Object Oriented Programming

Comp-361 : O.O. Programming
Lecture 4

Alexandre Denault
Original notes by
Jörg Kienzle and Hans Vangheluwe
Computer Science
McGill University
Winter 2008

On my desk

- You should all have a team by now.
- For Monday, on a sheet of paper
  - Name of team members, with email
  - Name of the team (be creative, or I chose for you)
  - A number between 0 and 100
  - Course conflict with on Monday/Wednesday between 10h30 and 12h00.

Decomposition

- Divide a large tasks in smaller components.
- Easier to complete smaller components individually.

- Dividing into subproblems
  - Subproblems can be solved independently.
  - Solutions to subproblems can be combined to solve the whole problem.

Cooking supper example

Naval Battle

What are the components of Naval Battle?

Naval Battle (2)

Units

Terrain

Actions

Rules

Players

What is O.O. programming?

- A computer programming paradigm.
- Emphasizes the following
  - Abstraction
  - Information/Implementation hiding (encapsulation)
  - Modularity

Abstraction

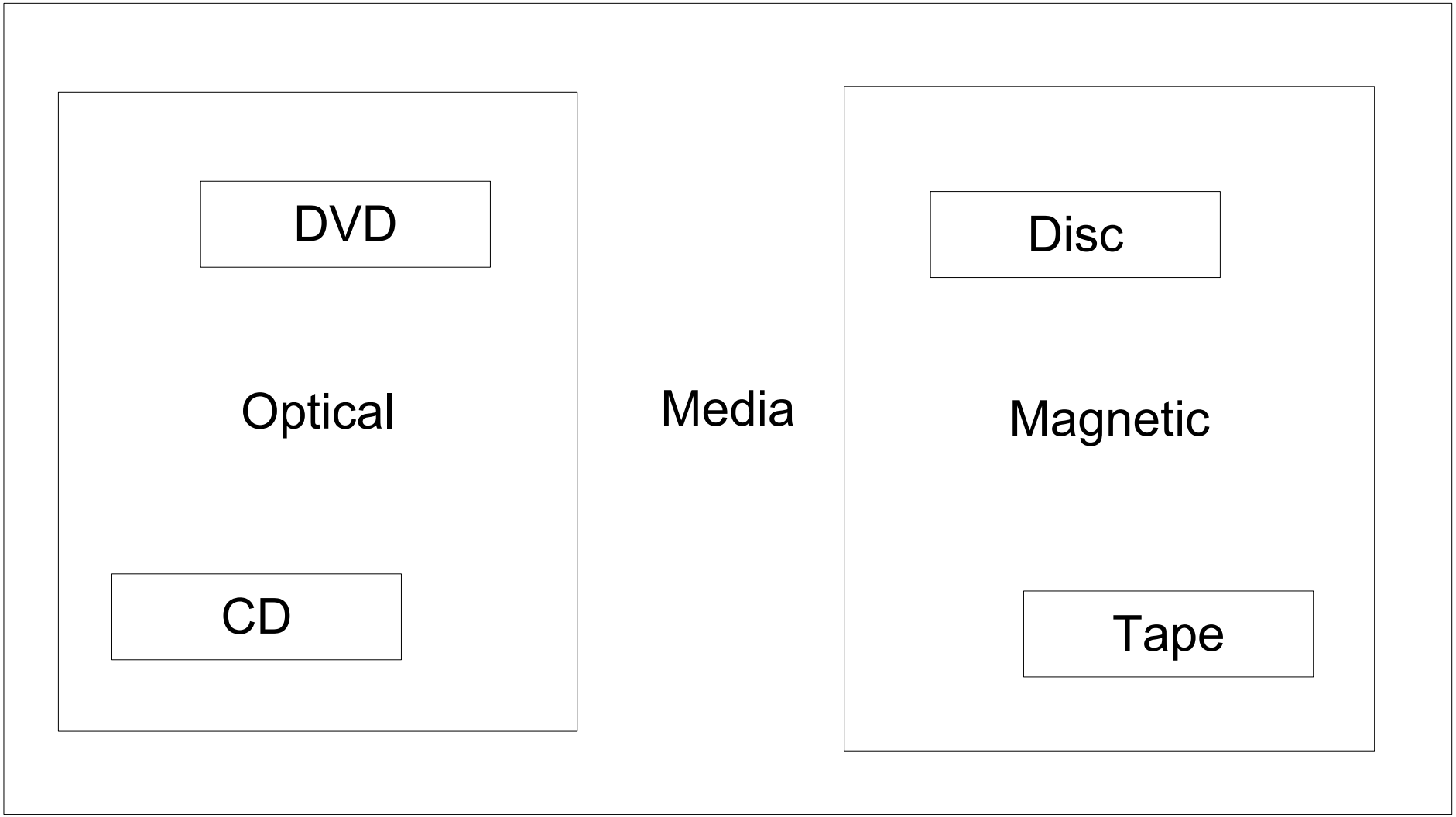
simplify different things and
treat them as the same

