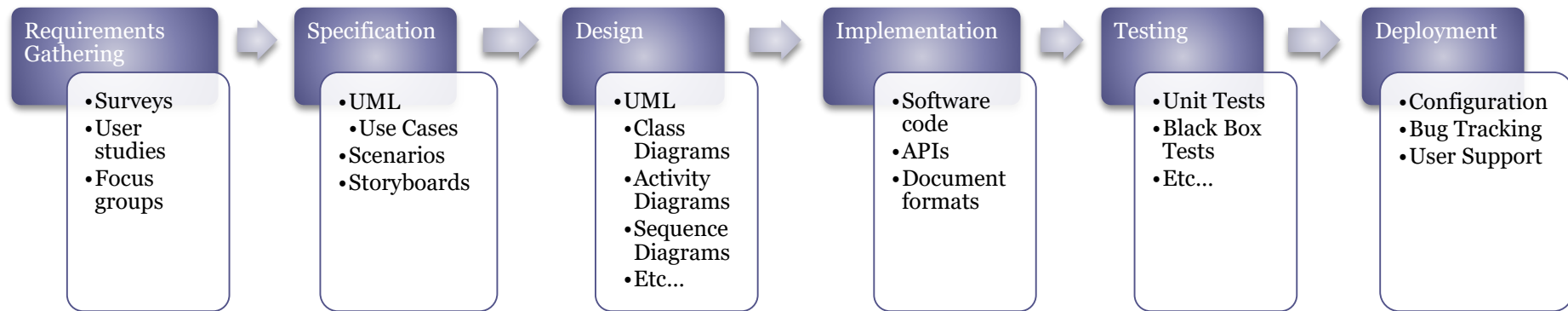
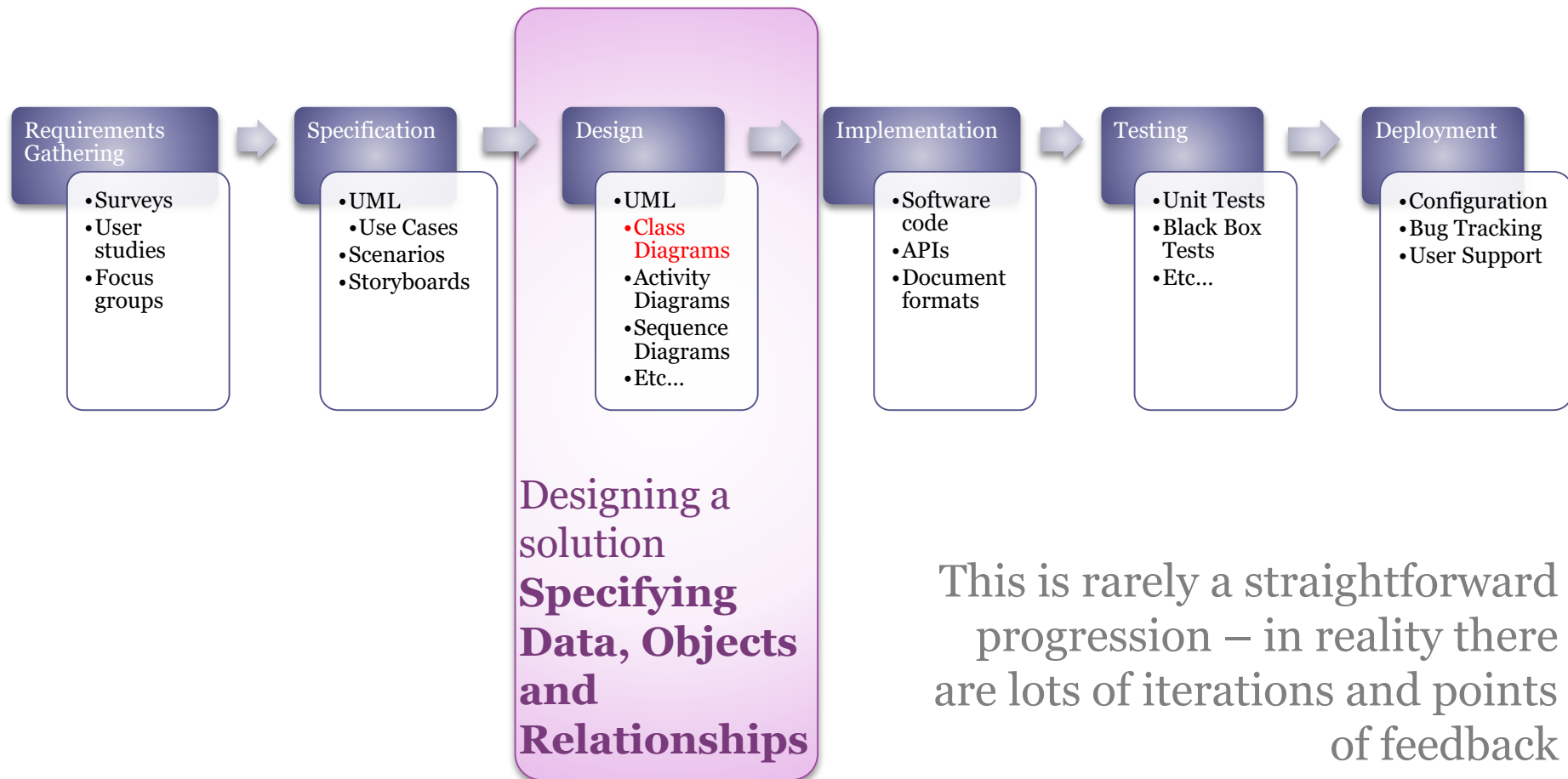


OOAD: Big Picture



This is rarely a straightforward progression – in reality there are lots of iterations and points of feedback

OOAD: Big Picture



UML Class Diagrams

Lecture 6

A series of horizontal lines in teal, light blue, and white, stacked and offset to create a modern, layered effect.

Types of Diagram

Structure Diagrams

- Provide a way for representing the data and static relationships that are in an information system
- You are connecting different parts together to get the final design

Behavioral Model

- Behavioral modeling refers to a way to model the system based on its functionality.

Two Types of Diagram

Behavior

Activity

Use Case

State Transition

Interaction

Sequence

Communication

Interaction
Overview

Timing

Structure

Class

Object

Component

Composite

Deployment

Package

Profile

What is UML Class Diagrams

- What is a UML class diagram? Imagine you were given the task of drawing a family tree. The steps you would take would be:
 - Identify the main members of the family
 - Identify how they are related to each other
 - Find the characteristics of each family member
 - Determine relations among family members
 - Decide the inheritance of personal traits and characters

Basics of UML Class Diagrams

- A software application is comprised of classes and a diagram depicting the relationship between each of these classes would be the class diagram.
- A class diagram is a pictorial representation of the detailed system design

Relationship between Class Diagram and Use Cases

- How does a class diagram relate to the use case diagrams that we learned before?

Relationship between Class Diagram and Use Cases

- When you designed the **use cases**, you must have realized that the use cases talk about "**what are the requirements**" of a system.
- The aim of designing **classes** is to **convert this "what" to a "how"** for each requirement
- **Each use case is further analyzed and broken up that form the basis for the classes that need to be designed**

Elements of a Class Diagram

- A class diagram is composed primarily of the following elements that represent the system's business entities:
 - **Class:** A class represents an entity of a given system. It provides captured implementation in the form of certain functionality of a given entity. These are exposed by the class to other classes as *methods*
 - Apart from functionality, a class also has properties that reflect unique features of a class. The properties of a class are called *attributes*.

Naming Convention

Class naming: Use **singular names**

- Because each class represents a generalized version of a singular object.

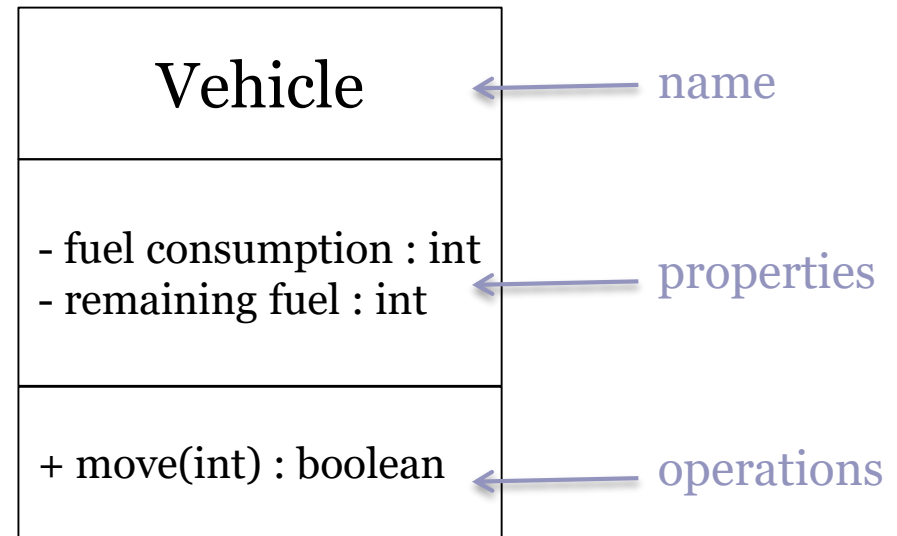
Classes

We need to store several sorts of data about vehicles, including their fuel consumption and level of remaining fuel. Vehicles can move a given distance assuming that they have enough fuel.

