University of Tripoli – Faculty of Information Technology

Software Engineering Department

Software Architecture & Design

ITSE411

y Marwa Solla

Software architecture and design

for modern large-scale systems

Lecture 7:

Object-oriented design using the UML

What We Learn In This Lecture

- Class Diagram
- Object Diagram
- State Machine Diagrams
- Activity Diagram.

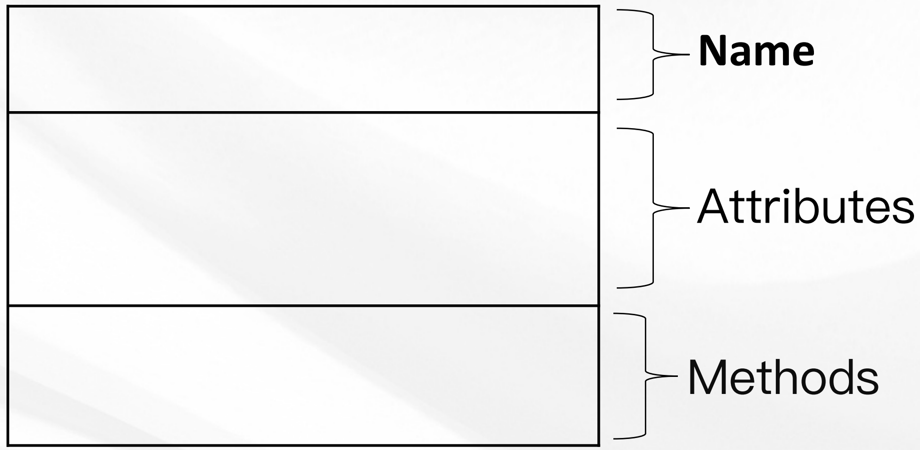
UML Class Diagram

- A **class diagram** is a static structure diagram in the Unified Modeling Language (UML) that represents the structure and behavior of a system or application through its classes, attributes, methods, and relationships.
- In class diagram, classes are depicted as boxes, and the static relationships between them are depicted as lines connecting the boxes.
- The class diagram describes the classes of applications being modeled along with their relationship.

Fundamental concepts of Class diagram

❖ Class

- Represents objects or Entities that share similar attributes, operations, relationships, and behaviors.
- can identify a class by a box of three compartment.
- Each class typically has a name and Define Attributes (variables) and Behaviors (methods).



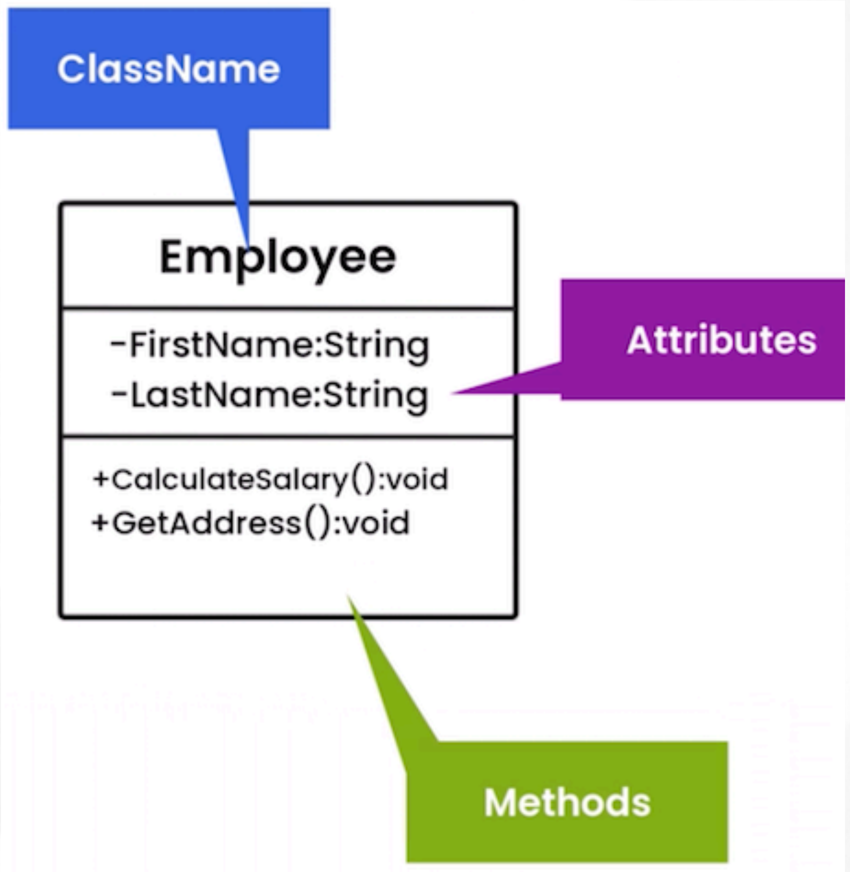
Fundamental concepts of Class diagram

Attributes:

Properties or characteristics of class. They describe the state of an object and are typically shown as variables within the class.

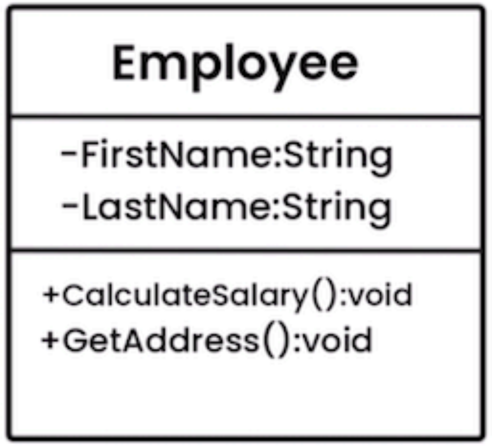
Methods:

Actions that a class can perform.



From Diagram to Code.

Class diagram represents the blueprint of the code.



Employee.cs

```
public class Employee
{
    private string FirstName { get; set; }
    private string LastName { get; set; }

     public void CalculateSalary()
     {

     }
     public void GetAddress()
     {

     }

}
```

Fundamental concepts of Class diagram

❖ Visibilities

- Define, which classes in the diagram have access to certain variables and methods.
- This concept is used in object-oriented programming.
- There are four visibilities types:

public	+	Accessible anywhere and within System
private	-	Accessible in a class that defines it.
protected	#	Accessible in a class that defines it OR Subclass
package	~	Accessible within same package

