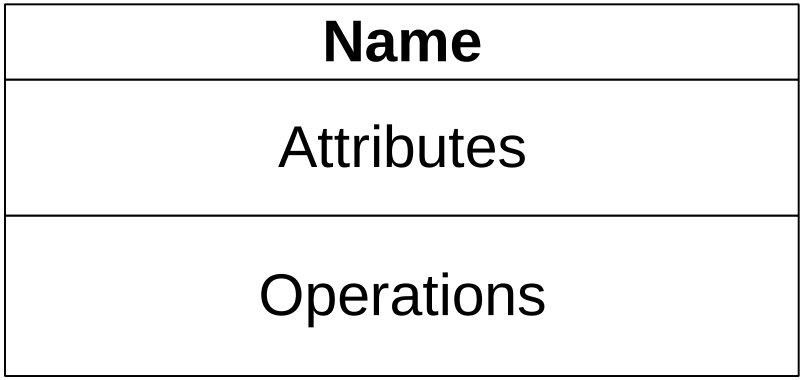
The UML Class Diagram

- Is a static diagram (describes system structure)
  - Combines a number of model elements:
    - Classes
    - Attributes
    - Operations (methods)
    - Associations
    - Aggregations
    - Compositions
    - Generalisations

A UML Class



Name

Attributes

Operations

Properties of class diagrams:

- Static model;
- Models structure *and* behaviour;
- Used as a basis for other diagrams;
- Easily converted to an object diagram.

Determining Classes (1/2)

- Is there data that requires storage, transformation or analysis?
- Are there external systems interacting with the one in question?
- Are any class libraries or components being use (from manufacturers, other colleagues or past projects)?
- Does the system handle any devices?
- Does the system model organisational structures?
- Analyse all actor roles.

Determining Classes (2/2)

- **Textual Analysis** *(based on Dennis, 2002)*
  - ***A common or improper noun*** implies a class
  - ***A proper noun or direct reference*** implies an object (instance of a class)
  - ***A collective noun*** implies a class made up of groups of objects from another class
  - ***An adjective*** implies an attribute
  - ***A "doing" verb*** implies an operation
  - ***A "being" verb*** implies a classification relationship between an object and its class
  - ***A "having" verb*** implies an aggregation or association relationship
  - ***A transitive verb*** implies an operation
  - ***An intransitive verb*** implies an exception
  - ***A predicate or descriptive verb phrase*** implies an operation
  - ***An adverb*** implies an attribute of a relationship or an operation

UML Class Attributes (1/2)

- Very system dependent
- Describe characteristics of objects belonging to that class
- Can be informative - or confusing
- Has a definite type
  - – Primitive (Boolean, integer, real, enumerated, etc.)
  - – language specific
  - – other classes
  - – any user defined type
- Has different visibility, including:
  - – public (viewed and used from other classes)
  - – private (cannot be accessed from other classes)

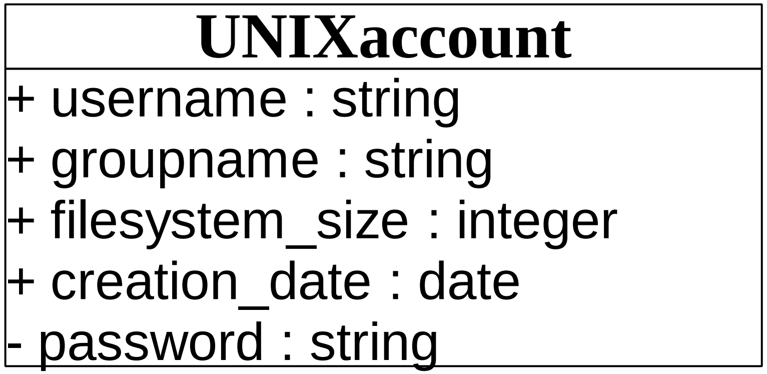
UML Class Attributes (2/2)

- Can be given a default value
- Can be given class-scope
- Can list possible values of enumeration
- Directly implementable into most modern programming languages with object-oriented support *(e.g. Java)*

Attribute syntax:

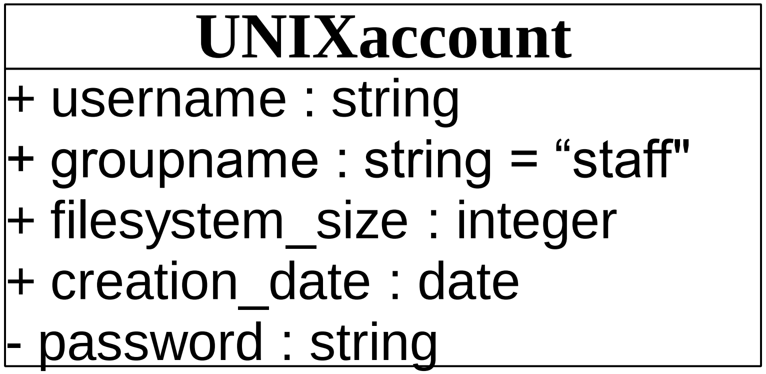
```
Visibility name:type=init_value{property_string}
```

UML Class Attribute Examples



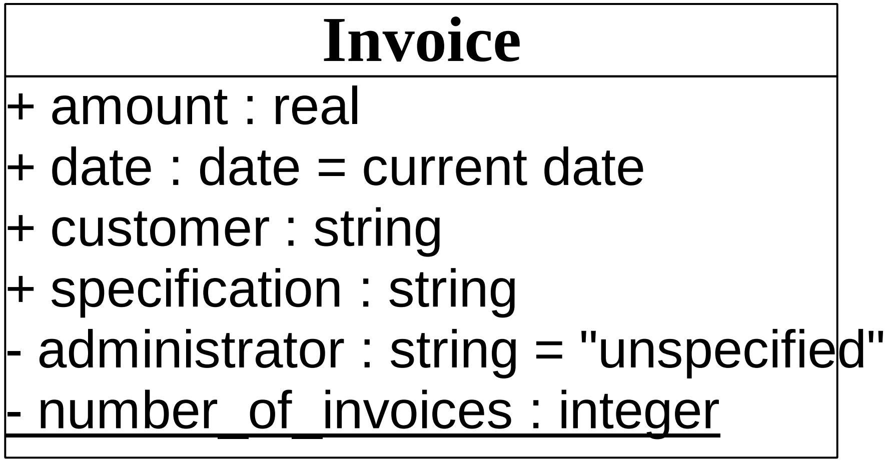
UNIXaccount

+ username : string
+ groupname : string
+ filesystem_size : integer
+ creation_date : date
- password : string



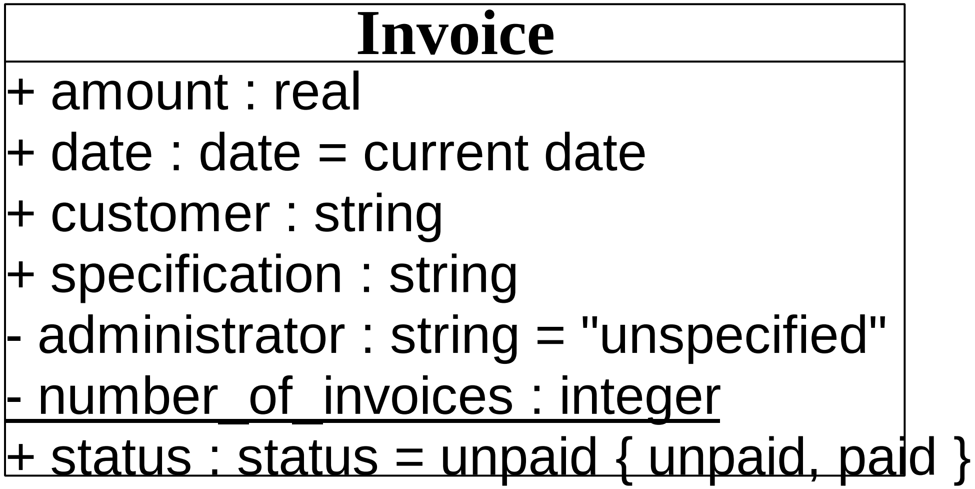
UNIXaccount

+ username : string
+ groupname : string = “staff"
+ filesystem_size : integer
+ creation_date : date
- password : string



Invoice

+ amount : real
+ date : date = current date
+ customer : string
+ specification : string
- administrator : string = "unspecified"
- number_of_invoices : integer

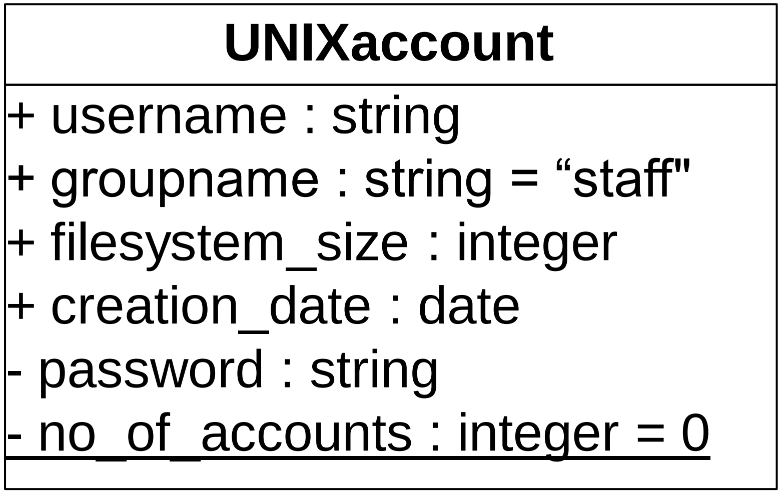


Invoice

+ amount : real
+ date : date = current date
+ customer : string
+ specification : string
- administrator : string = "unspecified"
- number_of_invoices : integer
+ status : status = unpaid { unpaid, paid }

UML Class-to-Java Example

```
Public class UNIXaccount
{
  public string username;
  public string groupname = "csai";
  public int filesystem_size;
  public date creation_date;
  private string password;
  static private integer no_of_accounts = 0
  public UNIXaccount()
  {
    //Other initialisation
    no_of_accounts++;
  }
  //Methods go here
};
```



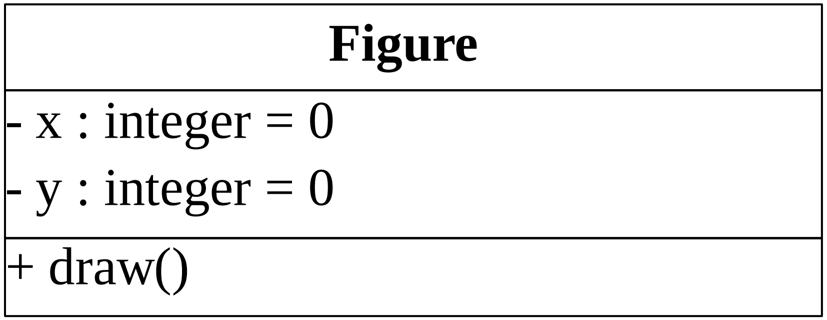
UNIXaccount

+ username : string
+ groupname : string = “staff"
+ filesystem_size : integer
+ creation_date : date
- password : string
- no_of_accounts : integer = 0

Operations (Methods)

```
Public class Figure
{
  private int x = 0;
  private int y = 0;
  public void draw()
  {
    //Java code for drawing figure
  }
};
```

```
Figure fig1 = new Figure();
Figure fig2 = new Figure();
fig1.draw();
fig2.draw();
```