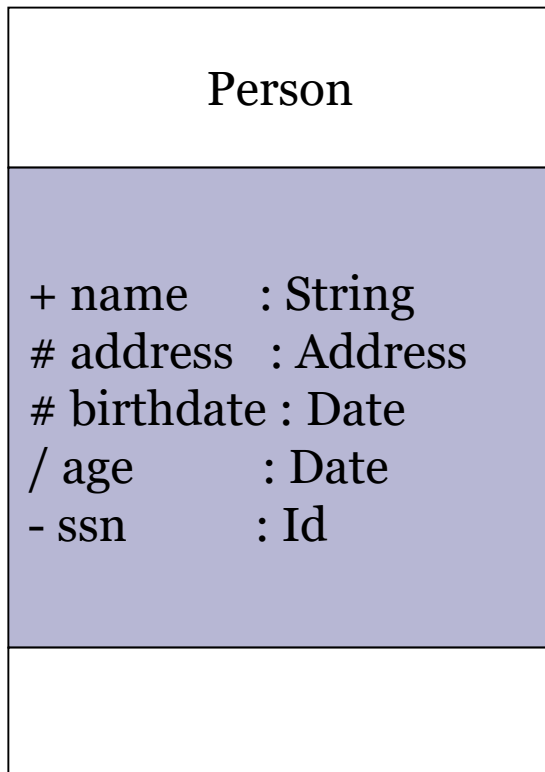
- What is the name of the class?
- What are its properties?
- What are its operations?

Classes

We need to store several sorts of data about vehicles, including their fuel consumption and level of remaining fuel. Vehicles can move a given distance assuming that they have enough fuel.



Class Attribute



attributeName : Type

“-” private

“#” protected

“+” public

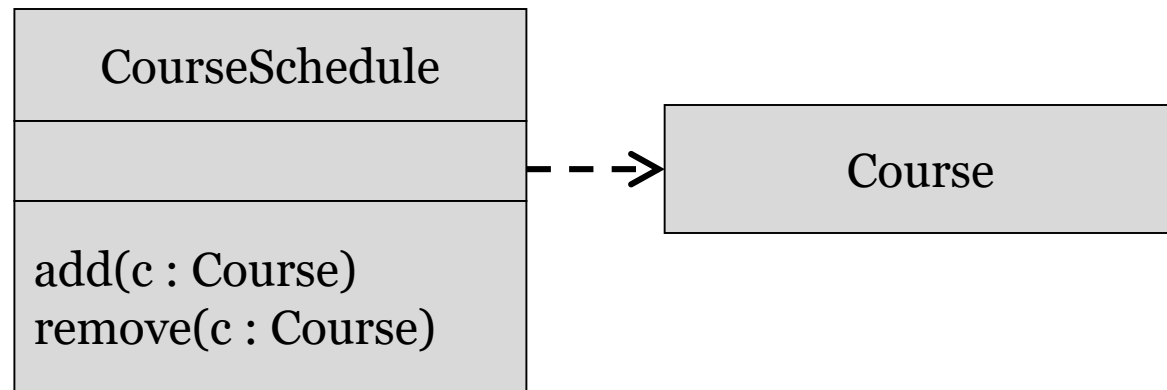
“/” derived

Relationships

- In UML, object interconnections (logical or physical), are modeled as relationships.
- There are three kinds of relationships in UML:
 - Dependencies
 - Generalizations
 - Associations

Dependency

- Dependency is represented when a reference to one class is passed in as a method parameter to another class.



```
public class A {  
    public void doSomething(B b) {
```


Generalization

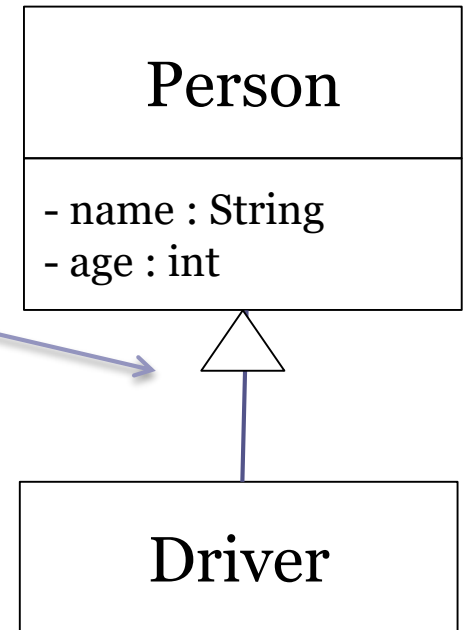
Drivers are a type of person. Every person has a name and an age.

UML Class Diagrams: Generalization

Drivers are a type of person. Every person has a name and an age.

Note: we use a special kind of arrowhead to represent generalization

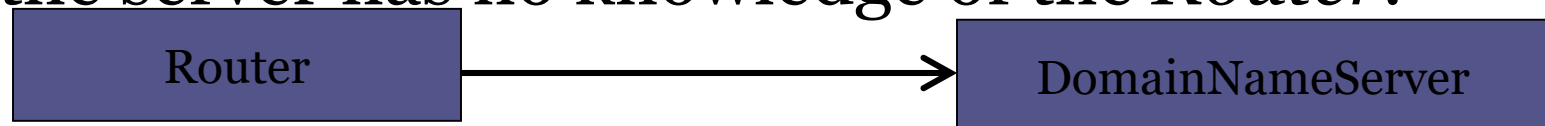
```
public Person {  
    ...  
} // class Person  
public class Driver extends Person{  
    ....  
} // class Driver
```



We assume that **Driver inherits** all the properties and operations of a **Person** (as well as defining its own)

One-way Association

- We can constrain the association relationship by defining the *navigability* of the association.
- In one way association, We can navigate along a single direction only
- Denoted by an arrow towards the server object
- A *Router* object requests services from a *DNS* object by sending messages to (invoking the operations of) the server.
- The direction of the association indicates that the server has no knowledge of the *Router*.

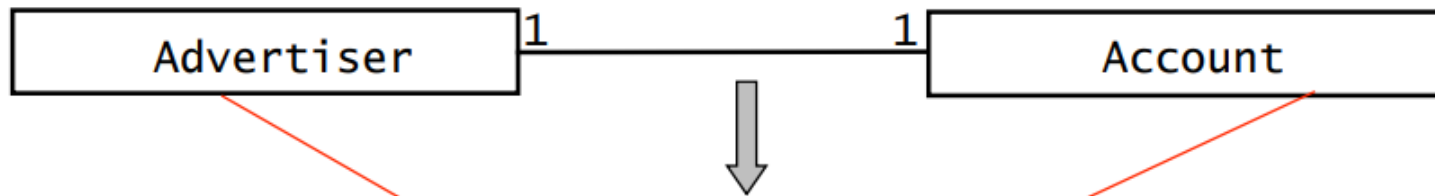


One way Association-Person-Address

```
class Person {  
    string Name;  
    Address addr;  
    int Age;  
    public:  
    Person(){..  
    ~Person{..  
};
```

```
class Address {  
  
    string Street;  
    long postalCode;  
    string Area;  
    ....  
}
```

One way Association



Source code after transformation:

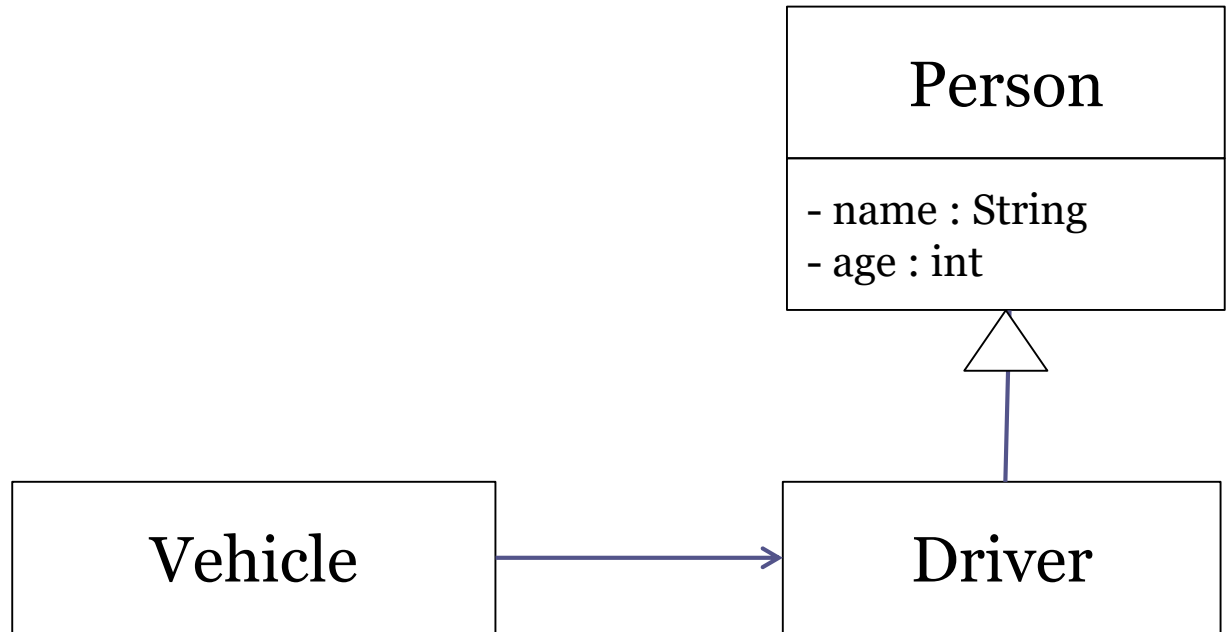
```
public class Advertiser {
    private Account account;
    public Advertiser() {
        account = new Account();
    }
    public Account getAccount() {
        return account;
    }
}
```

Red arrows indicate the transformation: one from the **Advertiser** class box to the **Advertiser** class definition, and another from the **Account** class box to the **private Account account;** line in the code.