Fundamental concepts of Class diagram

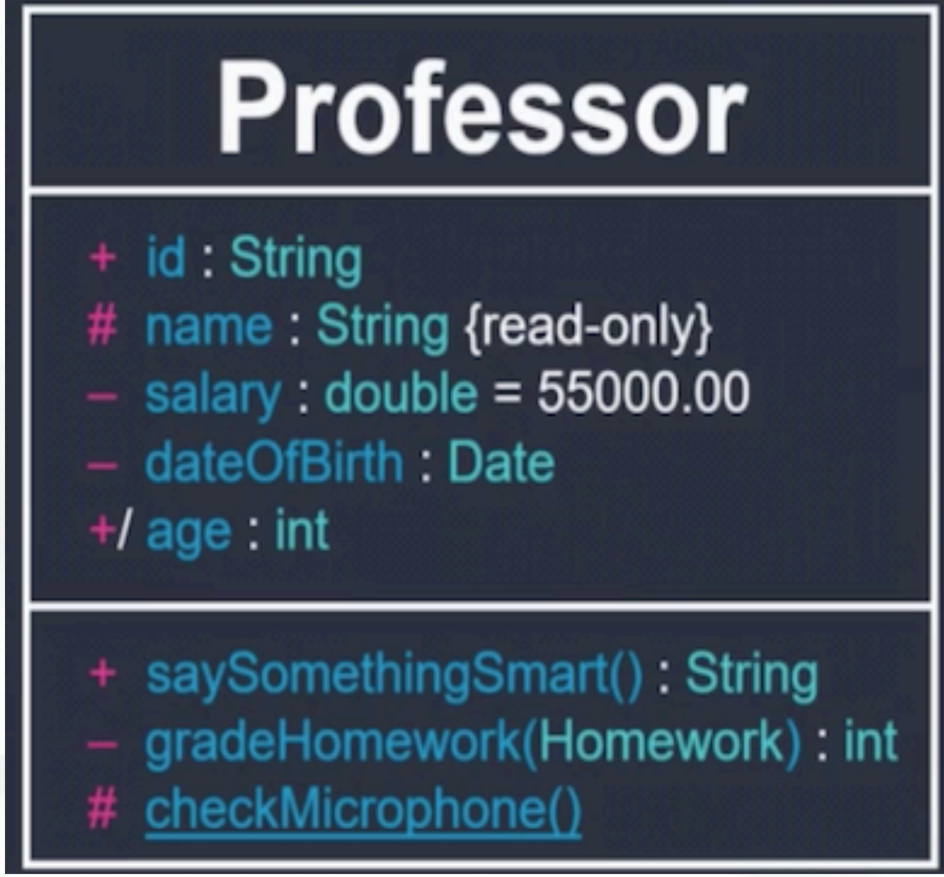
❖ Visibilities

Table 3. Visibility Options on UML Class Diagrams

Visib...ly	Symbol	Accessible to
Public	+	All objects within your system
Protected	#	Instances of the implementing class and its subclasses
Private	-	Instances of the implementing class
Package	~	Instances of classes within the same package

TIP: On detailed design models, you should always indicate the visibility of attributes and operations

Fundamental concepts of Class diagram



Attributes:
[Visibility][/] name [: type][{property}*]

Methods:
[Visibility] name [(parameter type*)] [: return type][{property}*]

Visibilities:
public (+), *private* (–), *protected* (#), *package* (~)

Class attributes/operations:
Those are underlined in the class diagram.
In programming this corresponds to the keyword *static*.

Fundamental concepts of Class diagram

❖ Relations

- Represents objects or Entities that share similar attributes, operations, relationships,
- There are 4 types of relations:

1) Associations

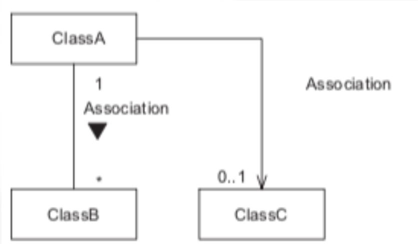
2) Aggregation

3) Composition

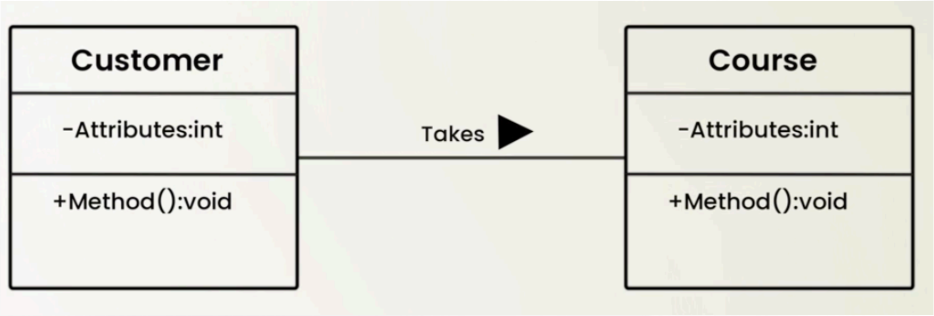
4) Inheritance

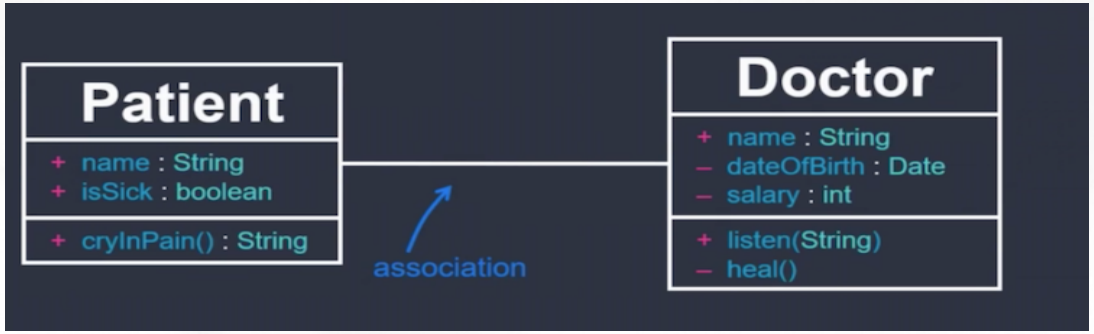
Relationships between Classes

❑ Associations:



How objects of one class interact with objects of another class





Multiplicity

❖ Multiplicity

- For each class involved in a relationship, there will always be a multiplicity.

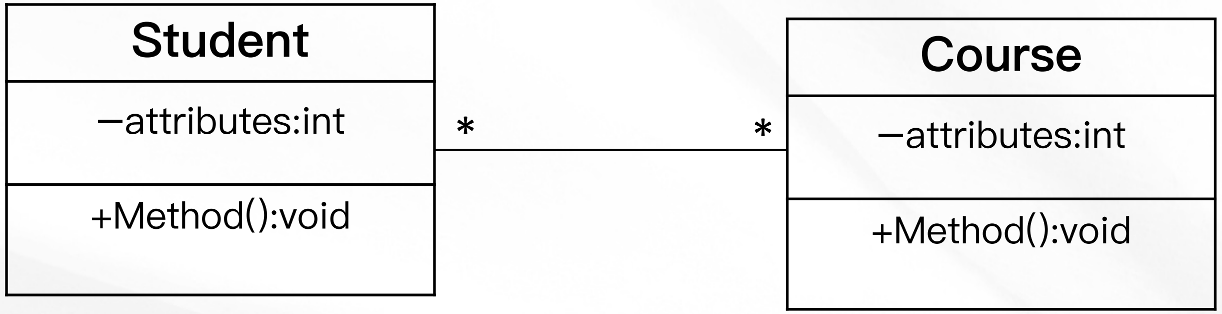
Table . UML Multiplicity Indicators

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

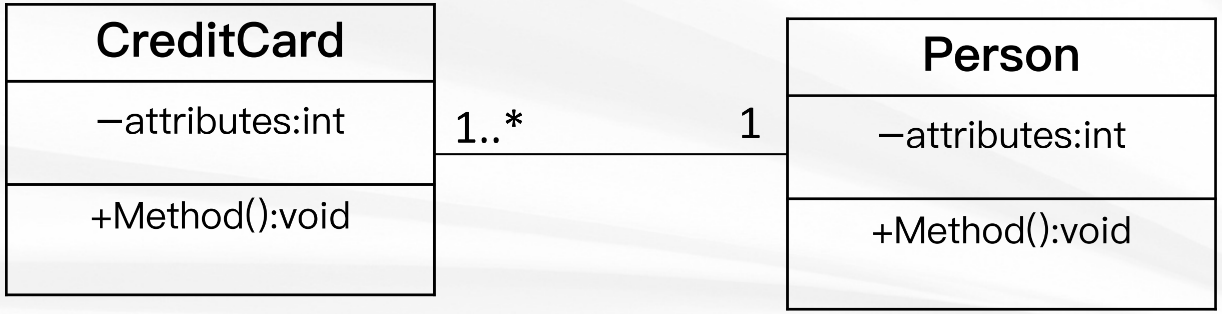
Multiplicity

❖ Multiplicity

- Many to Many(N:N)



