CMPS310

Fall 2021

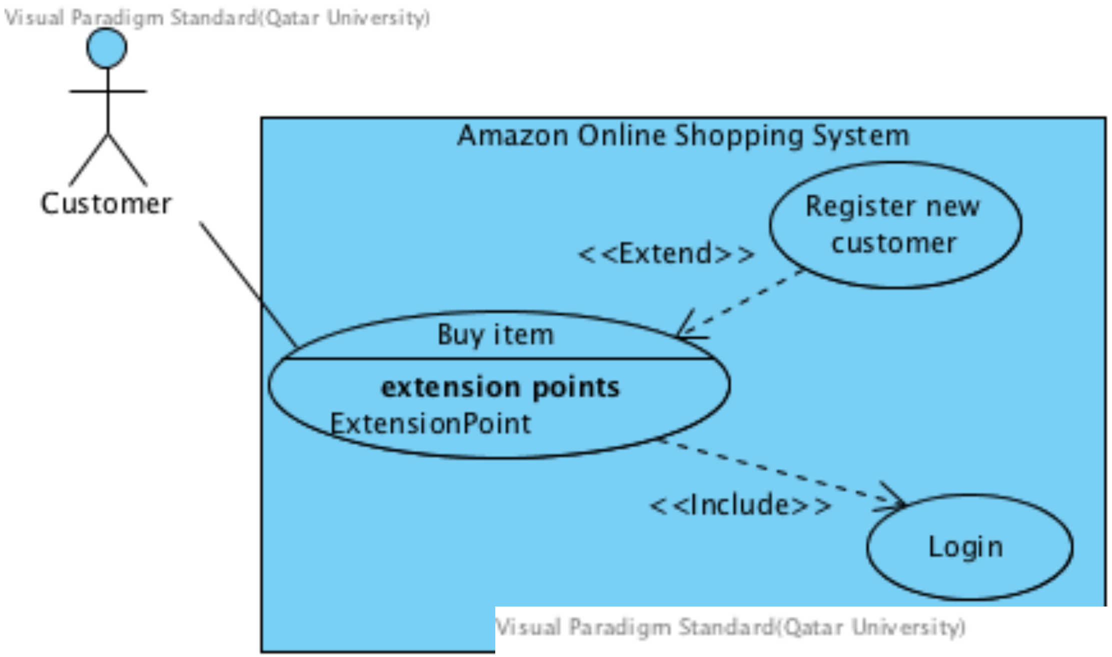
Lecture 6

More on Class Relationships in Design Class Diagram

Class Diagram: Basic Rules

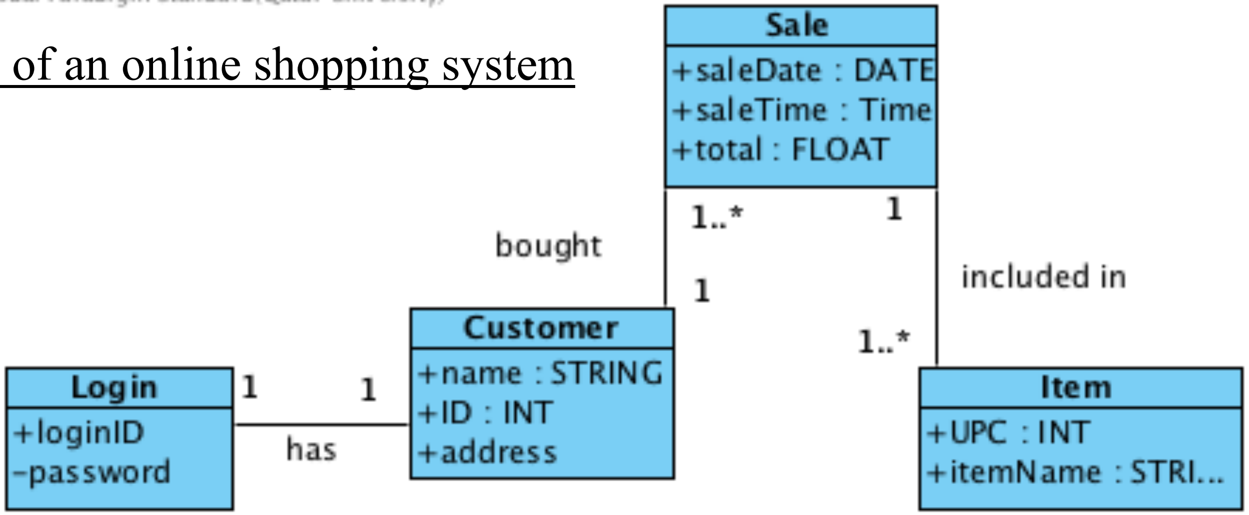
- An actor of a use case diagram will not always be a class in the class diagram – here are two scenarios:
  - **Customer** is an **actor** also a **class**
    - Examples: Online Shopping System
      - Amazon, eBay
      - Why?
  - **Customer** is an **actor**, but **not a class**
    - Examples: Retail Shopping System
      - Carrefour retail sale system
      - Why?
- An actor in a use case diagram can be a class in a class diagram **if the instance of the actor is used or manipulated in the system,** otherwise not
- **A use case will never be a method of a class**
- A use case is NOT a mere method, it is more than that
- A use case embodies multiple operations represented by methods of objects used in the system.

Example: Actor-and-Class


Visual Paradigm Standard(Qatar University)
Customer
Amazon Online Shopping System
Register new customer
<<Extend>>
Buy item
extension points
ExtensionPoint
<<Include>>
Login
Visual Paradigm Standard(Qatar University)

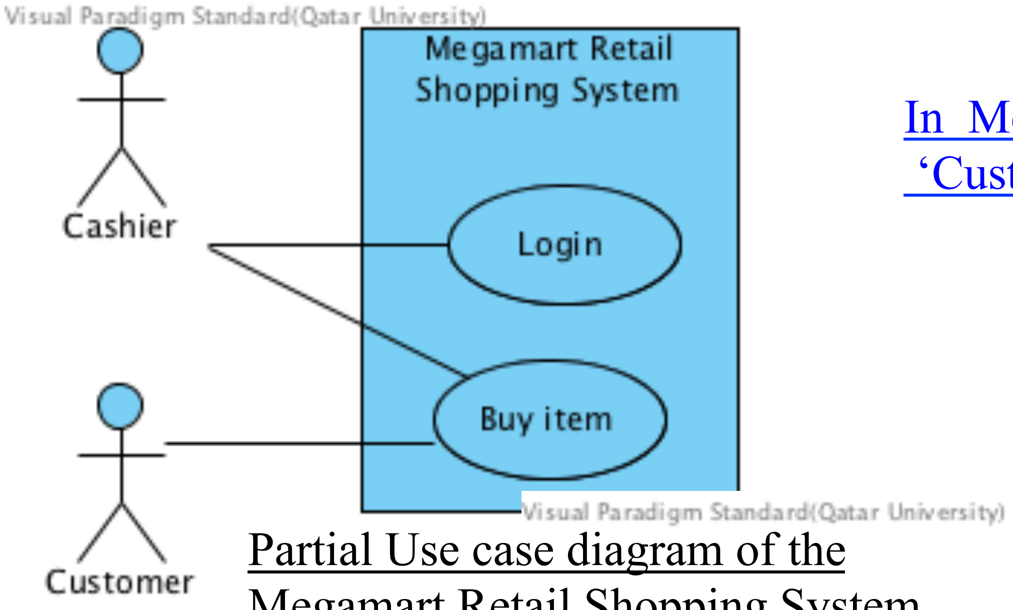
In an online shopping system where 'Customer" is an actor, and also a class.

Partial Use case diagram of an online shopping system



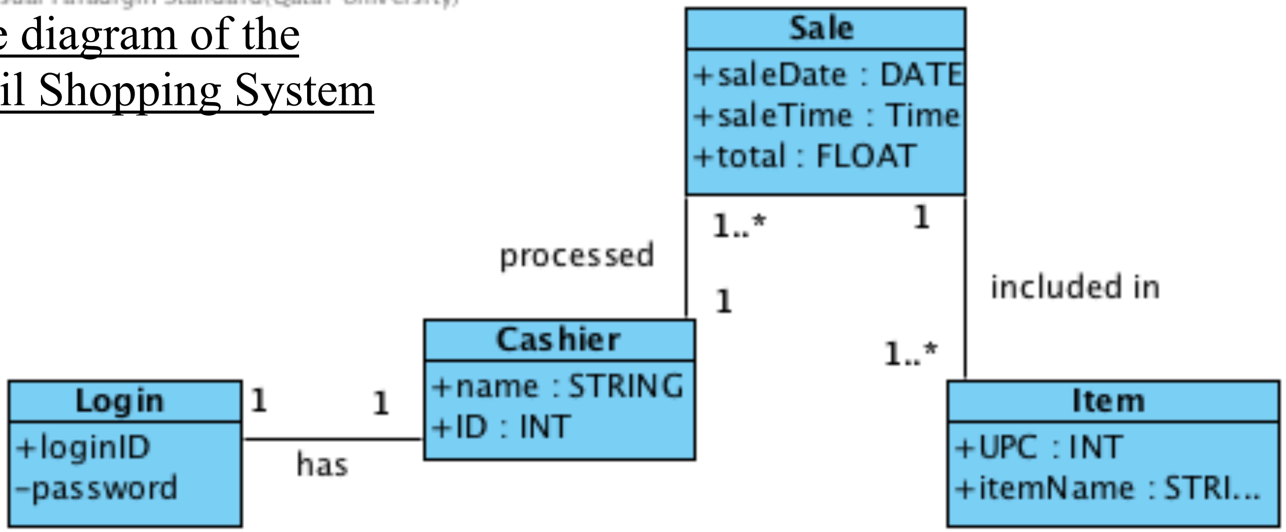
Partial Class diagram of the online shopping system

Example: Actor-not-as-Class


Visual Paradigm Standard(Qatar University)
Megamart Retail Shopping System
Cashier
Login
Buy item
Customer
Visual Paradigm Standard(Qatar University)

In MegaMart Retail Shopping system, the 'Customer' is an actor, but **not a class**