One to one Relationship

Composition

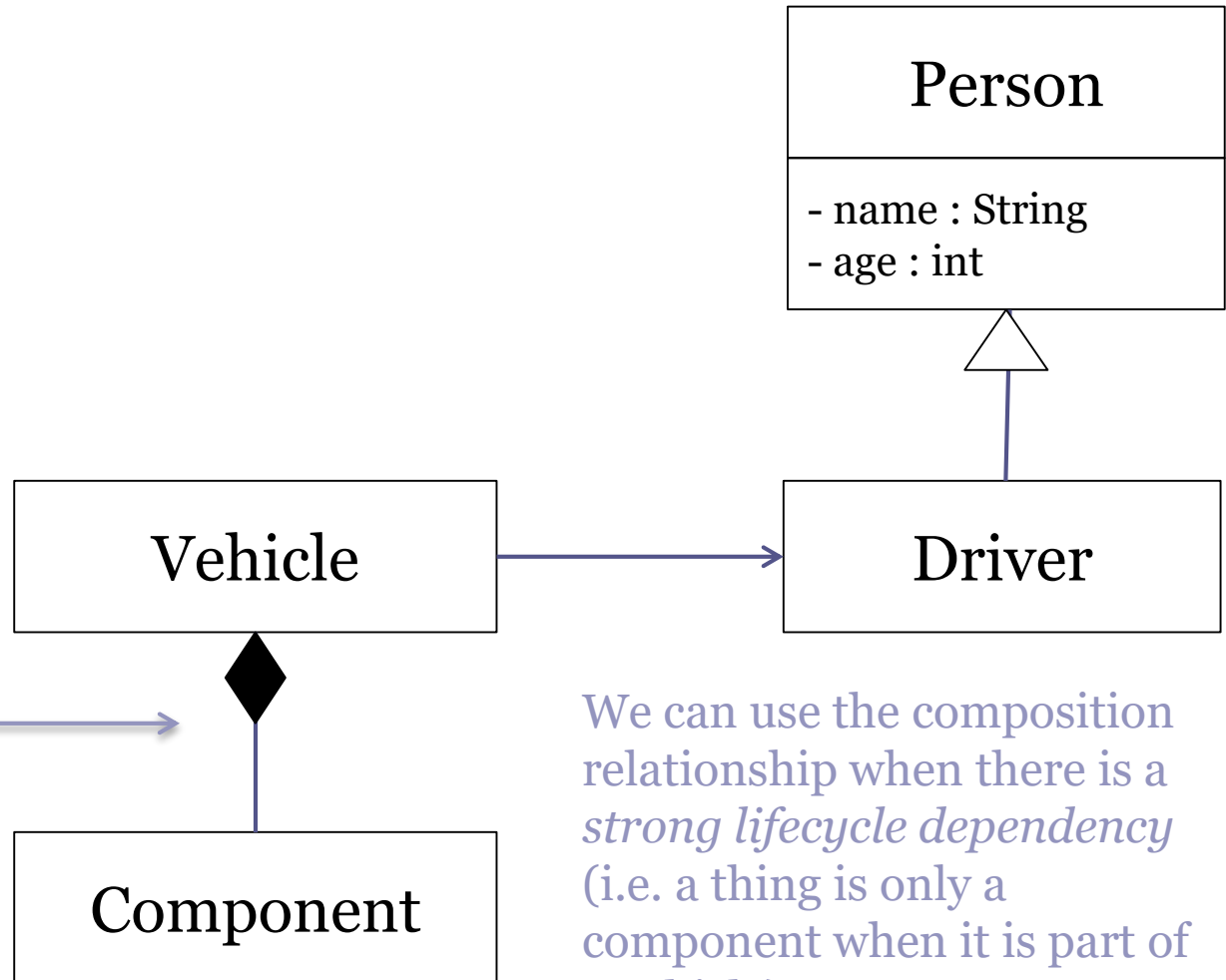
Vehicles are
made up of
many
components.



Composition

Vehicles are made up of many components.

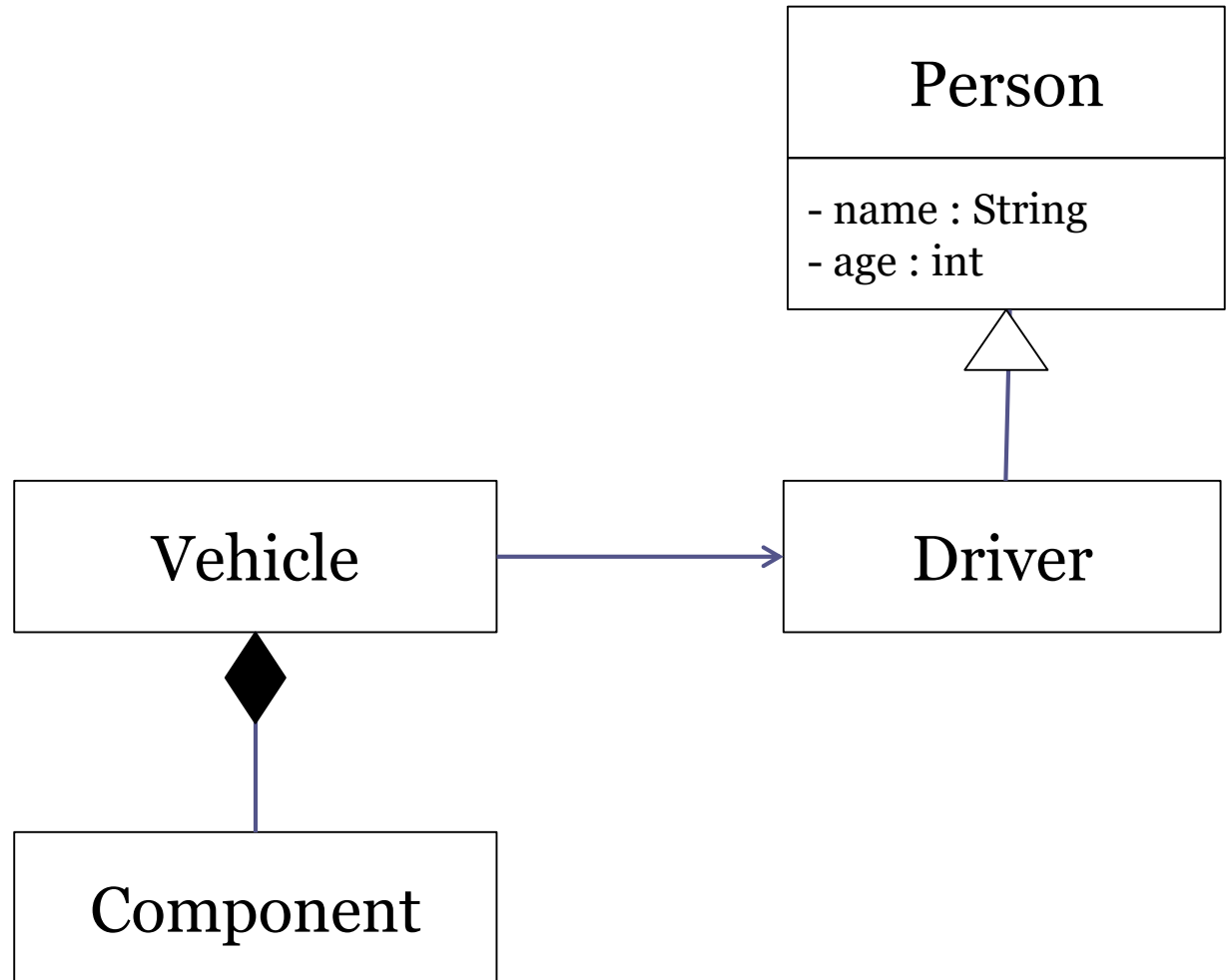
Note: we use a solid diamond to represent composition



We can use the composition relationship when there is a *strong lifecycle dependency* (i.e. a thing is only a component when it is part of a vehicle)

