UML Notation for

Class diagrams

Object diagrams

Class Diagram

A class diagram begins as a conceptual or analysis class model and evolves to a design class model

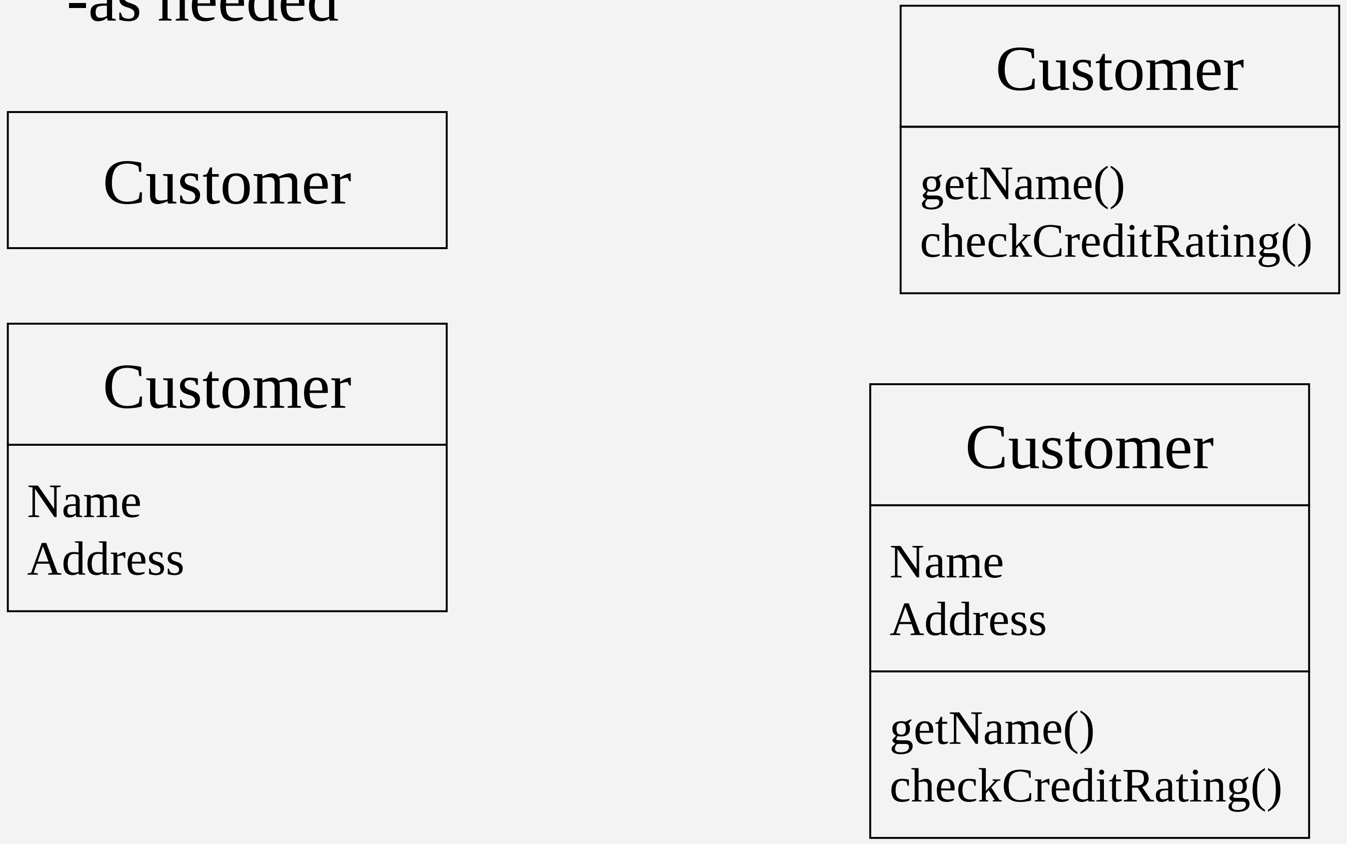
Used throughout the development process … More detail added as time goes by

A static view of classes – shows structure: data, operations, and associations

Classes

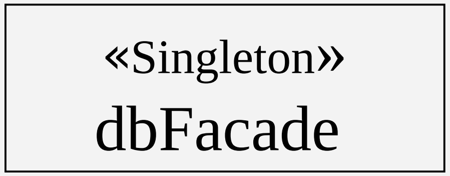
Represented by a rectangle with possibly 3 compartments

-as needed



Classes

Some classes are stereotyped:



Stereotyping dbFacade as singleton conveys substantial information about the class and its behaviour.

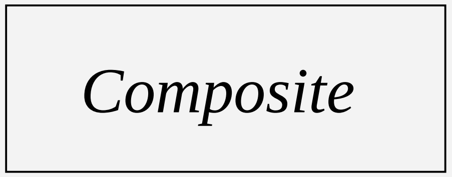
Some methodologies have 3 stereotypes for “analysis” classes: boundary, control, entity

-may have seen this in ACS-2913

Classes

Abstract Classes

An abstract class is one that is never instantiated (only concrete subclasses can be instantiated). To indicate an abstract class, the class name is given in italics or by using an “abstract” stereotype.

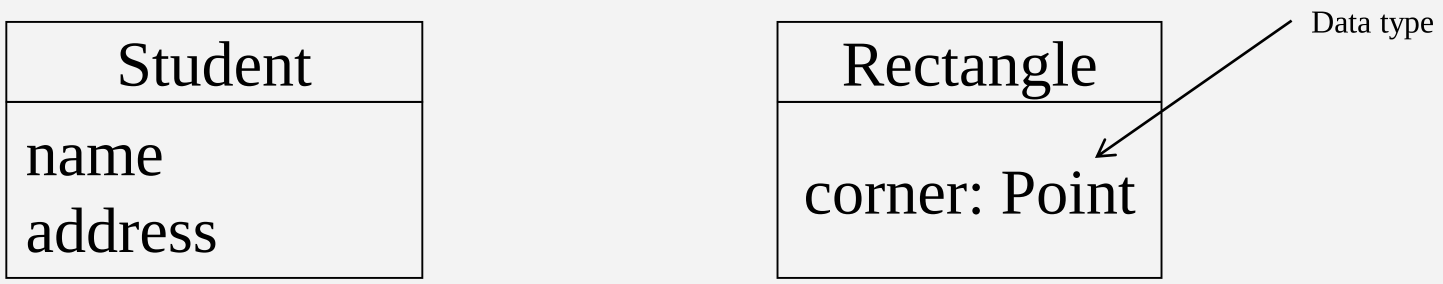


Attributes

an object contains data – data fields are defined as part of the Class definition

examples:

- Students have names, addresses, etc;
- Courses have titles, descriptions, prerequisite information.

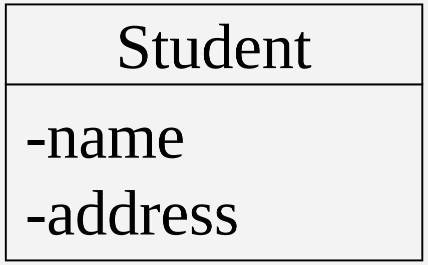


Level of detail present will depend on whether you are in analysis or design, and your purposes at the time

In ACS-3913 we won’t be showing data types

Attributes

To what degree is an attribute visible to other classes?
Private –
Public +
Protected #
Package ~



In ACS-3913 we won’t be showing attribute visibility in class diagrams

Attributes

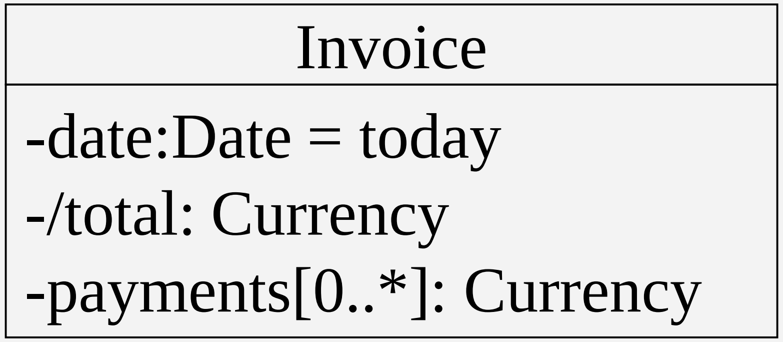
Default values =

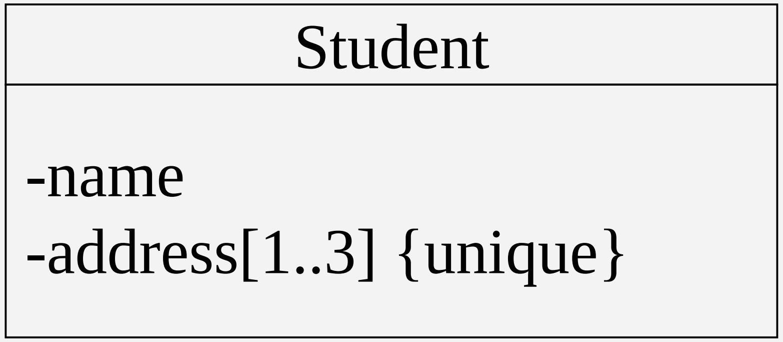
Derived values /

Multiplicity [ ]

Ordering {ordered}

Uniqueness {unique}



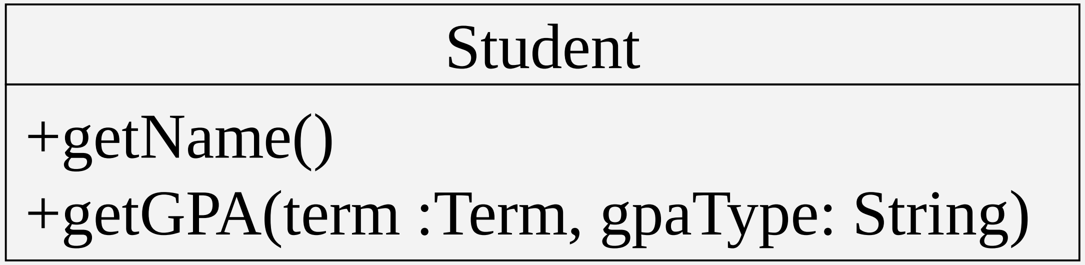


Operations.

