SWINBURNE UNIVERSITY OF TECHNOLOGY

OOP

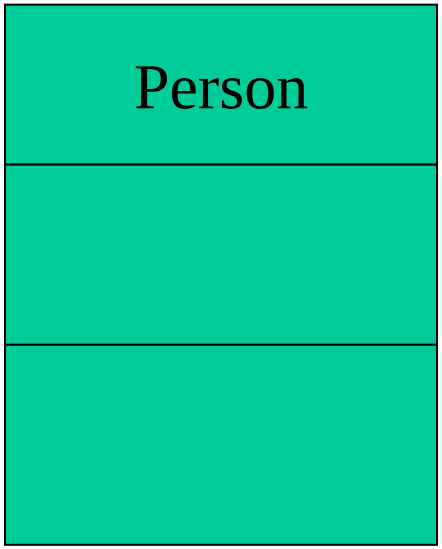
Class diagram and sequence diagram

Class diagram

- UML: Unified Modelling Language
- In software engineering, UML was set up as a standardized model to describe an object-oriented based system. Since classes are the building blocks of a system, class diagrams are the building blocks of UML.
- A class diagram is a visual representation that describes the structure of a system by showing the system's class names, their attributes, methods, and the relationships between objects.

Class names

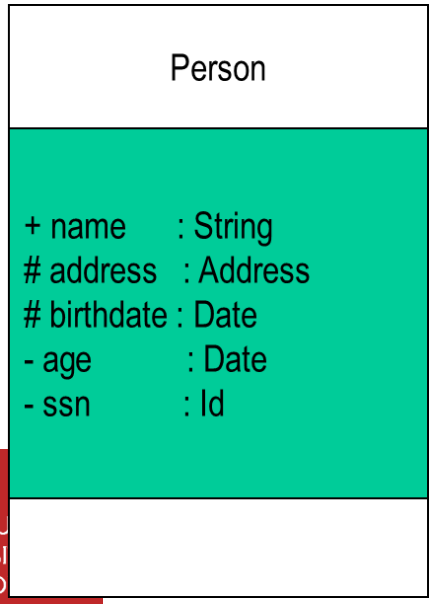
- A class is drawn as a rectangle, usually including its name, attributes, and methods in separate three compartments.
- Class name always appears in the top-most compartment.



Attributes in a Class

- Class or instance variables
- They are typed in middle compartment
- Attributes are usually listed in the form:

  visibility variableName: Type



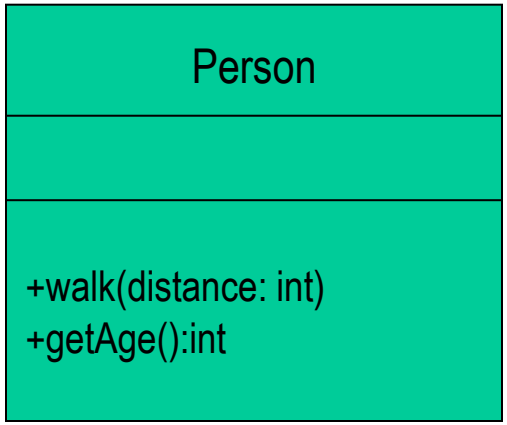
Attributes can be:

+ public
# protected
- private

Class methods

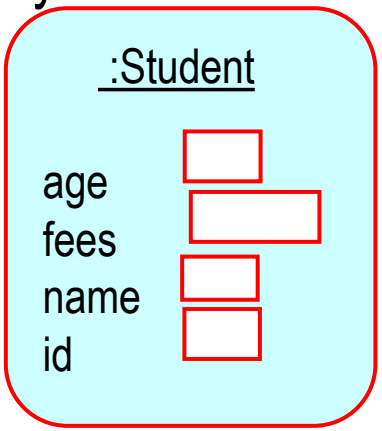
- Methods describe the class behaviour and appear in the bottom compartment.
- Methods in class diagram each include visibility, method name, parameters, and return type in the form below:

visibility name (param1:type=initial_value…):return-type



So Far

- To design a student class then we might choose variables like
  - int to represent their age
  - double to represent their fees
  - String to represent their name
  - int to represent their ID
- We would say the Student has attributes of age, fees and name

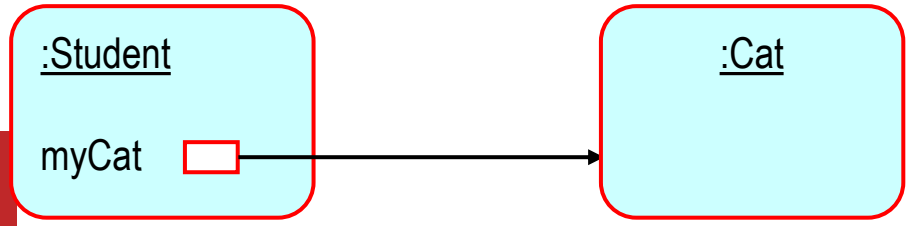


But, the student could have other knowledge …

Collaborating classes

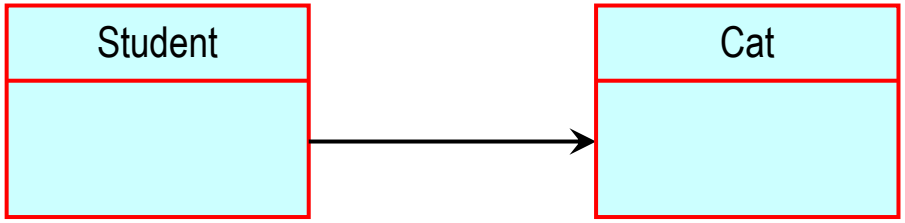
… other knowledge involving other objects

- For example, all students may have a cat…
- A student object can (potentially) access a cat object
- In this case we would declare an instance variable of type Cat in the student class
  - private Cat myCat;
- So AFTER we have given a value to this instance variable:



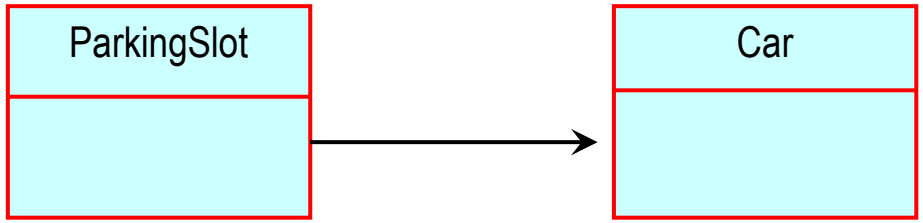
Collaborating classes: Association

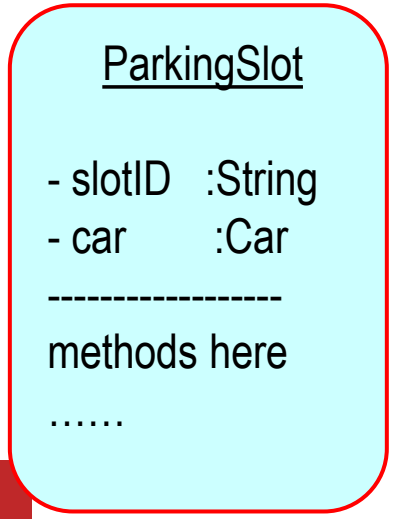
- We say there is an association between the two classes
- If Student calls methods in Cat then we say Cat is a collaborator of Student
  - "Cat helps Student"
  - or "Cat provides services for Student"
  - or "Cat is a server for Student"

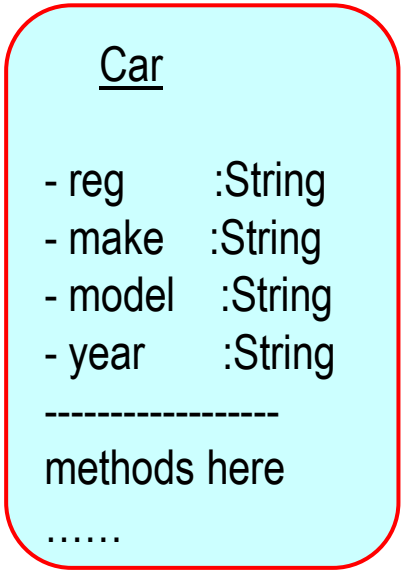


Collaborating classes: Association

- ParkingSlot







Association Relationship

- Directed Association


```
class A
{
        private B b;
        ……
        ……
        ……
}
```

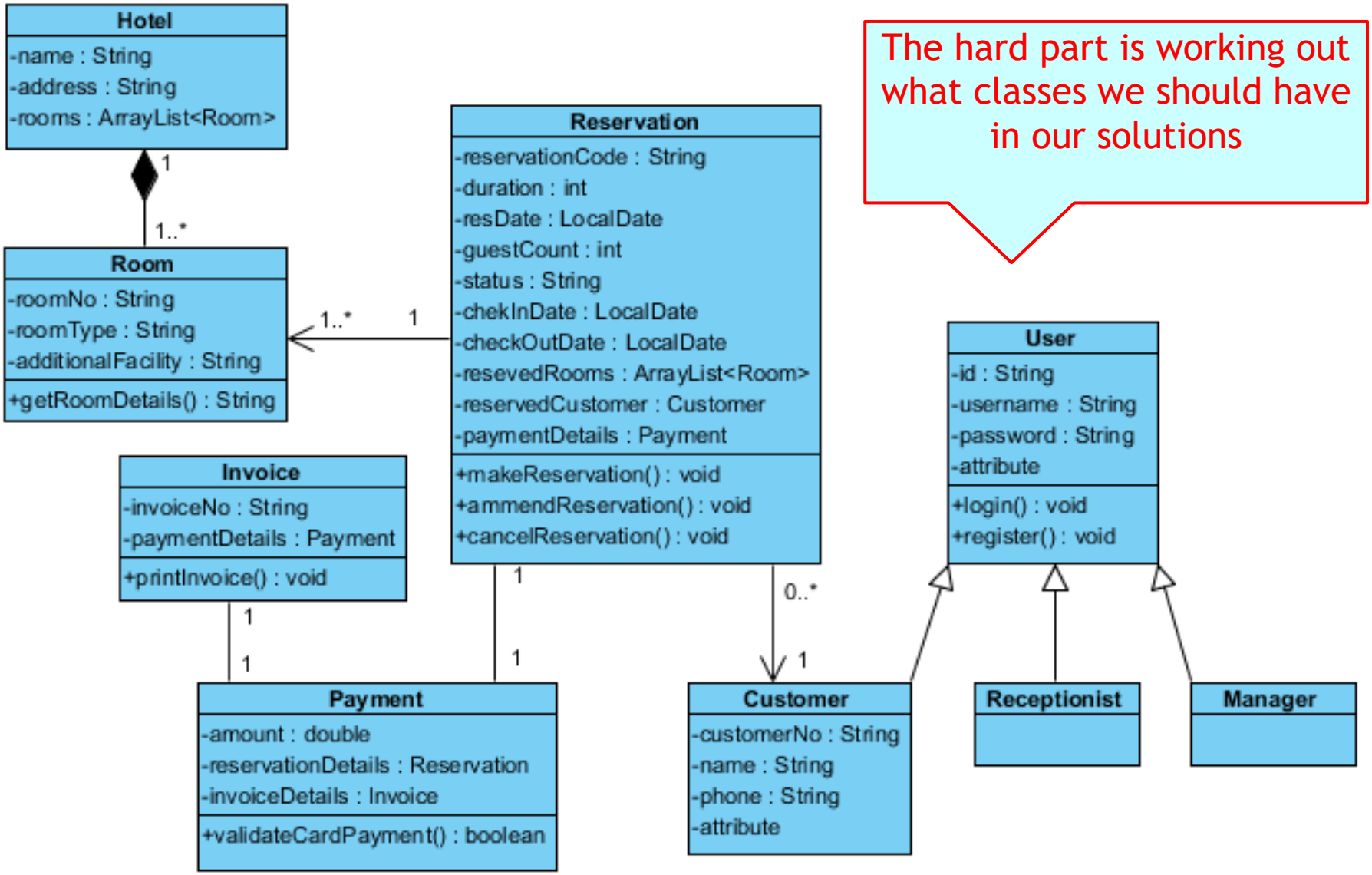
- Bidirectional Association


```
class A
{
        private B myB;
        ……
        ……
        ……
}
```

