Lect #6 UML: Class Digrams

Dr Hisham Salah

What is UML?

- UML stands for Unified Modelling Language.
- An industry standard modelling language for object-oriented software engineering.
- Developed in the mid-1990’s and standardized in 1997 (UML 1.1).

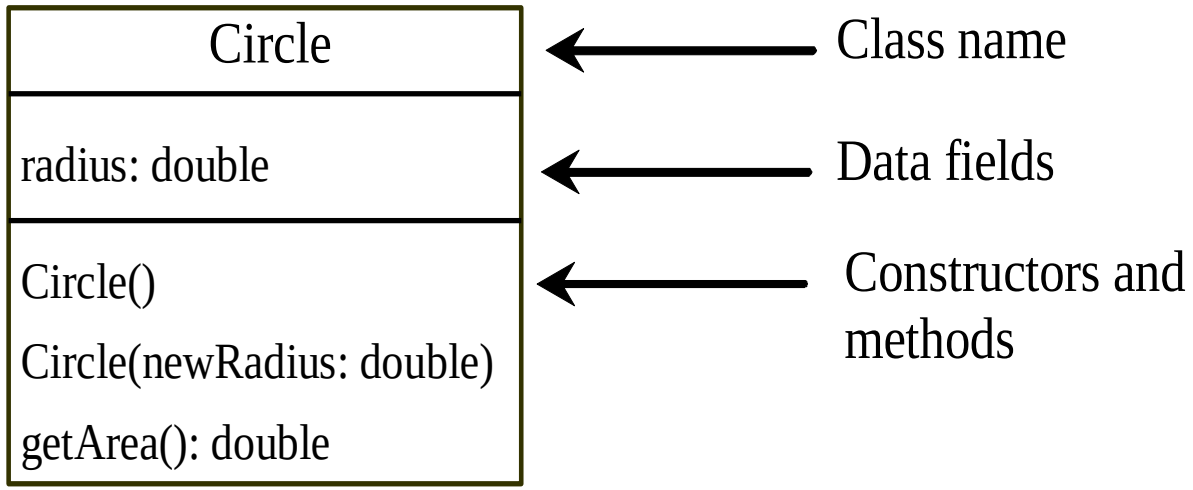
Why UML?

- **S.W. Design**: specifying the structure of how a software system will be written and function, without actually writing the complete implementation
- A transition from **"what"** the system must do, to **"how"** the system will do it
  1. What classes will we need to implement a system that meets our requirements?
  2. What fields and methods will each class have?
  3. How will the classes interact with each other?

UML Diagrams

- UML includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems.
- UML Diagrams include but are not limited to the following:
  - **Class diagrams**
  - Object diagrams
  - Sequence diagrams
  - State chart diagrams

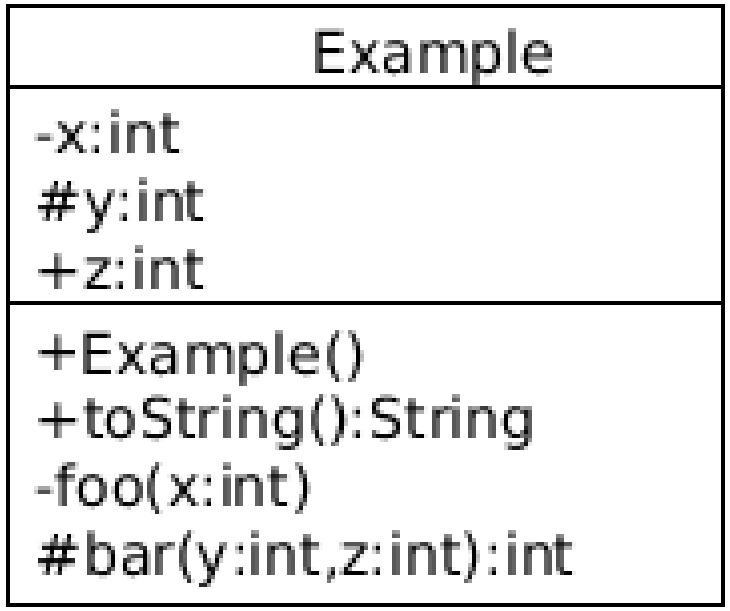
UML Class Diagram



Visibility Modifiers

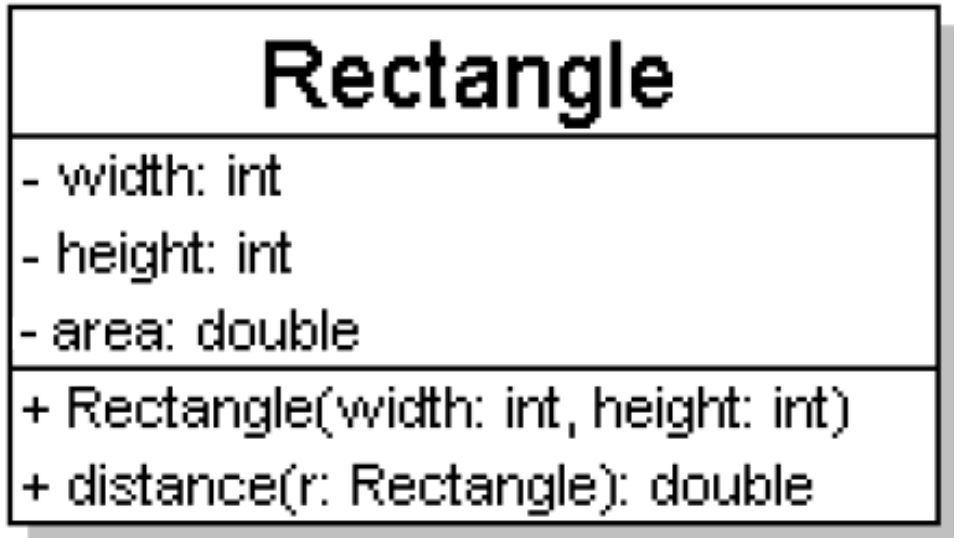
- The sign + indicates public
- The sign – indicates private
- The sign # indicates protected
- The sign ~ default