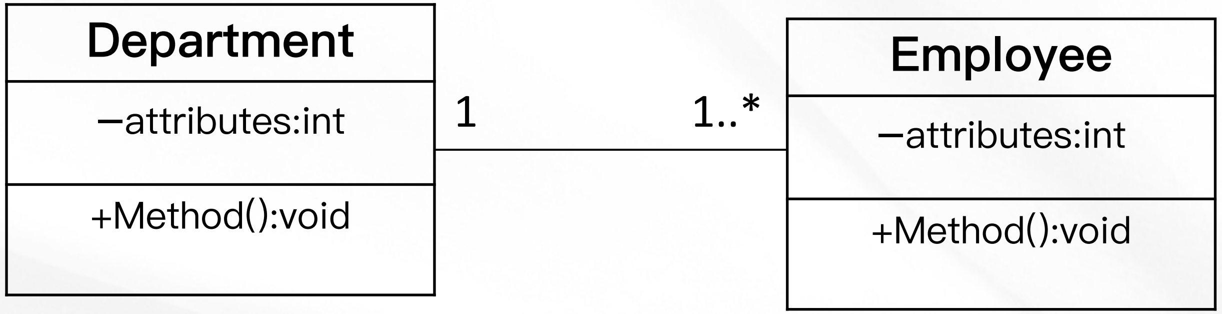
- Many to One(N:1)



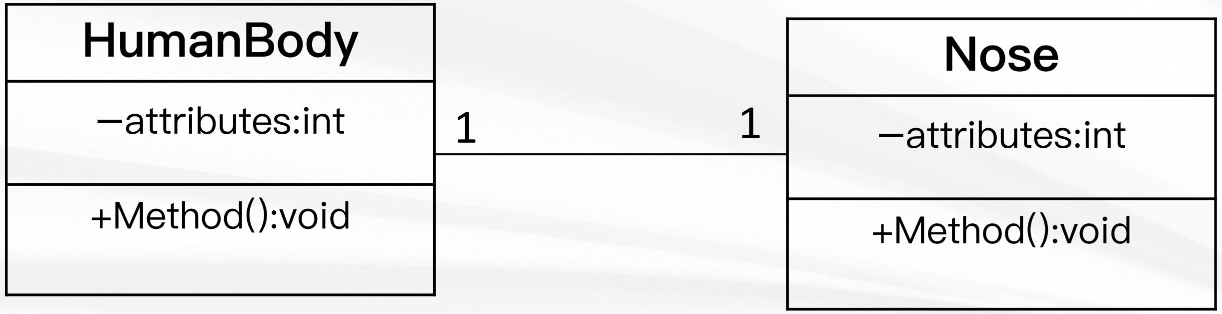
Multiplicity

❖ Multiplicity

- One to Many(1:N)



- One to One(1:1)

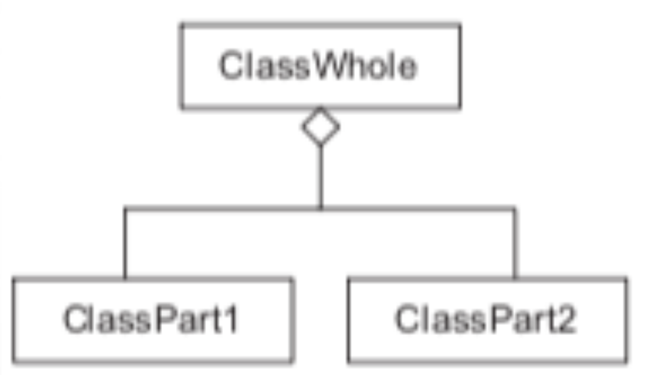


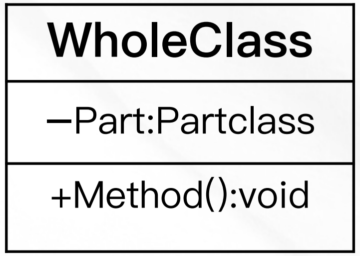
Relationships between Classes

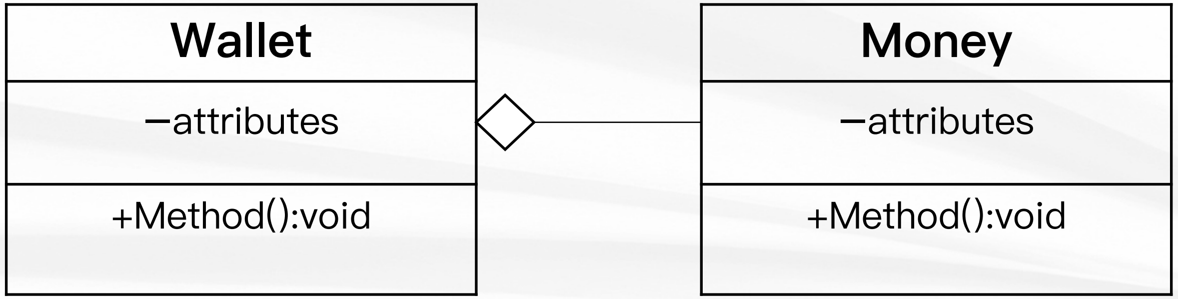
❑ Aggregation:

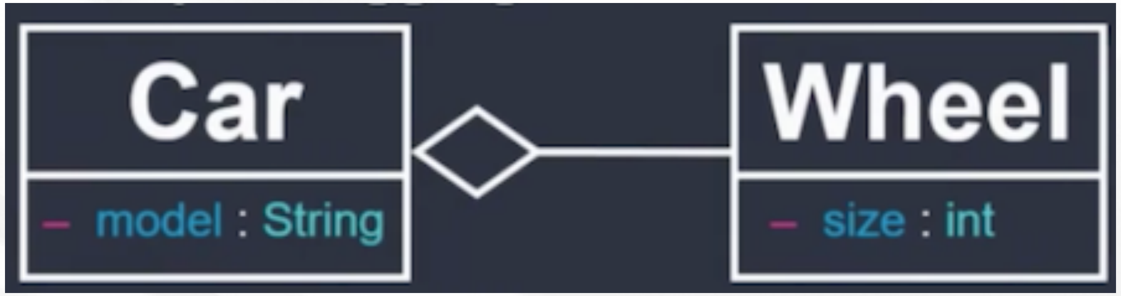
Represents a strong whole–part relationship between two classes.

Partclass can exist Independently of the Whole class.







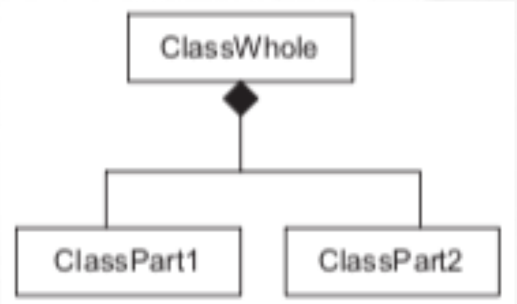


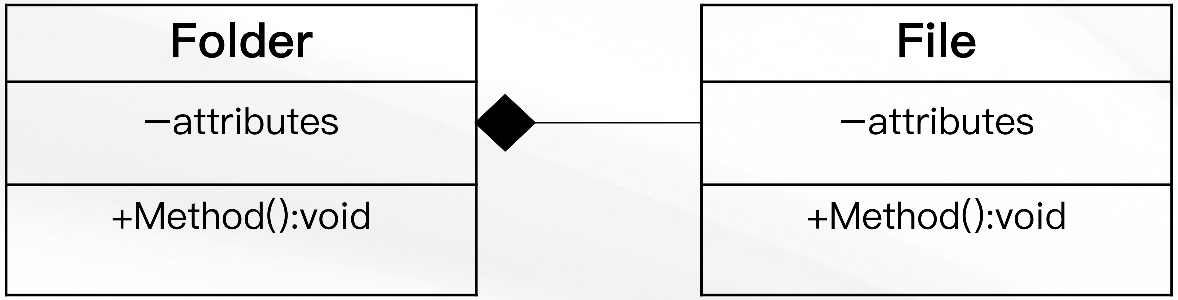
Relationships between Classes

❑ Composition:

Represents a strong whole–part relationship between two classes.

