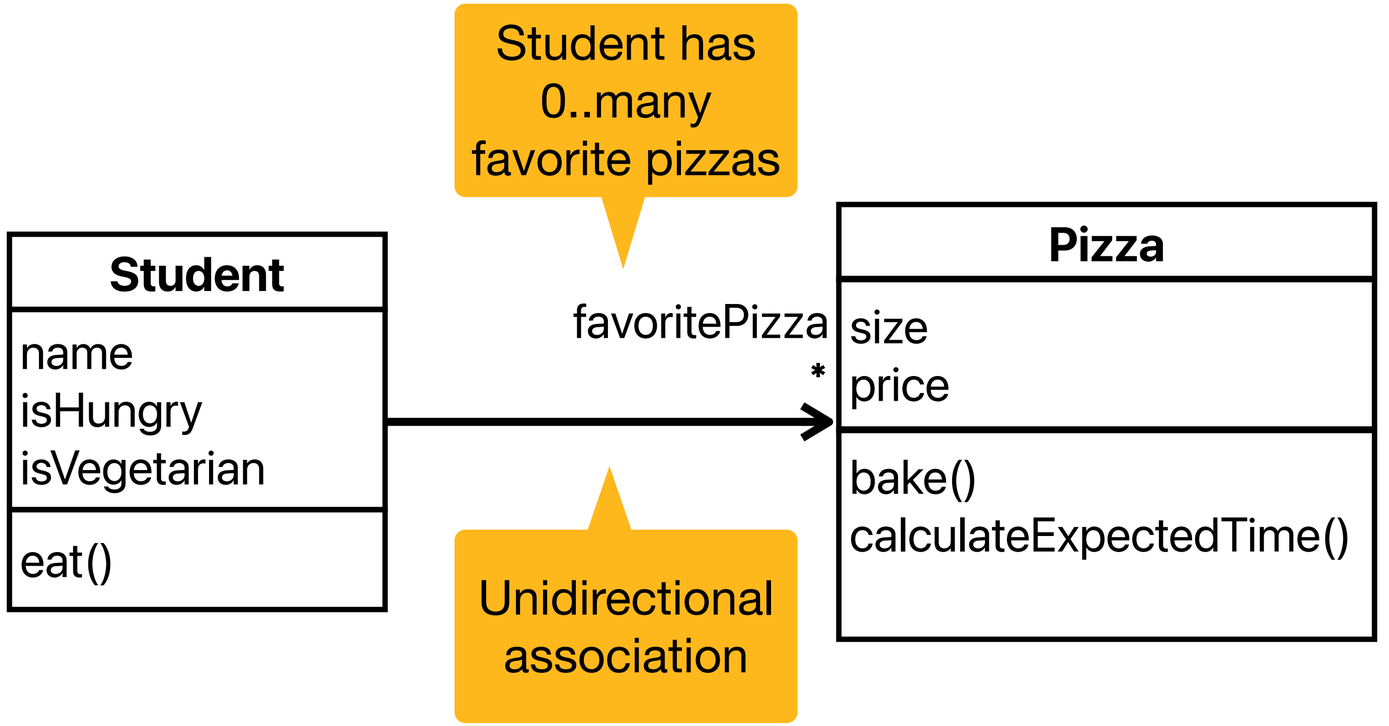


UML class diagram — Associations Overview

- Unidirectional association
- Dependency
- Bidirectional association
- Inheritance
- Aggregation
- Composition