- Decomposition by changing the level of detail to be considered.
- Forget information and consequently to treat different things as if they were the same.
  - Files on a hard disk
  - Units in a game

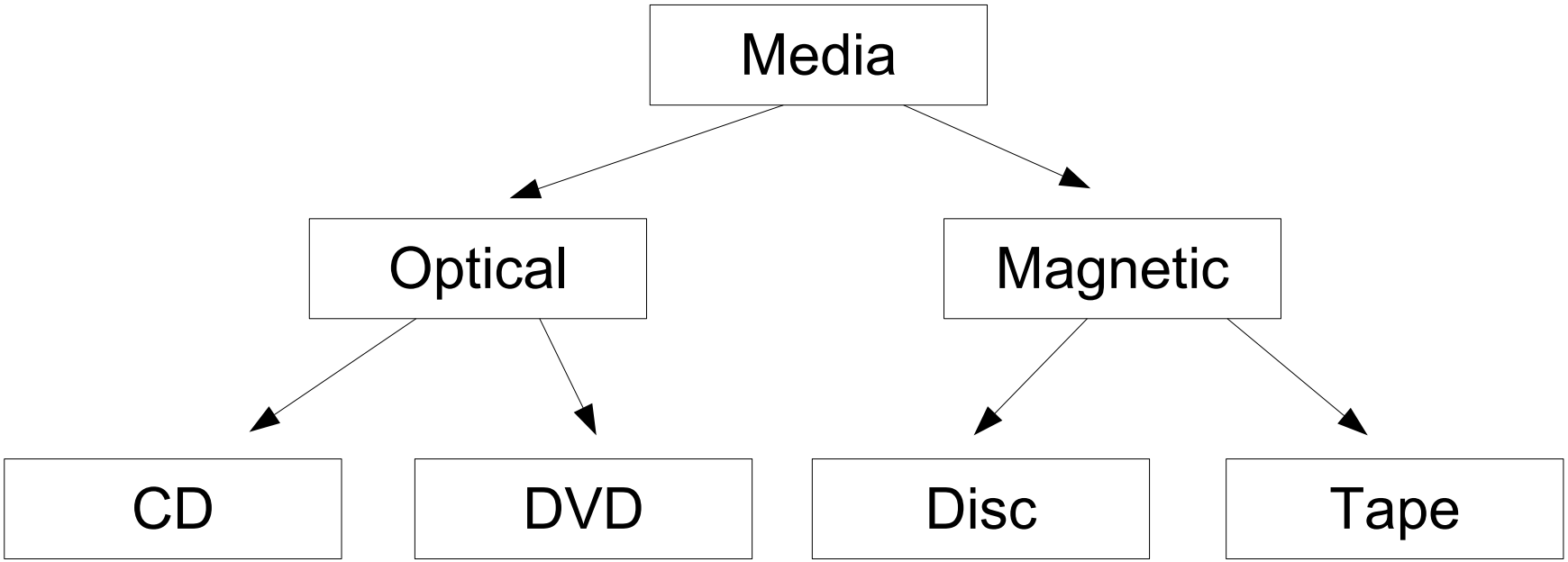
Abstraction / Type Hierarchy

- A type family is defined by a type hierarchy.
- At the top of the hierarchy is a supertype that defines behavior common to all family members.
- Other members are subtypes of this supertype.
- A hierarchy can have many levels.

Example of Abstraction Hierarchy



Example of Abstraction Hierarchy



Info. / Implementation hiding

separating implementation from interfaces

- When observing an encapsulation, we can have two point of view:
  - From the outside (public view)
  - From the inside (private view)
- The advantages of a good encapsulation is the separation of the private and public views.

Player

In a video game, how can I store the direction a player is facing?