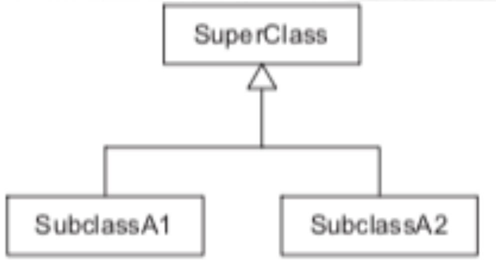
Part cannot exist Independently of the Whole



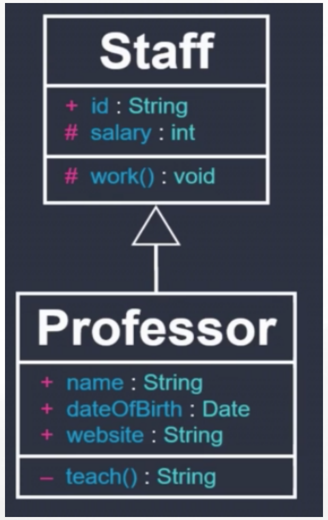


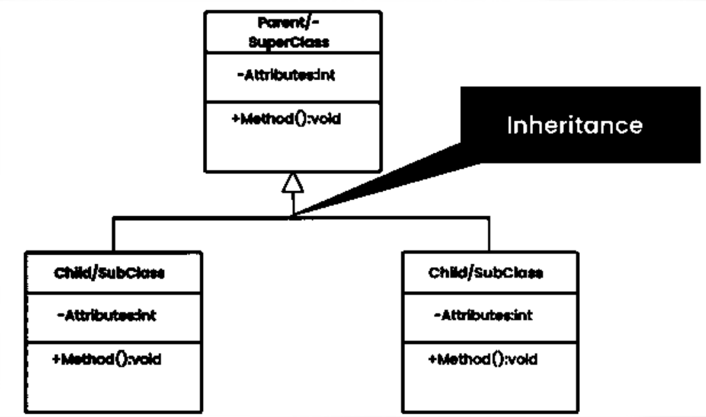
Inheritance Relationship

❑ Inheritance:



- Inheritance is also called generalization.
- Hierarchical relationship between classes, where a subclass inherits attributes and methods from its parent class
- Inheritance is represented by a solid line and the inheritance arrowhead.

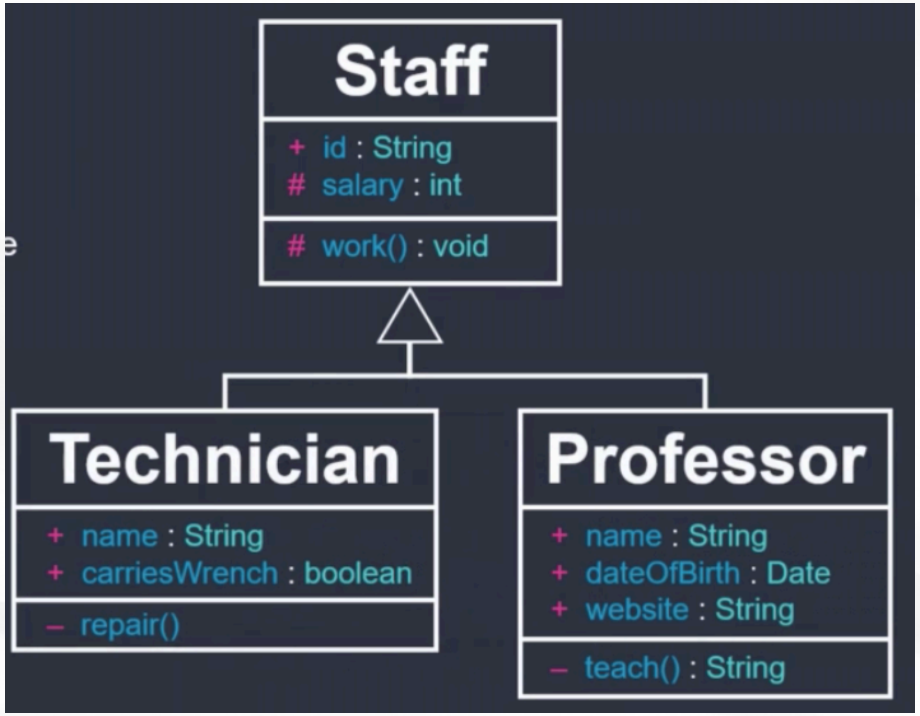


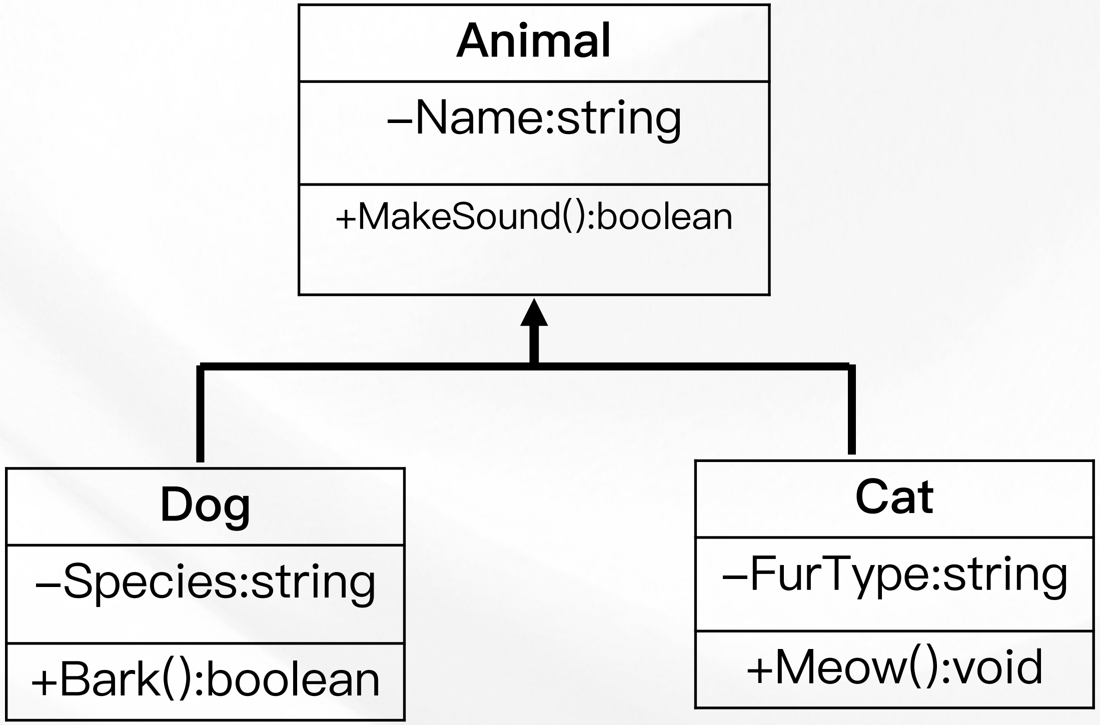


Inheritance Relationship

- Inherit from the same superclass: Multiple classes can inherit from the same superclass.

