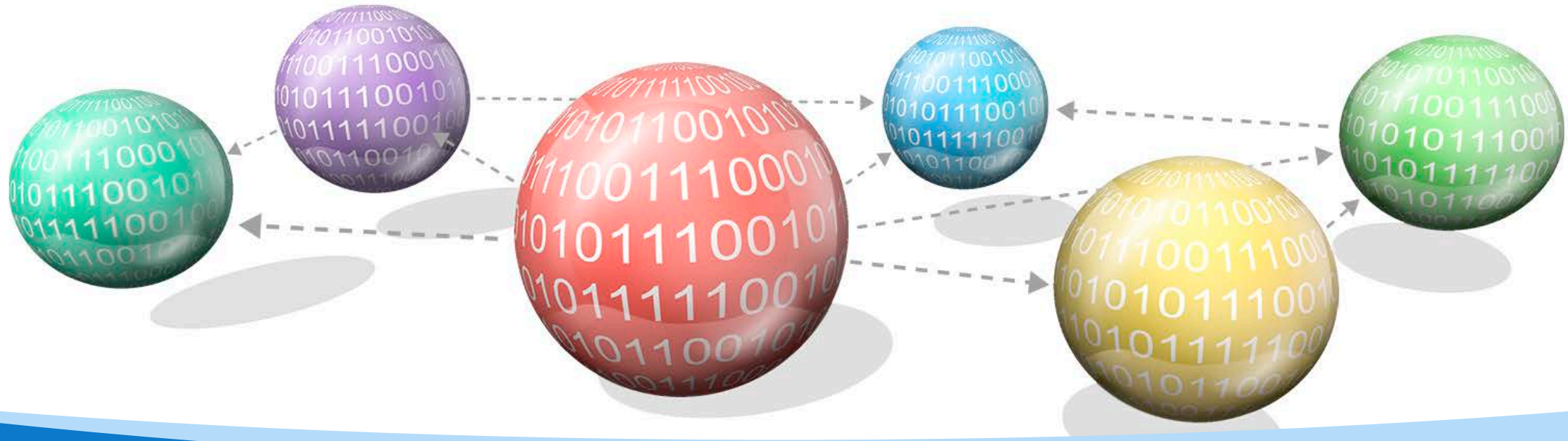


CE/CZ2002 Object-Oriented Design & Programming

Chapter 6: UML Model Class Relationships - Class Diagram

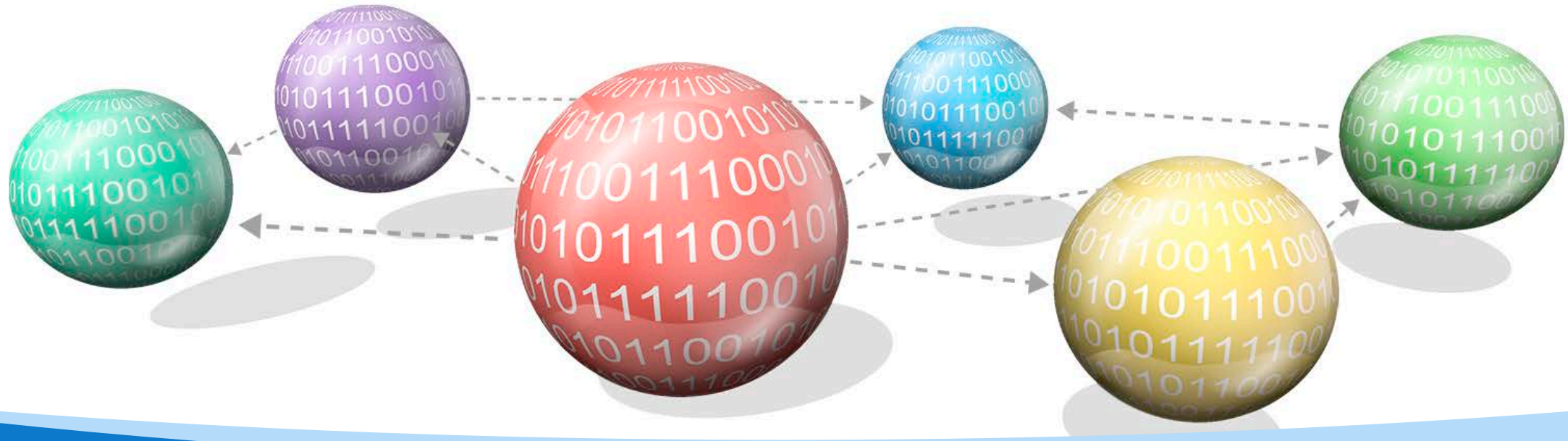
Mr Tan Kheng Leong
Lecturer, School of Computer Science and Engineering

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Objectives

By the end of this chapter, you should be able to:

- Explain the Unified Modeling Language (UML) model
- Explain class diagrams
- Explain the transition of UML class diagrams to Java code with examples
- Explain object diagram
- Explain class stereotypes



CE/CZ2002 Object-Oriented Design & Programming

Topic 1: UML Model

Chapter 6: UML Model Class Relationships - Class Diagram

Mr Tan Kheng Leong
Lecturer, School of Computer Science and Engineering

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Why Model?

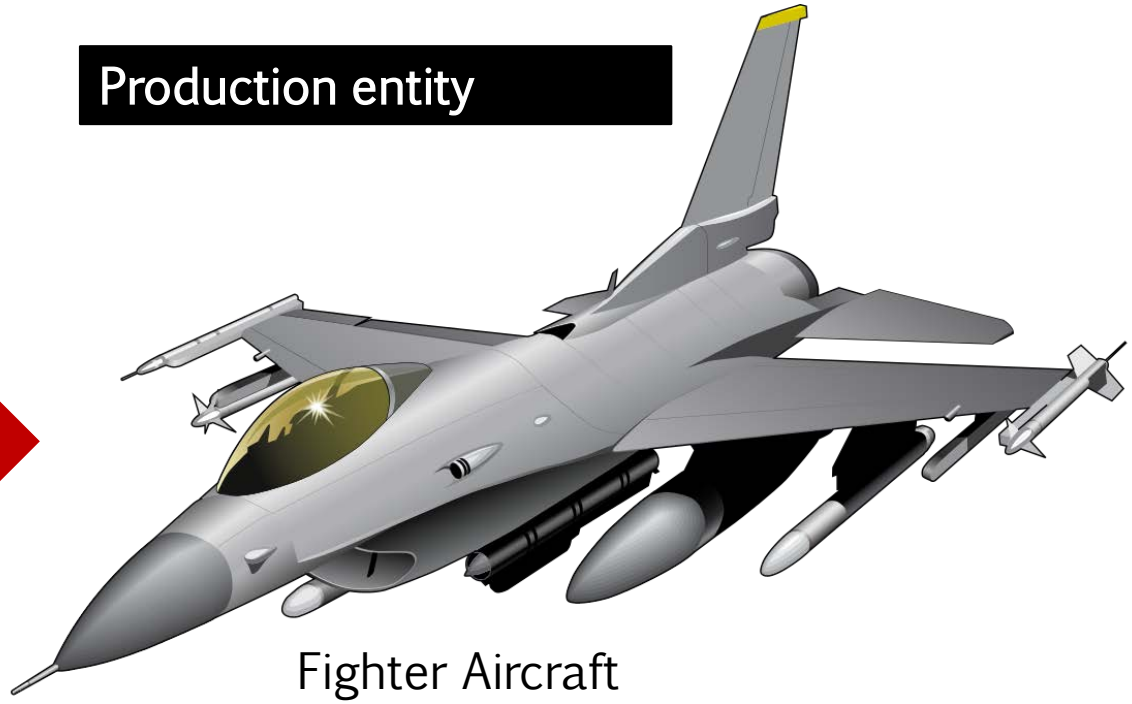
- Simplification of reality
- Helps problem visualisation, communication, understanding
  - 1-D **to** 2-D, 3-D, 4-D
- Used in design creation and investigation
- For documentation

Why Model?

Too abstract to be useful

Paper Airplane

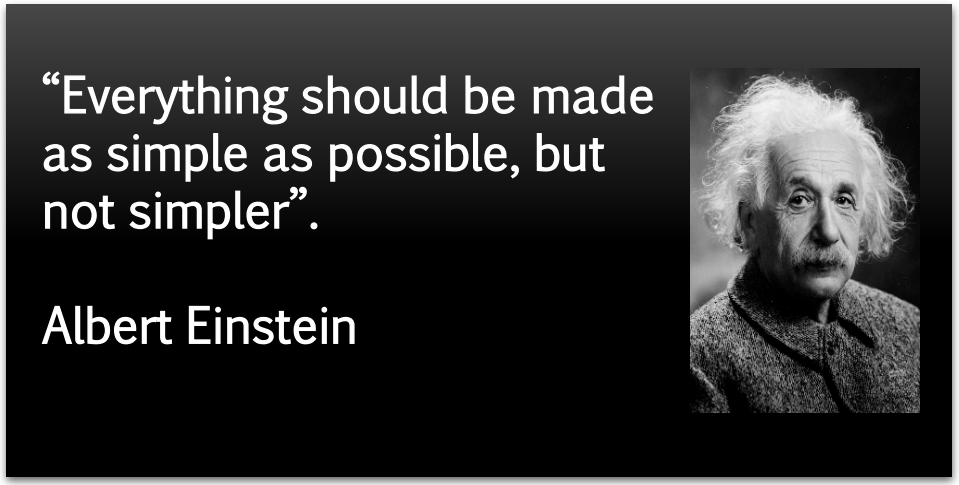
Production entity



Fighter Aircraft

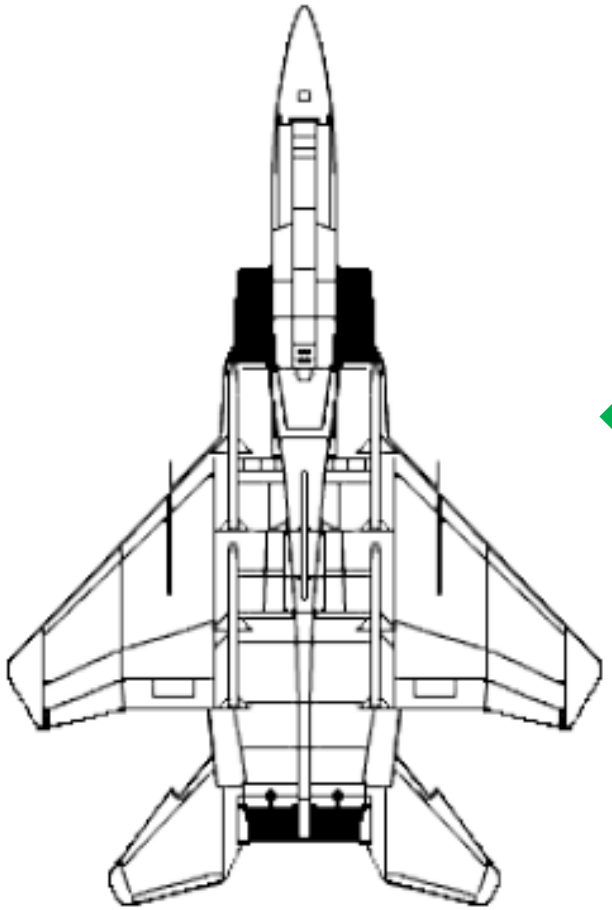
Why Model?

- Simplification of reality
- Helps problem visualisation, communication, understanding
  - 1-D **to** 2-D, 3-D, 4-D
- Used in design creation and investigation
- For documentation



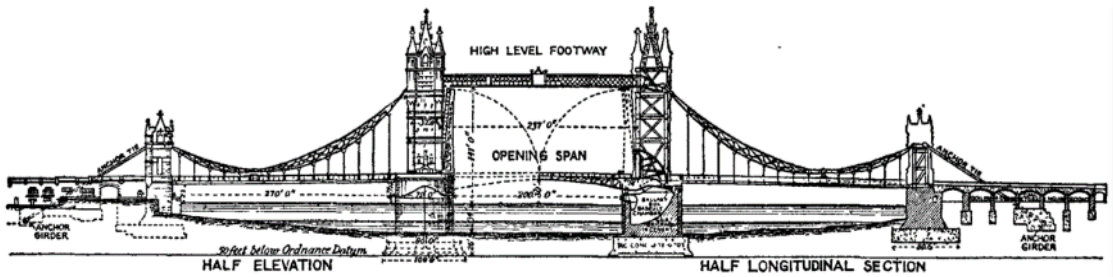
Albert Einstein. Retrieved December 16, 2016 from https://commons.wikimedia.org/wiki/File:Albert_Einstein_Head.jpg.

Why Model?



F-15 USAF. Retrieved December 19, 2016 from
https://commons.wikimedia.org/wiki/File:F-15_Eagle_drawing.png
https://commons.wikimedia.org/wiki/File:USAF_F-15D_Top.jpg

Why Model?

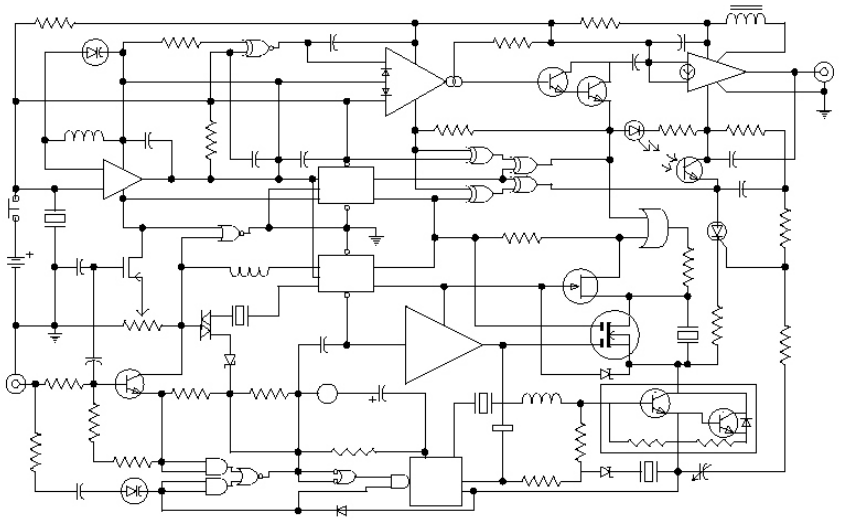


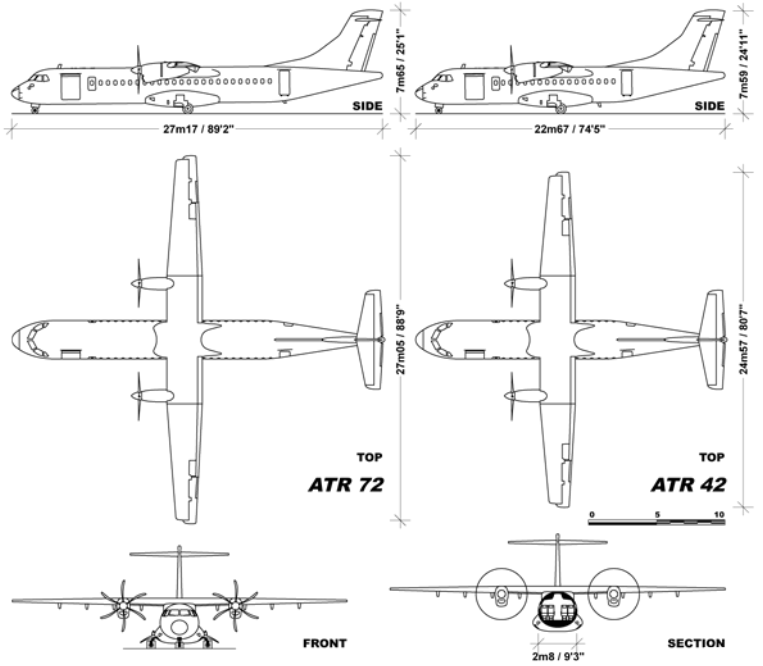
The **bridge** is 800 feet (240 metres) in length with two **towers** each 213 feet (65 metres) high, built on piers. The central span of 200 feet (61 metres) between the **towers** is split into two equal bascules or leaves, which can be raised to an angle of 86 degrees to allow river traffic to pass.

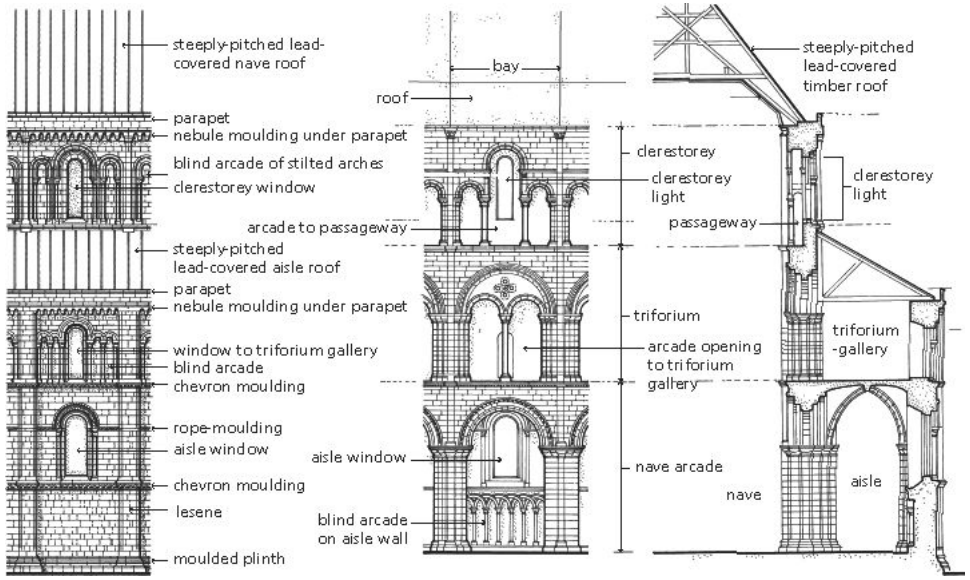
Tower Bridge London. Retrieved December 19, 2016 from
https://commons.wikimedia.org/wiki/File:Tower_Bridge_in_1911_Encyclop%C3%A6dia_Britannica.png
https://de.wikipedia.org/wiki/Darstellende_Geometrie#/media/File:Tower_Bridge_Vraneon.JPG
https://upload.wikimedia.org/wikipedia/commons/3/3d/Tower_Bridge,London_Getting_Opened_3.jpg