Player

- How do I store the direction a player is facing?
  - An integer ?
    - 4 possible values : 1=North, etc
    - Values from 0 to 360 ?
  - A float ?
    - Values from 0 to 99.9 ?
  - A character ?
    - n,s,e and w ?
  - 4 booleans ?
    - north, south, east, west ?

Player

- How do I hide this from the user?
  - IsFacingNorth() : boolean
  - IsFacingSouth() : boolean
  - IsFacingEst() : boolean
  - IsFacingWest() : boolean
  - GetDegreeFacing(): int
  - GetDirectionFacing(): int

Get / Set Rule

- Never allow other class to directly access your attribute.
- Once an attribute is public, it can never be changed.
  - Ex: img.pixeldData
- Make your attributes available using get/set methods.
  - ~~this.connectionStatus~~ Bad!
  - this.getConnectionStatus() Good!

Point

```
public interface Point {
    public set(int x, int y);
    public int getX();
    public int getY();
}
```

Point

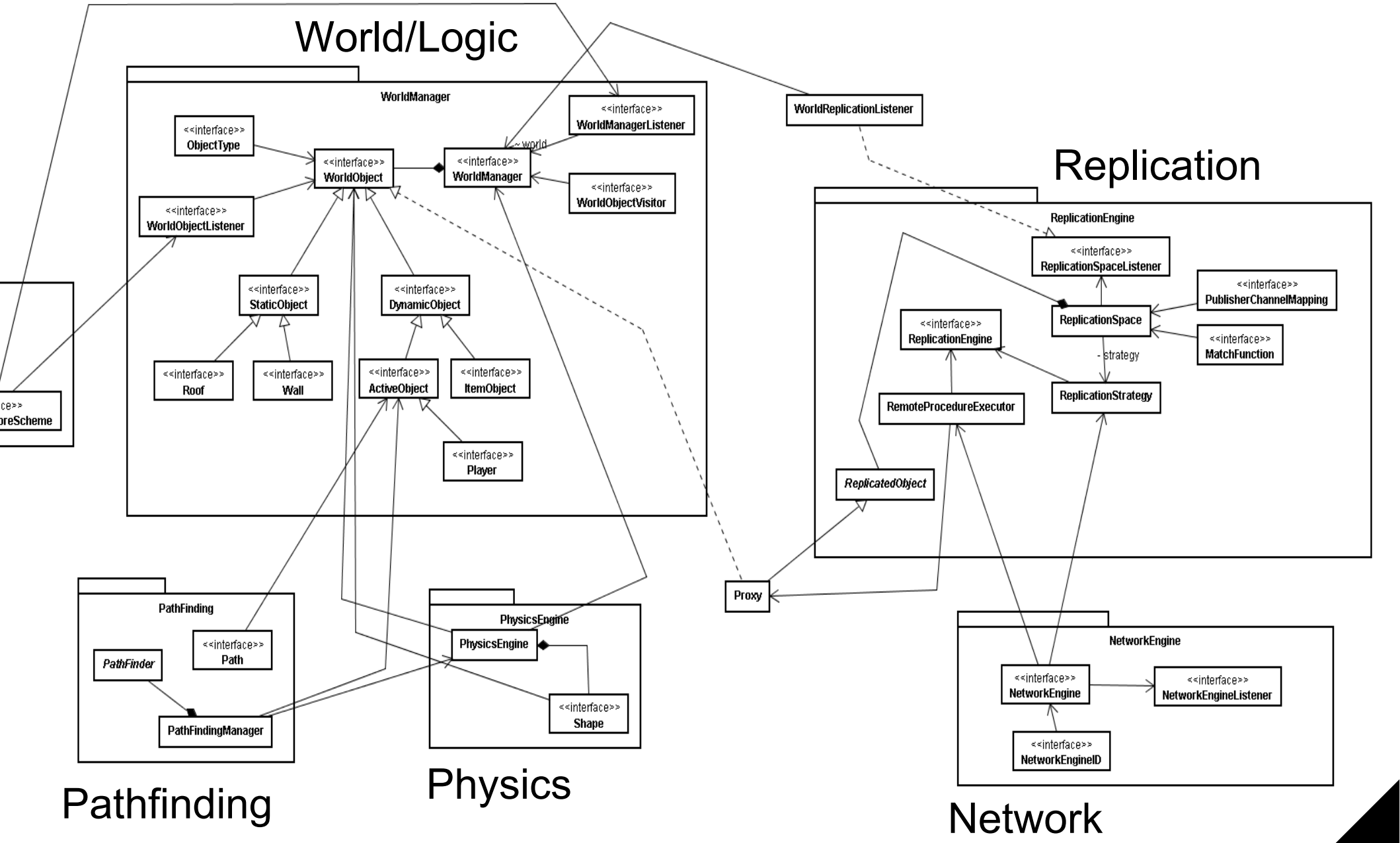
- Inside, point could be using Cartesian or Polar coordinates.
  - Cartesian coordinates are more efficient when dealing with lots of translations.
  - Polar coordinates are more efficient when dealing with lots of rotations.

