Object Oriented Programming using Python (I)

Lecture(5)

UML (Unified Modeling Language)

Prepared by: Ahmed Eskander Mezher

University of Information Technology and Communications

College of Business Informatics

What is UML?

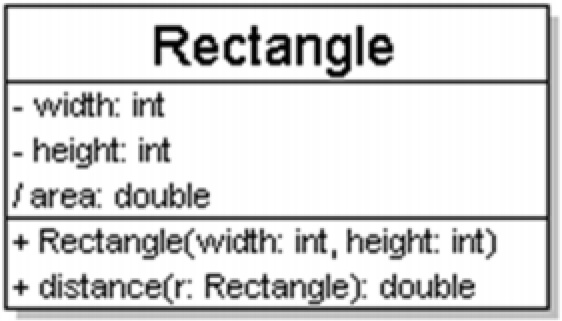
A **UML diagram** is a **diagram** based on the **UML** (Unified Modeling Language) with the purpose of visually representing a system along with its main actors, roles, actions, artifacts or classes, in order to better understand, alter, maintain, or document information about the system.

UML class diagram

UML class diagram: a picture of – the classes in an OO system and their attributes and methods

•connections between the classes

• that interact or inherit from each other



Rectangle

- width: int
- height: int
/ area: double

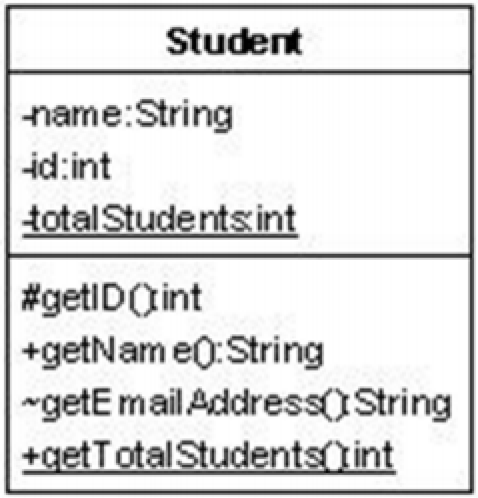
+ Rectangle(width: int, height: int)
+ distance(r: Rectangle): double

Description :

class name in top of box

• attributes – should include all fields of the object

• operations / methods ()



Student

-name:String
-id:int
+totalStudents:int

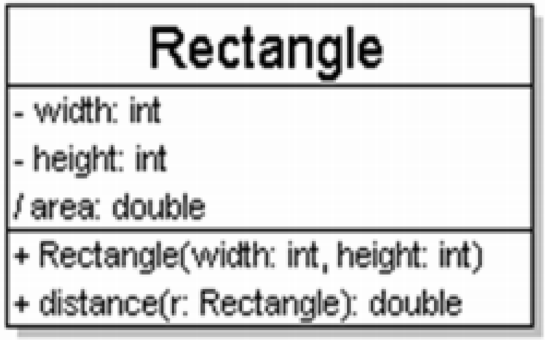
#getID():int
+getName():String
~getEmailAddress():String
+getTotalStudents():int

Class attributes (= fields)

- attributes (fields, instance variables)
  - *visibility name* : *type* [*count*] = *default_value*
  - visibility:

+	public
#	protected
-	private
~	package (default)
/	derived

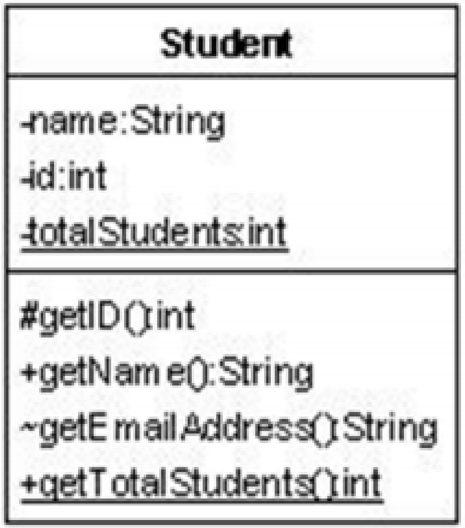
  - underline static attributes
  - **derived attribute**: not stored, but can be computed from other attribute values
  - attribute example:
    - balance : double = 0.00