Unified Modeling Language

Refer to video at: 03:09



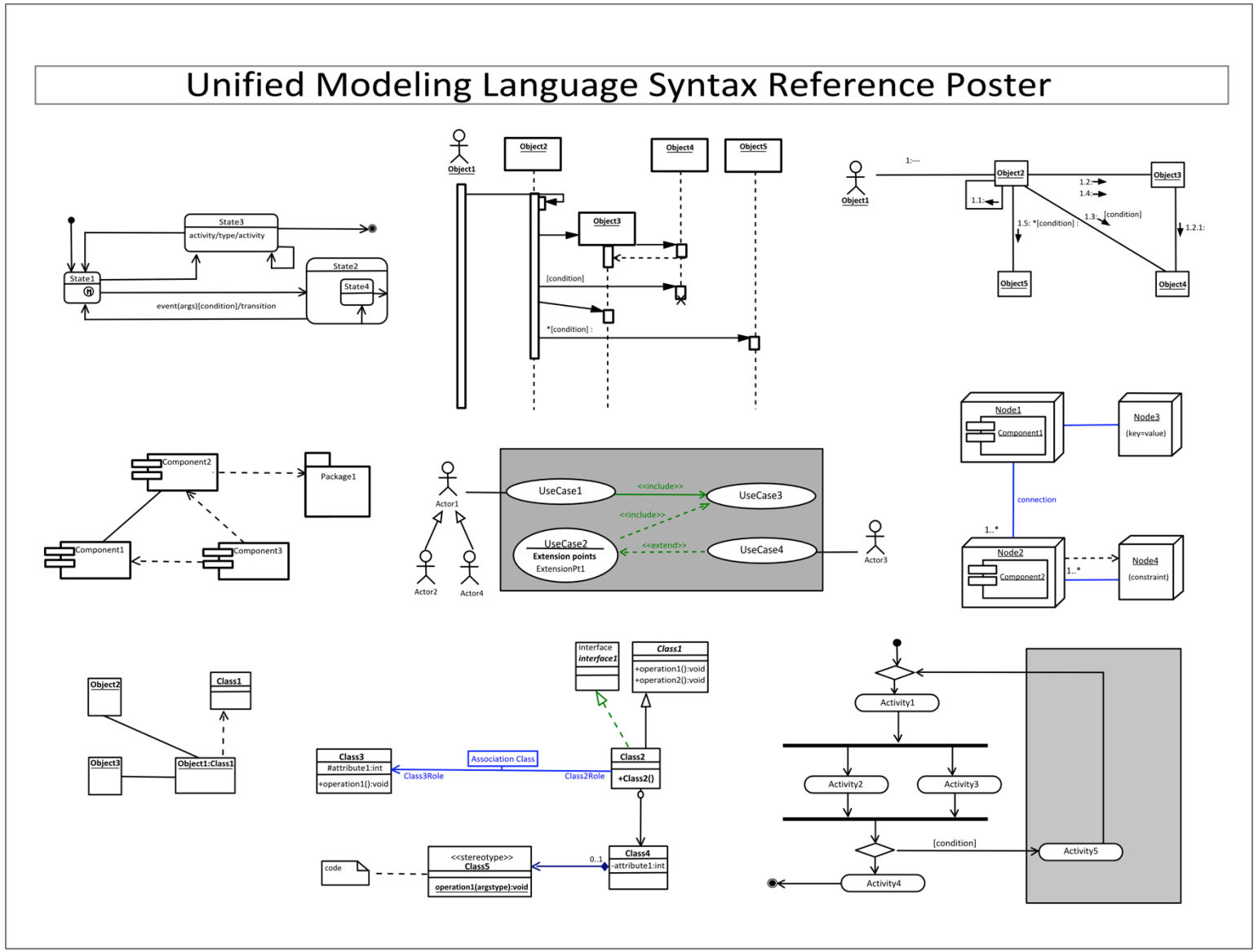




UML logo. Retrieved December 19, 2016 from https://commons.wikimedia.org/wiki/File:UML_logo.gif.
ATR42. Retrieved December 19, 2016 from https://upload.wikimedia.org/wikipedia/commons/1/1b/ATRv1.0.png.
3way. Retrieved December 19, 2016 from https://upload.wikimedia.org/wikipedia/commons/thumb/d/d1/California-3-way.svg/2000px-California-3-way.svg.png.
Romanesque. Retrieved December 19, 2016 from https://upload.wikimedia.org/wikipedia/commons/4/47/Romanesque.1.jpg.

Unified Modeling Language

UNIFIED MODELING LANGUAGE™



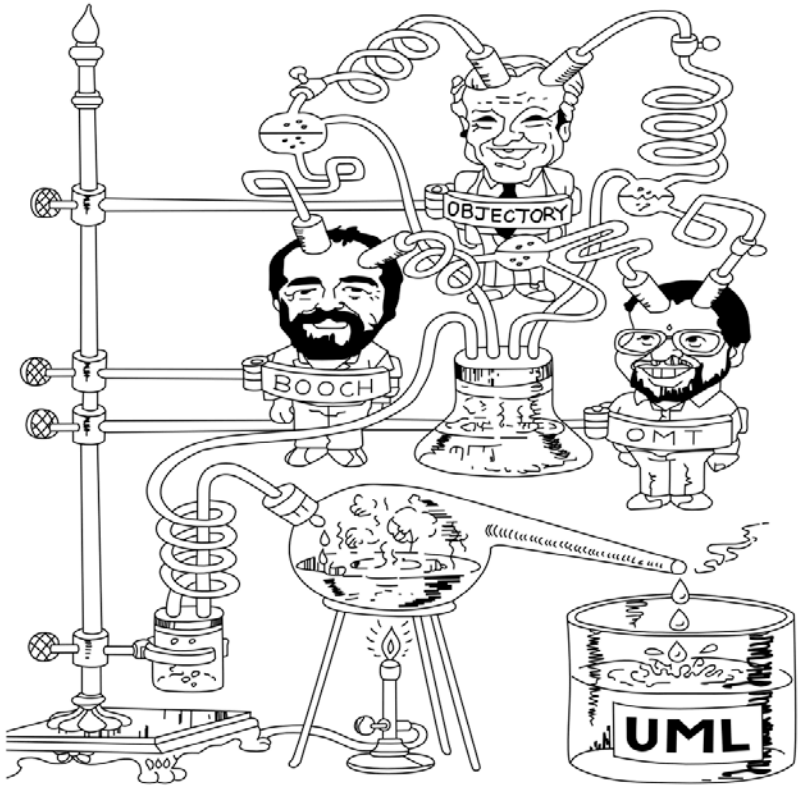
Unified Modeling Language

Refer to video at: 03:32

- Created by Booch, Jacobson & Rumbaugh in 1996.
- Version 1.1 adopted by Object Management Group (OMG) in 1997.

  http://www.omg.org/spec/UML/

- A visual language for specifying, documenting and communicating various aspects of complex software systems .



The Three Amigos

Unified Modeling Language

- Created by Booch, Jacobson & Rumbaugh in 1996.
- Version 1.1 adopted by Object Management Group (OMG) in 1997.

  http://www.omg.org/spec/UML/
- A visual language for specifying, documenting and communicating various aspects of complex software systems .

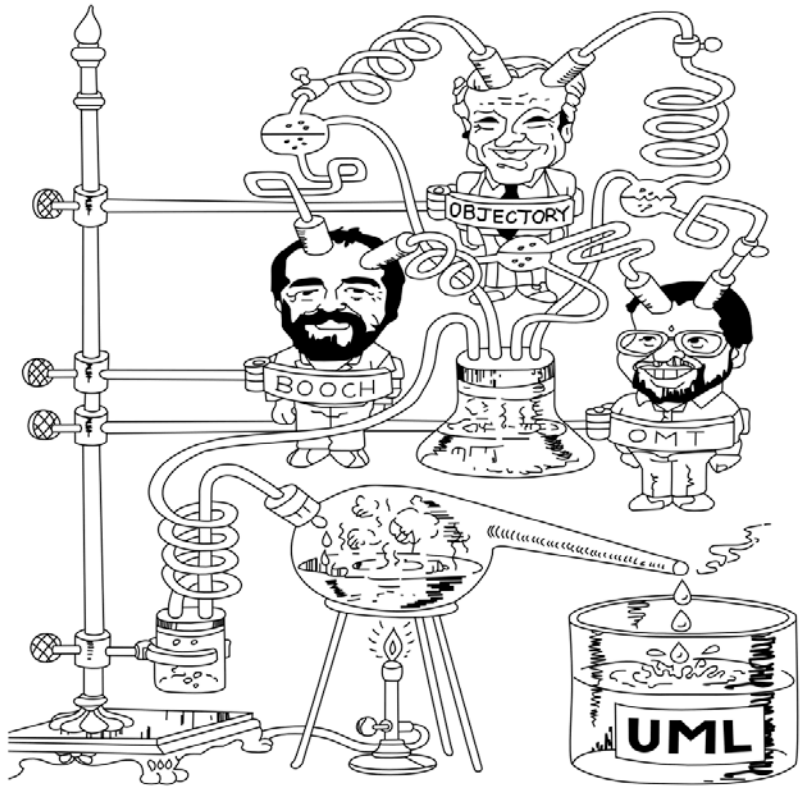
Reading Left to Right	Each A must be assigned to exactly one B	Each A must be assigned to one or many of B	Each A must be assigned to zero or one of B	Each A may be assigned any number of Bs
UML	A 1 B	A 1..* B	A 0..1 B	A * B
Martin/Odell (1st edition)	A B	A B	A B	A B
Booch (2nd edition)	A 1 B	A 1..N B	A 0..1 B	A N B
Coad/Yourdon	A 1 B	A 1,m B	A 0.1 B	A 0,m B
Jacobson (unidirectional)	A [1] B	A [1..M] B	A [0..1] B	A [0..M] B
OMT	A B	A 1+ B	A B	A B

Unified Modeling Language

- Created by Booch, Jacobson & Rumbaugh in 1996.
- Version 1.1 adopted by Object Management Group (OMG) in 1997.

  http://www.omg.org/spec/UML/

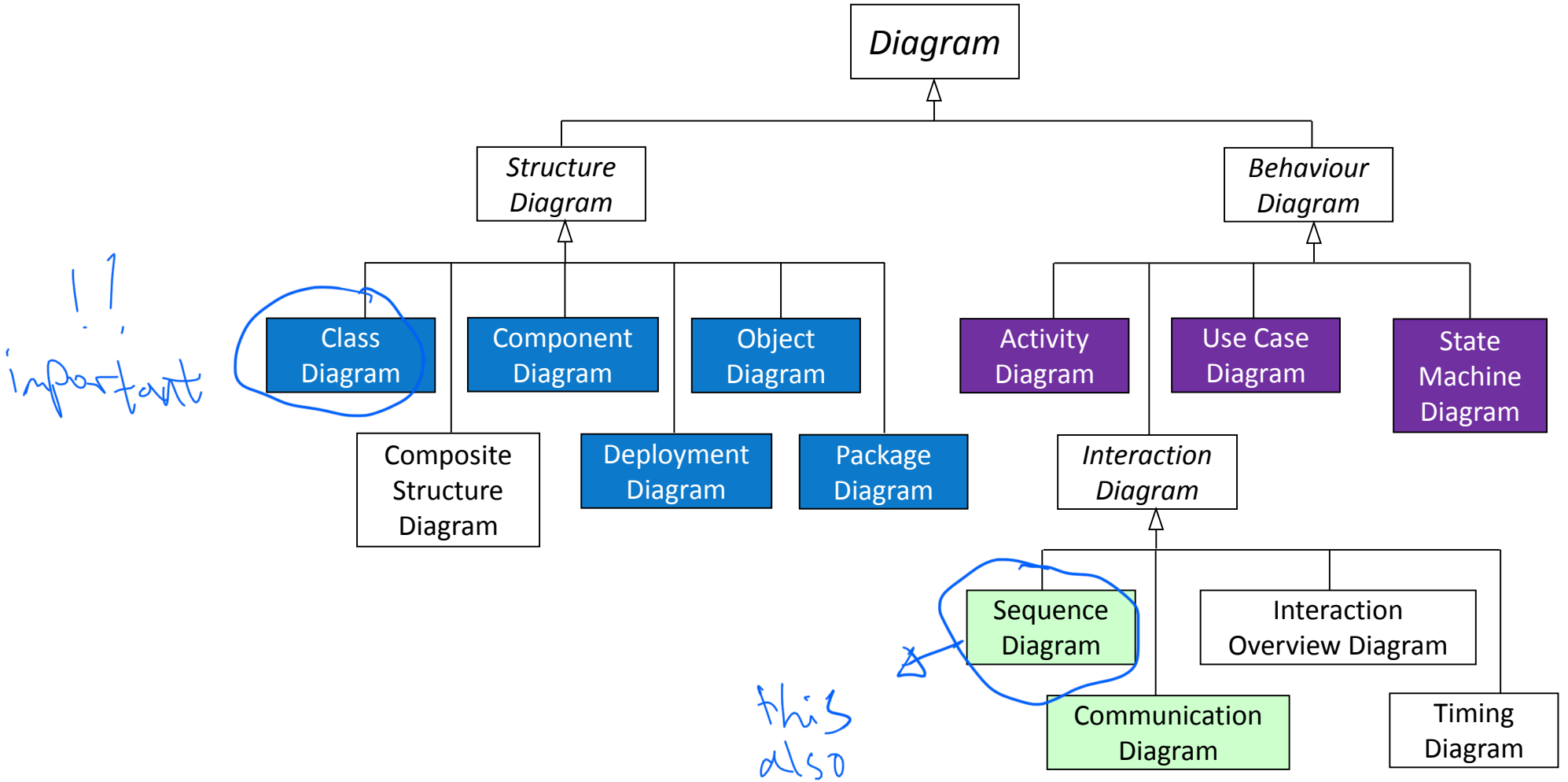
- A visual language for specifying, documenting and communicating various aspects of complex software systems .