What are the responsibilities of a class? What can it do? What are the methods?

Parameters

Signature the name, parameters, and return type of the operation



In ACS-3913 we will normally show the names of methods

Associations

- correspond to verbs expressing a relationship between classes

- example

  a Library Member *borrows* a Copy of a Book

- Multiplicities
  - we indicate via multiplicities the range of allowable cardinalities for participation in an association
  - examples:

    1

    1..*

    0..*

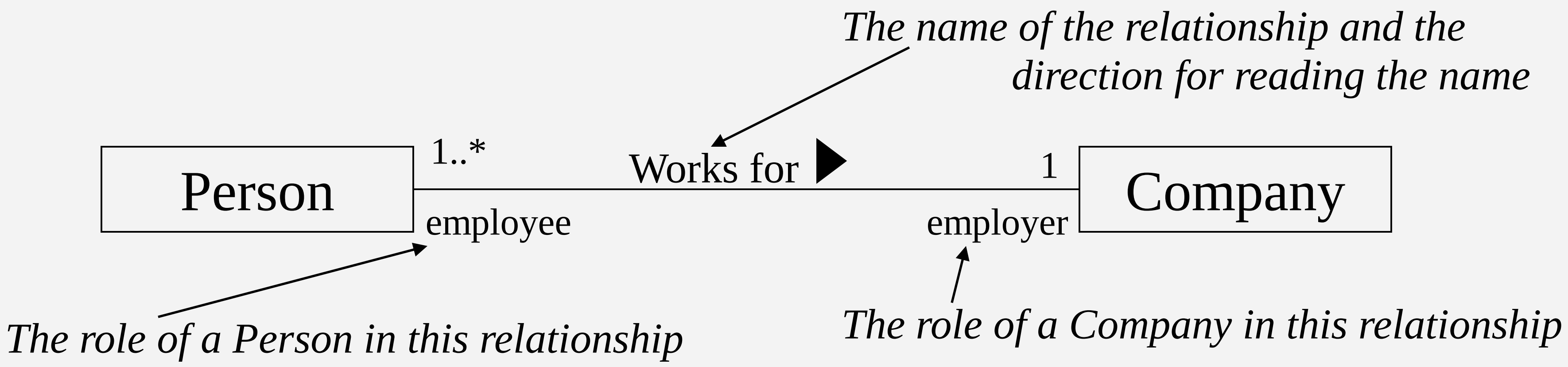
    *

    1..3

In ACS-3913 multiplicities and associations are very important to us

Associations

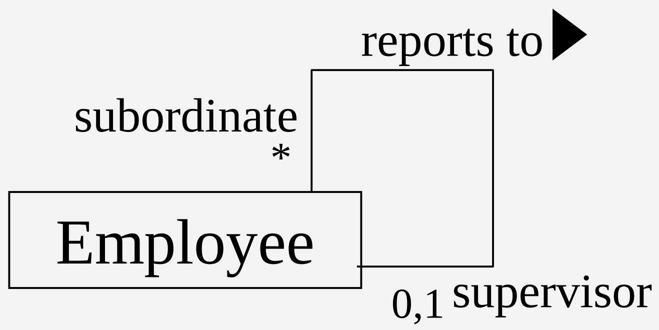
- Names and roles
  - you can name the relationship and indicate how to *read* it
  - you can give role names for participating objects


The name of the relationship and the direction for reading the name
Person
1..*
Works for ▶
1
Company
employee
employer
The role of a Person in this relationship
The role of a Company in this relationship

Associations

• example:

An employee reports to another employee (his/her supervisor)

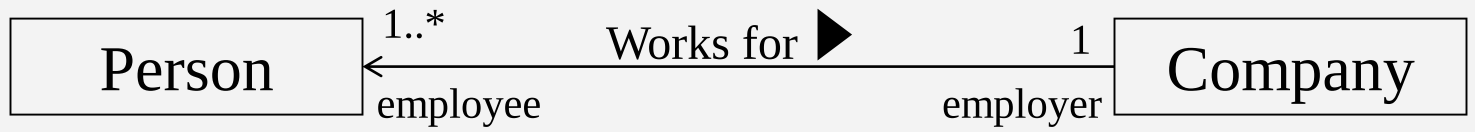


A ***reflexive*** association

Role names are very important for reflexive/recursive associations

Associations and Navigability

Navigability is an adornment (an arrowhead) to an association endpoint to indicate that an instance can be reached from an instance at other end.



An instance of Company can send a message to an instance of Person.

This implies that company holds references to its persons (e.g. an array list of persons).

Whether or not a Person can send a message to a Company is not specified above.

When possible we show this in our ACS-3913 class diagrams

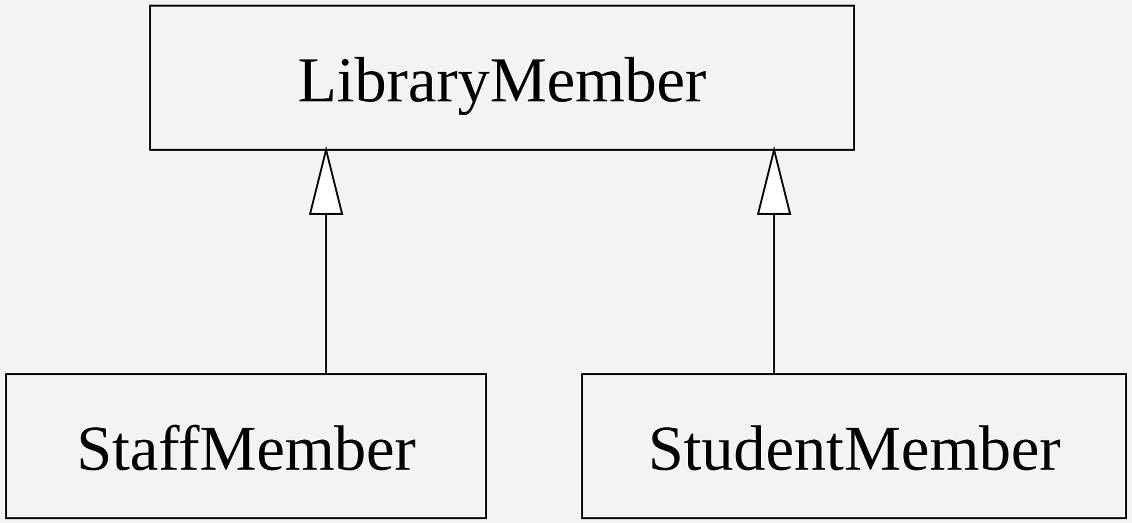
Generalization

a generalization is a relationship between a general thing (the superclass or parent class) and a more specific thing (the subclass or child class)

example:

a StaffMember is a specialized kind of LibraryMember

a StudentMember is a specialized kind of LibraryMember



In ACS-3913 generalization hierarchies appear very often in our class diagrams

Motivation for partitioning a class into subclasses:

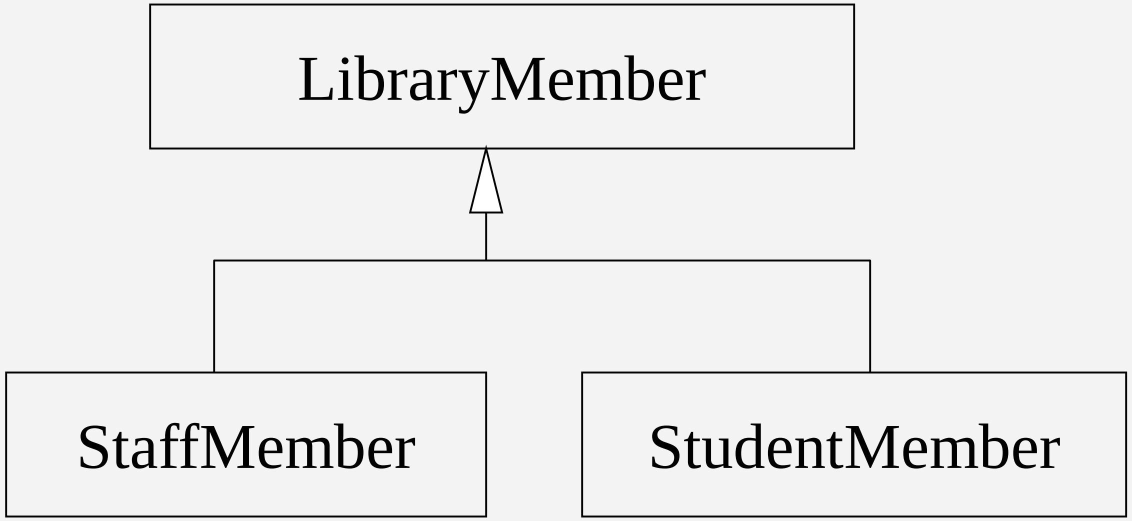
•subclass has additional attributes of interest

•subclass has additional associations of interest

•subclass is operated on, handled, reacted to, or manipulated differently than the superclass or other subclasses

Generalization

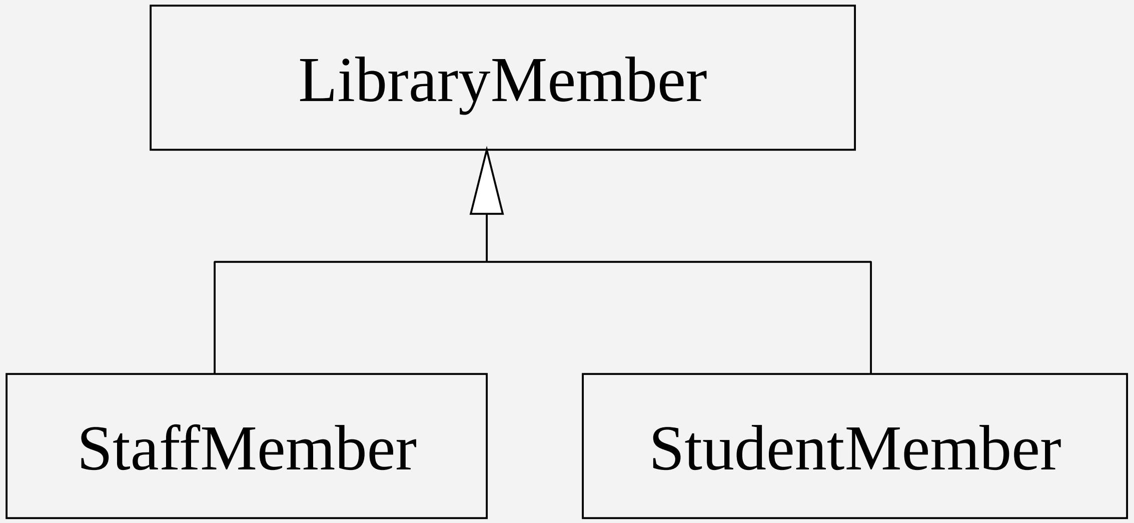
Multiple subclasses can be grouped to indicate they are related
Subclasses – very useful