Rectangle

- width: int
- height: int
/ area: double

+ Rectangle(width: int, height: int)
+ distance(r: Rectangle): double



Student

-name:String
-id:int
-totalStudents:int

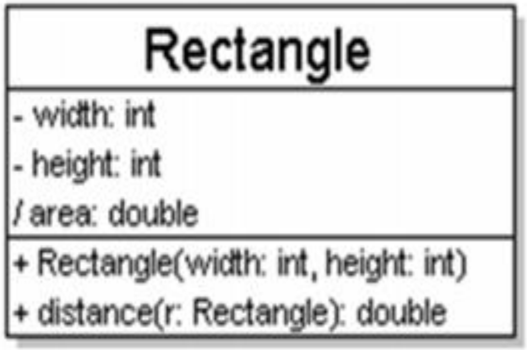
#getID():int
+getName():String
~getEmailAddress():String
+getTotalStudents():int

Class operations / methods

- operations / methods
  - *visibility name* (*parameters*) : *return_type*
  - visibility:

+	public
#	protected
-	private
~	package (default)

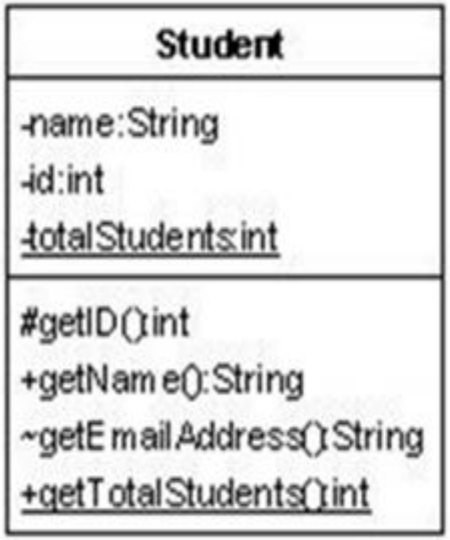
  - underline static methods
  - parameter types listed as (name: type)
  - method example:
    + distance(p1: Point, p2: Point): double



Rectangle

- width: int
- height: int
/ area: double

+ Rectangle(width: int, height: int)
+ distance(r: Rectangle): double



Student

-name:String
-id:int
-totalStudents:int

#getID():int
+getName():String
~getEmailAddress():String
+getTotalStudents():int

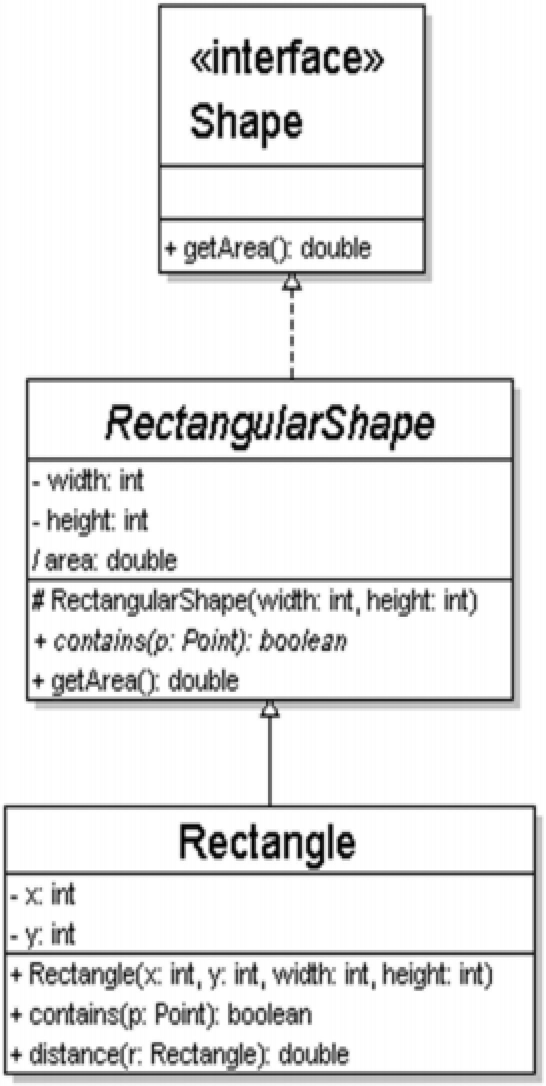
Relationships between classes

- **generalization**: an inheritance relationship
  - inheritance between classes
  - interface implementation

- **association**: a usage relationship
  - dependency
  - aggregation
  - composition

