

1

OBJECT LANGUAGE AND THEORY

12. CLASS DIAGRAMS

1

2

Objectives

- Describe the static view of the system and show how to capture it in a model.
- Demonstrate how to read and interpret a class diagram.
- Model an association and aggregation and show how to model it in a class diagram.
- Model generalization on a class diagram.

2

3

Content

1. Class diagrams

2. Association

3. Aggregation and Composition

4. Generalization

3

4

1.1. Classes in the UML

- A class is represented using a rectangle with three compartments:
 - The class name
 - The structure (attributes)
 - The behavior (operations)

Professor

- name
- employeeID : UniqueId
- hireDate
- status
- discipline
- maxLoad

+ submitFinalGrade()
+ acceptCourseOffering()
+ setMaxLoad()
+ takeSabbatical()
+ teachClass()

4

5

Classes and Objects

- A class is an abstract definition of an object
 - It defines the structure and behavior of each object in the class.
 - It serves as a template for creating objects.
- Classes are not collections of objects

Professor Torpie

Professor Meijer

Professor Allen

Professor

5

6

What Is an Attribute?

- An attribute is a named property of a class that describes the range of values that instances of the property may hold.
- A class may have any number of attributes or no attributes at all.

Student

Attributes {

- name
- address
- studentID
- dateOfBirth

6

7

Attributes in Classes and Objects

Class

Student

- name
- address
- studentID
- dateOfBirth

:Student

- name = "M. Modano"
- address = "123 Main St."
- studentID = 9
- dateOfBirth = "03/10/1967"

sv1:Student

- name = "D. Hatcher"
- address = "456 Oak Ln."
- studentID = 2
- dateOfBirth = "12/11/1969"

Objects

7

8

What Is an Operation?

- A service that can be requested from an object to effect behavior. An operation has a signature, which may restrict the actual parameters that are possible.
- A class may have any number of operations or none at all.

Student

Operations {

- + get tuition()
- + add schedule()
- + get schedule()
- + delete schedule()
- + has prerequisites()

8

9

Member Visibility

- Visibility is used to enforce encapsulation
- May be public, protected, or private

Public operations

Protected operations

Private operations

UML

9

10

How Is Visibility Noted?

- The following symbols are used to specify export control:
 - + Public access
 - # Protected access
 - - Private access

ClassName
- privateAttribute
+ publicAttribute
protectedAttribute
- privateOperation ()
+ publicOperation ()
protectedOperation ()

UML

10

11

Scope

- Determines number of instances of the attribute/operation
 - Instance: one instance for each class instance
 - Classifier: one instance for all class instances
- Classifier scope is denoted by underlining the attribute/operation name

Class1
- <u>classifierScopeAttr</u>
- instanceScopeAttr
+ <u>classifierScopeOp ()</u>
+ instanceScopeOp ()

UML

11

12

1.2. What Is a Class Diagram?

- Static view of a system

```

classDiagram
    class CloseRegistrationForm {
        +open()
        +closeRegistration()
    }
    class Schedule {
        -semester
        +commit()
        +selectAlternate()
        +removeOffering()
        +level()
        +cancel()
        +getCost()
        +delete()
        +submit()
        +save()
        +anyConflicts()
        +createWithOfferings()
        +updateWithNewSelections()
    }
    class CloseRegistrationController {
        +isRegistrationOpen()
        +closeRegistration()
    }
    class Student {
        +getTuition()
        +addSchedule()
        +getSchedule()
        +deleteSchedule()
        +hasPrerequisites()
    }
    class Professor {
        -name
        -employeeID : UniqueID
        -hireDate
        -status
        -discipline
        -maxLoad
        +submitFinalGrade()
        +acceptCourseOffering()
        +setMaxLoad()
        +takeSabbatical()
        +teachClass()
    }
    CloseRegistrationForm --> Schedule
    CloseRegistrationForm --> CloseRegistrationController
    Student --> Schedule
    Professor --> Schedule
  
```


UML

12

13


Static Structure vs. Dynamic Behavior

- **Static aspects:** Software component and how they are related to one another
- **Dynamic aspects:** How the components interact with one another and/or change state internally over time.