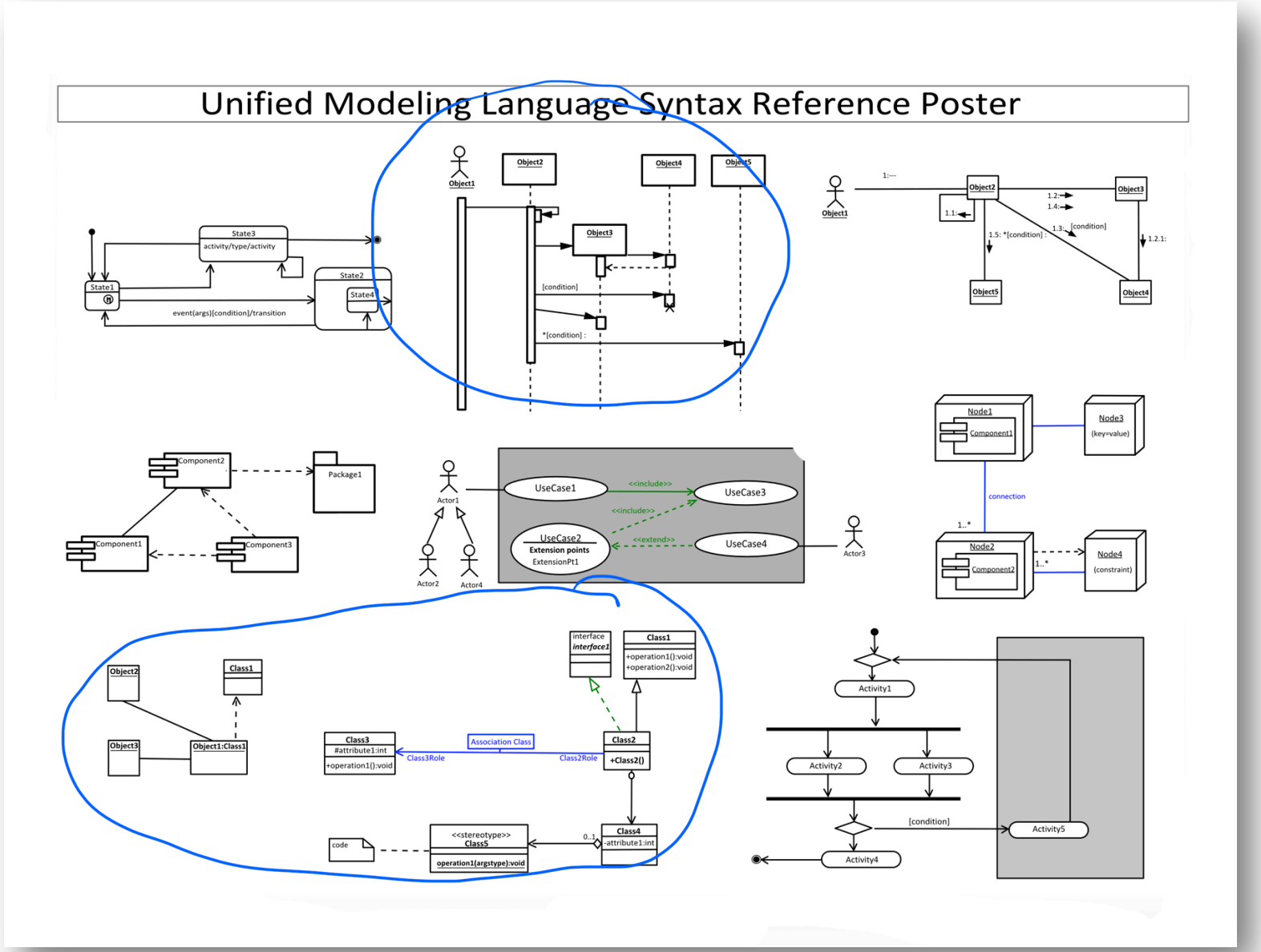
The Three Amigos

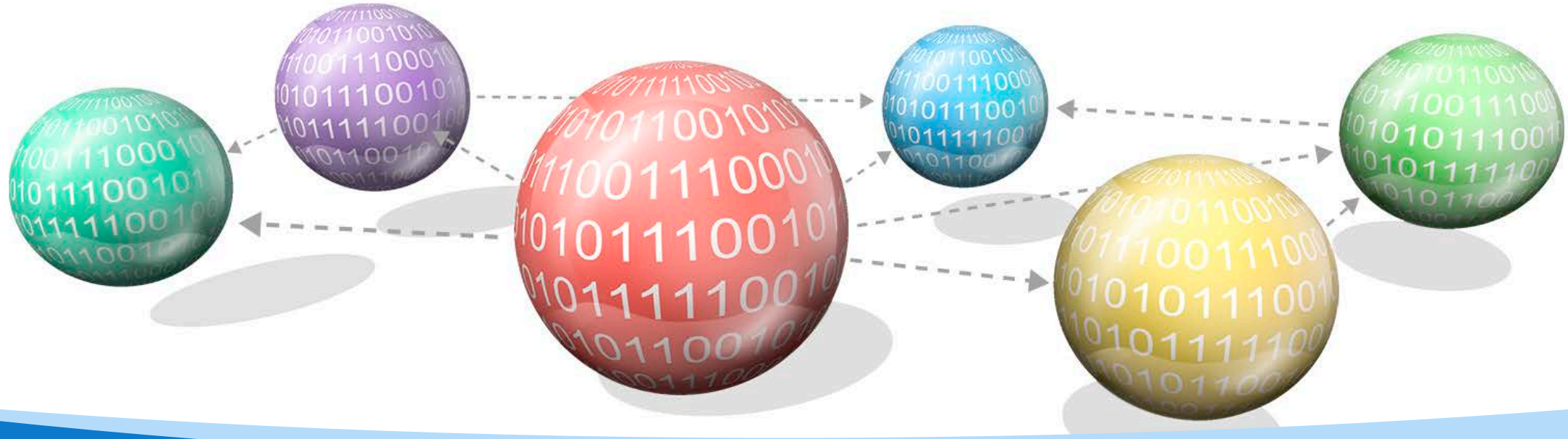
UML Diagram Types

Refer to video at: 04:50



UML Diagram Types





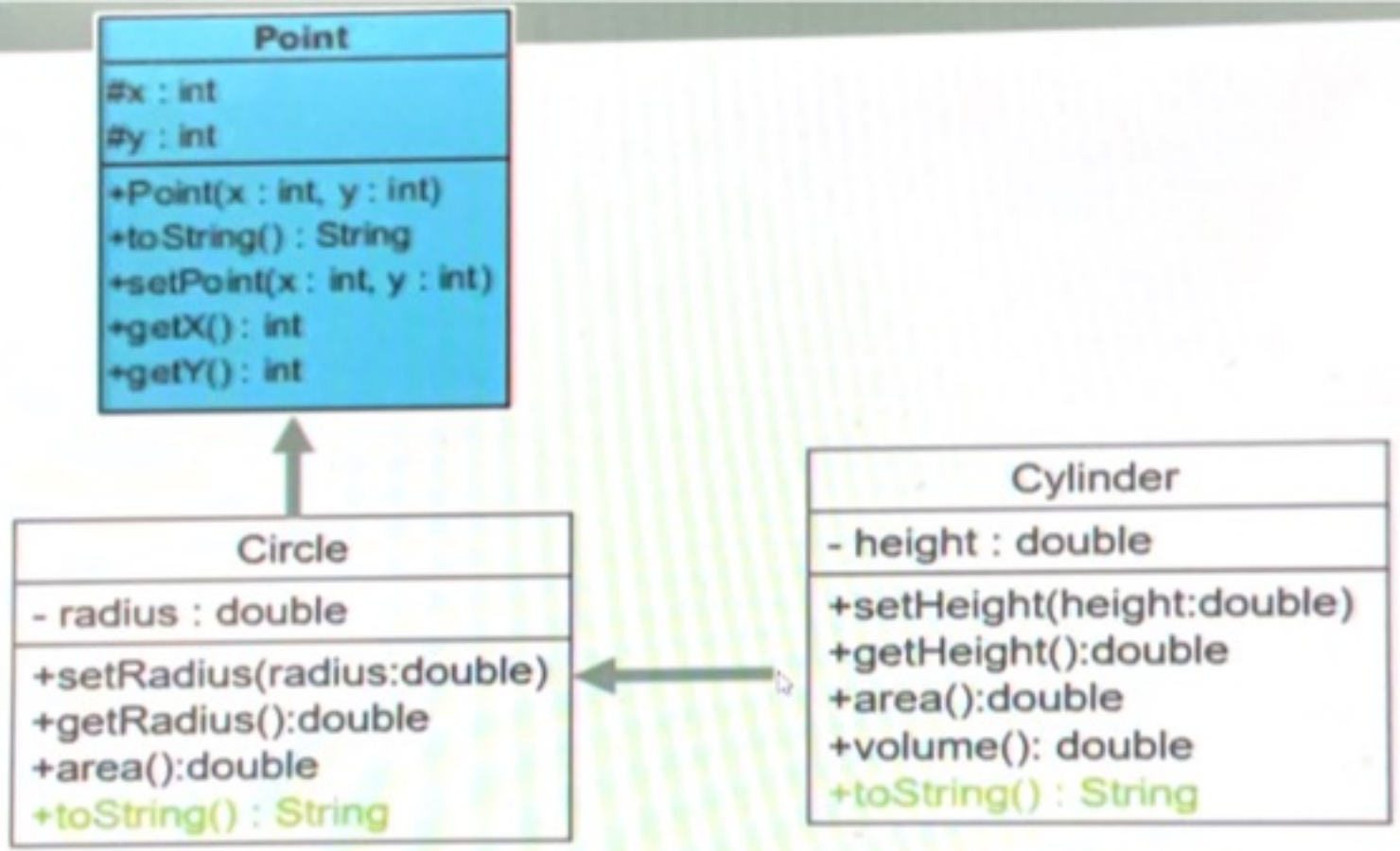
CE/CZ2002 Object-Oriented Design & Programming

Topic 2: Class Diagram

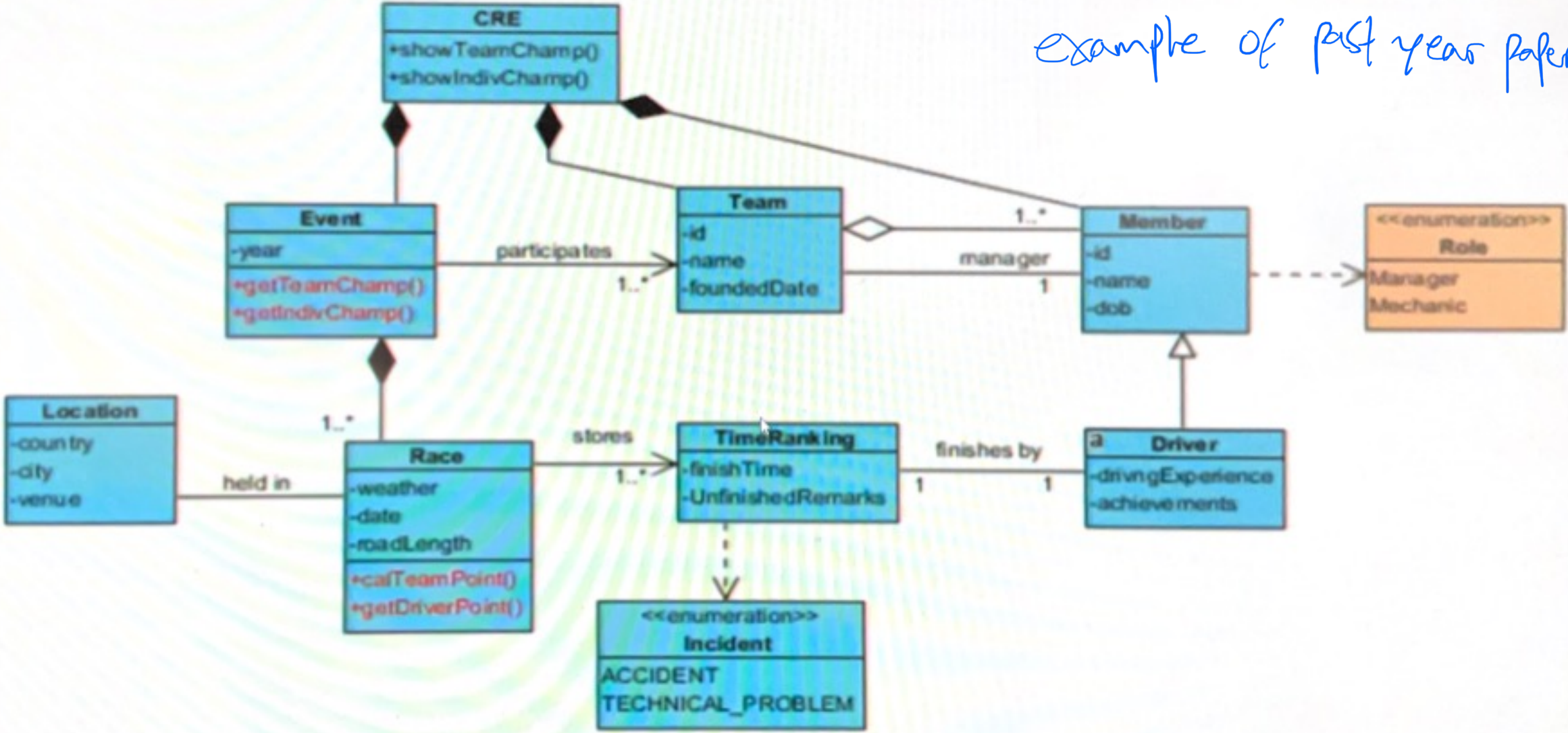
Chapter 6: UML Model Class Relationships - Class Diagram

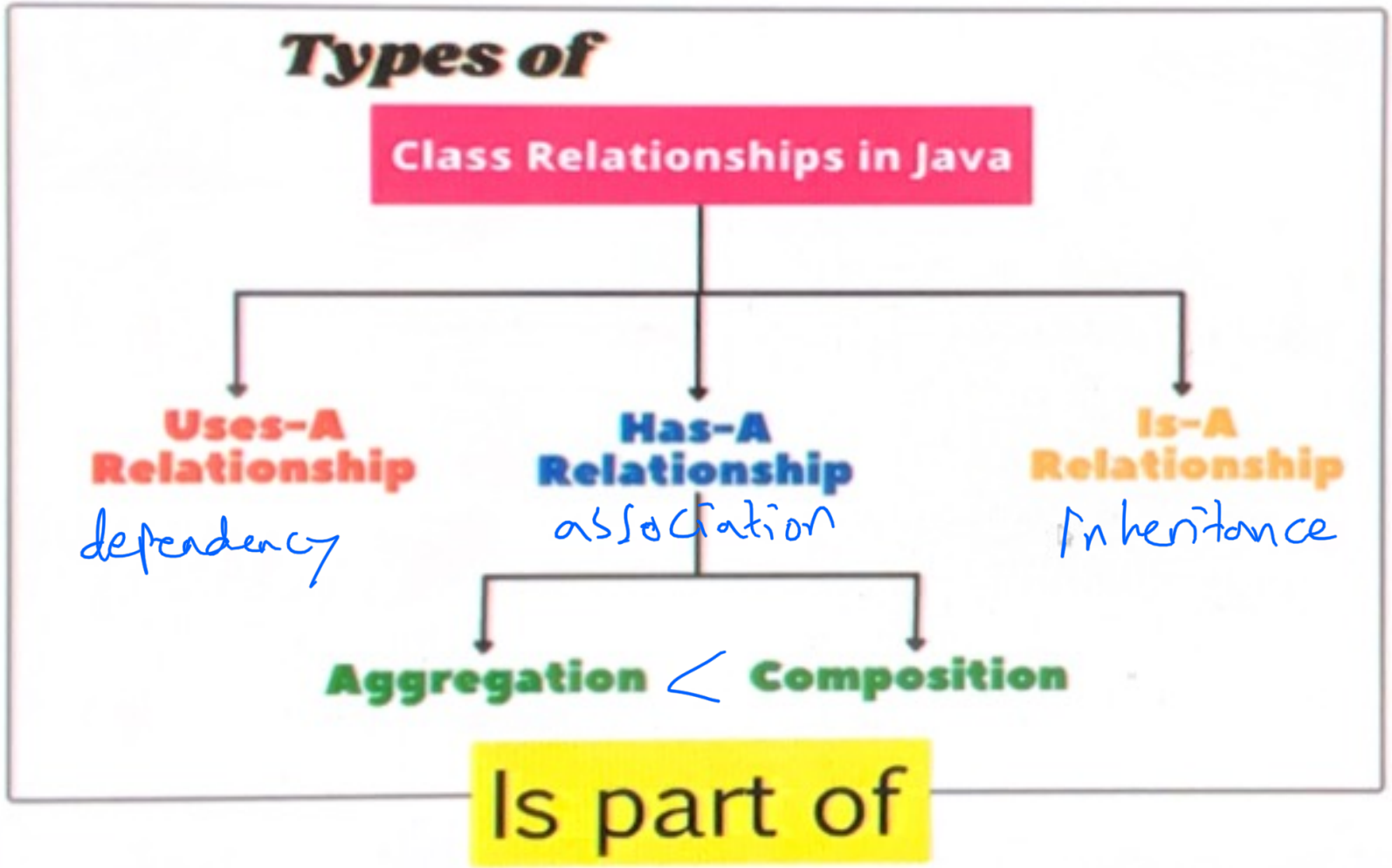
Mr Tan Kheng Leong
Lecturer, School of Computer Science and Engineering

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE


Point
#x : int
#y : int
+Point(x : int, y : int)
+toString() : String
+setPoint(x : int, y : int)
+getX() : int
+getY() : int
Circle
- radius : double
+setRadius(radius:double)
+getRadius():double
+area():double
+toString() : String
Cylinder
- height : double
+setHeight(height:double)
+getHeight():double
+area():double
+volume(): double
+toString() : String