Example 1



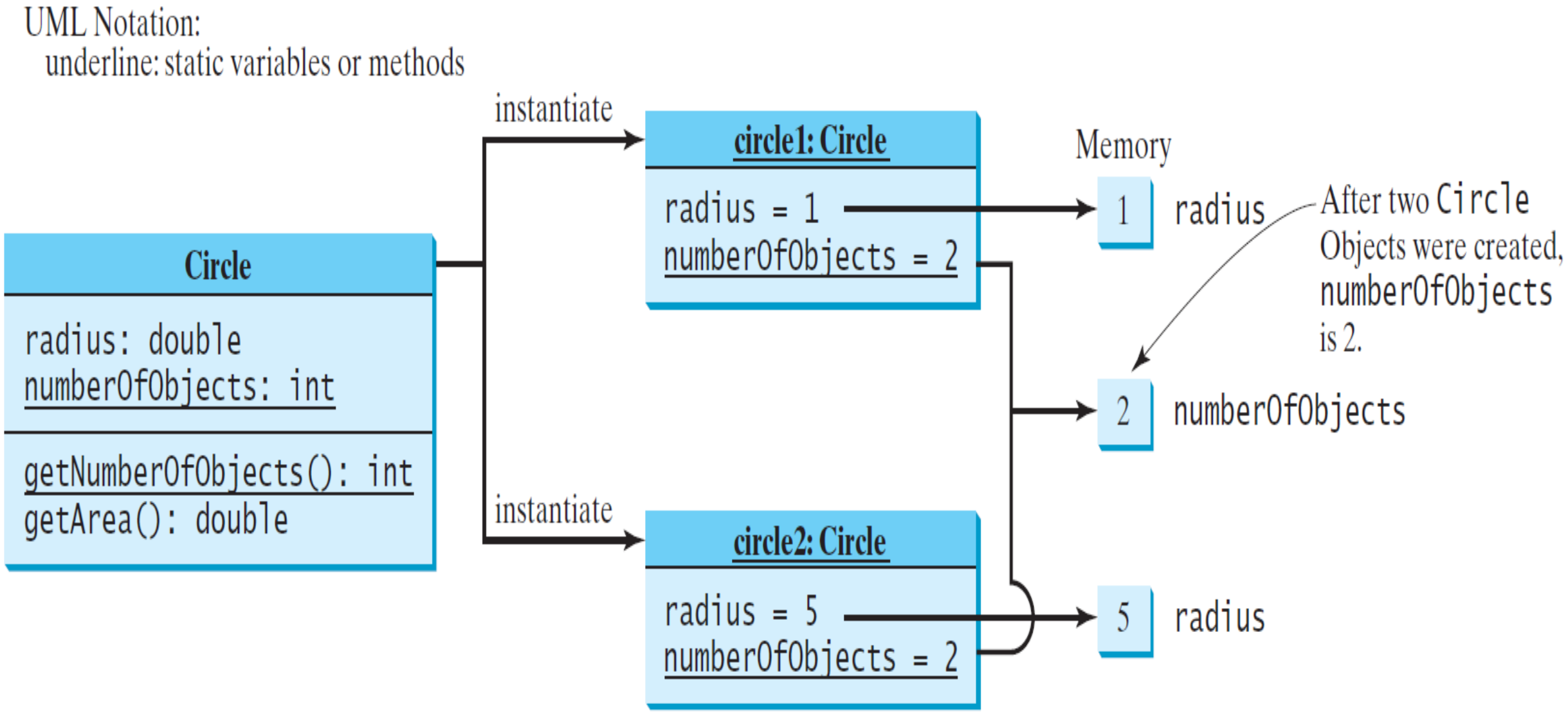

```
public class Example {
   private int x;
   protected int y;
   public int z;
   public Example() { ... }
   public String toString() { ... }
   private void foo(int x) { ... }
   protected int bar(int y, int z) { ... }
}
```

Example 2

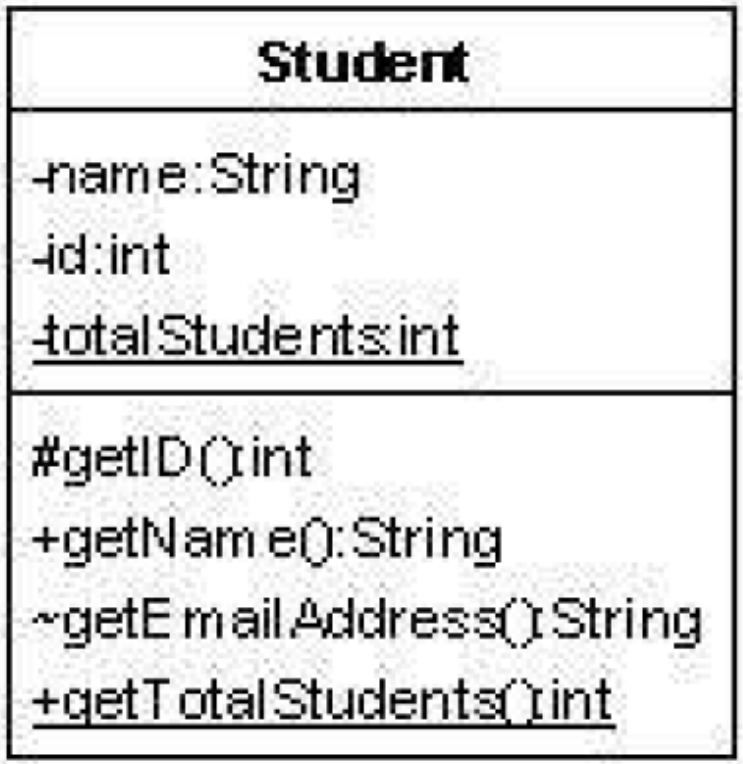


What is the corresponding java code?

Static Variables, and Methods



Example 1




```
class Student{
      …
      private static int totalStudents;
      ….
      public static int getTotalStudents()
            {
            …
            }
 ……….……….
}
```

Relationships

- Generalization
- Dependency
- Aggregation
- Composition

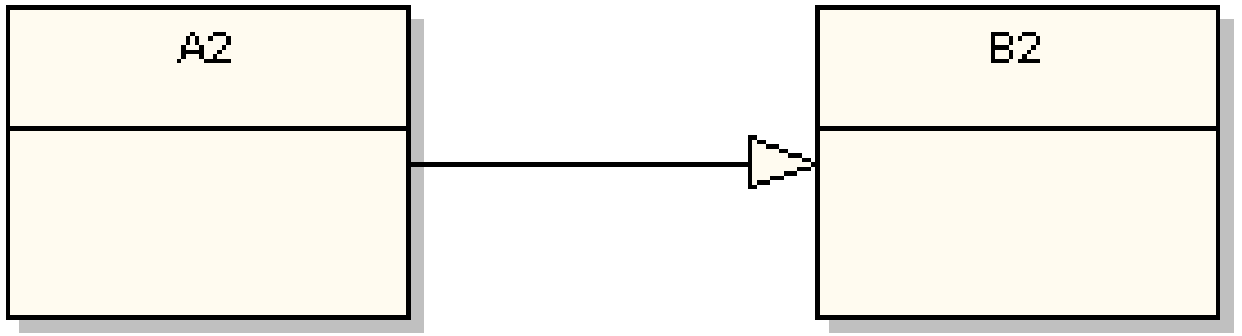
Relationships

- Generalization
  - Inheritance
    - “Is-a” relationship
    - Between classes or classes and interfaces
    - One class/interface is a specialized version of another.
    - Example, Honda is a type of car. So the class Honda would have a generalization relationship with the class car.