Partial Use case diagram of the Megamart Retail Shopping System



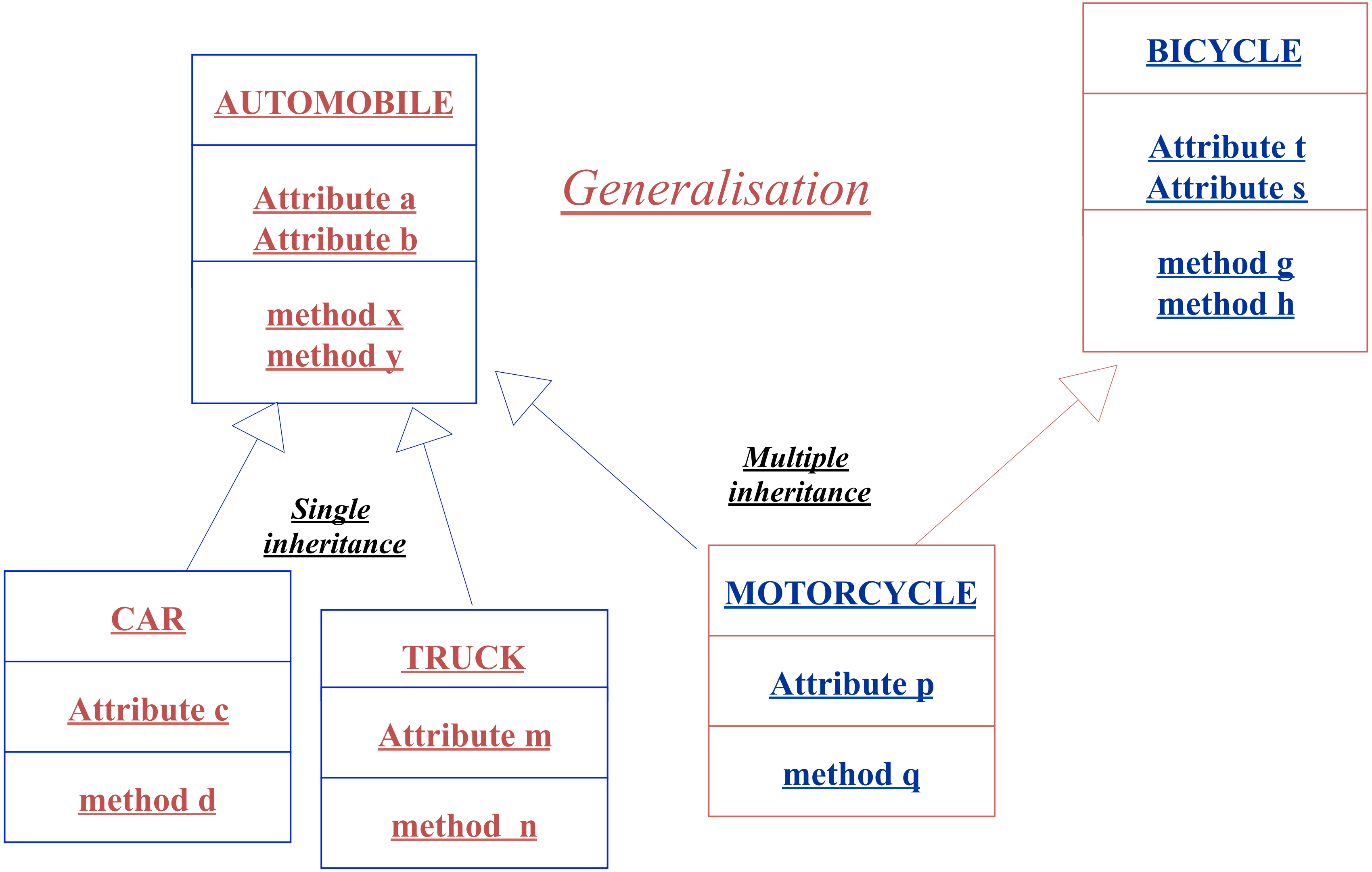
Partial Class diagram of the Mega Mart Retails Shopping system

Use Case Can Not Be Method

- A use case cannot be a method of a class
- A use case is a complete functionality that embodies several methods
- How and from where do you find the responsibilities of a use case?
- The responsibilities of a use case are specified in the normal scenario of the use case specification (Actor-Action-System-Response Table)
- The following is a use case for: **Pay for Regular Customer**
- Each of the seven system responses is an ideal candidate to become a method of a class or a database operation. Example:
  - **prepare()** can be a method of the class **Invoice**
  - **issueTicket()** can be a method of the class **Ticket**

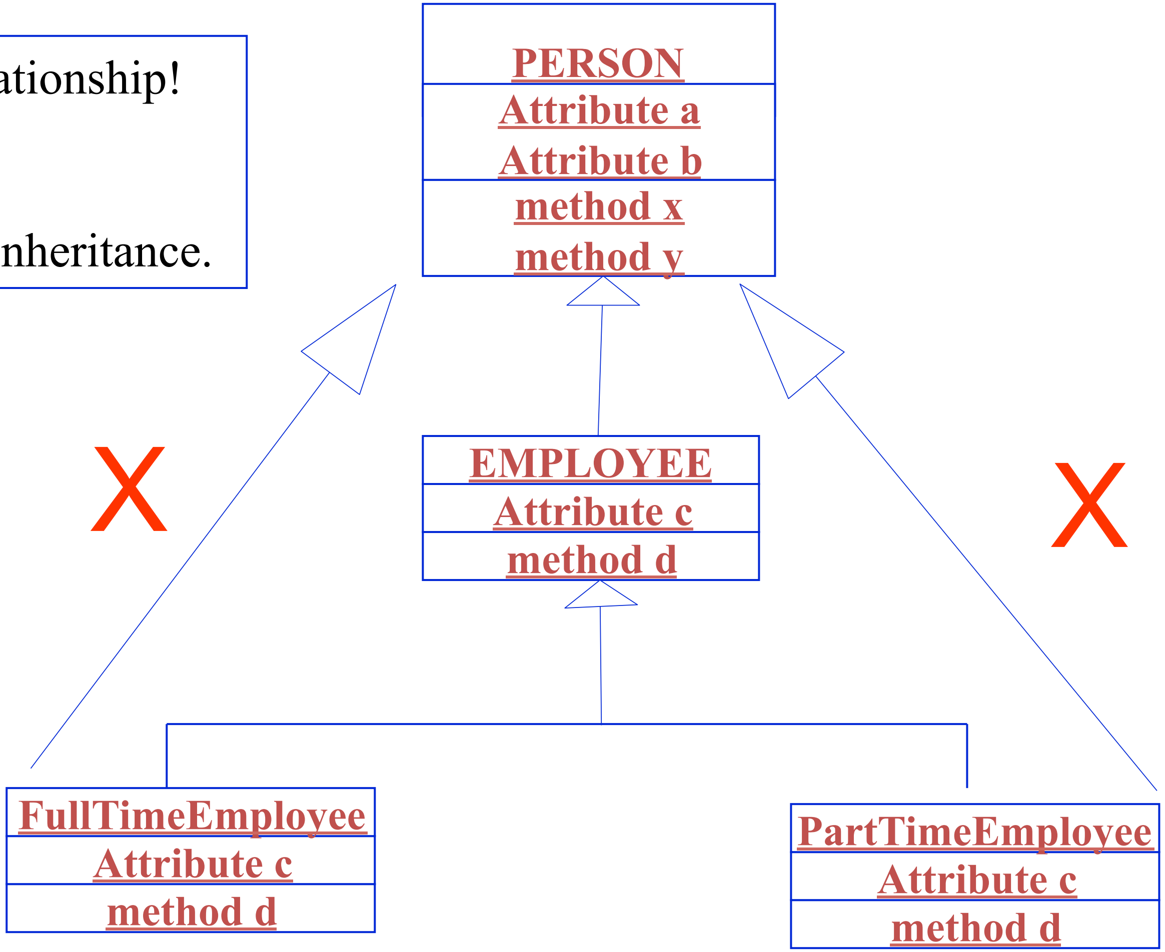
Actor action	System response
1. The customer enters ID and PIN	2. Find the account details
	3. Prepare an invoice
	4. Computes 10% discount
	5. Issue the invoice
	6. Forward the invoice to ACS
	7. Attach the invoice with the account
	8. Prepare and issue ticket.

Multiple Inheritance in UML



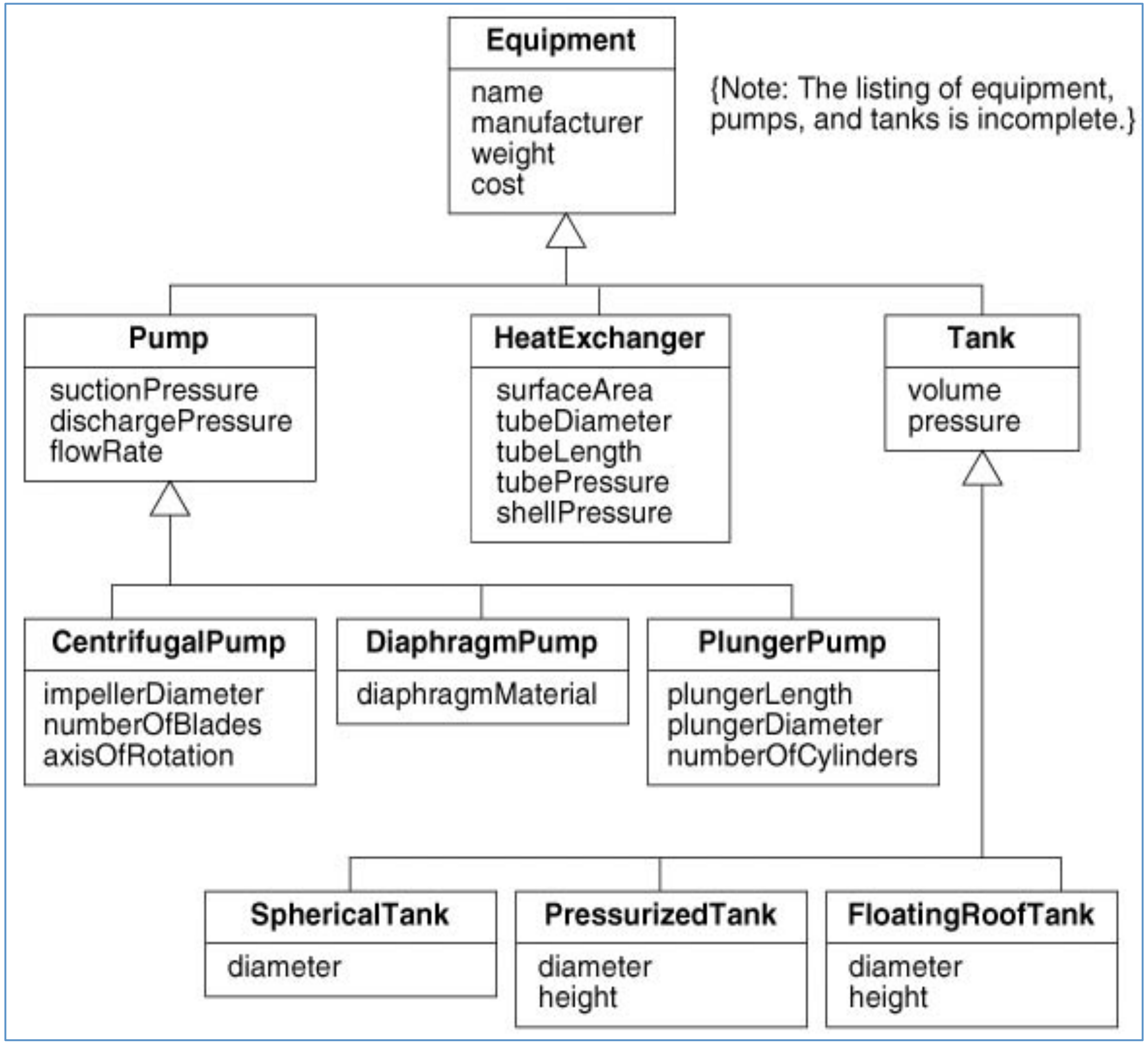
Incorrect Generalisation in UML

- Avoid circular inheritance relationship!
- It is redundant.
- Wrong example for multiple inheritance.



