Lecture VII:

Object Modeling

General Views/expectations/opinions

Something to think about...

We shall require a substantially new manner of thinking if mankind is to survive.

Albert Einstein

Topics Covered

- Object oriented analysis
- Objects in information systems
- Attributes,
- Messages,
- Methods

How Object-Oriented Analysis Describes an Information System

Object-Oriented Analysis

- Object-oriented (O-O) analysis describes an information system by **identifying things** called **objects**

- An object represents **a real person**, **place**, **event**, or **transaction**

- For example, when a patient makes an appointment to see a doctor, the **patient** is an **object**, the **doctor** is an **object**, and the **appointment** itself is an **object**

Object-Oriented Analysis (2)

- O-O analysis **sees a system** from the **viewpoint** of the **objects** themselves as they **function** and **interact**

- The **end product** of O-O analysis is an **object model**, which **represents** the information **system** in terms of **objects** and **O-O concepts**

- **UML** will be used to develop **object models**

What an Object Represents in an Information System

Objects

- An object represents a **person**, a **place**, an **event**, or a **transaction** that is **significant** to the **information system**

- **DFDs** are created to treat **data** and **processes separately**

- An object, however, **includes data** and the **processes** that **affect** the **data**

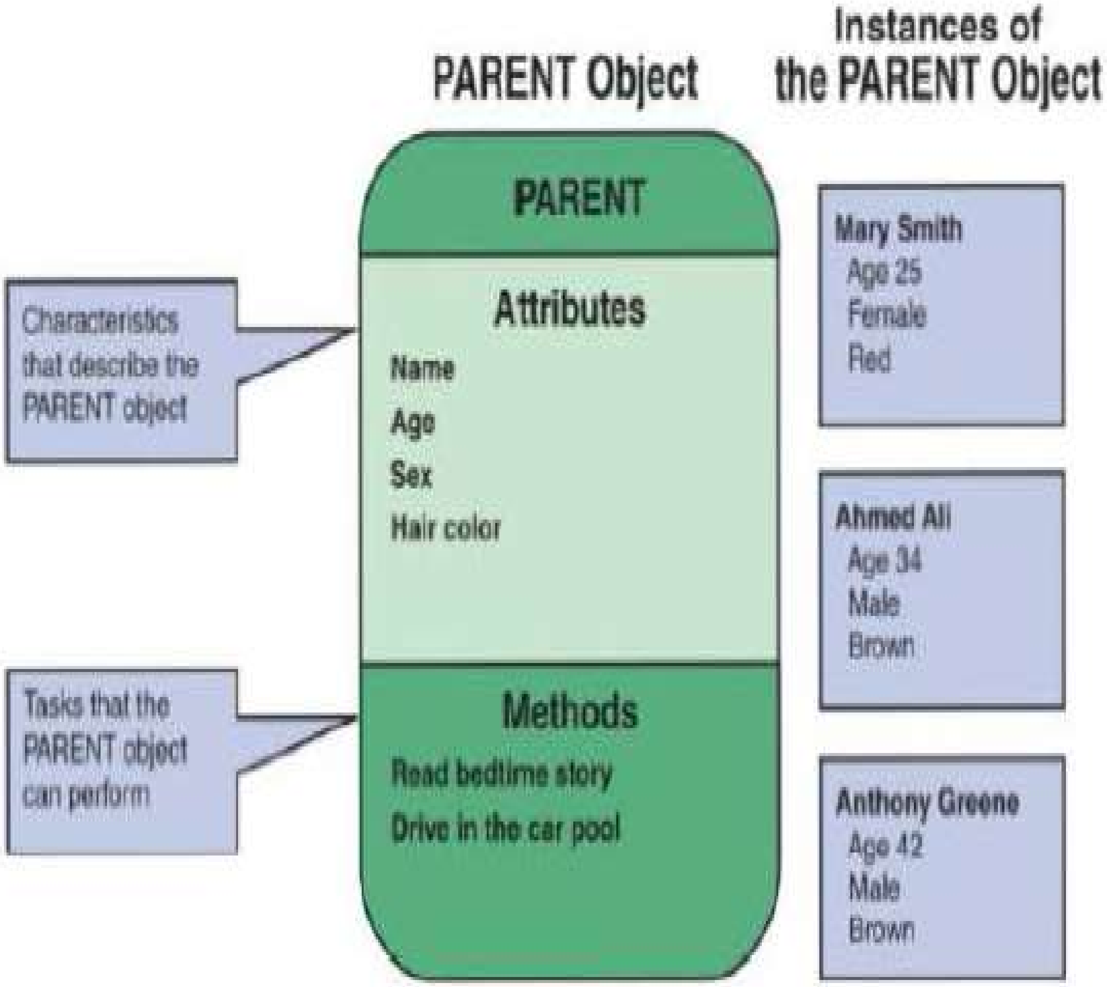
Objects - Example

- We can use UML to **describe** a **family** with **parents** and **children**

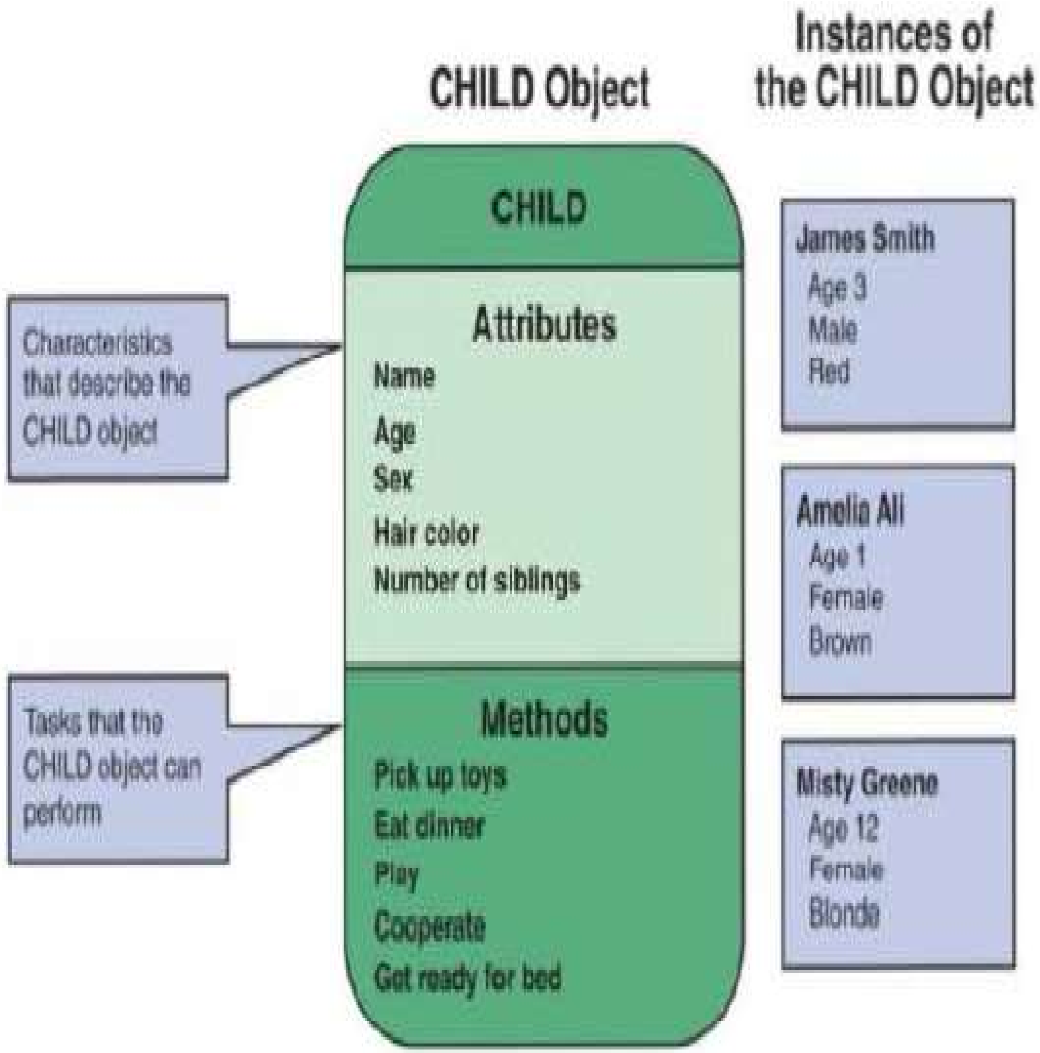
- UML represents an **object** as a **rectangle** with the object *name at the top*, followed by the **object's attributes** and **methods**

Objects – Example (2)

- A **PARENT** object with attributes ???
- If there are two parents, then there are _____ instances of the PARENT object
- The PARENT object can perform methods like ???



Objects – Example (3)



- A **CHILD** object attributes ???
- If there are four children, then there are ____ instances of the CHILD object
- The CHILD object can perform methods like ???