example of past year paper

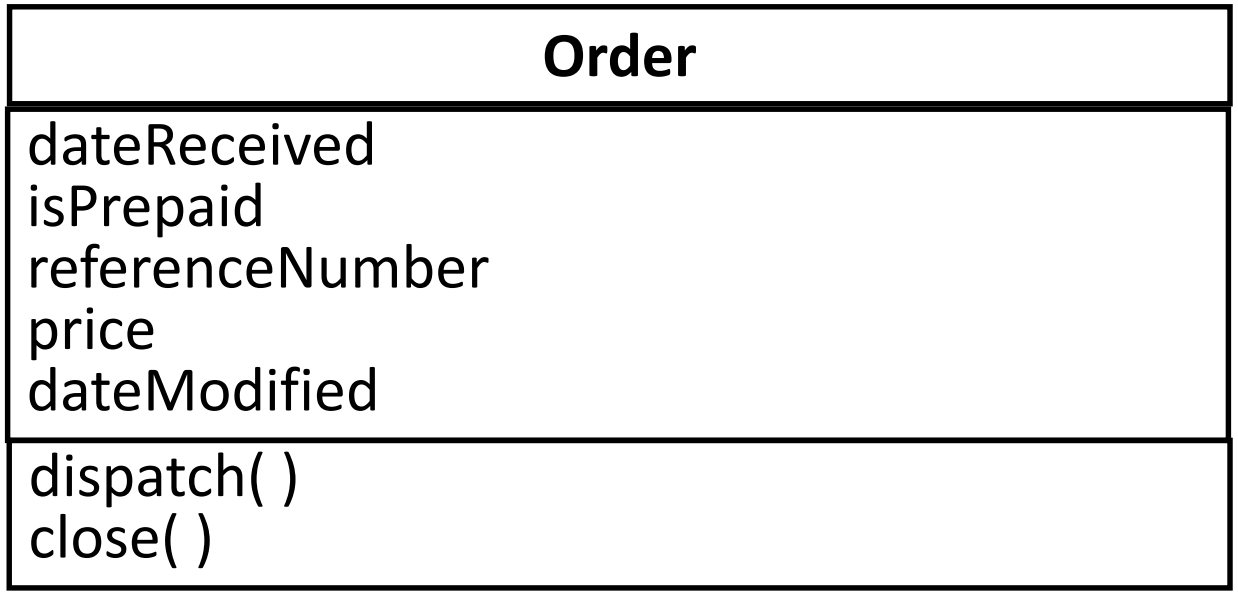




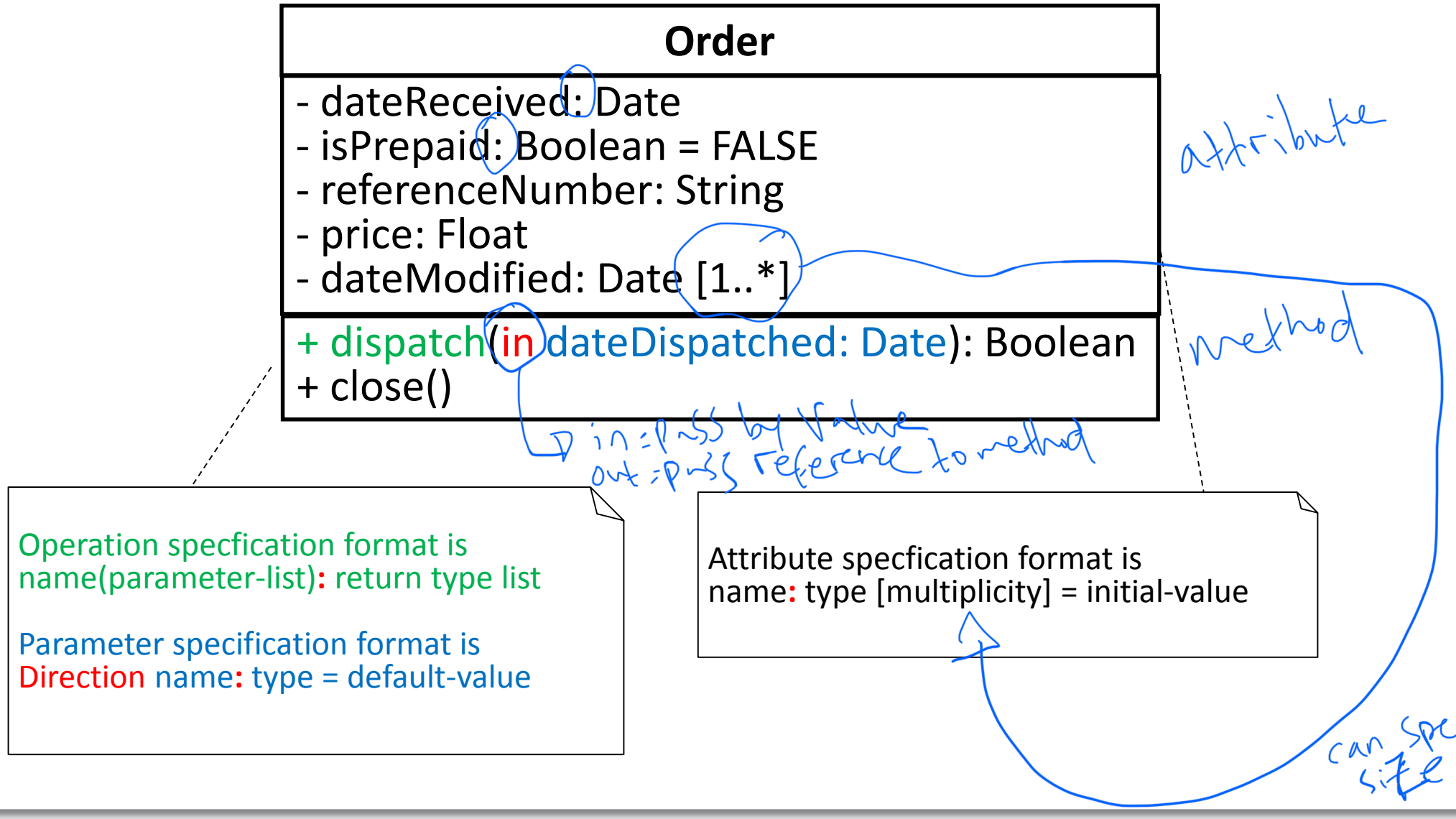
A has a B

B is part of A

Class Diagram

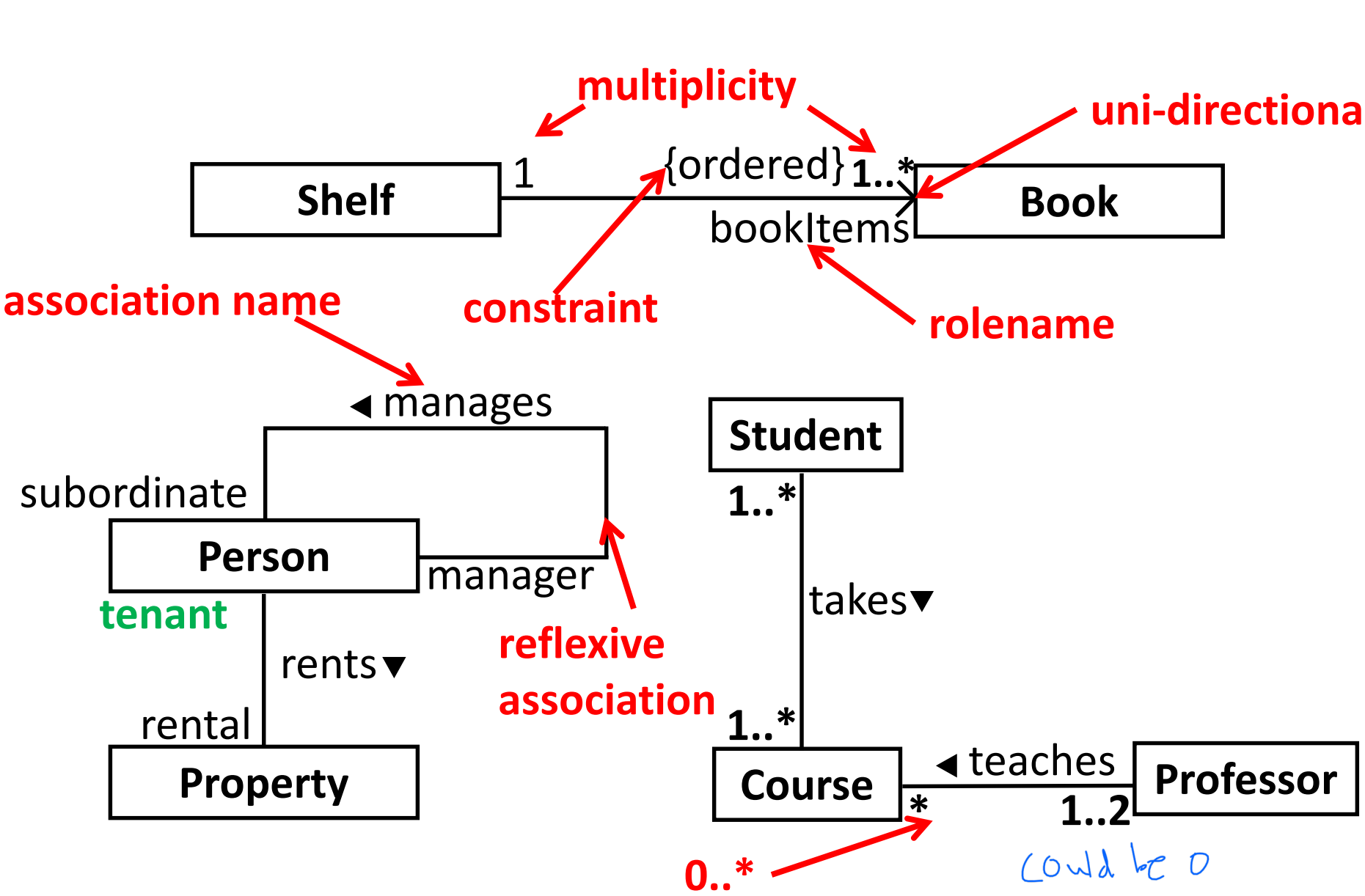


Class Diagram

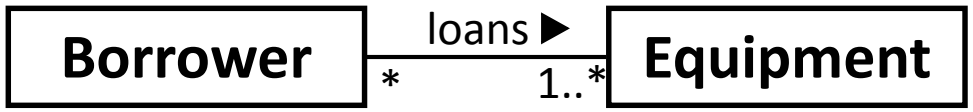


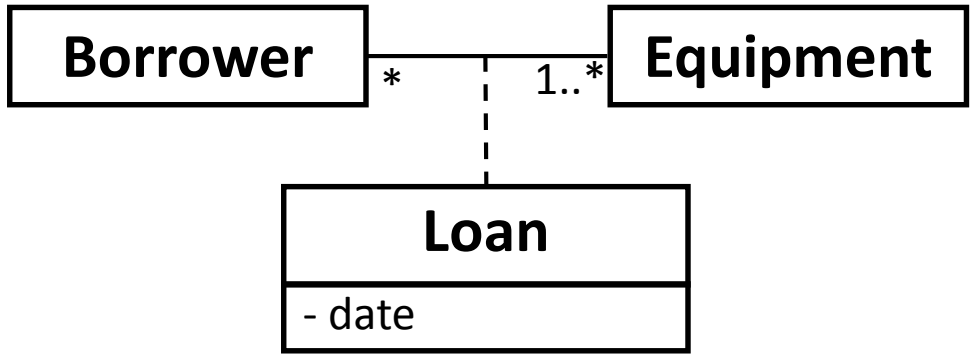
Class Diagram – Association

Refer to video at: 02:32

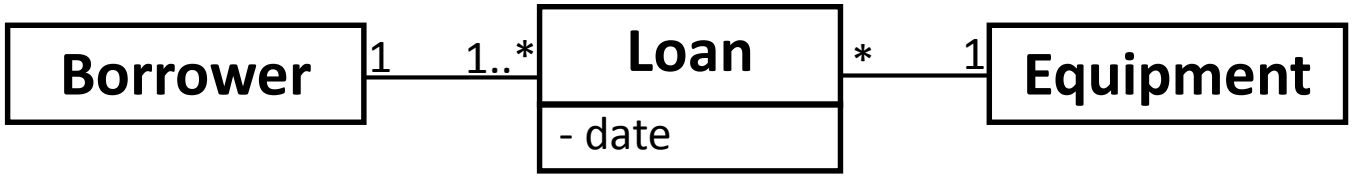


Association Classes



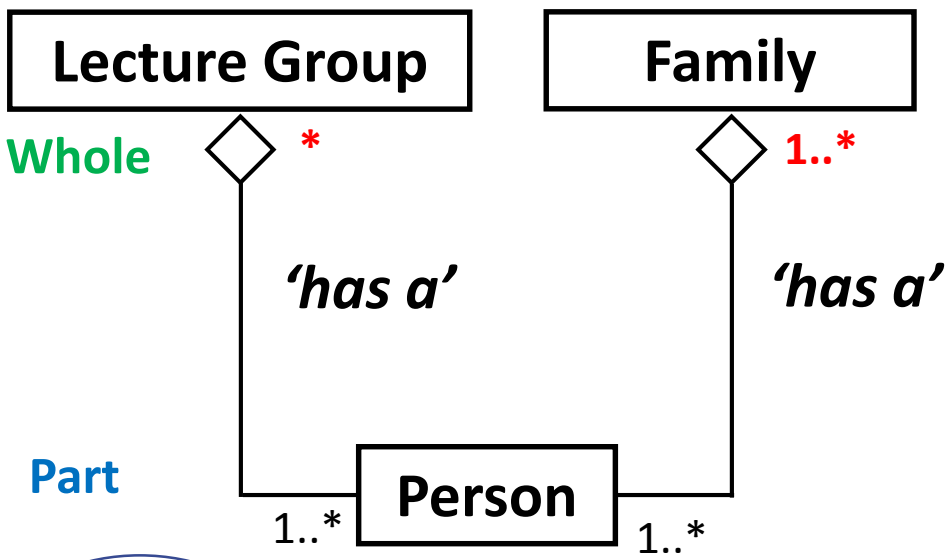


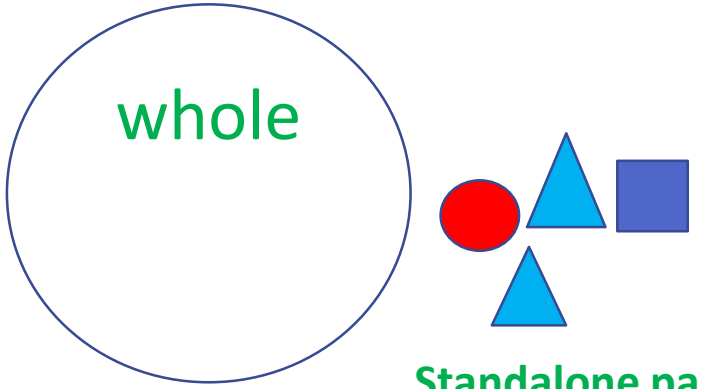
Loan Record		
Date	Borrower	Equipment
1 Mar	Tom	Tester
11 Jul	Tom	Multimeter
1 Oct	Ann	Tester
1 Oct	Tom	Solder