static

vs



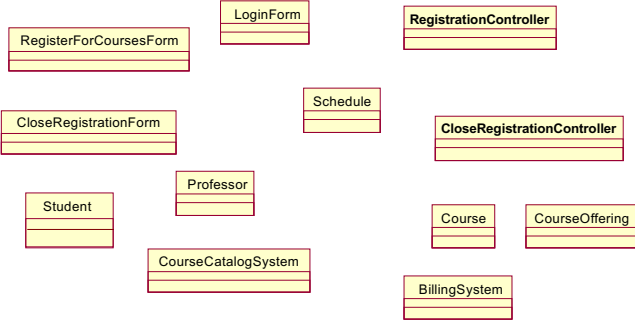
dynamic

13

14

Example: Class Diagram

- Is there a better way to organize class diagrams?

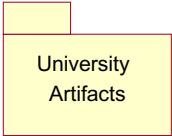


14

15

Review: What Is a Package?


- A general purpose mechanism for organizing elements into groups.
- A model element that can contain other model elements.
- A package can be used:
 - To organize the model under development
 - As a unit of configuration management



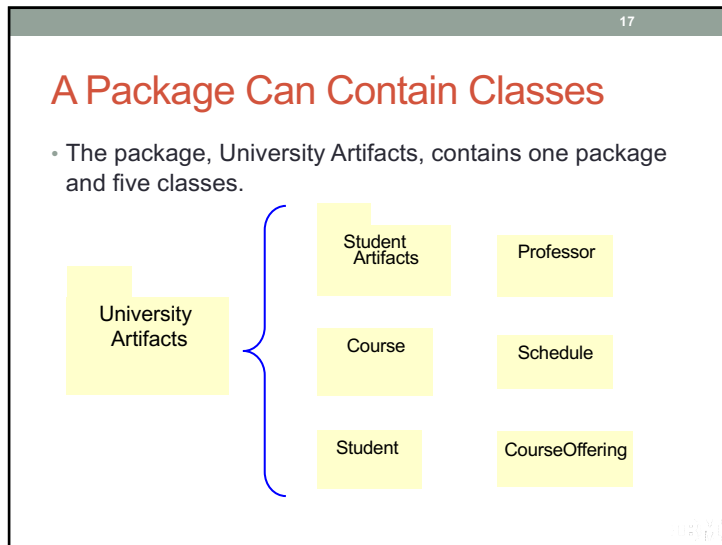
15

16

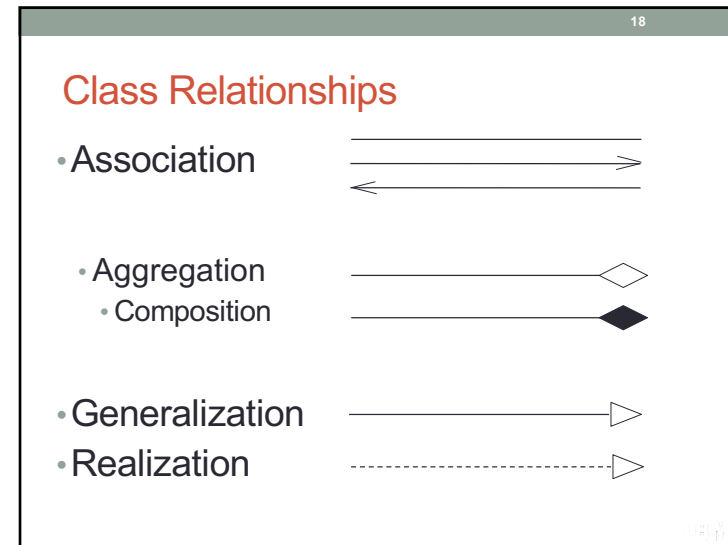
Example: Registration Package



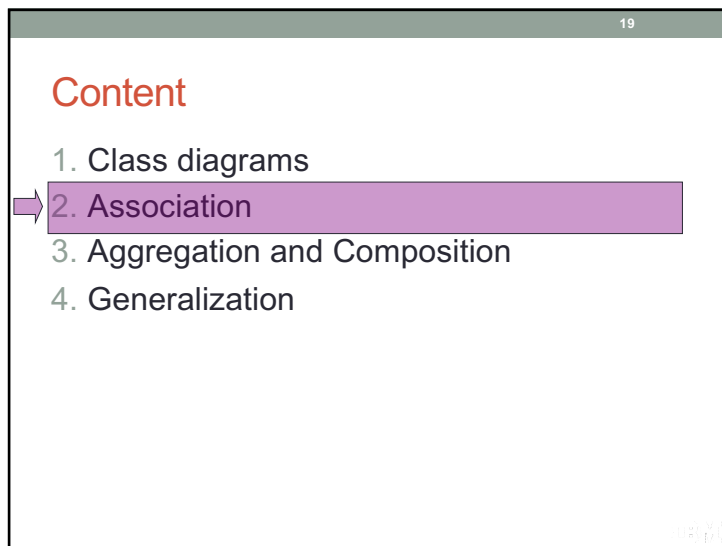
16



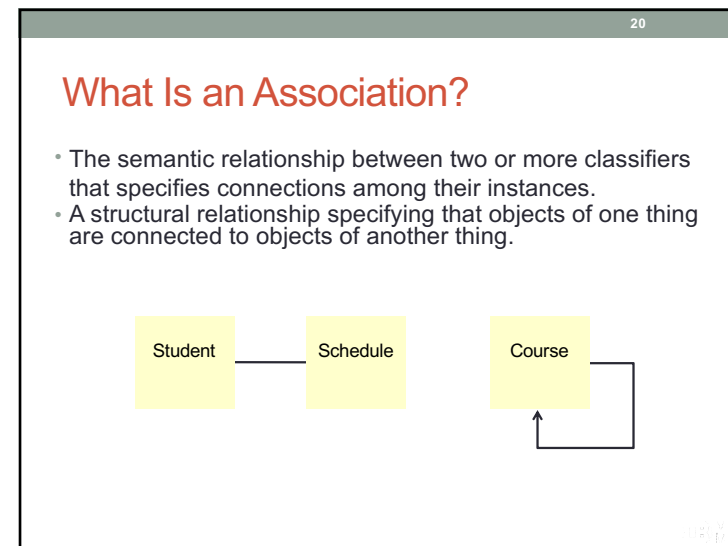
17



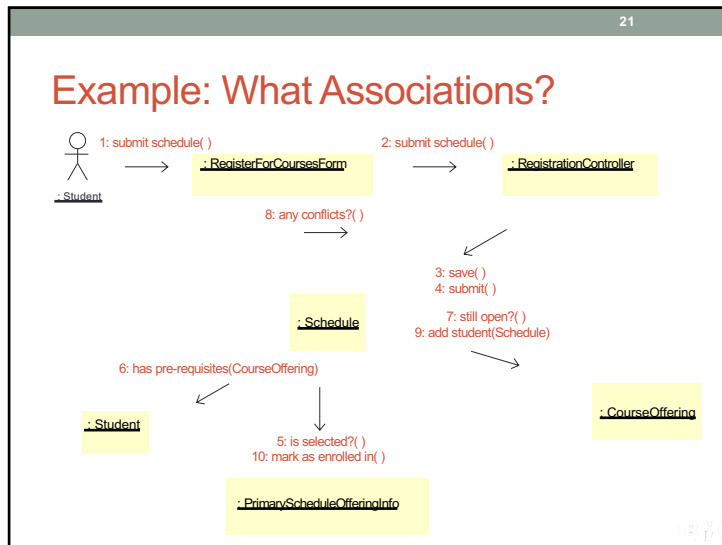
18