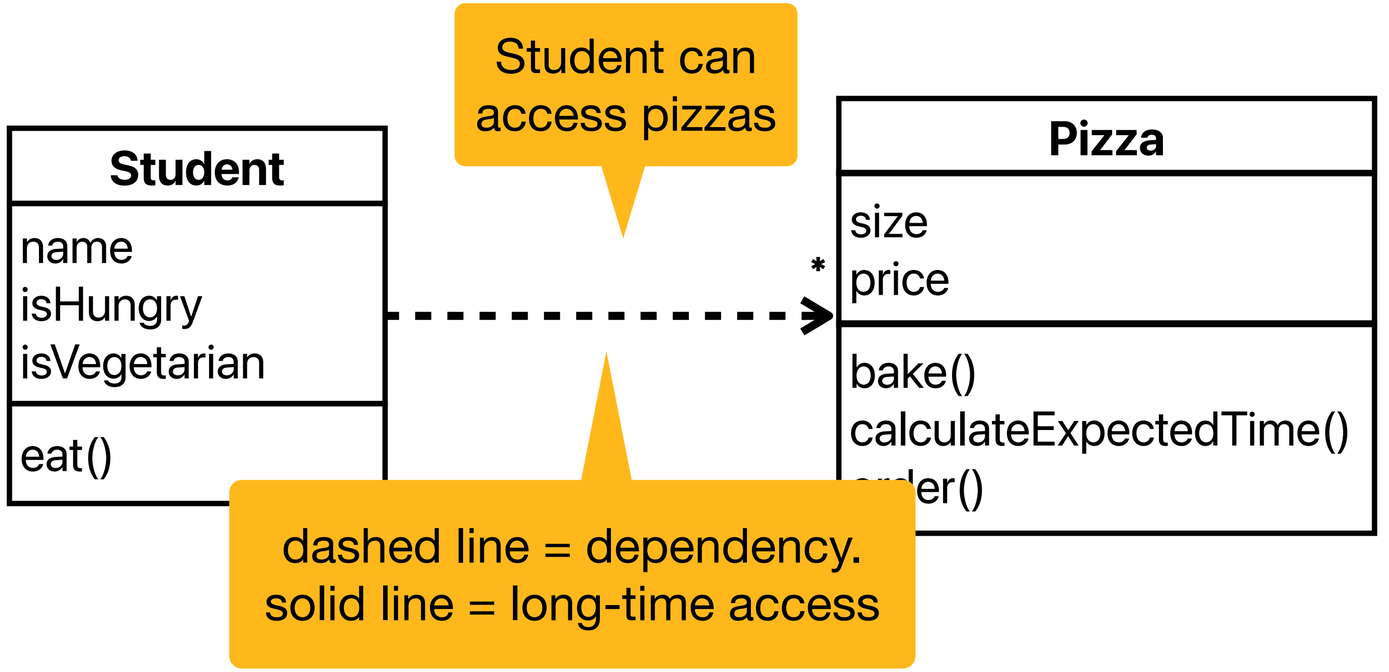
Unidirectional association

- Represents a (long-term) relationship where one element (client) relies on another element (supplier) in some way to perform its functionality
- Can be categorized into various types:
  usage, realization, generalization, and constraint dependencies



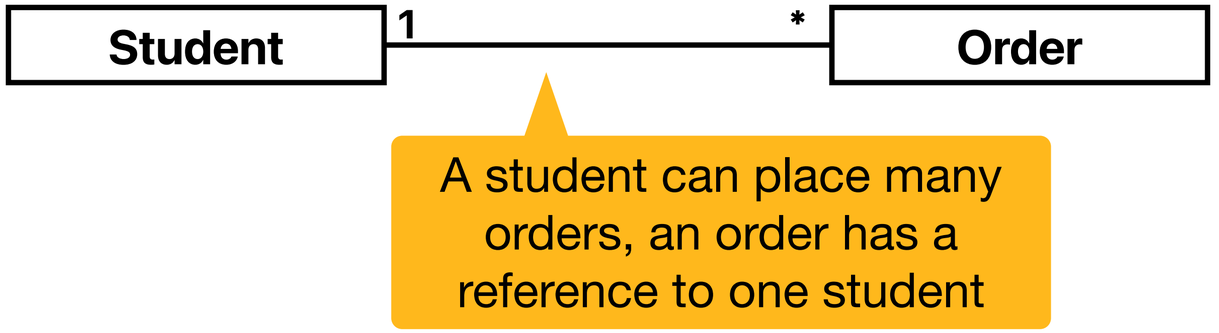
Dependency

- Represents a (temporary) relationship where one element (client) relies on another element (supplier) in some way to perform its functionality
- Can be categorized into various types:
  usage, realization, generalization, and constraint dependencies