Multilevel Inheritance Hierarchy and Objects

Classes



Equipment

name
manufacturer
weight
cost

{Note: The listing of equipment, pumps, and tanks is incomplete.}

Pump

suctionPressure
dischargePressure
flowRate

HeatExchanger

surfaceArea
tubeDiameter
tubeLength
tubePressure
shellPressure

Tank

volume
pressure

CentrifugalPump

impellerDiameter
numberOfBlades
axisOfRotation

DiaphragmPump

diaphragmMaterial

PlungerPump

plungerLength
plungerDiameter
numberOfCylinders

SphericalTank

diameter

PressurizedTank

diameter
height

FloatingRoofTank

diameter
height

Objects

P101:DiaphragmPump

name = “P101”
manufacturer = “Simplex”
weight = 100 kg
cost = $5000
suctionPres = 1.1 atm
dischargePres = 3.3 atm
flowRate = 300 l/hr
diaphragmMatl = Teflon

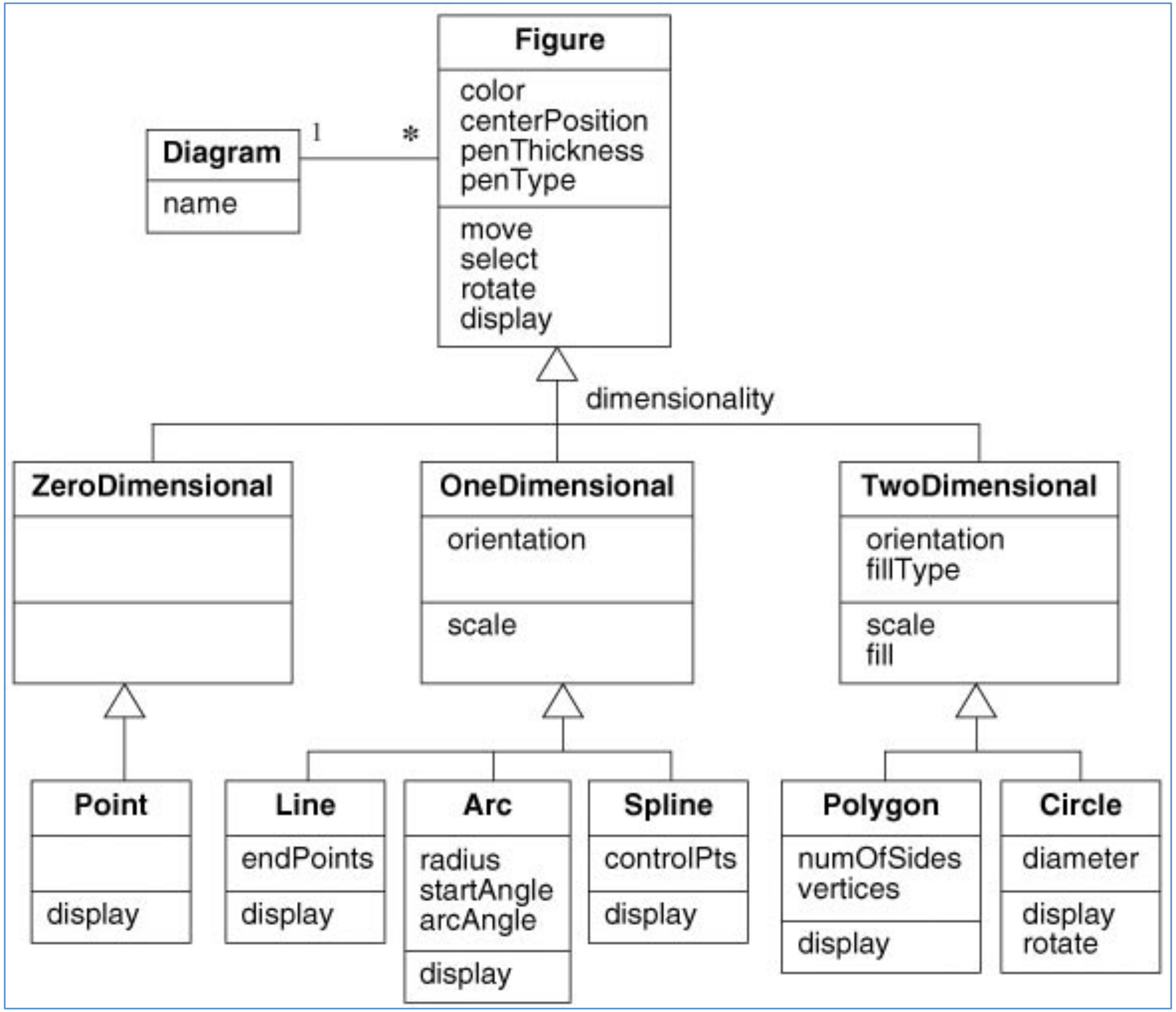
E302:HeatExchanger

name = “E302”
manufacturer = “Brown”
weight = 5000 kg
cost = $20000
surfaceArea = 300 m^2
tubeDiameter = 2 cm
tubeLength = 6 m
tubePressure = 15 atm
shellPressure = 1.7 atm

T111:FloatingRoofTank

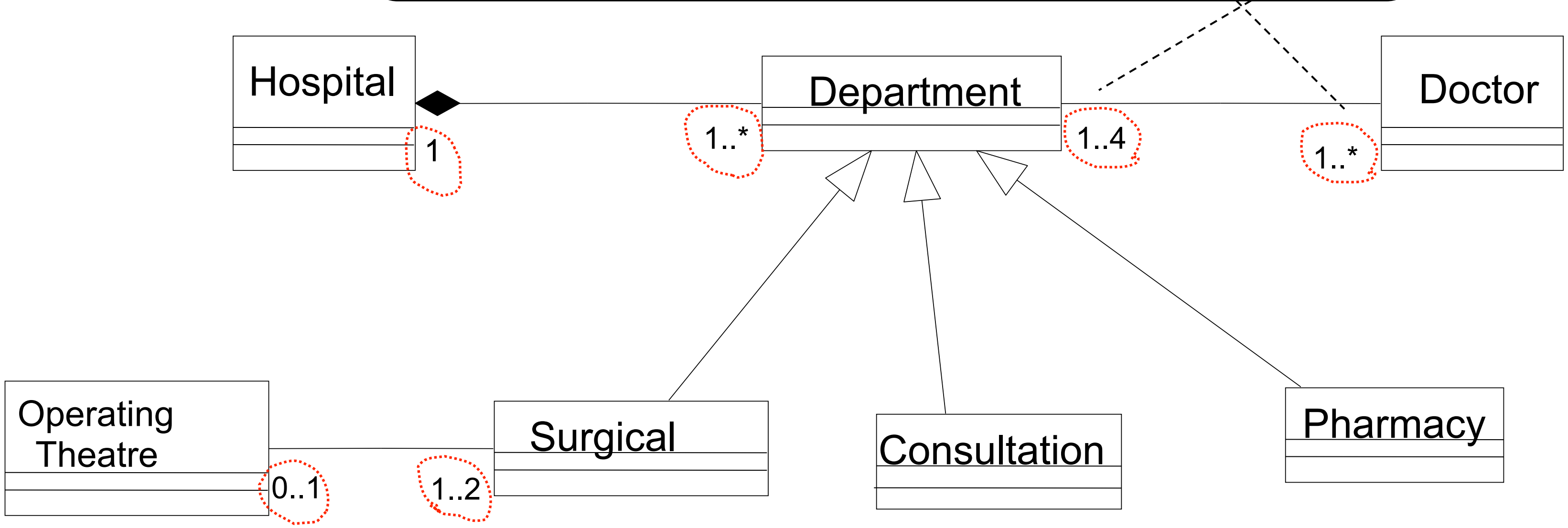
name = “T111”
manufacturer = “Simplex”
weight = 10000 kg
cost = $50000
volume = 400000 liter
pressure = 1.1 atm
diameter = 8 m
height = 9 m

An Example: Inheritance Hierarchy



Multiplicities in UML

Multiplicities:
Show one Object of a Class Relating to how many Objects of the Other Class; And vice versa.

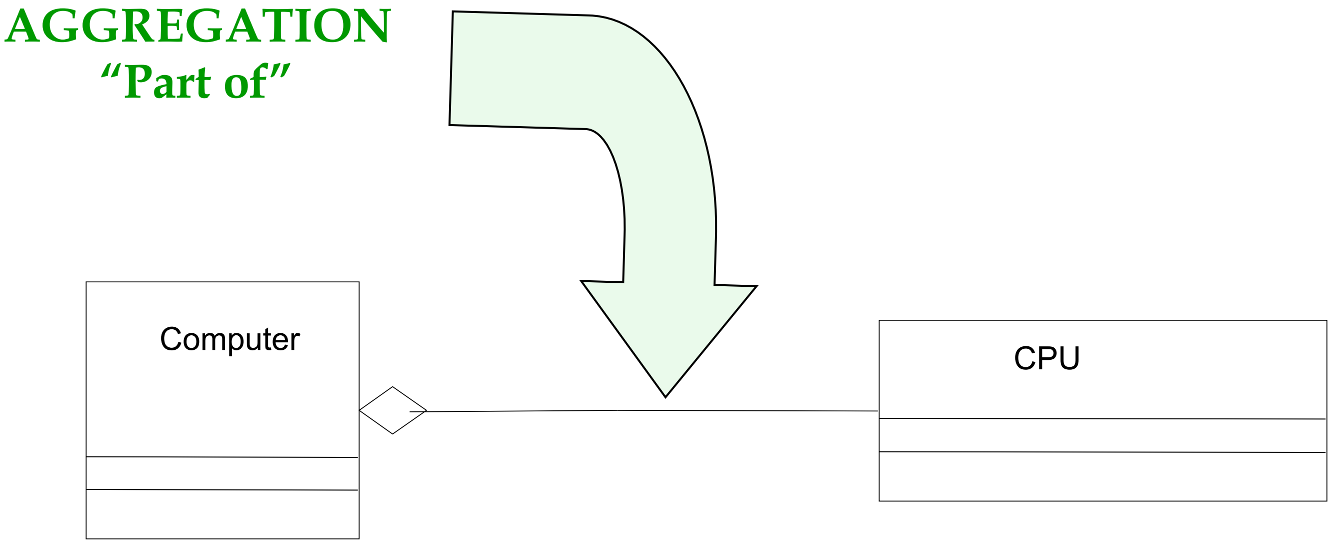


Note: Inheritance has NO multiplicities – Its meaningless, because inherited classes. Still result in a SINGLE object.

Aggregation

- The most significant property of aggregation is transitivity –that is, if A is part of B and B is part of C, then A is part of C.
- Aggregation is also anti-symmetric -that is, if A is part of B then B is not part of A
- Aggregation is a relationship between two classes where the instances of one class are in some way parts, members, or contents of the instances of the other.
- That is, two classes have a “part of” relationship.
  - e.g. a car **has a** steering wheel; a steering wheel is a **part of** a car.

Aggregation Relationship in UML

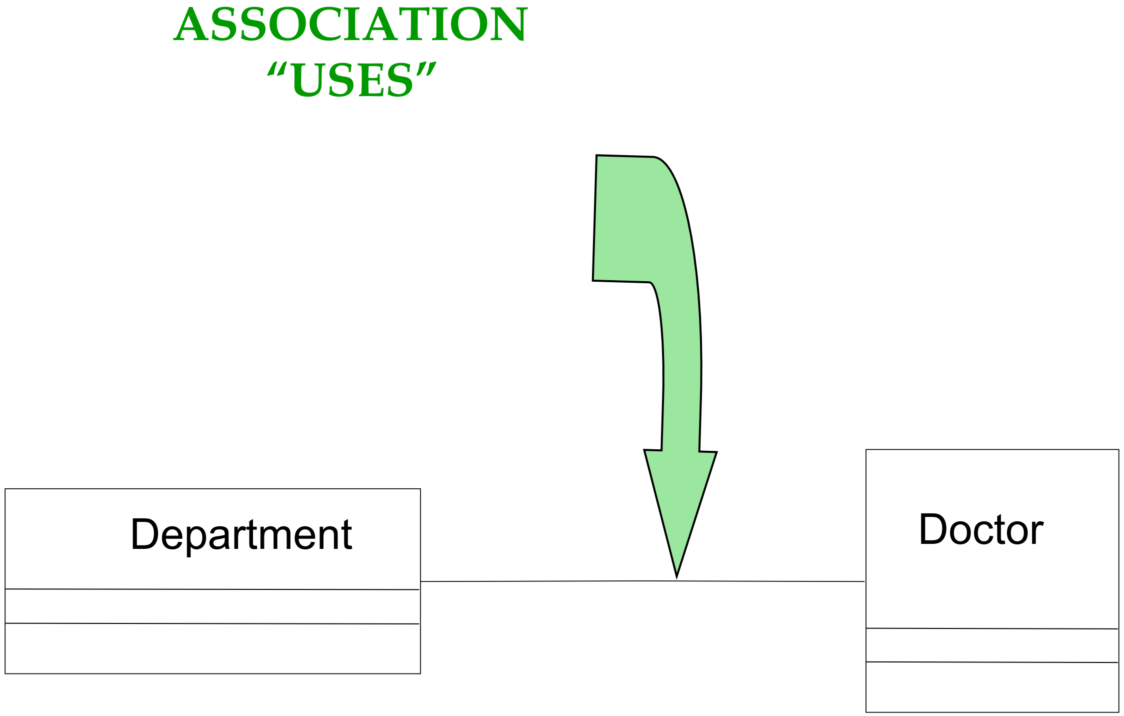


- Hospital *has* Department; Closer/Tighter Relationship; However not always true.
- e.g. Customer *has* Account is not Aggregation as they can be independent of each other.
- A Room is a part of a Building (Composition)
- Hand is a part of body (Aggregation)
- Page is a part of book (Aggregation)

Associations

- An association is a relationship among objects between two classes
- In the real word, objects have relationships, or associations, with other objects.
  - e.g., students TAKE courses, professors GRADE students, lions EAT antelopes, etc.
- Associations often appear as verbs
- How to recognise association:
  - We select pairs of objects from both classes and ask if there is a ‘use’ relationship between them
  - If the answer is yes, then ask the next two questions:
    - How do these objects relate, and
    - What service(s) does each provides to or receives from others?
- Not all “uses” relationship are always association, they might be dependency relationship !