Generalization: between 2 classes

"**Is-a**" relationship

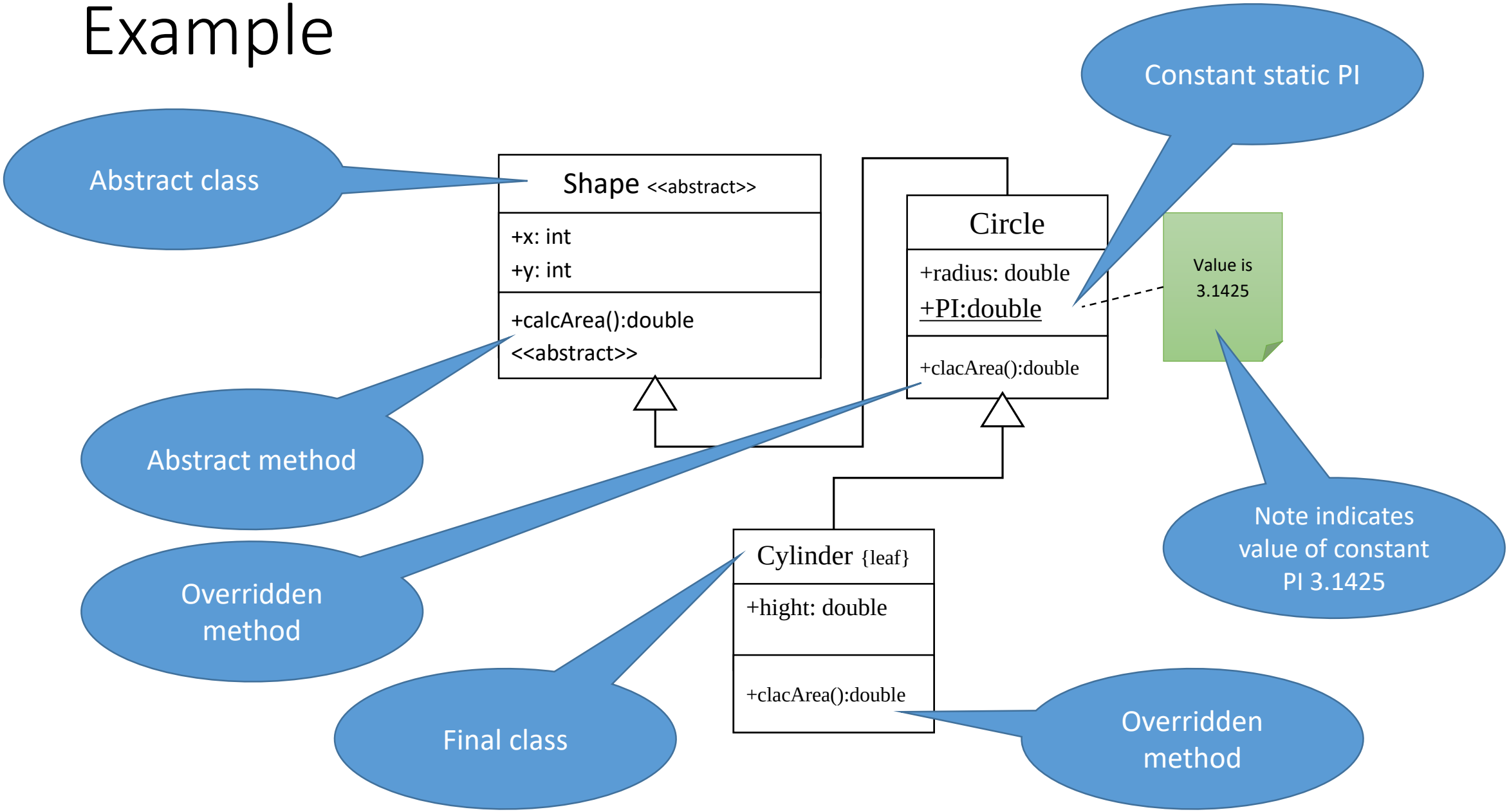
```
class A2 extends B2{
.....

}
```



final and **abstract** modifiers

- A final class and a final method both are represented by the stereotype **{leaf}**
- A final data member (constant) is written in capital letters
- Abstract class or abstract method are represented in italic style
  - A better way is using the stereotype **<<abstract>>**

Example


Abstract class
Shape <<abstract>>
+x: int
+y: int
+calcArea():double
<<abstract>>
Abstract method
Overridden method
Circle
+radius: double
+PI:double
+clacArea():double
Constant static PI
Value is 3.1425
Note indicates value of constant PI 3.1425
Cylinder {leaf}
+hight: double
+clacArea():double
Final class
Overridden method

Generalization : class and an interface

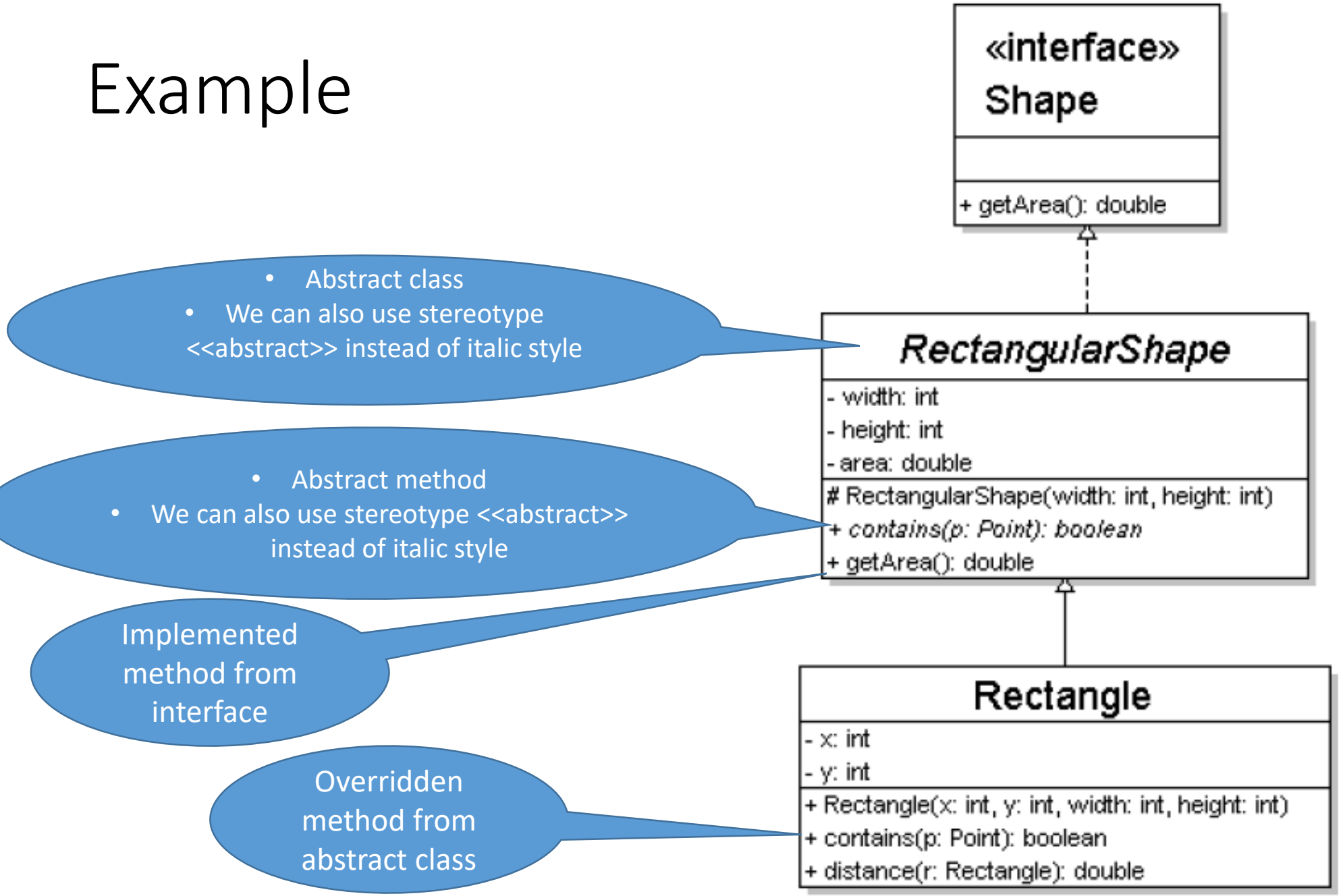
- Implements
  - “Kind-of” relationship
  - Known as realization
  - Between classes and interfaces