It then becomes meaningful to consider certain constraints:

complete, incomplete, disjoint, overlapping

Generalization



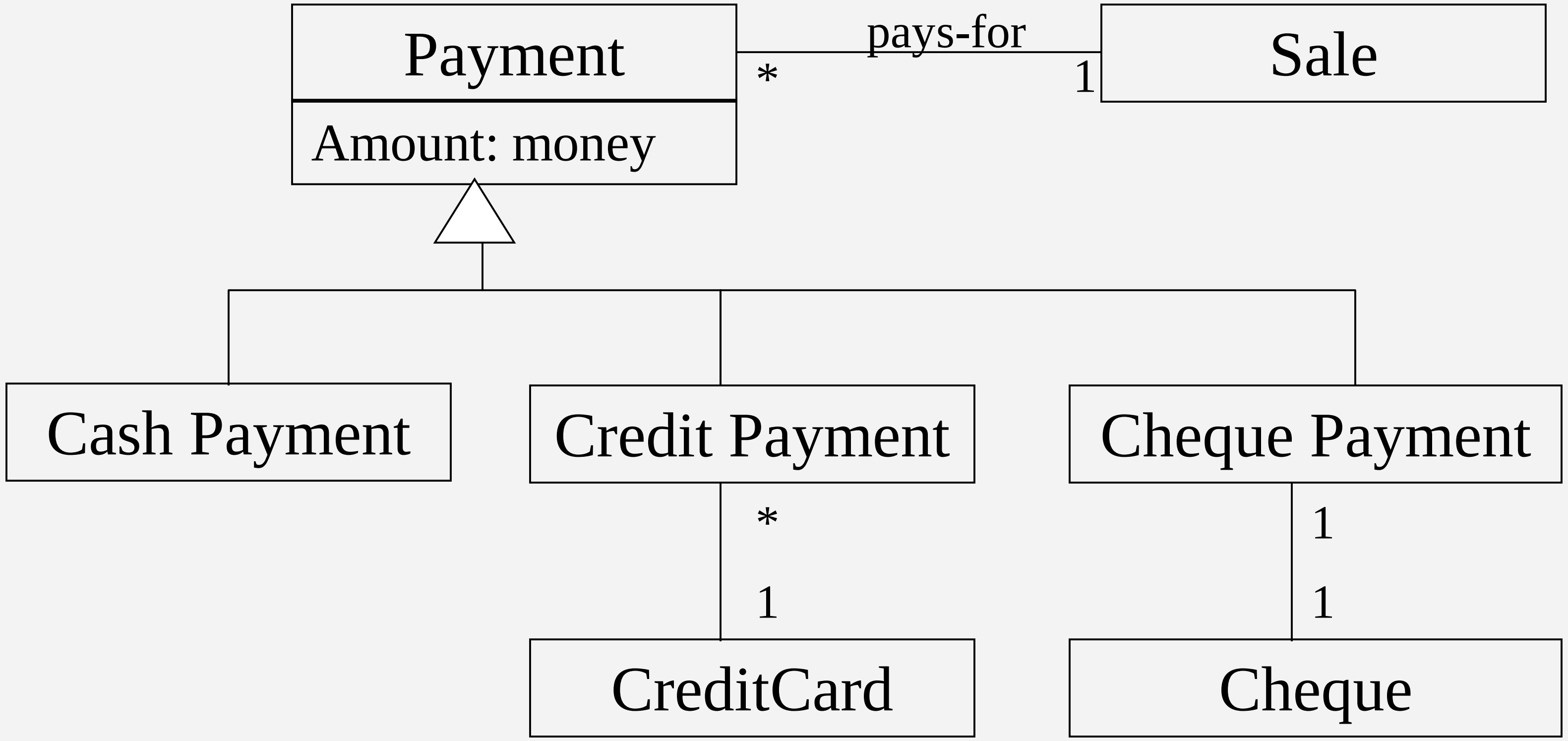
Inheritance of attributes and behaviour:

- everything a LibraryMember can do, a StudentMember can do
    - If a LibraryMember can borrow a book, so can a StaffMember and a StudentMember
- a StaffMember and a StaffMember have all the attributes the LibraryMember has, and possibly more

Specialization: there are some things that a specialized class can do that a LibraryMember cannot

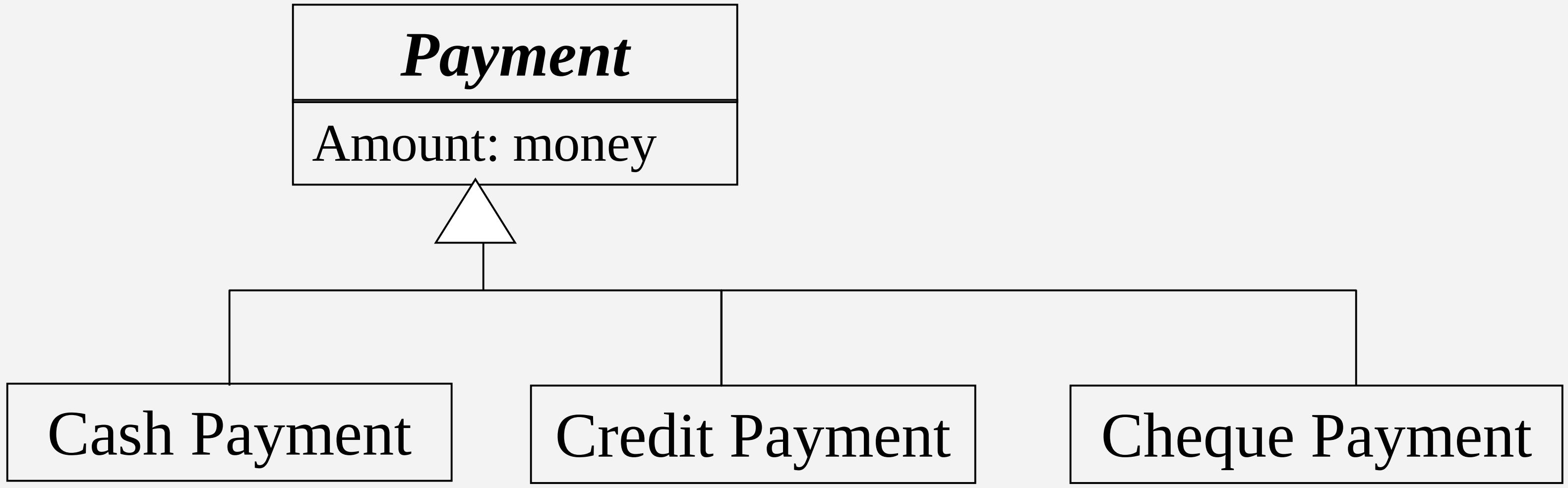
Example.

Every payment, regardless of whether it is cash, credit, or cheque, has an Amount and it is associated with a Sale



Only credit payments are associated with credit cards

Only cheque payments are associated with cheques

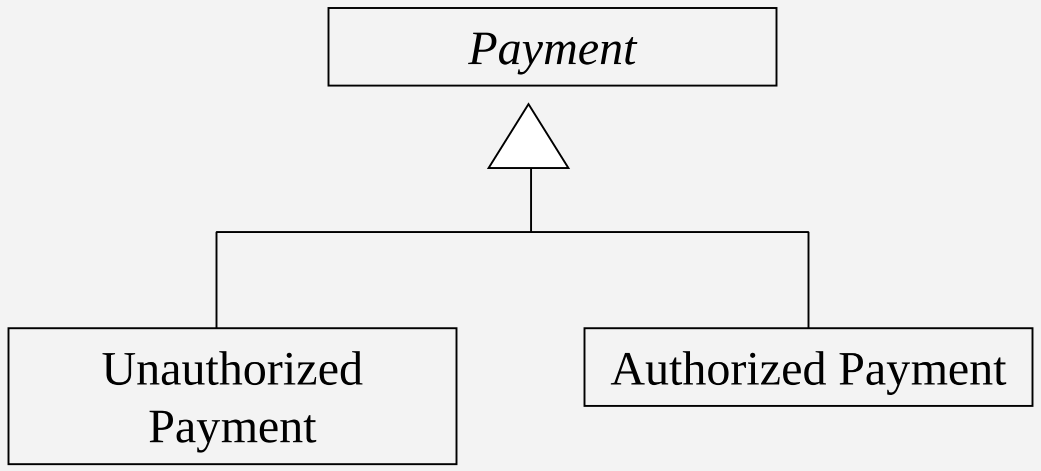


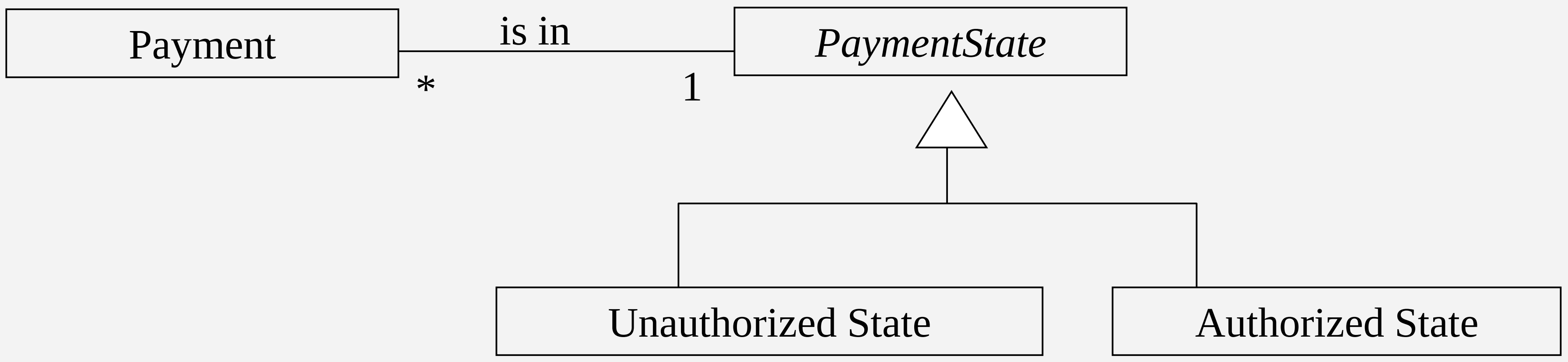
The name Payment is italicized - meaning it is an **abstract** class