Generalization (inheritance) relationships

- hierarchies drawn top-down
- arrows point upward to parent
- line/arrow styles indicate whether parent is a(n):
  - <u>class</u>:
    solid line, black arrow
  - <u>abstract class</u>:
    solid line, white arrow
  - <u>interface</u>:
    dashed line, white arrow



«interface»
Shape

+ getArea(): double

RectangularShape

- width: int
- height: int
/ area: double

# RectangularShape(width: int, height: int)
+ *contains(p: Point): boolean*
+ getArea(): double

Rectangle

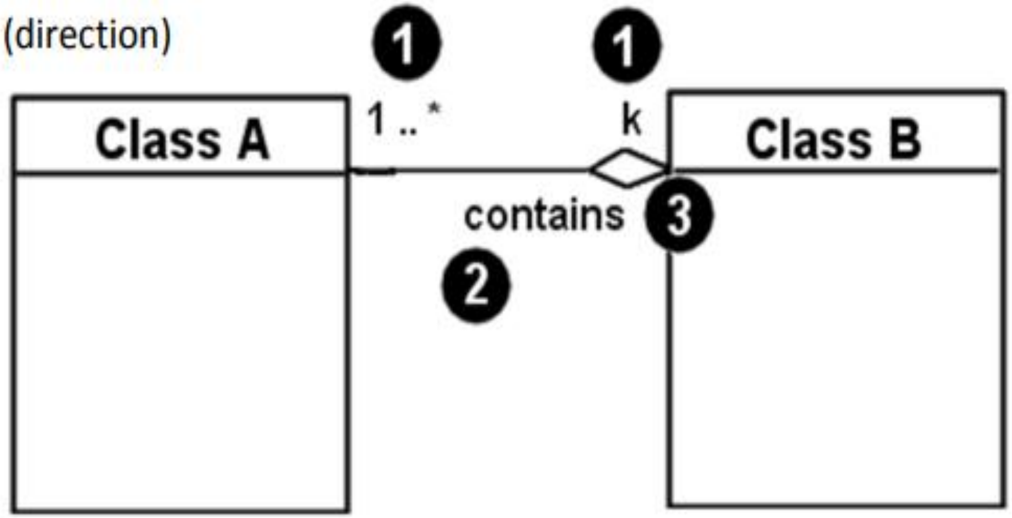
- x: int
- y: int

+ Rectangle(x: int, y: int, width: int, height: int)
+ contains(p: Point): boolean
+ distance(r: Rectangle): double

2

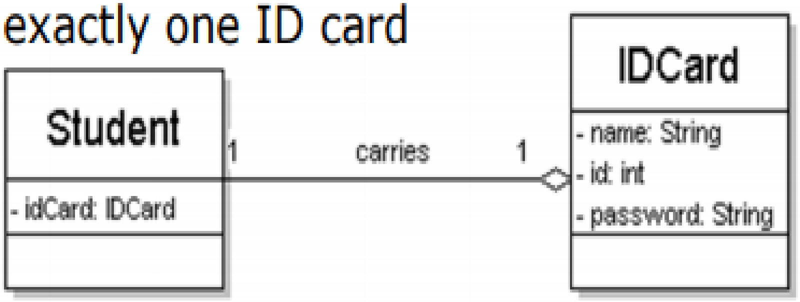
Associational relationships

- associational (usage) relationships
  1. multiplicity (how many are used)
     - * ⇒ 0, 1, or more
     - 1 ⇒ 1 exactly
     - 2..4 ⇒ between 2 and 4, inclusive
     - 3..* ⇒ 3 or more (also written as "3..")
  2. name (what relationship the objects have)
  3. navigability (direction)