interfaces

- Are represented by the stereotype **<<interface>>**
- Example


```
public interface FooListener {
  public void foo();
}
```

Example


«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
• Abstract class
• We can also use stereotype <<abstract>> instead of italic style
• Abstract method
• We can also use stereotype <<abstract>> instead of italic style
Implemented method from interface
Overridden method from abstract class

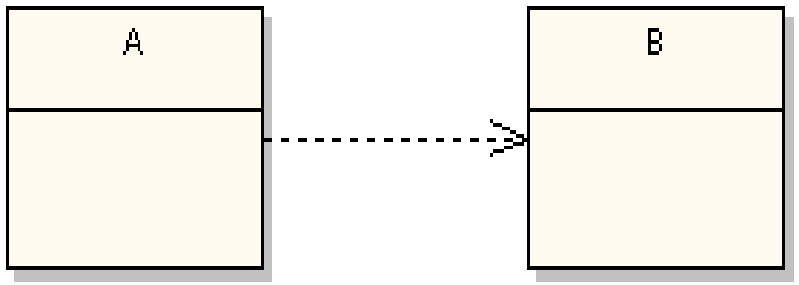
Relationships cont.

- Generalization √
- Dependency
- Aggregation
- Composition

Dependency

- Means the class at the source end of the relationship has some sort of dependency on the class at the target (arrowhead) end of the relationship
- Used to model temporarily-usage between things
- Symboled by a dashed arrow
- Do not over use in your design (very low importance)
- Example:

```
import B;
public class A {
                public void method1(B b) { // . . . }

}
```

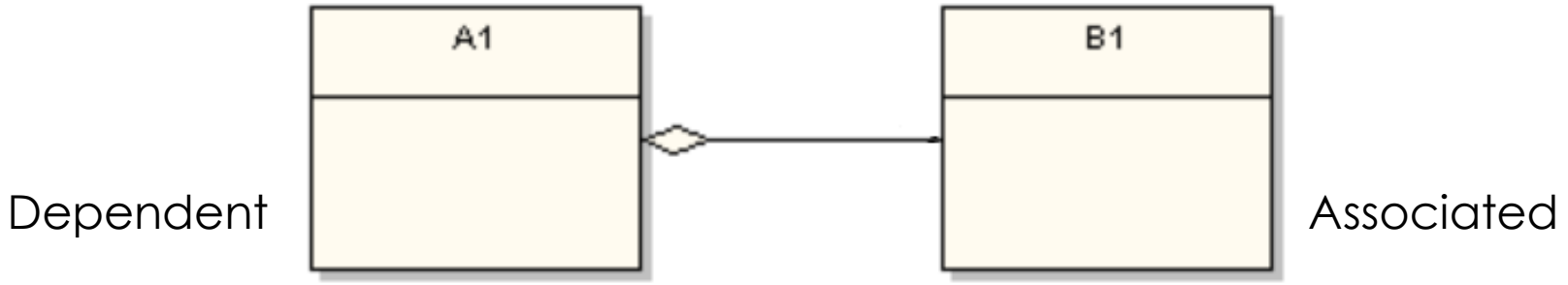


Relationships cont.

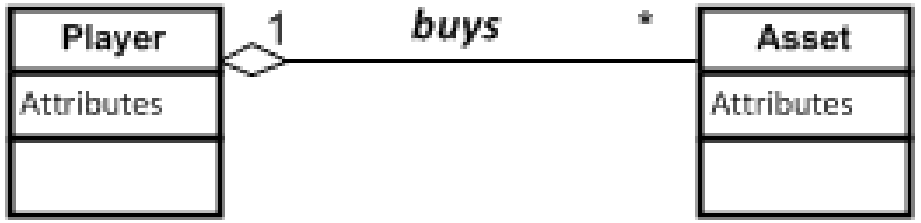
- Generalization √
- Dependency √
- Aggregation
- Composition

Aggregation

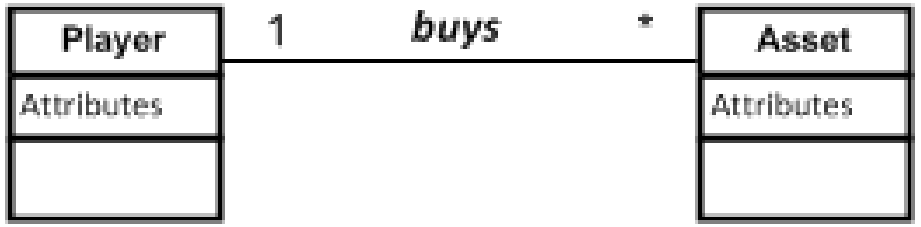
- Redundant form of association (represented as a solid line)
- Used to model “whole-part” relationships between things
- Defines the state of instances of the dependent class
- The dependent class (A1) holds a reference of the associated class (B1). The use of the specific instance of B1 is or may be shared among other aggregators.
- Therefore, if A1 goes out of scope (e.g. garbage collected), the instance of B1 does not go out of scope
- Symboled by a clear diamond at the dependent class side



Aggregation Example



Aggregation



FYI: alternative way called Association

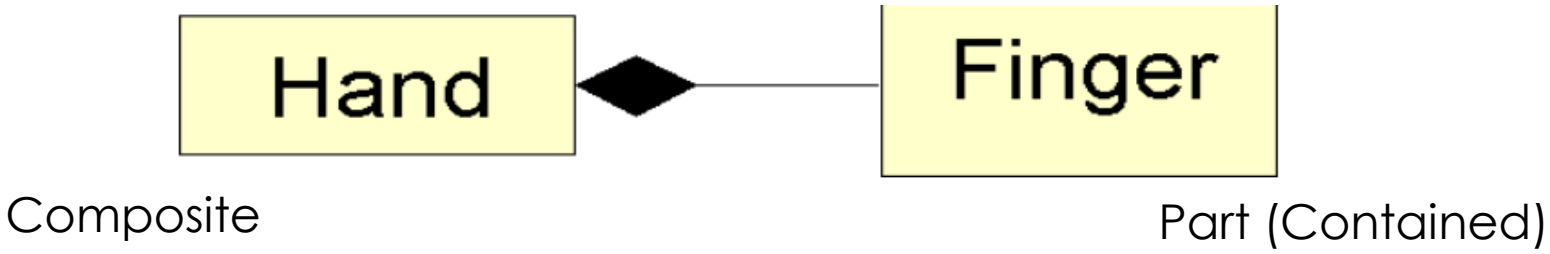
```
class Asset { … }
class Player {
List assets;
public void AddAsset(Asset newlyPurchasedAsset)
                    {
                    assets.Add(newlyPurchasedAssest);
          }
…  }
```

Relationships cont.