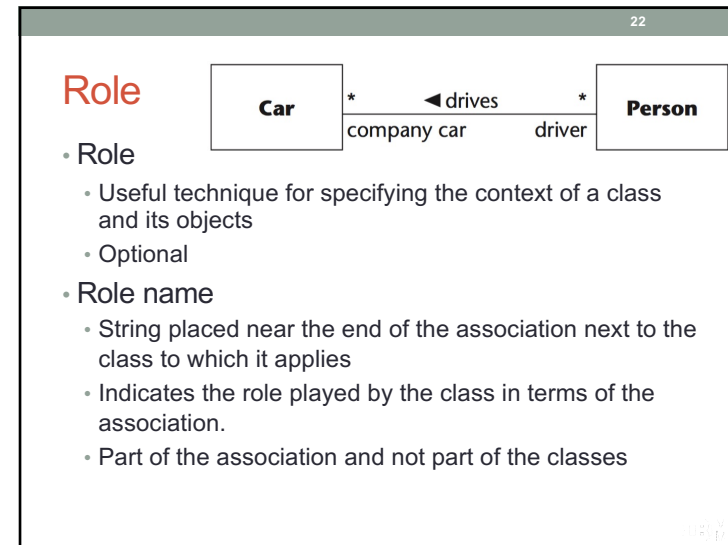
19



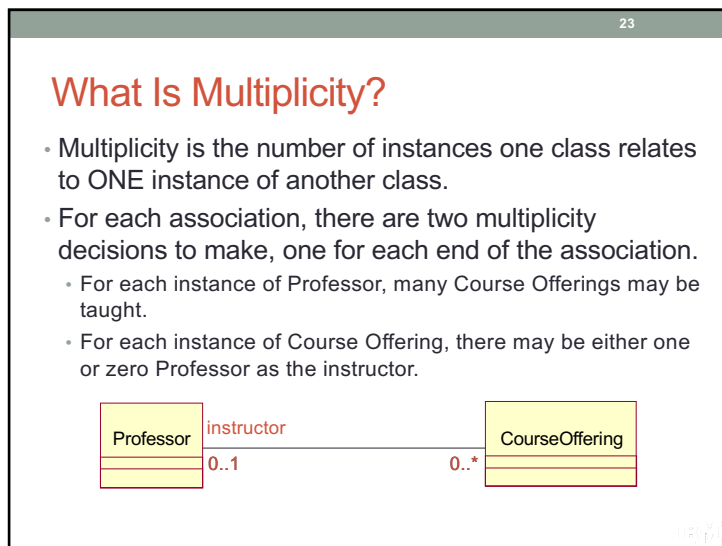
20



21



22



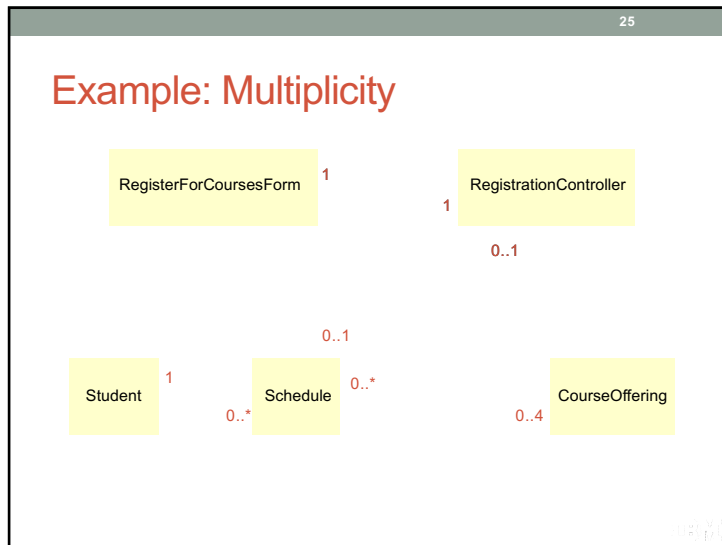
23

24

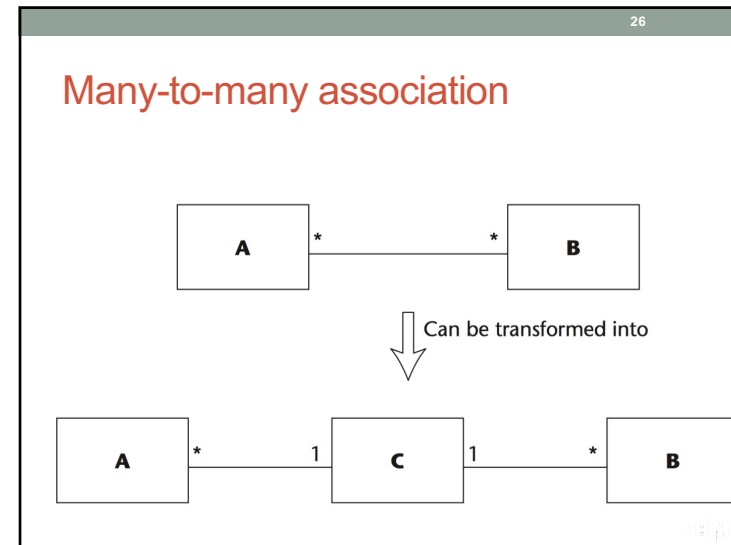
Multiplicity Indicators

Unspecified	
Exactly One	1
Zero or More	0..*
Zero or More	*
One or More	1..*
Zero or One (optional value)	0..1
Specified Range	2..4
Multiple, Disjoint Ranges	2, 4..6