The Association Relationship in UML

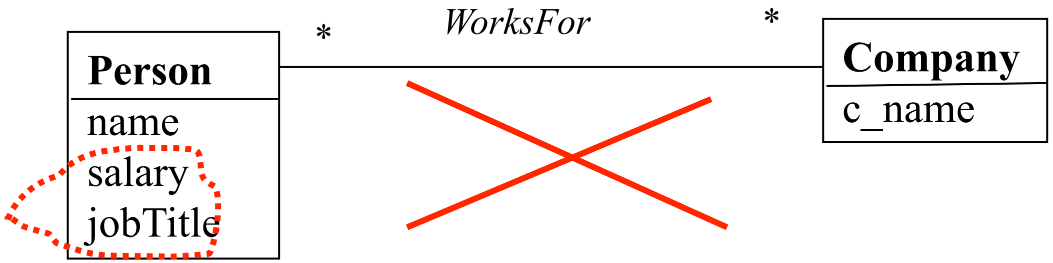


Doctor *uses* Department; Department also uses Doctor;
The Relationship is *loose.*

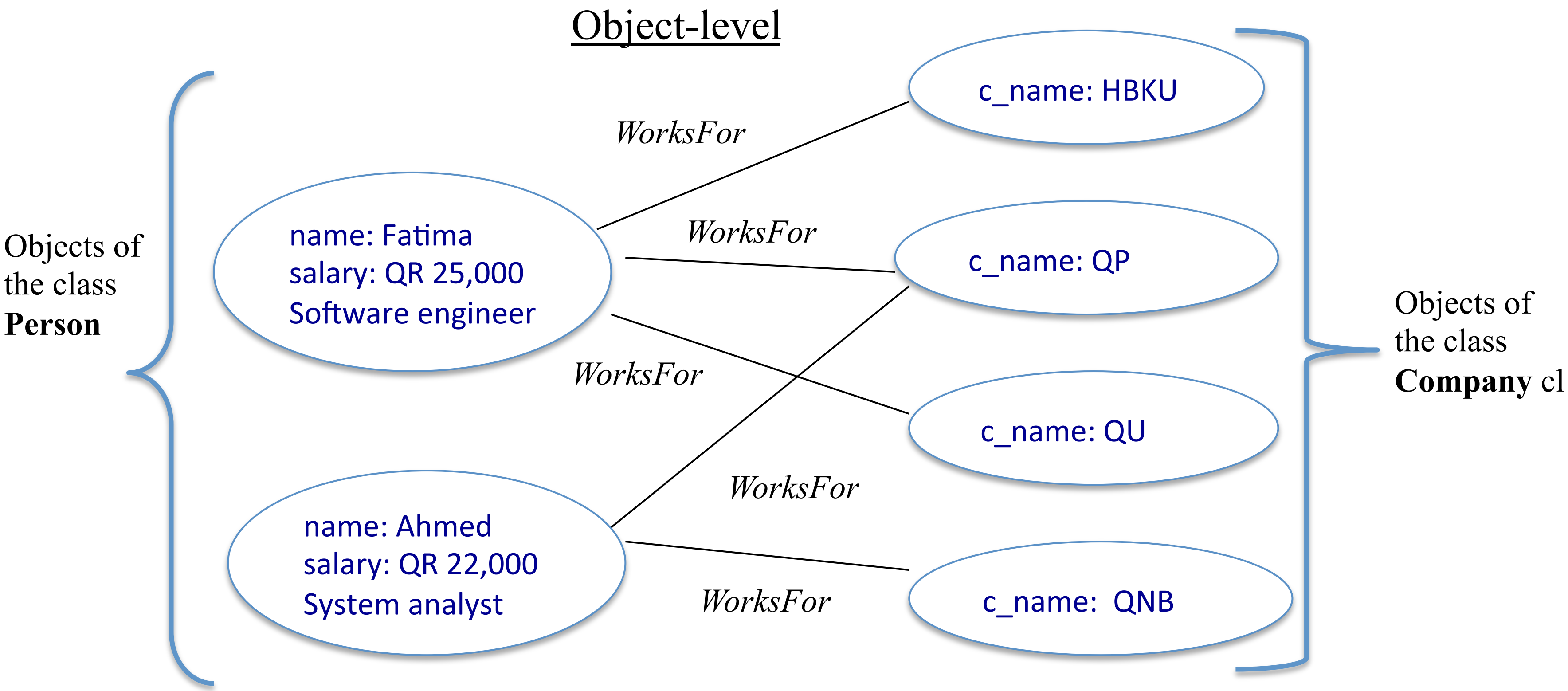
Aggregation versus Association

- An aggregation is a complex object composed of other objects
- Aggregation is special form of association, not an independent concept
- If two objects are tightly bound by a part-whole relationship, it is an aggregation
- If the two objects are usually considered as independent, even though they may often be linked, it is an association
- An association is used when one object wants another object to perform a service for it.
- Associations are typically an interaction described by a verb.

Occasional Semantic Problems of Association

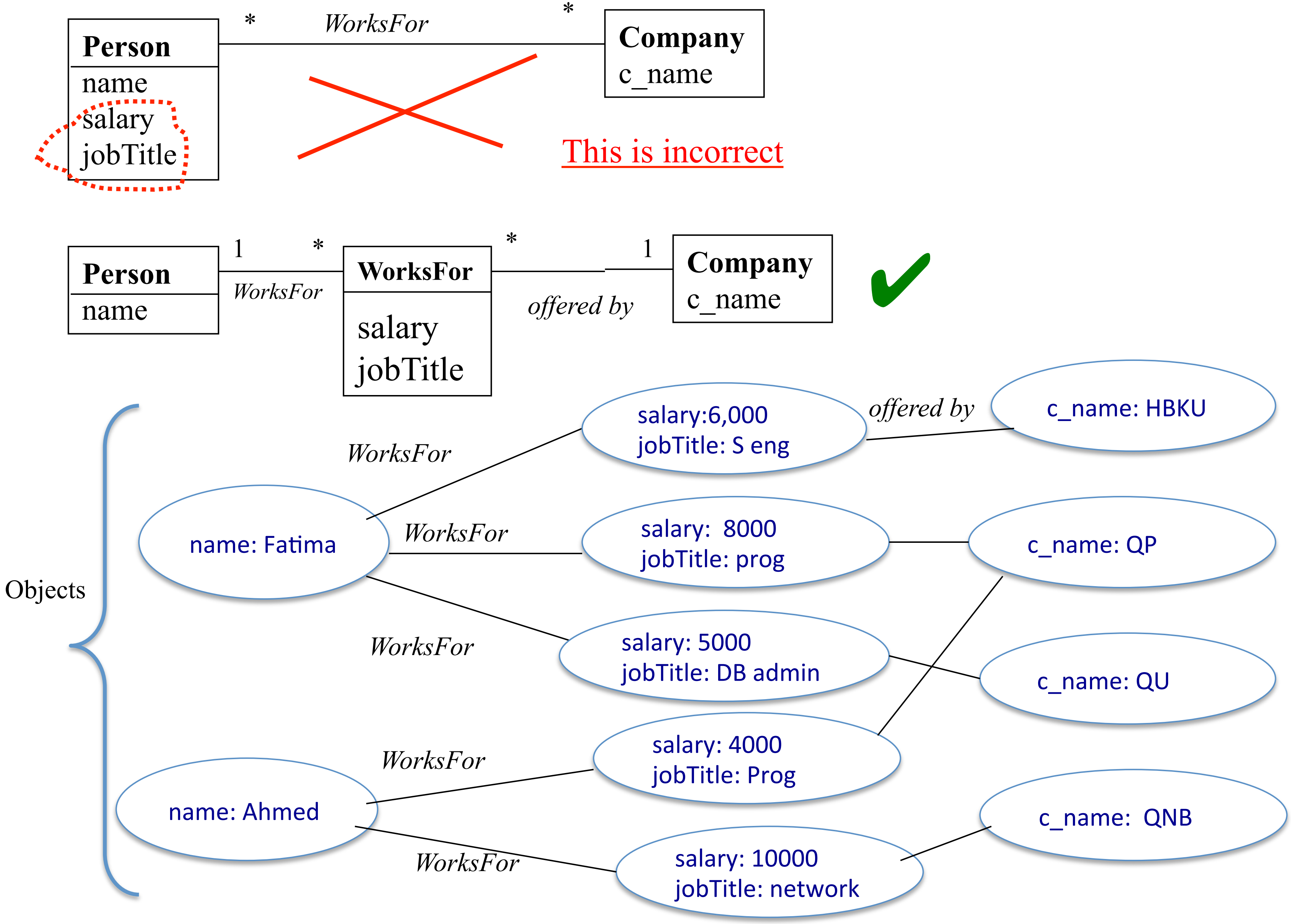


If a person works for many companies with different positions, and get different salary, this class diagram cannot handle this

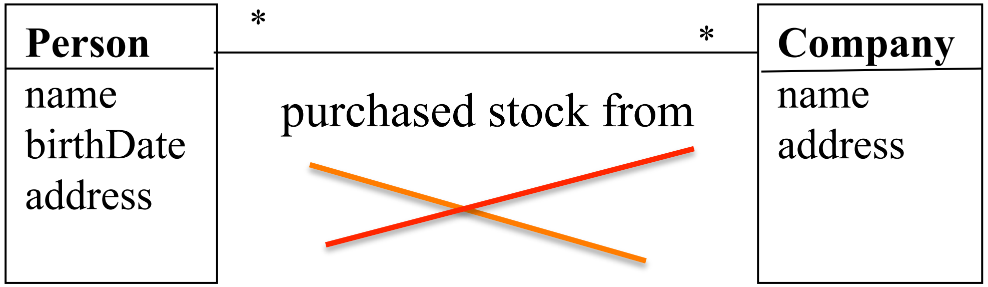


- Fatima has three jobs, three salaries from these three jobs, and she has three positions, but this class diagram is incorrect because it captures only one salary and one position

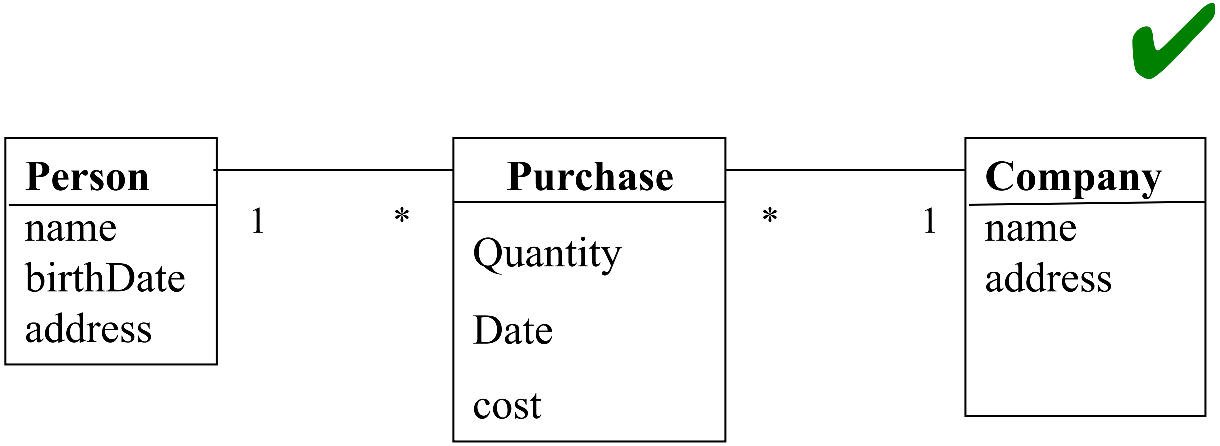
Solution to the Occasional Semantic Problem