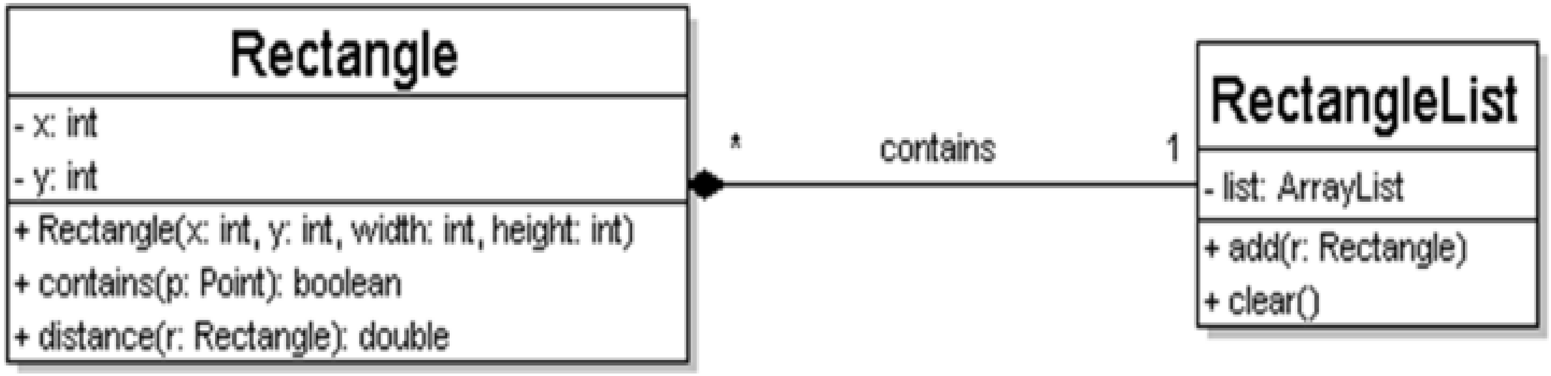


Association relations

- one-to-one
  - each student must carry exactly one ID card

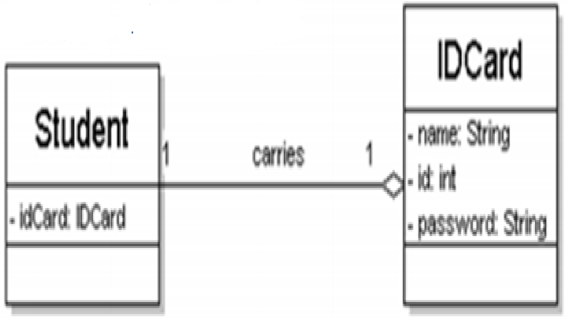


- one-to-many
  - one rectangle list can contain many rectangles

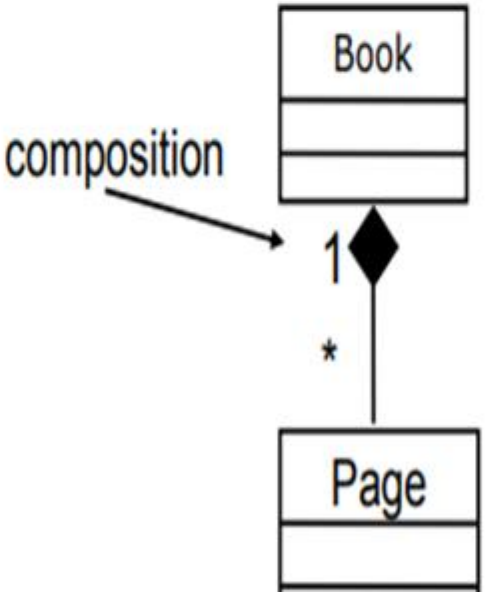

Rectangle
- x: int
- y: int
+ Rectangle(x: int, y: int, width: int, height: int)
+ contains(p: Point): boolean
+ distance(r: Rectangle): double
*
contains
1
RectangleList
- list: ArrayList
+ add(r: Rectangle)
+ clear()