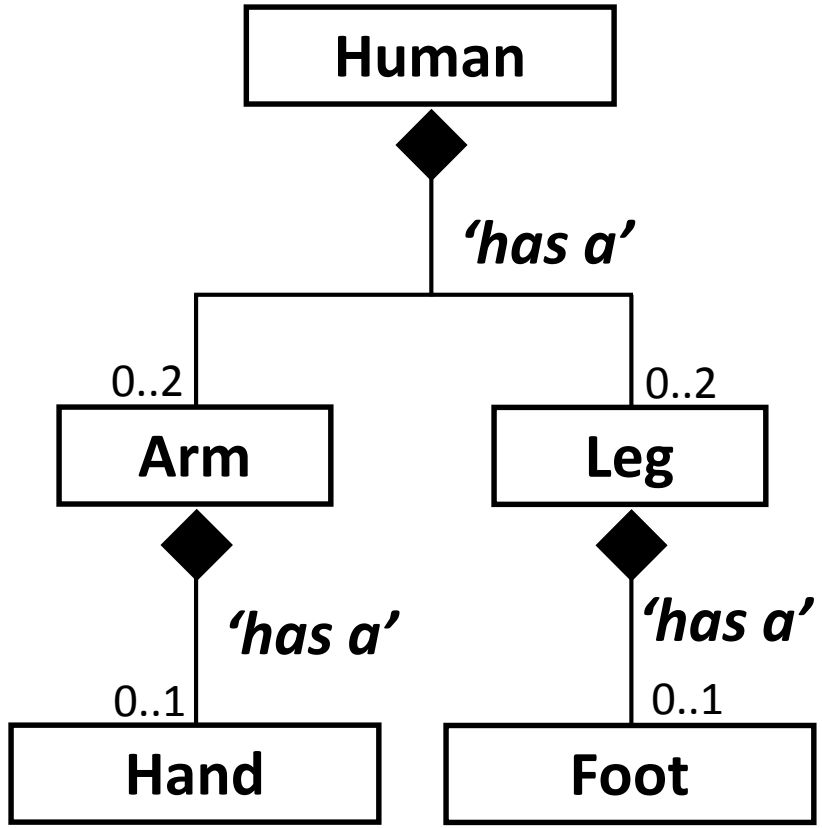
Aggregation and Composition

Whole – Part/s relationship





Standalone parts



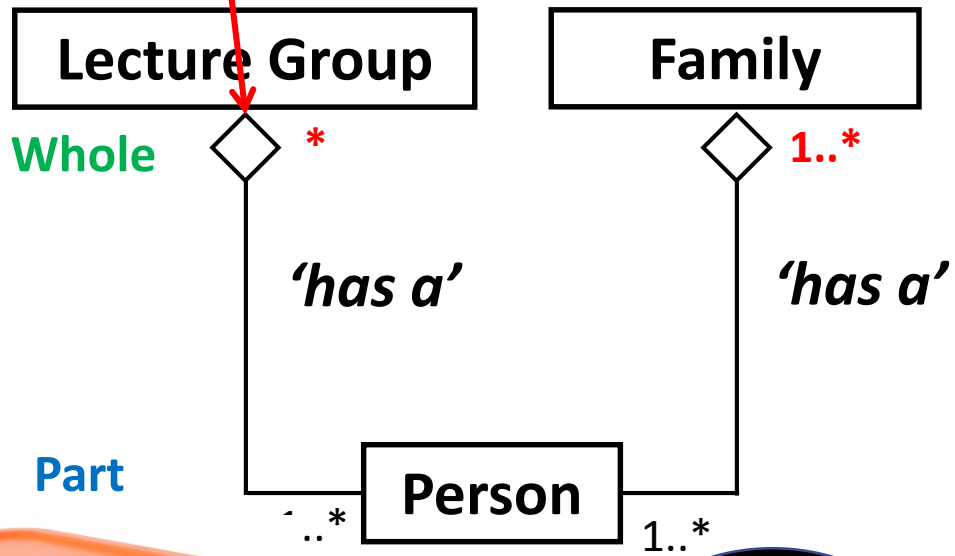
Whole creates/owns parts

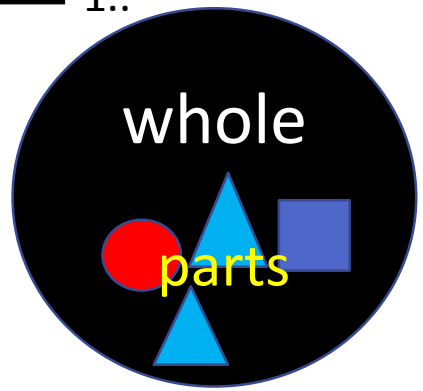
Aggregation and Composition

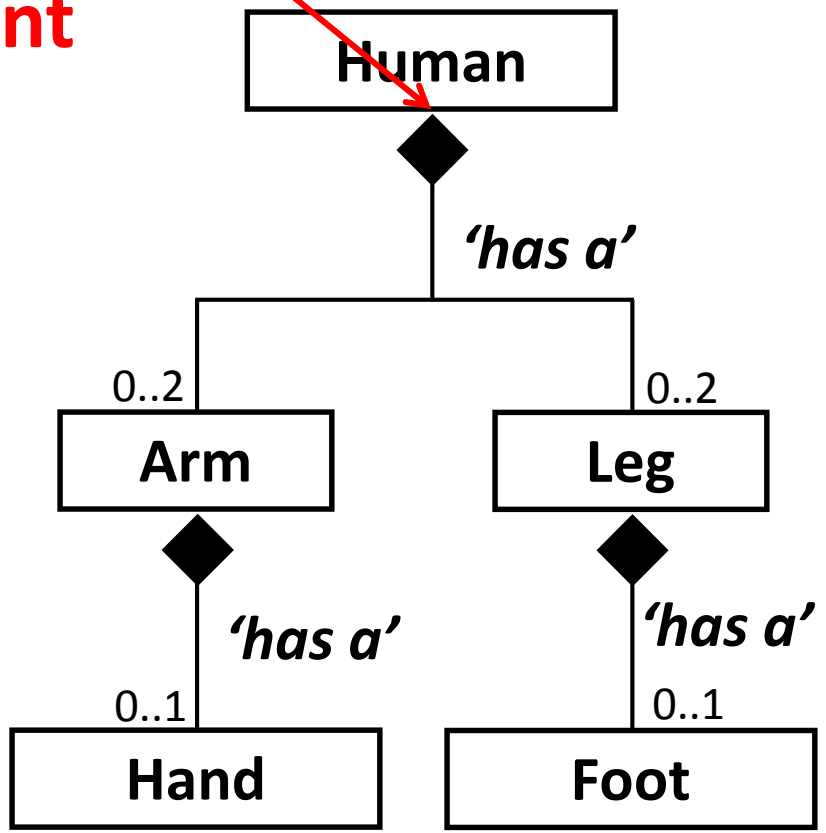
Aggregation and Composition

Whole – Part/s relationship

Containment





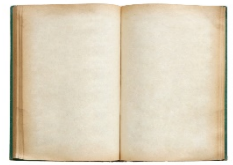


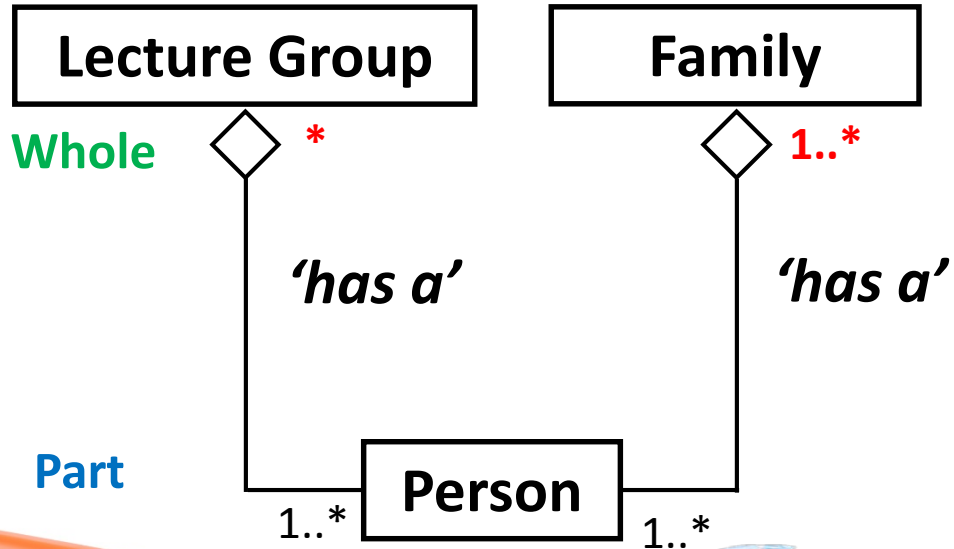
Whole creates/owns parts

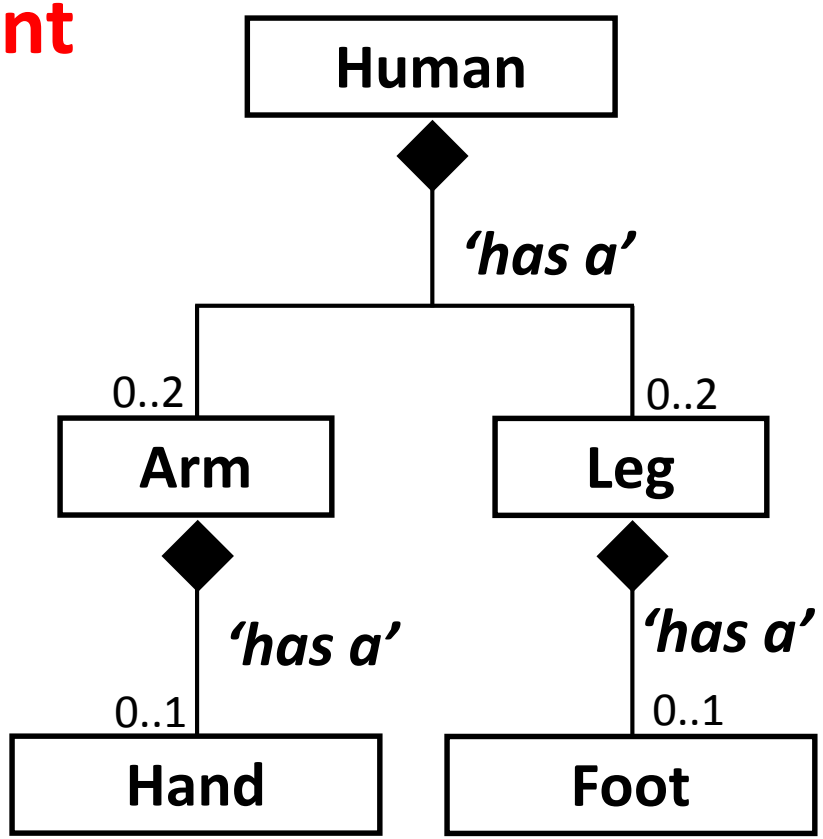
Aggregation and Composition

Whole – Part/s relationship

Containment



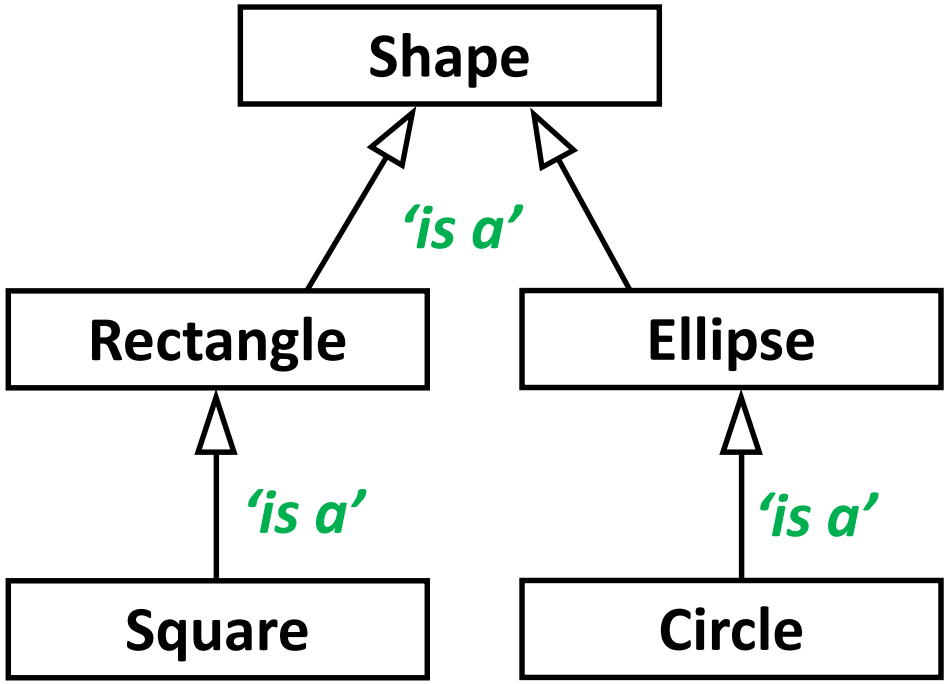


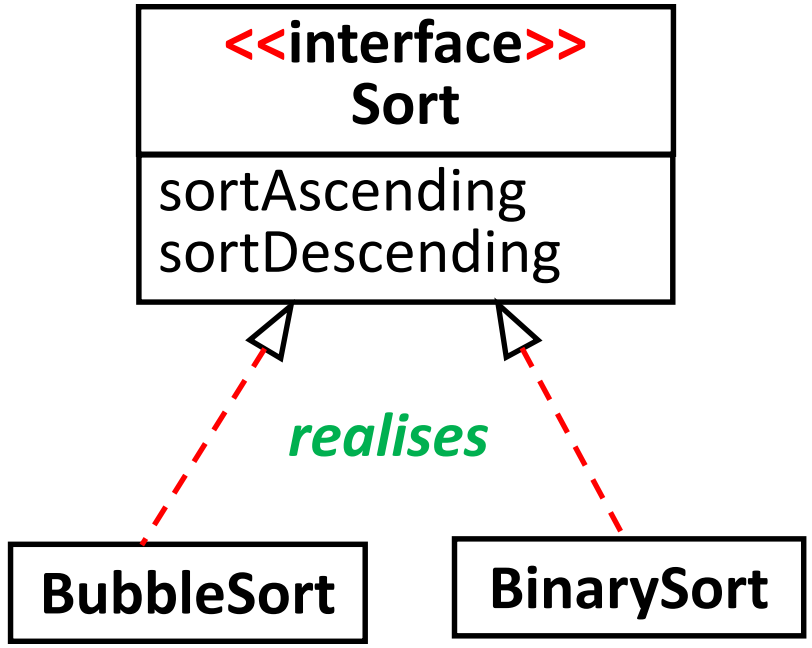


Whole creates/owns parts

Class Diagram – Generalisation and Interface Realisation

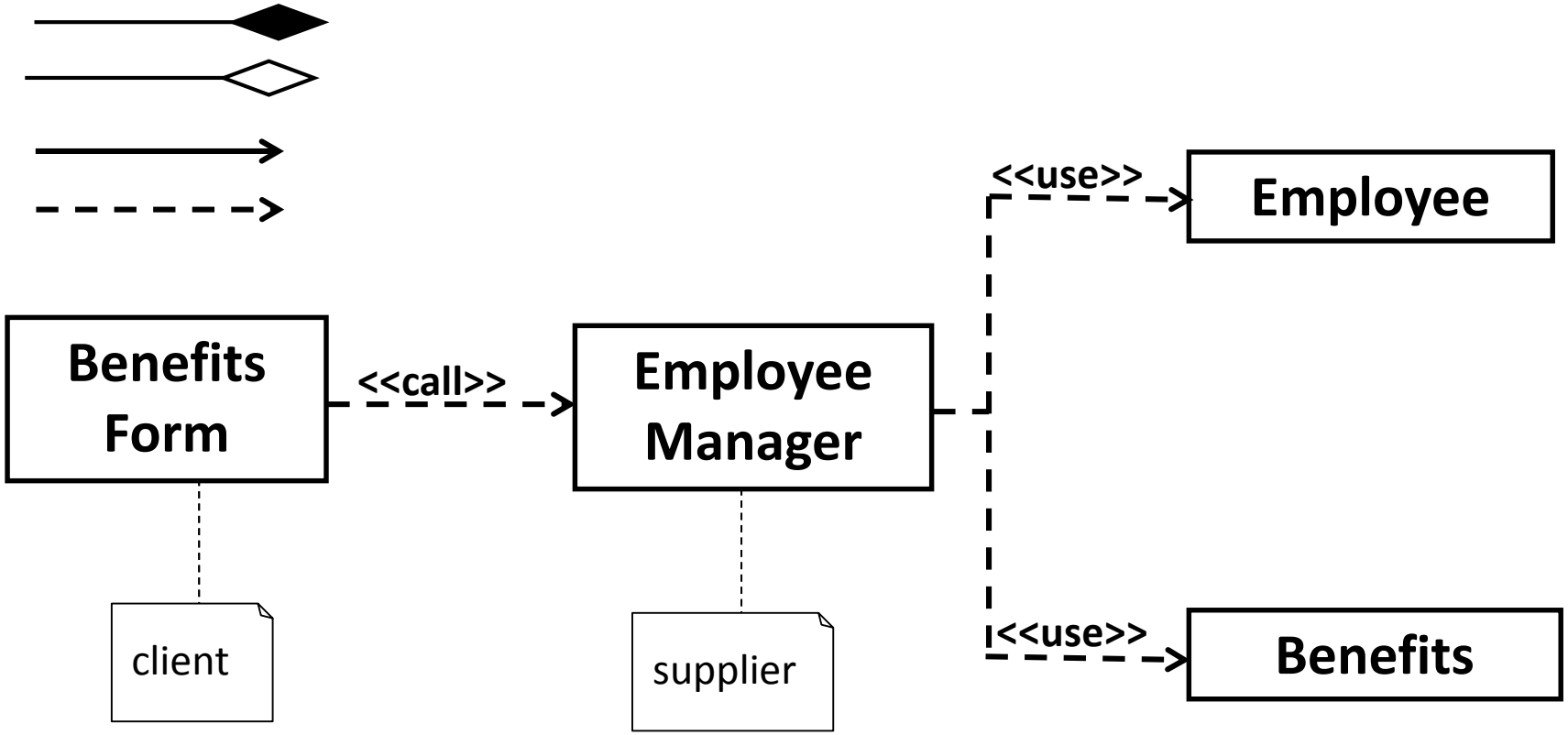
Refer to video at: 14:50





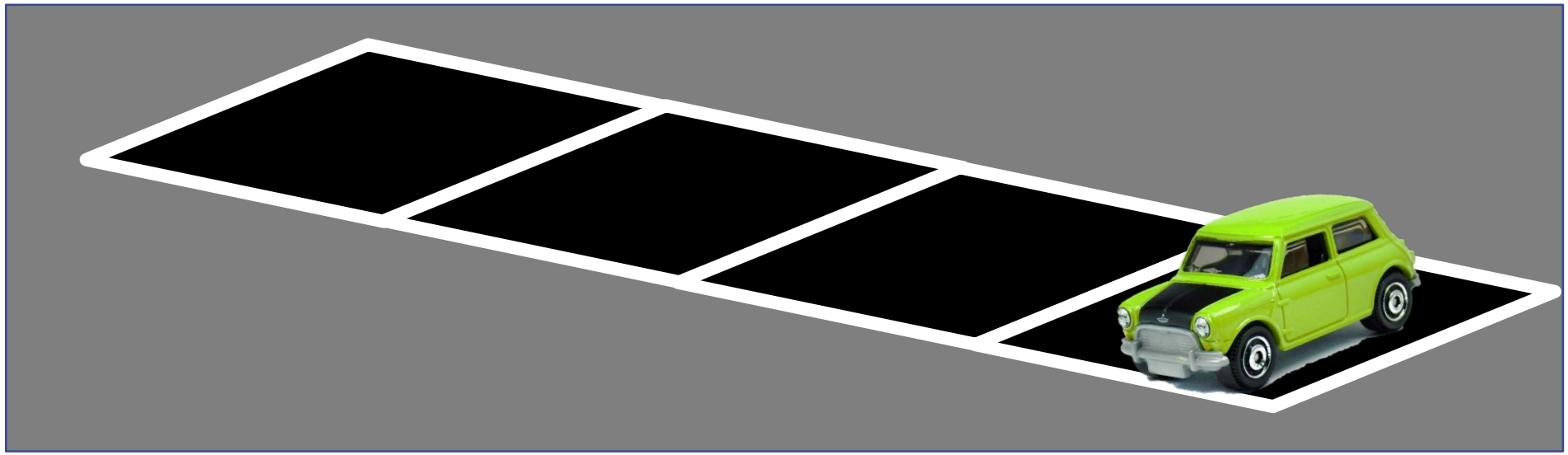
Class Diagram – Dependency

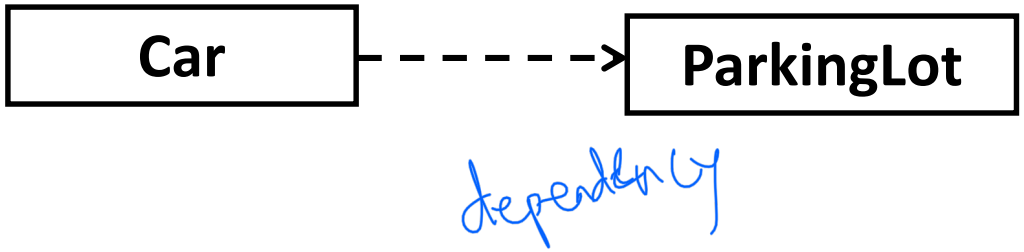
Refer to video at: 15:50



Dependency and Association

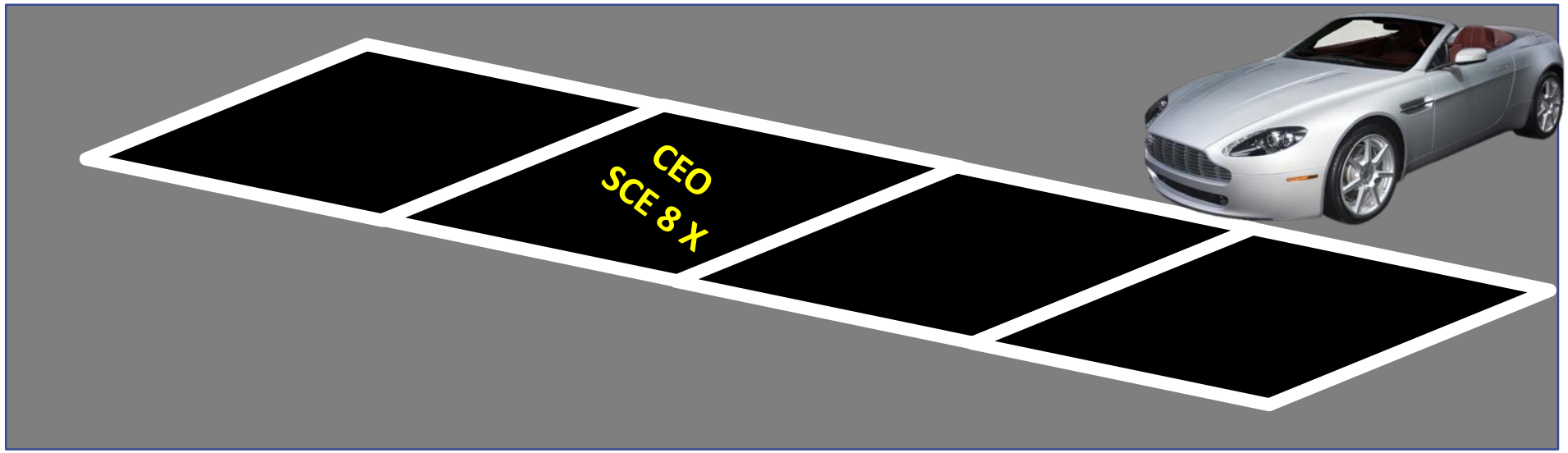
- What is the relationship between a **Car** and **ParkingLot** ?



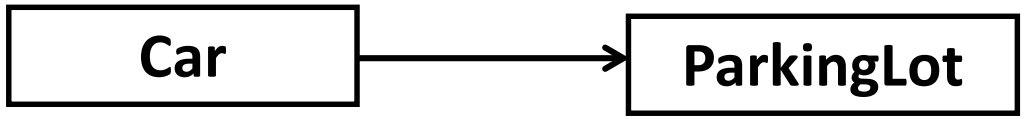


Dependency and Association

- What is the relationship between a **Car** and **ParkingLot** ?

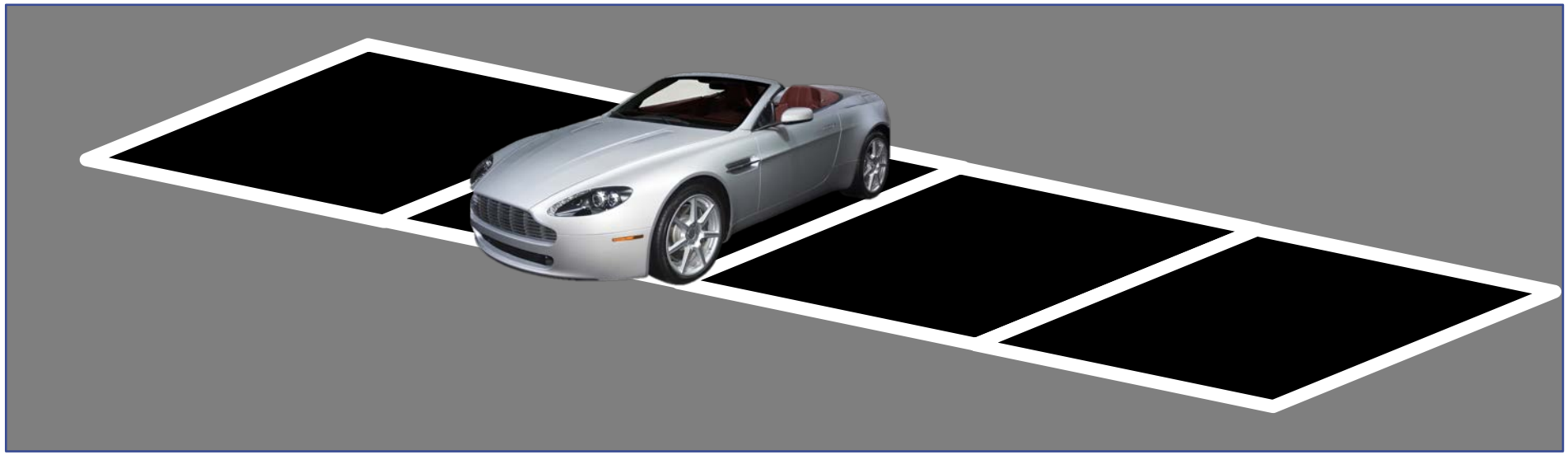


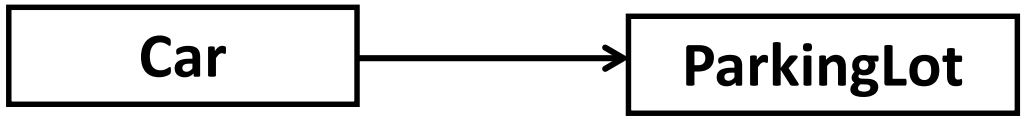
association



Dependency and Association

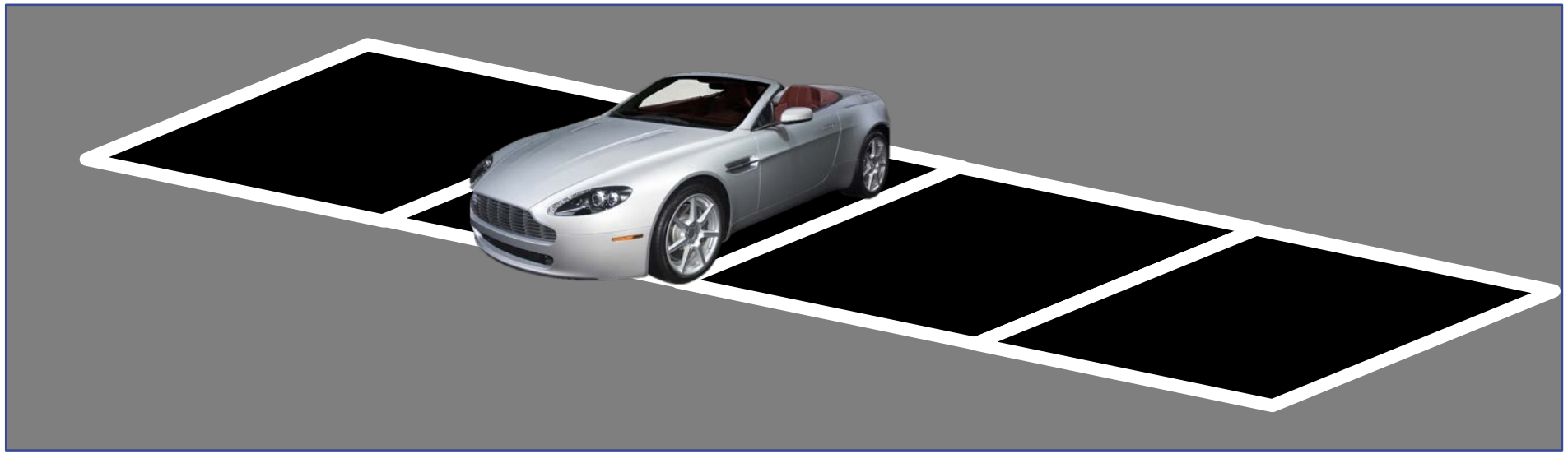
- What is the relationship between a **Car** and **ParkingLot** ?