Another Example



This is incorrect



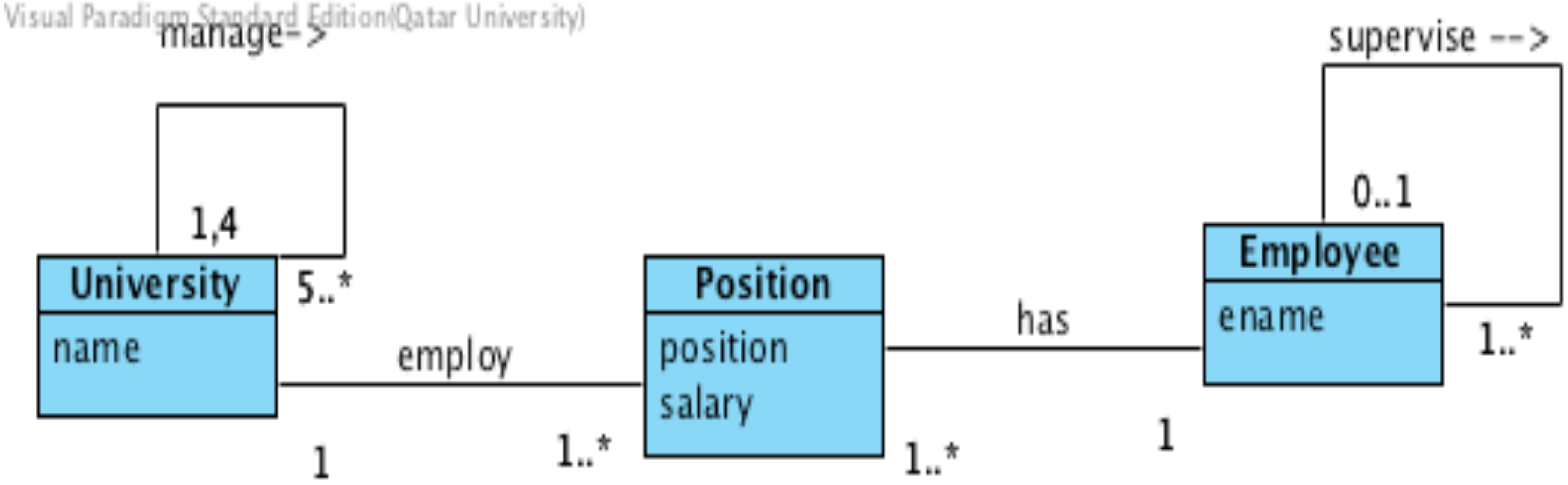
***Normal class:* There may be any number of occurences of a Purchase for each Person and Company. Each purchase is distinct and has its own quantity, date, and cost.**

Example-1

Requirements:

An university manages more than 4 other universities in a city, but one university is managed by either 1 or 4 universities. A person works for one or more universities with different positions and salaries. A university can employ any number of persons. A person supervises at least one person, but a person may be supervised by only one other person, but not all persons are supervised.

Solution:

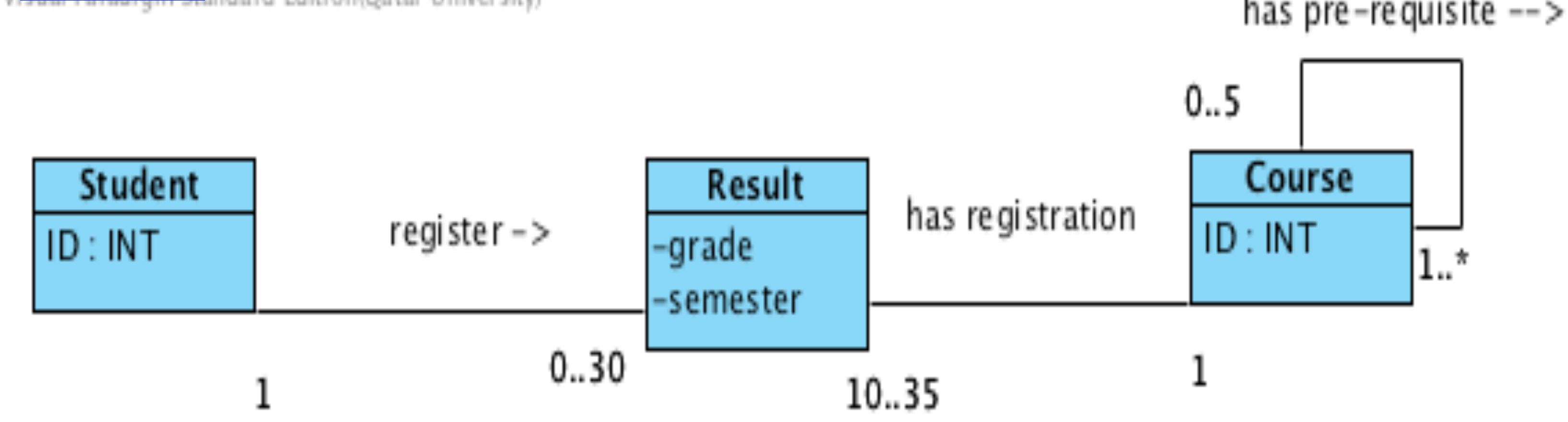


Example-2

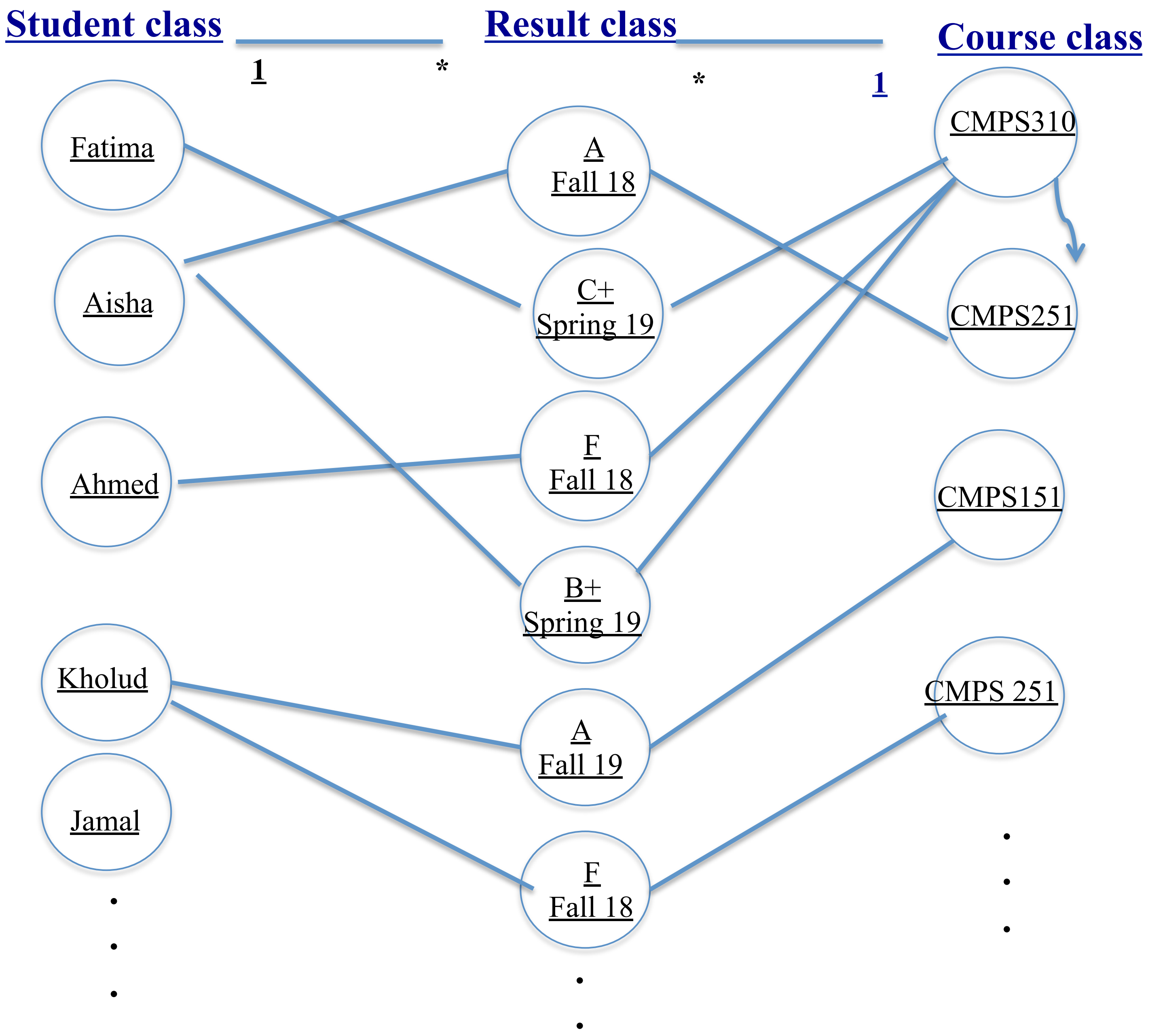
Requirements

- Some courses have one or more pre-requisite courses. Not all courses are pre-requisite of other courses. A course can be the pre-requisite of maximum of 5 courses. Some courses are not pre-requisite of other courses. A student can register for maximum 30 courses, one course can have maximum 35 and minimum 10 students. The course has ID and the student has an ID, and grade of each course with semester information is stored for each student.

Solution:



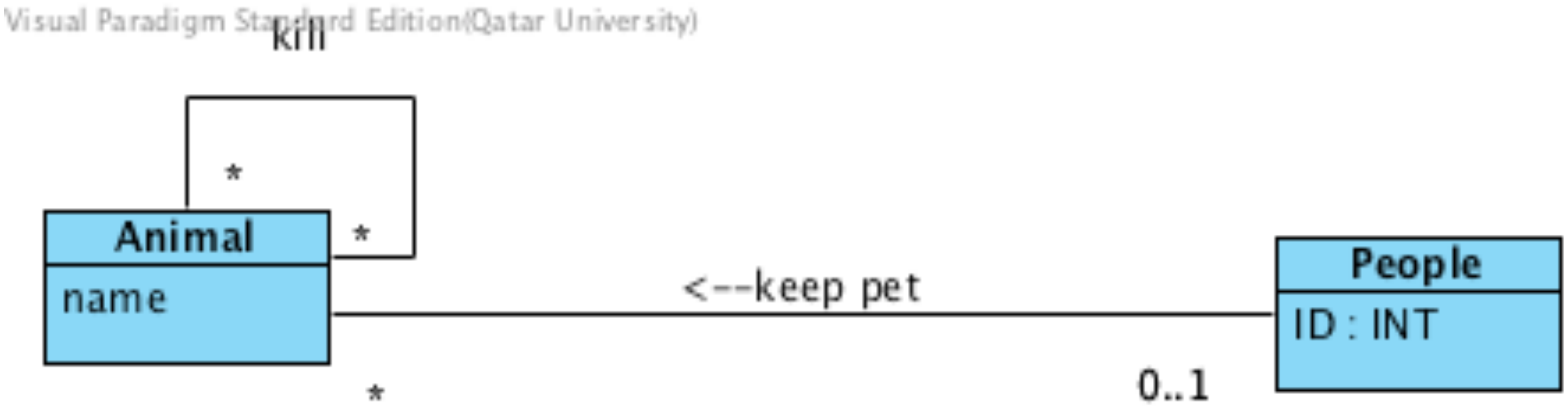
Object Level Association Representing Example 2



Example-3

- Requirements:
  - Some animals kill other animals(s), but not all animals are killed by other animals. Some animals are pet of some persons, but one animal can be pet of only one person. Some persons may keep any number of animal as their pets.