Object Attributes

Attributes

- An object has certain attributes, which are **characteristics** that **describe** the **object**

- **Some** objects might have **a few attributes**; others might **have dozens**

Attributes (2)

- Objects can **inherit**, or **acquire**, certain attributes from other objects
- Objects can have a **specific attribute** called a **state**
- The **state of an object** is an adjective that **describes** the object's **current status**
- For example, depending on the state, a bank account can be active, inactive, closed, or frozen

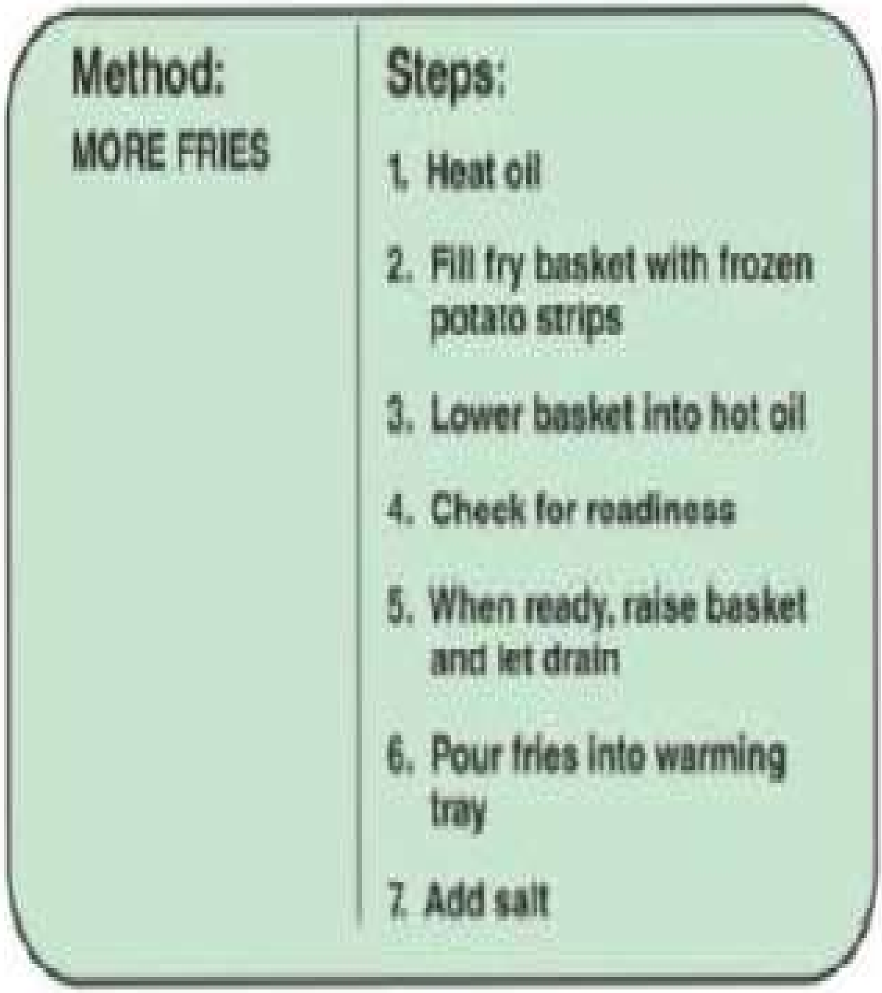
Object Methods

Methods

- Methods are **tasks** or **functions** that the object **performs** when it **receives a message**, or **command**, to do so
- For example, *a car performs a method* called OPERATE WIPERS when it is sent a message with the wiper control
- A method **defines specific tasks** that an object can perform
- Methods **resemble verbs** since they describe ***what*** and ***how*** an object **does something**

Methods - Example

- Consider a **CHEF** who prepares fries in a fast-food cafe



Method:

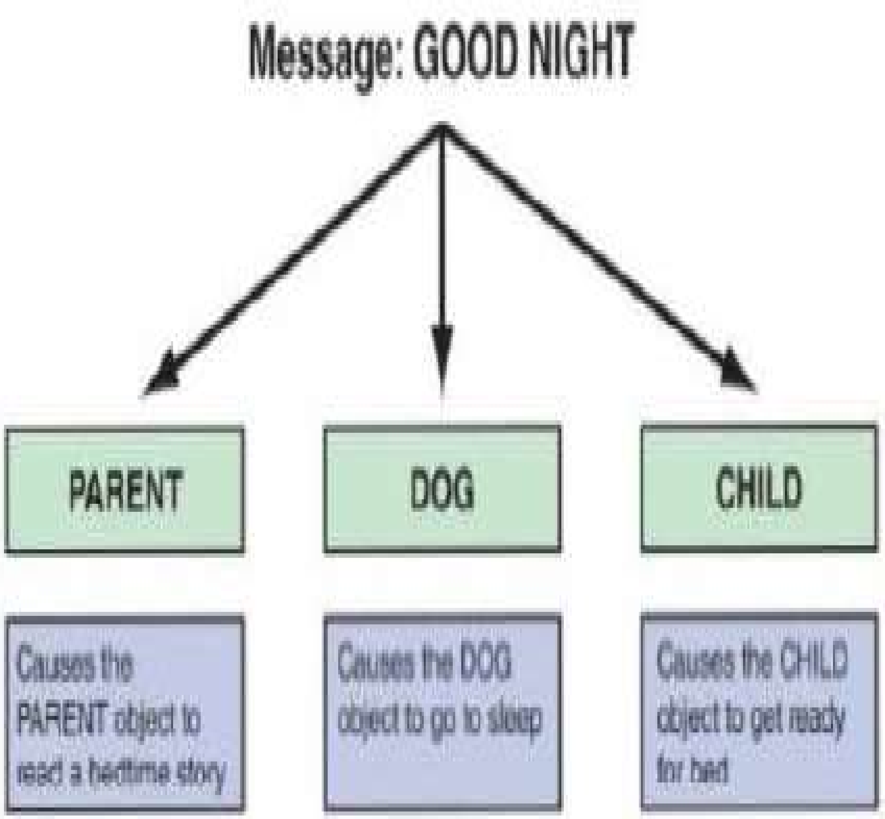
MORE FRIES

Steps:

1. Heat oil
2. Fill fry basket with frozen potato strips
3. Lower basket into hot oil
4. Check for readiness
5. When ready, raise basket and let drain
6. Pour fries into warming tray
7. Add salt

Object Messages

Messages


Message: GOOD NIGHT
PARENT
DOG
CHILD
Causes the PARENT object to read a bedtime story
Causes the DOG object to go to sleep
Causes the CHILD object to get ready for bed

- A message is a **command** that tells an **object** to **perform** a certain **method**
- The **same message** to *two different objects* can produce **different results**
- The concept that a **message gives different meanings to different objects** is called **polymorphism**

Messages (2)

- An object can be viewed as a **black box**, because a **message** to the object **triggers changes** within the object **without specifying how** the changes must be carried out
- The black box concept is an example of **encapsulation**, which means ***that all data and methods are self-contained***
- A black box **does not want** or **need outside interference**, hence **modularity** is well accomplished

Messages (3)

- O-O designs are implemented with **O-O programming languages** like ???

- A major advantage of O-O designs is that
  - systems analysts can **save time** and **avoid errors** by using objects, and
  - programmers can **translate the designs** into code, then
  - work with **reusable program modules** that have been tested and verified

Classes

Classes

- An object **belongs** to a **group** or **category** called a **class**
  - For example, Ford belong to a class called CAR

- An **instance** is a **specific member** of a class

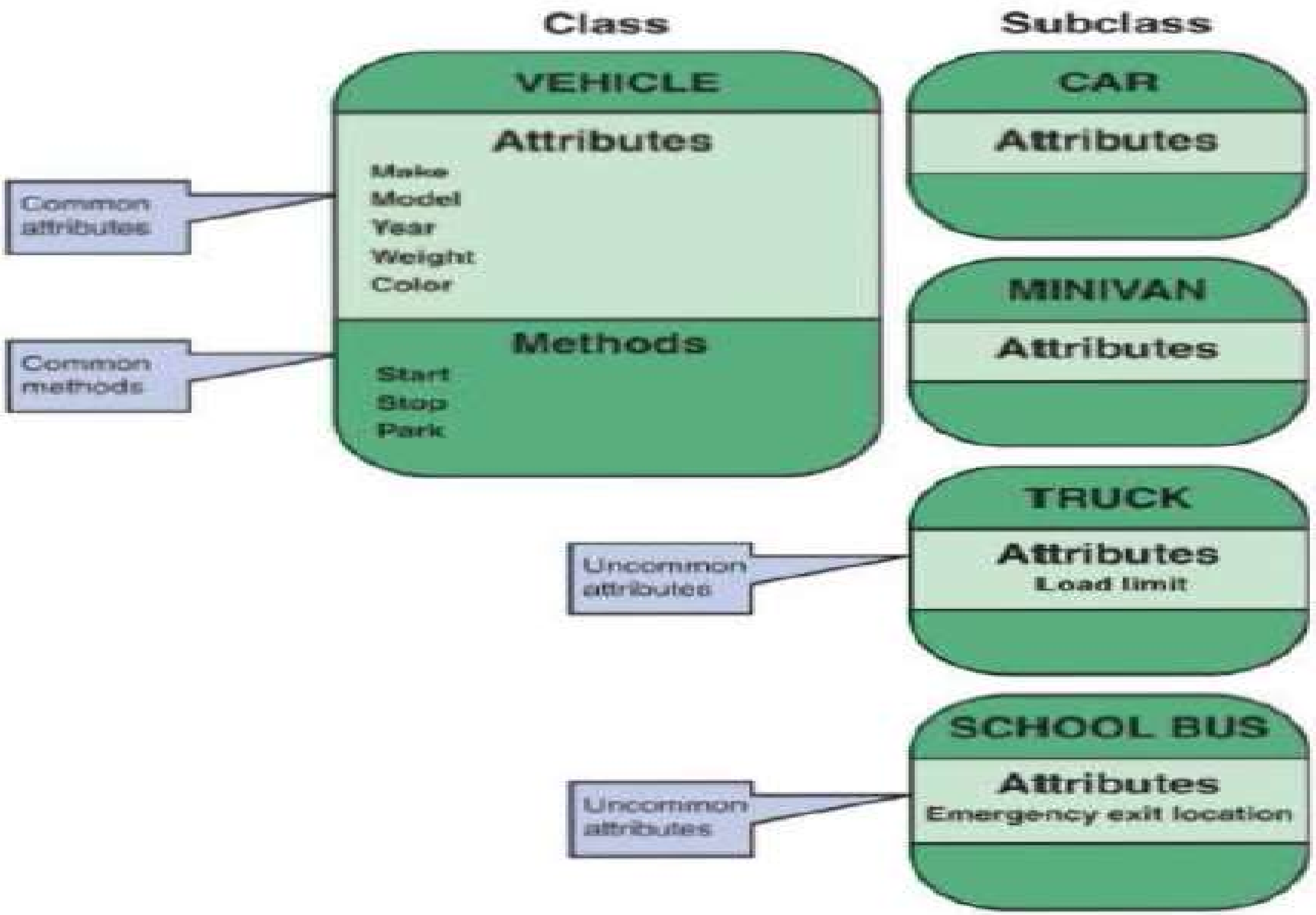
- **Many instances** of the CAR class may be observed:
  - The TRUCK class, the MINIVAN class, and the SPORT UTILITY VEHICLE class