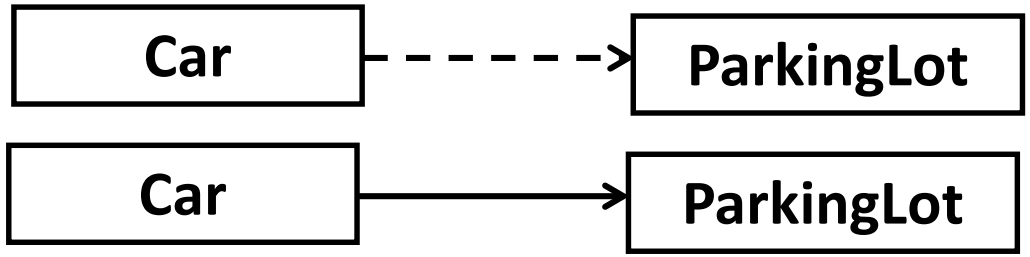




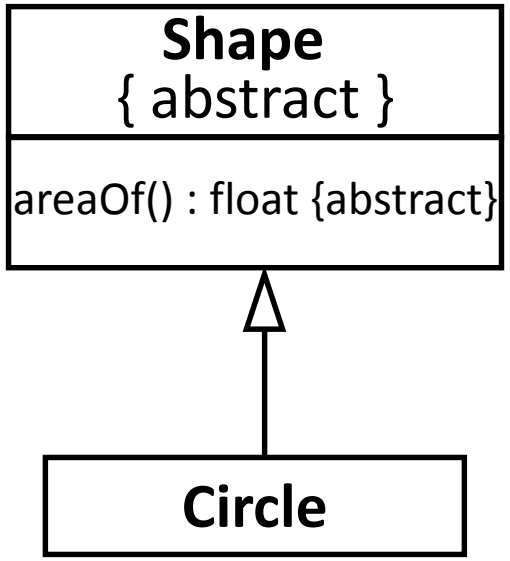
Dependency and Association

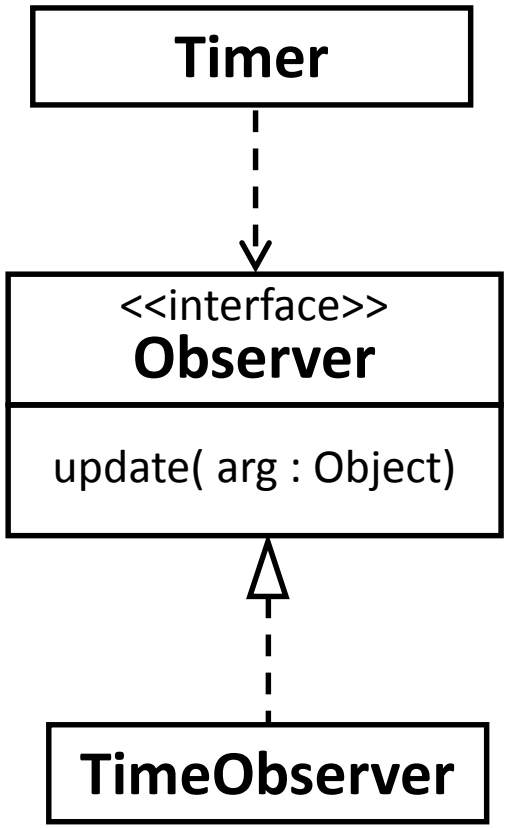
- What is the relationship between a **Car** and **ParkingLot** ?





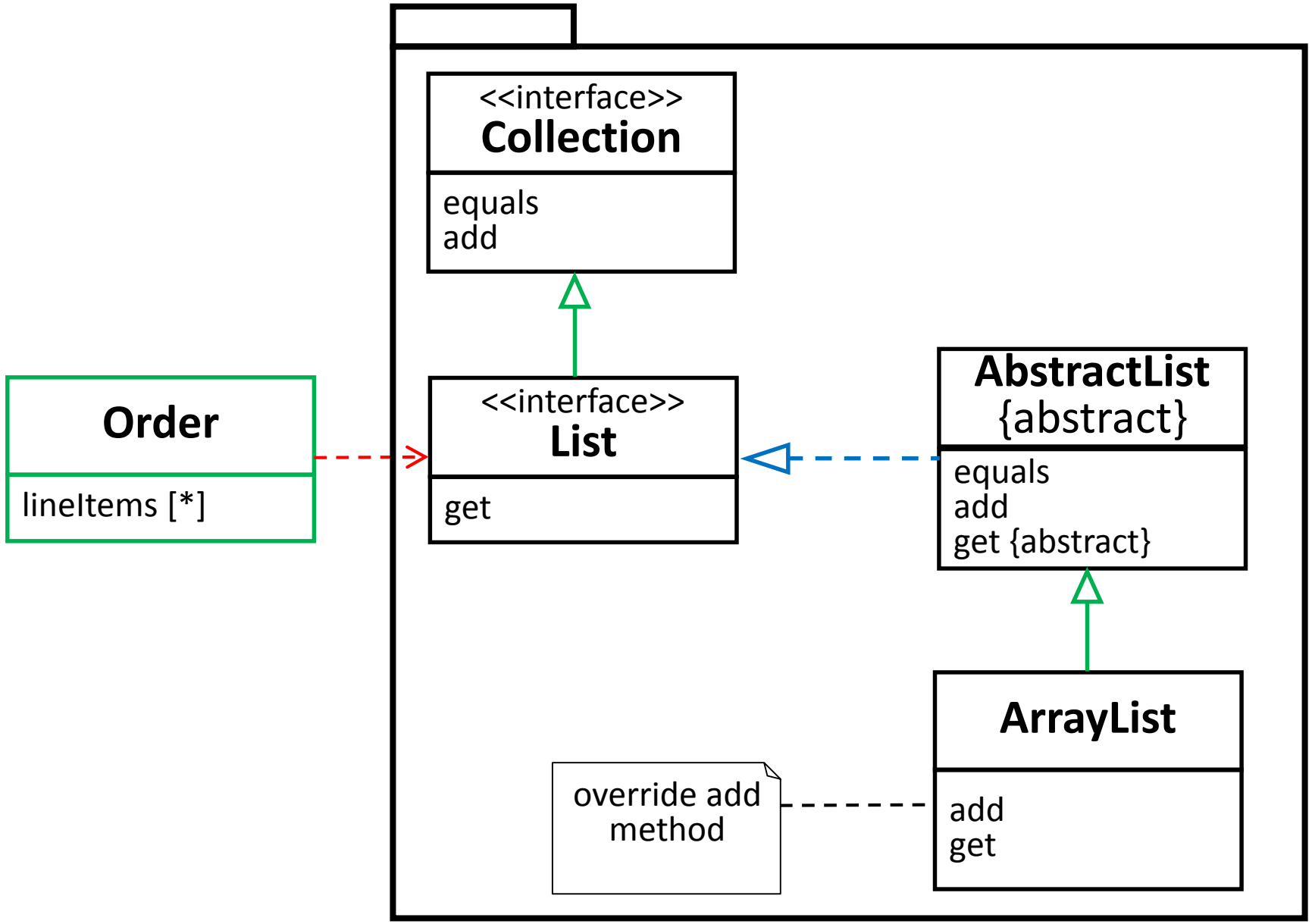
Abstract Classes and Interfaces



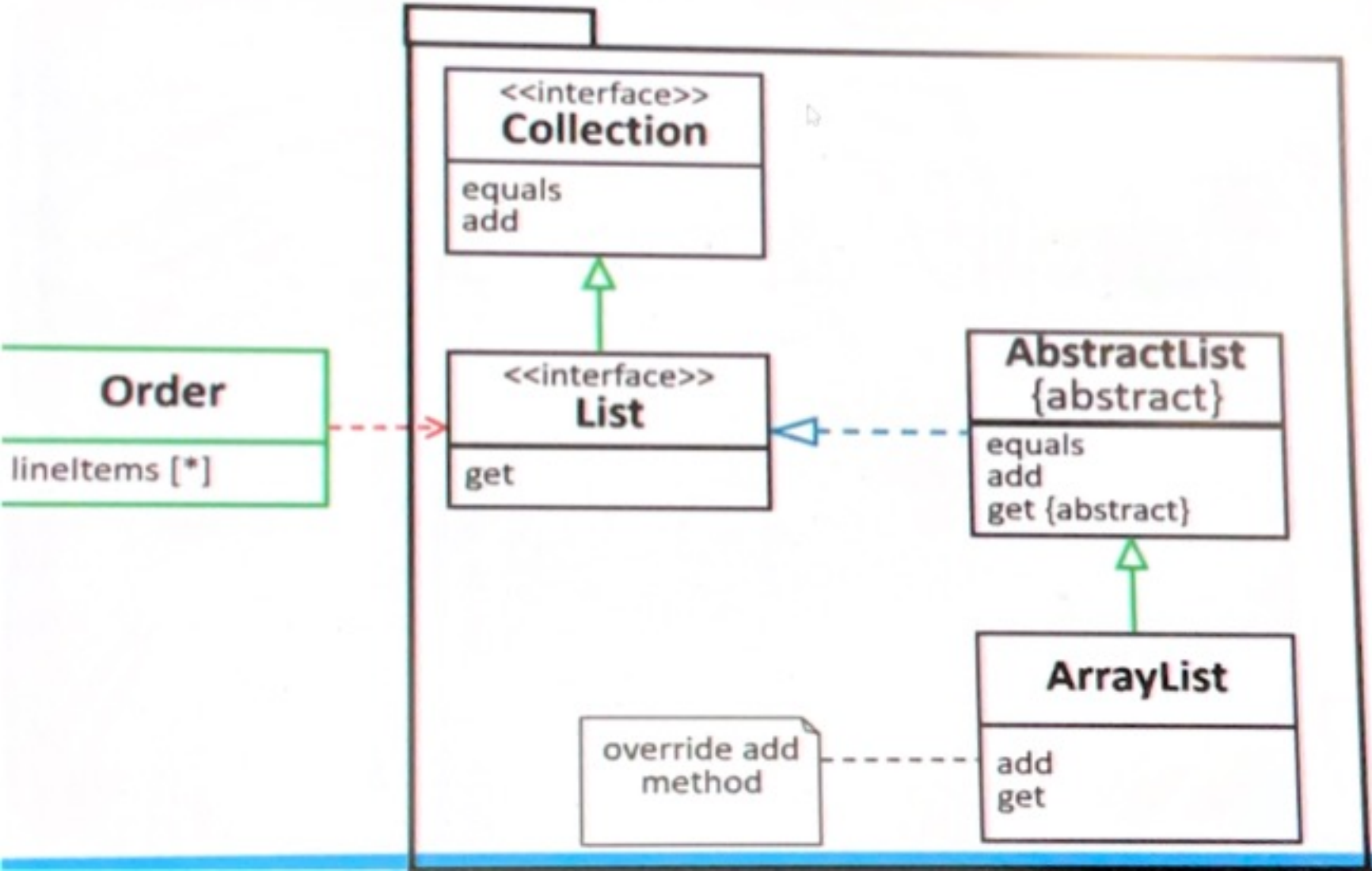


Java Collections Framework

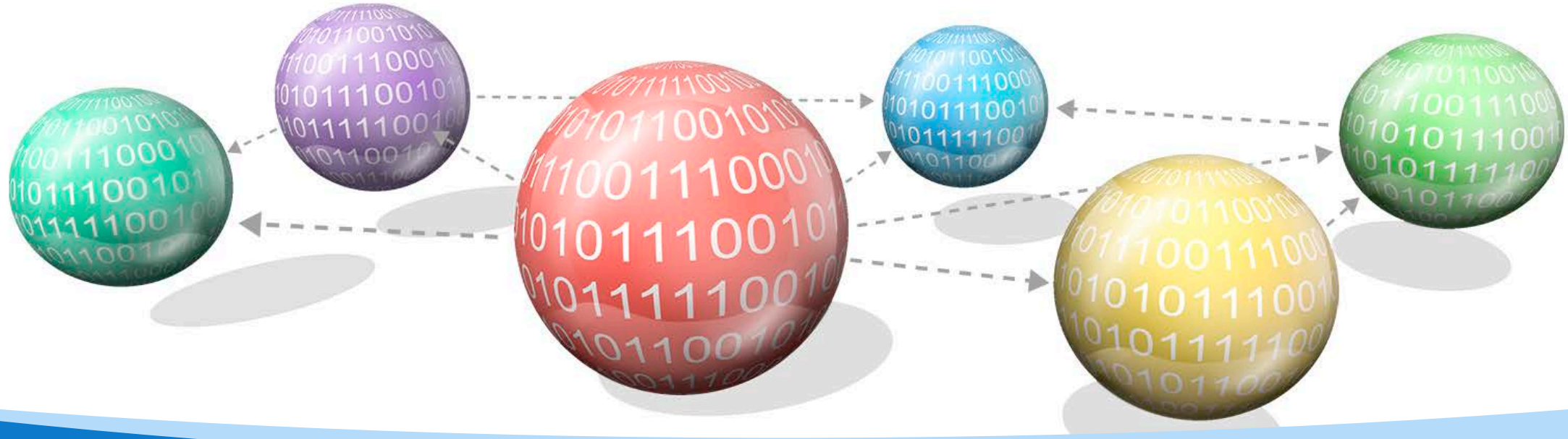
Refer to video at: 20:44



Java Collections Framework



- Interface inherits Interface,
- implementation only happen when class inherits interface.
- Abstract class are classes.



CE/CZ2002 Object-Oriented Design & Programming

Topic 3: UML Class Diagram to Java Code

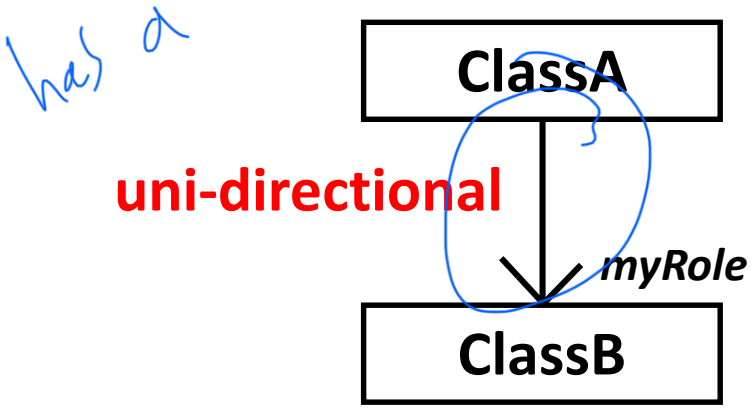
Chapter 6: UML Model Class Relationships - Class Diagram

Mr Tan Kheng Leong
Lecturer, School of Computer Science and Engineering

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

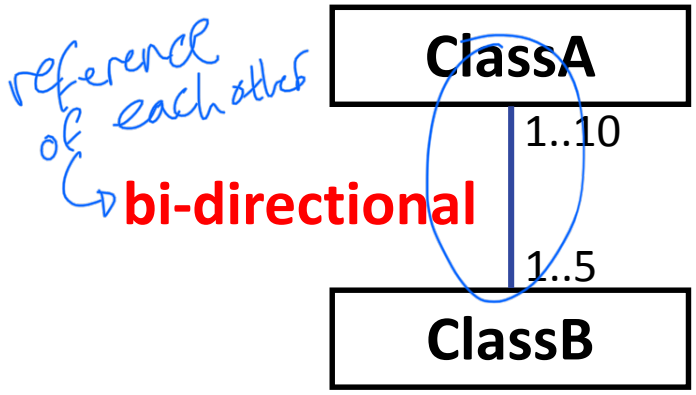
UML Class Diagram to Java Code

Association

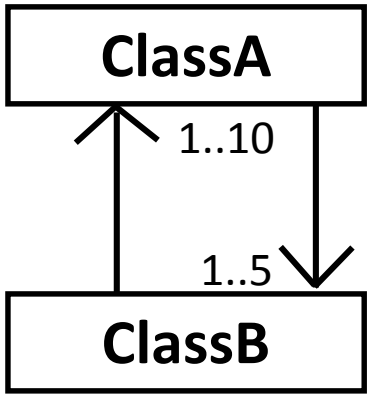



```
class  ClassA  {
       ClassB  myRole ;  // attribute
       ……..
       …..
}
```

```
class  ClassA  {
       ClassB[ ] bObjs = new ClassB[ 5 ];  // attribute
       ……..
……..                  bObjs[0] = new ClassB(this) ;
}
```

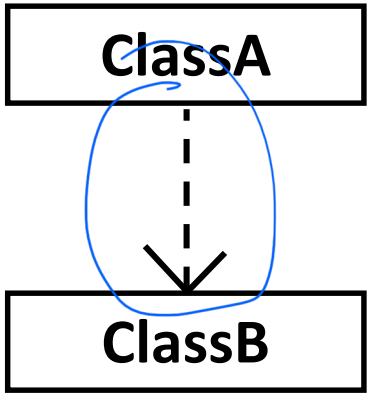



```
class  ClassB  {
   ClassA[ ] aObjs = new ClassA[ 10 ];  // attribute
       ……..
……..                  public ClassB(ClassA a) { aObjs[0] =  a  ; …}
}
```



UML Class Diagram to Java Code

Dependency



use a

Without attribute list of A in b then it is dependency

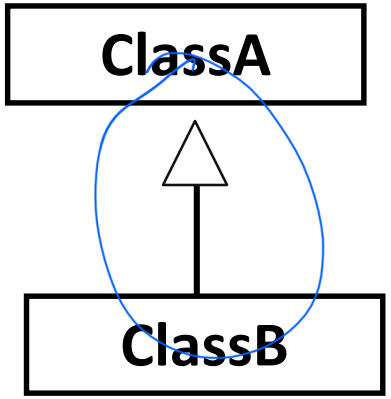
```
class ClassA {
      .....
       ClassB doSomeThing(...) { // as method return type
       ........
       .....
       }
}
```

```
class ClassA {
      .....
       <ReturnType> doSomeThing(ClassB b, ...) { // as method parameter type
       ........
       .....
       }
}
```

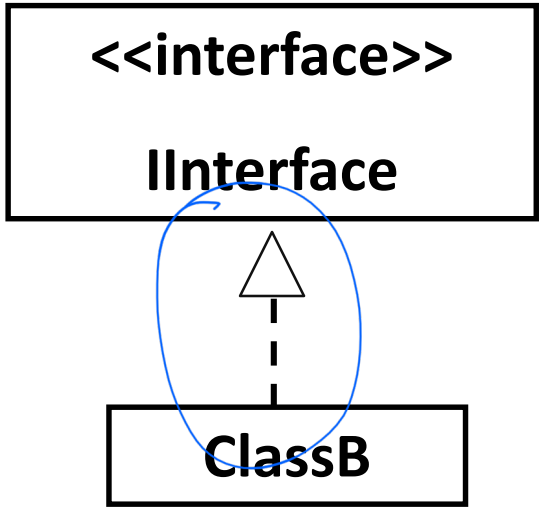
```
class ClassA {
      .....
       <ReturnType> doSomeThing(.....) {
              ClassB b = new ClassB() ; // as variable type within the method (local)
               .....
       }
}
```