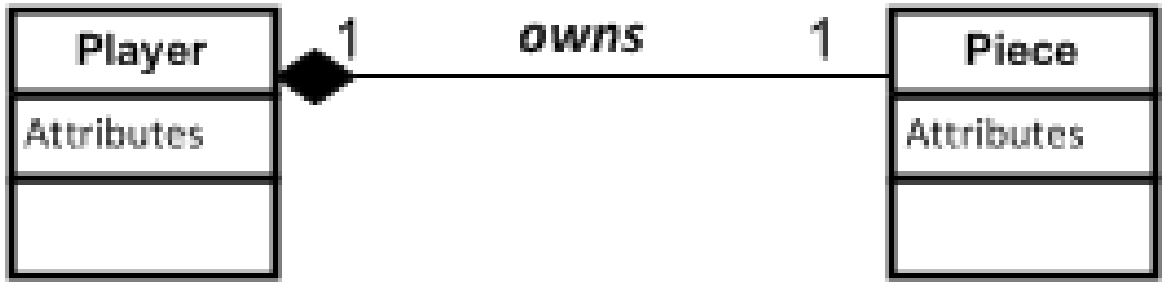
- Generalization √
- Dependency √
- Aggregation √
- Composition

Composition

- A much stronger aggregation (or association) relation ship
- A “has a“ relationship from the dependent class view
- The whole is generally called the composite (we will refer to the parts as contained)
- The lifetime of the part is bound within the lifetime of the composite. i.e. if the composite object goes out of scope, then the contained object also goes out of scope
- Symboled by a filled diamond
- Example:



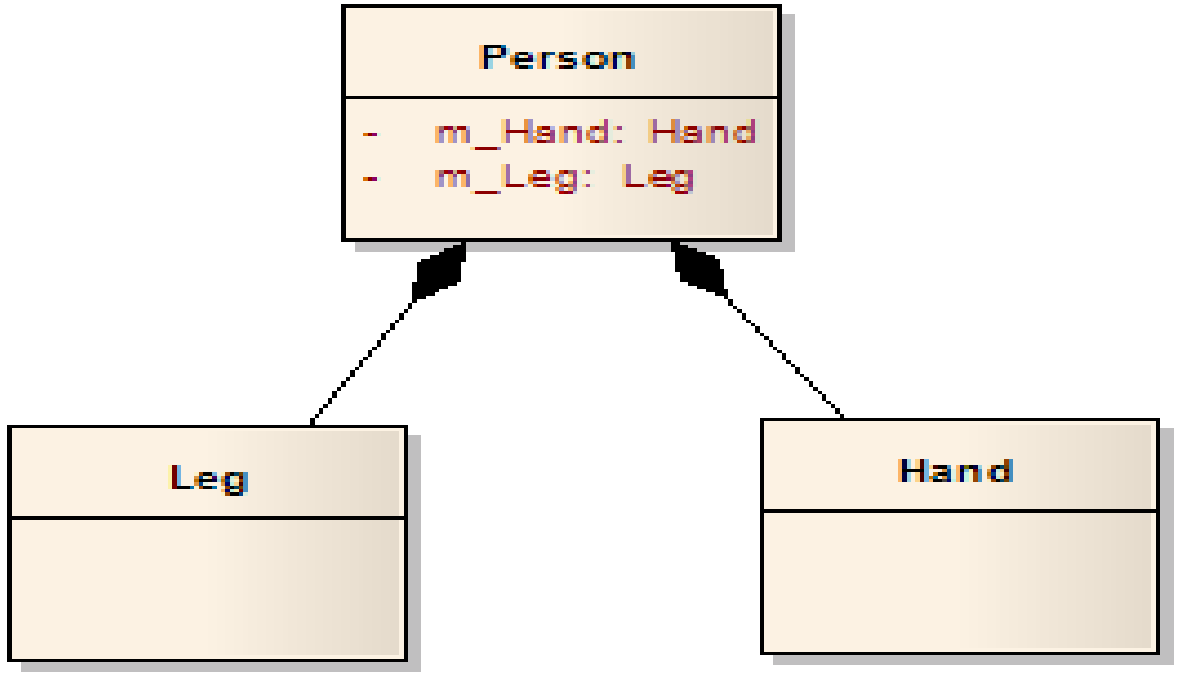
Composition




```
public class Piece { ... }
public class Player
{
Piece piece = new Piece(); /*Player owns the responsibility
of creating the Piece*/
...
}
```