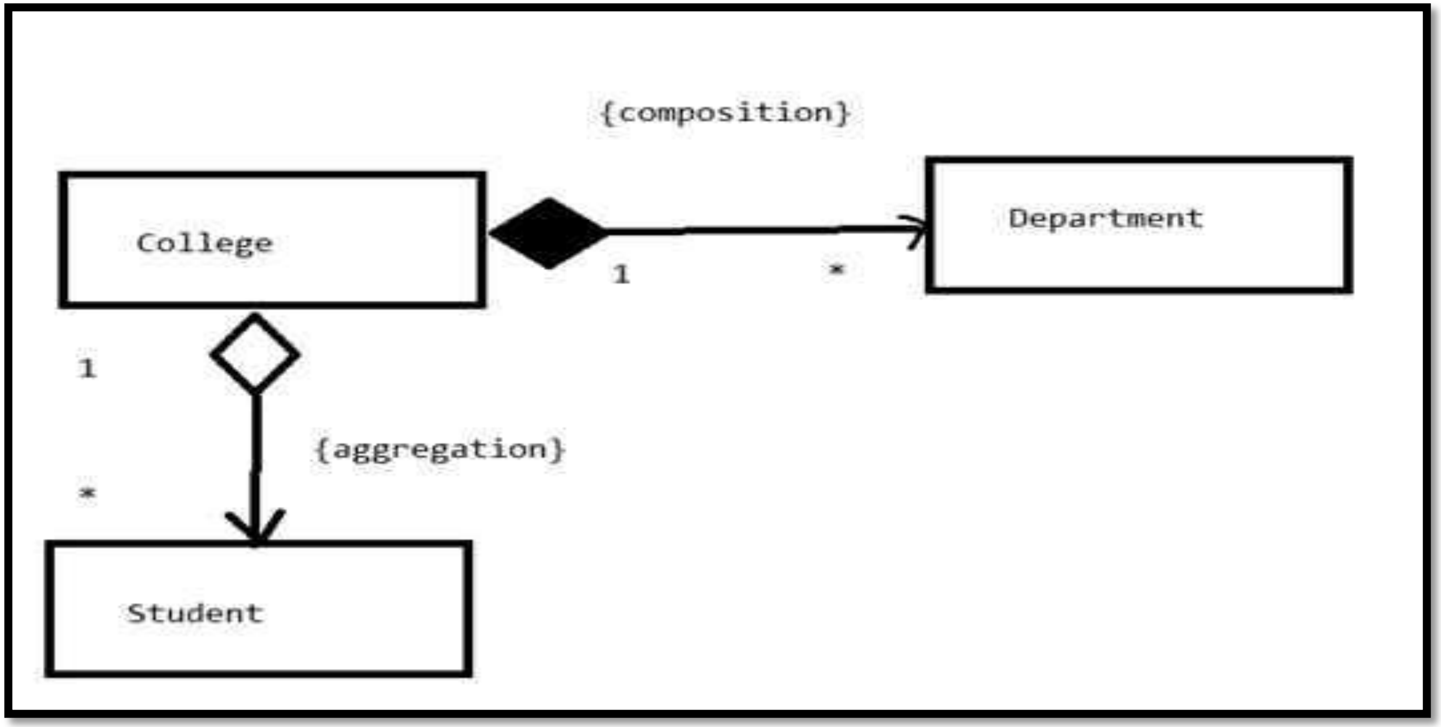
Association types

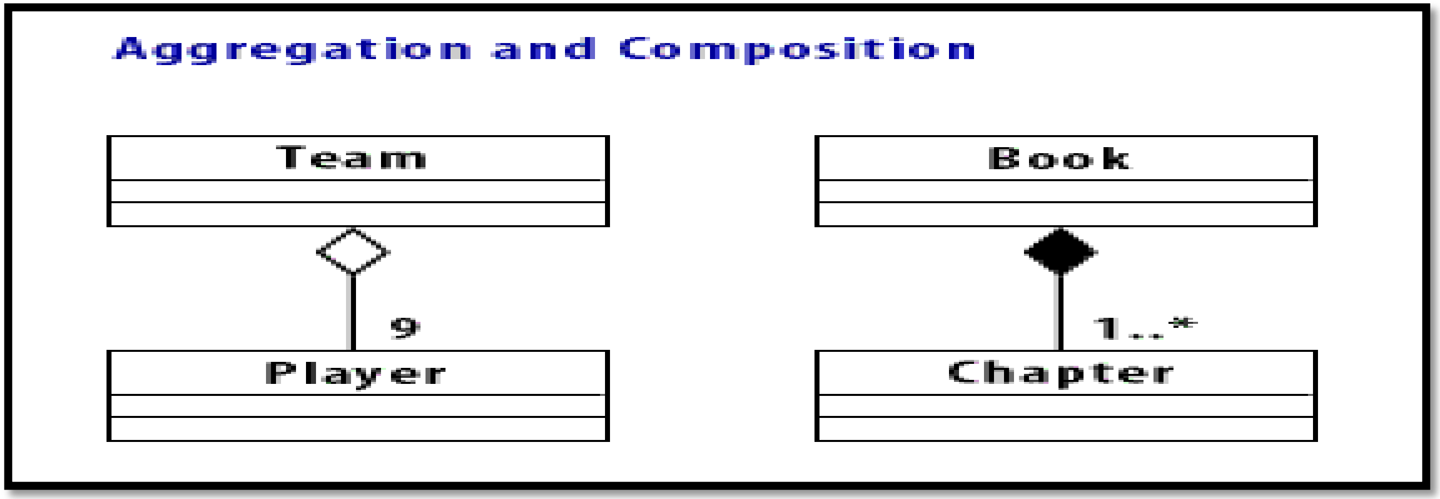
- **aggregation**: “is part of”
  - symbolized by a clear white diamond