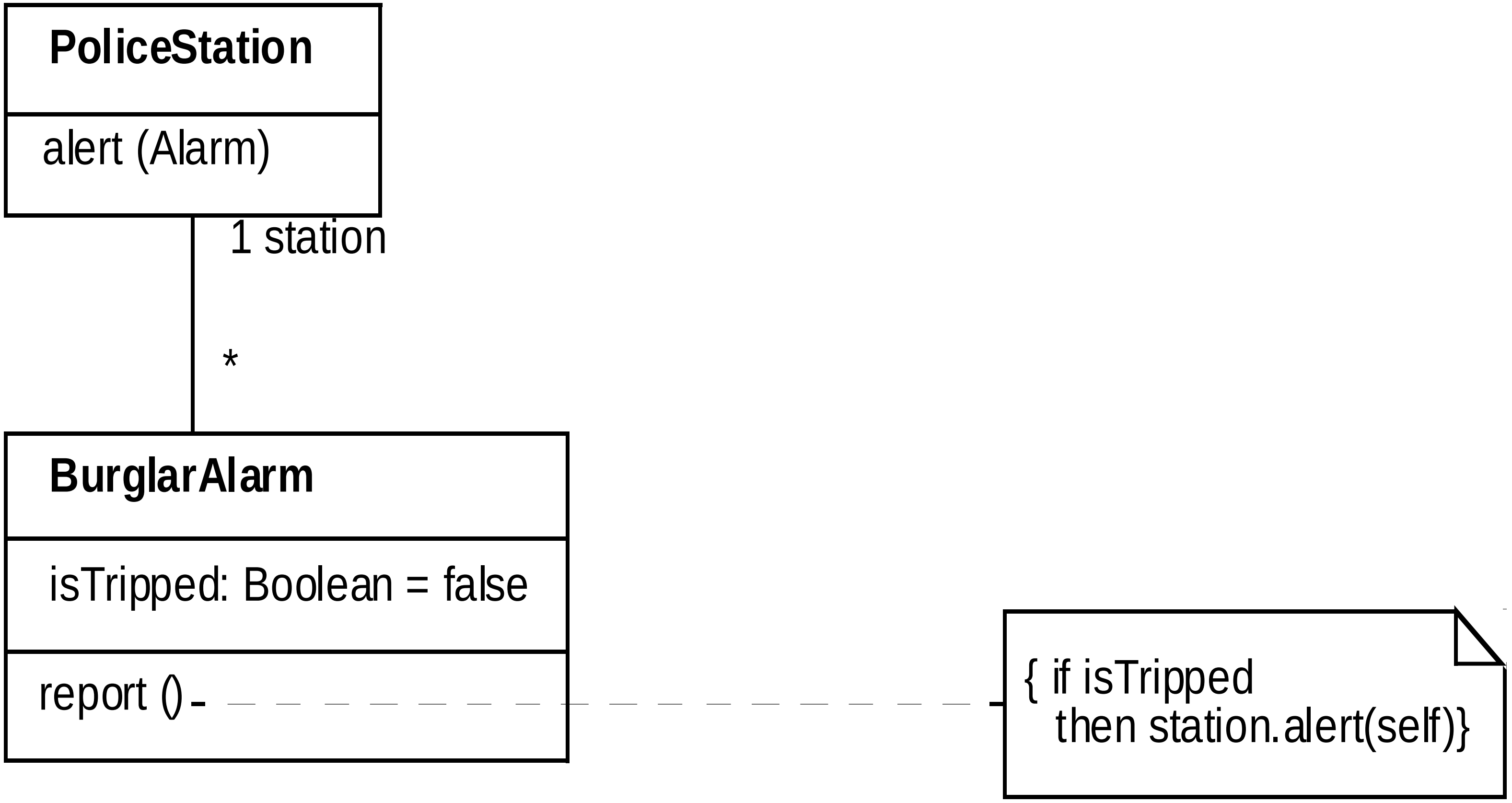
Figure

- x : integer = 0

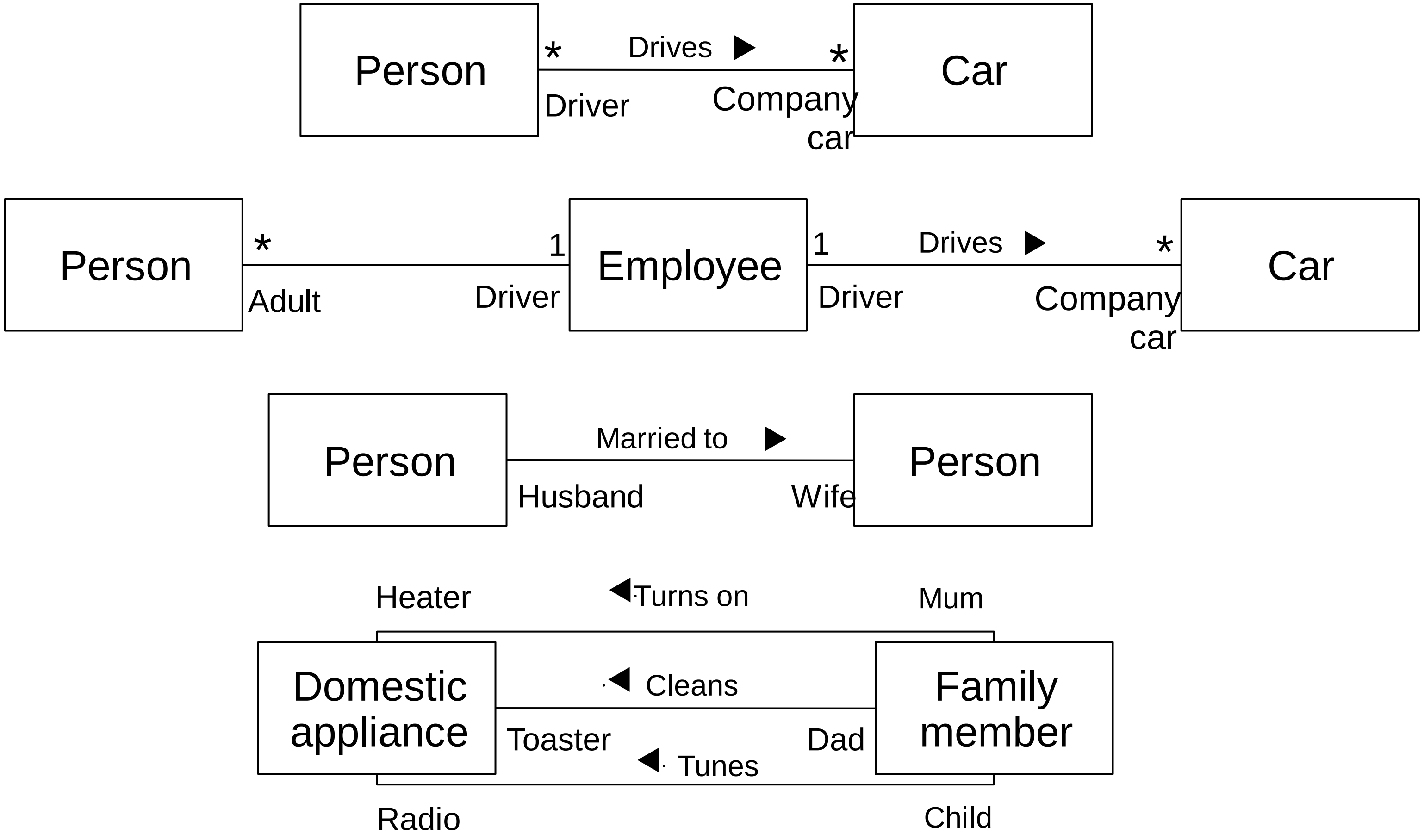
- y : integer = 0

+ draw()