UML Class Diagram to Java Code

Generalisation (Inheritance)




```
class  ClassB extends ClassA {
.....................
}
```




```
class  ClassB  implements  IInterface{
.....................
}
```

Realisation

AGGREGATION VERSUS COMPOSITION

AGGREGATION	COMPOSITION
An association between two objects which describes the "has a" relationship	The most specific type of aggregation that implies ownership
Destroying the owning object does not affect the containing object	Destroying the owning object affects the containing object
Diamond symbol represents the aggregation in UML	Highlighted diamond symbol represents the composition in UML

B Part of A

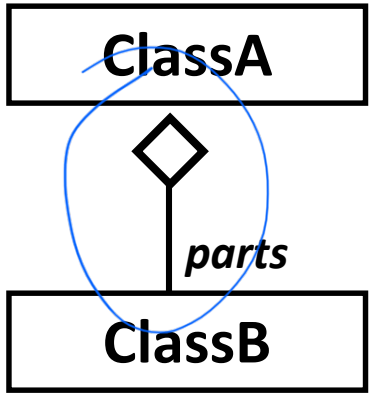
A — B

↓

normal association

UML Class Diagram to Java Code

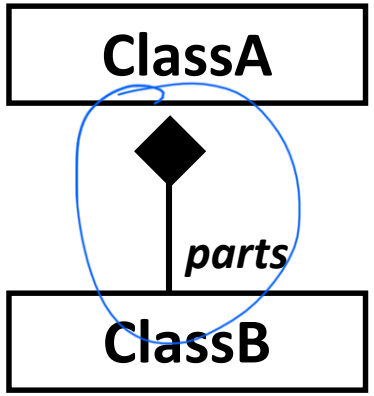
Aggregation




```
class  ClassA  {
      Vector  parts = new Vector() ;  // attribute
      ……..  // parts stores a collection of ClassB objs
      public void addClassB(ClassB  b ) {
            ……..// added through reference
               parts.add(b) ;
      }
     ……..……..
} // use of Vector class is just to show dynamic (*) Collection,
//  can be ArrayList, etc OR array[ ] (if size is fixed)
```

b Created Outside

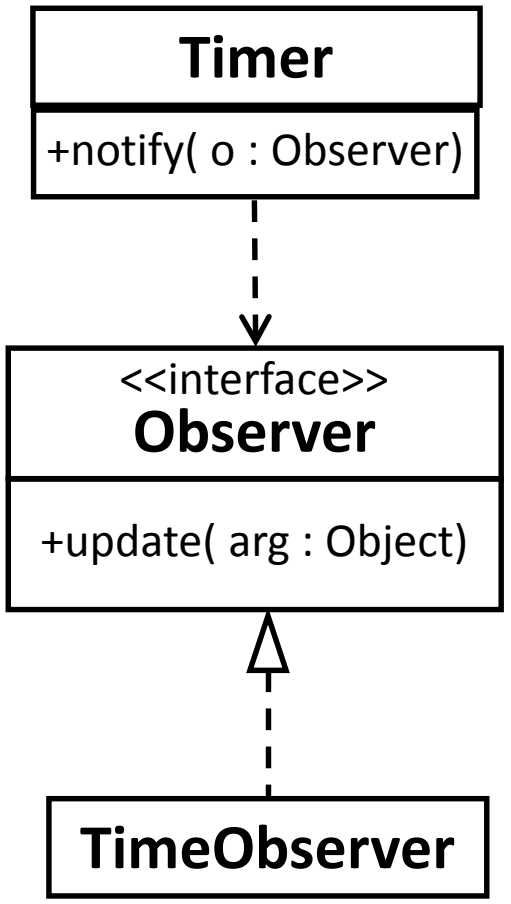
Composition




```
class  ClassA  {
      Vector  parts = new Vector();  // attribute
      ……..
      public ClassA( ) {
            ClassB  b1 = new ClassB() ;  // created in class
               ClassB b2 = new ClassB() ;
               parts.add(b1) ;
               parts.add(b2) ;
      }
     ……..……..
}
```

b Created inside

Example

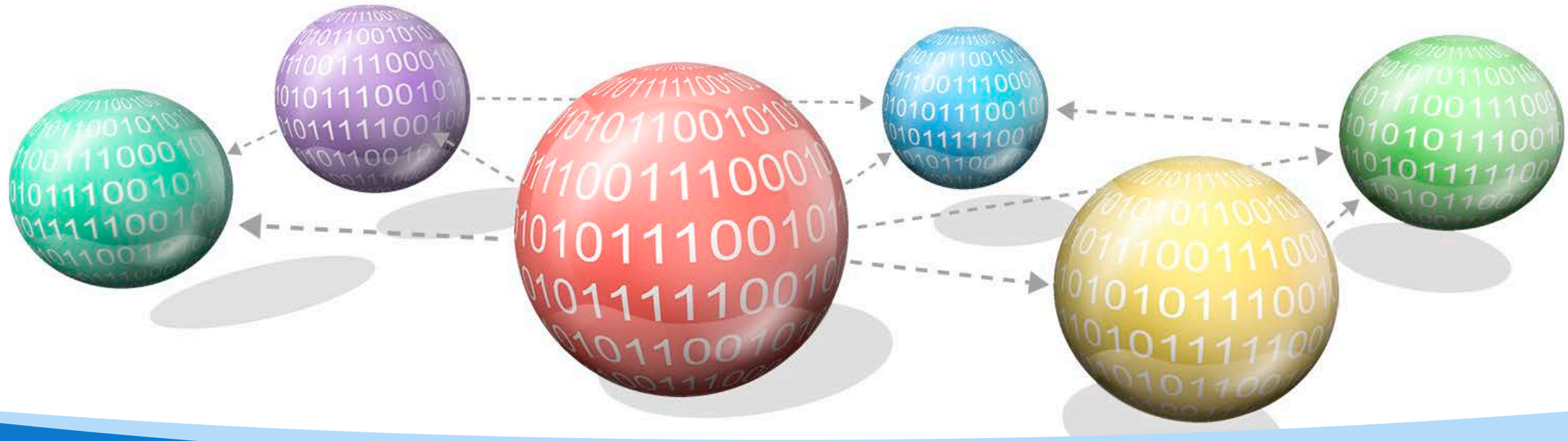



```
class  Timer{
   public void notify(Observer o) {

  }
}
```

```
public interface Observer{
    public void update(Object arg);
}
```

```
class  TimeObserver  implements  Observer{
            public void update(Object arg) {
        }
}
```



CE/CZ2002 Object-Oriented Design & Programming

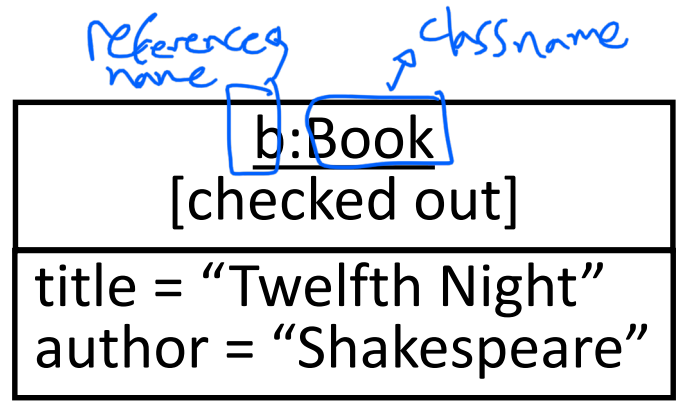
Topic 4: Object Diagram

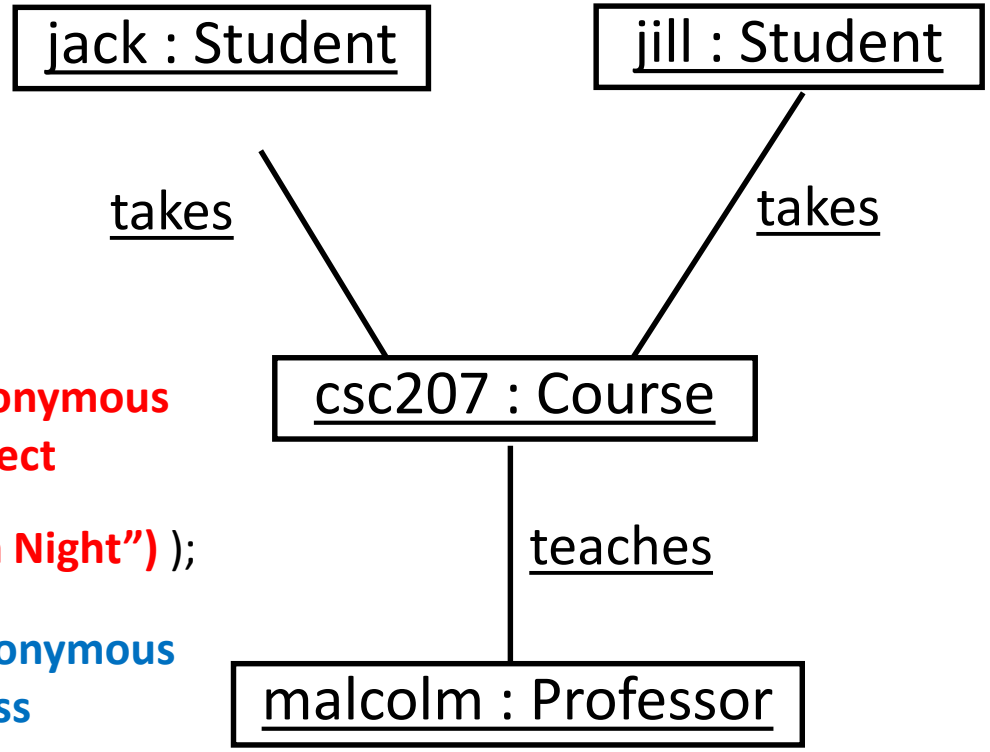
Chapter 6: UML Model Class Relationships - Class Diagram

Mr Tan Kheng Leong
Lecturer, School of Computer Science and Engineering

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Object Diagram

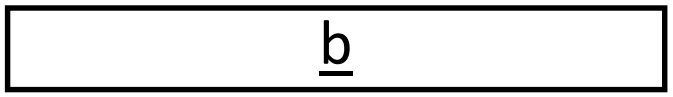




Anonymous object

Common in multi-threading

library.checkout(**new Book(“Twelfth Night”)**);



Anonymous class

```
Thread t = new Thread() {
        public void run() {
                // do something……
        } };
t.run();
```

new interface-or-class-name() { class-body }

More about Class Diagram

Student

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

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

Employee

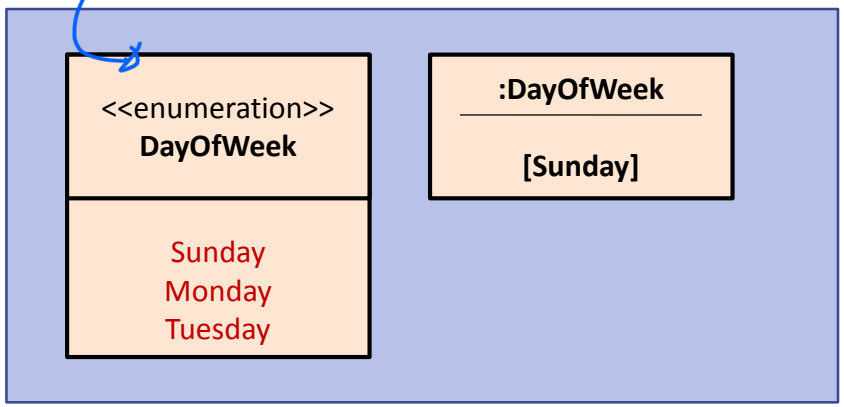
-name:String
-payRate:double
-EMPLOYEE_ID:int
-nextID:int
+STARTING_PAY_RATE:double

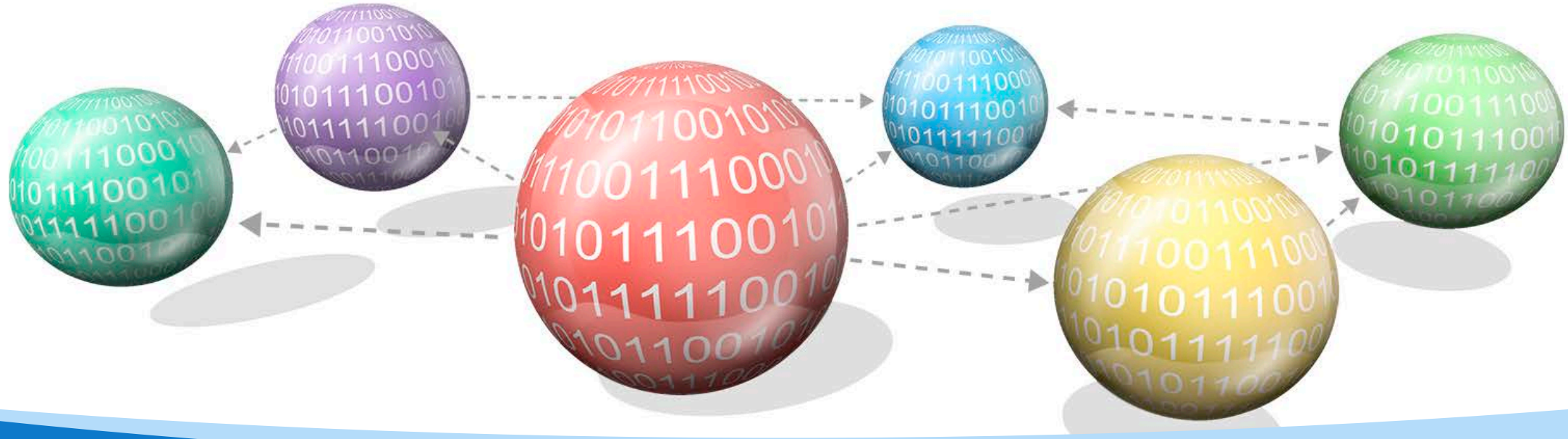
+Employee(String)
+Employee(String, double)
+getName():String
+getEmployeeID():int
+getPayRate():double
+changeName(String):void
+changePayRate(double):void
+getNextID():int

- visibility:
  - \+ public
  - # protected
  - \- private
  - ~ package (default)
- underline static methods

For Java

Can be use for constant





CE/CZ2002 Object-Oriented Design & Programming

Topic 5: Class Stereotypes

Chapter 6: UML Model Class Relationships - Class Diagram

Mr Tan Kheng Leong
Lecturer, School of Computer Science and Engineering

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Class Stereotypes

Entity

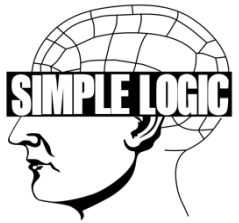
- Persistent information tracked by Program/Application/System

Noun with alot of description

Boundary

(input output)

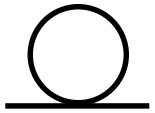
- Interaction between System and External (System Surrounding) - interfaces



Control

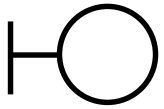
- Logic to coordinate and realise use case (functional usage)

Class Stereotypes (Common Naming)



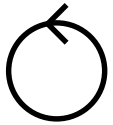
Entity

- Example: Student, Course, Group



Boundary

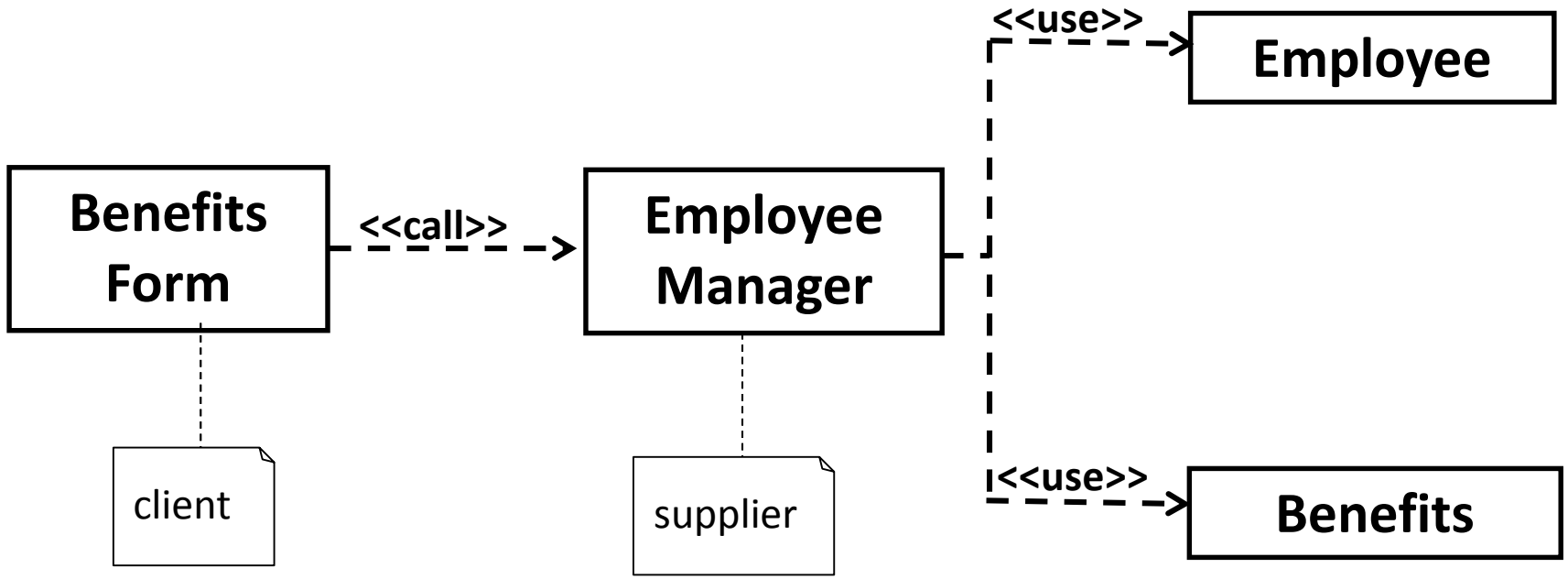
- Example: xxxUI, xxxForm, xxxInterface
- Example: InvoiceForm