Aggregation

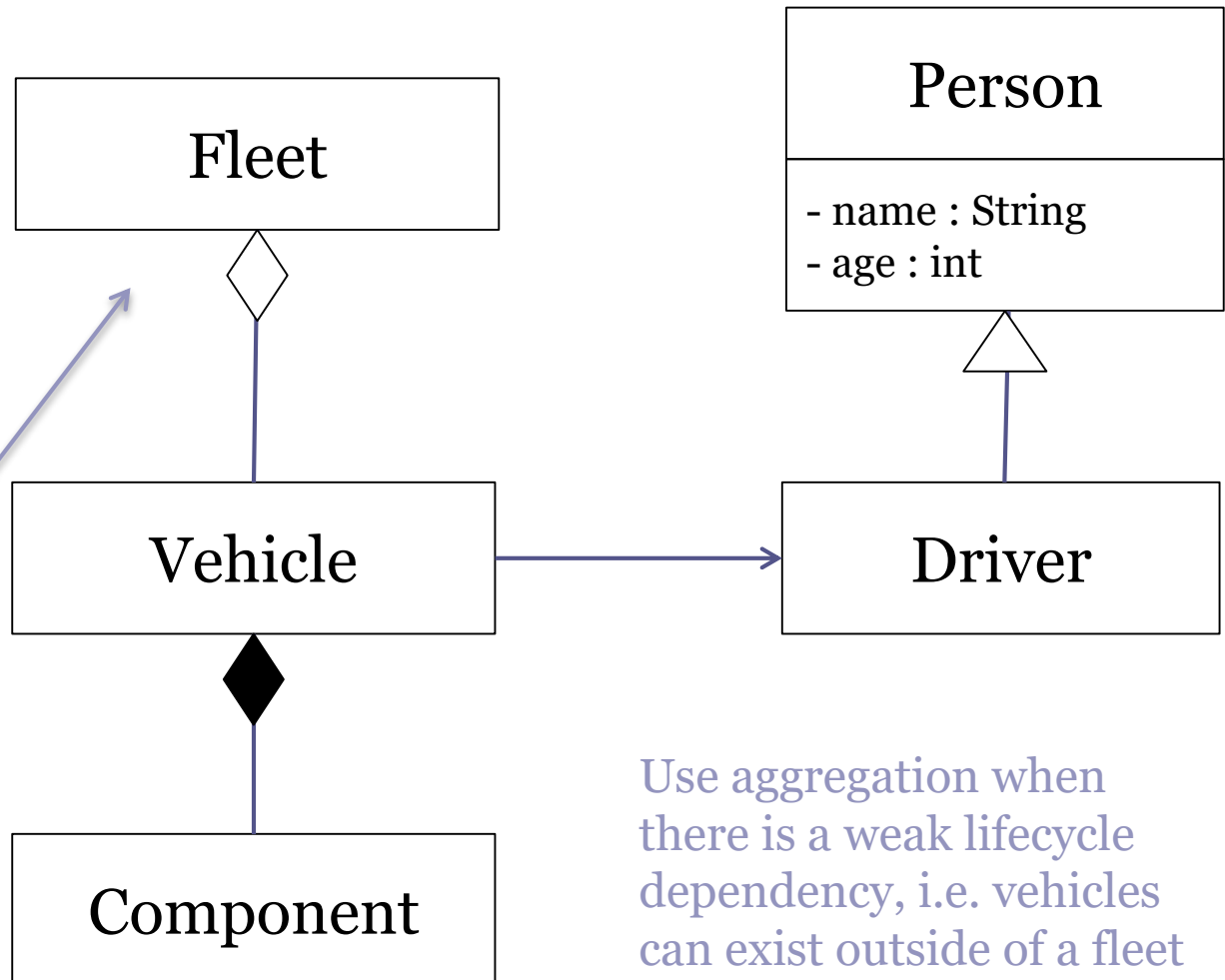
Vehicles are managed in a collection called a Fleet.



Aggregation

Vehicles are managed in a collection called a Fleet.

Note: we use an empty diamond to represent aggregation



Use aggregation when there is a weak lifecycle dependency, i.e. vehicles can exist outside of a fleet

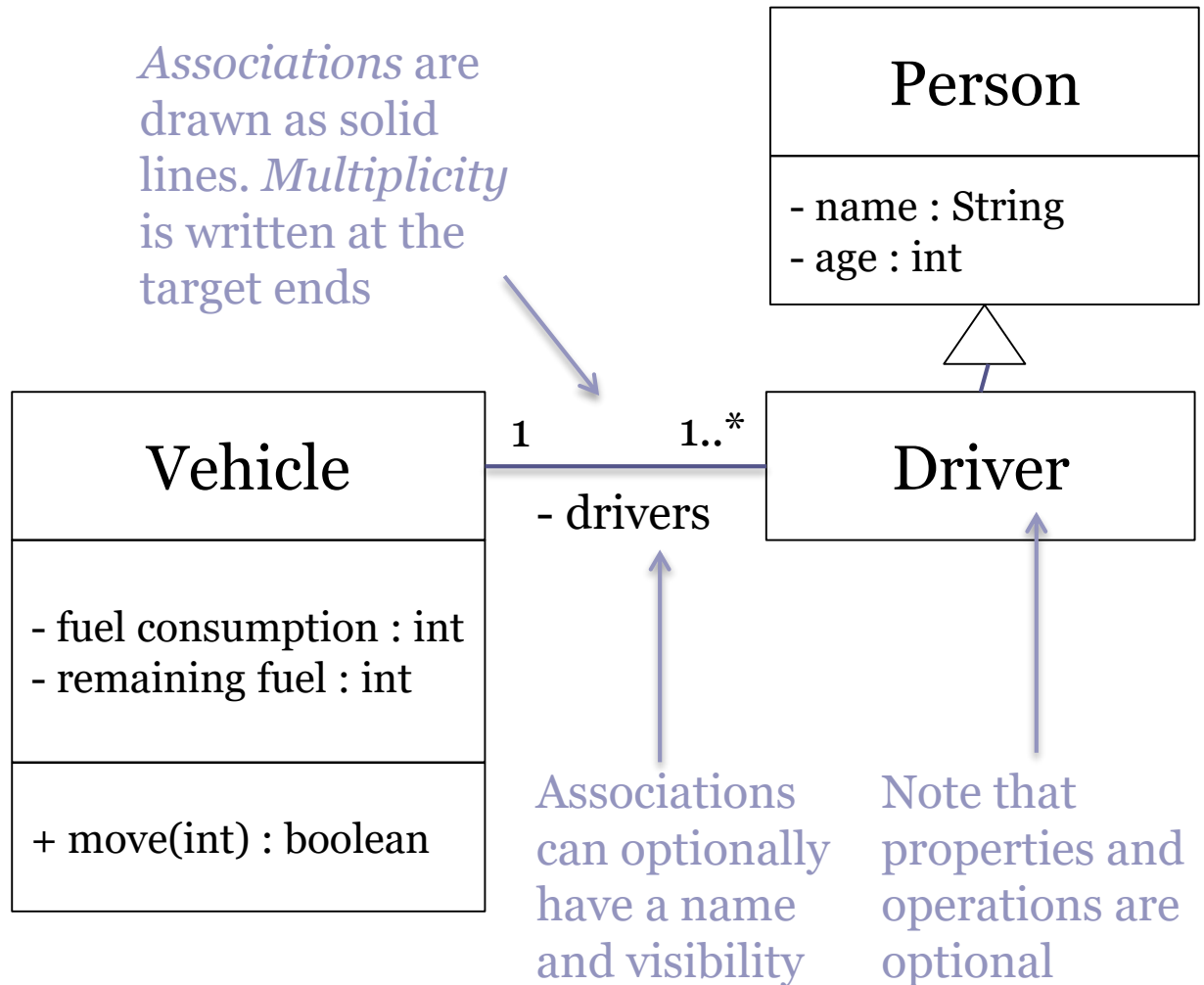
Two Way Associations

Vehicles
always have
at least one
driver. Each
driver must
have a single
vehicle.

Vehicle
- fuel consumption : int - remaining fuel : int
+ move(int) : boolean

Two way Associations

Vehicles always have at least one driver. Each driver must have a single vehicle.