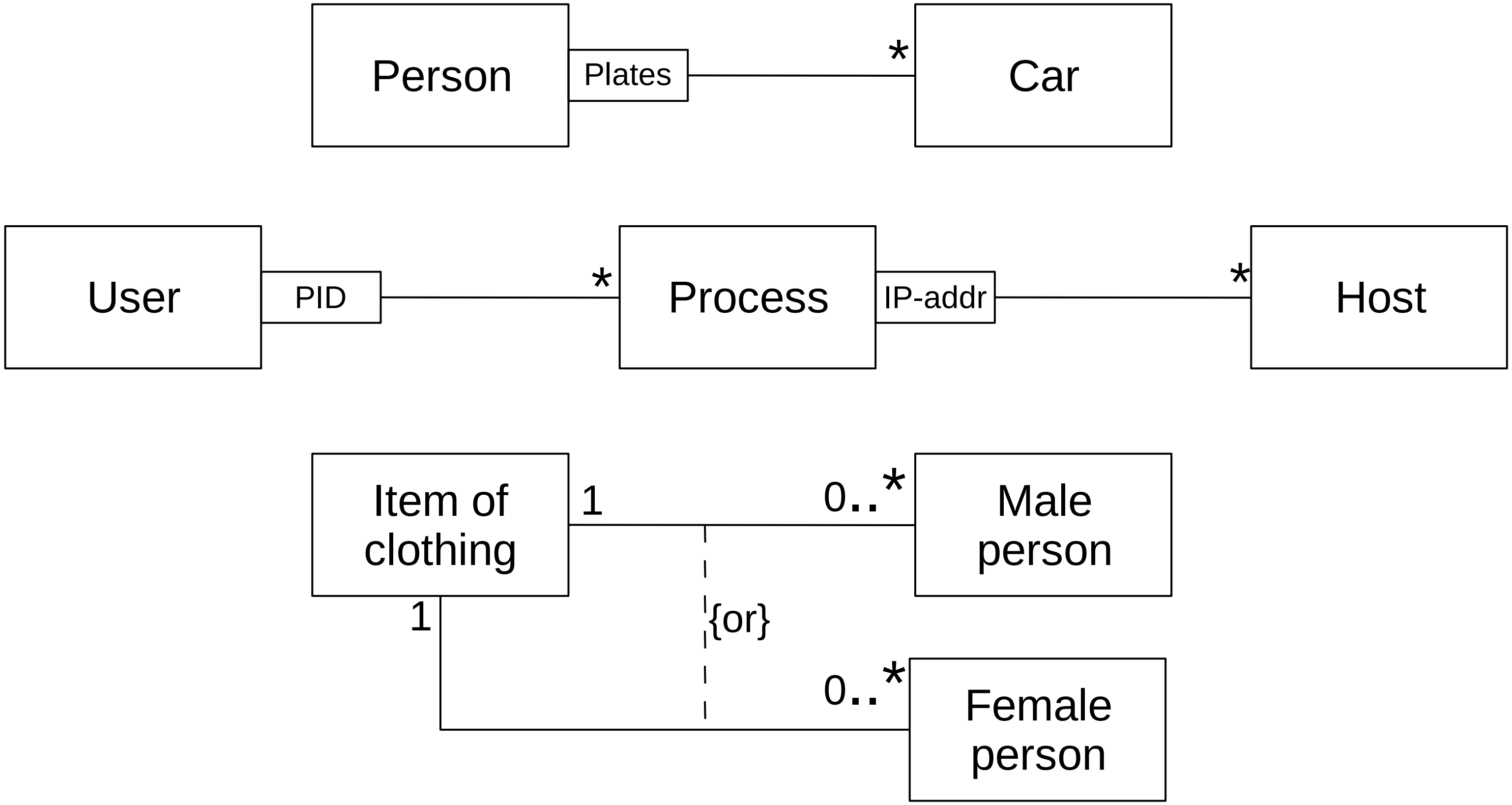
Constraints on Operations



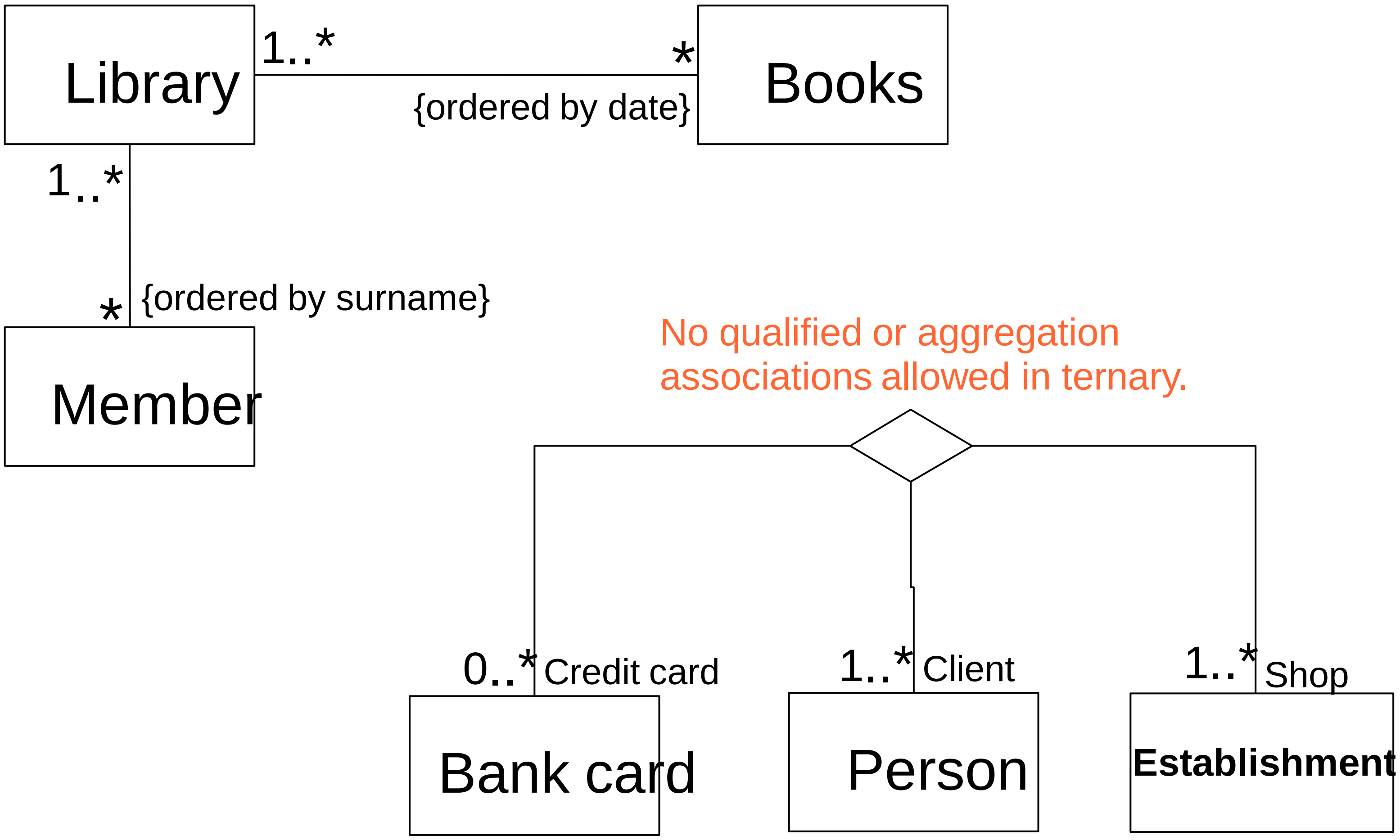
Association Examples



Qualified and "Or" Associations

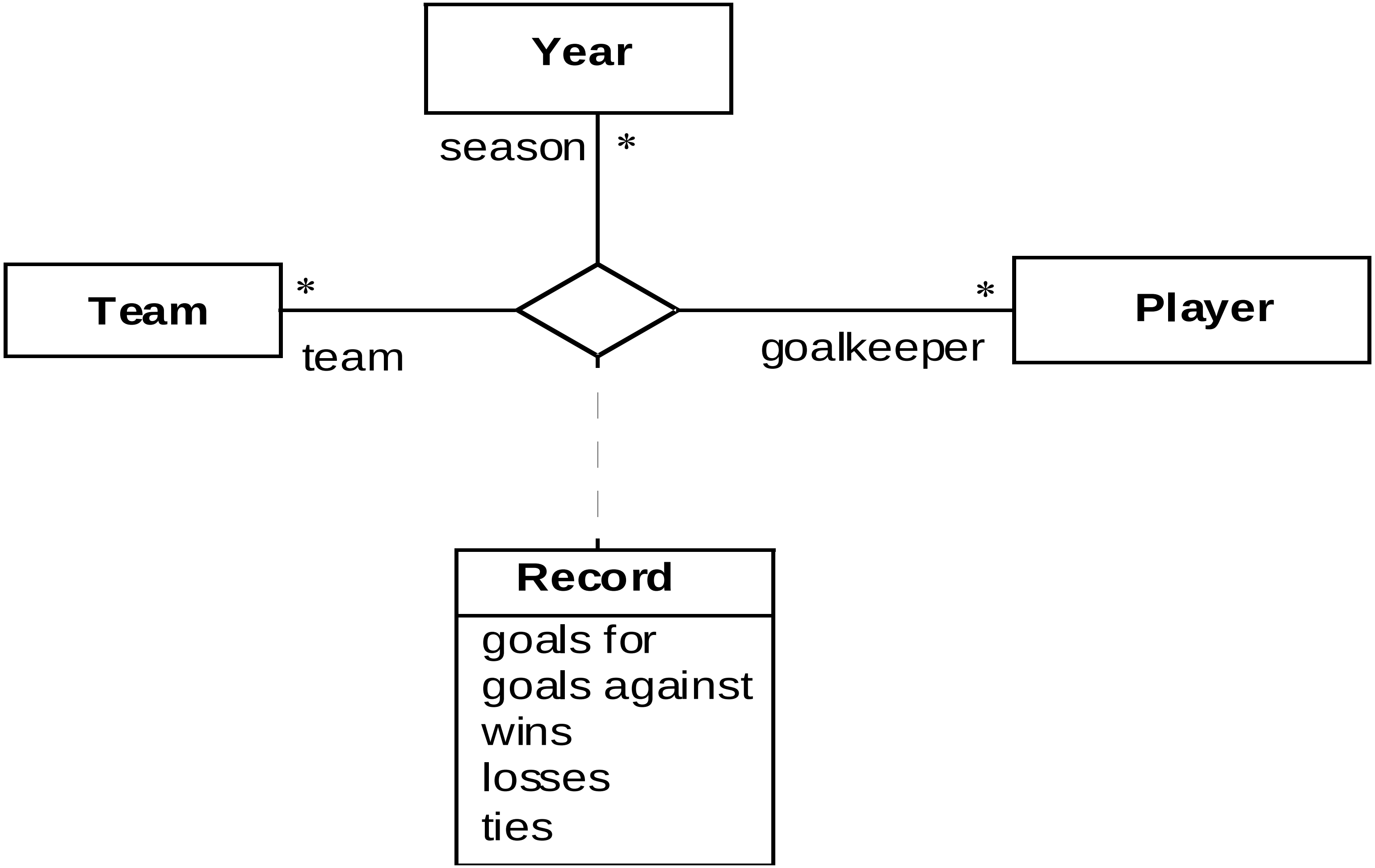


Ordered and Ternary Associations

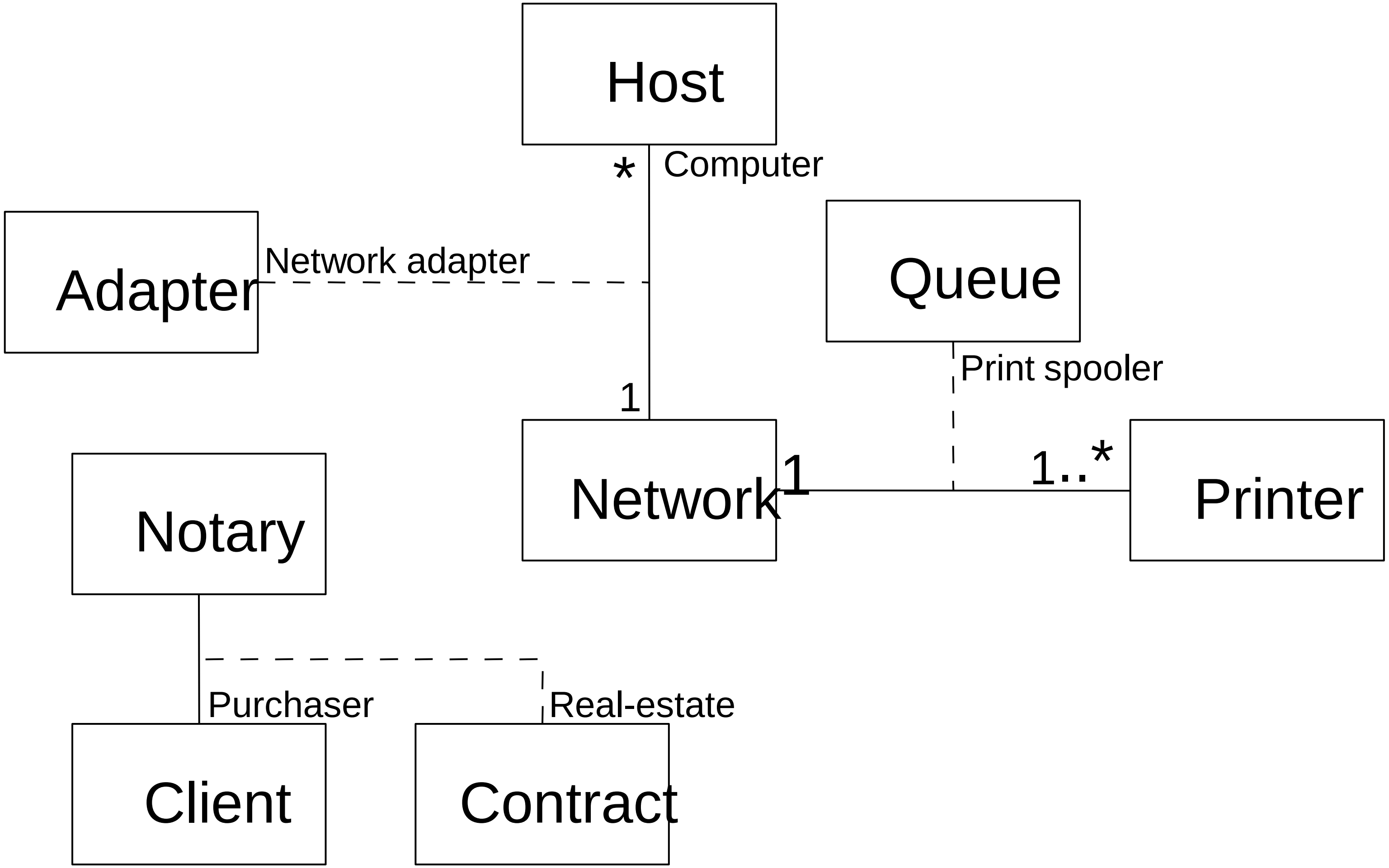


No qualified or aggregation associations allowed in ternary.