Classes (2)

- All **objects** within a class share **common attributes** and **methods**, so a class is a *blueprint* or *template* for all the objects within the class

- **Objects** within a class can be **grouped** into **subclasses**, which are **more specific categories** within a class
  - For example, TRUCK objects represent a subclass within the VEHICLE class, along with other subclasses called CAR, MINIVAN, and SCHOOL BUS

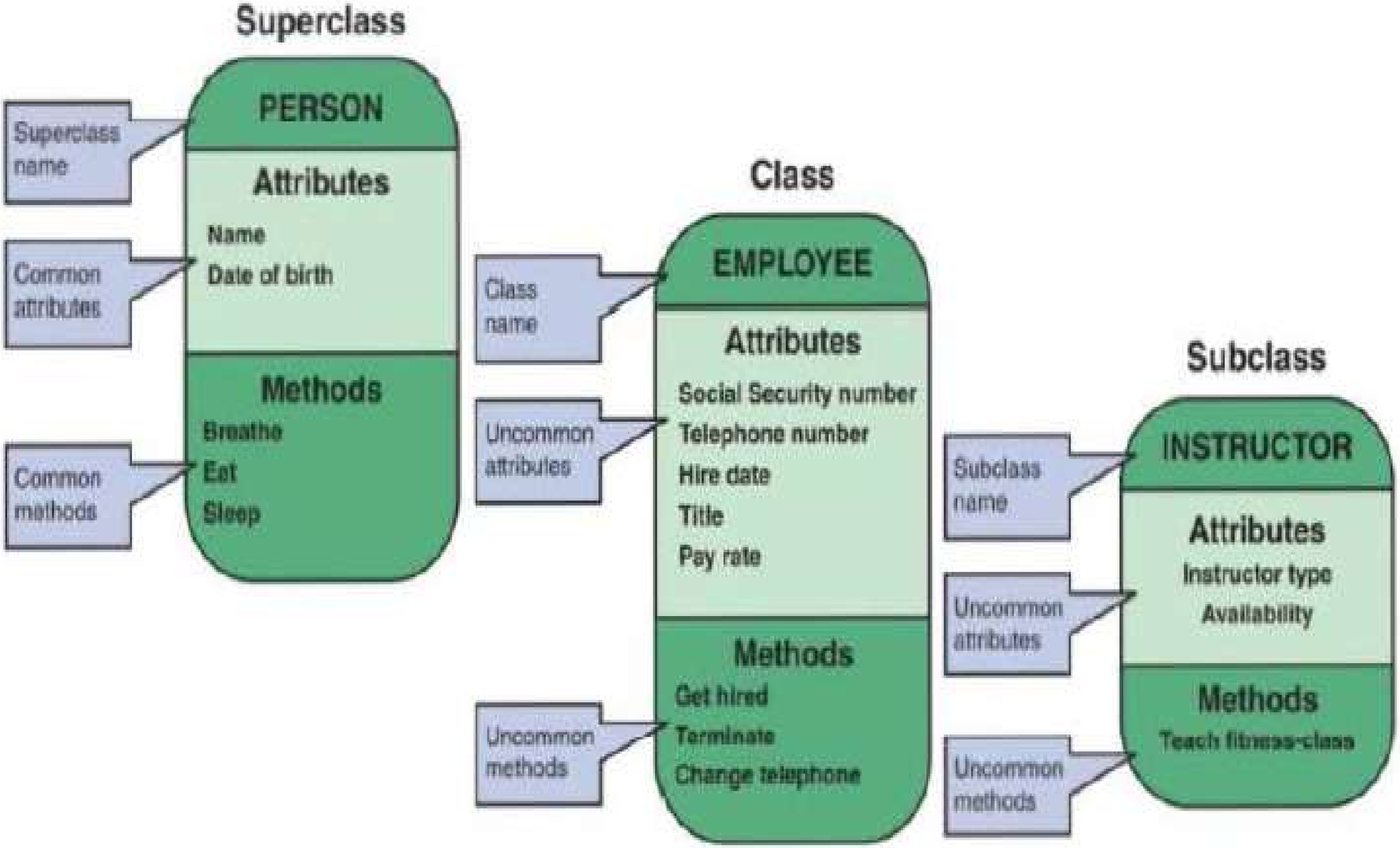
Classes (3)


Class
Subclass
VEHICLE
Attributes
Make
Model
Year
Weight
Color
Methods
Start
Stop
Park
Common attributes
Common methods
CAR
Attributes
MINIVAN
Attributes
TRUCK
Attributes
Load limit
Uncommon attributes
SCHOOL BUS
Attributes
Emergency exit location
Uncommon attributes

Classes Example (4)

- Consider a **fitness center** that might have **students**, **instructors**, **class schedules**, and a **registration process**
  - The EMPLOYEE class belongs to the PERSON superclass, because every employee is a person, and the INSTRUCTOR class is a subclass of EMPLOYEE

Classes (5)


Superclass
PERSON
Attributes
Name
Date of birth
Methods
Breathe
Eat
Sleep
Superclass name
Common attributes
Common methods
Class
EMPLOYEE
Attributes
Social Security number
Telephone number
Hire date
Title
Pay rate
Methods
Get hired
Terminate
Change telephone
Class name
Uncommon attributes
Uncommon methods
Subclass
INSTRUCTOR
Attributes
Instructor type
Availability
Methods
Teach fitness-class
Subclass name
Uncommon attributes
Uncommon methods

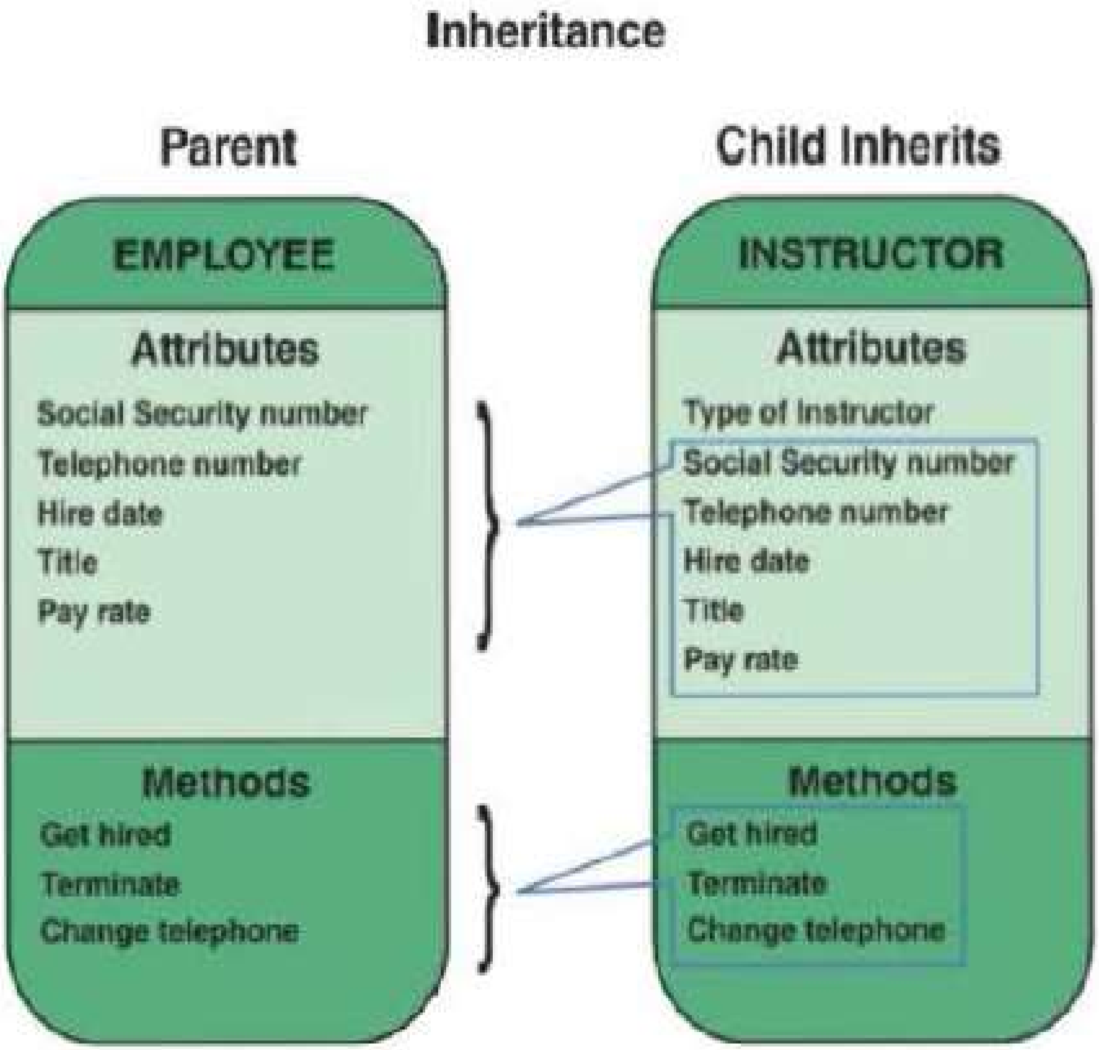
Relationships Among Objects and Classes

Relationships Among Objects and Classes

- Relationships enable **objects to communicate** and **interact** as they perform business functions and transactions required by the system
- Relationships describe what objects **need to know about each other**, how **objects respond to changes in other objects**, and the **effects of membership** in classes, superclasses, and subclasses
- Some relationships are stronger than others