Two-way Association(Bidirectional)

- We can navigate in both directions
- Denoted by a line between the associated objects

Employee $\frac{\text{works-}}{* \text{ for } 1}$ Company

- Employee works for company
- Company employs employees

Two way Association-Contractor-Project

```
class Contractor
{
private:
string Name;
Project MyProject;
...
};
```

```
class Project
{
string Name;
Contractor person;
....
};
```

Bidirectional Association



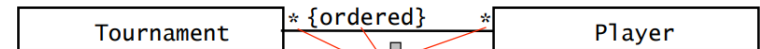
Source code after transformation:

```

public class Advertiser {
    private Set accounts;
    public Advertiser() {
        accounts = new HashSet();
    }
    public void addAccount(Account a) {
        accounts.add(a);
        a.setOwner(this);
    }
    public void removeAccount(Account a) {
        accounts.remove(a);
        a.setOwner(null);
    }
}

public class Account {
    private Advertiser owner;
    public void setOwner(Advertiser newOwner) {
        if (owner != newOwner) {
            Advertiser old = owner;
            owner = newOwner;
            if (newOwner != null)
                newOwner.addAccount(this);
            if (oldOwner != null)
                old.removeAccount(this);
        }
    }
}
  
```

One to many



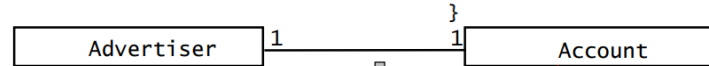
Source code after transformation

```

public class Tournament {
    private List players;
    public Tournament() {
        players = new ArrayList();
    }
    public void addPlayer(Player p) {
        if (!players.contains(p)) {
            players.add(p);
            p.addTournament(this);
        }
    }
}

public class Player {
    private List tournaments;
    public Player() {
        tournaments = new ArrayList();
    }
    public void addTournament(Tournament t) {
        if (!tournaments.contains(t)) {
            tournaments.add(t);
            t.addPlayer(this);
        }
    }
}
  
```

many to many



Source code after transformation:

```

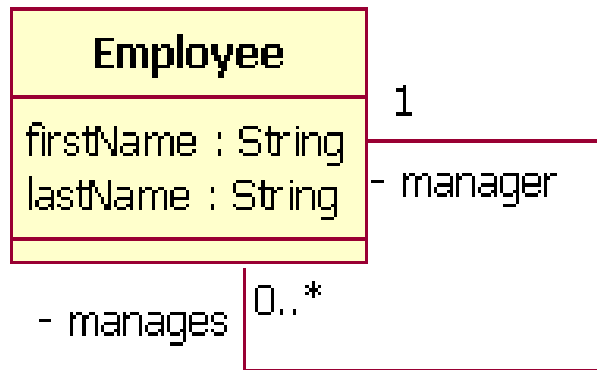
public class Advertiser {
    /* account is initialized
    * in the constructor and never
    * modified. */
    private Account account;
    public Advertiser() {
        account = new Account(this);
    }
    public Account getAccount() {
        return account;
    }
}

public class Account {
    /* owner is initialized
    * in the constructor and
    * never modified. */
    private Advertiser owner;
    public Account(owner:Advertiser) {
        this.owner = owner;
    }
    public Advertiser getOwner() {
        return owner;
    }
}
  
```

One to one

Self Association

A class can have *a self association/ reflexive Association*.



Two instances of the same class:
Pilot
Aviation engineer

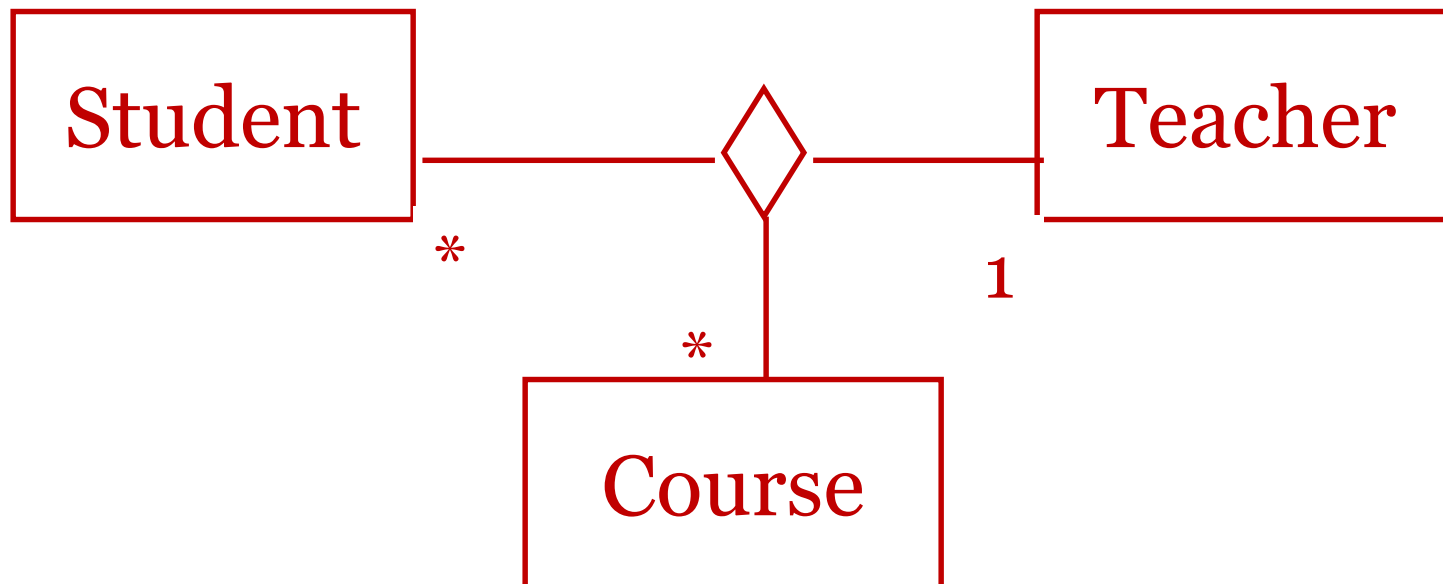
Self Association

```
class Course
{
private:
    std::string m_name;
    Course *m_prerequisite;

public:
    Course(std::string &name, Course *prerequisite=nullptr):
        m_name(name), m_prerequisite(prerequisite)
    {
    }
};
```

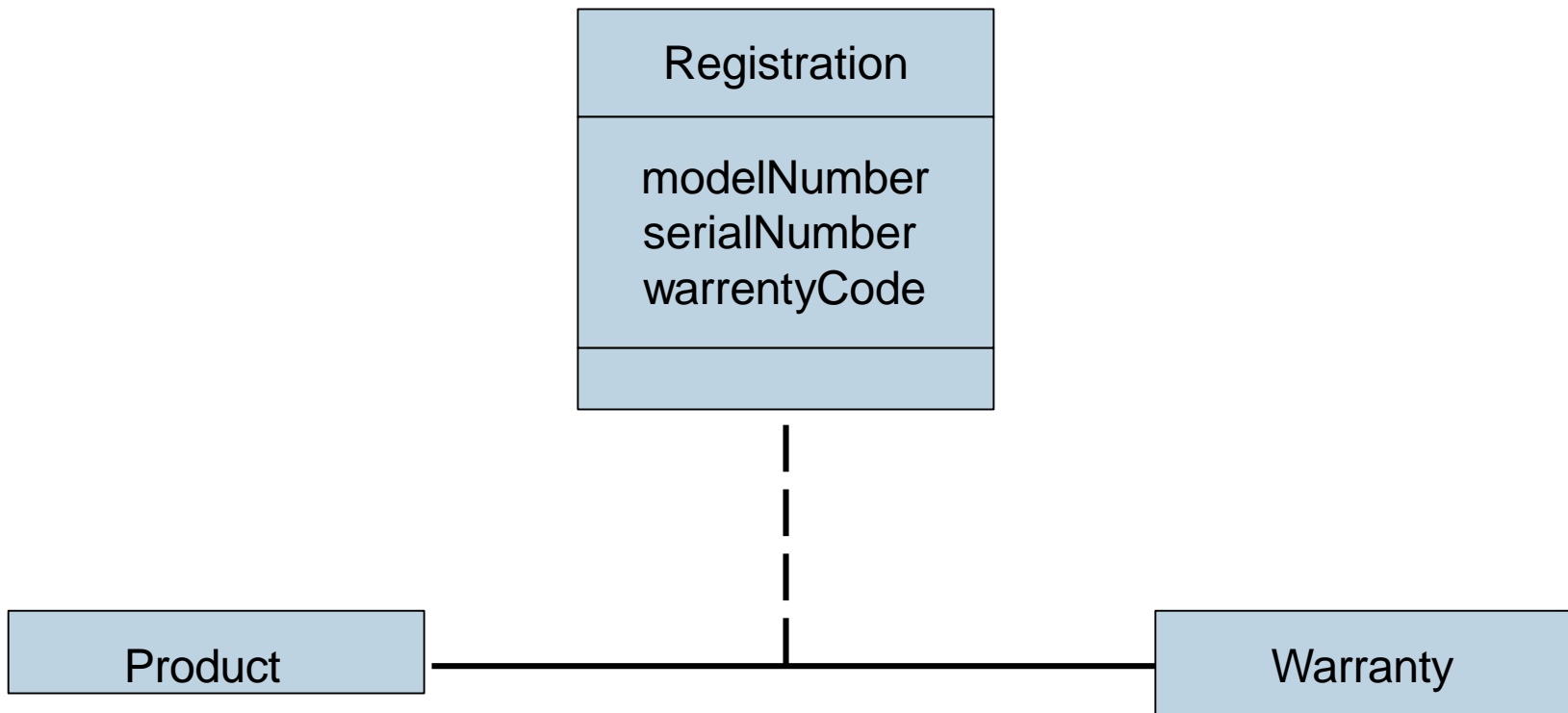

N-Ary Association

- Associates objects of more than two classes.
- Denoted by a diamond with lines connected to associated objects.