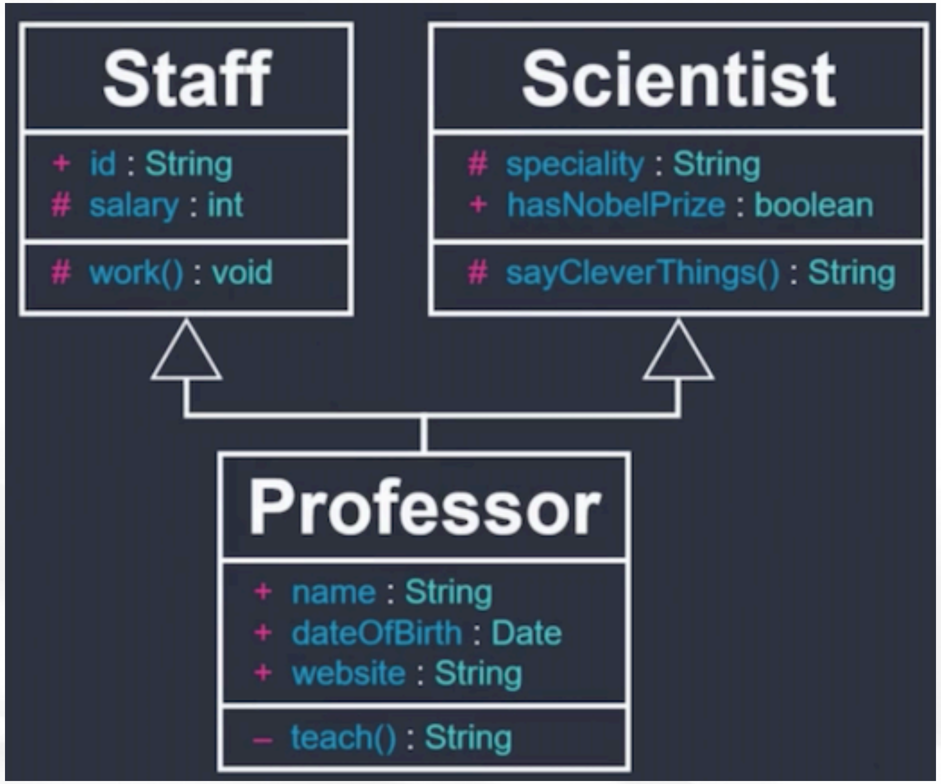
Examples:



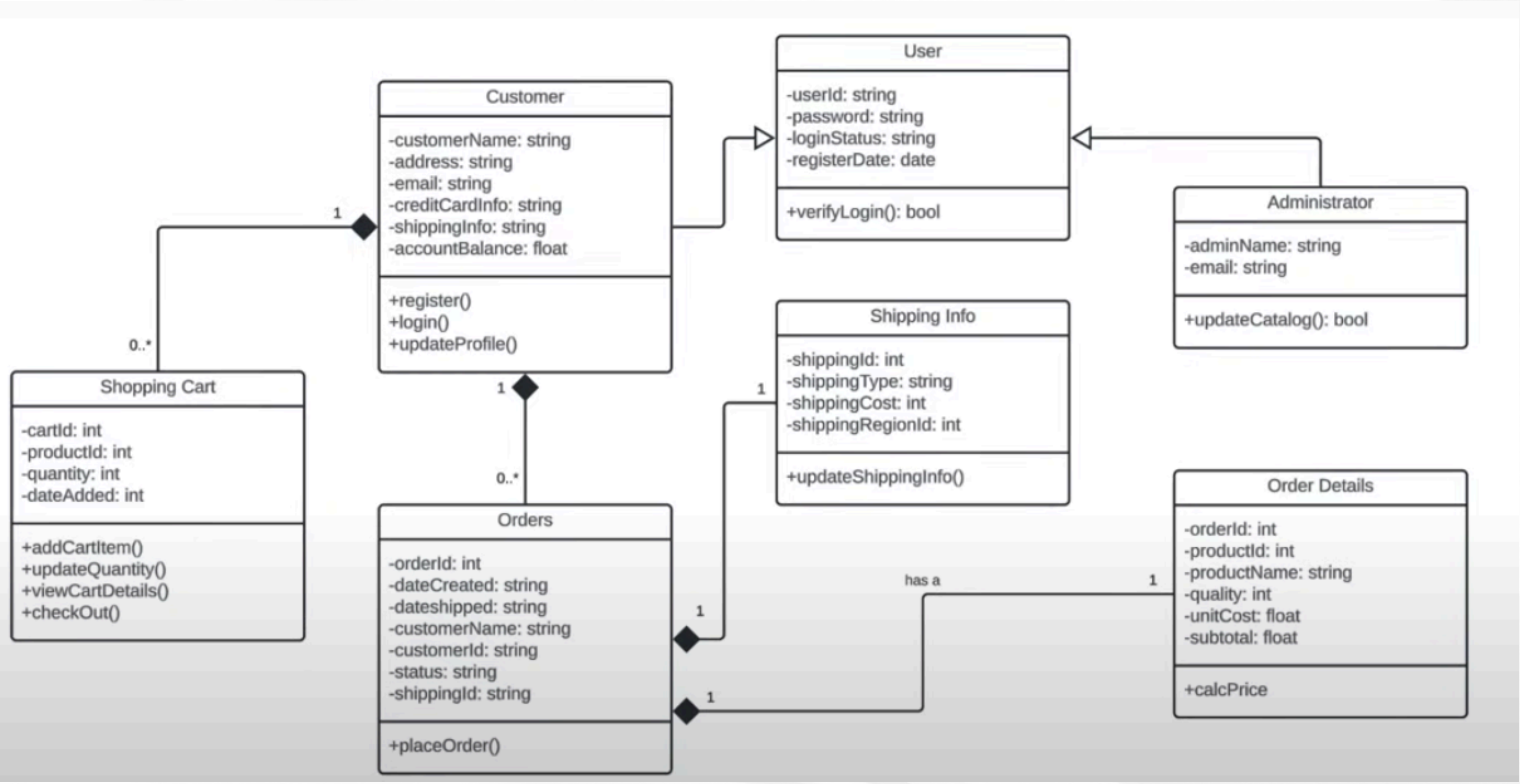


Inheritance Relationship

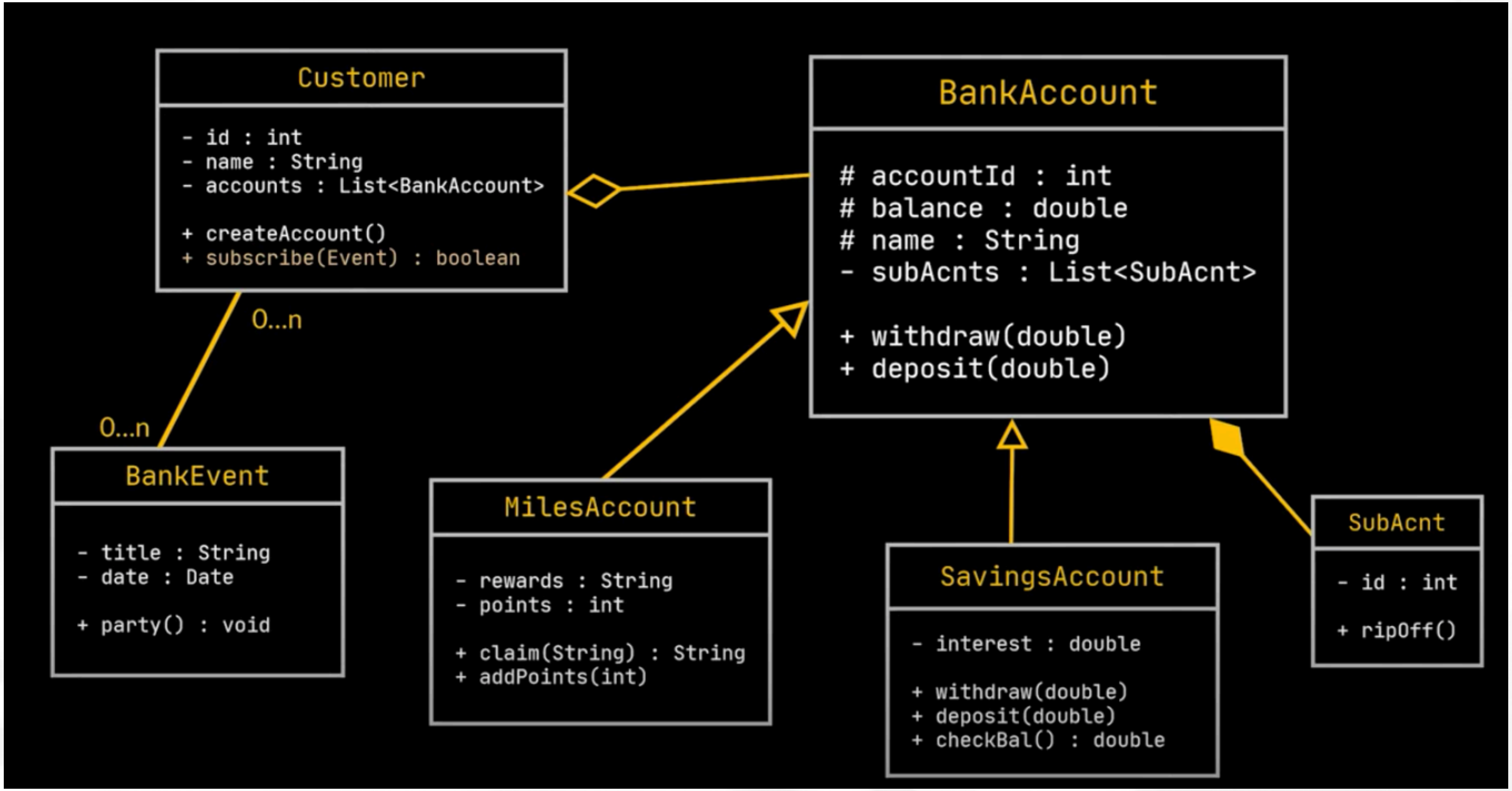
- Multiple inheritance: In UML one subclass can inherit from multiple superclasses. HOWEVER, this is not possible in all programming languages.



Example 1:


User
-userId: string
-password: string
-loginStatus: string
-registerDate: date
+verifyLogin(): bool
Customer
-customerName: string
-address: string
-email: string
-creditCardInfo: string
-shippingInfo: string
-accountBalance: float
+register()
+login()
+updateProfile()
Administrator
-adminName: string
-email: string
+updateCatalog(): bool
Shopping Cart
-cartId: int
-productId: int
-quantity: int
-dateAdded: int
+addCartItem()
+updateQuantity()
+viewCartDetails()
+checkOut()
Shipping Info
-shippingId: int
-shippingType: string
-shippingCost: int
-shippingRegionId: int
+updateShippingInfo()
Orders
-orderId: int
-dateCreated: string
-dateshipped: string
-customerName: string
-customerId: string
-status: string
-shippingId: string
+placeOrder()
Order Details
-orderId: int
-productId: int
-productName: string
-quality: int
-unitCost: float
-subtotal: float
+calcPrice
1
0..*
1
0..*
1
1
has a
1
1

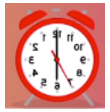
Example 2:



Customer
- id : int
- name : String
- accounts : List<BankAccount>
+ createAccount()
+ subscribe(Event) : boolean
BankAccount
accountId : int
balance : double
name : String
- subAcnts : List<SubAcnt>
+ withdraw(double)
+ deposit(double)
0...n
0...n
BankEvent
- title : String
- date : Date
+ party() : void
MilesAccount
- rewards : String
- points : int
+ claim(String) : String
+ addPoints(int)
SavingsAccount
- interest : double
+ withdraw(double)
+ deposit(double)
+ checkBal() : double
SubAcnt
- id : int
+ ripOff()

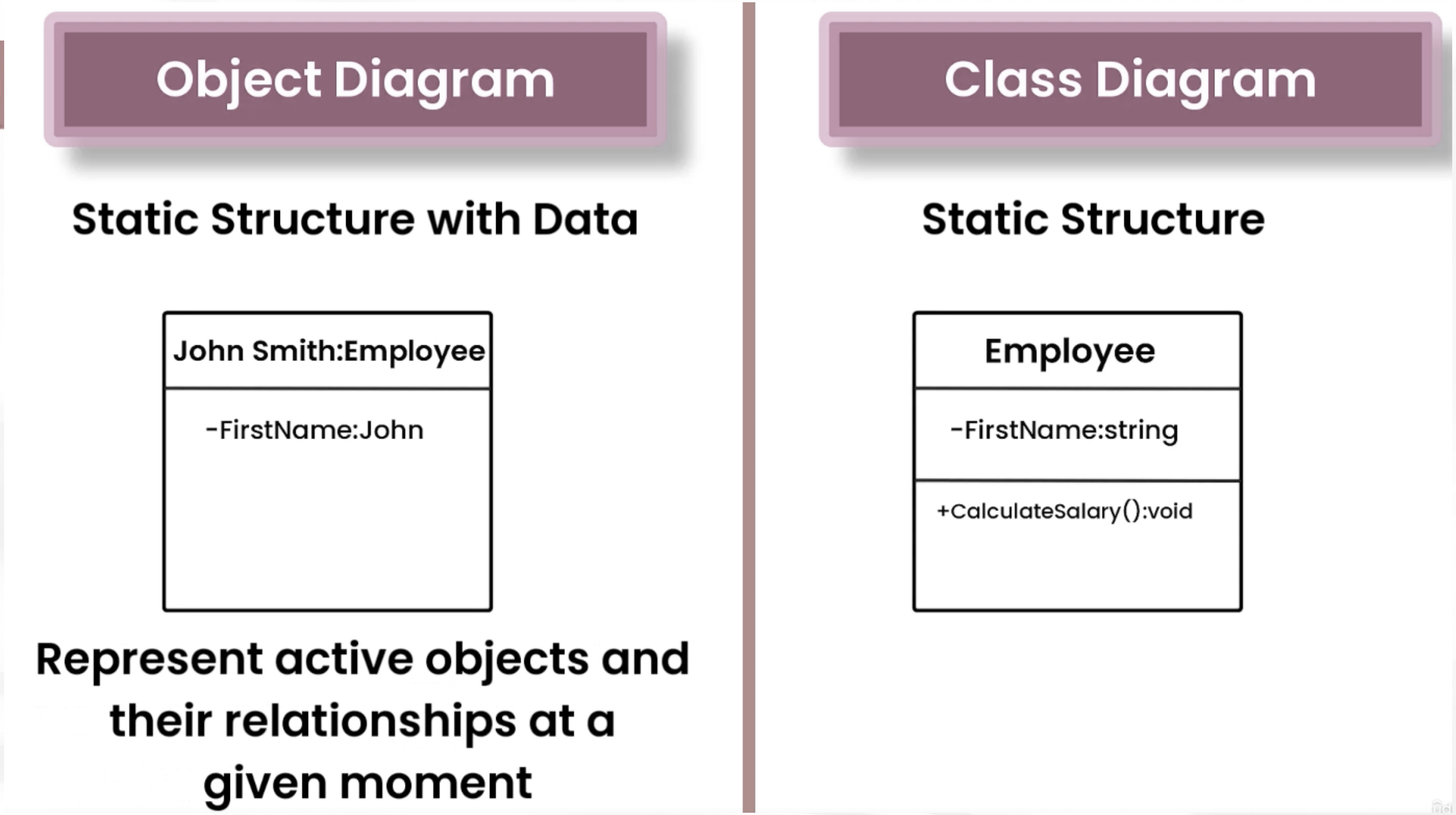
UML object Diagram

- Object diagram are closely related to class diagrams



- Snapshot of a class diagram at a particular moment in time. by representing an instance of that class diagram.
- Object diagram is also known as instance diagram.
- Explore "real–world" examples of objects and the relation– ships between them.