Relationships Among Objects and Classes (2)

- The **strongest relationship** is called **inheritance**
- Inheritance enables an object, called **a *child***, to **derive** one or more of its **attributes** from **another object**, called a ***parent***



Drawing an Object Relationship Diagram

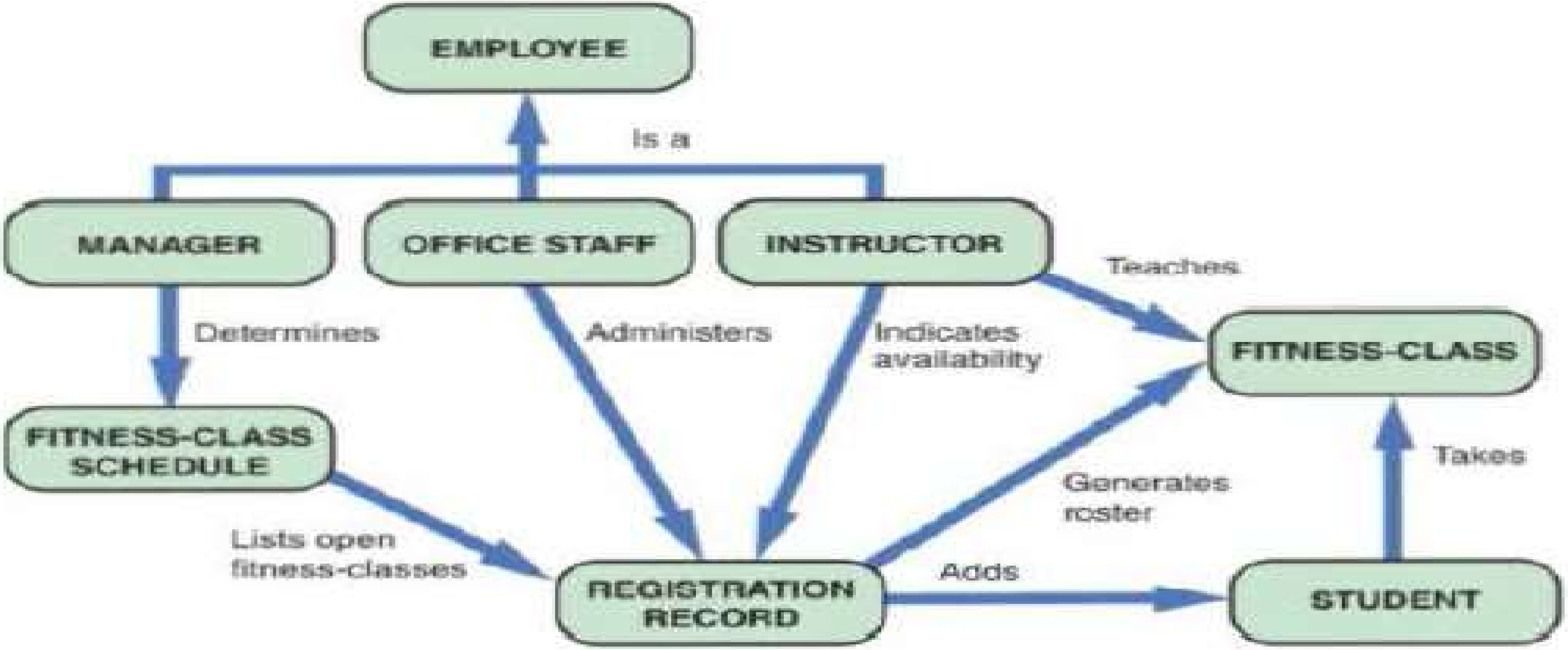
Object Relationship Diagram

- After objects, classes, and relationships have been identified, an object relationship diagram can be prepared **to provide an overview of the system**

- **That model** is used as a **guide to continue** to develop additional **diagrams** and **documentation**

Object Relationship Diagram (2)

- An object relationship diagram for a **fitness center**
- The model **shows the objects** and how they **interact** to **perform various functions**



Use of UML to Describe Object-Oriented Systems:

√Use Case Diagram

√ Class Diagram

√ Sequence Diagram

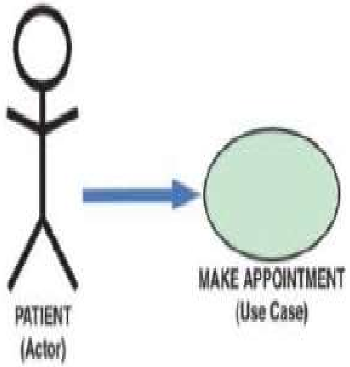
√ State Transition Diagram

√ Activity Diagram

Use Case Diagrams

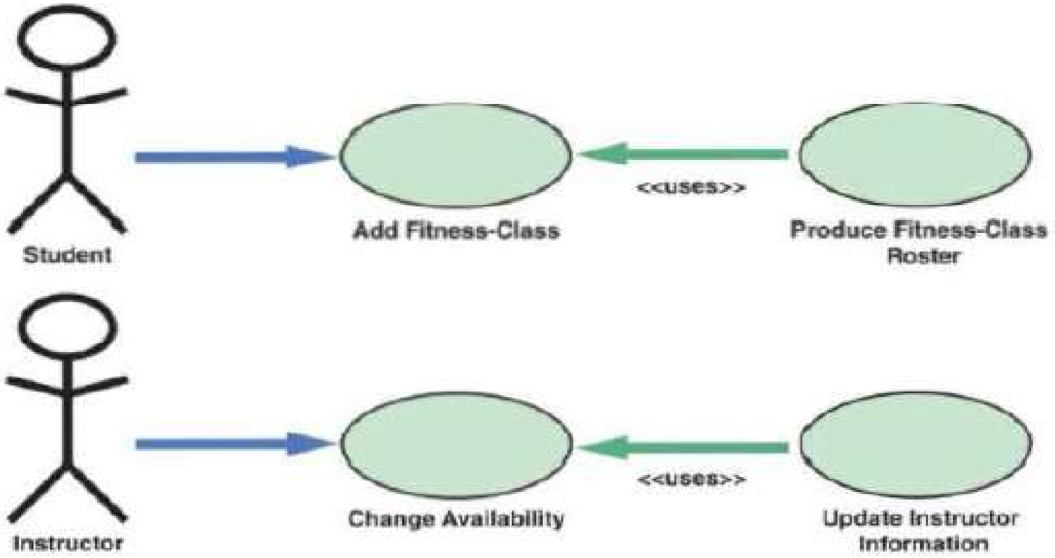
Use Case Modeling

- A use case **represents the steps** in a specific business process
- An **external entity/an actor** initiates a use case by **requesting** the system to **perform** a **process**
- The **UML symbol** for a use case
  - is an ***oval*** with a ***label*** that describes the action or event
- **The actor**
  - is shown as a ***stick figure*** with a label that ***identifies*** the ***actor's role***
- The **line** from the actor to the use case
  - is called an ***association*** because it **links** a particular **actor** to a **use case**



Use Case Modeling (2)