Modularity

decomposing into a set of cohesive and loosely coupled units

- Break down elements into units depending on themes and concerns.
- Minimizing interaction between these units improves maintainability.

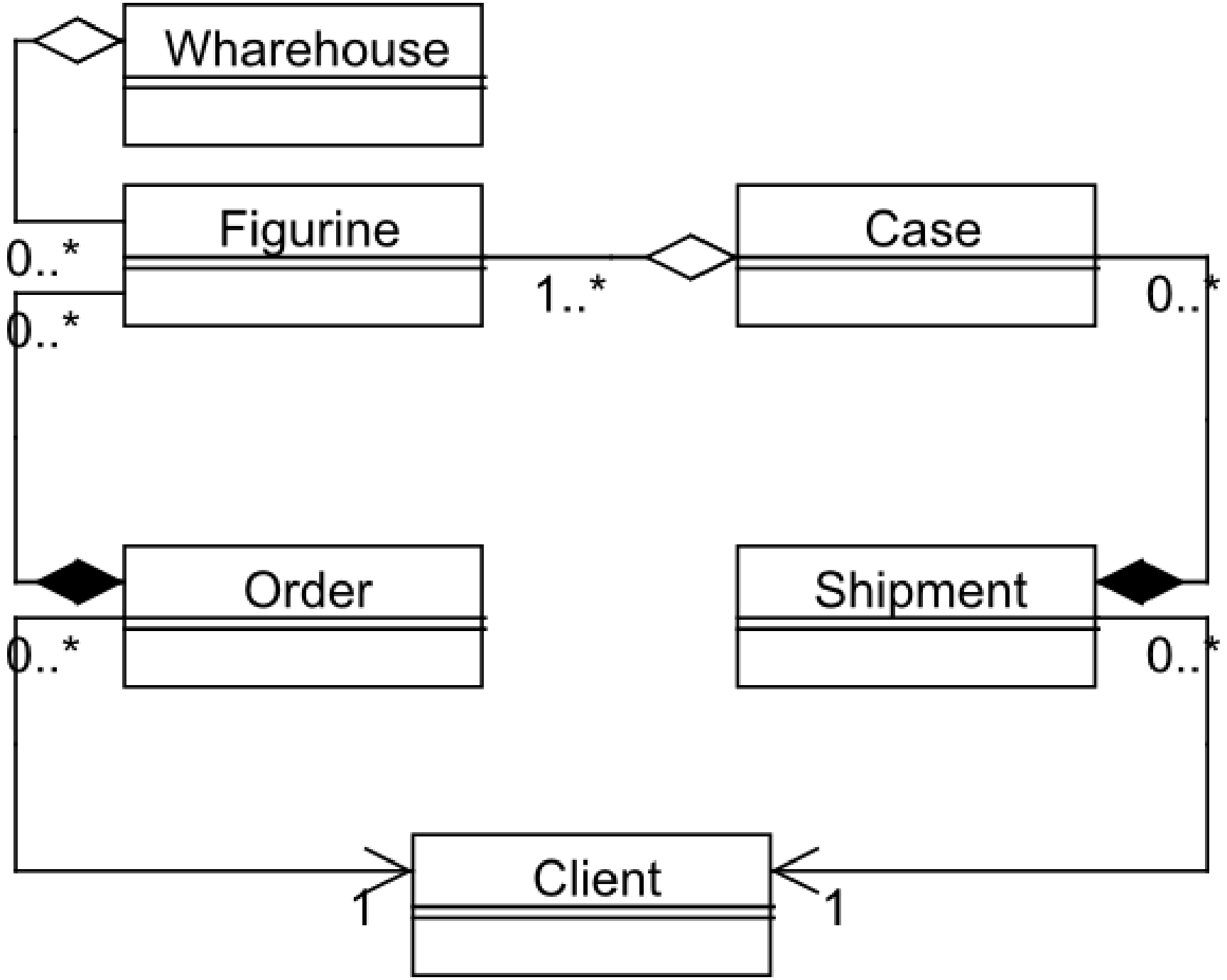
Modularity in Mammoth



Specifications in English

- Company XYZ is a manufacturing company that produces cartoon figurines for big entertainment companies.
- This company needs an inventory and tracking system.
- The inventory system keeps track of how many of each figurines is stored in each warehouse.
- Figurines are stored in cases.
- Clients order the figurines and the cases are eventually shipped to clients.