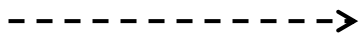
```
class B
{
        private A myA;
        ……
        ……
        ……
}
```

Sample class diagram: Hotel Reservation System

OO systems are made up of objects working with other objects



Class Diagram - Relationship Types

- **Associations** 
  - Associations between classes most often represent instance variables that hold references to other objects.
  - E.g: Student and Course
- **Generalization** 
  - Models “is a” and “is like” relationships, helping to easily reuse existing data
  - E.g: A student is a person. A faculty member is a person.
- **Realization** 
  - Is a relationship between the blueprint class and the object containing its respective implementation level details.
  - Relationship between the interface and the implementing class
- **Dependency** 
  - Used when one class depends on another class.
  - indicate that the connection between them is at a higher level of abstraction than an association relationship.

Class Diagram - Relationship Types

- ***Aggregations*** 
  - Is a specialization of association, specifying whole-part relationship between objects
  - E.g: team and employees
- ***Compositions*** 
  - Stronger form of aggregation, where the whole and the parts have coincident lifetimes
  - When an object contains the other object, if the contained object cannot exist without the existence of container object
  - E.g: House and rooms

Class Diagram - Relationship Types

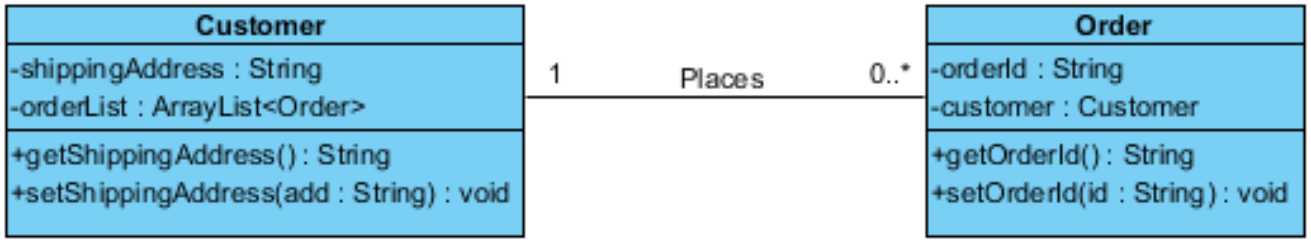
Multiplicity Indicators

- Is the description of the number of objects that can participate in the association

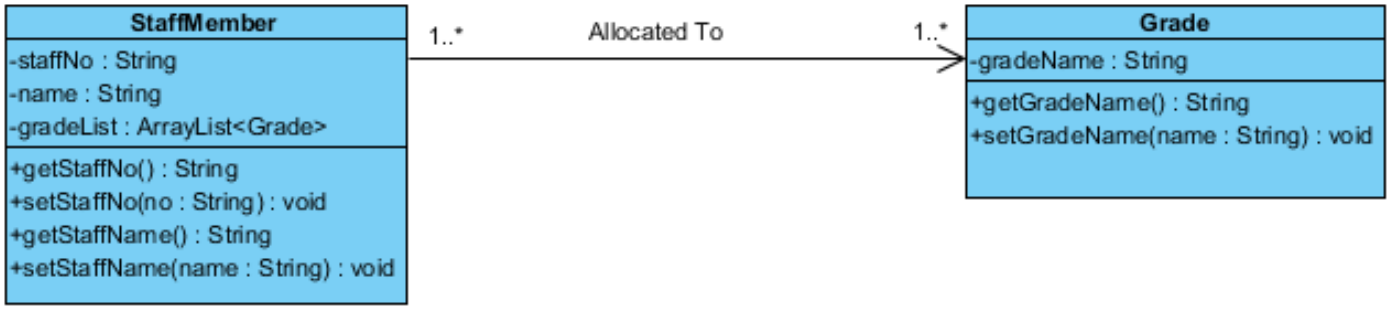
Indicator	Meaning
0..1	Zero or one
1	One only
0..*	Zero or more
1..*	One or more
N	Only n (where n>1)
*	Many
0..n	Zero to n (where n>1)
1..n	One to n (where n>1)
n..m	Where n and m both>1
n..*	n or more, where n>1

Class Diagram - Relationship Types

- Association Guidelines
  - Center names on associations
  - Word association names left to right

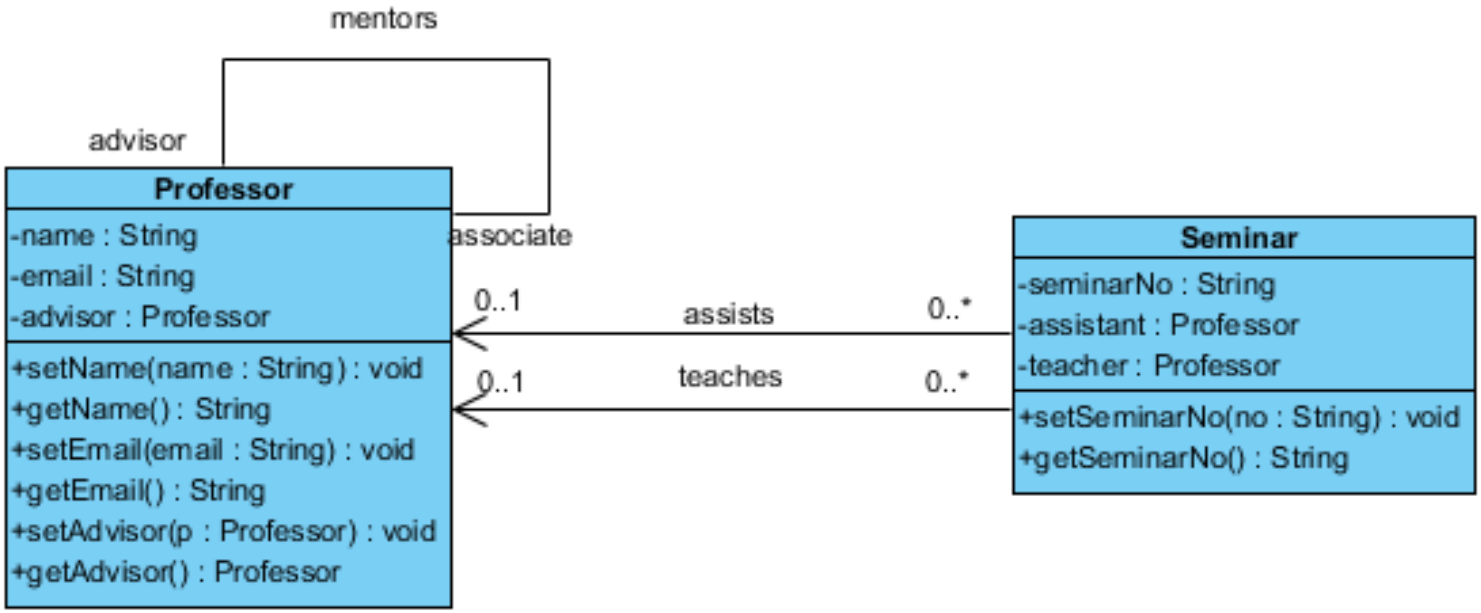


  - Indicate direction only on unidirectional associations



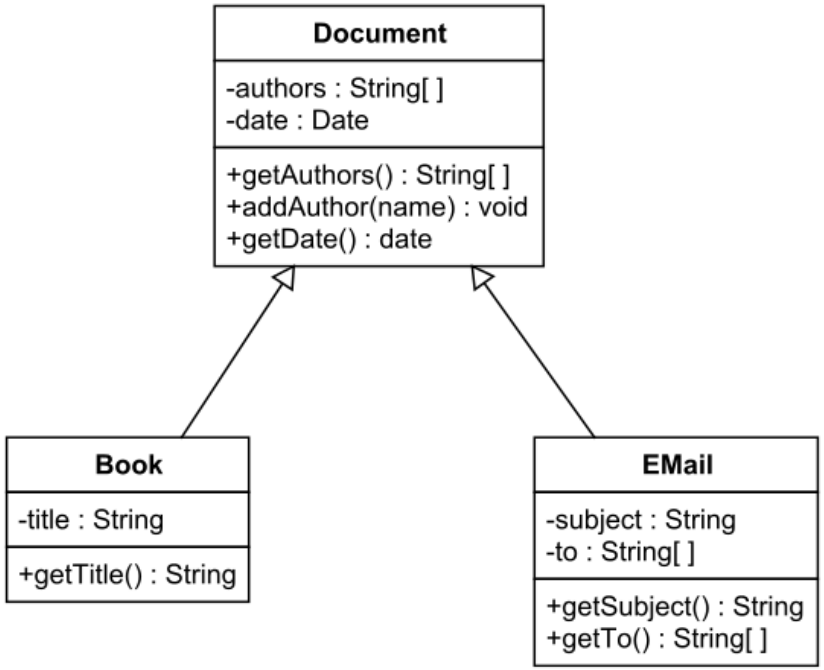
Class Diagram - Relationship Types

- Association Guidelines Cont..
  - Indicate role names on recursive associations
  - Indicate role names when multiple associations between two classes exists