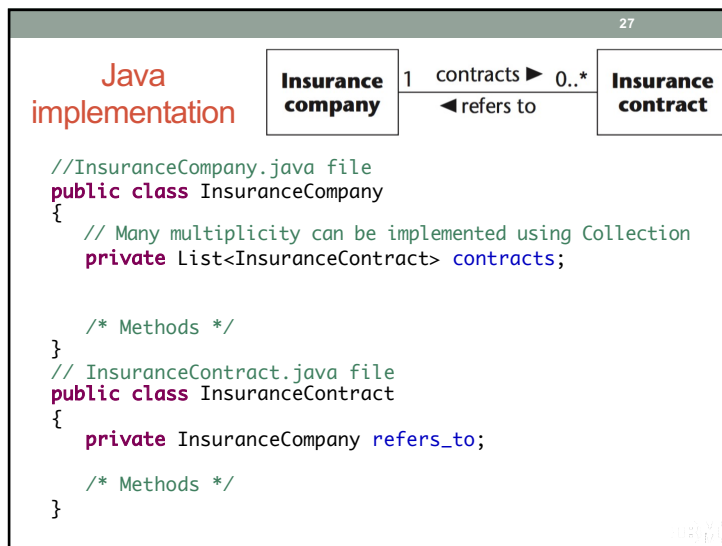
24



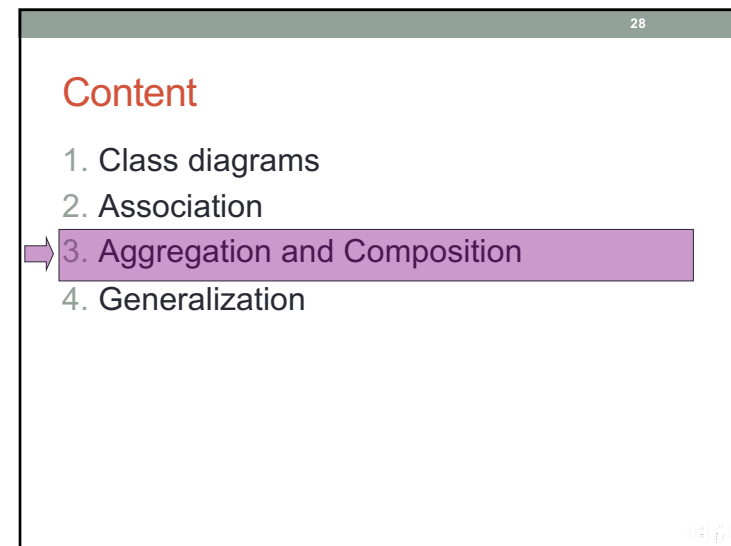
25



26



27



28

What Is an Aggregation?

- A special form of association that models a whole-part relationship between the aggregate (the whole) and its parts.
 - An aggregation is an “is a part-of” relationship.
- Multiplicity is represented like other associations.



29

What is Composition?

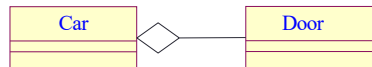
- A special form of aggregation with strong ownership and coincident lifetimes of the part with the aggregate
 - Also called composition aggregate
- The whole “owns” the part and is responsible for the creation and destruction of the part.
 - The part is removed when the whole is removed.
 - The part may be removed (by the whole) before the whole is removed.