Composition

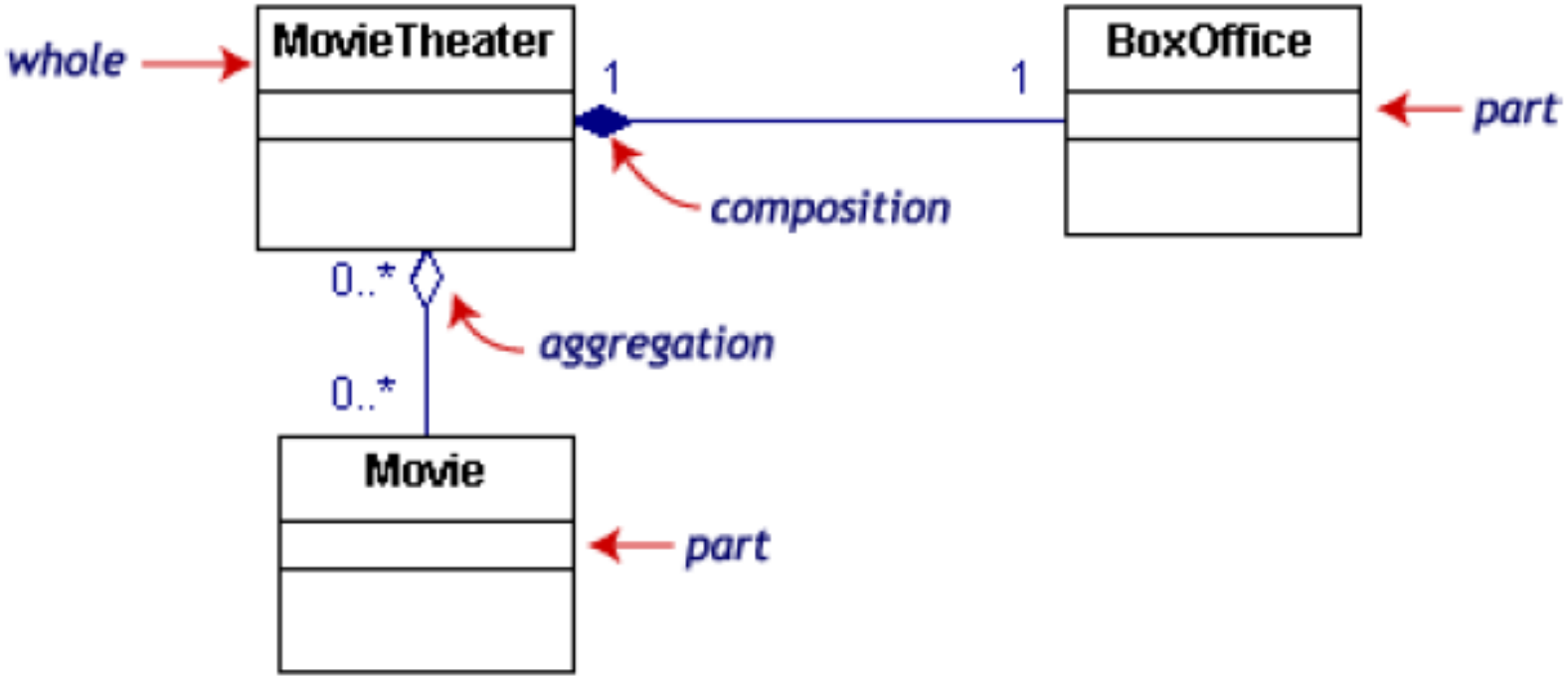
- Example 2: a person has a hand and a leg
- You can work and complete the human structure
  - E.g. another hand & leg, head, abdomen …etc.



Relationships cont.

- Generalization √
- Dependency √
- Aggregation √
- Composition √

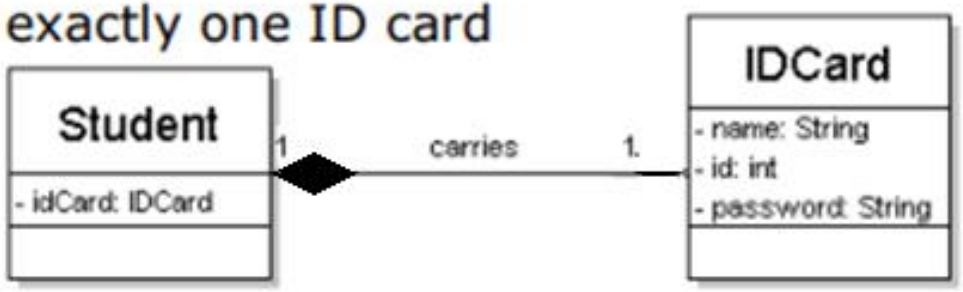
Aggregation and Composition Example



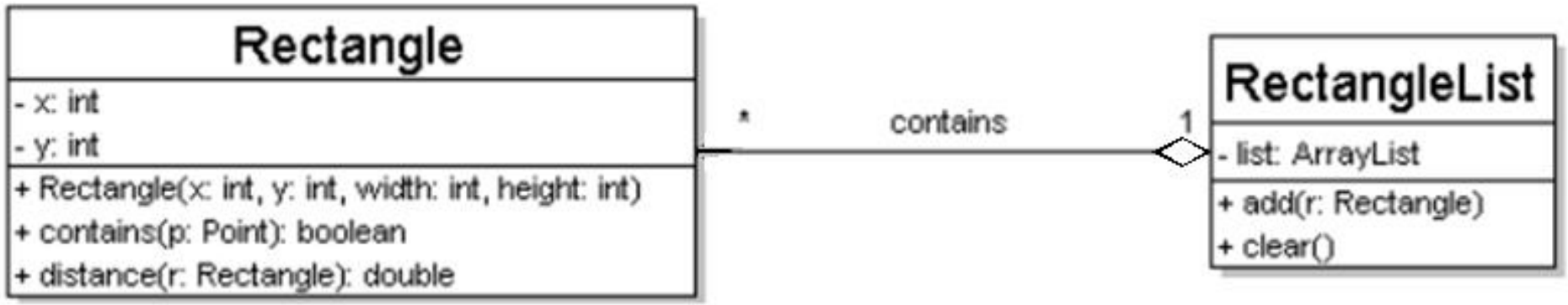
If the movie theater goes away
so does the box office => composition
but movies may still exist => aggregation

Multiplicity

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



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



Multiplicity

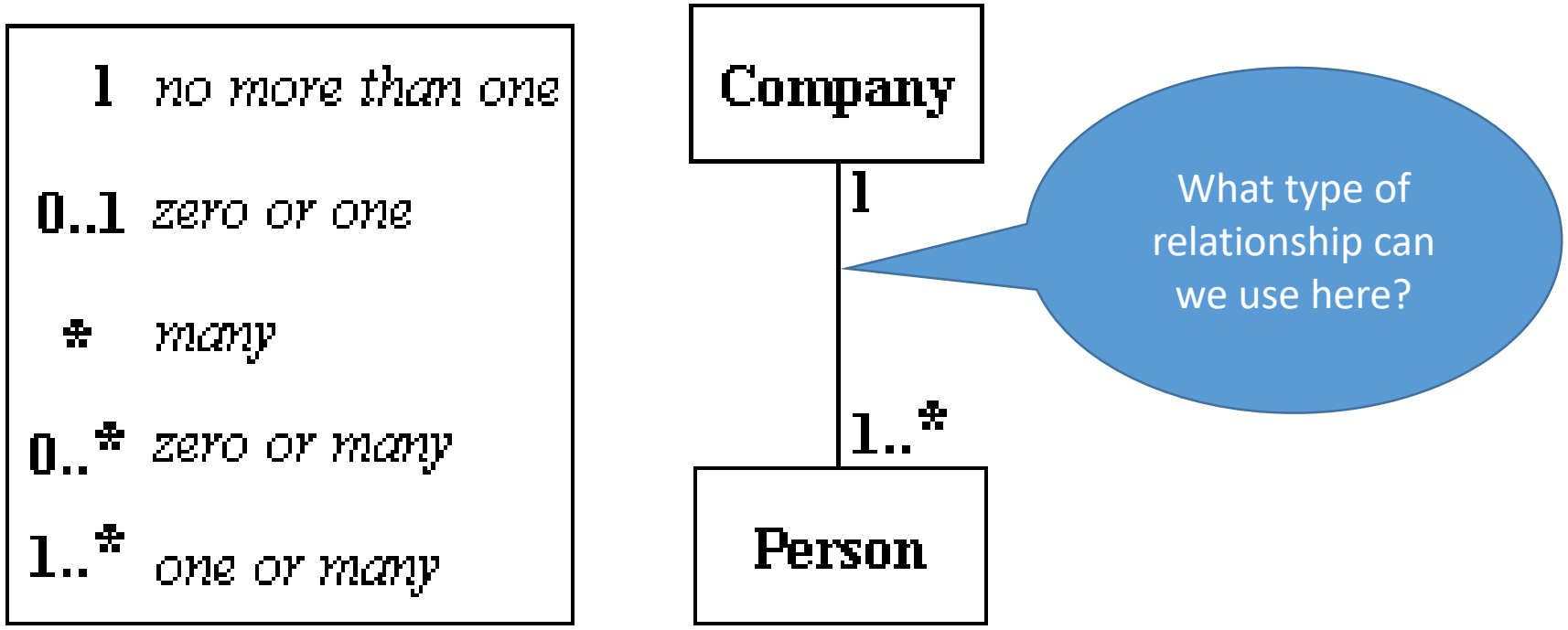
Indicator	Meaning
0..1	Zero or one
1	One only
0..*	0 or more
1..* *	1 or more
n	Only n (where n > 1)
0..n	Zero to n (where n >1)
1..n	One to n (where n > 1)