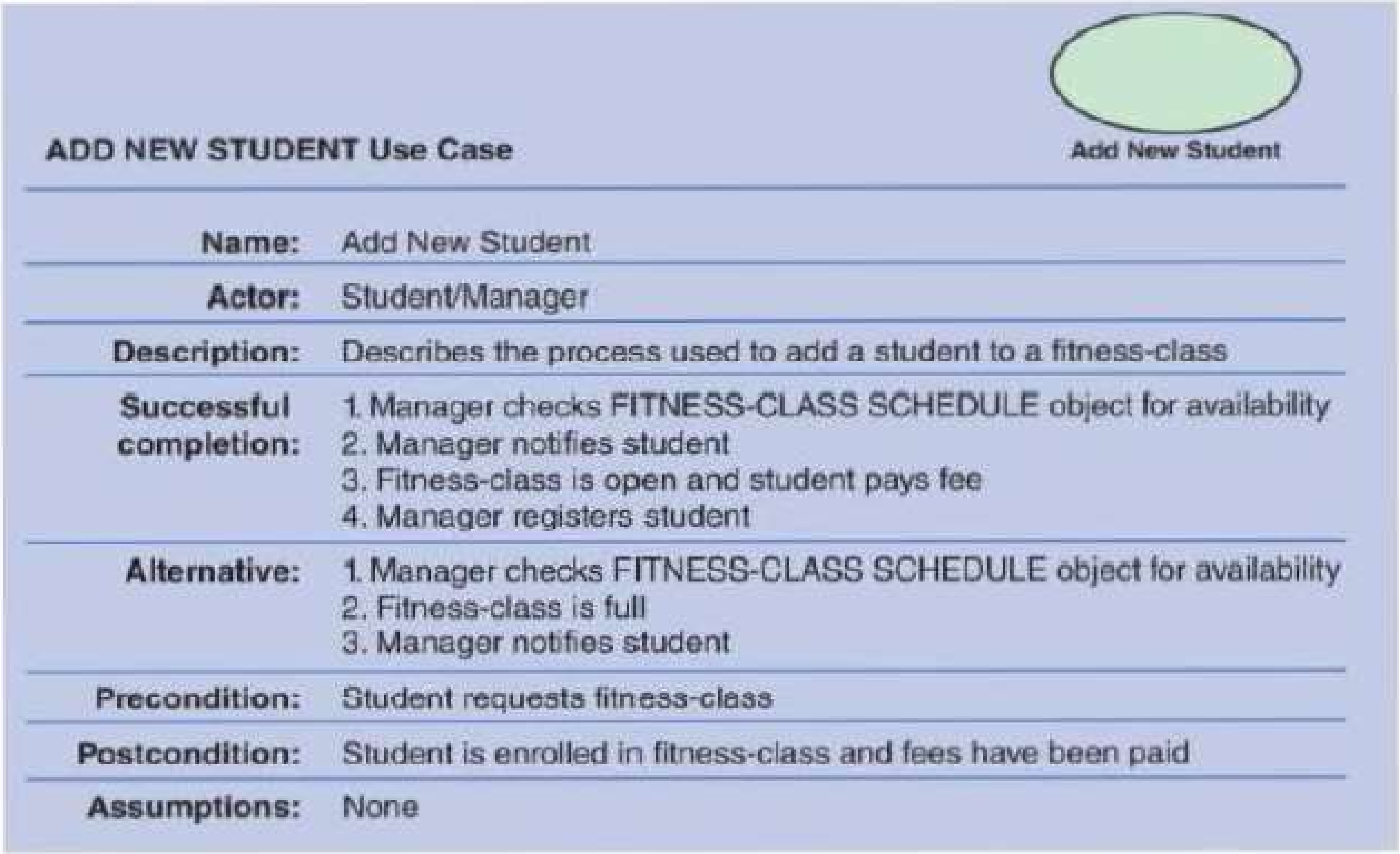
- Use cases also **can interact with other use cases**
- When the **outcome** of one use case is incorporated by another use case, we say that the second case ***uses*** the first case
  - UML indicates the relationship with an arrow that *points at* the use case being used



Use Case Modeling (3)

- To create use cases, start by **reviewing** the **information** that was gathered during the **requirements engineering phase**
- The objective is to **identify the actors** and the **processes** they initiate
- For each use case, **develop a use case description** in the form of a table
- **A use case description documents**
  - the ***name*** of the use case, the ***actor***, a ***description*** of the use case, a step-by-step ***list of the tasks*** and actions required for successful completion, a description of ***alternative courses of action***, ***preconditions***, ***postconditions***, and ***assumptions***

Use Case Modeling (4)



ADD NEW STUDENT Use Case

Add New Student

Name:	Add New Student
Actor:	Student/Manager
Description:	Describes the process used to add a student to a fitness-class
Successful completion:	1. Manager checks FITNESS-CLASS SCHEDULE object for availability 2. Manager notifies student 3. Fitness-class is open and student pays fee 4. Manager registers student
Alternative:	1. Manager checks FITNESS-CLASS SCHEDULE object for availability 2. Fitness-class is full 3. Manager notifies student
Precondition:	Student requests fitness-class
Postcondition:	Student is enrolled in fitness-class and fees have been paid
Assumptions:	None

Use Case Diagrams (5)

- A **use case diagram** is a **visual summary** of several related use cases **within a system** or **subsystem**
- When a use case diagram is created, the first step is to **identify the system boundary**, which is represented by a ***rectangle***
  - The system boundary shows what is included in the system (inside the rectangle) and what is not included in the system (outside the rectangle)
- After the system boundary is identified, **use cases are placed on the diagram**, the **actors are added**, and the **relationships shown**

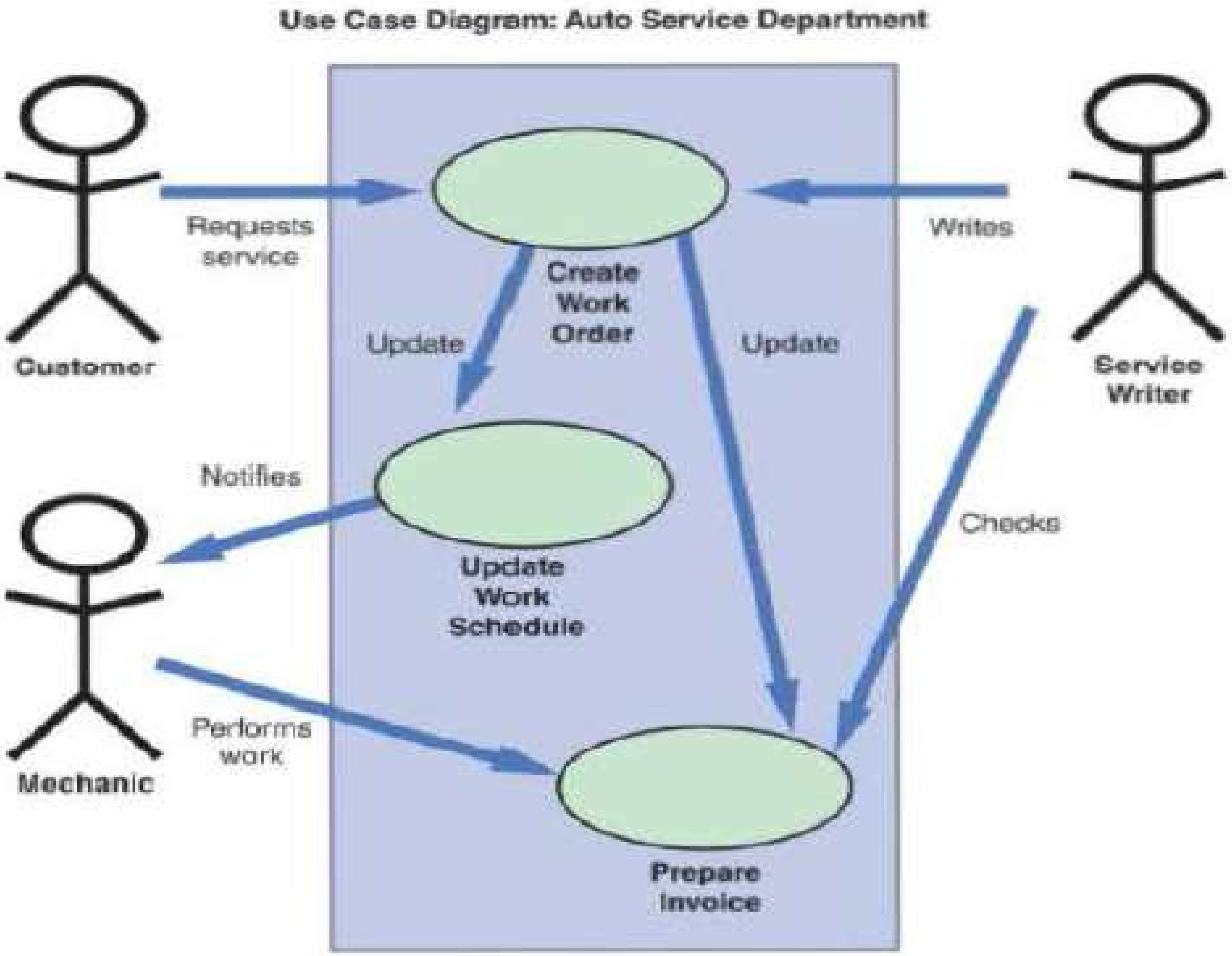
Use Case Diagram – Example

- Consider a typical Kenyan garage.

- The service department involves customers, service writers who prepare work orders and invoices, and mechanics who perform the work

- Draw a use case diagram for the garage!

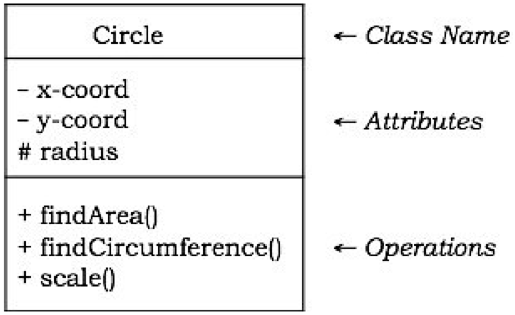
Suggested Use Case Diagram



Class Diagrams

Class Diagram

- A class diagram shows the **object classes** and **relationships** involved in a **use case**
- Class diagrams **evolve** into code modules, data objects, and other system components
- In a class diagram, **each class** appears as a **rectangle**, with the **class name** at the **top**, followed by the **class's attributes** and **methods**
- **Lines** show **relationships** between classes and have **labels identifying** the **action** that relates the two classes