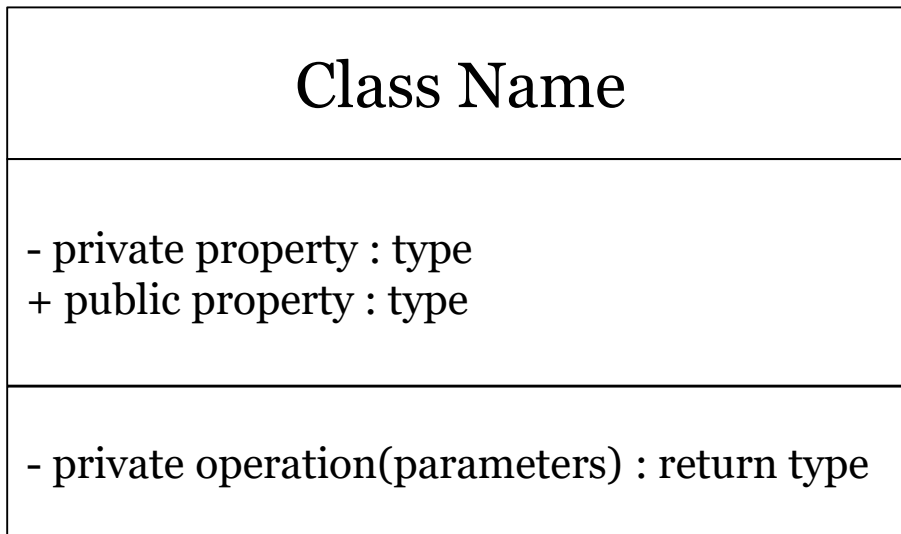
Association Class

- Associations can also be objects themselves, called link classes or an association classes.
- A link is an instance of an association.



UML Class Diagrams

Class



Association



Generalization (Inheritance)



Composition (strong)

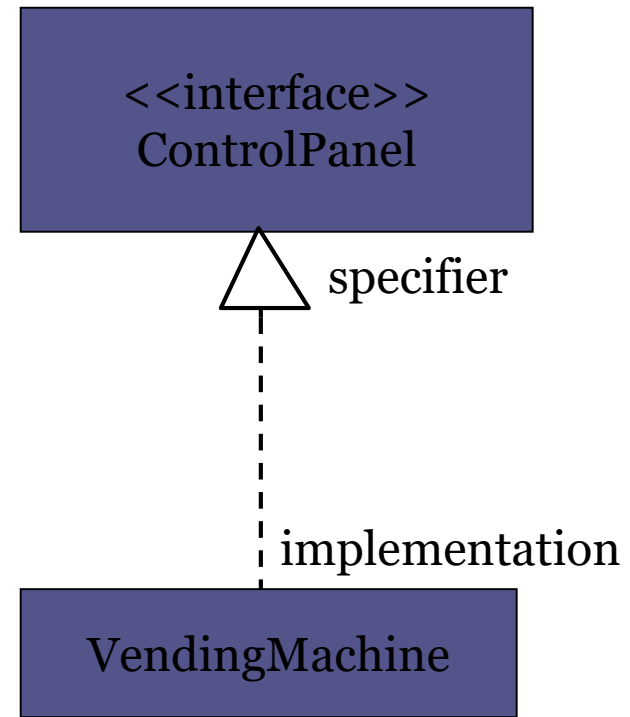


Aggregation (weak)



Interface Realization Relationship

A *realization* relationship connects a class with an interface that supplies its behavioral specification. It is rendered by a dashed line with a hollow triangle towards the specifier.

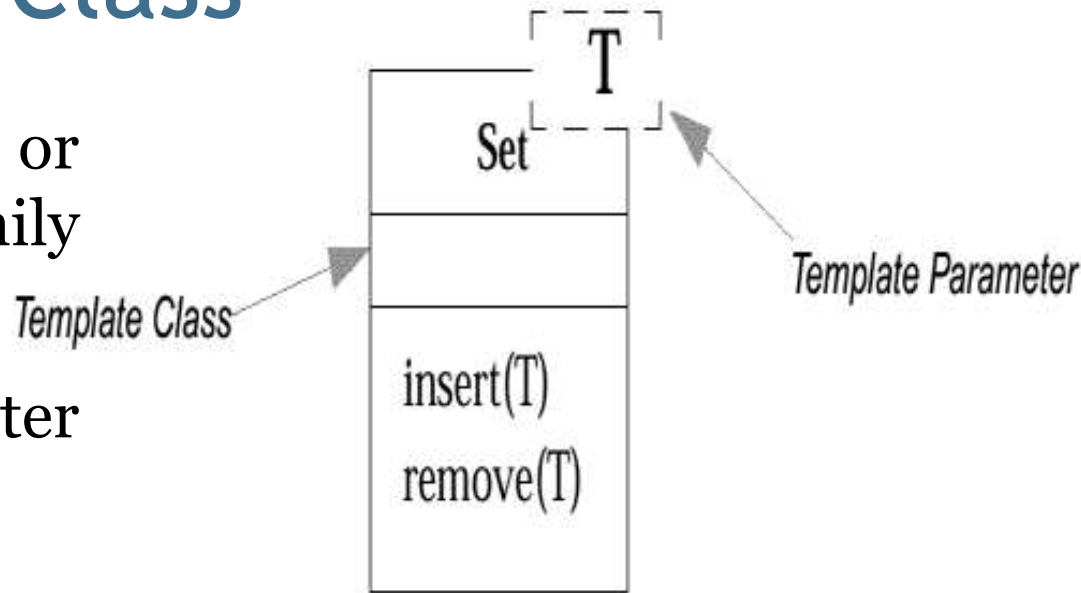


```
public interface A {  
    } // interface A  
public class B implements A {  
    } // class B
```

Parameterized Class

A *parameterized class* or *template* defines a family of potential elements.

To use it, the parameter must be bound.



```
class Set <T> {  
    void insert (T newElement);  
    void remove (T anElement);  
}
```

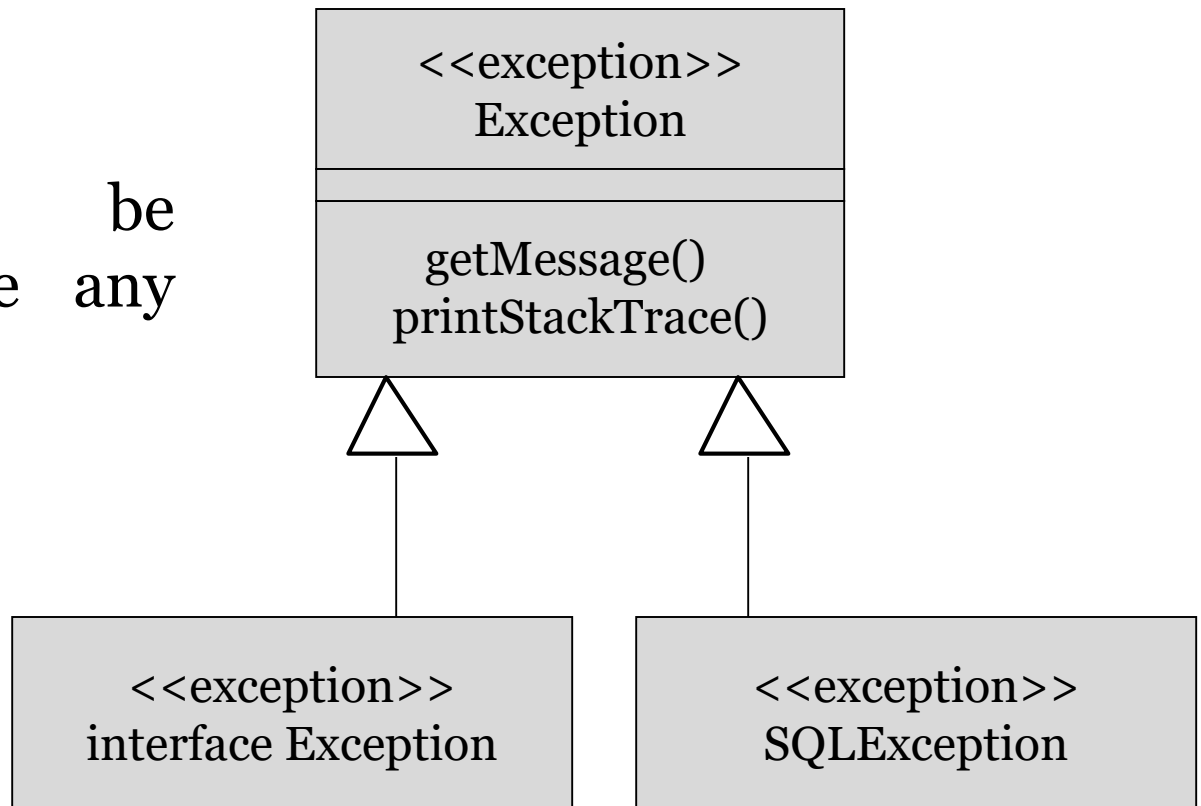
Enumeration

An *enumeration* is a user-defined data type that consists of a name and an ordered list of enumeration literals.