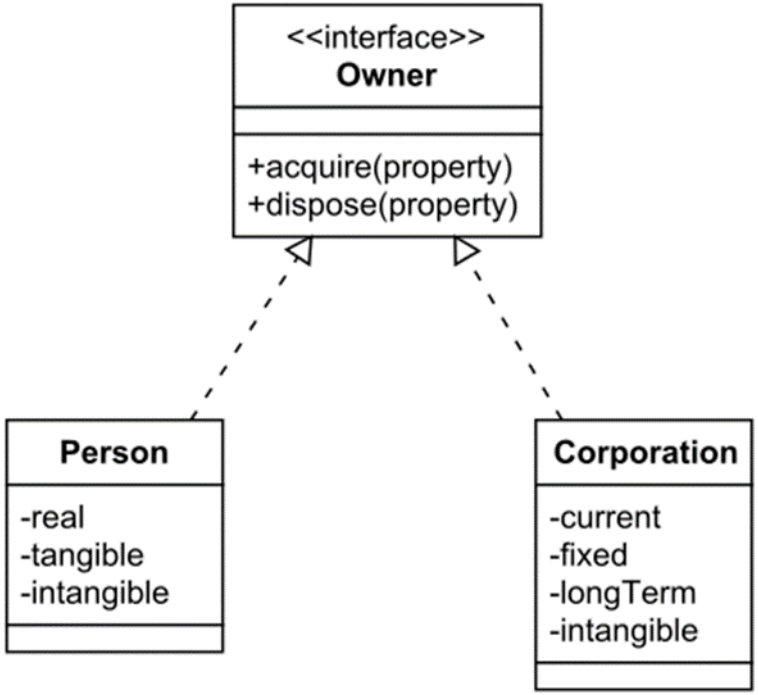
Class Diagram - Relationship Types

- Generalization Guidelines
  - Apply the sentence rule for inheritance
  - A subclass IS A KIND OF a superclass
  - Place subclasses below superclasses
  - A subclass should inherit everything



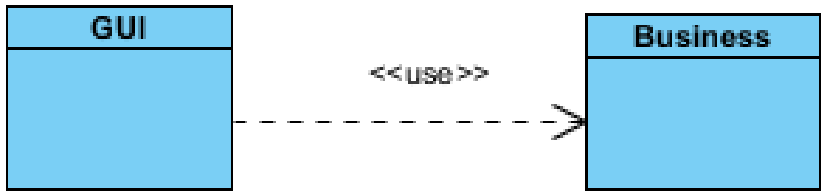
Class Diagram - Relationship Types

- Realization Guidelines
  - Realization is the relationship between the class and the interface.
  - Place implementation class below the interface class



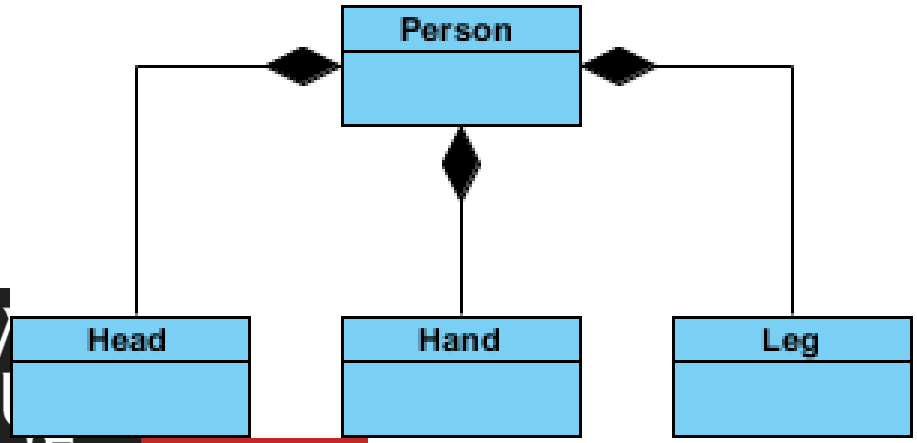
Class Diagram - Relationship Types

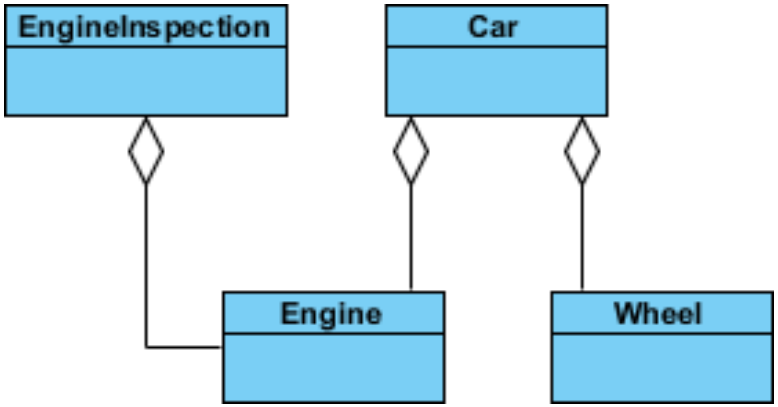
- Dependency Guidelines
  - A dependency exists between two classes if changes to the definition of one class may cause changes to the other.
  - Between classes dependencies may exist due to various reasons:
    - A client class uses a supplier class that has global scope
    - A client class uses a supplier class as a parameter for one of its operations
    - A client class uses a supplier class as a local variable for one of its operations
    - A client class sends a message to a supplier class



Class Diagram - Relationship Types

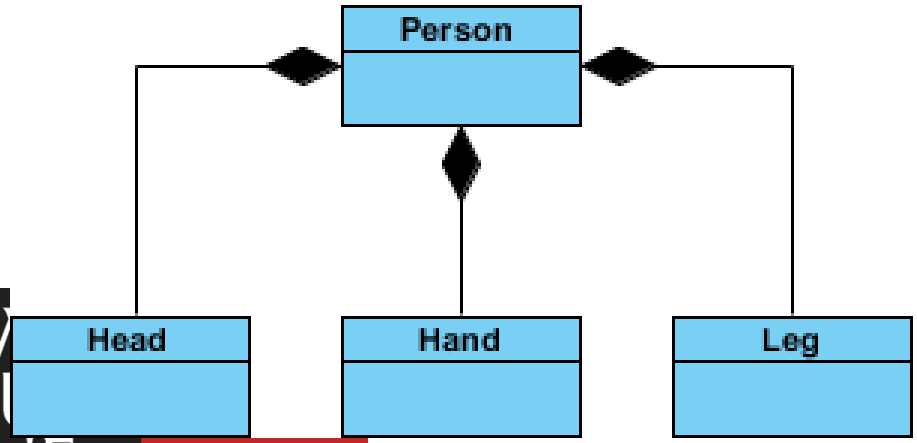
- Aggregation/ Composition Guidelines
  - The whole **has (a)** part(s)
  - Aggregation and Composition are specific cases of association. In both aggregation and composition object of one class "owns" object of another class.

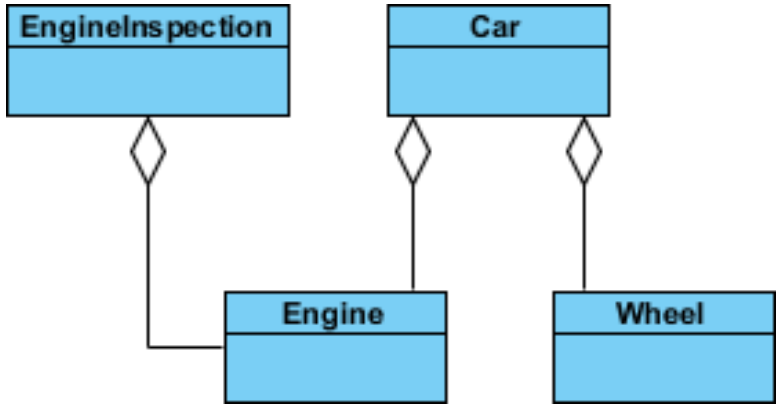




Class Diagram - Relationship Types

- Aggregation/ Composition Guidelines
  - Aggregation implies a relationship where the part can exist independently of the whole. Example: Class (whole) and Student (part). Delete the Class and the Students still exist.
  - Composition implies a relationship where the part cannot exist independent of the whole. Example: House (whole) and Room (part). Rooms don't exist separate to a House.