An abstract class is a class that will never be instantiated; only its subclasses can exist

If "Payment" was not in italics then a Payment could exist that is not a Cash, Credit, or Check payment (see previous slide)

What differences are there in the two hierarchies below:



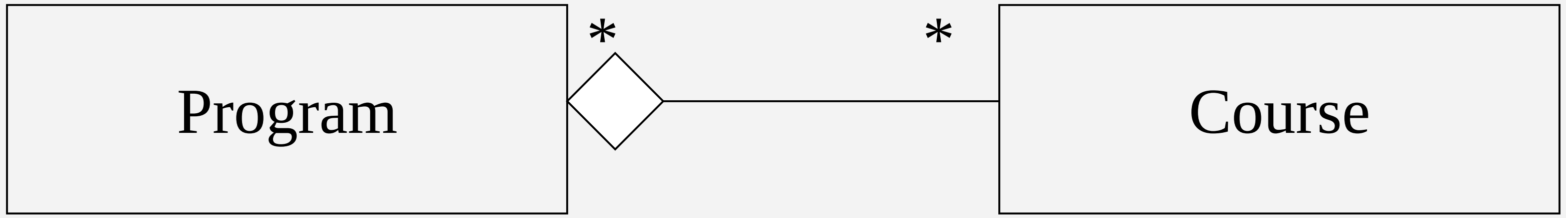


Does one of the above use composition?
At runtime, how many objects exist?
What is required to change a payment from unauthorized to authorized?

Aggregation and Composition

both are associations used to denote that an object from one class is part of an object of another class

An example of **Aggregation**: a course is part of a program. The same course could be part of several programs

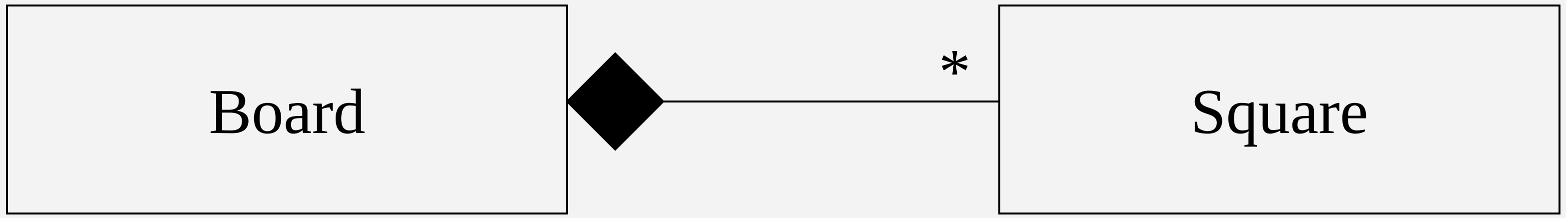


Suppose a course “Data structures and algorithms” is part of both the “Applied CS” and the “Scientific computing” programs

Deleting a program should not result in a course being deleted

Aggregation and Composition

Composition is similar to, but stronger than aggregation. If you specify composition, then you are saying that one object *owns* its parts.



A Board is made up of several Squares. A Square will belong to just one Board.

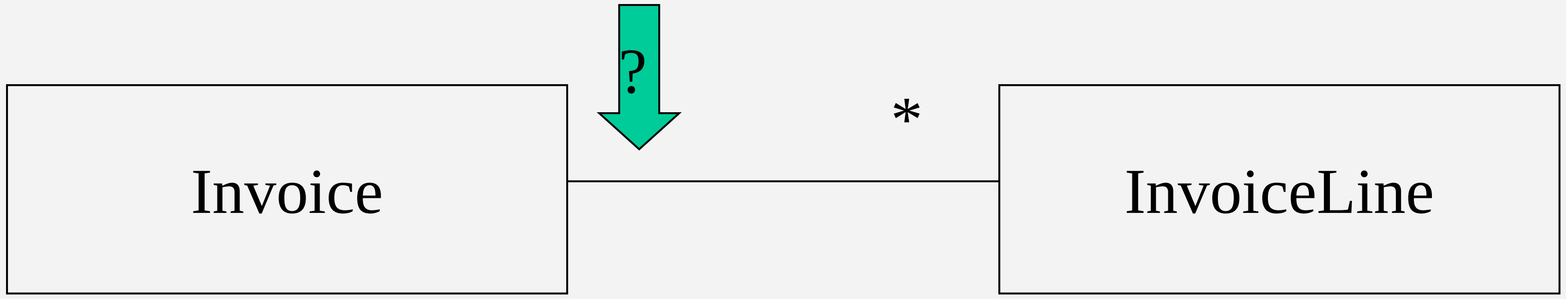
If a Board is deleted, then its Squares are deleted too.

What is the multiplicity at the composition end of the association?

Aggregation and Composition

Consider Invoices and their Invoice Lines

Question: Is the association aggregation or composition? ◆ ? ◇



In ACS-3913 we won't be concerned with showing ◆ or ◇

However, we do need to show multiplicities

Composite Pattern

A composite is a group of objects in which some objects contain others; one object may represent groups, and another may represent an individual item, a leaf.

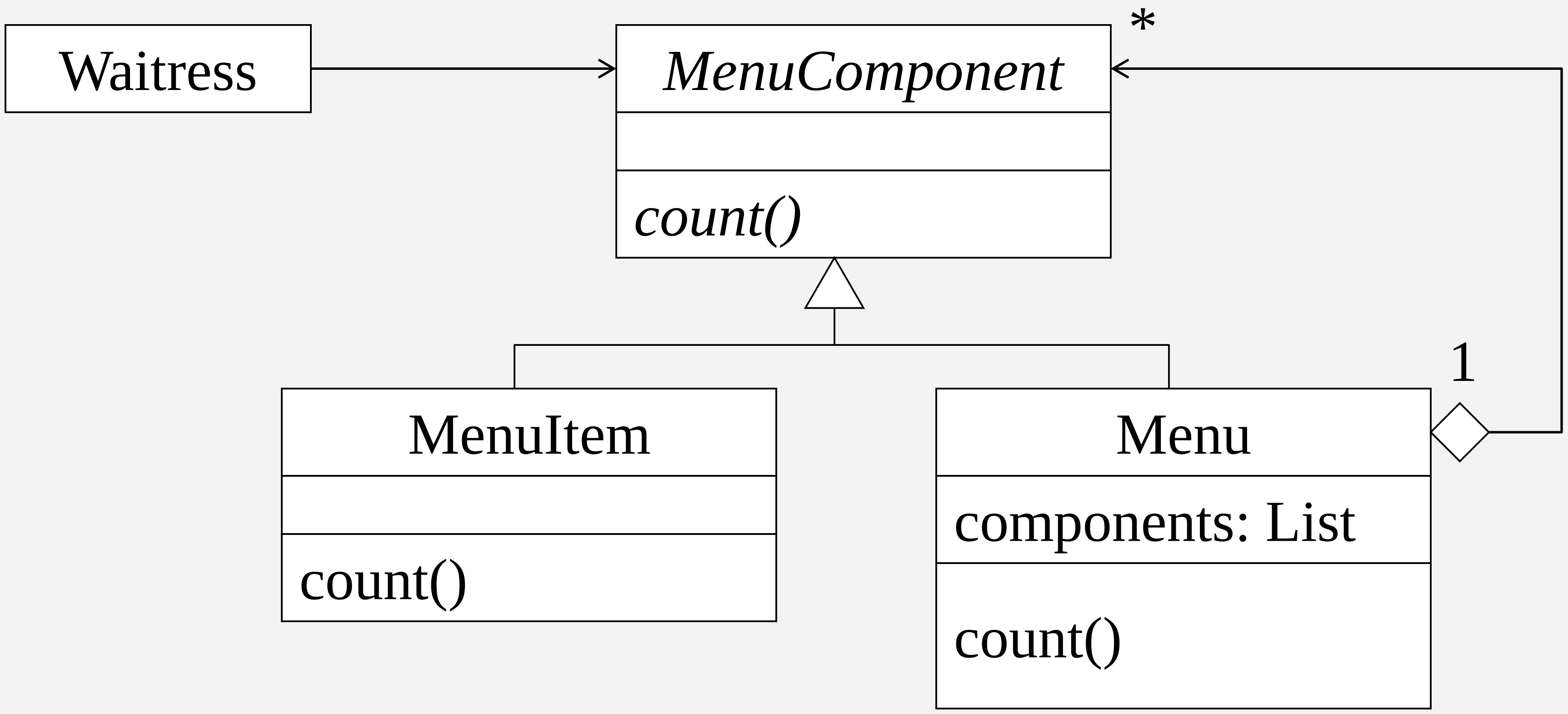
We will examine the composite pattern later in the course. At this time, we are many concerned with its structural aspect.

Consider the class diagram that follows.

> What objects does it allow us to instantiate and how will they relate to one another?
>
> What is this data structure?
>
> What does an instance look like?