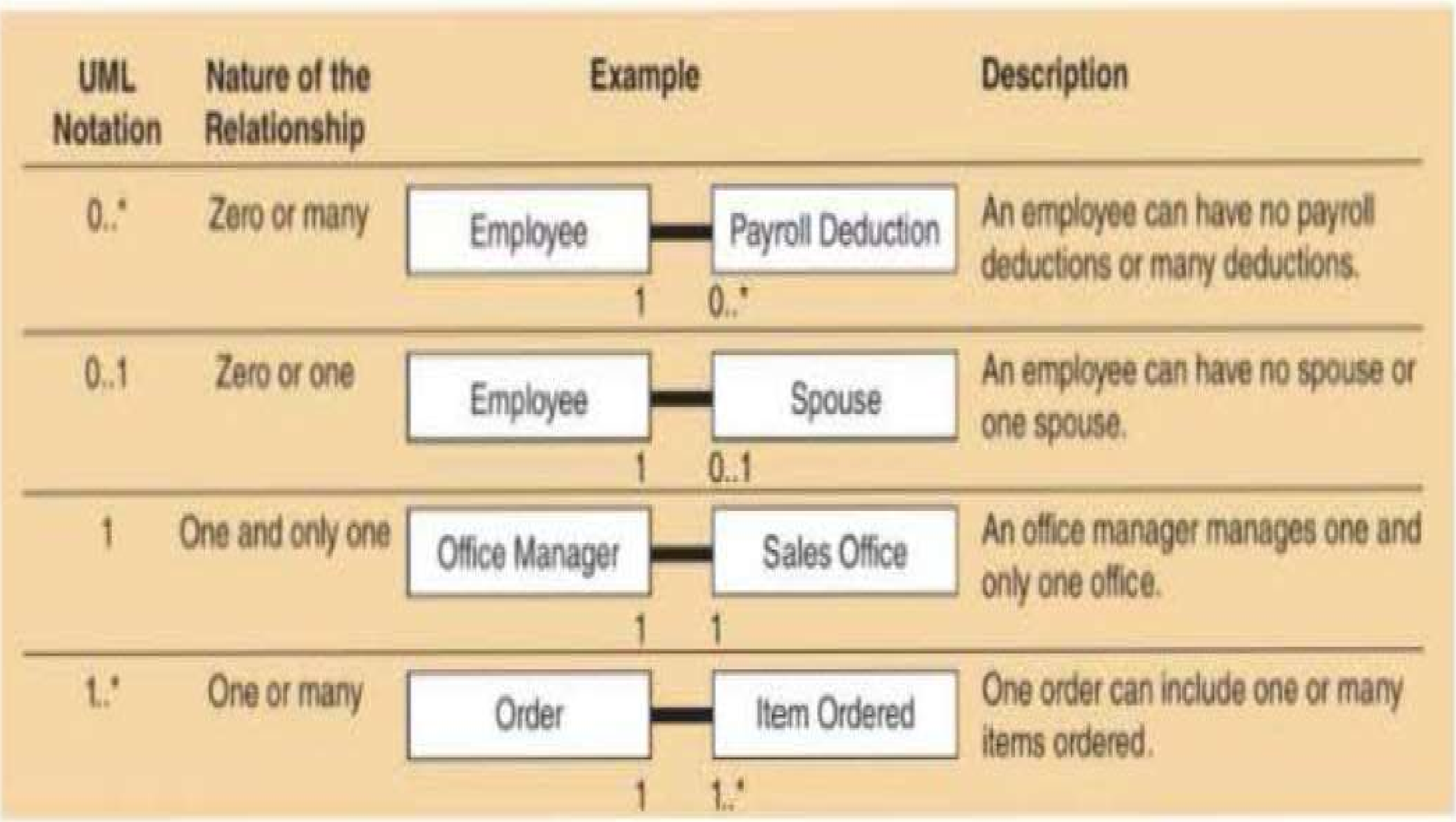


Class Diagram (2)

- The class diagram includes a concept called **cardinality**, which **describes** how **instances** of **one class** **relate** to **instances** of **another class**

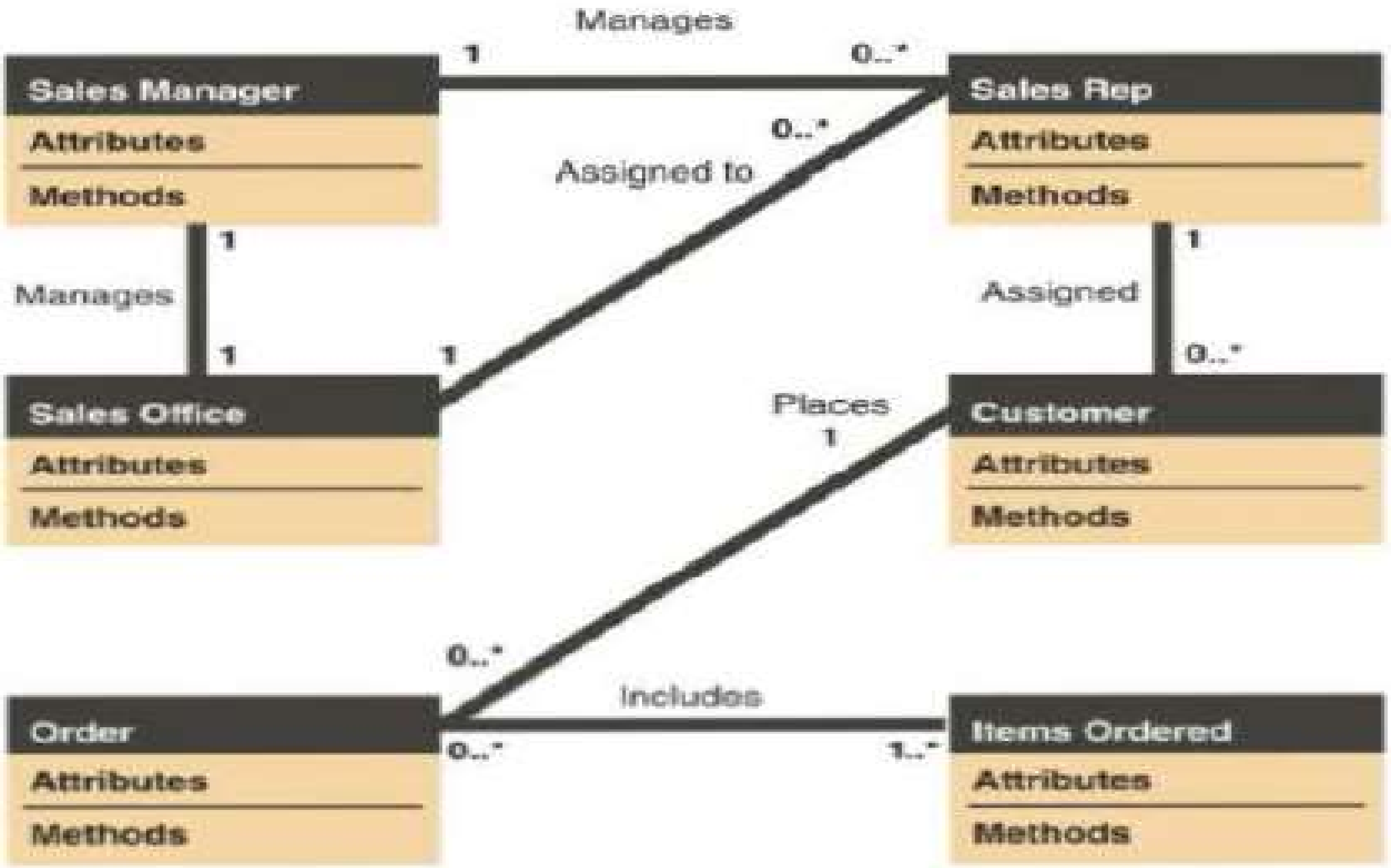
- For example, ***an employee*** might have earned ***no vocation days*** or ***one vacation day*** or ***many vacation days***

Class Diagram-Cardinality Examples



UML Notation	Nature of the Relationship	Example	Description
0..*	Zero or many	Employee Payroll Deduction 1 0..*	An employee can have no payroll deductions or many deductions.
0..1	Zero or one	Employee Spouse 1 0..1	An employee can have no spouse or one spouse.
1	One and only one	Office Manager Sales Office 1 1	An office manager manages one and only one office.
1..*	One or many	Order Item Ordered 1 1..*	One order can include one or many items ordered.

Class Diagram Example



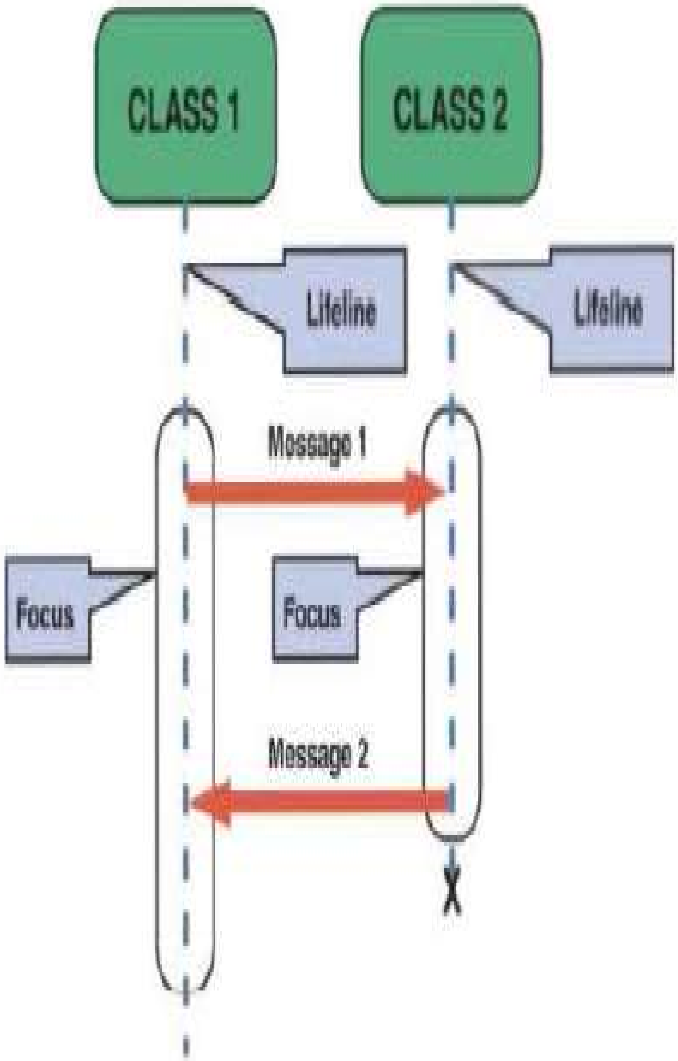
Sequence Diagram

Sequence Diagram

- A sequence diagram is a **dynamic model of a use case**, showing the **interaction among classes** during a **specified time period**
- A sequence diagram graphically **documents** the **use case** by showing the ***classes***, the ***messages***, and the ***timing*** of the ***messages***
- Sequence diagrams include **symbols** that represent **classes**, **lifelines**, **messages**, and **focuses**