Defining Directed Associations


```
class A
{
      private B b;
      ……
      ……
      ……
}
```

Defining Directed Associations


```
import java.util.ArrayList;
public class A
{
      private ArrayList<B> bList;
      ……
}
```

Defining Directed Associations

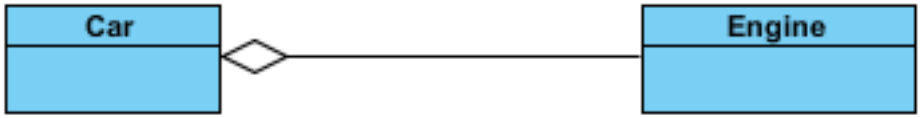

```
public class A
{
      private B[] bList;
      public A()
      {
            bList = new B[10];
      }
}
```

Defining Bidirectional Associations


```
class A
{
      private B myB;
      ……
      ……
      ……
}
```

```
class B
{
      private A myA;
      ……
      ……
      ……
}
```

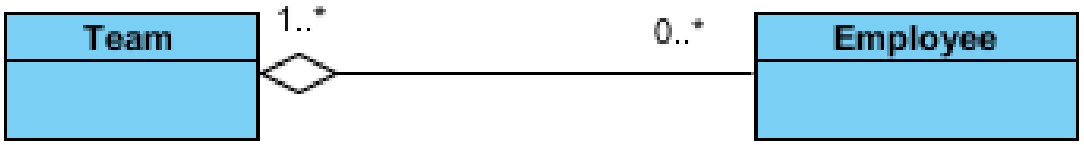
Defining Aggregations




```
public class Car
{
        private Engine engine;
        public Car(Engine e)
        {
                engine = e;
        }
}
```

```
public class Engine
{
        public Engine()
        {
                ……
        }
}
```

Defining Aggregations




```
public class Employee
{
	public Employee()
	{
		……
	}
}
```

```
public class Team
{
	private ArrayList<Employee> memberList;
	public Team()
	{
		this.memberList = new ArrayList<Employee>;
	}
}
```

Defining Compositions

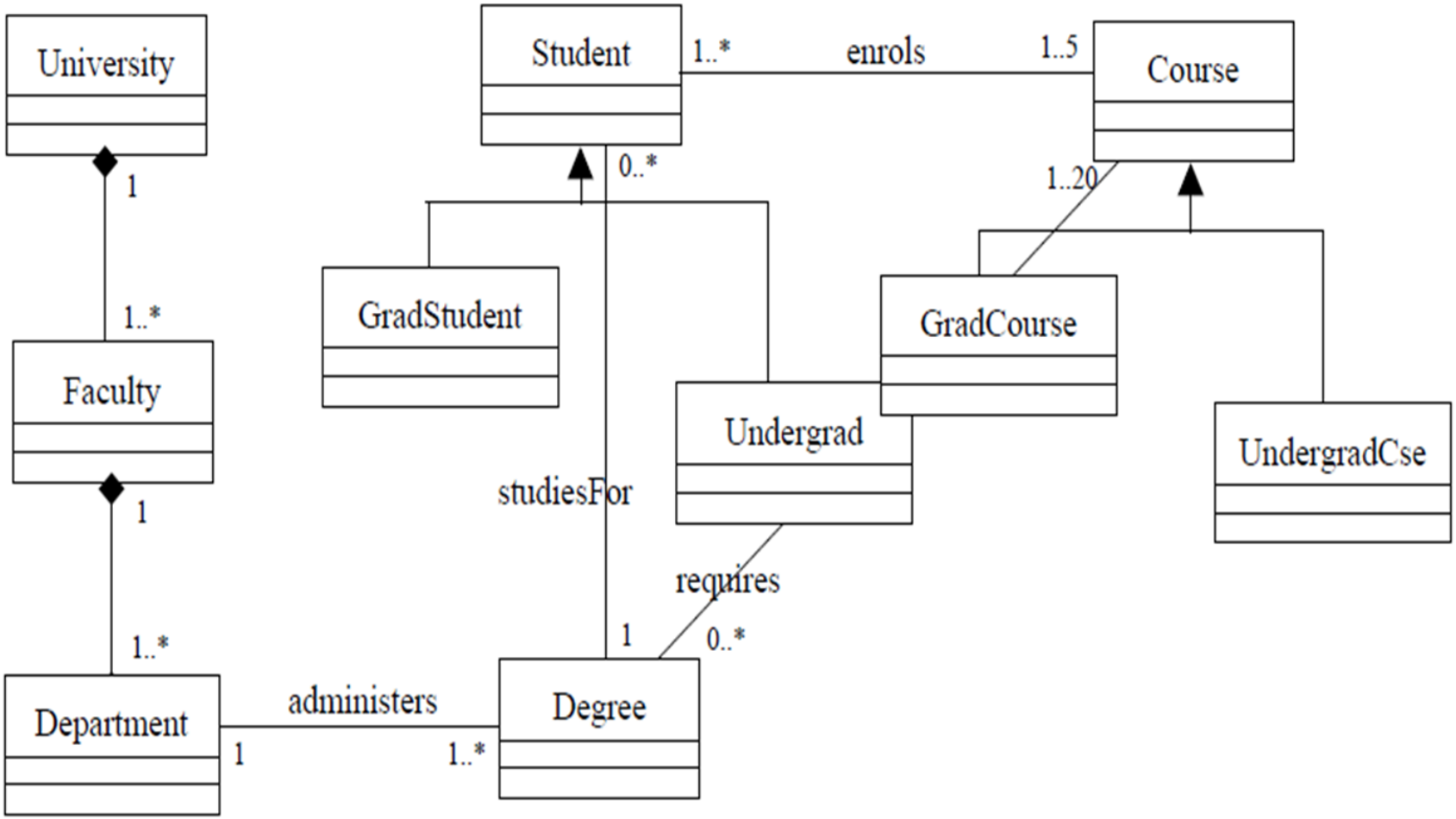

```
public class Room
{
    ……
}
```

```
Public class Building
{
    private Room room;
    public Building()
    {
        this.room = new Room();
    }
}
```

Scenario 1

- **Create a class diagram for the scenario below:**

A university offers degrees to students. The university consists of faculties each of which consists of one or more departments. Each degree is administered by a single department. Each student is studying towards a single degree. Each degree requires 1 to 20 courses. A student enrolls in 1-5 courses. A course can be either graduate or undergraduate, but not both. Likewise, students are graduates or undergraduates but not both.



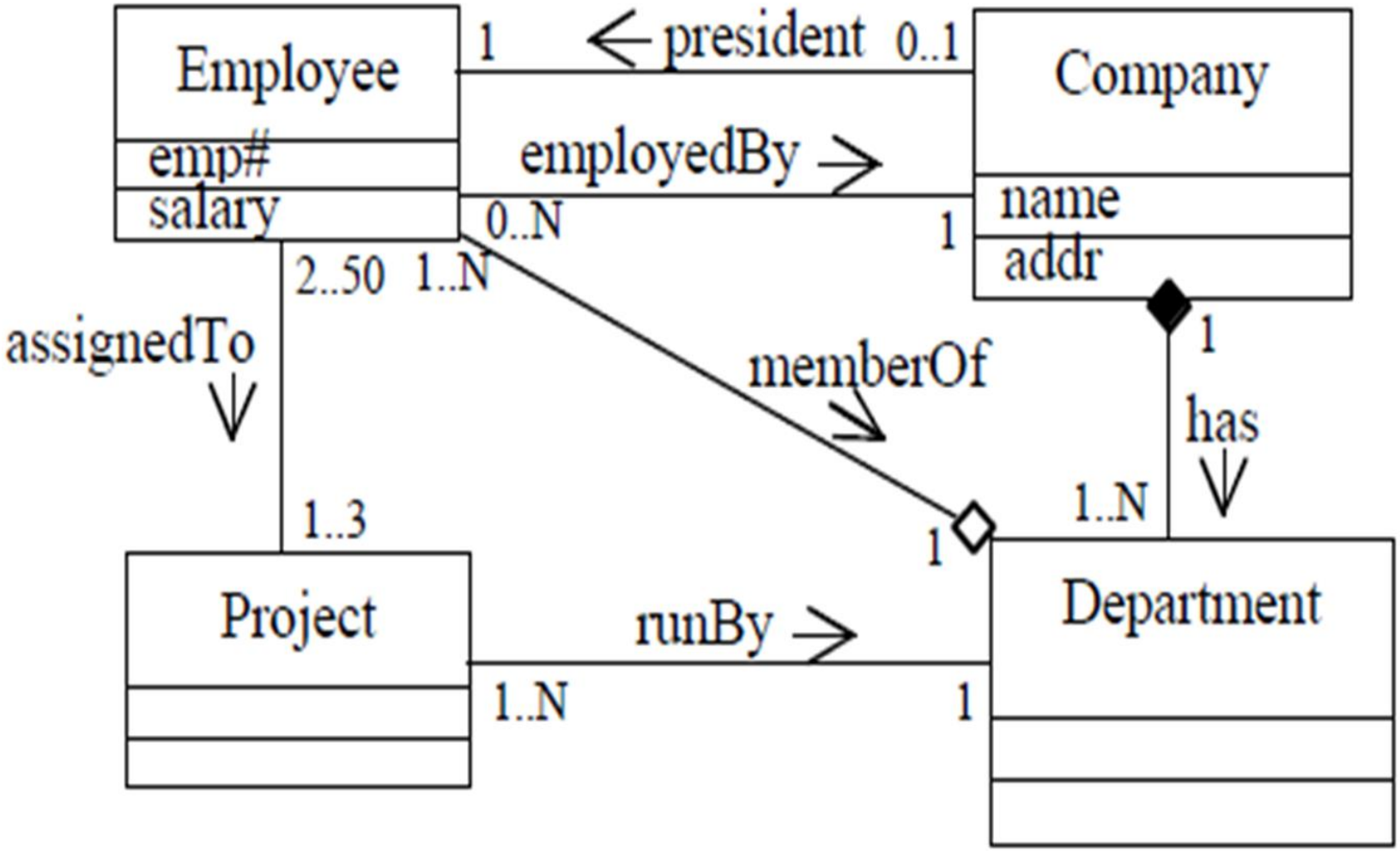
Scenario 2

- **Consider the world of companies**:

Companies employ employees (who can only work for one company), and consist of one or more departments.

Each company has a single president, who is an employee. Departments have employees as members and run projects (one or more.)

Employees can work in 1 to 3 projects, while a project can have 2 to 50 assigned employees.



SWINBURNE UNIVERSITY OF TECHNOLOGY

OOP

Sequence Diagrams

Learning outcome

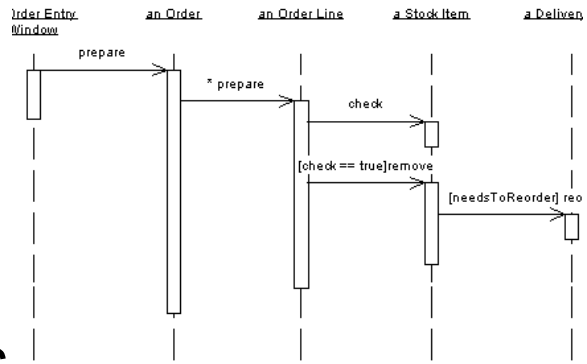
- Students should be able to draw sequence diagrams for identified use cases using appropriate notations

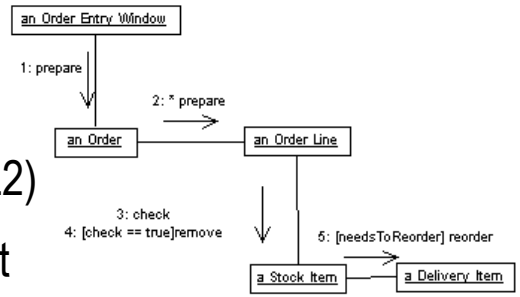
What are we Designing?