30

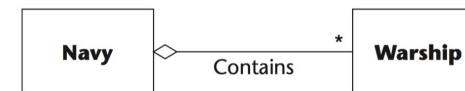
Examples: Association Types

- Association
 - use-a
 - Objects of one class are associated with objects of another class
- Aggregation
 - has-a/is-a-part
 - Strong association, an instance of one class is made up of instances of another class
- Composition
 - Strong aggregation, the composed object can't be shared by other objects and dies with its composer
 - Share life-time

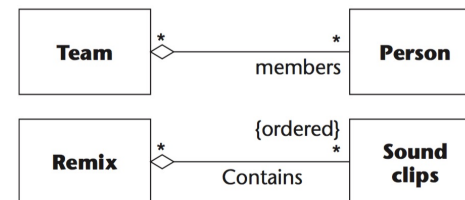


31

Aggregation Example



- A *shared aggregation* is one in which the parts may be parts in any wholes



32

33

Aggregation – Java implementation

```
class Car {
    private List<Door> doors;
    Car(String name, List<Door> doors) {
        this.doors = doors;
    }

    public List<Door> getDoors() {
        return doors;
    }
}
```

UML icons

33

34

Composition Example

- A compound aggregate is shown as attributes in a class

A containing object can have as many parts as we want. However, **all of the parts need to have exactly one container.**

```
classDiagram
    class MessageBoxWindow {
        +Button ok
        +Button cancel
        +Icon information
    }
    class Button
    class Icon
    MessageBoxWindow "0..1" -- "0..1" Button : ok
    MessageBoxWindow "0..1" -- "0..1" Button : cancel
    MessageBoxWindow "0..1" -- "0..1" Icon : information
```

MessageBox Window

ok [0..1] : Button
cancel [0..1] : Button
information [0..1] : Icon

UML icons

34

35

Aggregation – Java implementation

```
final class Car {
    private Engine engine;

    void setEngine(Engine engine) {
        this.engine = engine;
    }

    void move() {
        if (engine != null)
            engine.work();
    }
}

class Engine {
    // starting an engine
    public void work() {
        System.out.println("Engine of car has been started ");
    }
}
```

UML icons

35

36

Composition – Java implementation

```
final class Car {
    private final Engine engine;

    Car(EngineSpecs specs) {
        engine = new Engine(specs);
    }

    void move() {
        engine.work();
    }
}
```

UML icons

36

37

Composition – Java implementation

```

class Person {
    private final Brain brain;

    Person(Brain humanBrain) {
        brain = humanBrain;
    }
}

Brain b = new Brain();
// or we have an instance of Brain in other scopes
// not exactly in this scope

Person p1 = new Person(b);
Person p2 = new Person(b);

```

37

38

Composition – Java implementation

```

public class House {
    private final Room room;

    public House() {
        room = new Room();
    }
}

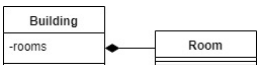
```

The child cannot exist independent of the parent

38

39

Composition – Java implementation



```

class Building {
    class Room {}
    private final List<Room> rooms;
    public Building() {
        rooms = new ArrayList<Room>();
        rooms.add(new Room());
        rooms.add(new Room());
    }
}

class Building {
    List<Room> rooms;

    interface Room {
        void doInRoom();
    }

    Room createAnonymousRoom() {
        return new Room() {
            @Override
            public void doInRoom() {}
        };
    }

    Room createInlineRoom() {
        class InlineRoom implements Room {
            @Override
            public void doInRoom() {}
        }
        return new InlineRoom();
    }

    Room createLambdaRoom() {
        return () -> {};
    }
}

```

The objects' lifecycles are tied. If the whole is destroyed, the part will also be destroyed with it.

Note that doesn't mean that the whole can't exist without any of its parts. E.g., we can tear down all the walls inside a building, hence destroy the rooms. But the building will still exist

39

40

Content

1. Class diagrams
2. Association
3. Aggregation and Composition
- ➔ 4. Generalization

40

41

Review: What Is Generalization?

- A relationship among classes where one class shares the structure and/or behavior of one or more classes.
- Defines a hierarchy of abstractions where a subclass inherits from one or more superclasses.
 - Single inheritance
 - Multiple inheritance
- Is an “is a kind of” relationship.

UML

41

42

Example: Single Inheritance

- One class inherits from another.