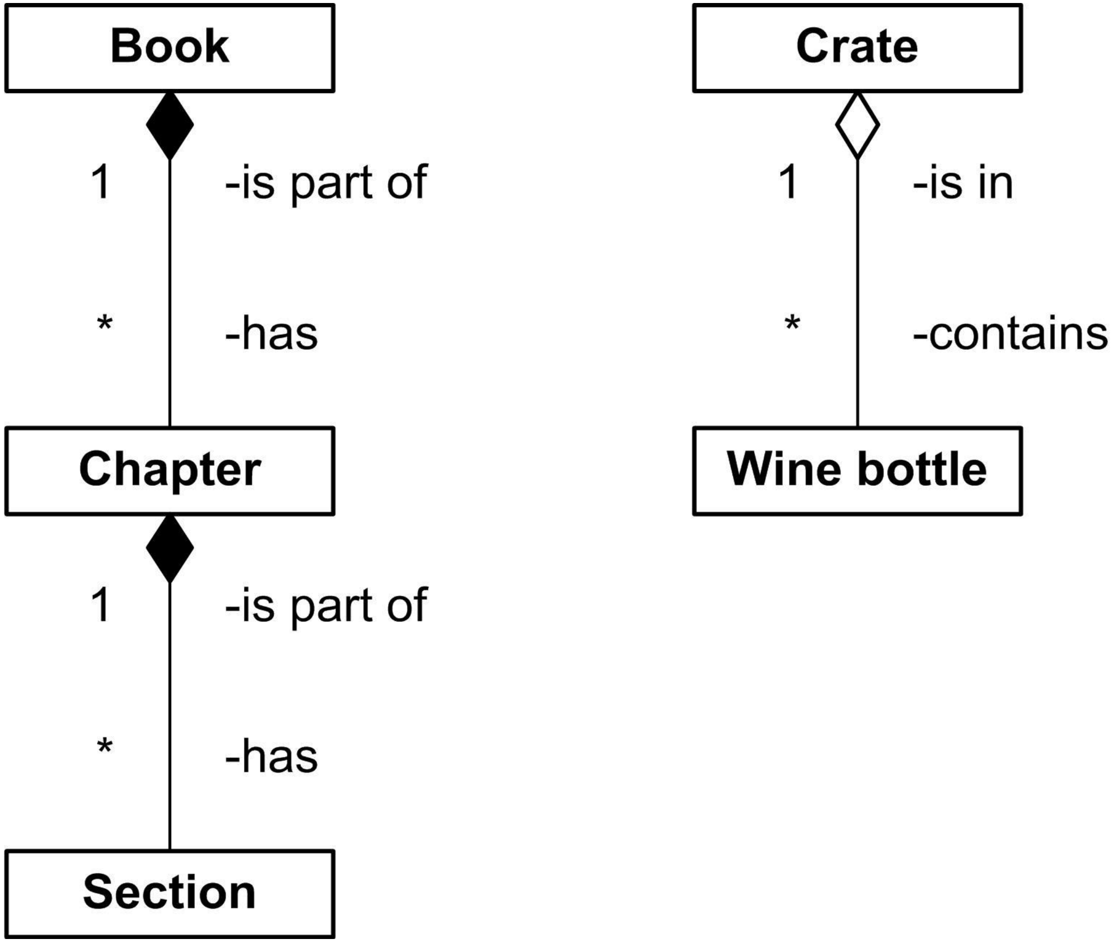
Another Ternary Association Example



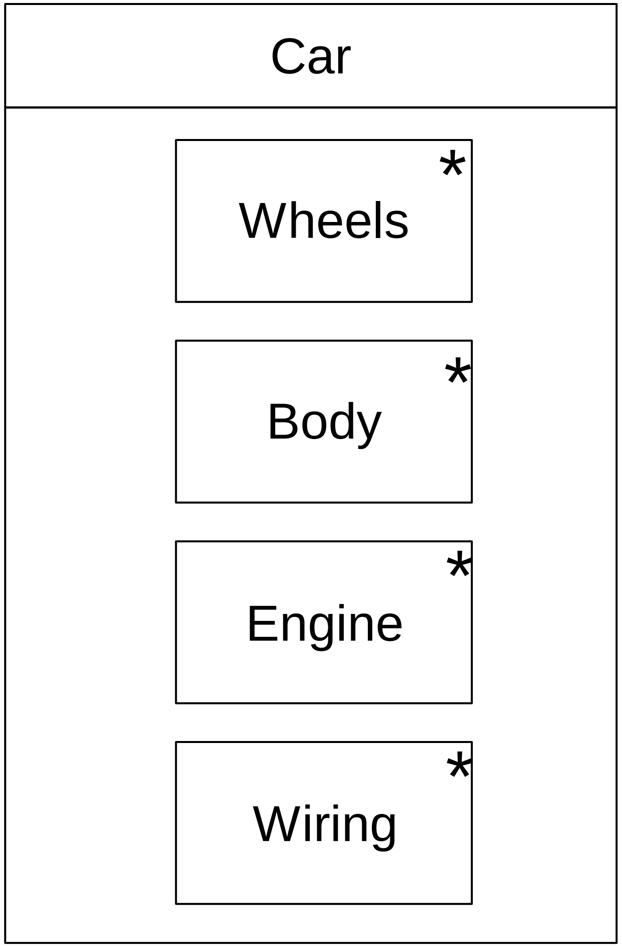
Association Classes



Association by Aggregation

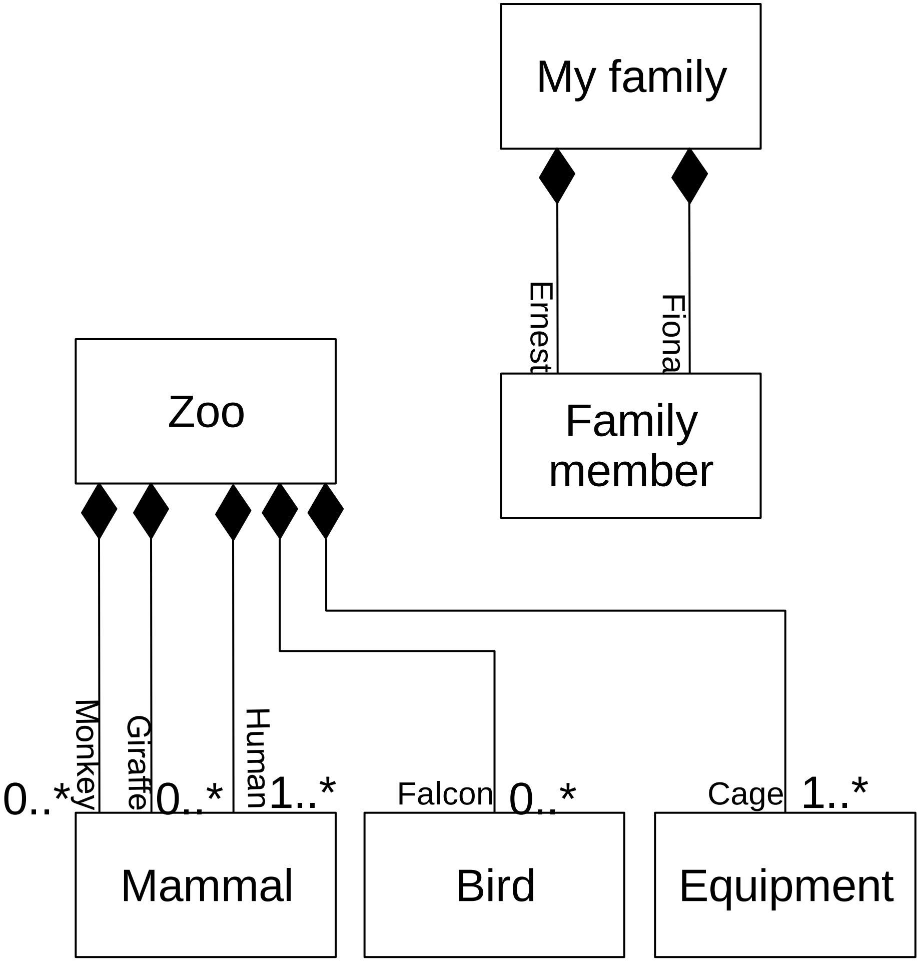


Alternative Notation for Composition Association



Note that association multiplicity is shown within the classes

Roles in Aggregation



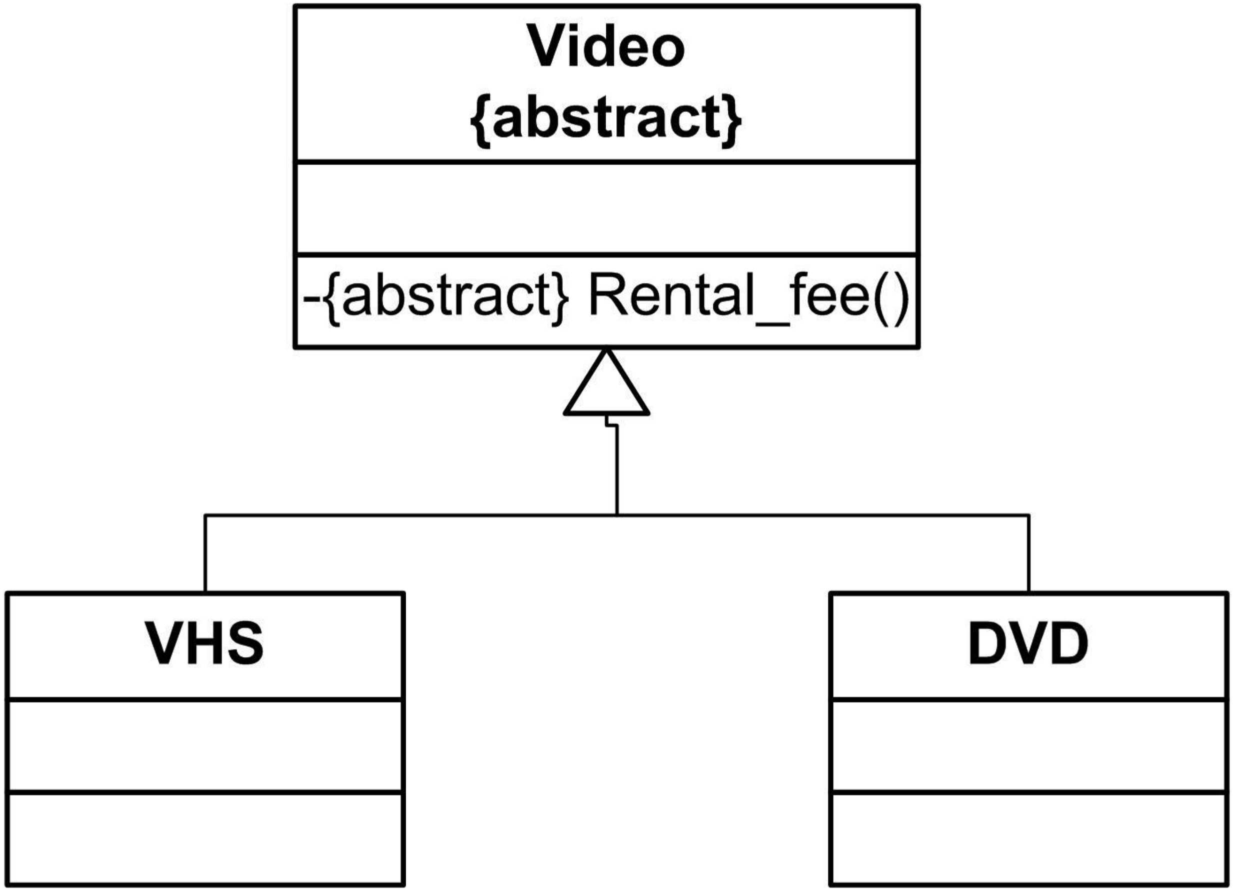
My family

Ernest: Family member
Fiona: Family member

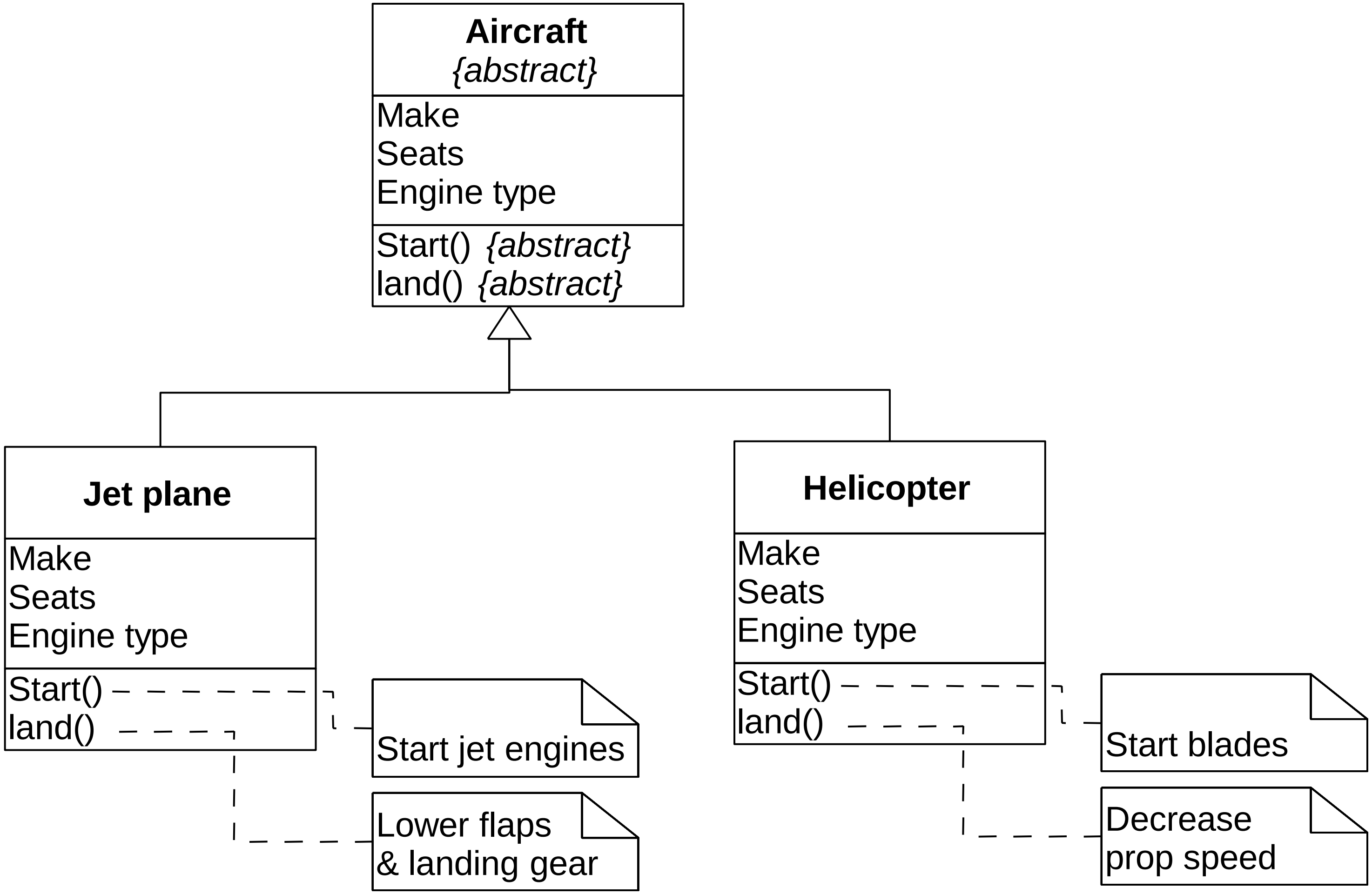
Zoo

Monkey[0..*]: Mammal
Giraffe[0..*]: Mammal
Human[1..*]: Mammal
Falcon[0..*]: Bird
Cage[1..*]: Equipment