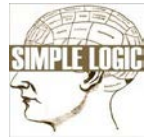
Control

- Example: xxxMgr, xxxCtrl, xxxController
- Example: CourseMgr

Stereotyping to Indicate Class Responsibilities



Stereotyping to Indicate Class Responsibilities



Benefits Form

<<call>>

Employee Manager

<<use>>

Employee

<<use>>

Benefits

DB

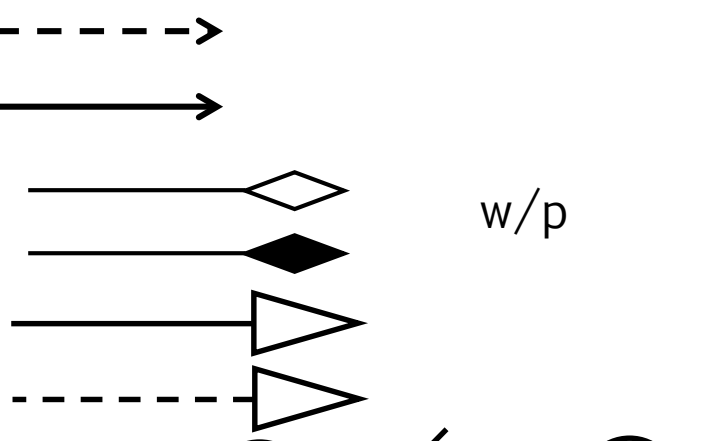
DB

client

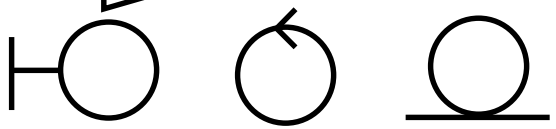
supplier

Summary

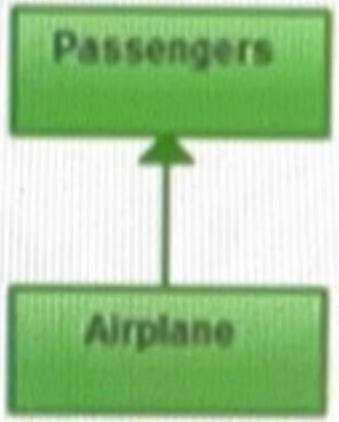
- Class Diagram Notations
- Type of Class relationships
  - Dependency
    - Association
      - Aggregation
        - Composition
    - Generalisation
    - Realisation
- Stereotype responsibilities



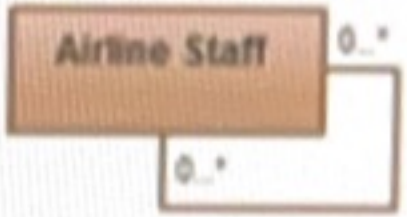
w/p



Association

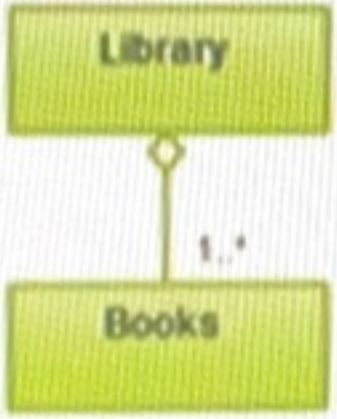


Directed
Asscoation

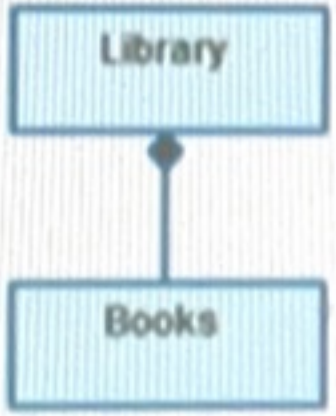


Reflexive
Assciation

Multiplicity

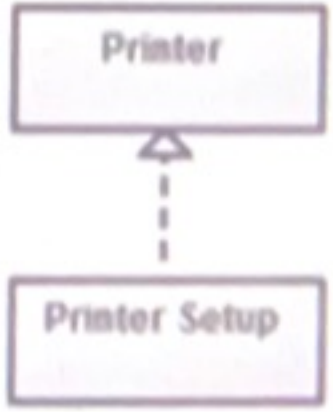


Aggregation



Composition

Inheritance



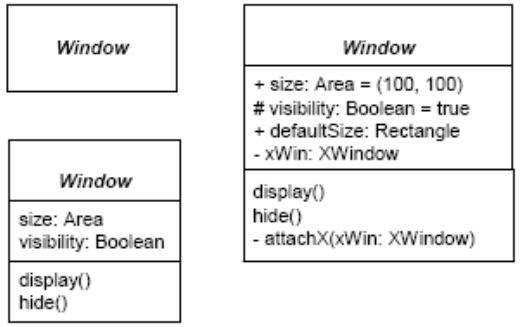
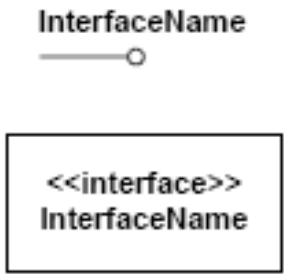
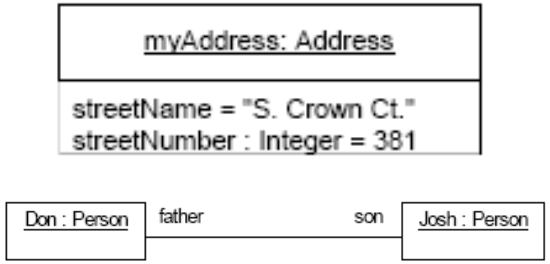
Realization

Further Reading

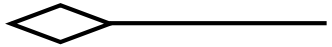
- UML Distilled
  - CHAPTER 3
    - *CLASS DIAGRAMS: THE ESSENTIALS*
  - CHAPTER 5
    - CLASS DIAGRAMS: ADVANCED CONCEPTS
  - CHAPTER 6
    - OBJECT DIAGRAMS

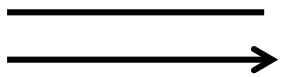
- UMLSpecNotes

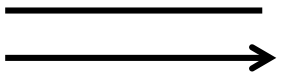
Further Reading: Graphic Nodes Included in Structure Diagrams

Node Type	Notation	Description
Class		Specifies a classification of objects, as well as, the features that characterise the structure and behaviour of those objects.
Interface		An interface is a kind of classifier that represents a declaration of a set of coherent public features and obligations. An interface specifies a contract; any instance of a classifier that realises that the interface must fulfill that contract.
Instance Specification		Instances of any classifier can be shown by prefixing the classifier name by the instance name followed by a colon and underlining the complete name string. An instance specification whose classifier is an association describes a link of that association.

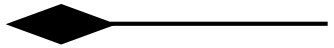
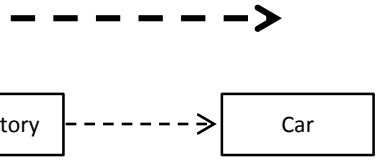
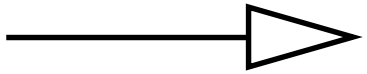
Further Reading: Graphic Paths Included in Structure Diagrams

Path Type	Notation	Description
Aggregation		An aggregation represents a whole/part relationship.

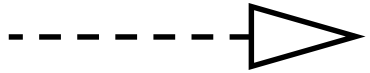
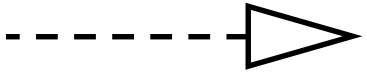
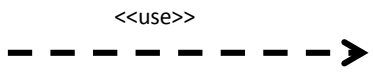
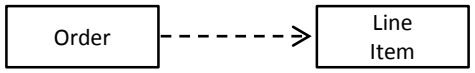
Association		An association specifies a semantic relationship that can occur between typed instances.
		
		An open arrowhead on the end of an association indicates the end is navigable. A small x on the end of an association indicates the end is not navigable.
		
		Notations that can be placed near the end of the line as follows:
		• name – name of association end
		• multiplicity
		• property string enclosed in curly braces
		  ○ {ordered} to show that the end represents an ordered set.
		  ○ {bag} to show that the end represents a collection that permits the same element to appear more than once.
		  ○ {sequence} or {seq} to show that the end represents a sequence

Link		An instance of an association

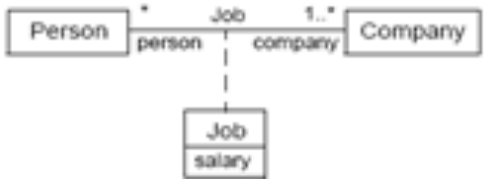
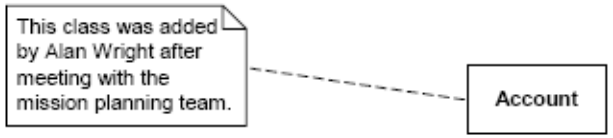
Further Reading: Graphic Paths Included in Structure Diagrams

Path Type	Notation	Description
Composition		Composite aggregation is a strong form of aggregation that requires a part instance be included in at most one composite at a time. If a composite is deleted, all of its parts are normally deleted with it. Deleting an element will also result in the deletion of all elements of the subgraph below that element.
Dependency	 **instantiate dependency**	A dependency is a relationship that signifies that a single or a set of model elements requires other model elements for their specification or implementation. The modification of the supplier may impact the client model elements.
Generalisation		A generalisation is a taxonomic relationship between a more general classifier and a more specific classifier. Generalisation hierarchies must be directed and acyclical.

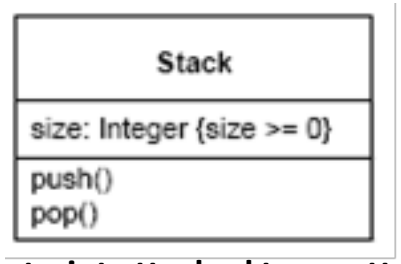
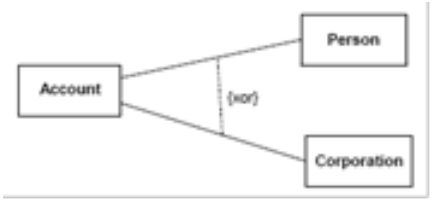
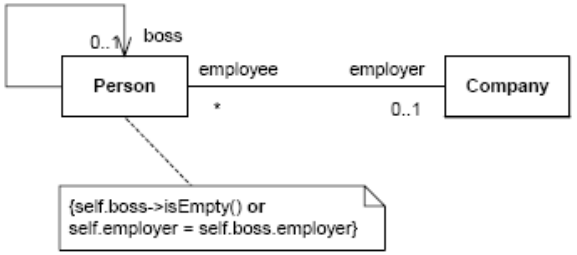
Further Reading: Graphic Paths Included in Structure Diagrams

Path Type	Notation	Description
Interface Realisation		A specialised realisation relationship between a Classifier and an Interface. This relationship signifies that the realising classifier conforms to the contract specified by the Interface. A classifier may implement a number of interfaces. The set of interfaces implemented by the classifier are its provided interfaces and signify the set of services the classifier offers to its clients.
Realisation		Signifies that the client set of elements are an implementation of the supplier set, which serves as the specification.
Usage	  An Order class requires the Line Item class for its full implementation.	A usage is a relationship in which one element requires another element (or set of elements) for its full implementation or operation.

Further Reading: Miscellaneous Elements Included in Structure Diagrams

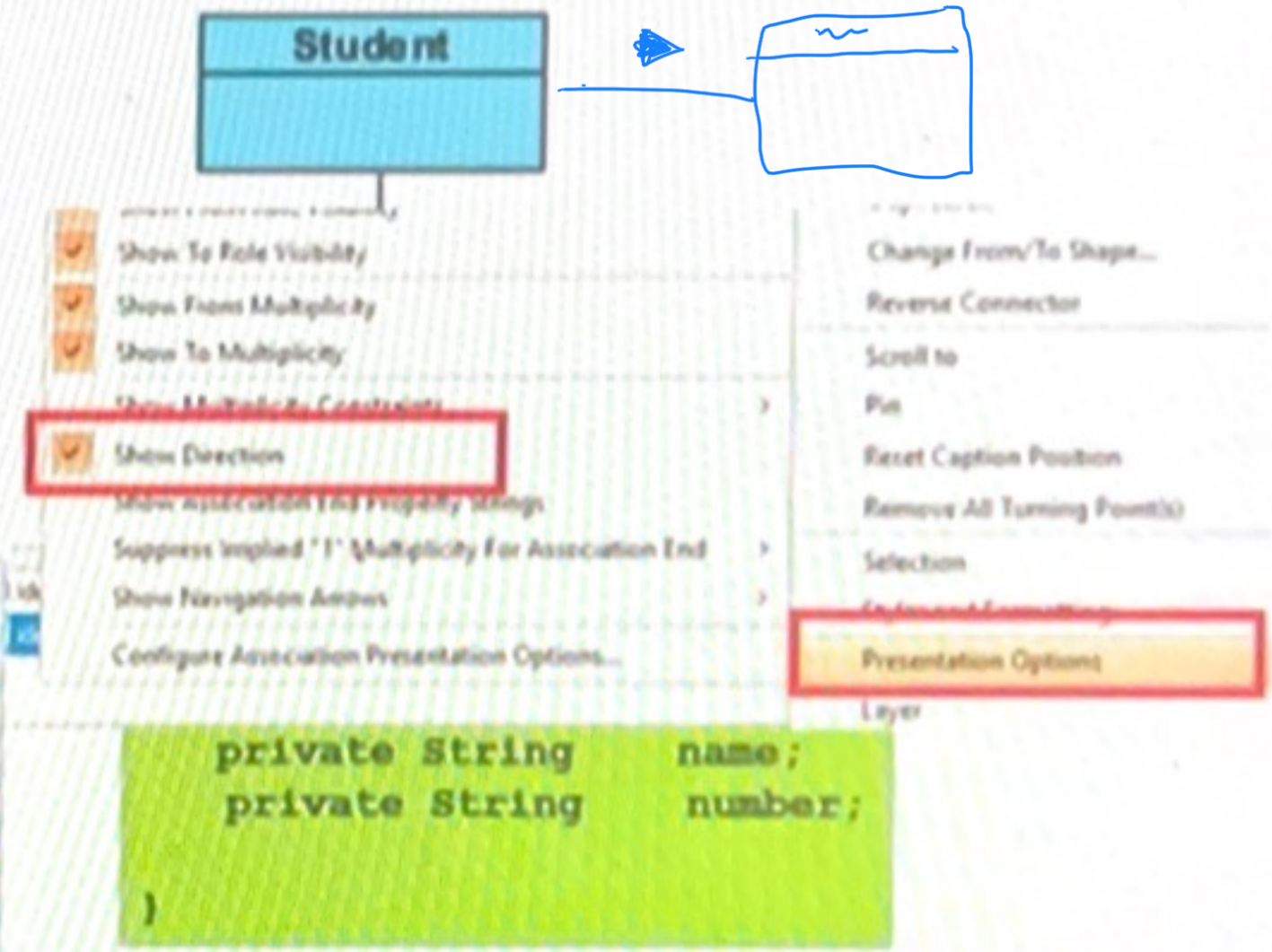
Type	Notation	Description
Association Class		Defines a set of features that belong to the relationship itself and not to any of the classifiers.
Comment		A comment is a textual annotation that can be attached to a set of elements. A comment gives the ability to attach various remarks to elements. A comment carries no semantic force, but may contain information that is useful to a modeler. The connection to each annotated element is shown by a separate dashed line.

Further Reading: Miscellaneous Elements Included in Structure Diagrams

Type	Notation	Description
Constraint	 **Constraint attached to an attribute**  **{xor} constraint**  **Constraint in a note symbol**	A constraint is a condition or restriction expressed in natural language text or in a machine readable language for the purpose of declaring some of the semantics of an element. One predefined language for writing constraints is OCL.

Directed Association (Uni-directional Association)

You can define the flow of the association by using a directed association. The arrowhead identifies the container-contained relationship.

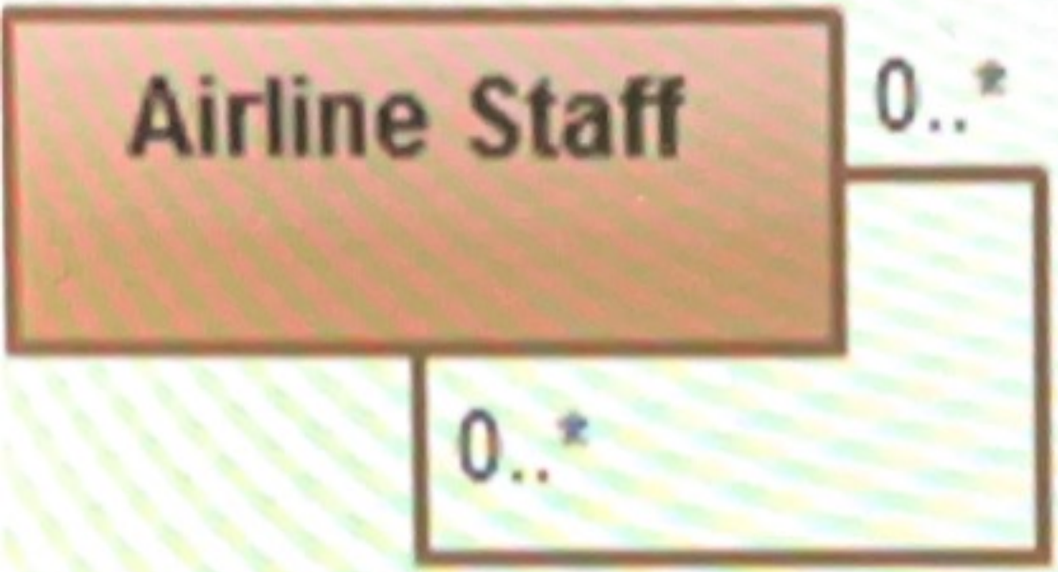



```
public class Student {
    private int     studentId;
    private char    gender;
    private double  height;
    private int     age;
    private CreditCard[] cards;
...
}
```

```
    private String     name;
    private String     number;

}
```

Reflexive Association



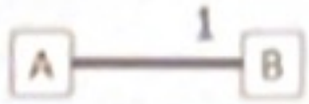
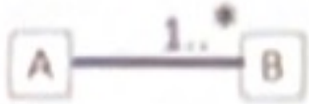
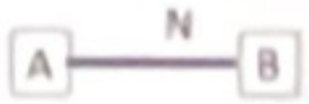
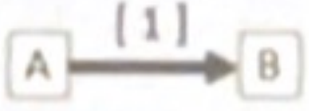
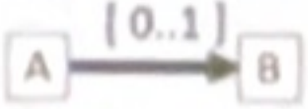

```
Public class Staff {
    Staff manager
    Staff cleaner
}
```

No separate symbol. However, the relation will point back at the same class.

This occurs when a class may have multiple functions or responsibilities. For example, a staff member working in an airport may be a pilot, aviation engineer, a ticket dispatcher, a guard, or a maintenance crew member. If the maintenance crew member is managed by the aviation engineer there could be a managed by relationship in two instances of the same class.

Multiplicity

only for Has a relationship

Reading from Left to Right	Each A must be assigned to exactly one B	Each A must be assigned to one and many of B	Each A must be assigned to zero or one of B	Each A may be assigned any number of Bs
UML				
Booch (2nd Edition)				
Jacobson (unidirectional)				
OMT (Object-modeling technique)				

Multiplicities examples:

1	Exactly one, no more and no less
0..1	Zero or one
*	Many
0..*	Zero or many → list of Collection
1..*	One or many