- Our design will include:
- Static Models
  - Structure of the classes and their relationships
    - Class Diagram
- Dynamic Models
  - Functionality, how the model works
    - Interaction diagrams
    - State diagrams

Interaction Diagrams

- Is a dynamic modelling technique describe how a group of objects collaborate in some behavior - typically a single use-case
- Modelling sequence of messages between objects
- There are two types of interaction diagrams
  - UML Sequence Diagrams
    - Looks at timing and order of messages
  - UML Collaboration Diagrams (Communication Diagram in UML2)
    - Looks at interactions between entities, focusing on object relationships





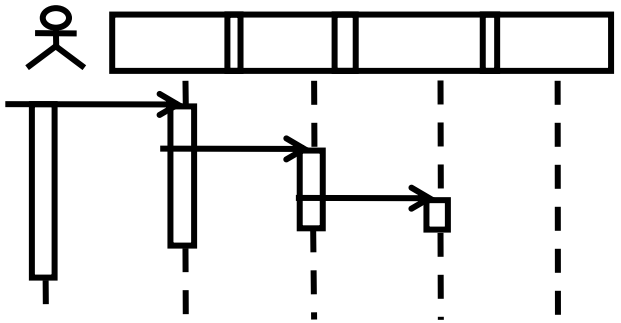
UML Sequence Diagrams

- Describe the flow of messages, events, actions between objects
- Show concurrent processes and activations
- Show time sequences that are not easily depicted in other diagrams
- Typically used during analysis and design to document and understand the logical flow of your system

Emphasis on time ordering!

Sequence Diagrams

- Describes a single operation or task being performed upon the model (one use case)
- Two Dimensions
  - Vertical
    - Time
    - Normally increases as you travel down the page
  - Horizontal
    - Object instances
    - Placed as you like, suggest trying to get arrows to travel left to right



Key parts of a sequence diag.

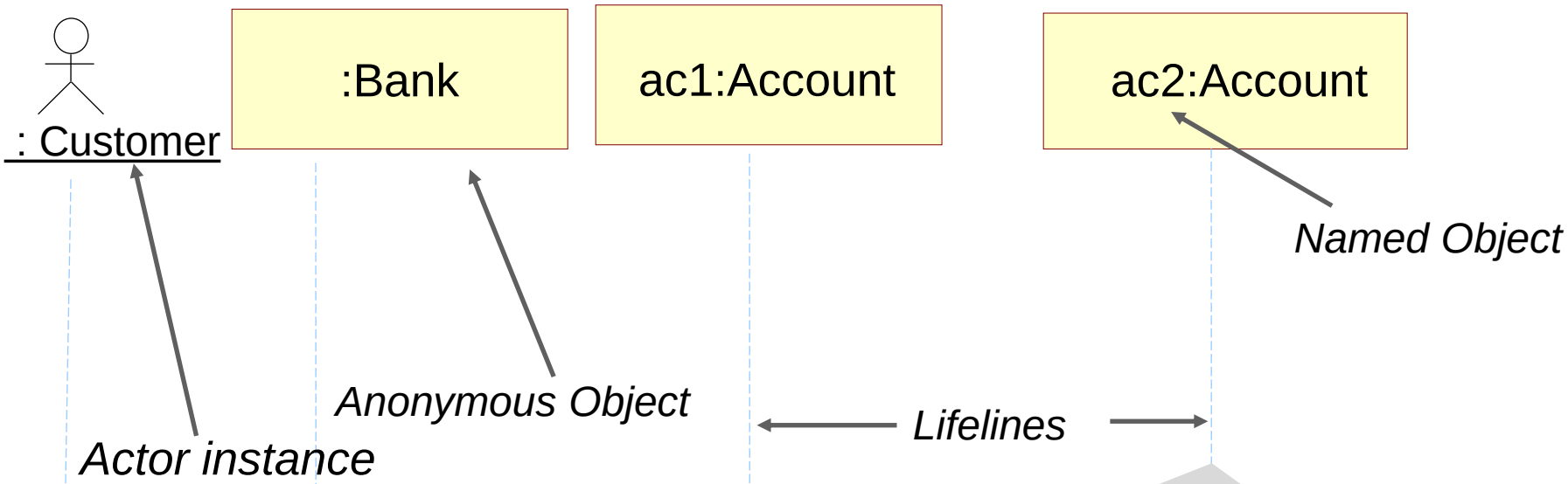
- participant: an object or entity that acts in the sequence diagram
- message: communication between participant objects
- the axes in a sequence diagram:
  - horizontal: which object/participant is acting
  - vertical: time

Diagram Purpose

- Helps to describe the design
  - What objects (entities) are involved in an interaction
  - What order method calls are performed in
- Do when
  - You need to describe the messages that occur when a non-trivial task is being performed
  - Test your class diagram
  - Improve the class diagram

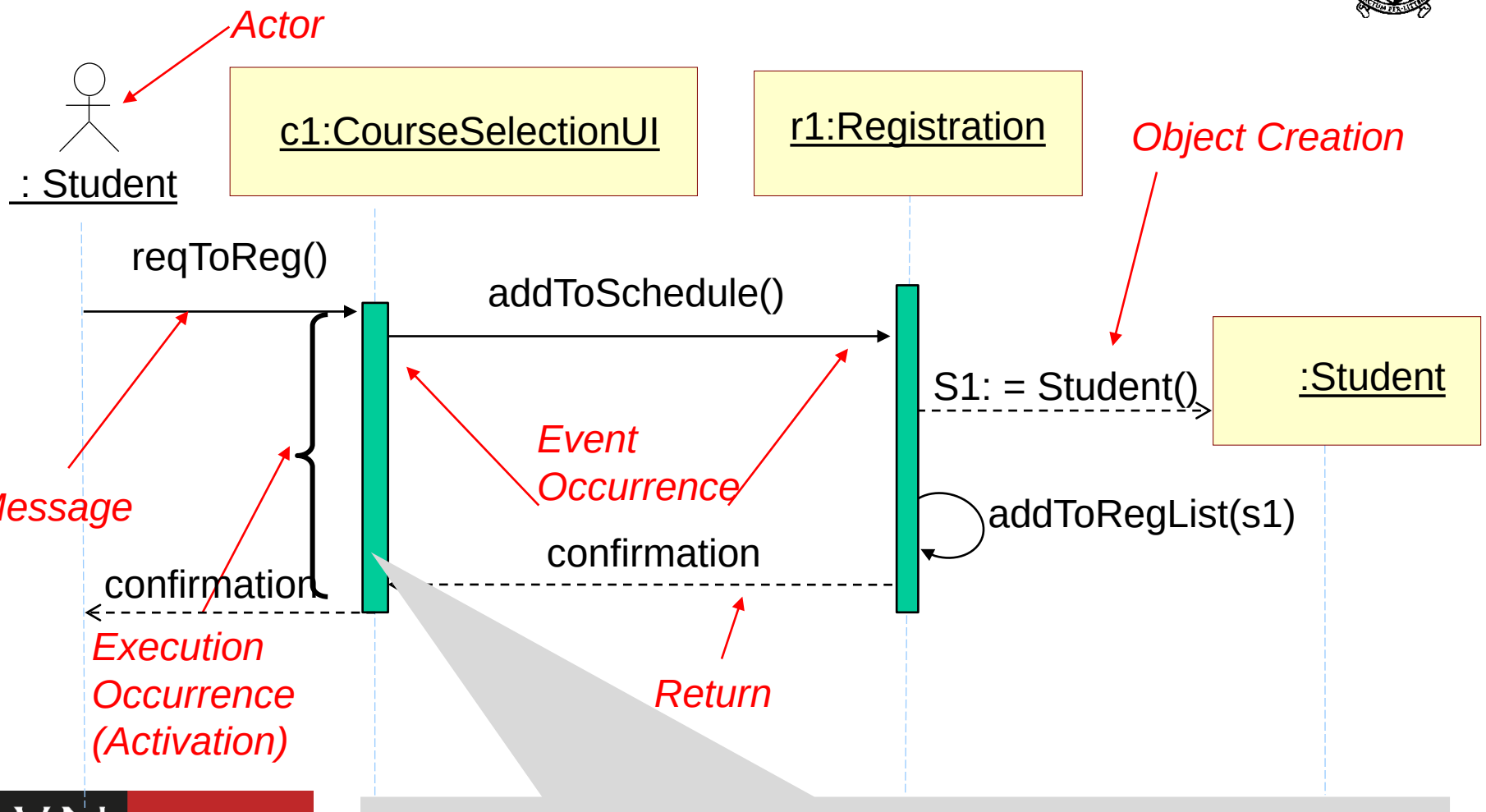
Representing objects

- Squares with object type, optionally preceded by object name
  - object's name
  - object's "life line" represented by dashed vert. line



Life line indicates life of the object for the role specified:

Sequence Diagrams: contents

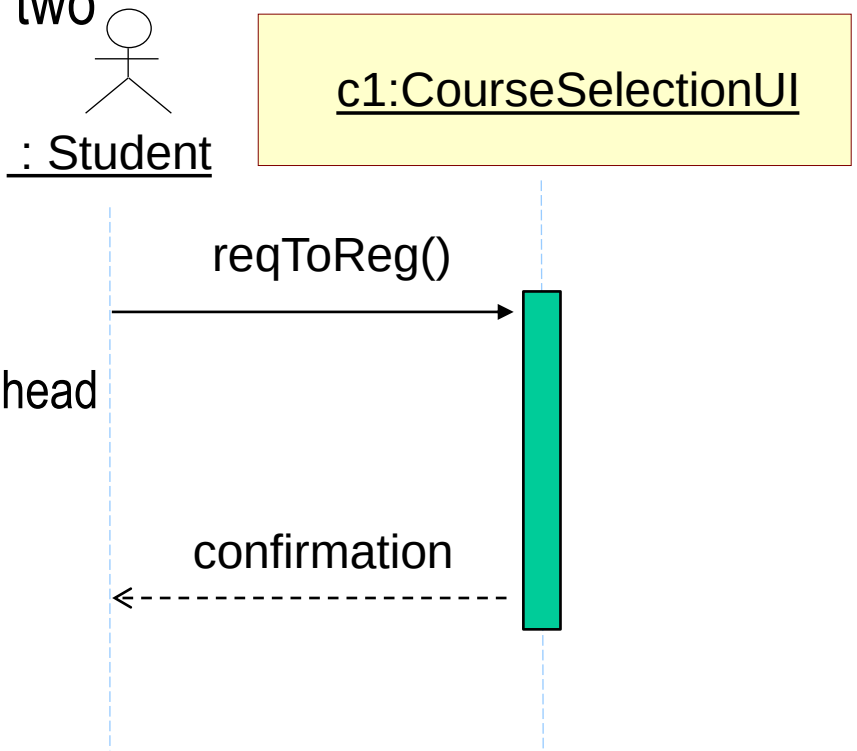


Activation

- Shows period during which the object is active
- Active from initial message until return