Abstract Classes



Abstract Classes and Generalisation Example



Aircraft
{abstract}
Make
Seats
Engine type
Start() *{abstract}*
land() *{abstract}*

Jet plane
Make
Seats
Engine type
Start()
land()

Start jet engines

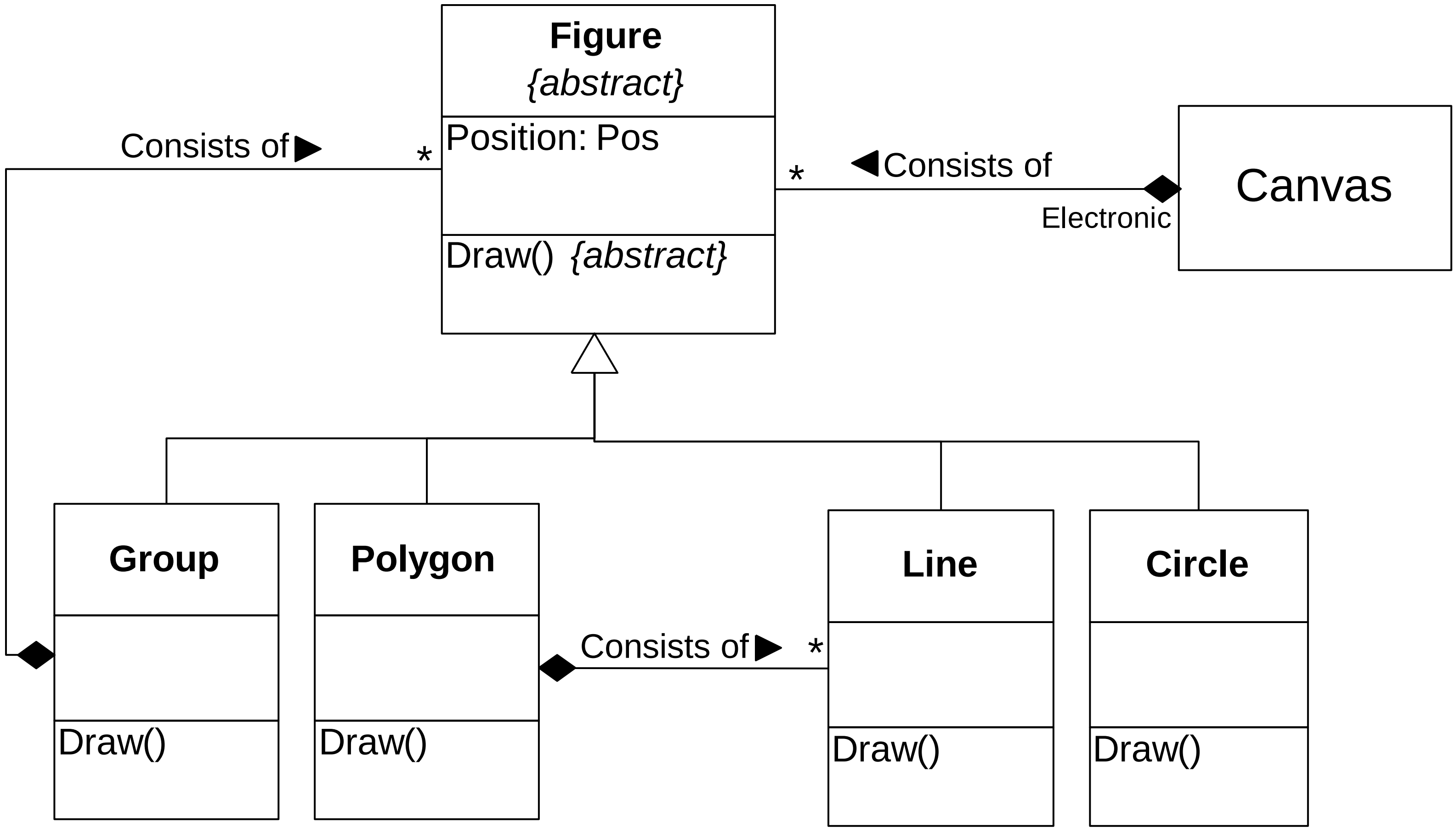
Lower flaps
& landing gear

Helicopter
Make
Seats
Engine type
Start()
land()

Start blades

Decrease
prop speed

Aggregation and Generalisation



Implementing it (e.g. in Java)

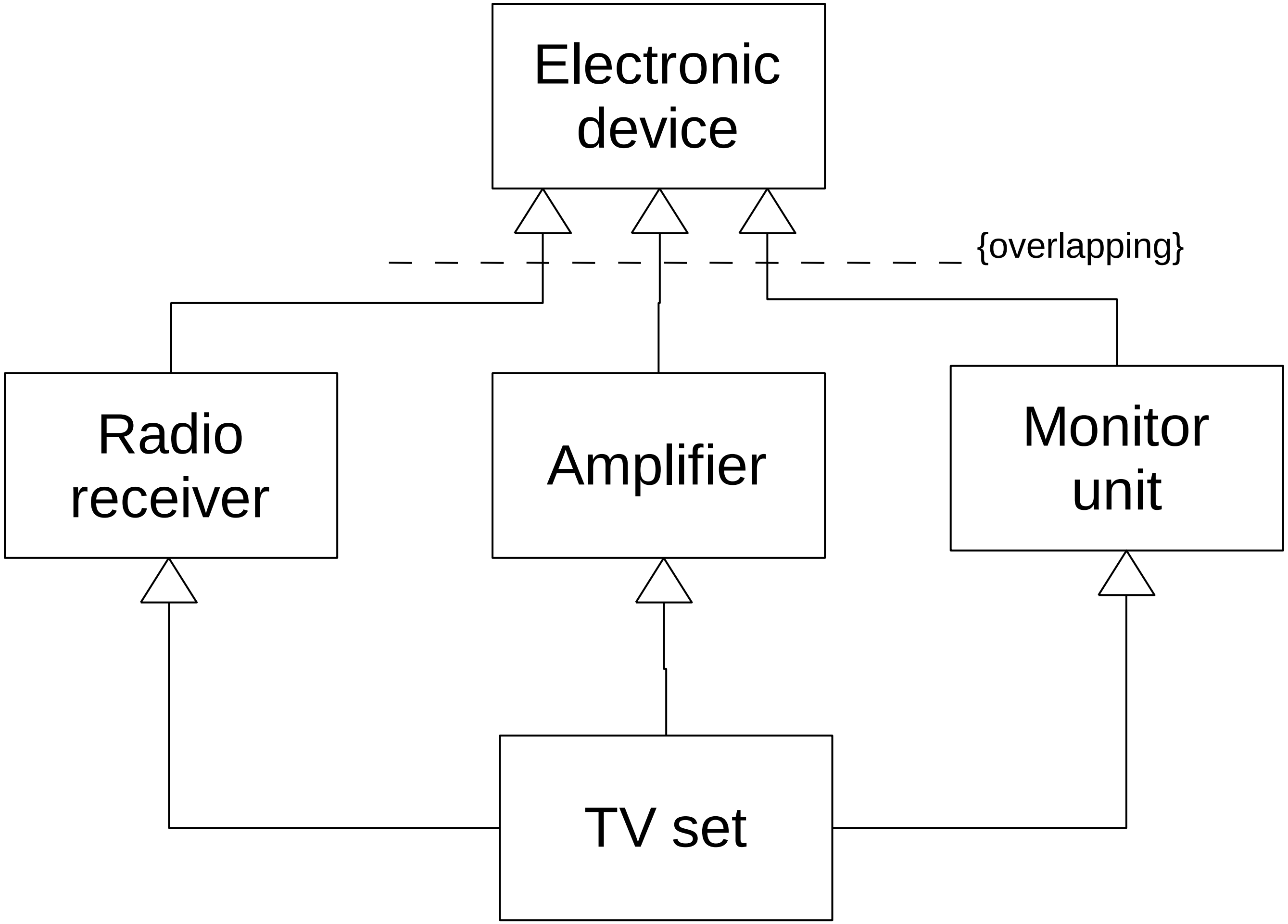
```
abstract public class Figure
{
  abstract public void Draw();
  Pos position;
}
public class Group extends Figure
{
  private FigureVector consist_of;
  public void Draw()
  {
    for (int i = 0; i < consist_of.size(), i++)
    {
      consist_of[i].draw();
    }
  }
}
public class Polygon extends Figure
{
  public void Draw()
  {
    /* something similar to group
        only using lines instead */
  }
}
```

```
public class Line extends Figure
{
  public void Draw()
  {
    /* code to draw line */
  }
}
public class circle extends Figure
{
  public void Draw()
  {
    /* code to draw circle */
  }
}
```

Constrained Generalisations

- Overlapping
  - A type of inheritance whereby sharing of common sub-classes by other sub-classes is allowed.
- Disjoint *(the default)*
  - The opposite of overlapping.
- Complete
  - A type of inheritance whereby the existing sub-classes are said to fully define a given super-class. No further sub-classing may be defined.
- Incomplete *(the default)*
  - Further sub-classes can be added later on to more concretely specify a given super-class.