Ancestor

Superclass (parent)

```

classDiagram
    class Account {
        - balance
        - name
        - number
        + withdraw()
        + createStatement()
    }
    class Savings
    class Checking
    Account <|-- Savings
    Account <|-- Checking
  
```

Generalization Relationship

Subclasses (children)

Descendents

UML

42

43

Example: Multiple Inheritance

- A class can inherit from several other classes.

Multiple Inheritance

```

classDiagram
    class FlyingThing
    class Animal
    class Airplane
    class Helicopter
    class Bird
    class Wolf
    class Horse
    FlyingThing <|-- Airplane
    FlyingThing <|-- Helicopter
    FlyingThing <|-- Bird
    Animal <|-- Wolf
    Animal <|-- Horse
    Bird <|-- Wolf
    Bird <|-- Horse
  
```

Use multiple inheritance only when needed and always with caution!

UML

43

44

Inheritance Tree Example

```

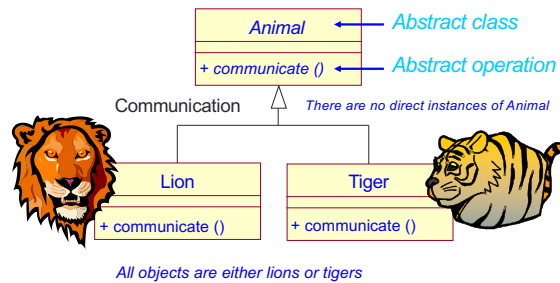
classDiagram
    class Vehicle
    class Car
    class Boat
    class Sports_car
    class Passenger_car
    class Truck
    class Sailing_boat
    class Motorboat
    class Cargo_ship
    Vehicle <|-- Car
    Vehicle <|-- Boat
    Car <|-- Sports_car
    Car <|-- Passenger_car
    Car <|-- Truck
    Boat <|-- Sailing_boat
    Boat <|-- Motorboat
    Boat <|-- Cargo_ship
  
```

UML

44

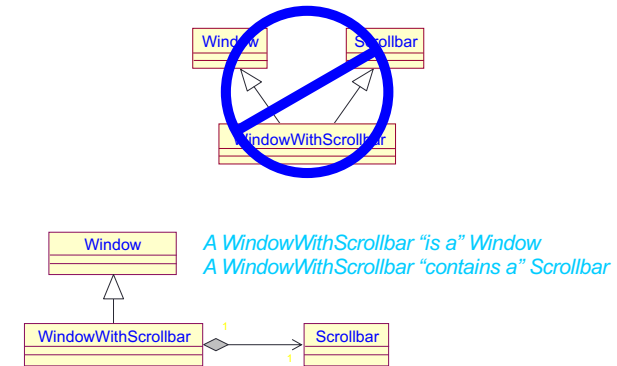
Abstract and Concrete Classes

- Abstract classes cannot have any objects
- Concrete classes can have objects



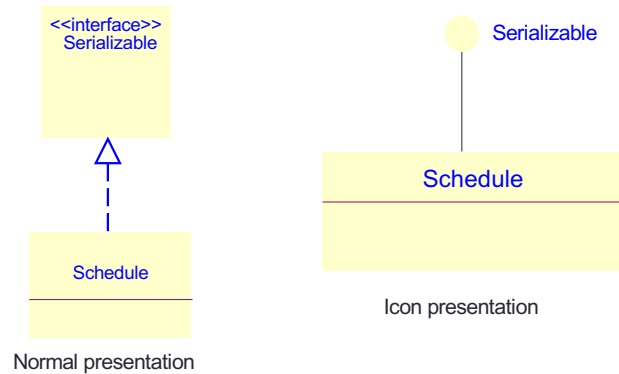
45

Generalization vs. Aggregation



46

Interfaces and Realizes Relationships



47

Exercise

Document a class diagram using the following information:

- A class diagram containing the following classes: Personal Planner Profile, Personal Planner Controller, Customer Profile, and Buyer Record.
- Associations drawn using the following information:
 - Each Personal Planner Profile object can be associated with up to one Personal Planner Controller object.
 - Each Personal Planner Controller object must be related to one Personal Planner Profile.
 - A Personal Planner Controller object can be associated with up to one Buyer Record and Customer Profile object.
 - An instance of the Buyer Record class can be related to zero or one Personal Planner Controller.
 - Zero or one Personal Planner Controller objects are associated with each Customer Profile instance.

48