<<enumeration>> Boolean
false true

Exceptions

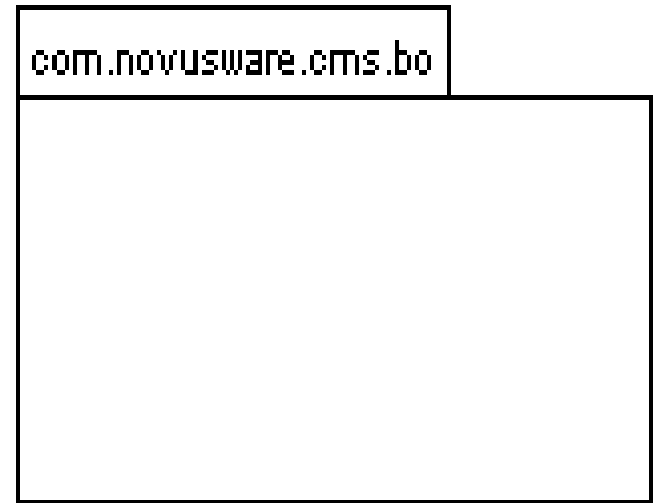
Exceptions can be modeled just like any other class.



Package

provides the ability to group together classes and/or interfaces that are either similar in nature or related.

Grouping these design elements in a package element provides for better readability of class diagrams, especially complex class diagrams.

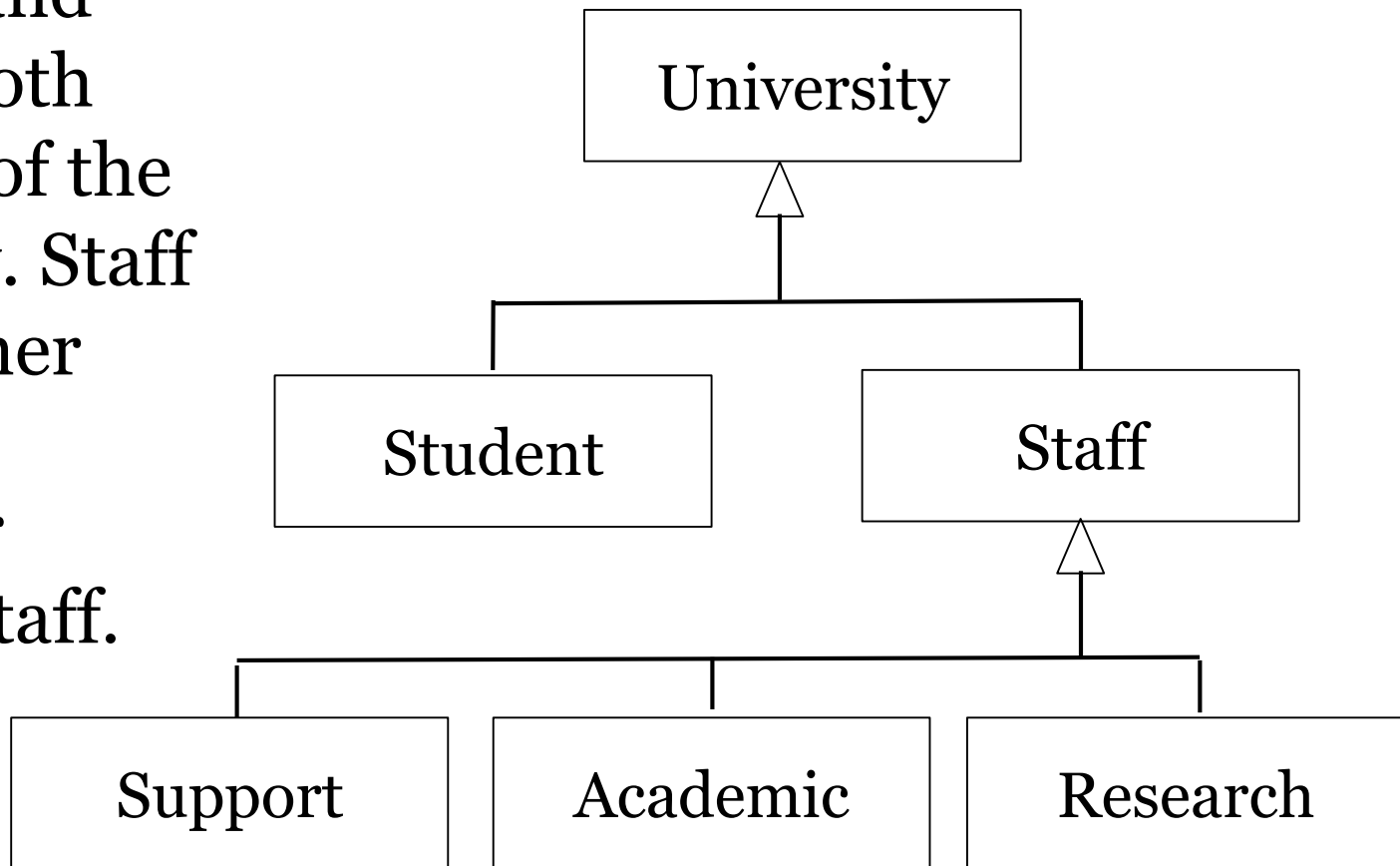


How might we model...

Students and staff are both members of the University. Staff can be either academic, support or research staff.

How might we model...

Students and staff are both members of the University. Staff can be either academic, support or research staff.

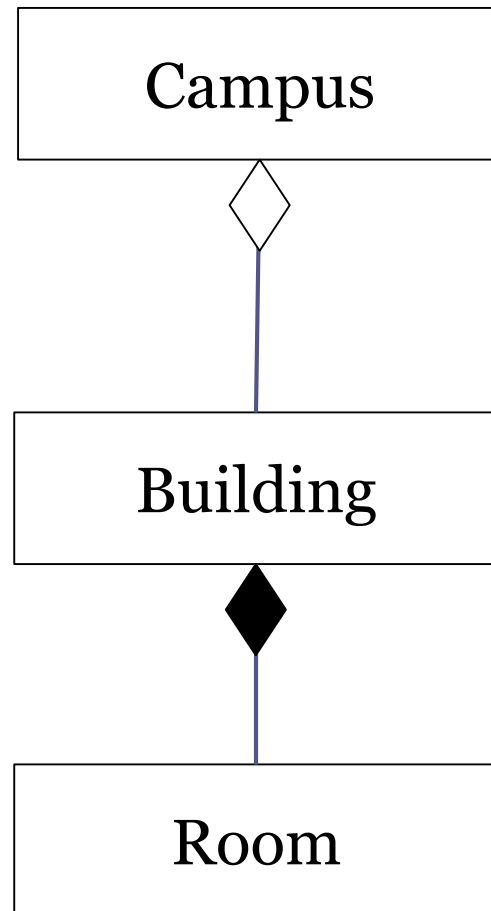


How might we model...

A campus is
made up of
many
buildings. A
building is
made of many
rooms.

How might we model...

A campus is made up of many buildings. A building is made of many rooms.

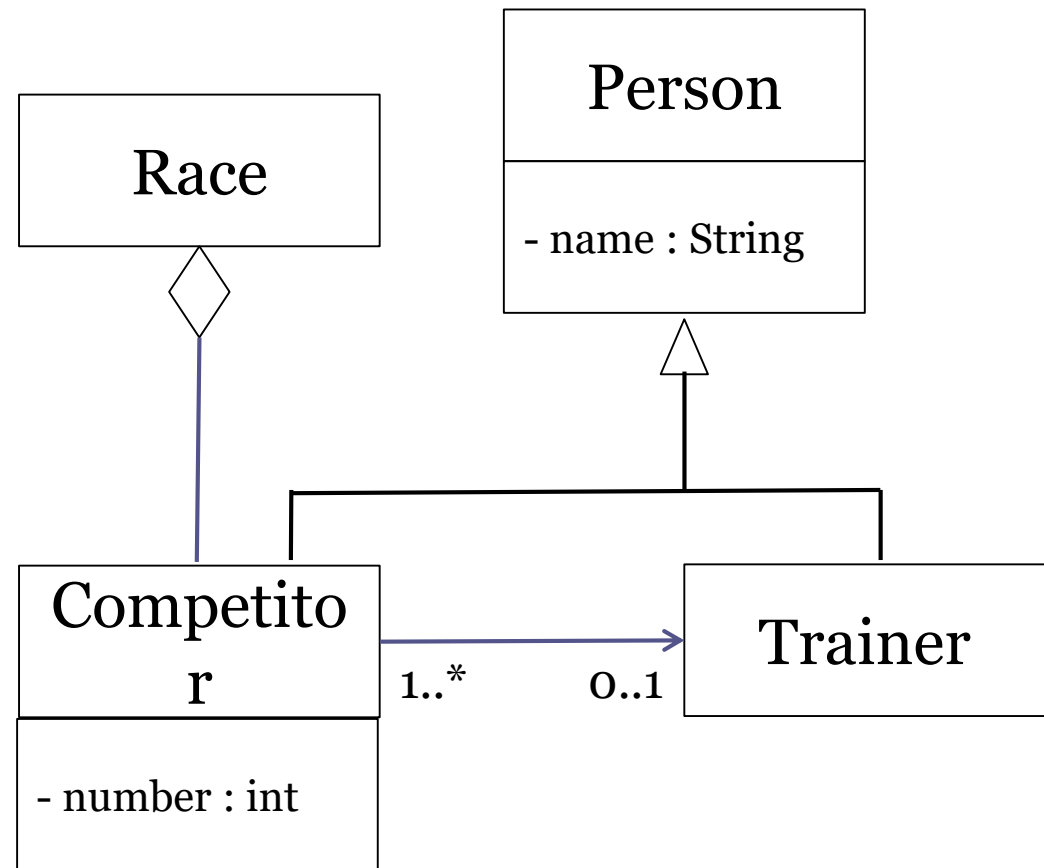


How might we model...