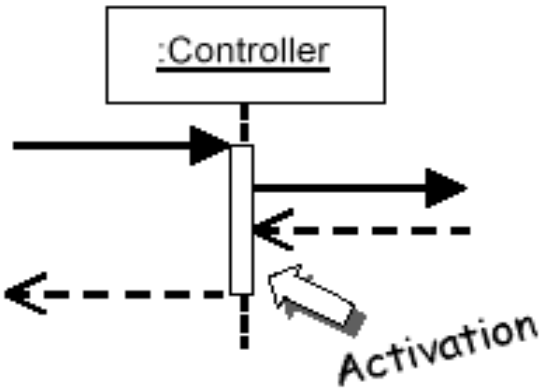
Sequence Diagrams: contents

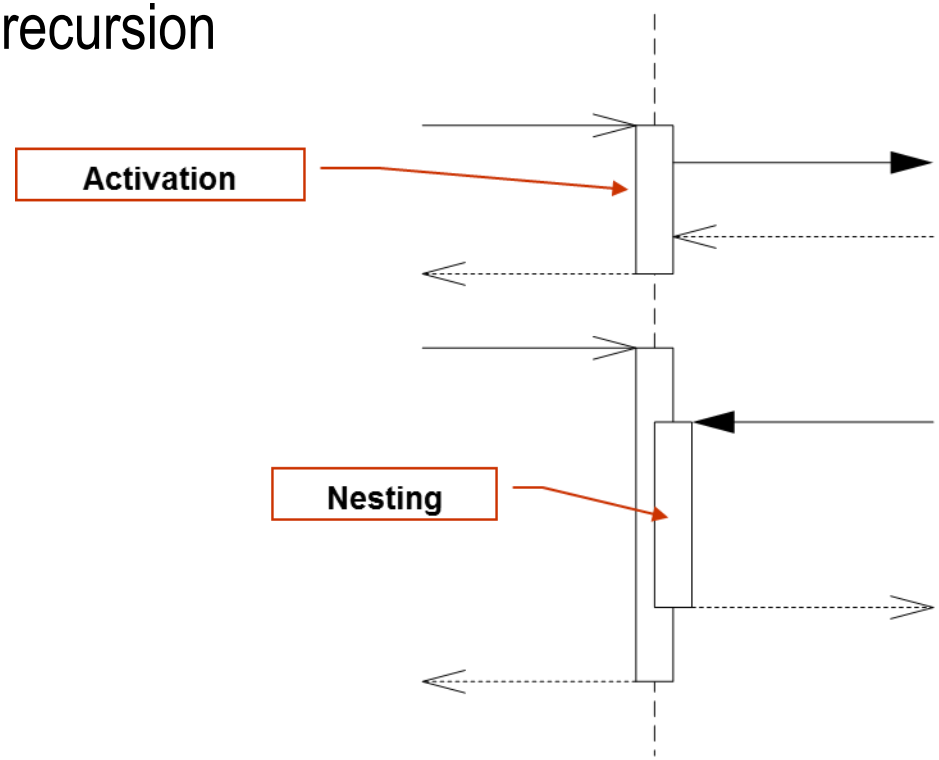
- Messages (method calls)
  - Indicates a communication between two objects
  - Operation Name (and Arguments)
  - Different based on situation
    - Solid line with Filled or Unfilled Arrow head for method calls
    - Dashed line with Arrow head for returning data



Activation

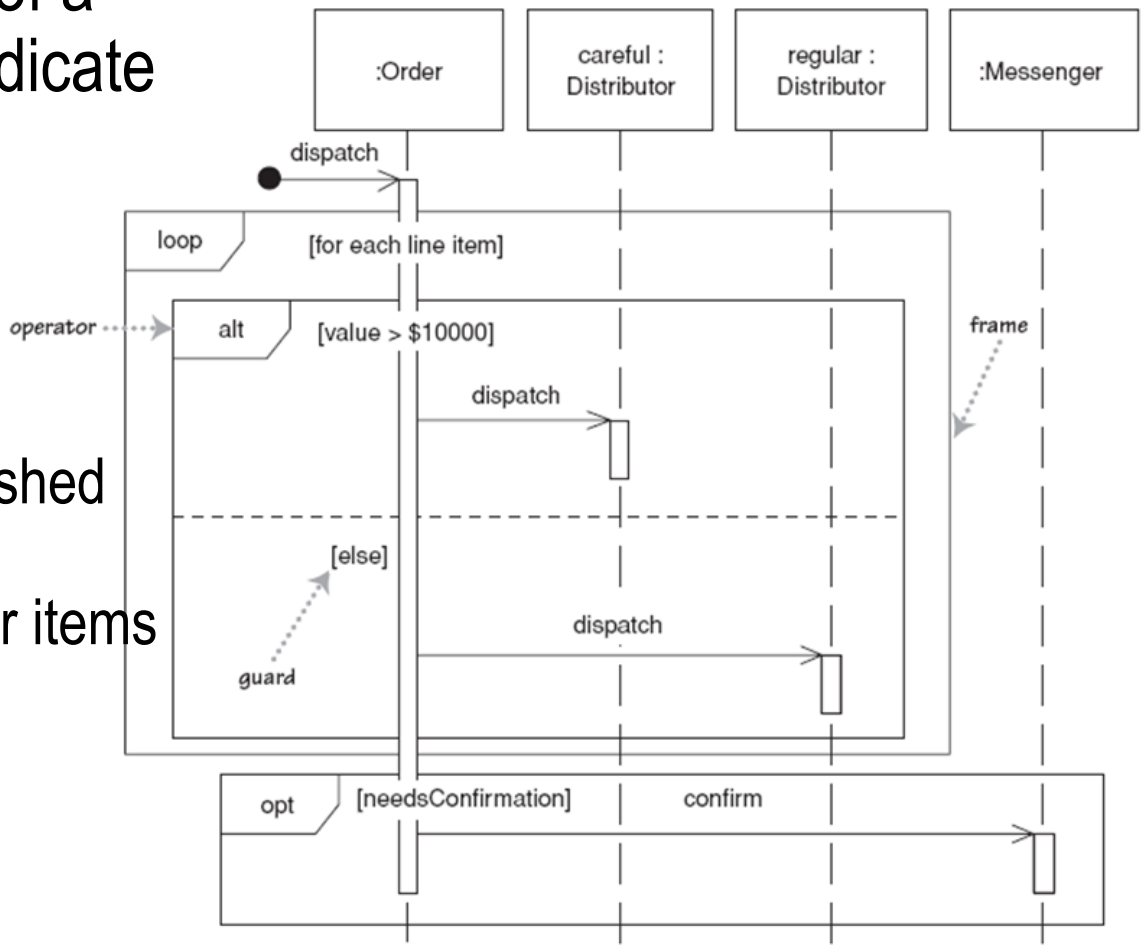
- **activation**: thick box over object's life line; drawn when object's method is on the stack
  - either that object is running its code, or it is on the stack waiting for another object's method to finish
  - nest to indicate recursion





Indicating selection and loops

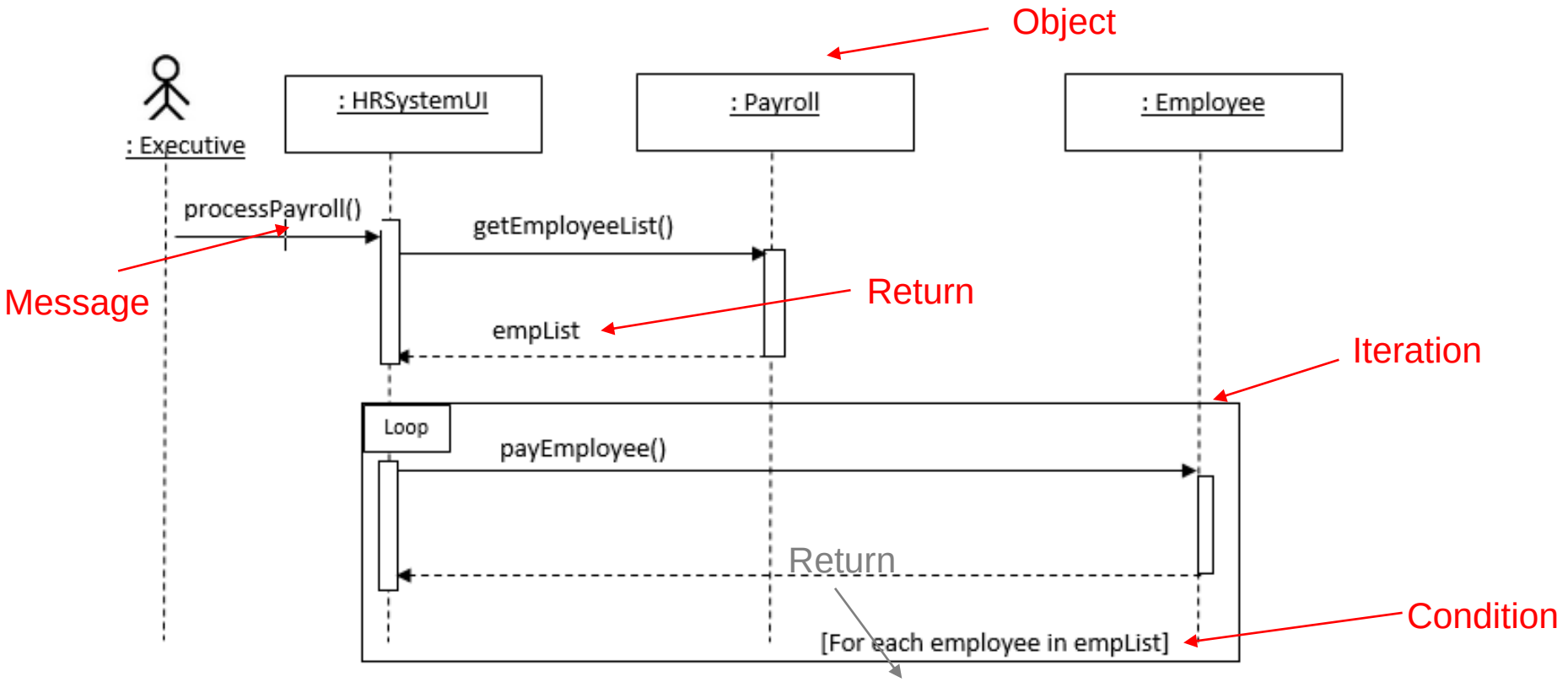
- frame: box around part of a sequence diagram to indicate selection or loop
  - if -> (opt) [condition]
  - if/else -> (alt) [condition], separated by horizontal dashed line
  - Loop -> (loop) [condition or items to loop over]