Overlapping Generalisation



Electronic device

{overlapping}

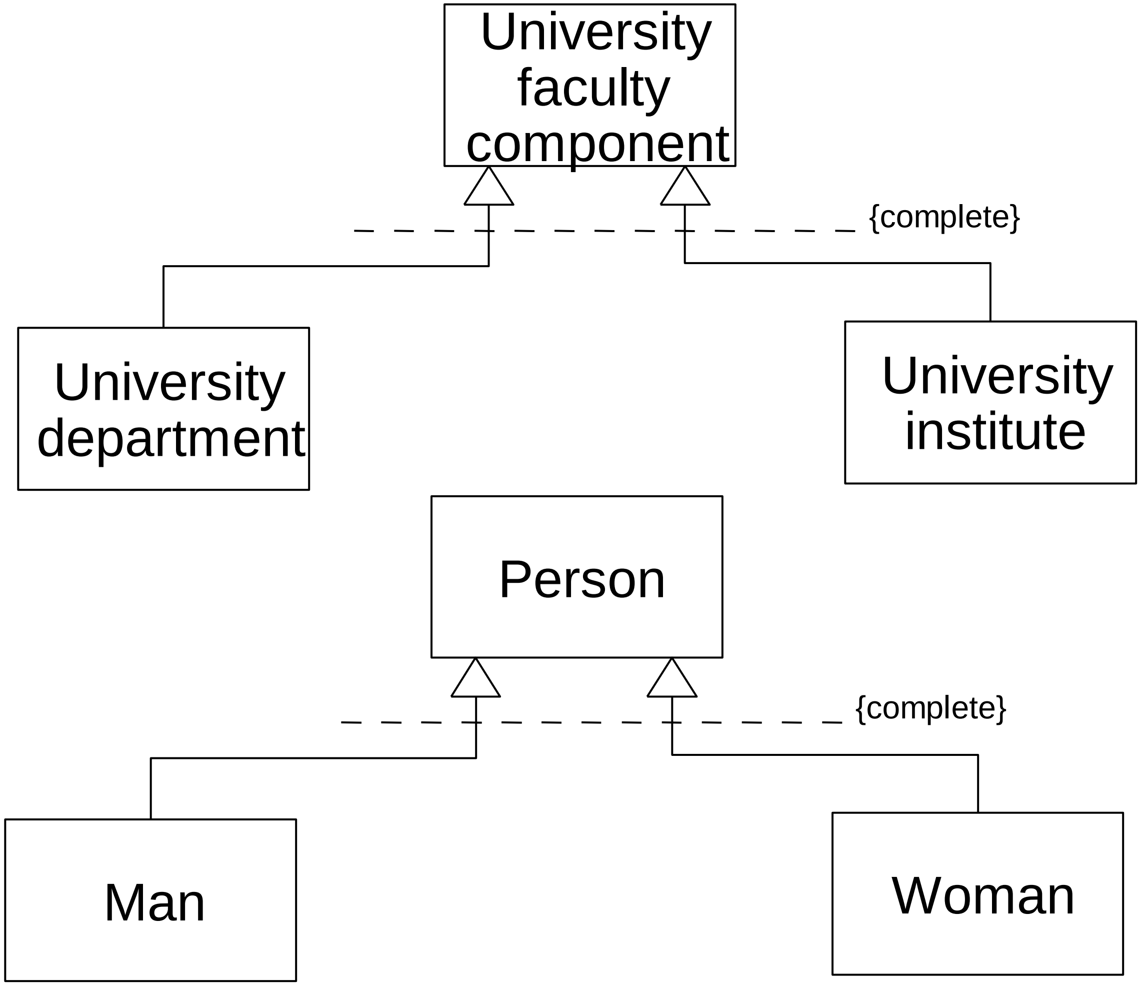
Radio receiver

Amplifier

Monitor unit

TV set

Complete Generalisation

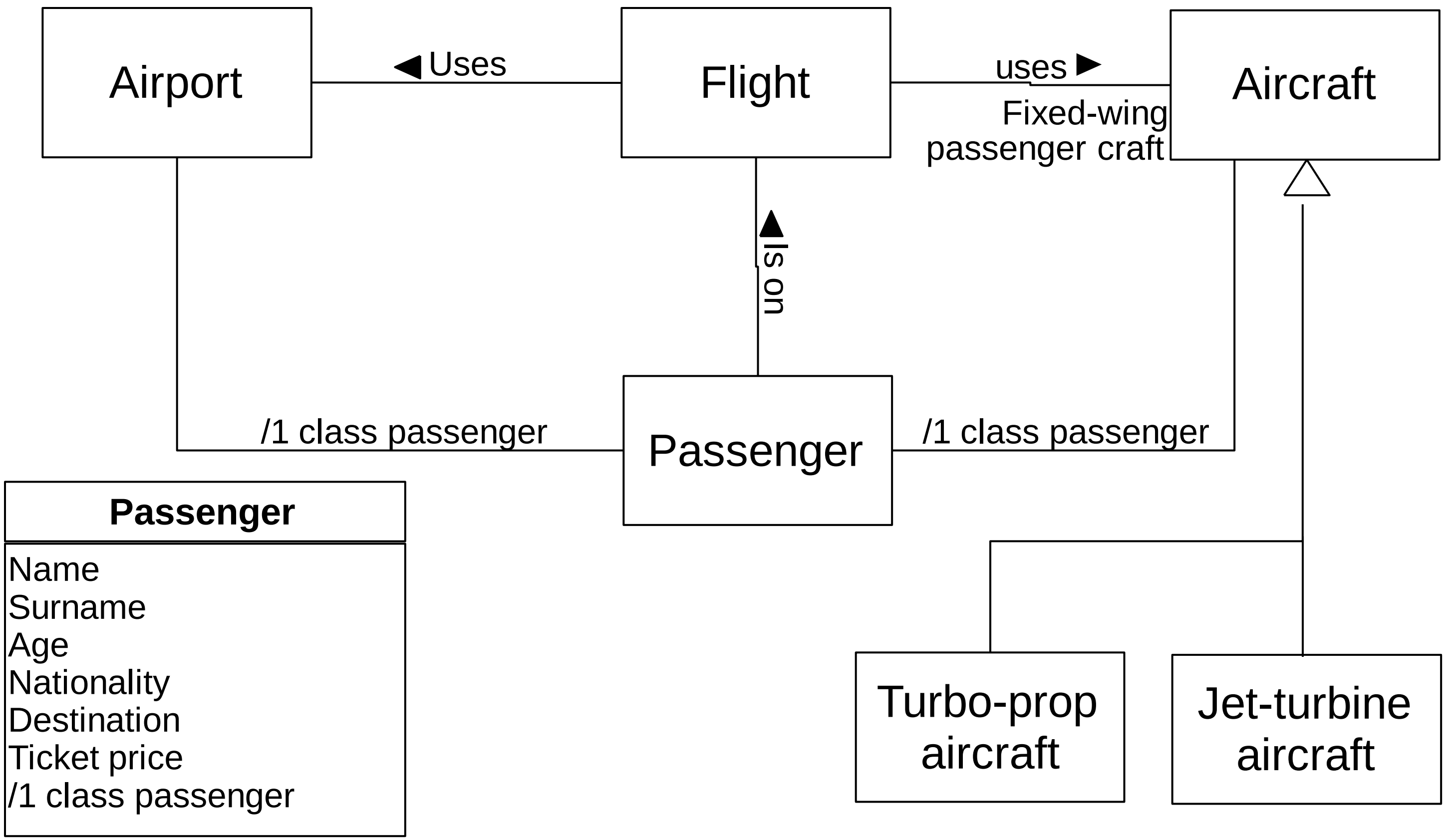


Expressing Rules in UML

- Rules are expressed using constraints and derivations
  - Constraints were mentioned earlier *(e.g. or-associations, ordered associations, inheritance constraints, etc.)*

  - Derivations are rules governing how entities can be derived *(e.g. age = current date - DOB)*