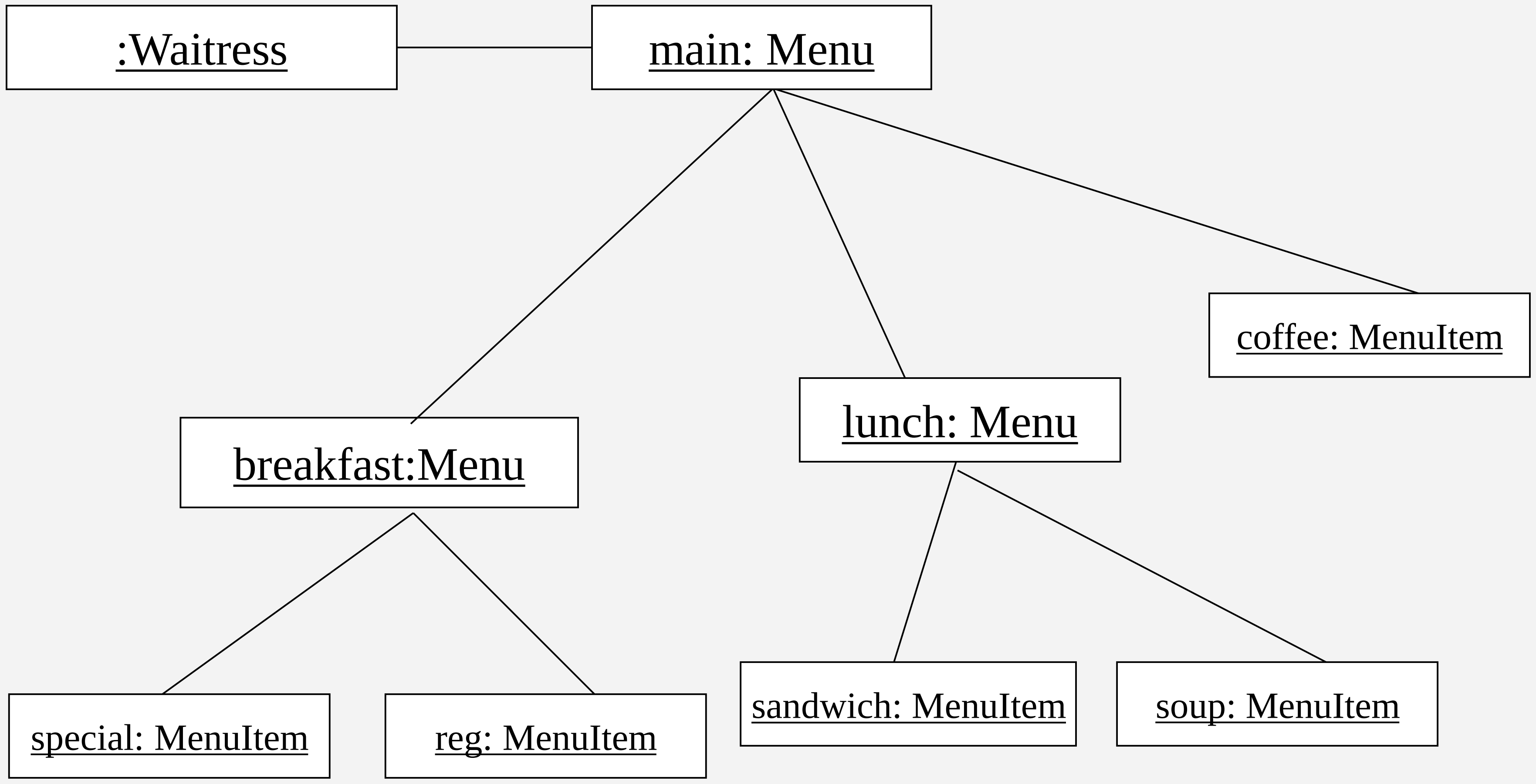
Composite Pattern

Consider a UML class diagram for menus. Menus may be complex and contain other menus.



Composite Pattern

An object diagram illustrating a tree
… a possible main menu comprising several connected objects



Decorator Pattern

The decorator pattern allows us to enclose an object inside another object. The enclosing object is called a decorator. The other object is the component, it is the decorated object.

The decorator conforms to the interface of the enclosed component and so its presence is transparent to the components clients. The decorator forwards requests to the component, but may perform some processing before/after doing so.

We will examine the decorator pattern later in the course. Consider the class diagram that follows.

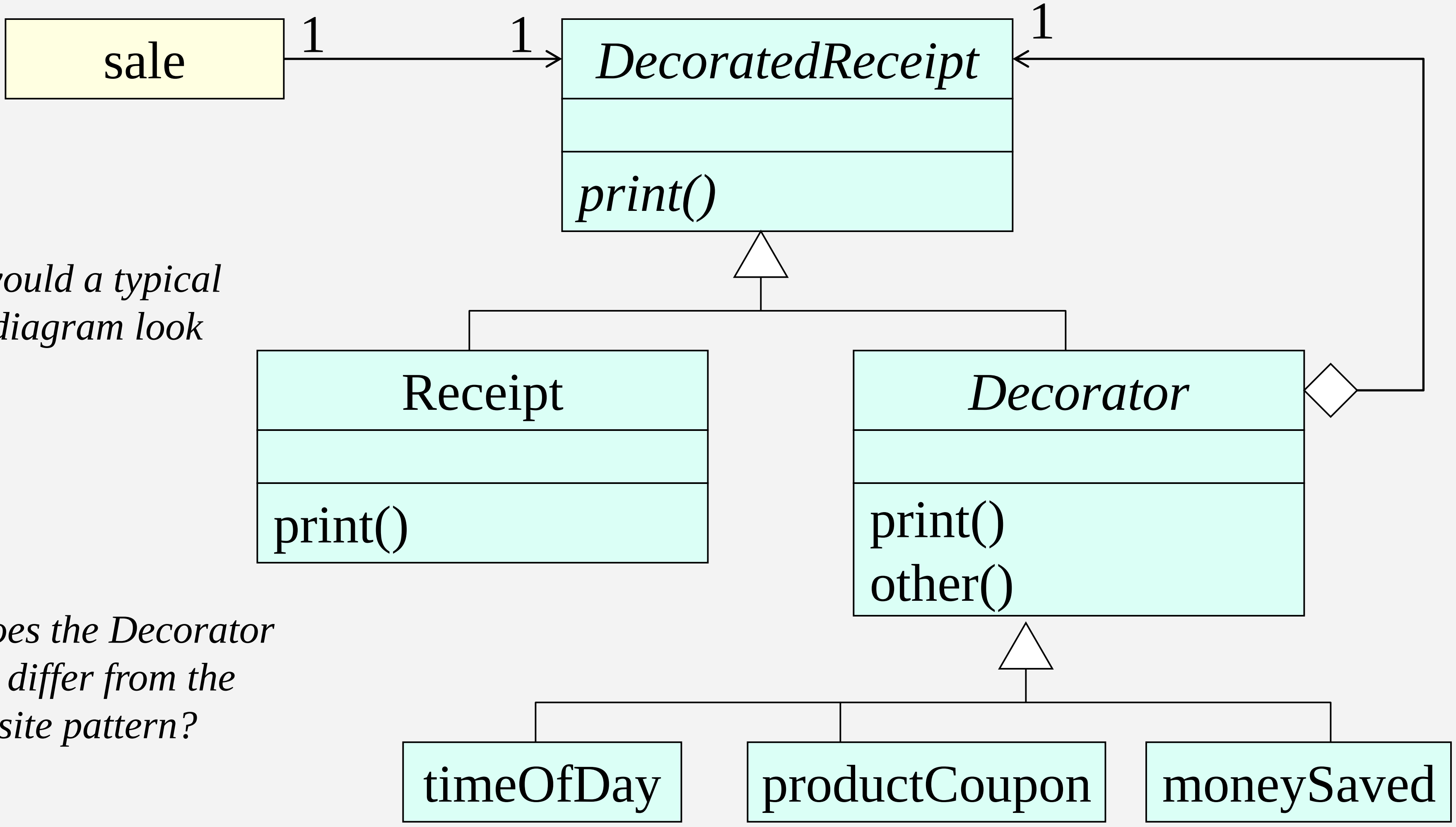
What objects does it allow us to instantiate and how will they relate to one another?

What is this data structure?

What does an instance look like?

Decorator Pattern

UML class diagram



What would a typical object diagram look like?

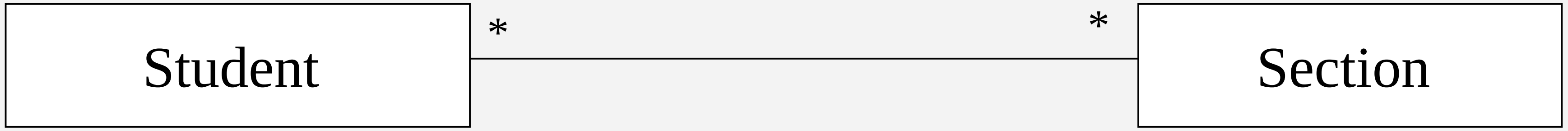
How does the Decorator pattern differ from the Composite pattern?