Sequence Diagram example

Use Case: Processing payroll

(get all employees, iterate through the list and pay each employee)

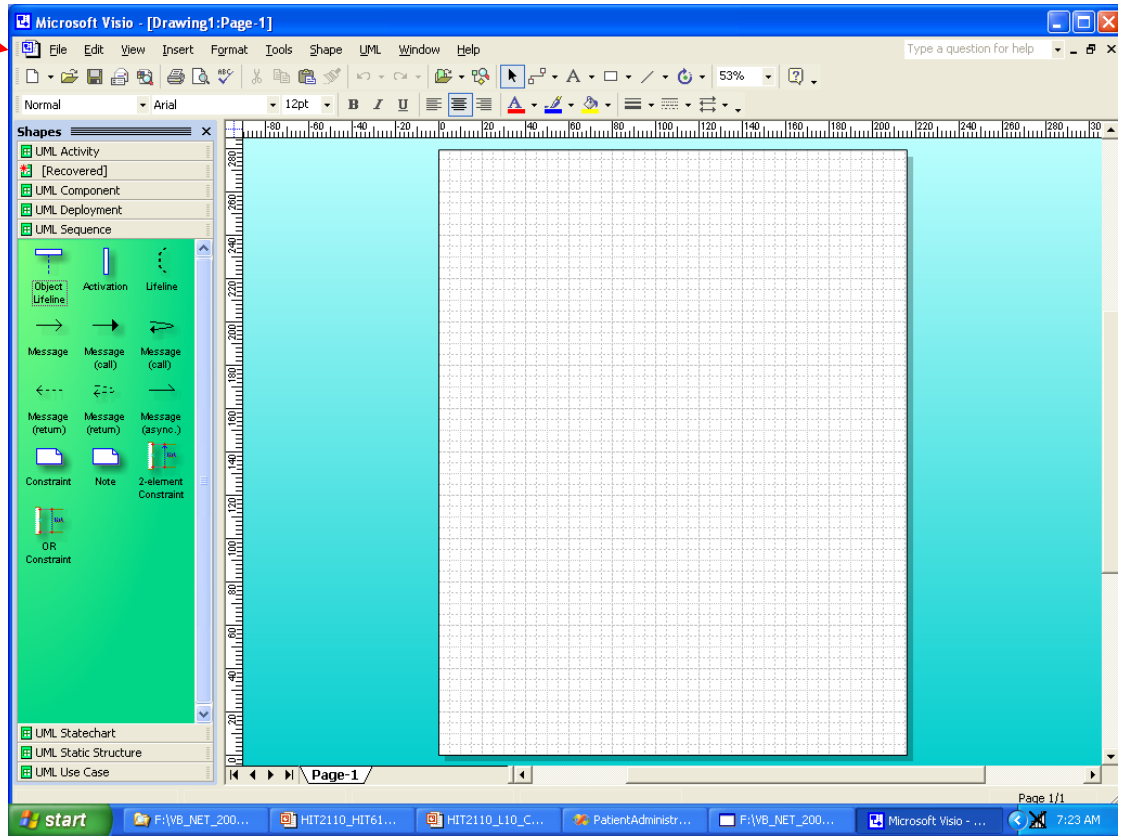


Review: Sequence Diagrams

- Describe a single use case
- Model Objects (Not a Class diagram)
- Model interactions between objects
- Can include
  - The User instance
  - The User Interface instance

Sequence Diagram designing tools

- Visio
- Visual Paradigm
- Rational Rows



Lets do it

- Use Case : Add Patient
- The user wants to add a new patient to the Patient Admin System.
  - ☐ User selects add patient at menu
  - ☐ System Creates Patient Object
  - ☐ User enters patients details
  - ☐ System adds patient to collection

Issues

? What Objects will be used to do this job

? In what order will the objects be used

? What methods will be called

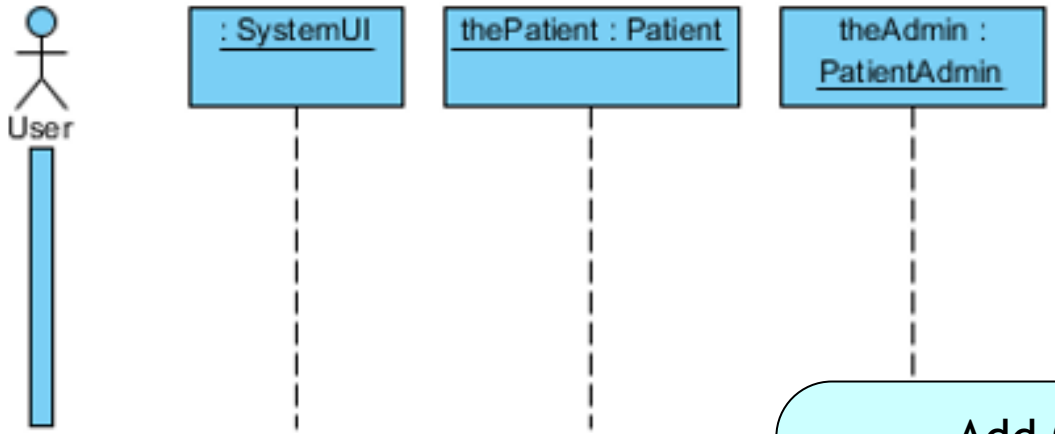
? What messages will be passed

? What data will be passed

? Where do you start your sequence diagram

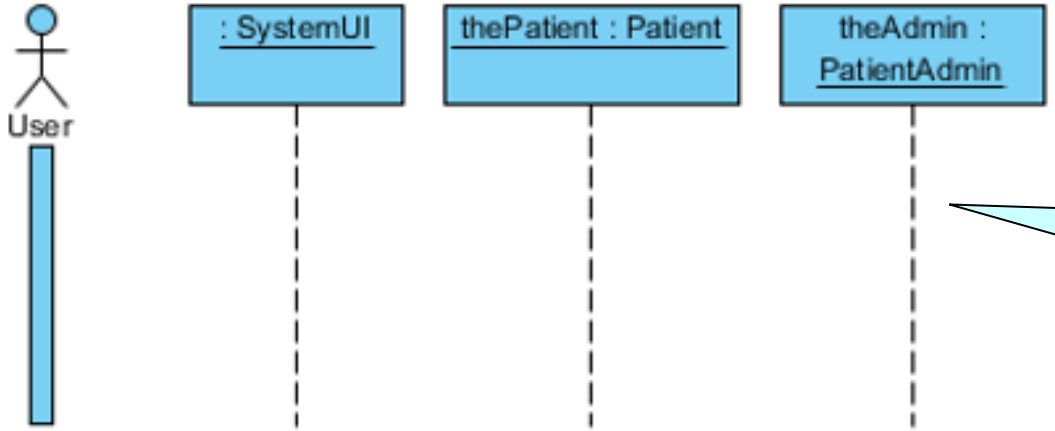
Objects Identified

- User - Actor
- User Interface (The console Application)
- A Patient Object
- A Patient Admin Object



Add Objects/Entities
from left to right
User first
Boundary Objects Next
Objects in order of use

Naming Conventions



Lifeline
Defines the life (existence of the object)

- See page 183 Schaum's Outlines
- Typically object name : class name
- If object anonymous then class name only

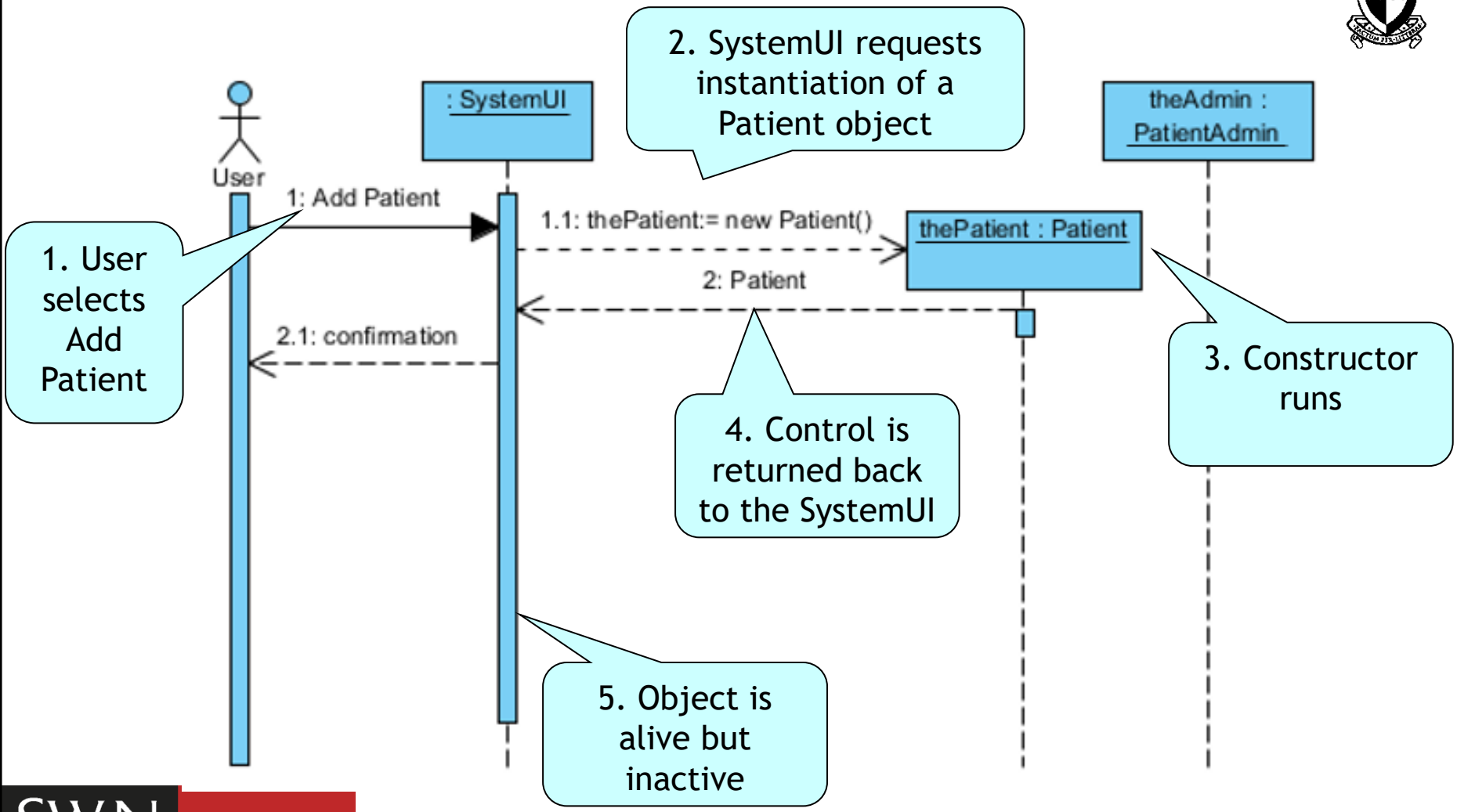
Objects used, may exist before & after this sequence.