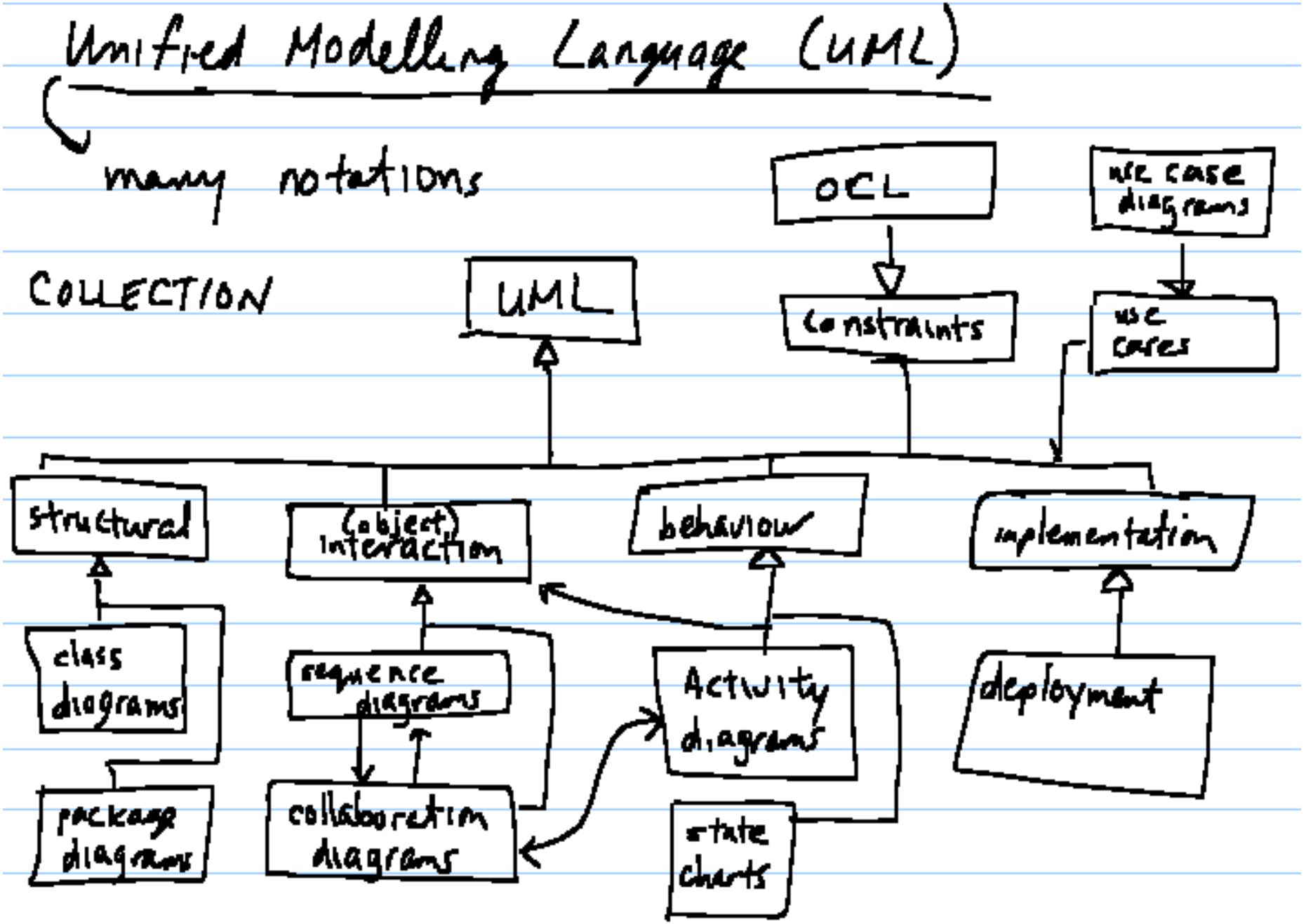
This time, in UML



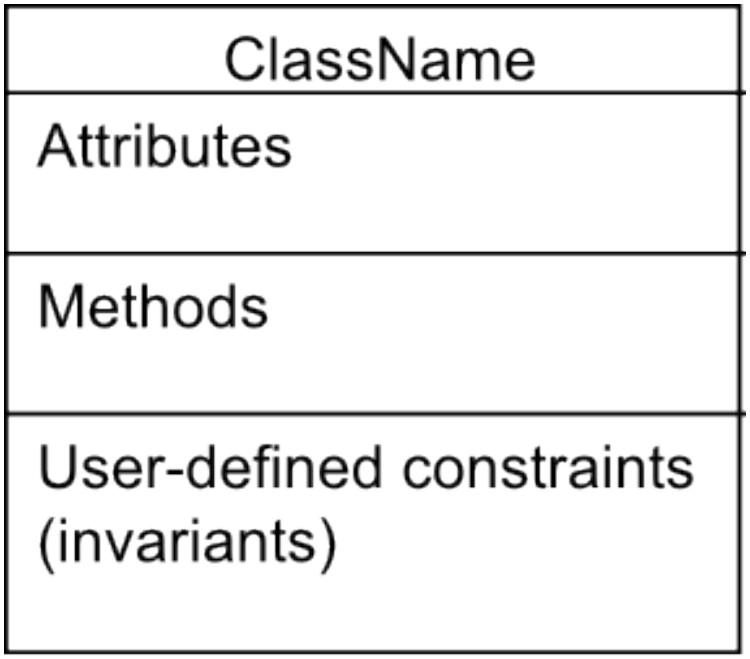
Unified Modeling Language (UML)

- A language, both graphical and textual, used throughout the entire process of project design (from requirements analysis to deployment).
- Semi-formal specification that captures structure of O.O.D.
- A standard tool for communicating a design.

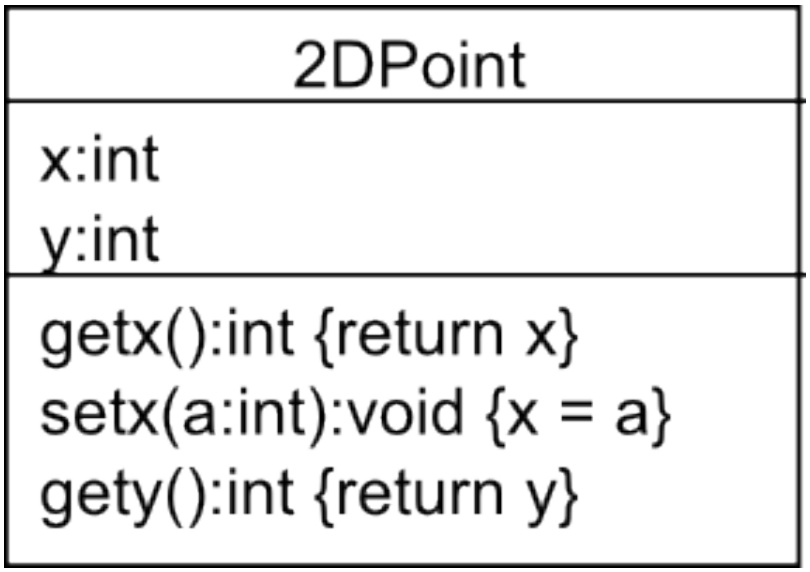
Diagrams


Unified Modelling Language (UML)
many notations
COLLECTION
UML
OCL
use case diagrams
constraints
use cases
structural
(object) interaction
behaviour
implementation
class diagrams
package diagrams
sequence diagrams
collaboration diagrams
Activity diagrams
state charts
deployment

Class Diagram



Class Example



Classes vs Objects

- Classes are static, depict the design and structure at design-time
- Objects are dynamic and are instantiated (from a class) at run-time, they have state

Attributes vs Variables

- Attributes are considered at design-time, are some abstractly defined property
- Variables are considered at implementation-time, are concretely defined properties

Objects