Class Diagram

Association classes

Used when there is data or behaviour related to the association

Useful for many-to-many associations

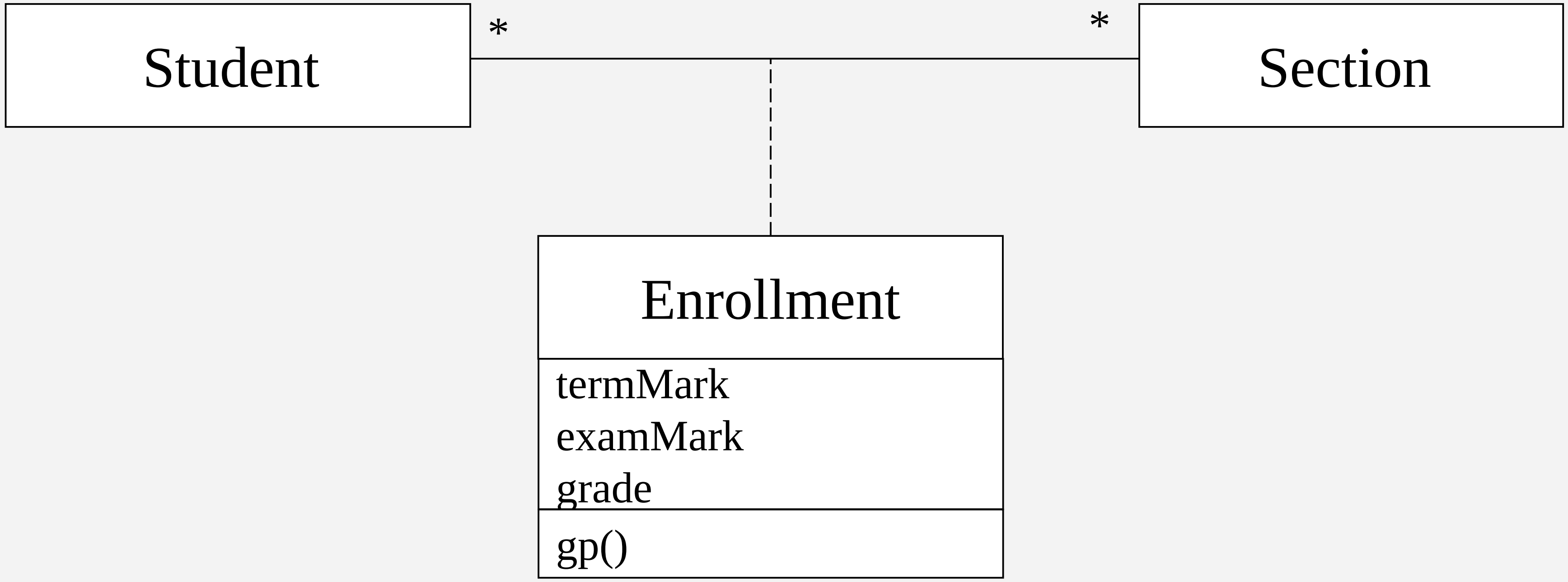


A student registers for a section and a section has many students registered

Class Diagram

Association classes

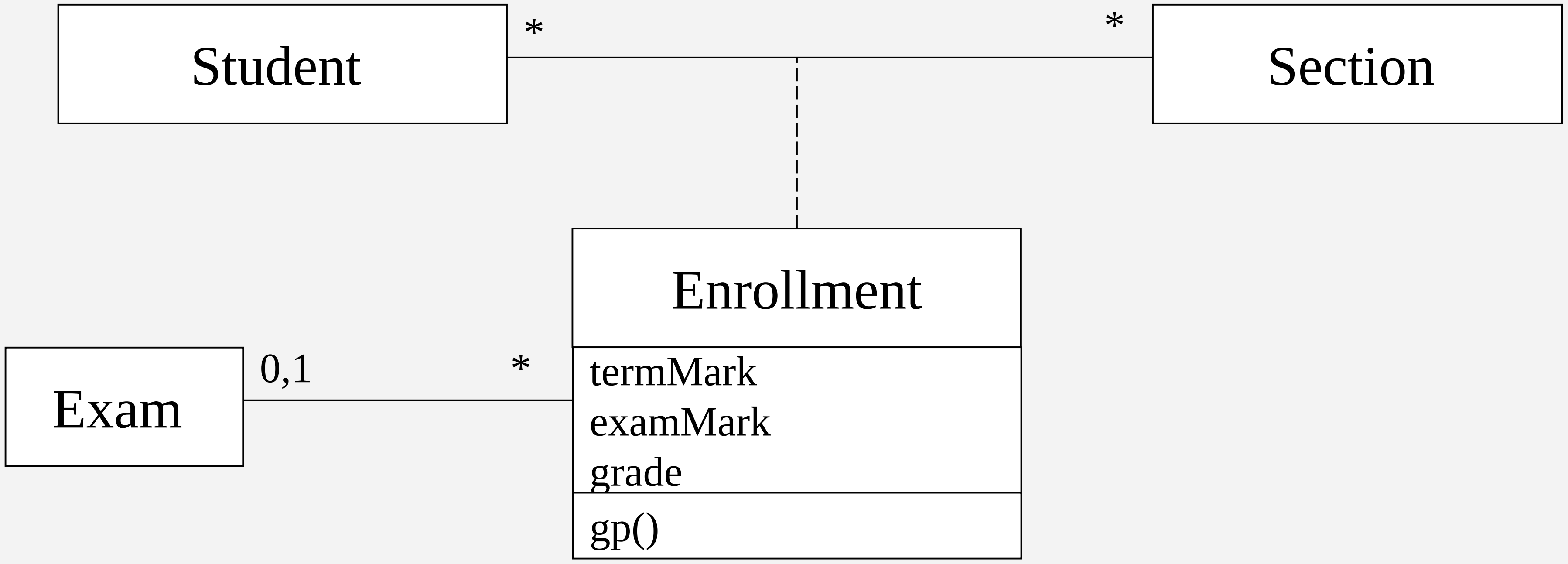
Suppose we need to store information that describes the association



Class Diagram

Association classes

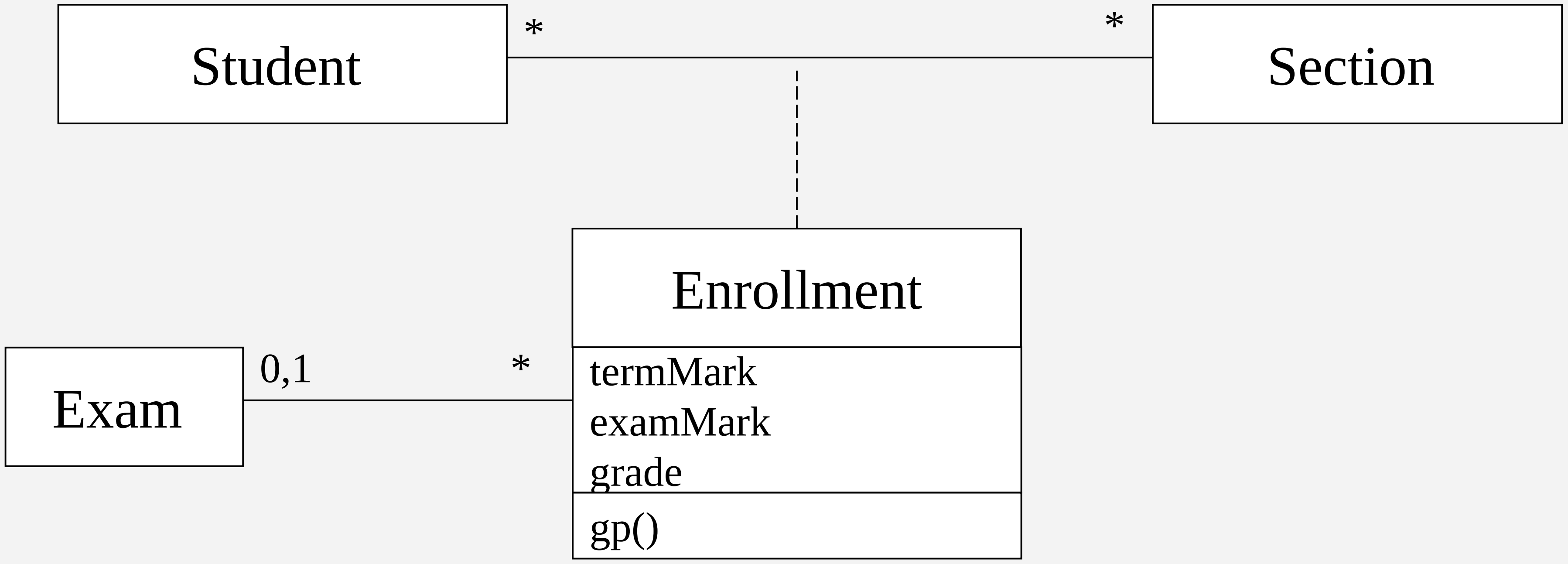
An association class is an association and it is a class– it can participate in associations



Class Diagram

Association classes

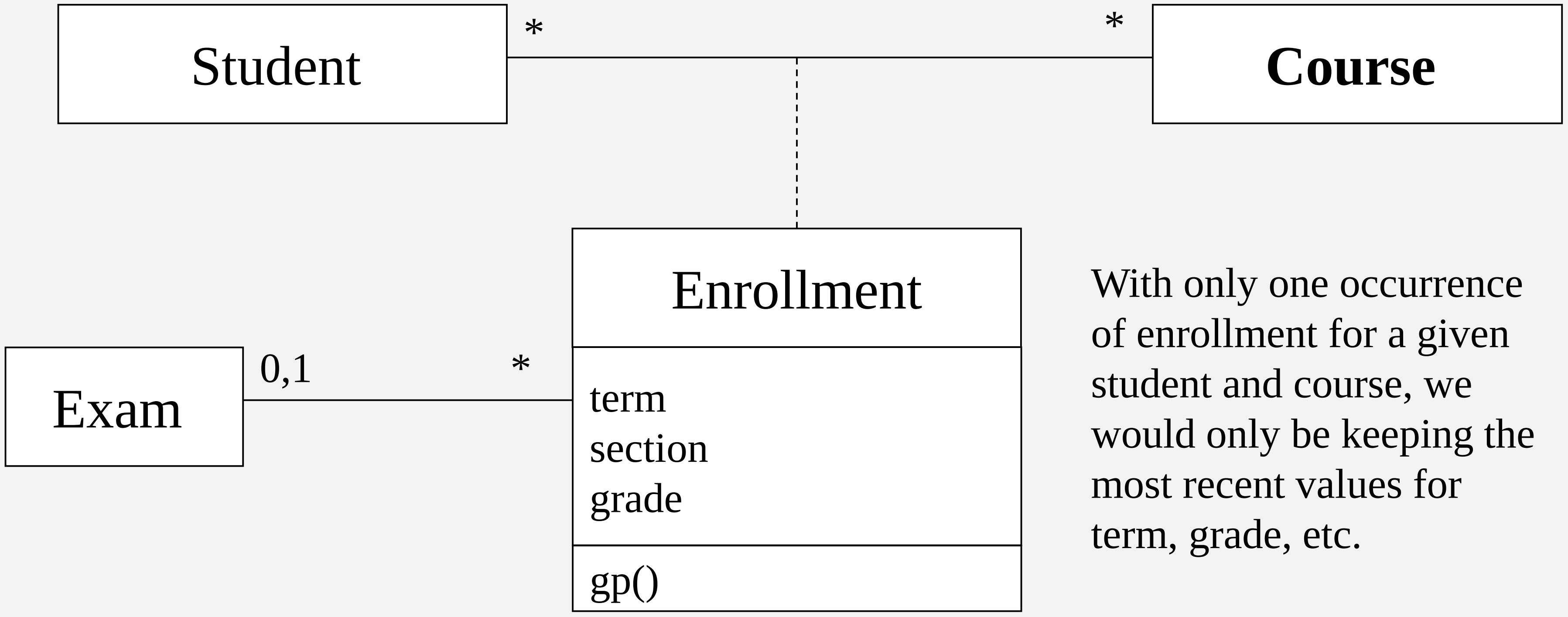
For a given student and section there can be only one occurrence of Enrollment. This rule is very restrictive.