You can use a number instead of n. e.g. 0..4 which means minimum zero & max 4

Example

- One company will have one or more employees, but each employee works for just one company.
  - In implementation you may do one of the following:
    1. Define an array of length N, where the value N is determined at object creation
    2. Use dynamic allocation, such as: java.Util.ArrayList https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html

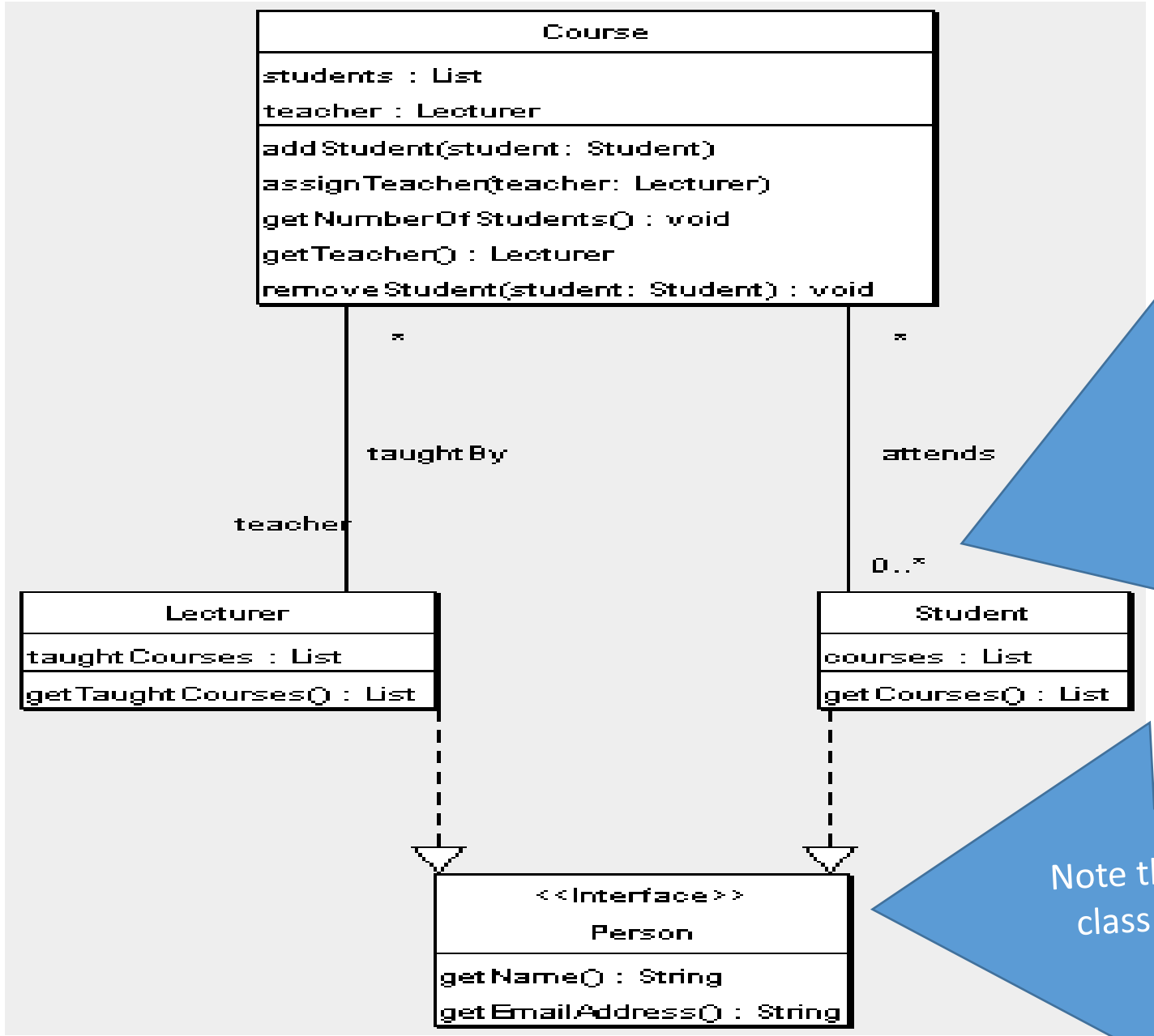
Example


1 no more than one
0..1 zero or one
* many
0..* zero or many
1..* one or many
Company
1
1..*
Person
What type of relationship can we use here?

Example: Students, Lecturers & Courses

- A Lecturer and a Student both is a Person who has a name and an email.
- A Lecturer teaches many courses while a course can be taught by one Lecturer only.
- A Student can attend many Courses while a Course can have zero or many Students.