Example of Derived Associations



Airport

◀ Uses

Flight

uses ▶

Fixed-wing passenger craft

Aircraft

▲ Is on

/1 class passenger

Passenger

/1 class passenger

Turbo-prop aircraft

Jet-turbine aircraft

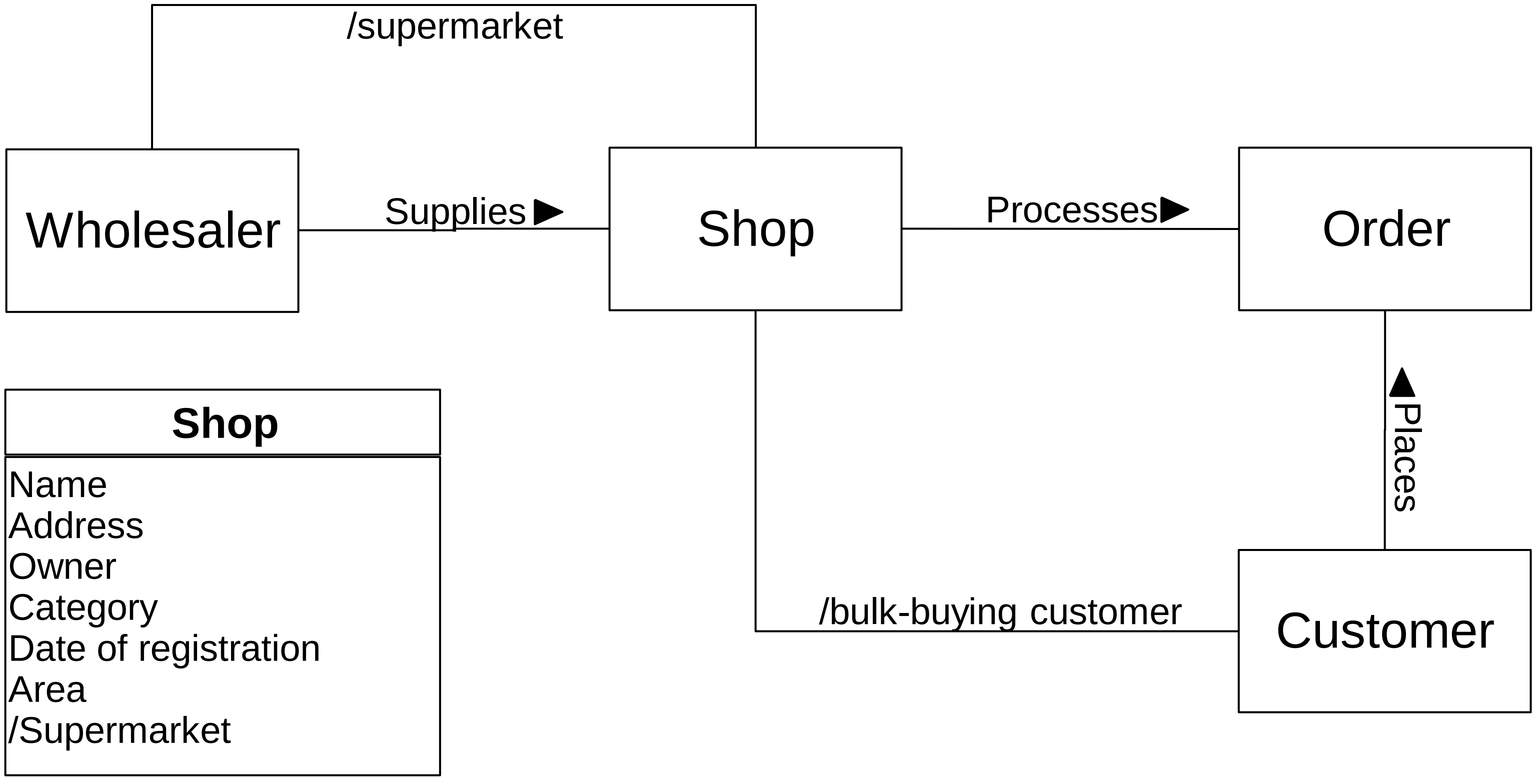
Passenger

Name
Surname
Age
Nationality
Destination
Ticket price
/1 class passenger

{1 class passenger = = (Ticket price > 400)}

N.B. Relation cardinality is omitted for example clarity

Another Example of a Derived Association



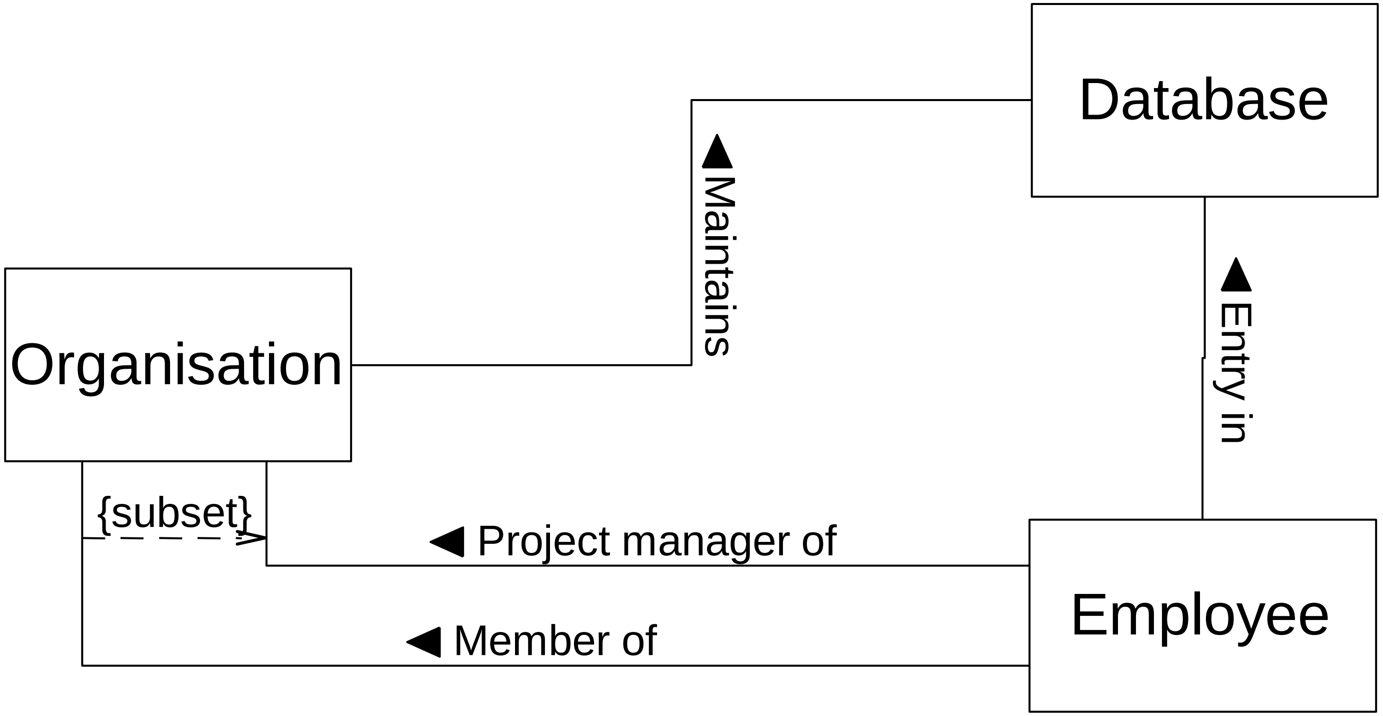
Shop

Name
Address
Owner
Category
Date of registration
Area
/Supermarket

{Supermarket = = (Area > 200 && Category = "dept")}

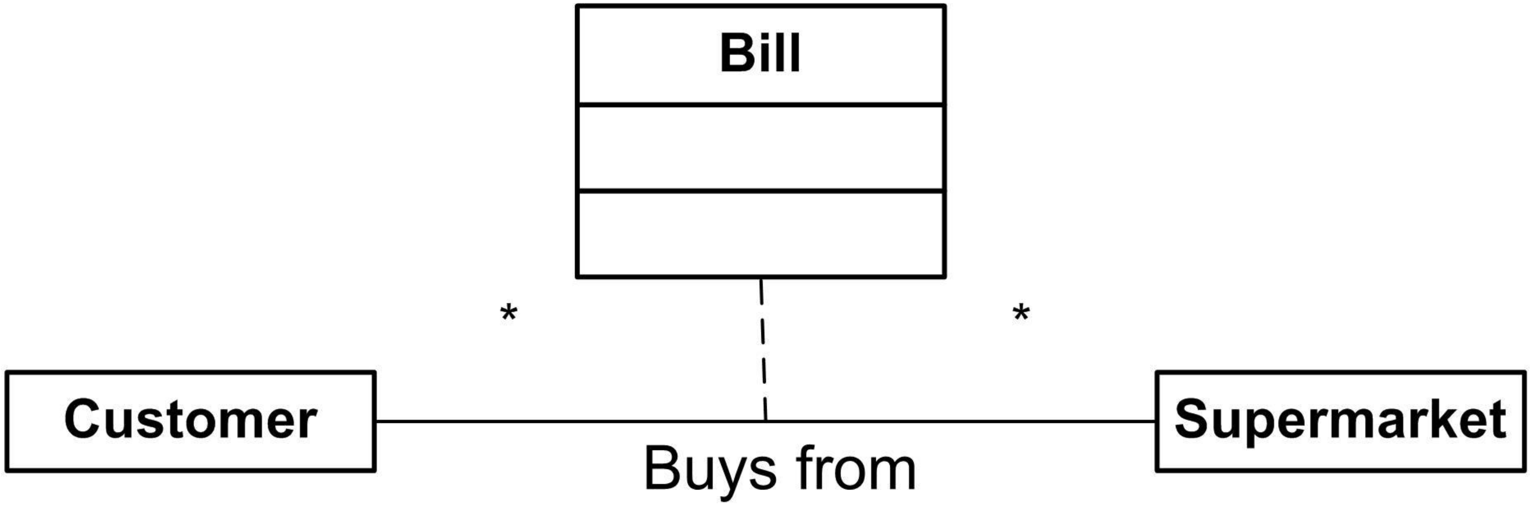
N.B. Relation cardinality is omitted for example clarity