Solution:



Messages

- Messages are the requests that an object sends to another object to make it do something
- An object responds to messages that are passed to it from other objects
- An object oriented system consists of objects, communicating with one another through the passing messages
- When the object receives the message it “wakes up” and executes its method having the same name as the message that it received
- When the execution is complete the object will pass the result back to the object that sent the message
- The operation performs the appropriate method and optionally, returns a response.
- Objects request other objects to perform an activity via **messages**.
- Thus, a message is the way in which a sender object requests another object to run one of its methods.
  - In most OO programming languages, the syntax for a message is:

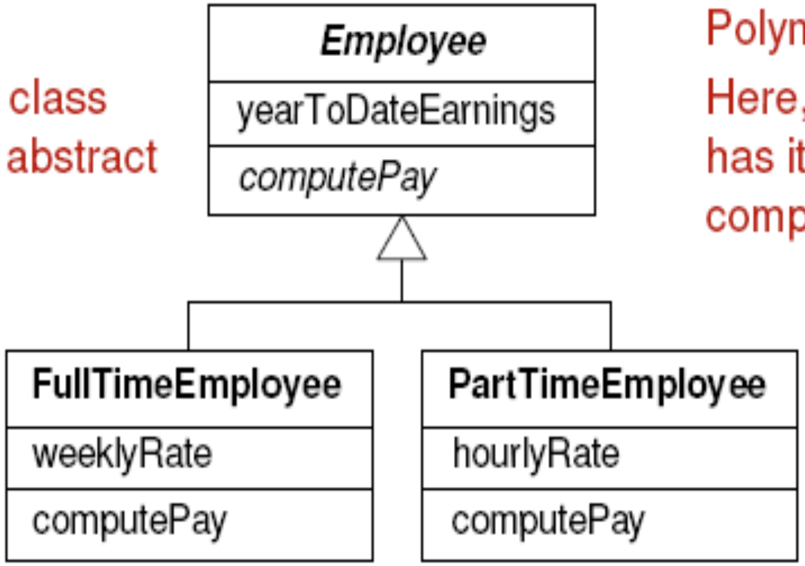
```
OBJECTNAME.METHODNAME( ARGUMENT1, ARGUMENT2, ...)
```

Abstract Classes

- An abstract class has one or more abstract/pure virtual methods.
- An abstract class cannot have instances/objects.
- An abstract class cannot be used to instantiate objects.
- An abstract class can contain concrete subclass(s).

Abstraction:

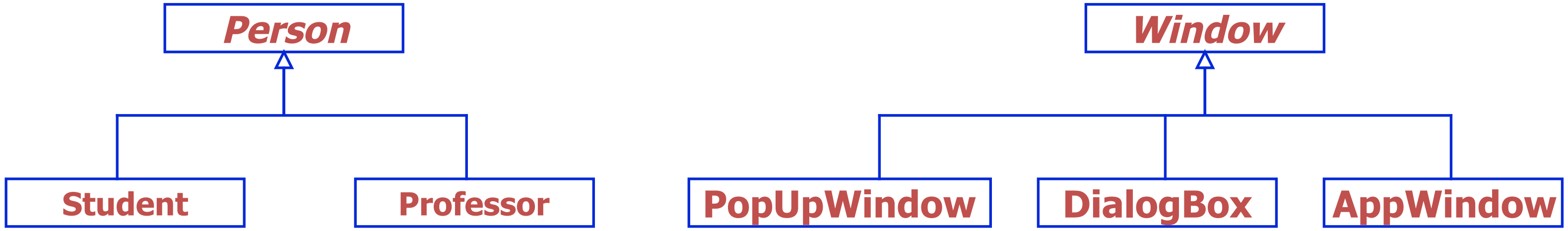
Employee is an abstract class and *computePay()* is an abstract operation (italicized)



Polymorphism:

Here, each type of Employee has its own version of computePay()

Abstract and Concrete classes



- An **abstract class** is one that doesn't have objects instantiated from it, meaning that here are no direct instances of it, but the behaviour it defines belongs to all instances of its subclasses

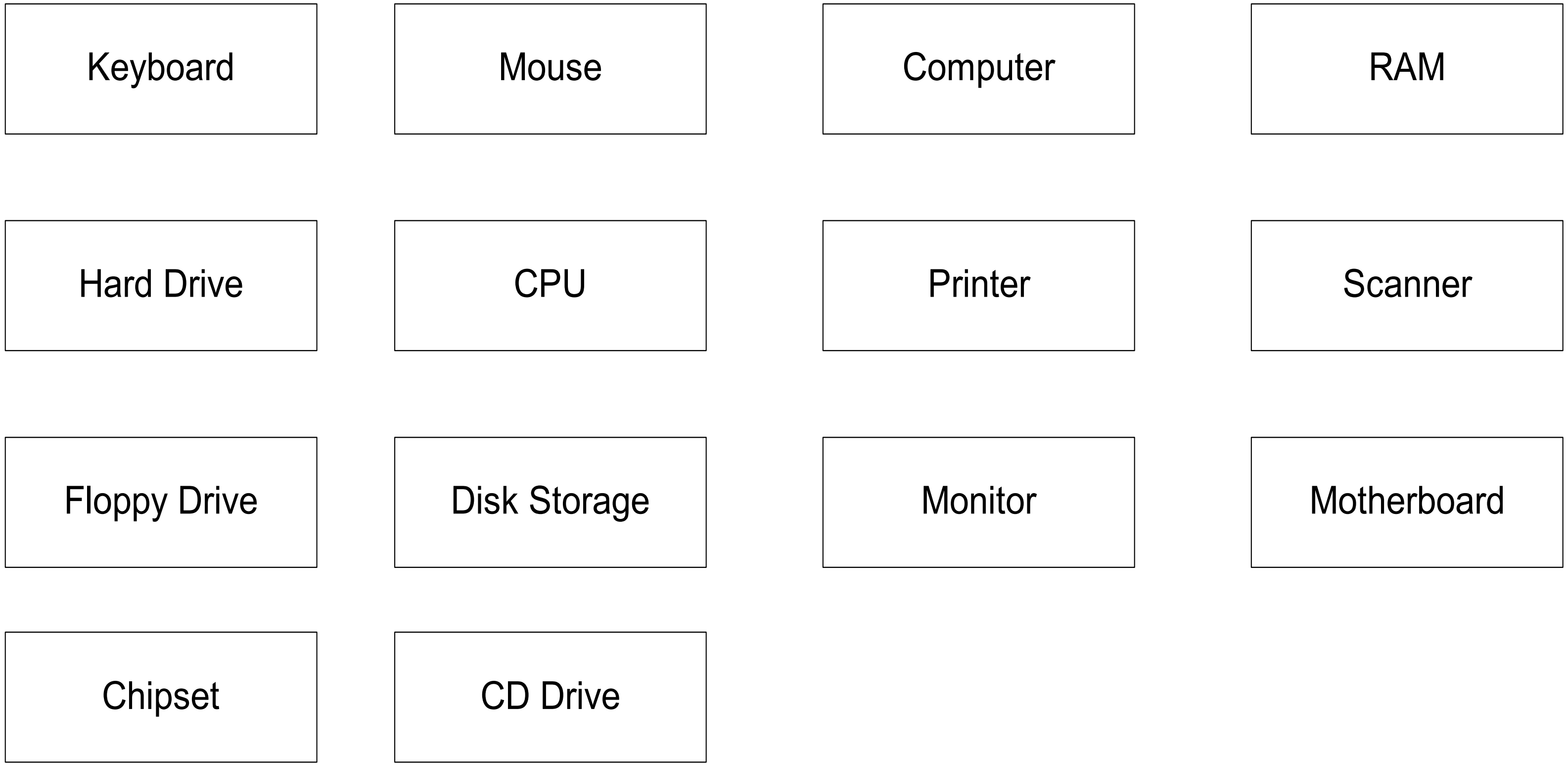
- The creation of abstract super classes also improves the extensibility of a software product

- In diagrams, an abstract class is indicated via italics

- A **concrete class** is one that does have objects instantiated from it.
- Abstract classes are helpful way to encapsulate similarities between classes (i.e., to create inheritance).

Relationship Exercise

Describe the relationships (**association, aggregation, inheritance**) among the following **classes** using UML notation.



Keyboard

Mouse

Computer

RAM

Hard Drive

CPU

Printer

Scanner

Floppy Drive

Disk Storage

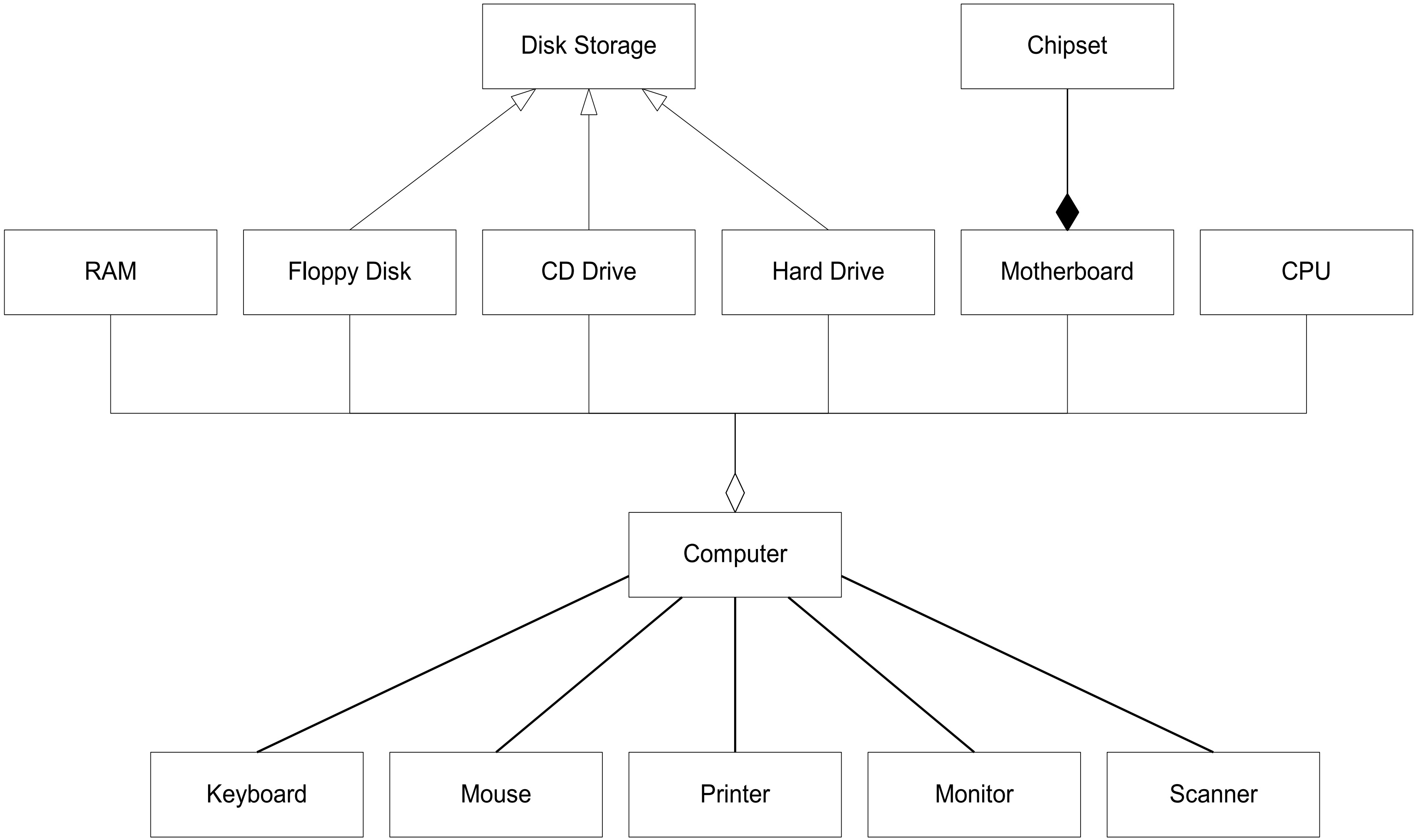
Monitor

Motherboard

Chipset

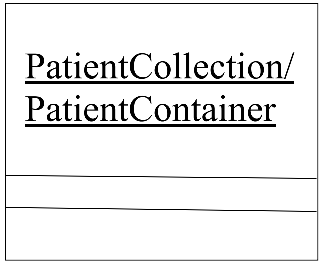
CD Drive

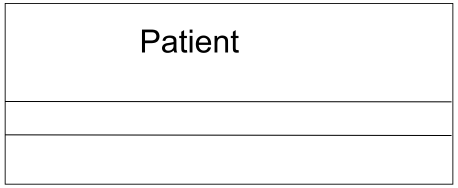
Association Solution



A Collection or Container Class

- In practice, having a class Customer, storing details of a Customer, is not enough;
- A Collection/Container class, that will enable storage, retrieval, sorting and manipulation of a group of customers, is usually required.
- A Collection/Container Class is actually a database, stores multiple objects of the same type or class.
- In a class diagram we usually do not show the Collection/Container class.
- It is assumed that every class has at least one container/collection class without showing this explicitly in the class diagram.