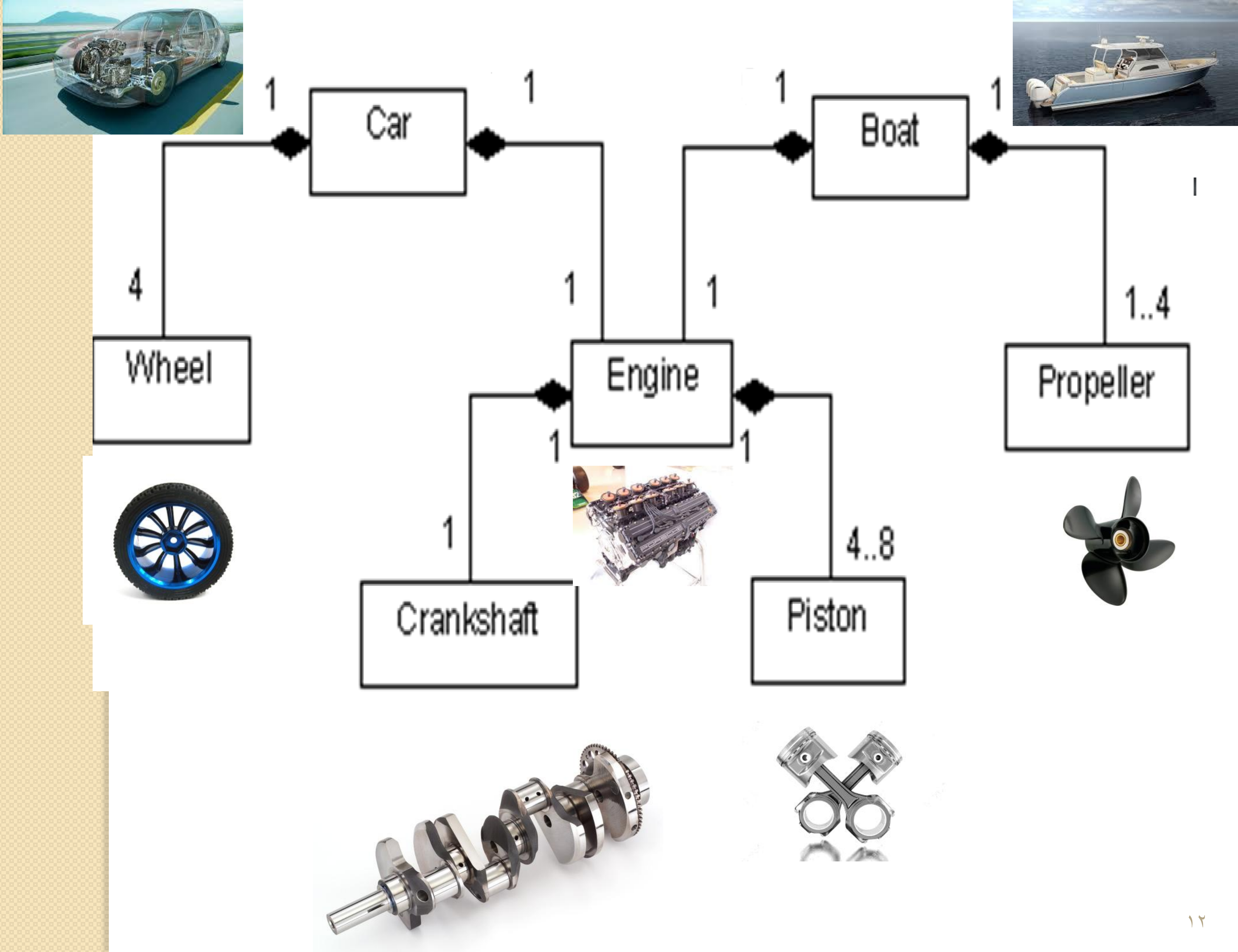


- **composition**: “is entirely made of”
  - stronger version of aggregation
  - the parts live and die with the whole
  - symbolized by a black diamond