Example of a Constraint Association

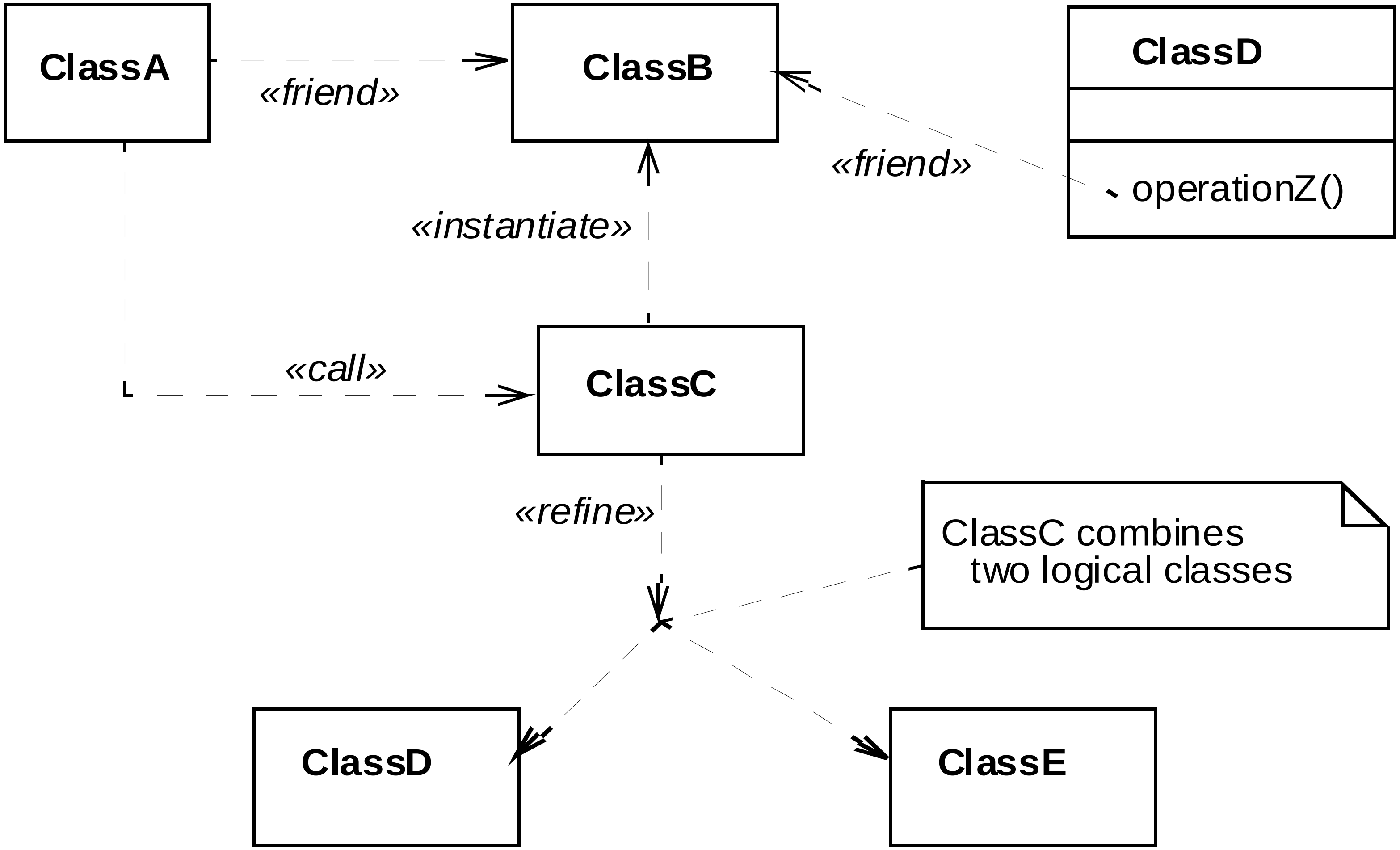


N.B. Relation cardinality is omitted for example clarity

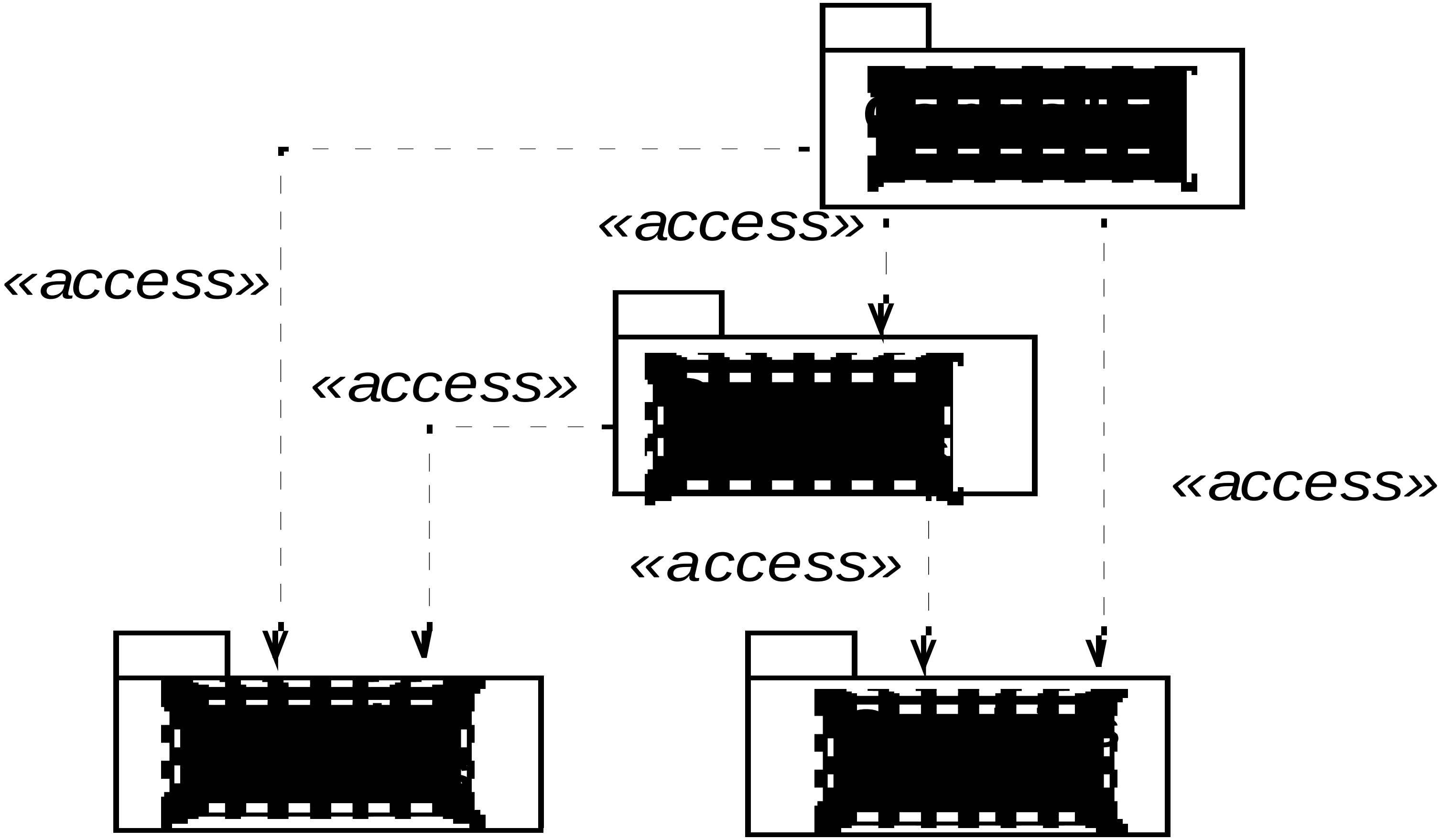
Association Class



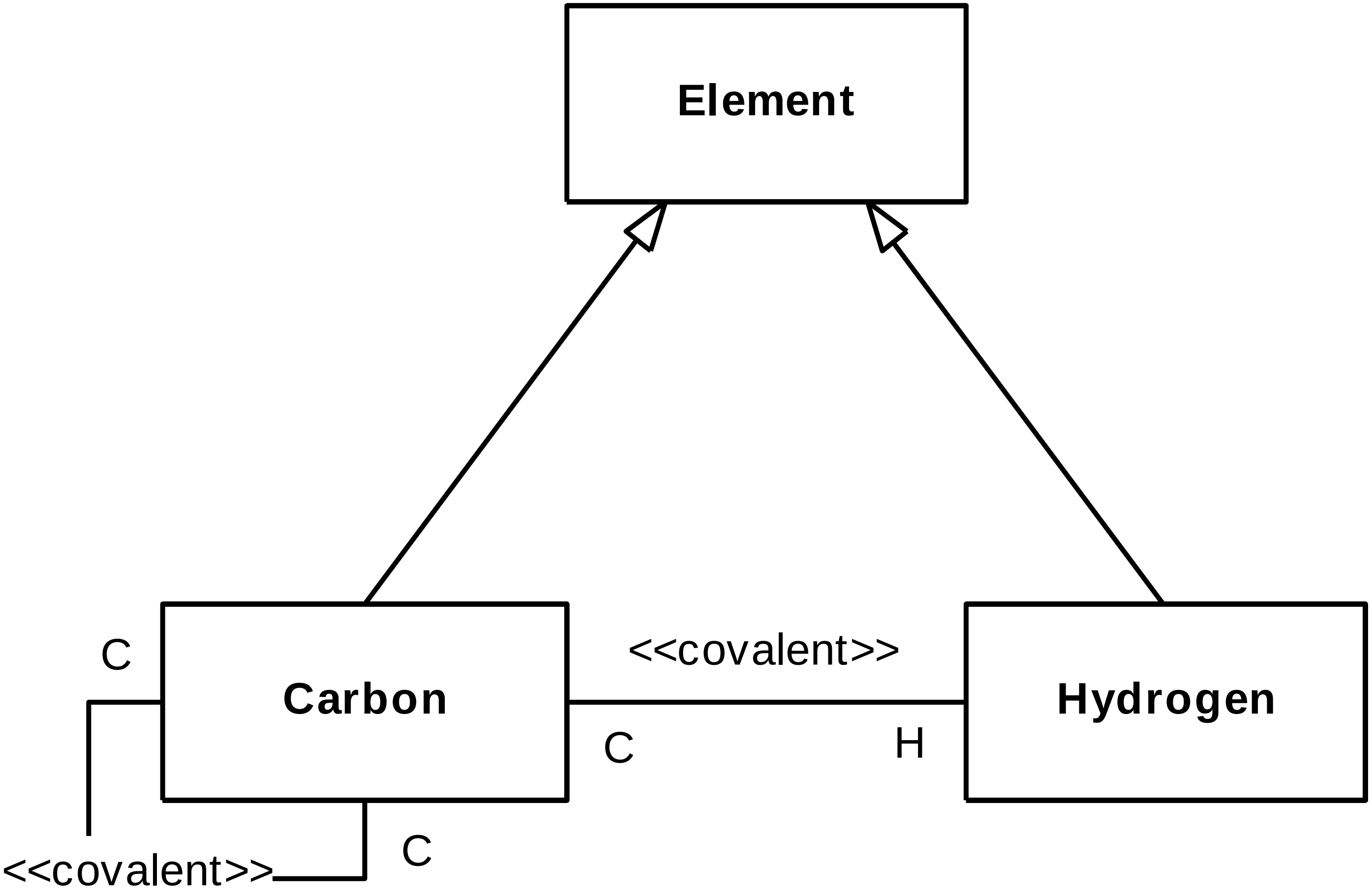
Class Dependencies



Concrete Dependency Example

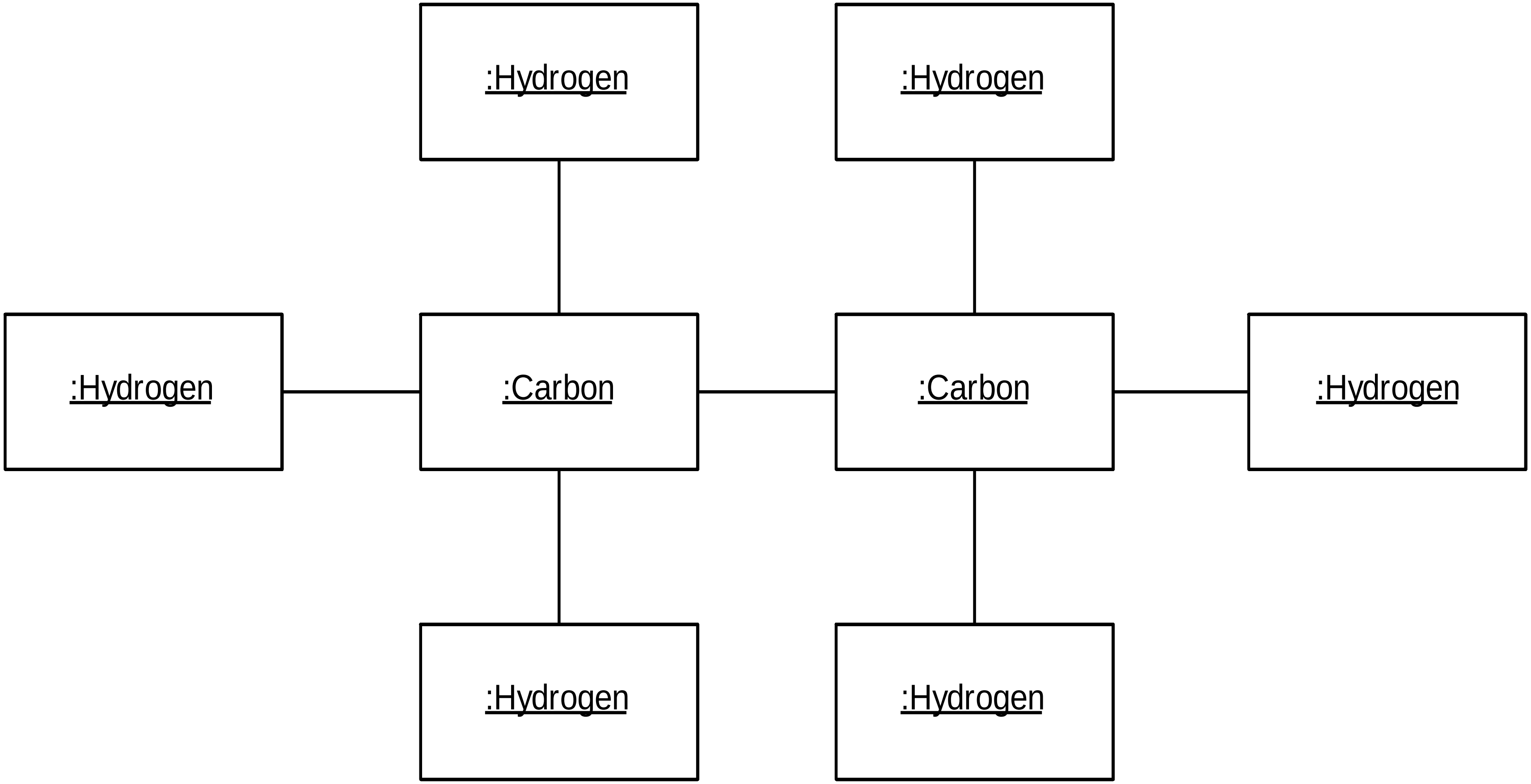


Class Diagram Example

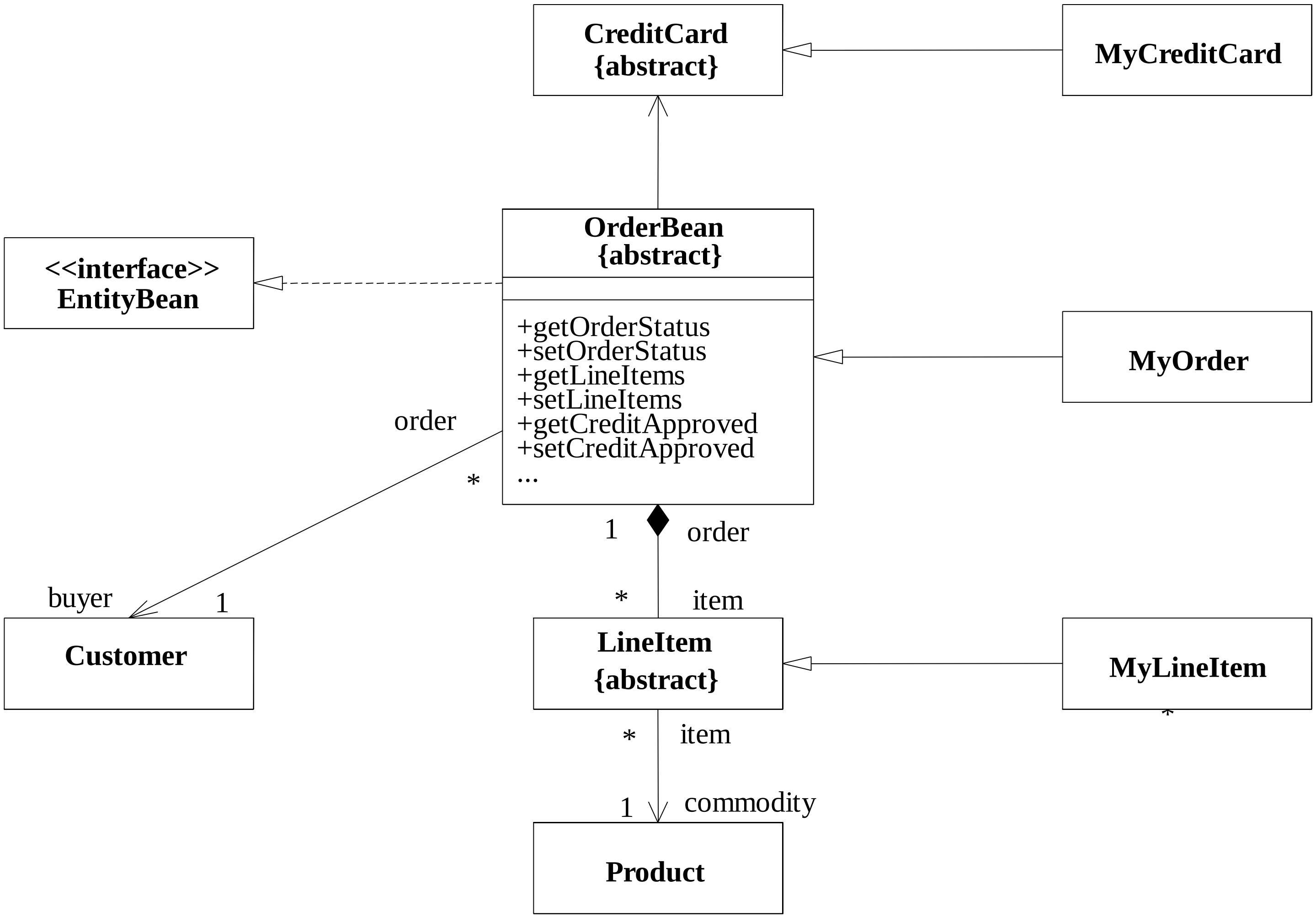


Instantiation of Class Diagram

(in previous slide)

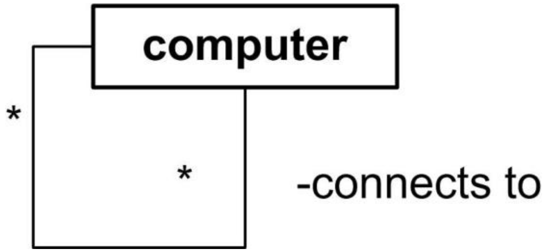


Another Class Diagram Example

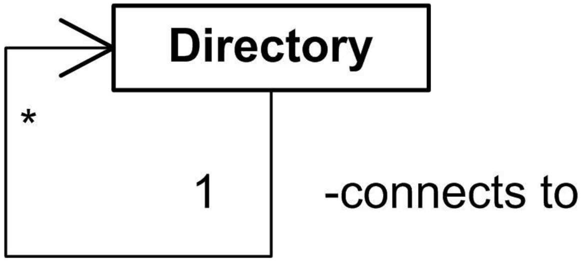


Try This Yourselves…

- Create a class diagram to represent a arbitrary interconnection of computers



  - Create a class diagram to represent a hierarchical directory system in any OS


Directory
*
1
-connects to