- A Collection or Container Class for Patient objects
- Not usually explicitly shown in the design class diagram
- By default, every entity class has a corresponding Collection/Container class which is usually not shown in the Class diagram

Typical Class Stereo Types

- **<<boundary>> Classes (Views + Interfaces of External Systems)**
  - Interface between the internal objects of the system and the outside world, could be either:
    - User interface for human users

    => Ignore the UI as they are often designed separately. UI is a simply the visual representation of the model

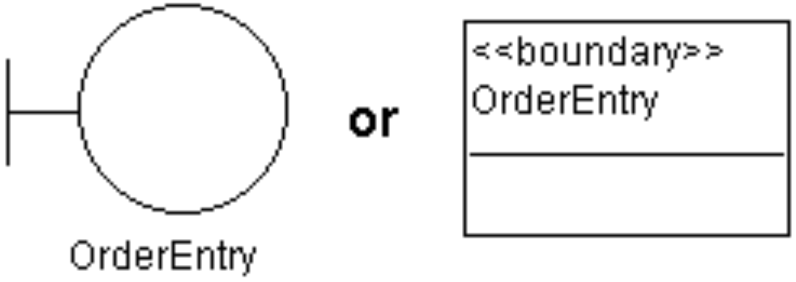
    - Intermediate communication with other systems
- **<<entity>> Classes (Model)**
  - Represent data that have to be stored and managed by the system
  - Implement the application logic
- **<<controller>> Classes (Control class)**
  - Coordinate the flow of events of a use case

- **NOTE**: Controller class can be used as a boundary class, that is, instead of a boundary class, a controller class can act as an interface between the use case and the actors.

Boundary Class

- A Boundary Class is a stereotype of a class that is specified in the UML
- Objects that interface with system actors (e.g. a user or external service). Windows, screens and menus are examples of boundaries that interface with users.
- A "Boundary Class" is a class that lies on the periphery of a system, but within it.
- It interacts with actors outside the system as well as objects of all three kinds of analysis classes within system
- It can be shown as a regular class rectangle with stereotype of "boundary", or as the following special icon:

Symbol of boundary class in sequence diagram (We will see later)

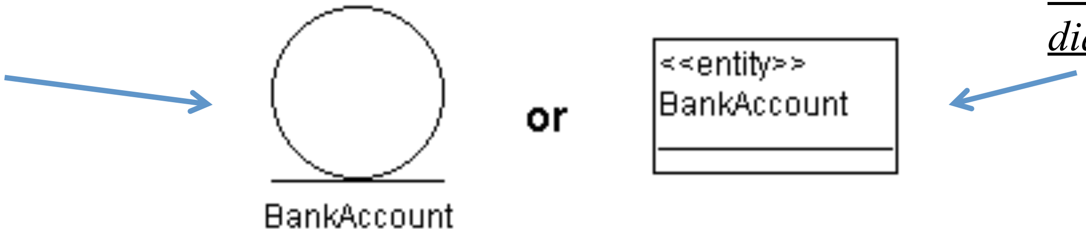


Symbol of boundary class in class diagram

Entity Class

- An <<entity>> Class is a stereotype of a class that is specified in the UML for Business Modeling.
- An "Entity Class" is a class that is passive -- it does not initiate interactions on its own.
- Objects representing system data
- An entity object may participate in many different Use Case realizations and usually outlives any single interaction.
- It can be shown as a regular class rectangle with stereotype of "entity", or as the following special icon :

Symbol of entity class in sequence diagram (We will see later)



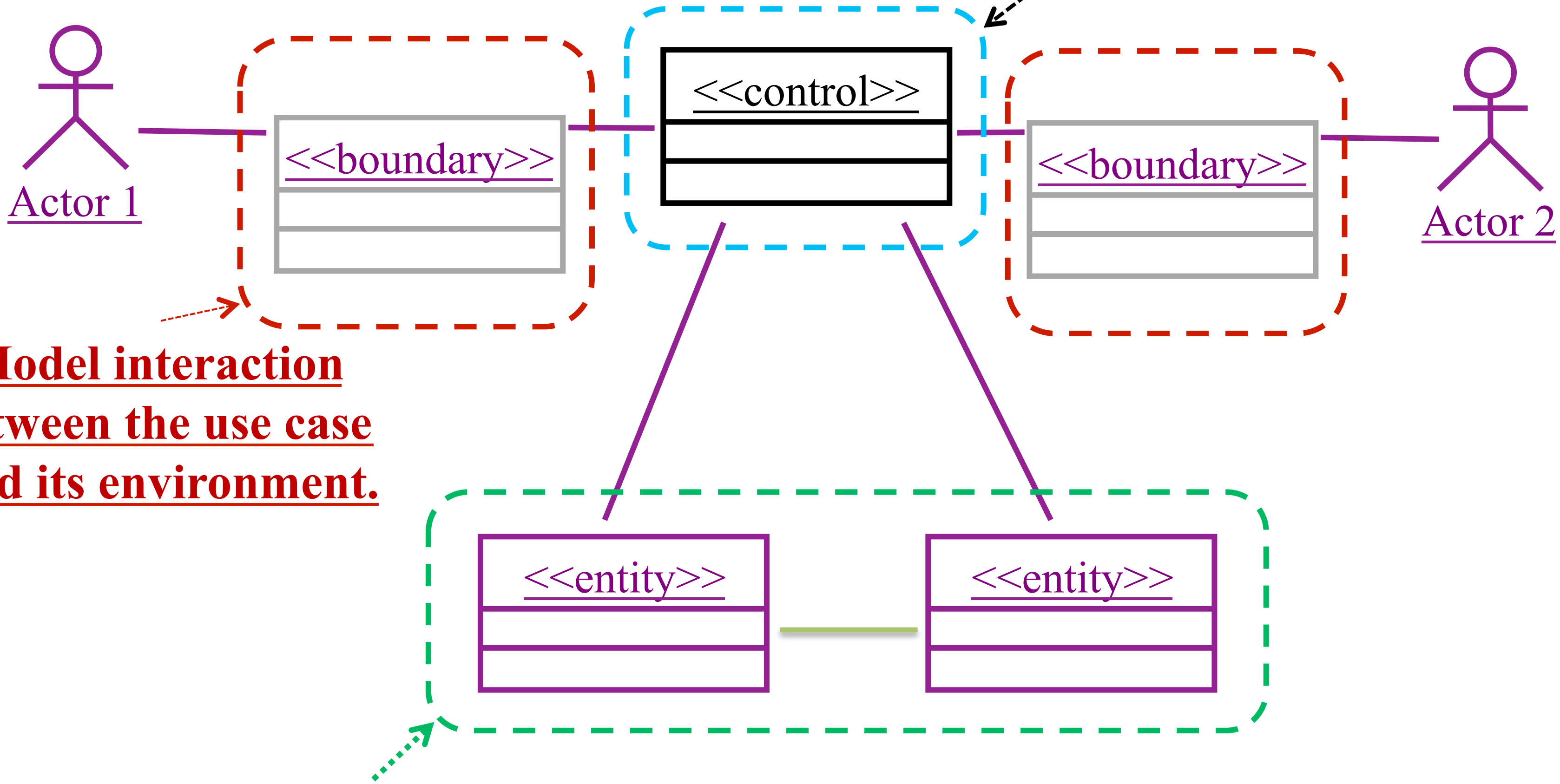
Symbol of entity class in class diagram

Controller Class

- Objects that mediate between boundaries and entities. These serve as the glue between boundary elements and entity elements.
- A "Control Class" is a class that contains an object which denotes an entity that controls interactions between a collection of objects.
- A <<control>> class usually has behavior specific for one Use case
- A Control Class is a stereotype of a class that is specified in the UML
- Objects that mediate between boundaries and entities. These serve as the glue between boundary elements and entity elements

Relationship Between Class Stereotypes

Controller class coordinates the use-case behavior, can also act as a boundary class.

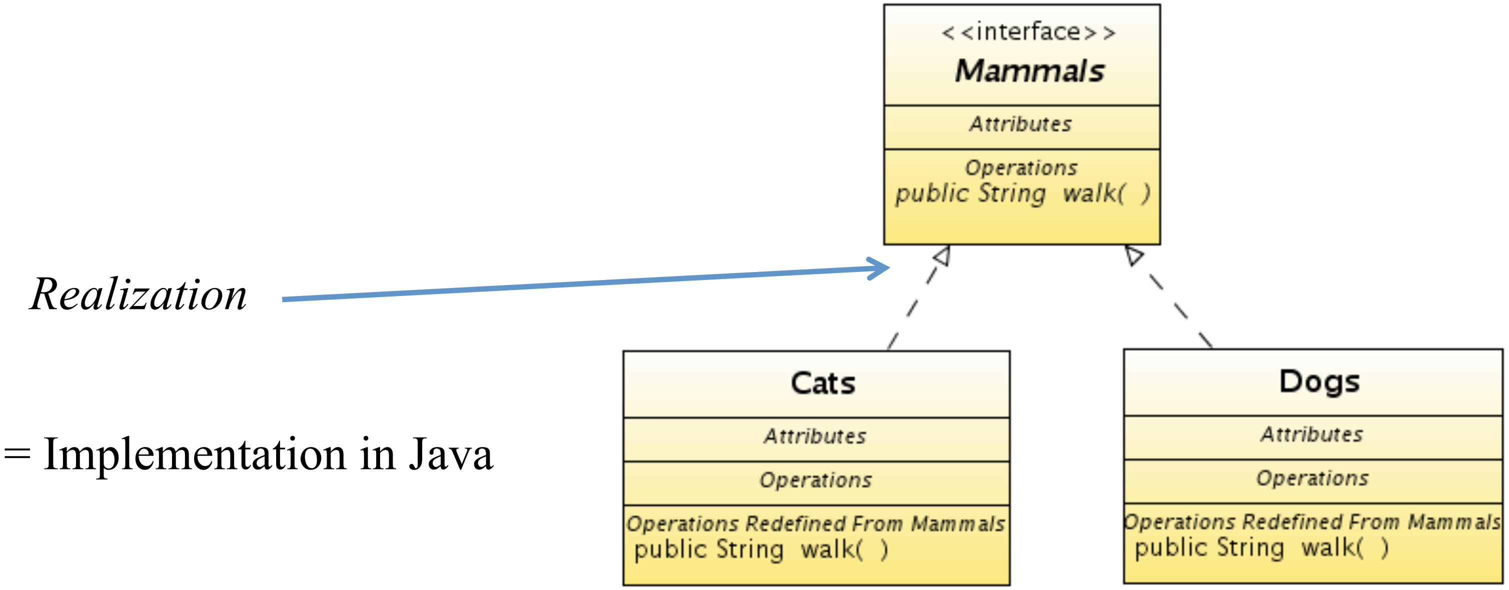


Model interaction between the use case and its environment.