Example: Students, Lecturers & Courses



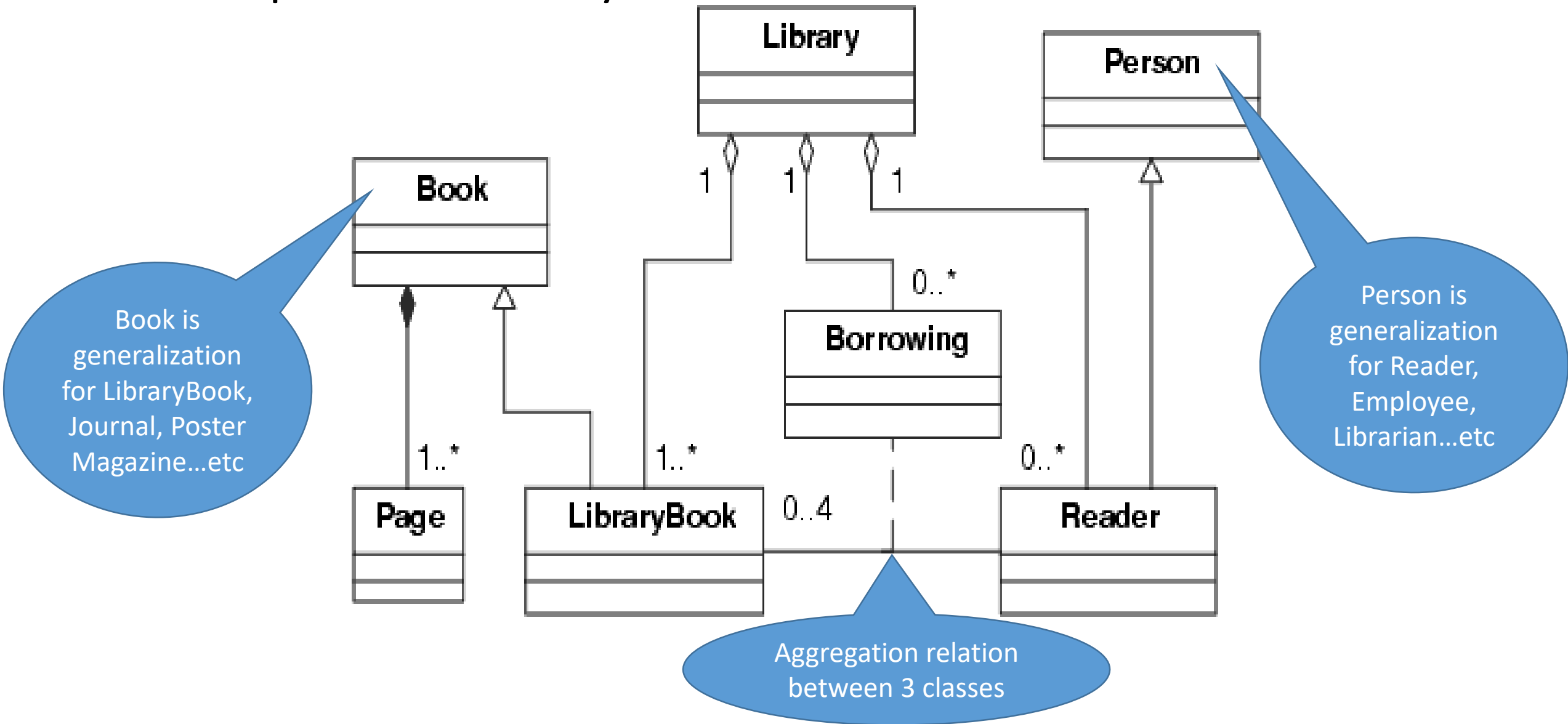
Note that you may have a dummy course with 0 students, hence 0..* cardinality.
e.g.1 closed course
e.g. 2 CE student taking a course in CS dept.

Note that you can still use a class instead of interface

Example: a library

- A library has many readers as members but a reader can be a member of one library only.
- A reader is a person.
- A book is made out of one or more pages.
- Library book is a type of book which can be found in one library only.
- A library has many library books.
- A reader can borrow up to 4 library books at a time.
- A reader can make another borrow as long as he returns the books he borrowed.

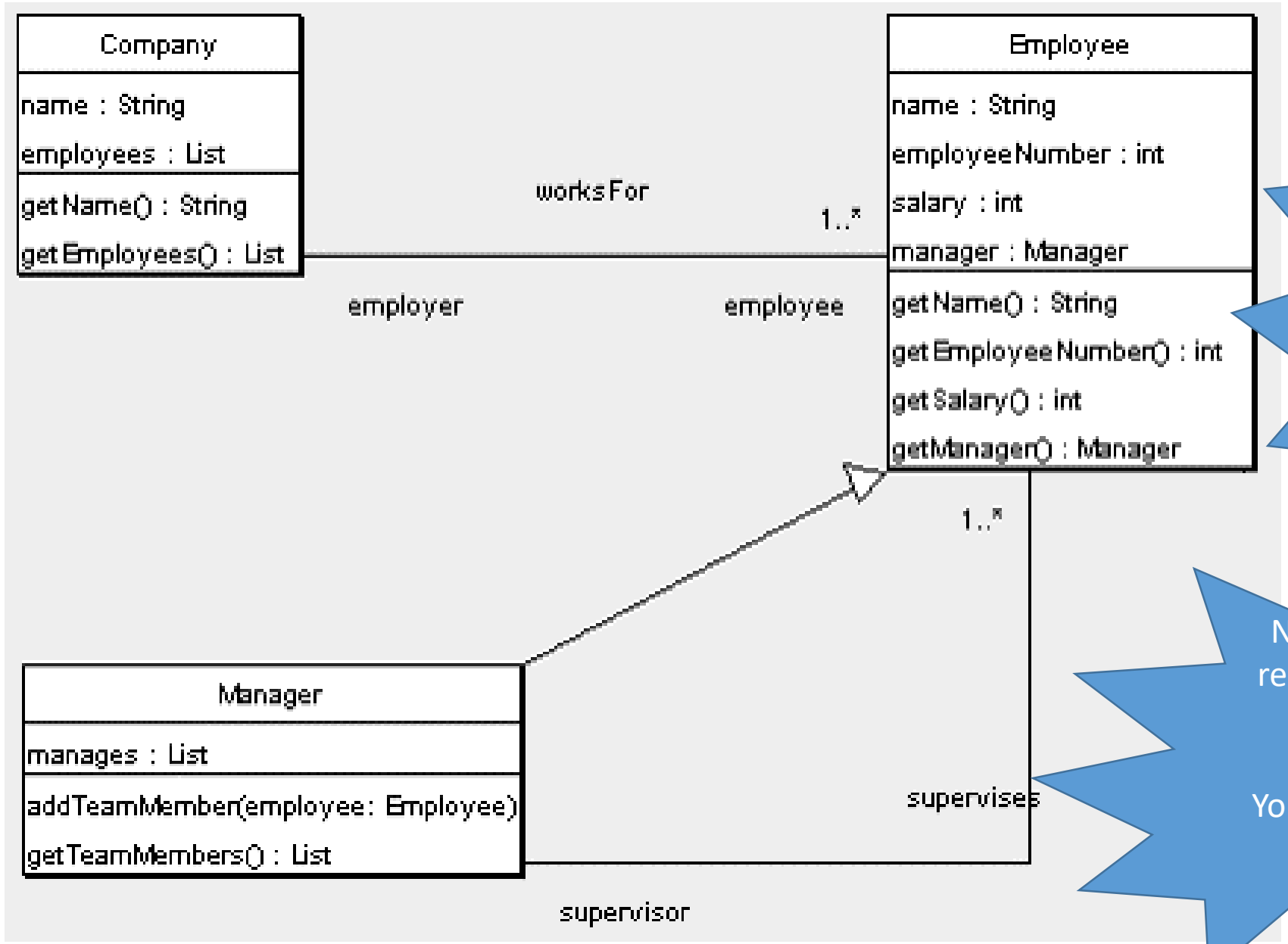
Example: a library



Example: Company, Employee & Manager

- A company has a name and employs one or more employees.
- An employee works for one company only.
- An employee has a name, work id, salary, and a supervisor manager.
- A manager is an employee who supervises one or more employees.

Example: Company, Employee & Manager



Add ref. to company obj in Employee class to keep track of Employee's Company

Note you have 2 relations between Manager and Employee
You can have more if needed

Thank you