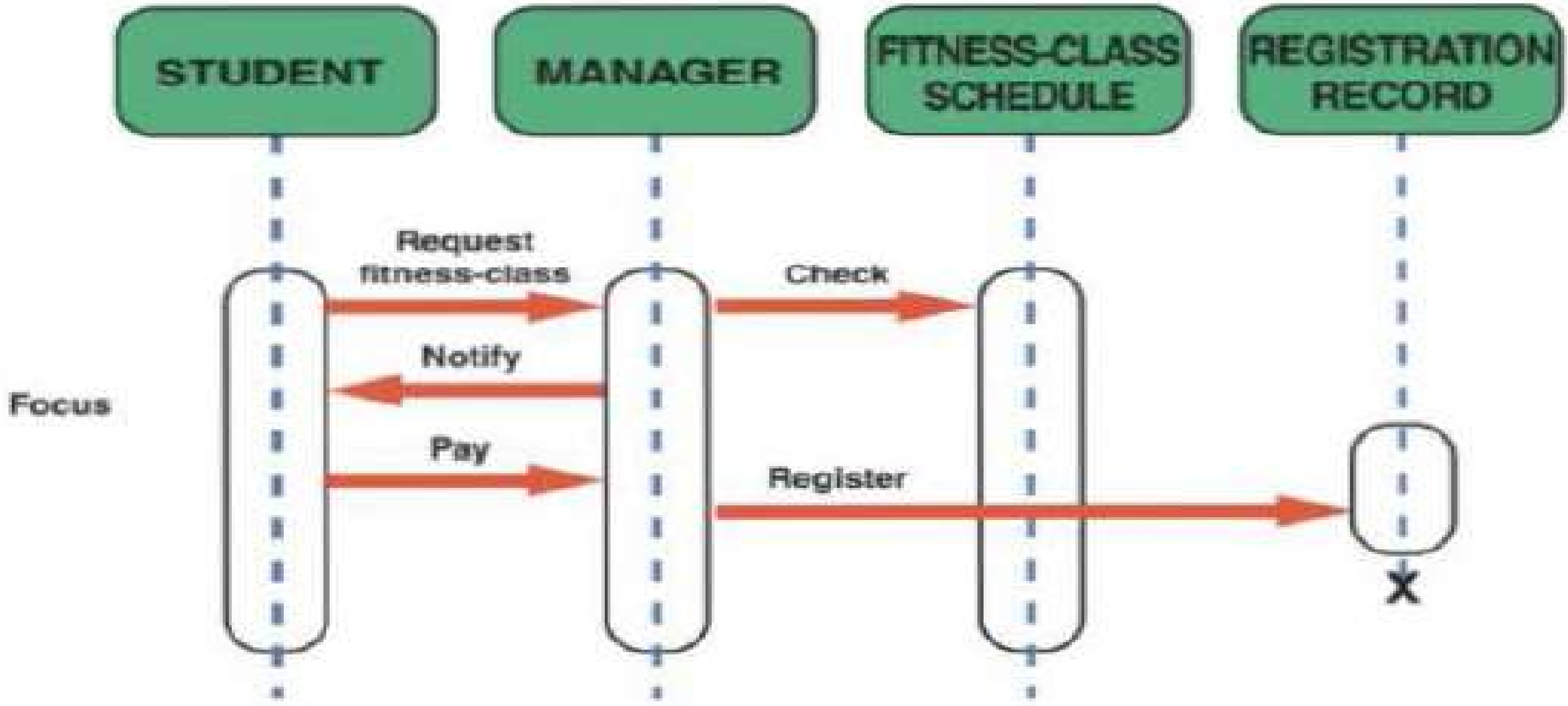
Sequence Diagram

- **A class** is identified by a **rectangle** with the **name inside**
  - Classes **send** or **receive messages**
- A **lifeline** is identified by a **dashed line**.
  - The lifeline represents the **time** the **object** above it is able to **interact** with the **other objects** in the use case
  - An **X marks** the **end** of the lifeline
- A **message** is identified by **a line showing direction** that runs between two objects
- A **focus** is identified by a **narrow vertical shape** that covers the lifeline
  - The focus indicates when an object sends or receives a message