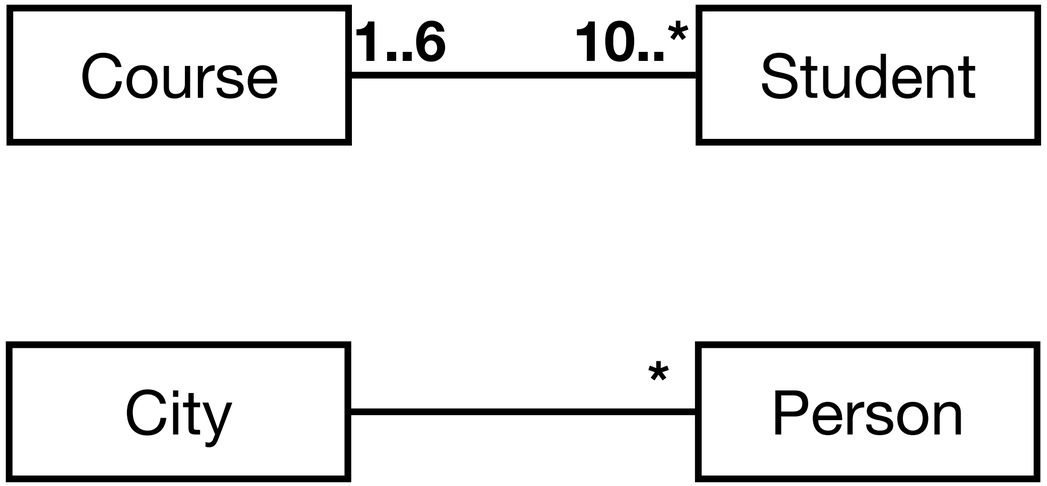
Bidirectional Association

- Mutual relationship between two classes in which both classes have references to each other
- Enables direct interaction and information exchange between the two classes in both directions
- Bi-directional associations are used when both classes need to maintain a symmetrical connection and collaborate with each other



UML class diagram — Multiplicities

- Specifies more details about an association
- Indicates the number of objects that participate in an association
- Also indicates whether an association is mandatory



Multiplicity	Meaning
1	Exactly 1 (default)
*	Zero or more (unlimited)
0..*	Zero or more (unlimited); same as *
1..*	One or more
0..1	Zero or one (optional)
2..42	Specified range
2, 4, 6..8	Multiple, disjoint ranges

Take-Away: Analysis object model (AOM)

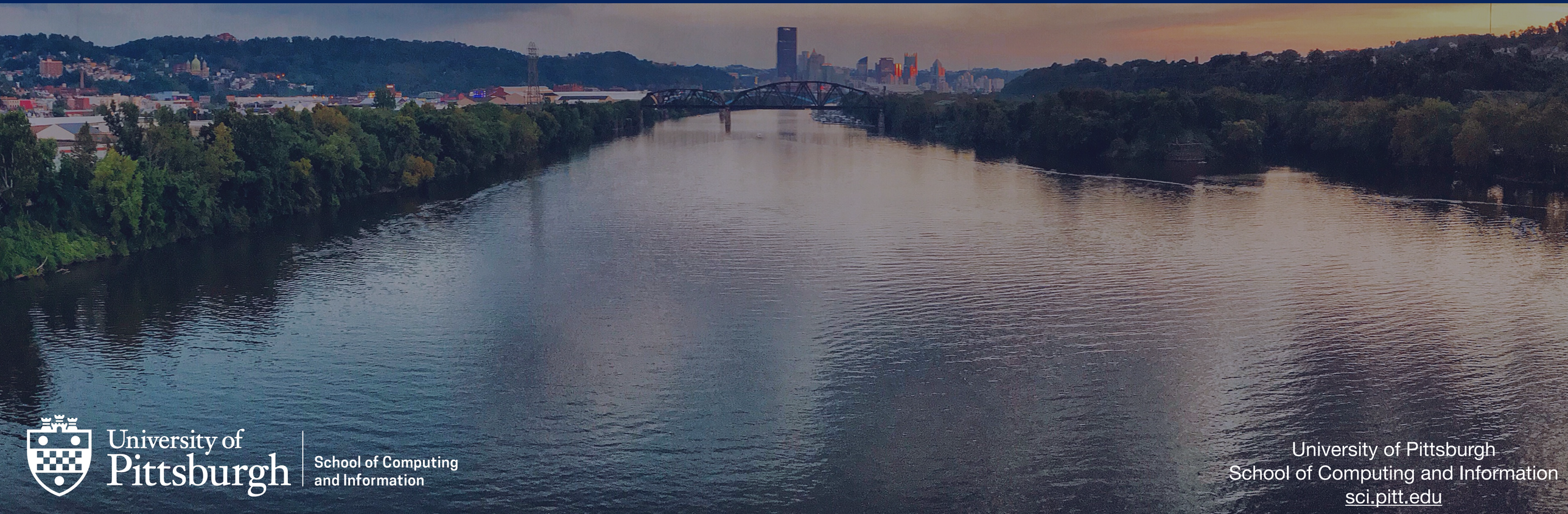
- Defines the system's structure
  - Models the most important objects
  - Defines relevant characteristics and behavior objects provide
  - Defines how objects interact with each other
- High-level view of the problem!
  - Helps in defining a system early on without getting lost in too much detail

Fall 2025

L11 Requirements Analysis & OOP Recap

CS 1530 Software Engineering

Nadine von Frankenberg



University of Pittsburgh
School of Computing and Information

University of Pittsburgh
School of Computing and Information
sci.pitt.edu