Class Diagram

Association classes

In this example we are consider an association between Student and **Course**.

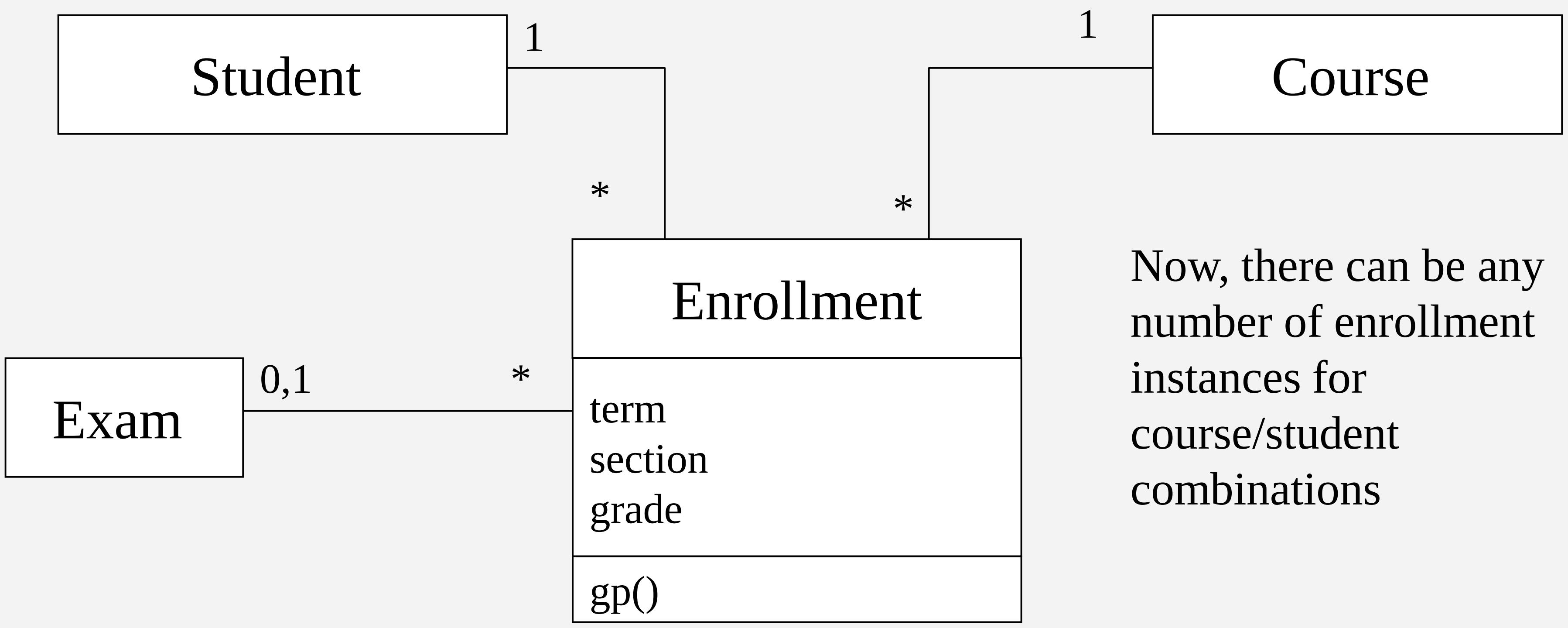


With only one occurrence of enrollment for a given student and course, we would only be keeping the most recent values for term, grade, etc.

Class Diagram

Association classes

To allow more flexibility, the modeler might promote an association class to a full class, as in:

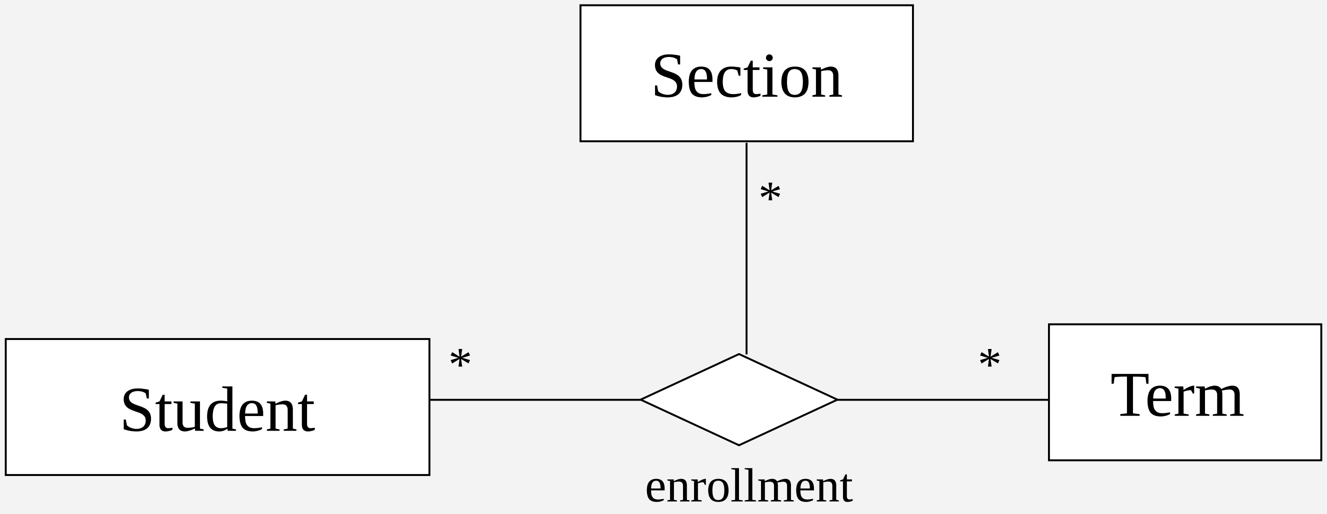


Now, there can be any number of enrollment instances for course/student combinations

Class Diagram

N-ary associations

An n-ary association is an association among 3 or more classes



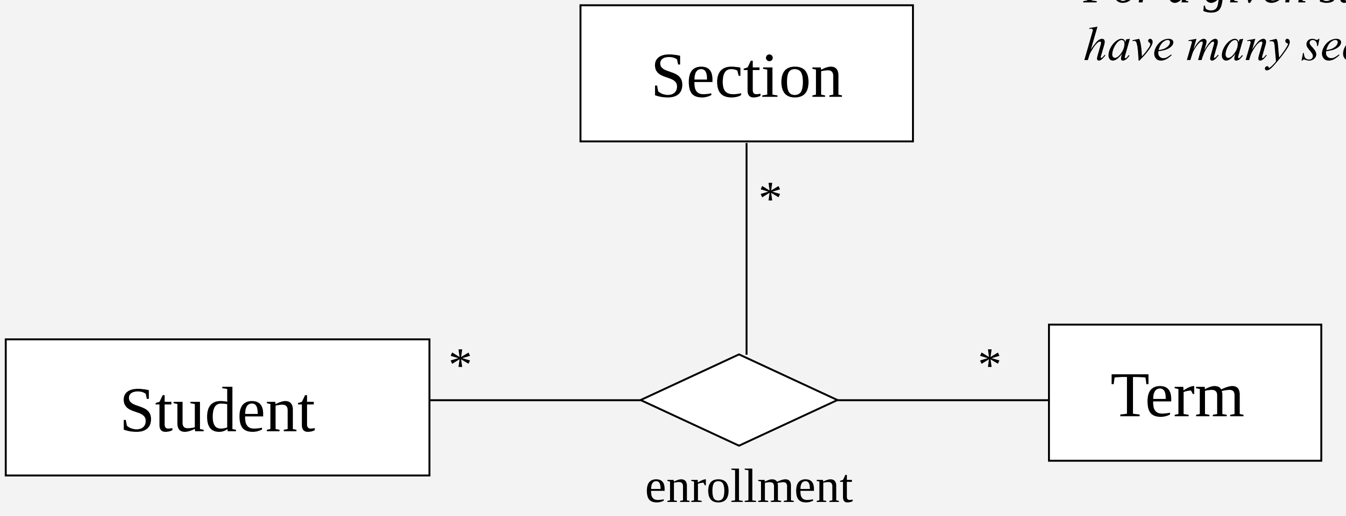
Class Diagram

N-ary associations

Each instance of the association is an n-tuple of values from the respective classes. For each association we have one student, one section, one term

The multiplicity on a role represents the potential number of instance tuples in the association when the other n-1 values are fixed.