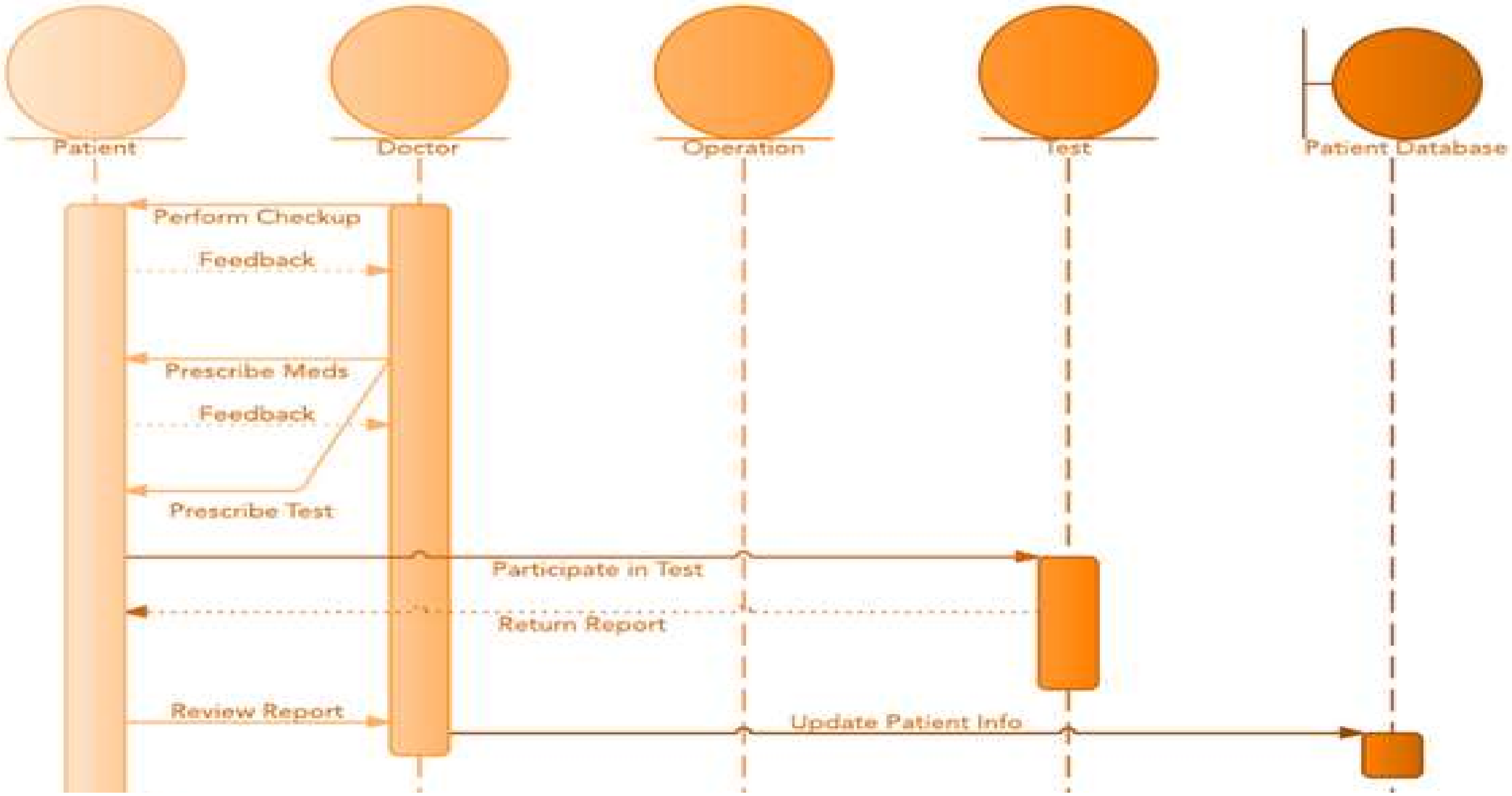
Example of a Sequence Diagram

- A sequence diagram for the **ADD NEW STUDENT** use case



Example of a Sequence Diagram

- A sequence diagram for the **UPDATE PATIENT RECORDS** use case

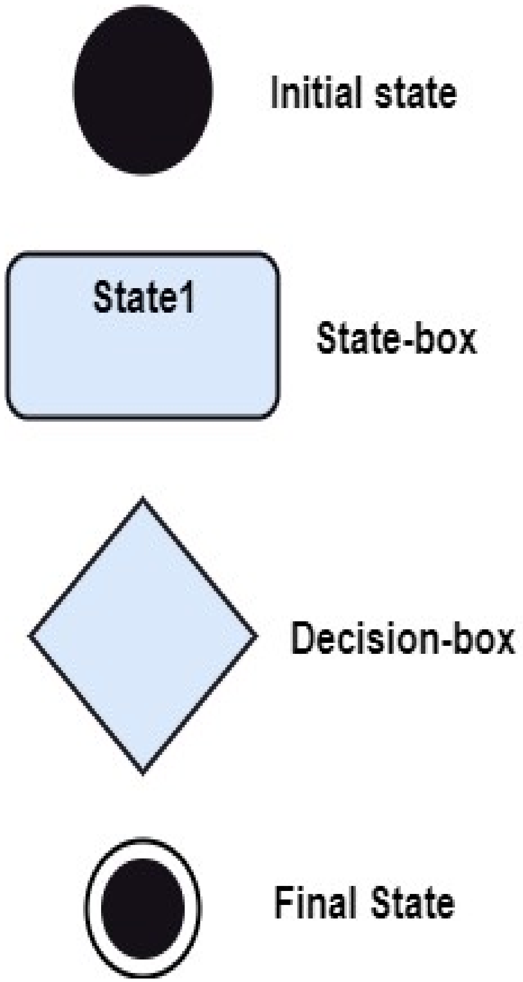


State Transition Diagram

State Transition Diagram

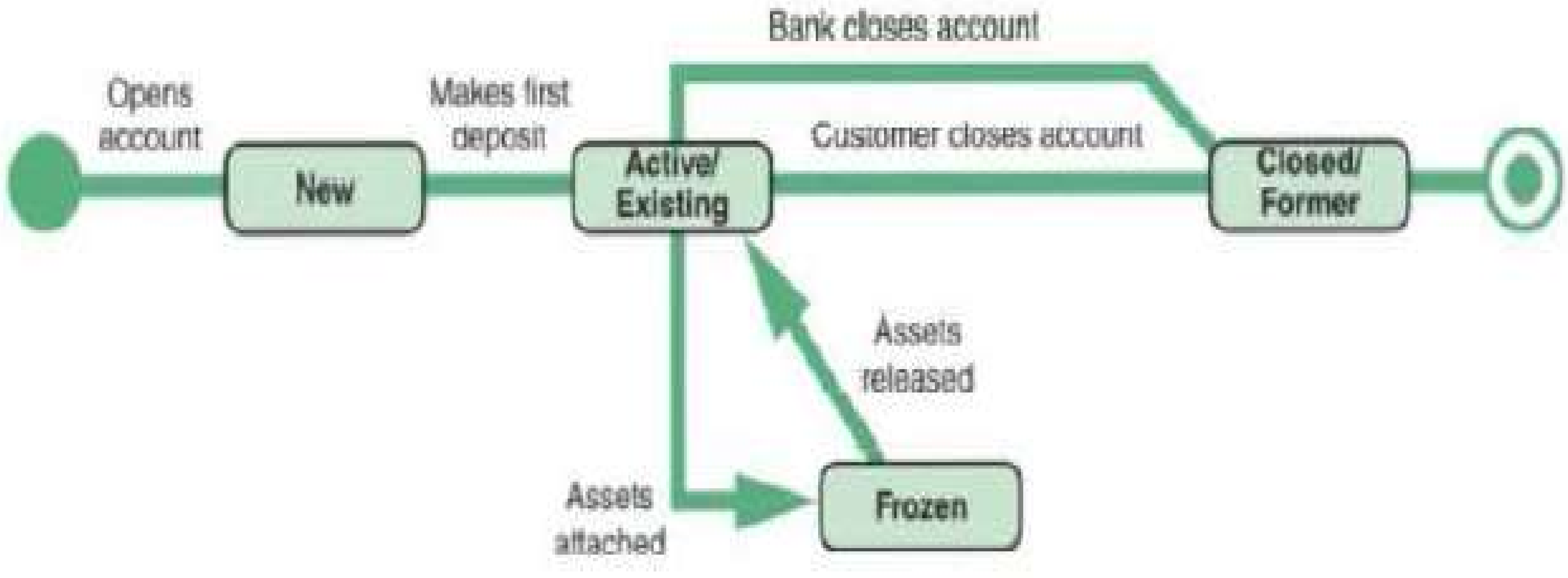
- A state transition diagram shows how an **object** changes from **one state** to **another**, **depending** on **events** that **affect** the **object**

- All **possible states** must be **documented** in the state transition diagram



State Transition Diagram – Example (2)

- For example, a bank account can change state from **NEW**, **ACTIVE**, **CLOSED** or **FROZEN**



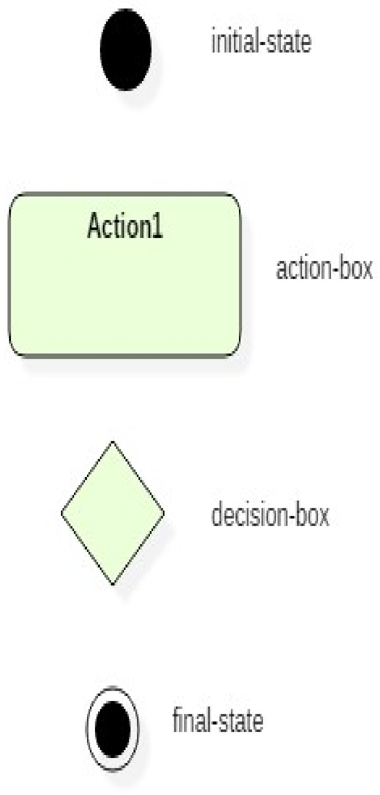
State Transition Diagram (3)

- The **states** appear as **rounded rectangles** with the state **names inside**
- The **small circle** to the **left** is the **initial state** or the point where the **object first interacts** with the system
- Reading from **left to right**, the lines show **direction** and **describe** the **action** or event that **causes** a **transition** from one state to another
- The **circle** at the **right** with a **hollow border** is the **final state**

Activity Diagram

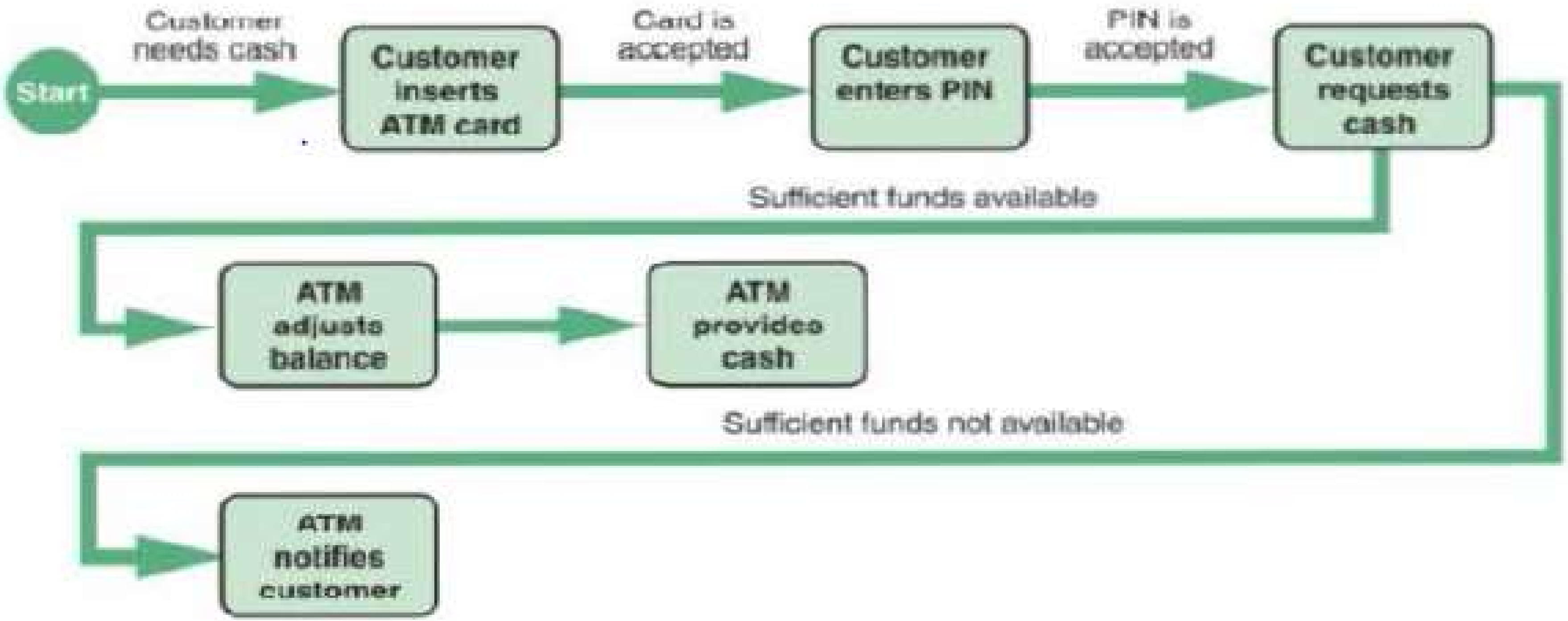
Activity Diagram

- An activity diagram **resembles a horizontal flowchart** that shows the **actions** and **events as they occur**
- Activity diagrams **show the order** in which the **actions take place** and **identify** the **outcomes**



Activity Diagram Example (2)

- An activity diagram for a **cash withdrawal** at an **ATM machine**


Start
Customer needs cash
Customer inserts ATM card
Card is accepted
Customer enters PIN
PIN is accepted
Customer requests cash
Sufficient funds available
ATM adjusts balance
ATM provides cash
Sufficient funds not available
ATM notifies customer