A Race has
many
competitors.
Each competitor
may have a
trainer. Both
types of person
have a name,
but competitors
also have a
number.

How might we model...

A Race has many competitors. Each competitor may have a trainer. Both types of person have a name, but competitors also have a number.



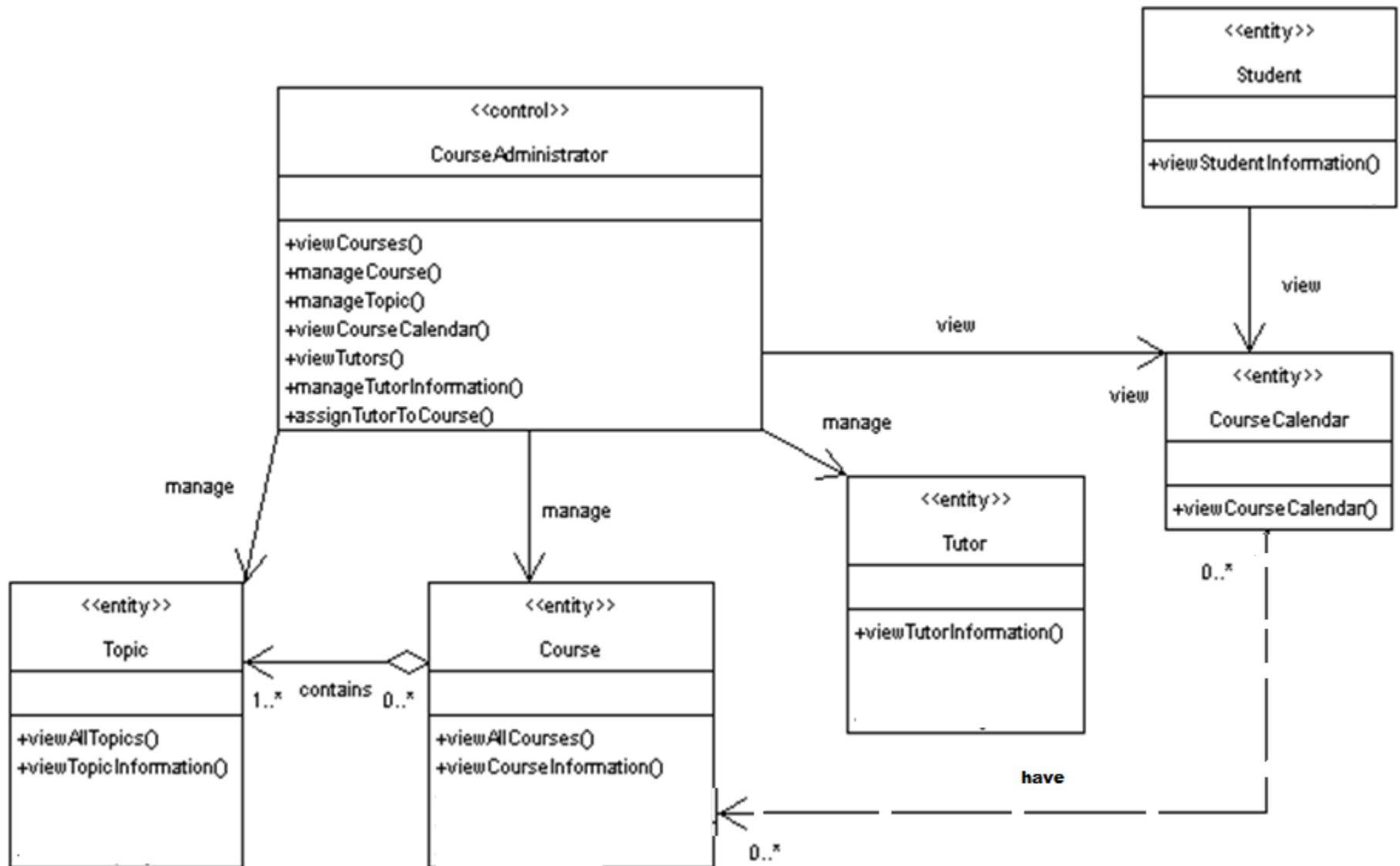
The UML Class Diagram in Action

Case study—Courseware Management System

- In the use case lecture, we identified the primary actors and use cases in the use case model of the case study.
- Let us recap the analysis that was performed when the use case model was designed.
- The following terms and entities specific to the system were identified from the problem statement:
 - Courses and Topics that make up a course
 - Tutors who teach courses
 - Course administrators who manage the assignment of the courses to tutors
 - Calendar or Course Schedule is generated as a result of the Students who refer to the Course schedule or Calendar to decide which courses for which they wish to sign up
 - Student can view his information

The UML Class Diagram in Action....

Identifying relationship and drawing the class diagram



Home Work Exercises

A series of horizontal lines in teal and light blue colors, some solid and some dashed, extending across the width of the slide below the title.

University Team Management

- In the OOAD course at Fast University, students are member of teams.
- Each team has 2 or 3 members.
- Each team completes 0 to 3 assignments.
- Each student takes exactly two midterm test.
- Computer Science students have a single account on Coding Development facility , while each engineering student has an account on the Engineering facility.
- Each assignment and midterm is assigned a mark.

University System

- A Fast university offers degrees to students.
- The university consists of faculties each of which consists of one or more departments.
- Each degree is administered by a single department.
- Each student is studying towards a single degree.
- Each degree requires one to 20 courses.
- A student enrolls in 1-5 courses (per term).
- A course can be either graduate or undergraduate, but not both.
- Likewise, students are graduates or undergraduates but not both.

Library System

- *This application will support the operations of a technical library for an R&D organization. This includes the searching for and lending of technical library materials, including books, videos, and technical journals. Users will enter their company ids in order to use the system; and they will enter material ID numbers when checking out and returning items. Each borrower can be lent up to five items. Each type of library item can be lent for a different period of time (books 4 weeks, journals 2 weeks, videos 1 week). If returned after their due date, the library user's organization will be charged a fine, based on the type of item(books \$1/day, journals \$3/day, videos \$5/day).Materials will be lent to employees with no overdue lendable, fewer than five articles out, and total fines less than \$100.*

Draw the UML class diagram showing the domain model for online shopping. The purpose of the diagram is to introduce some common terms, "dictionary" for online shopping - Customer, Web User, Account, Shopping Cart, Product, Order, Payment, etc. and relationships between. It could be used as a common ground between business analysts and software developers.

- Each customer has unique id and is linked to exactly one account. Account owns shopping cart and orders. Customer could register as a web user to be able to buy items online. Customer is not required to be a web user because purchases could also be made by phone or by ordering from catalogues. Web user has login name which also serves as unique id. Web user could be in several states - new, active, temporary blocked, or banned, and be linked to a shopping cart. Shopping cart belongs to account.
- Account owns customer orders. Customer may have no orders. Customer orders are sorted and unique. Each order could refer to several payments, possibly none. Every payment has unique id and is related to exactly one account.
- Each order has current order status (new, hold, shipped, delivered, closed). Both order and shopping cart have line items linked to a specific product. Each line item is related to exactly one product. A product could be associated to many line items or no item at all.

Car Rental System

We want to develop a car rental system. When any customer/user wants to rent a car, he/she must be registered with the system. The system must also create the list of cars according to their category (i.e. Gomini, Go, Goplus). When a user requests booking of a car of certain category, start day of rental is the current day or a day after the current day; end day of rental lies after the start day. Customer can also cancel his or her booking. When the customer requests a pickup, a suitable car must be found among the currently available cars. When the trip is completed, booking closes and car must be returned.

Banking System Application

- We have to develop a banking system application which provides many services to the customers like opening and closing accounts, balance enquiry, deposit money, cash withdrawal, and taking cards. Customer can open two types of accounts i.e. saving and current account. Bank also has an ATM machine which provides the services related to balance. Customer can take loan from the bank against his/her account. One customer can take only one loan at a time.

Organization Scenario

- Consider the world of companies:
- Companies employ employees (who can only work for one company), and consist of one or more departments.
- Each company has a single president, who is an employee.
- Departments have employees as members and run projects (one or more.)
- Employees can work in 1 to 3 projects, while a project can have 2 to 50 assigned employees.
- You may assume that companies have a name and address, while employees have a emp# and a salary