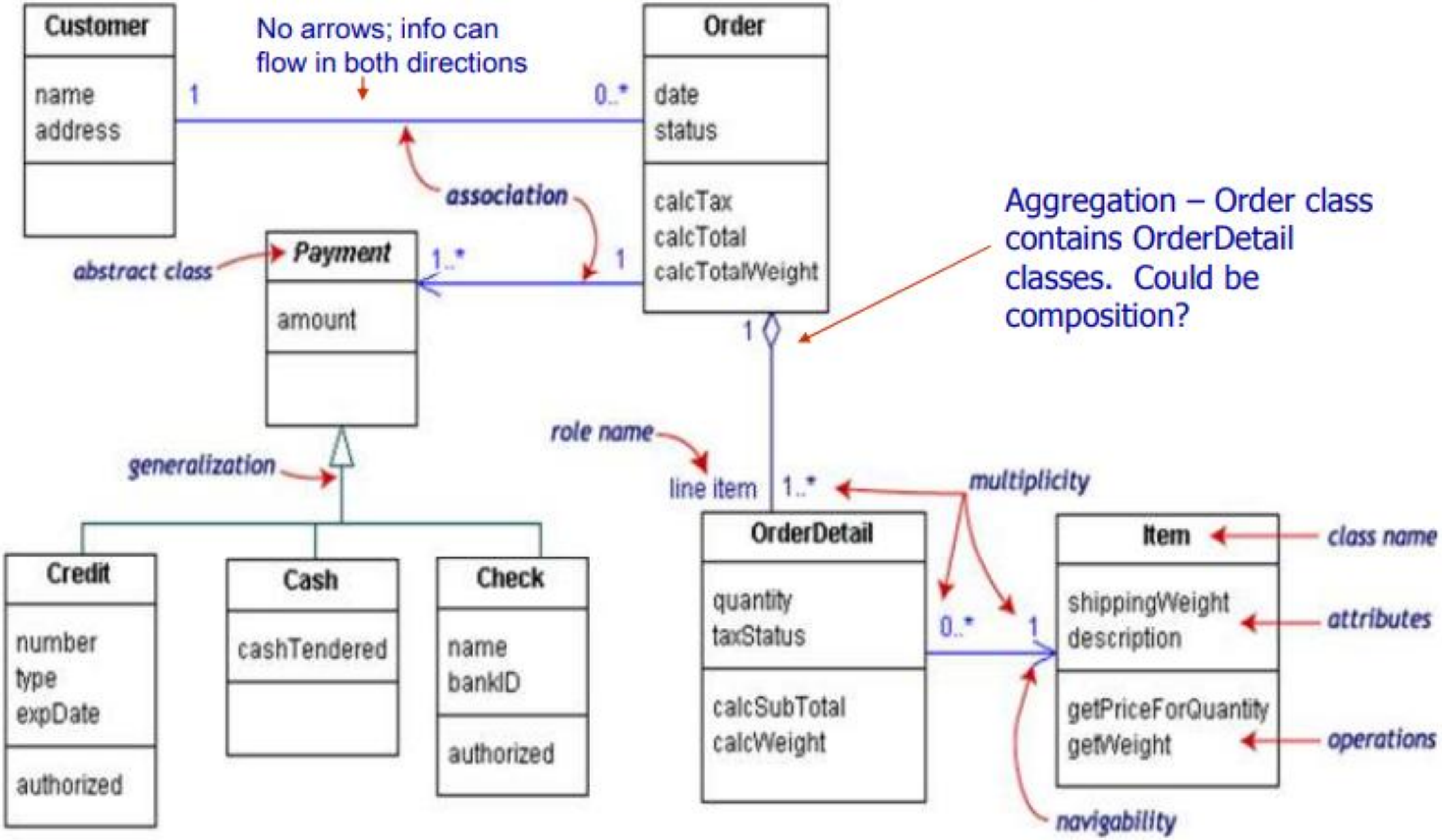



{composition}
College
Department
1
*
1
{aggregation}
*
Student


Aggregation and Composition
Team
Book
9
1..*
Player
Chapter


1
Car
1
1
Boat
1
4
1
1
1..4
Wheel
Engine
Propeller
1
1
1
4..8
Crankshaft
Piston

Class diagram example



No arrows; info can flow in both directions

Aggregation – Order class contains OrderDetail classes. Could be composition?