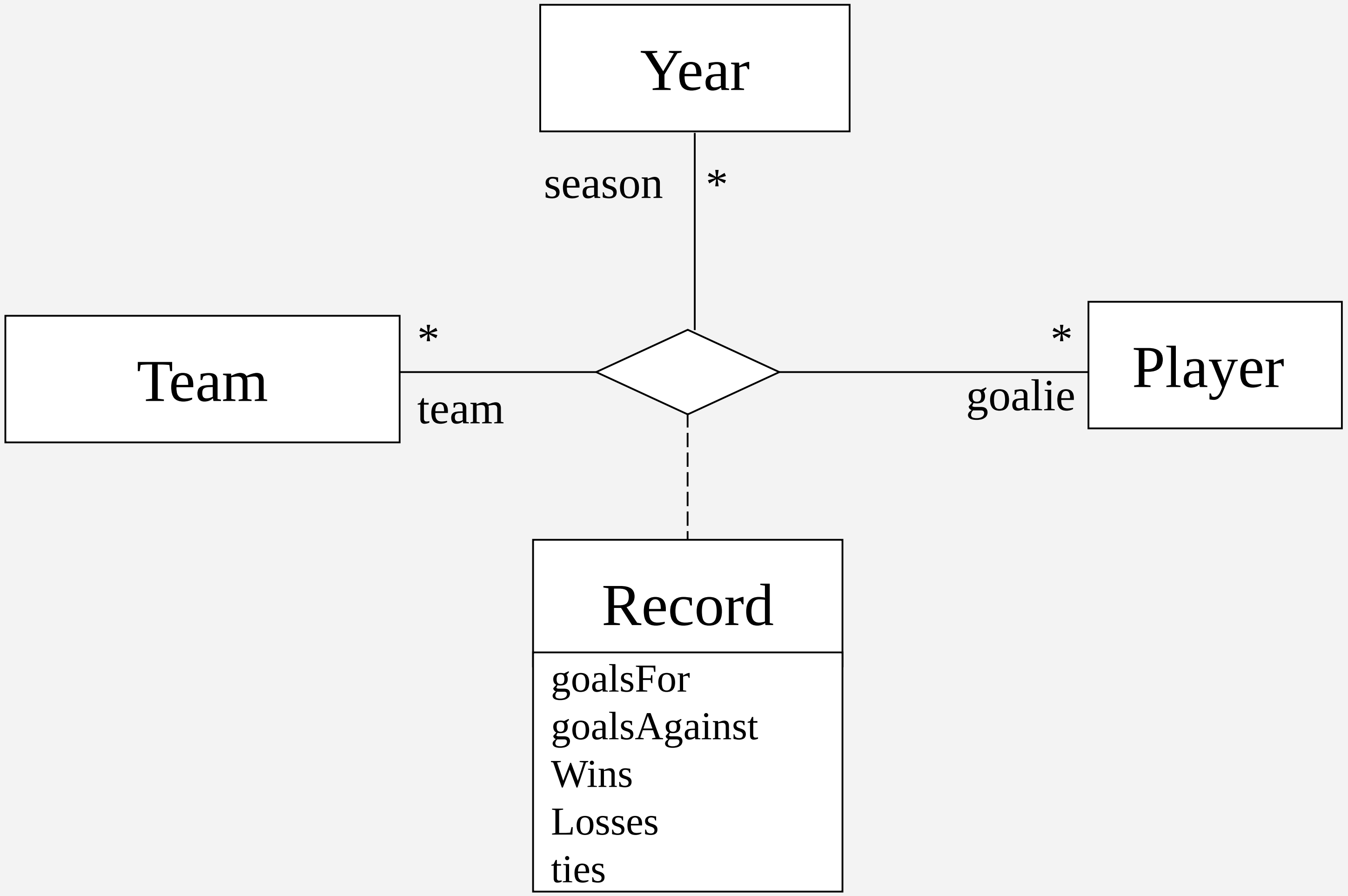
For a given student and term, we have many sections, …



Class Diagram

N-ary associations

Consider a team, goalies, and the season. We could record the performance of each goalie for each team and each season.



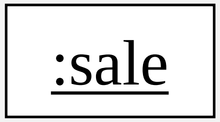
Object Diagram

An object diagram is an object graph showing objects (possibly named and including attribute values) and links.

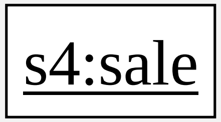
A link shows a connection from one object to another.

Objects

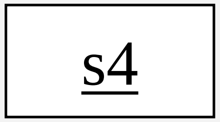
An individual object (instance of a class) is shown with naming information underlined



An unnamed sale object

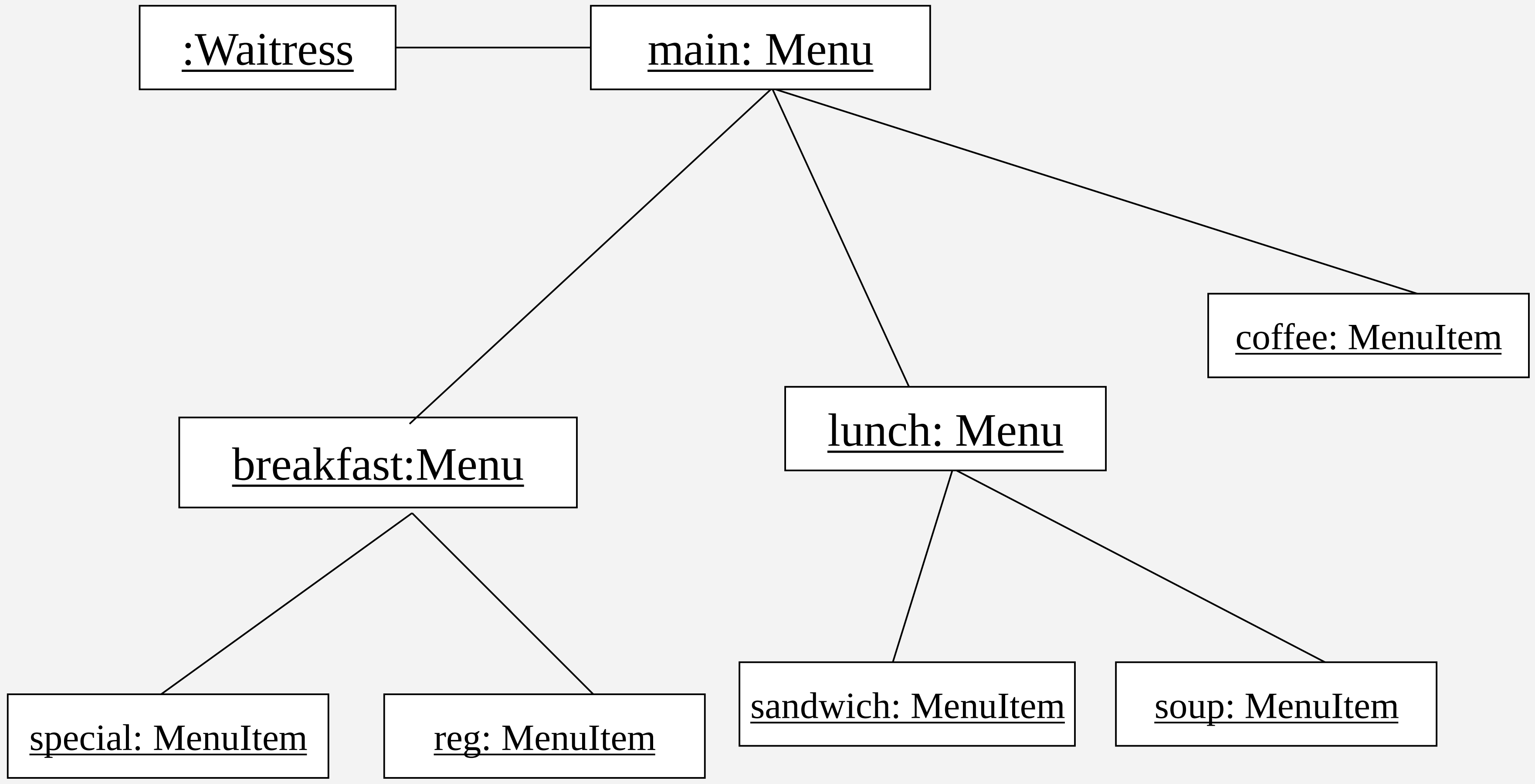


A sale object named s4



An object named s4

An object diagram



Interfaces

An interface is special type of class that cannot be instantiated. An application can never instantiate an interface.

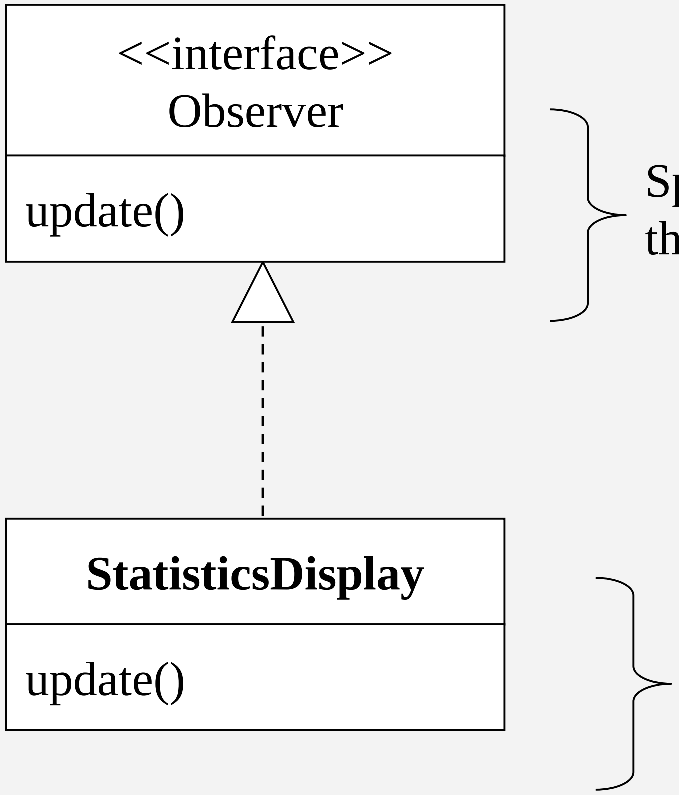
An interface defines a set of public attributes and operations that some class must use

There is no behaviour defined, no method coded (exceptions now with Java 8 where we can now specify default interface methods

– see https://docs.oracle.com/javase/tutorial/java/IandI/defaultmethods.html)

Normally, a class inherits the interface and provides the implementation

From Head First …



Specifies the signature for update but there is no implementation, no code

Note : the line is a dashed line!

implements the update operation – there is code here

Next slide shows code for this

From Head First …

```
public interface Observer {
        public void update(float temp, float humidity, float pressure);
}
```

```
import java.util.*;
public class StatisticsDisplay implements Observer {
        …
        public void update(float temp, float humidity, float pressure) {
                tempSum += temp;
                numReadings++;
                if (temp > maxTemp) {
                        maxTemp = temp;
                }
                if (temp < minTemp) {
                        minTemp = temp;
                }
                display();
        }
```