Interactions Identified

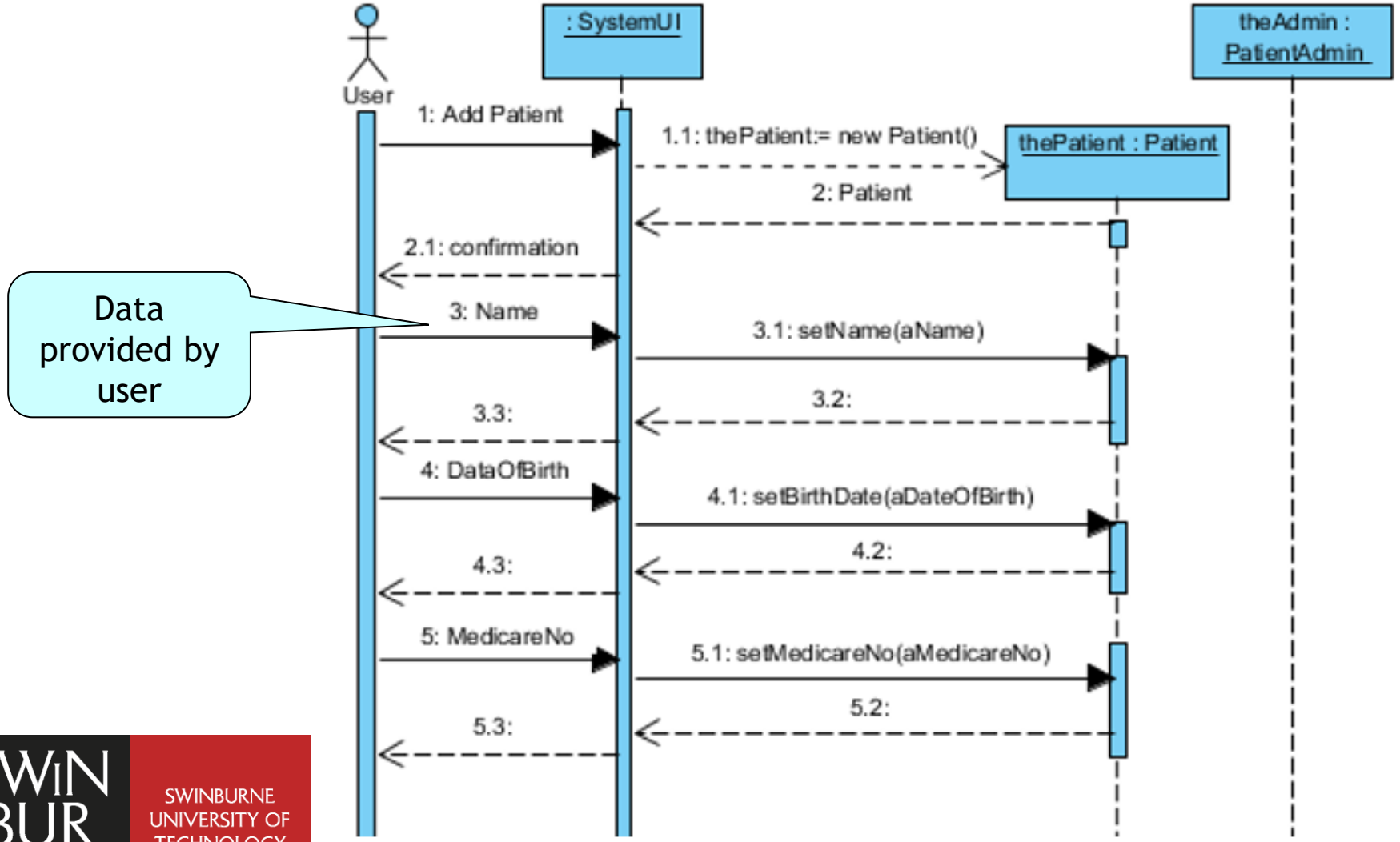
- User selects option - AddPatient
- SystemUI creates Patient Object
- SystemUI prompts user for details…
  - (User enters details….Name, Date of Birth, Medicare No)
- SystemUI sets objects values
- Check with admin object if Patient already exists
- If patient does not exist in the system, SystemUI calls PatientAdmin object to add student; else; a “Patient exist” message is displayed to user

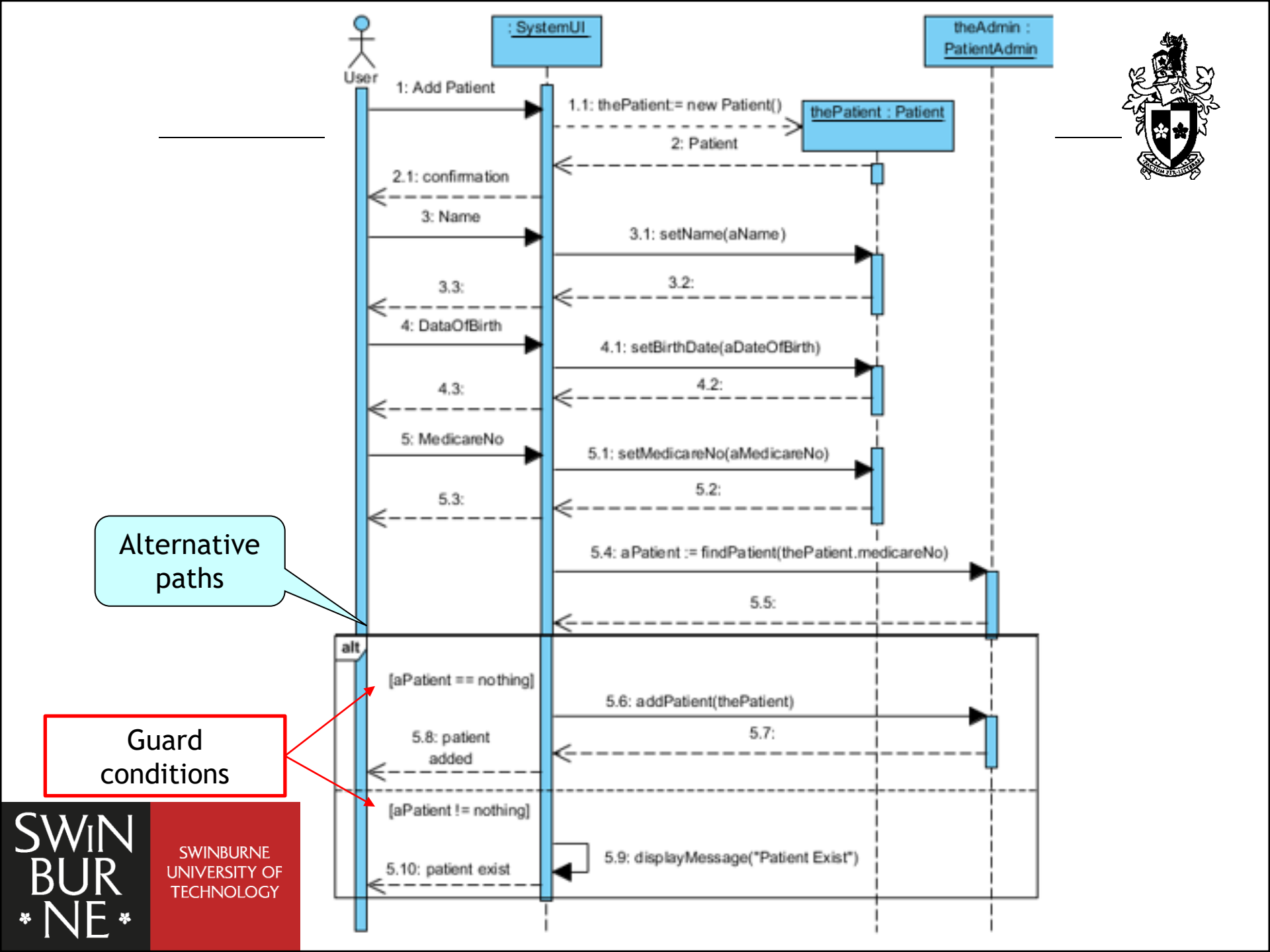
Remember Messages map to methods

When user selects Add Patient...



When user selects Add Patient…





Relationship with Class Diagrams

- Sequence diagrams allow us to
  - Develop the class diagram
  - Test the class diagram
  - Add more details to a class diagram
- Each Object in the sequence diagram will have a corresponding class in the class diagram
- If an Object passes a message to another object, then there is a relationship between those objects.
- All relationships should be shown in the class diagram

Additional Work

- Reading
  - Schaum's Outlines Chapter 9
- UML
  - http://www.uml.org/
- Sequence Diagrams
  - http://www.agilemodeling.com/artifacts/sequenceDiagram.htm