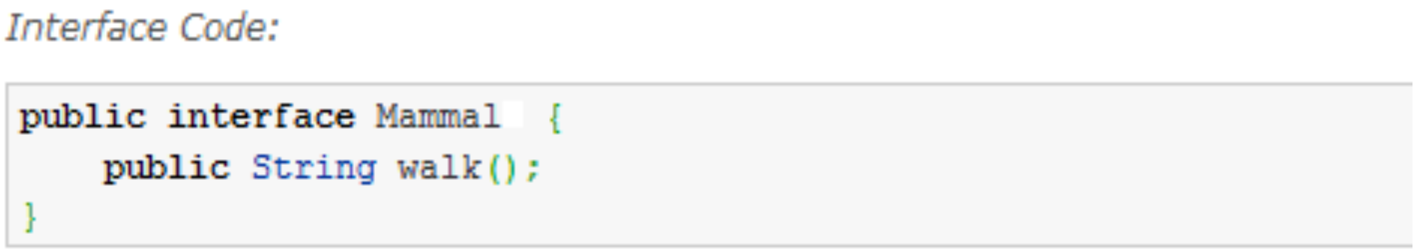
Deals Data - Store and manage information in the system.

Interfaces

- An interface describes a *portion of the visible behaviour* of a set of objects.
  - All methods in the interface are public.
  - Interface cannot be used to instantiate objects.
  - There is no data members in an interface class.
  - UML: use <<interface>> to prefix the class name.
  - An *interface* is similar to a class, except it lacks instance variables and implemented methods

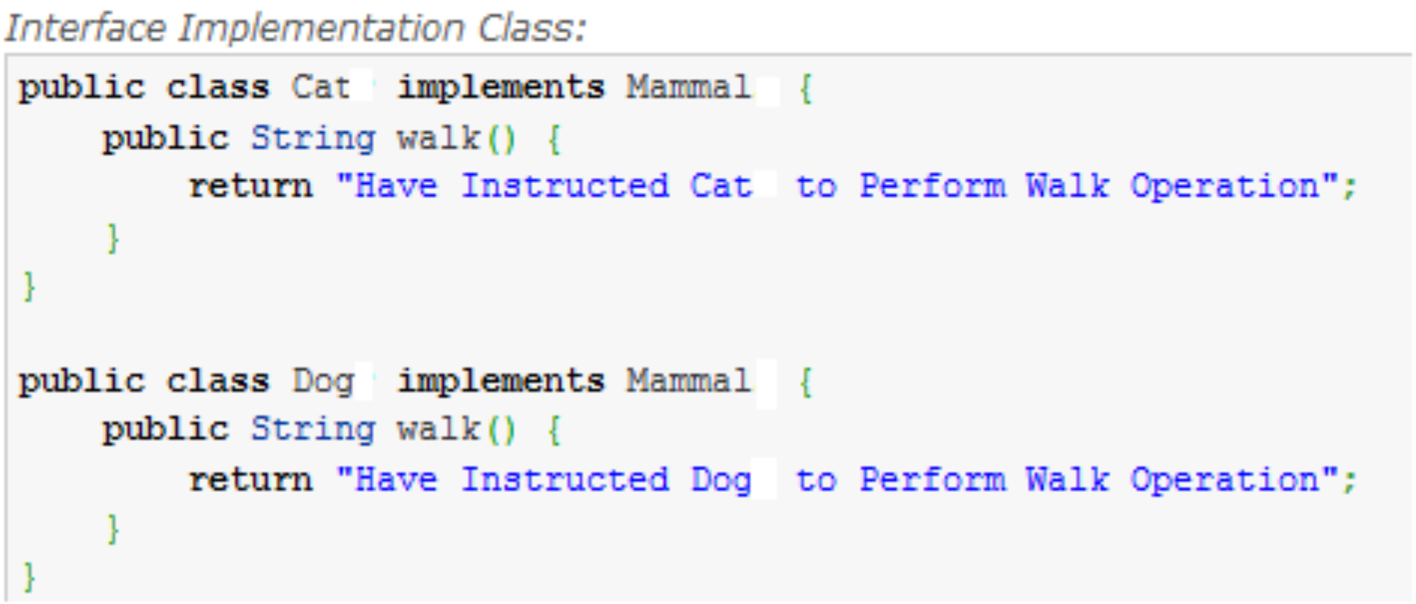


Realization = Implementation in Java



Interface Code:

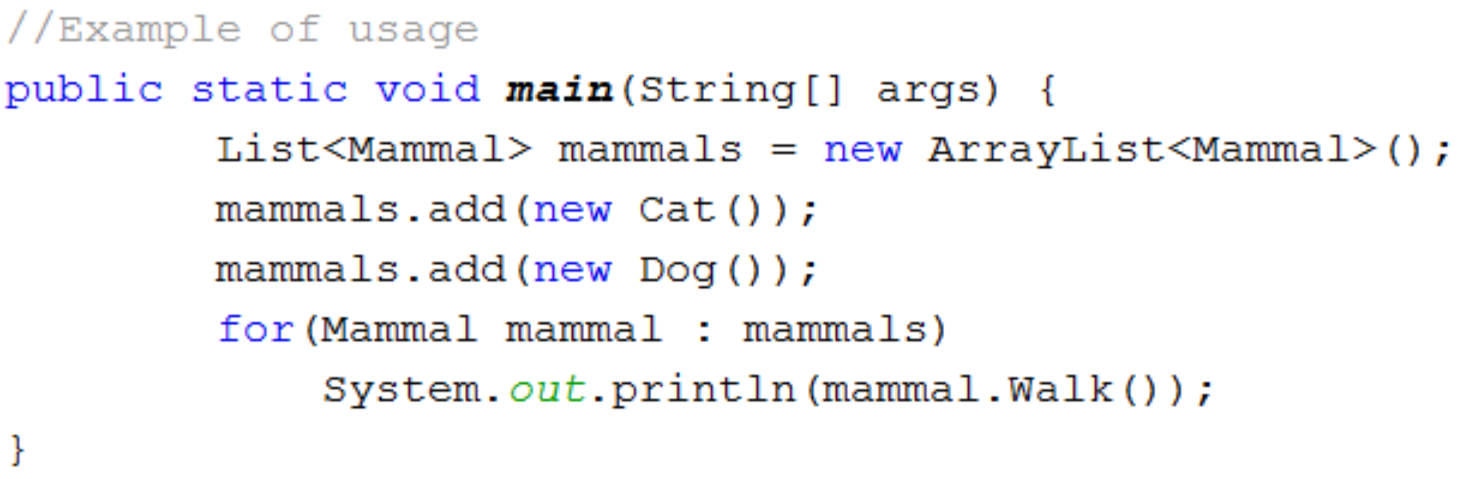
```
public interface Mammal  {
    public String walk();
}
```



Interface Implementation Class:

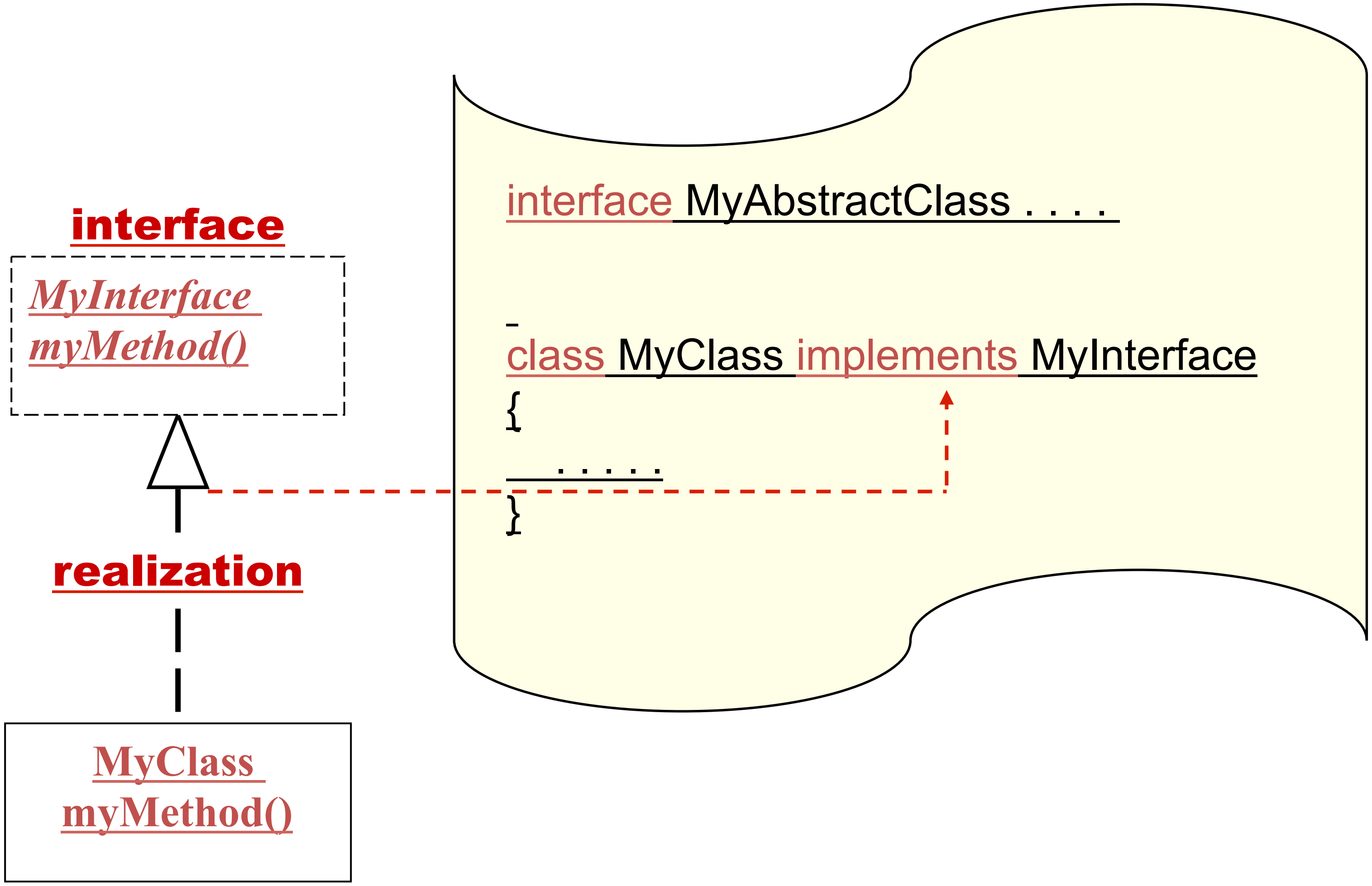
```
public class Cat  implements Mammal  {
    public String walk() {
        return "Have Instructed Cat  to Perform Walk Operation";
    }
}

public class Dog  implements Mammal  {
    public String walk() {
        return "Have Instructed Dog  to Perform Walk Operation";
    }
}
```

Java code for the example shown in the previous slide.



```
//Example of usage
public static void main(String[] args) {
        List<Mammal> mammals = new ArrayList<Mammal>();
        mammals.add(new Cat());
        mammals.add(new Dog());
        for(Mammal mammal : mammals)
            System.out.println(mammal.Walk());
}
```

Interfaces
UML Notation …… Typical Java Implementation



interface

MyInterface
myMethod()

realization

MyClass
myMethod()

```
interface MyAbstractClass . . . .

_
class MyClass implements MyInterface
{
    . . . . .
}
```

Conclusion

- Class relationships
  - Generalization
  - Association
  - Aggregation
- Association class
- Multiplicity
- Messages – objects communicate with one another through the passing messages
- Abstract class -does not have any direct instances
- Class stereotypes –most classes do not need a stereotype.

References

- Booch, G.: Object Oriented Analysis and Design with Applications, Addison-Wesley, 1993, 2nd Edition (chapters 3, 4)
- Blaha, M. and Rumbaugh, J.: Object-Oriented Modelling and Design with UML. Pearson Prentice-Hall, 2005. ISBN: 0-13-196859-9. (chapters 3 ,4)