UML object Diagram



UML object Diagram

Object Diagram

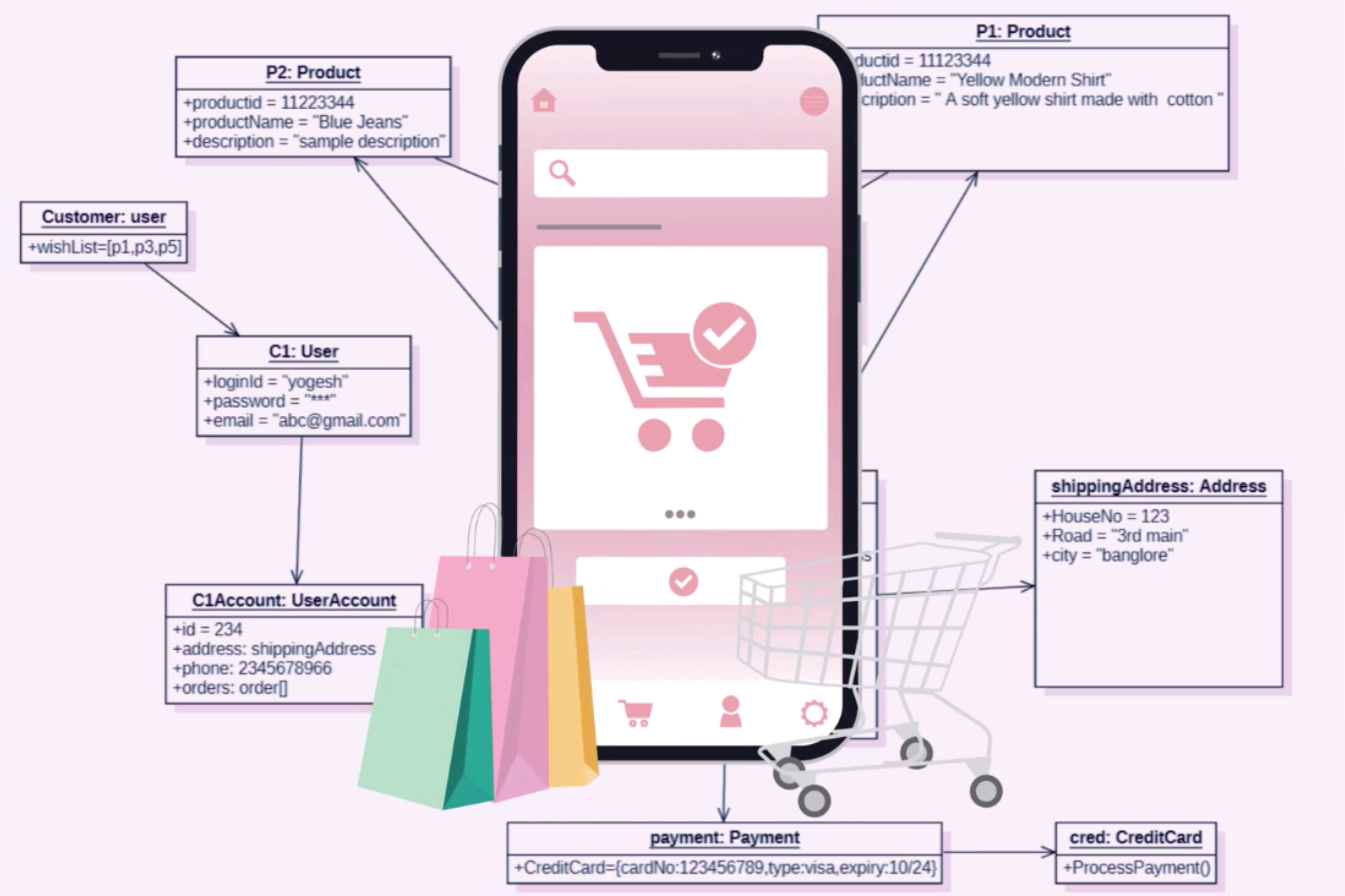
Concrete Instances of classes at a particular Moment

Class Diagram

Abstract Model consists of classes & their relationship

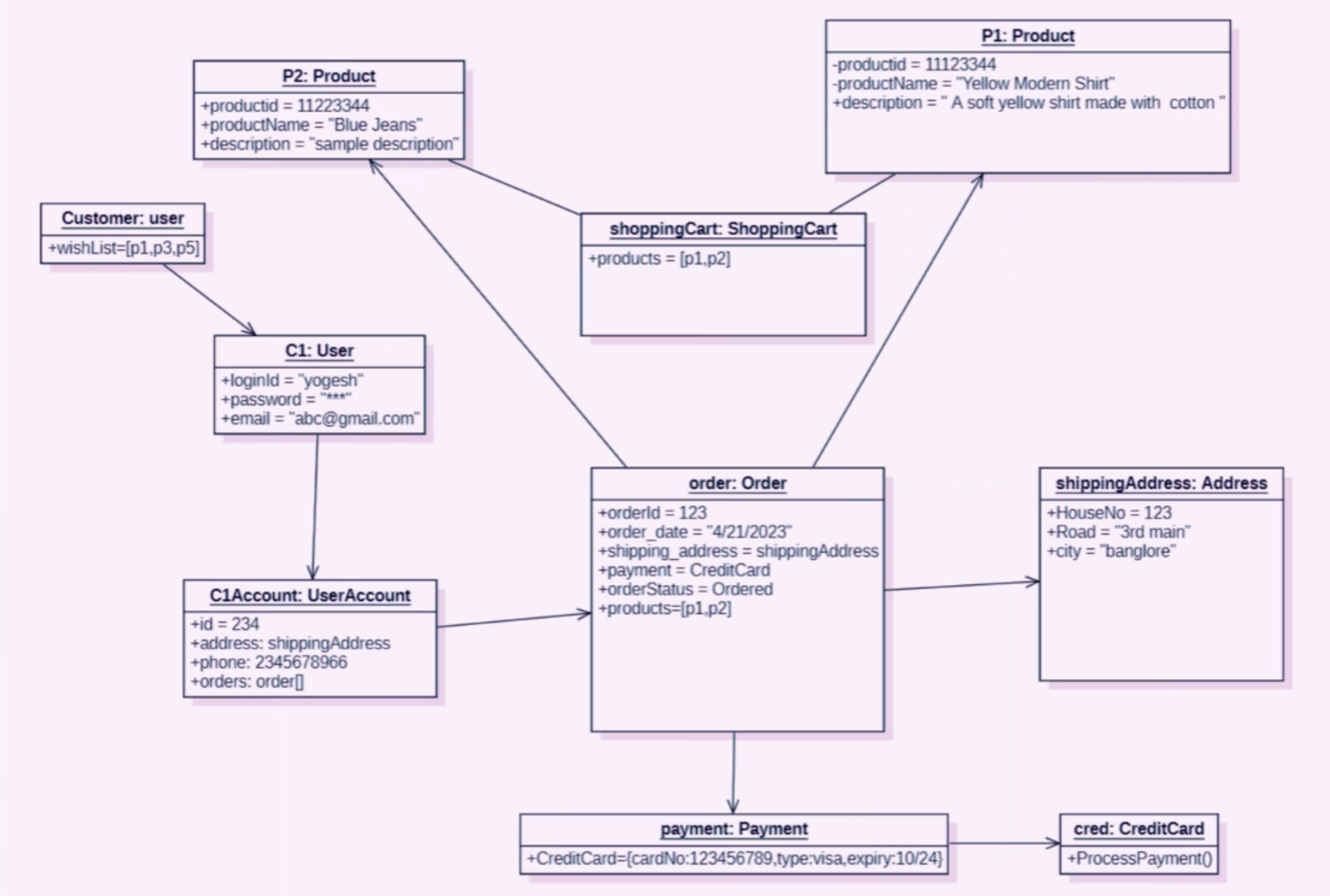
UML object Diagram

Example 1:



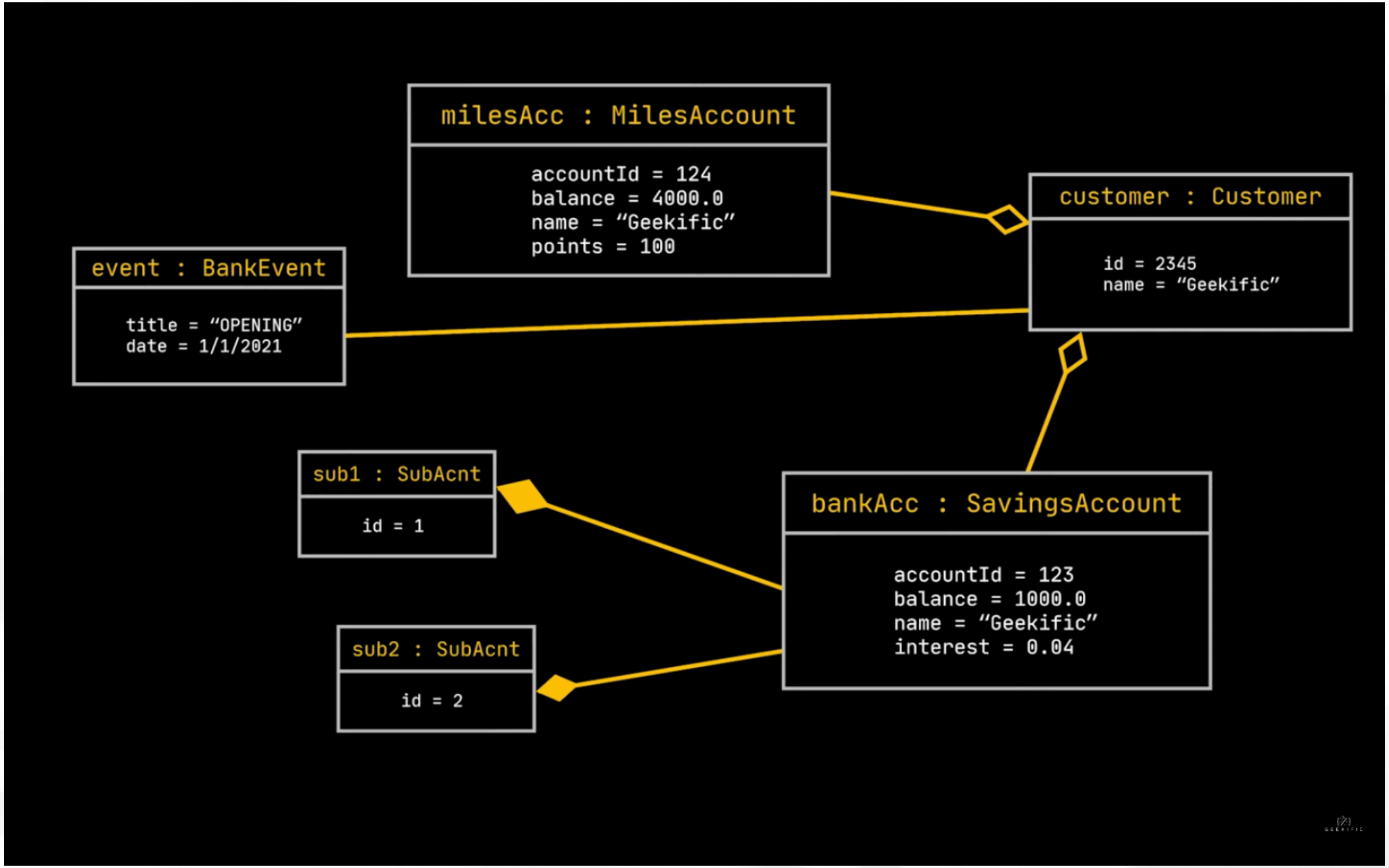
UML object Diagram

Example 1:



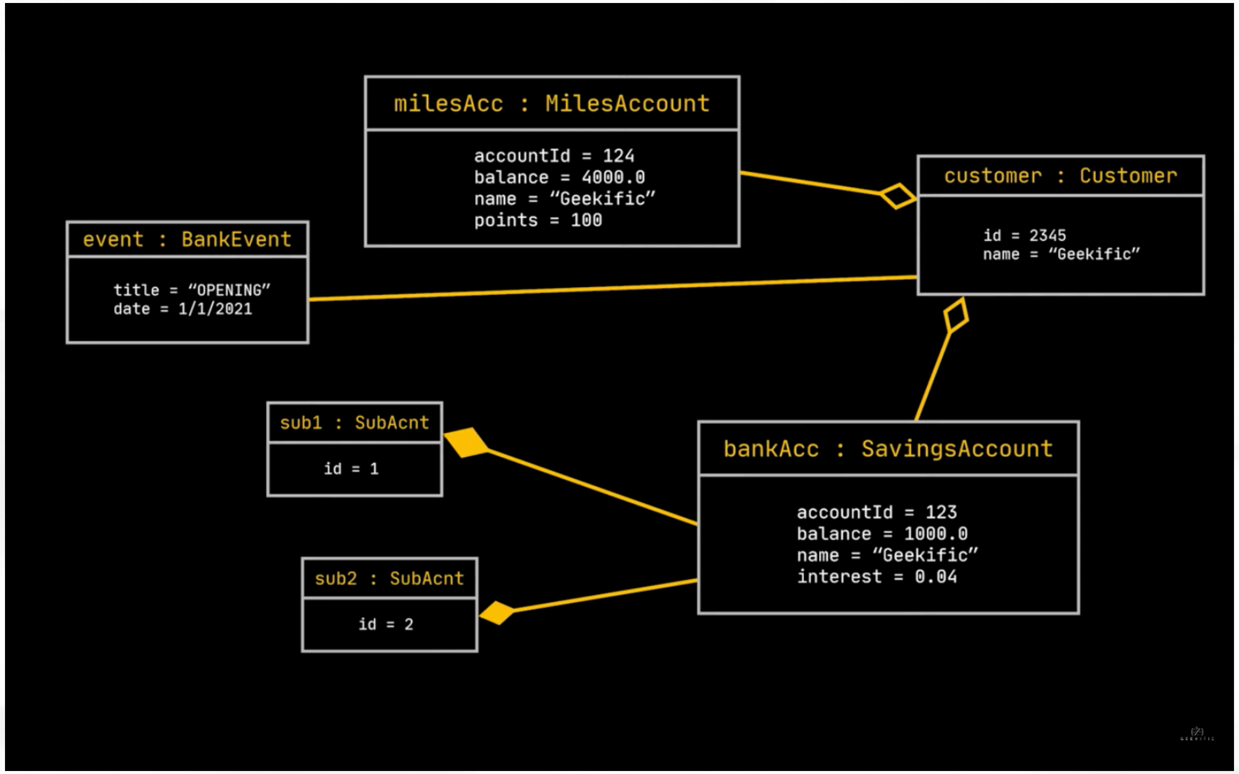
UML object Diagram

Example 2:

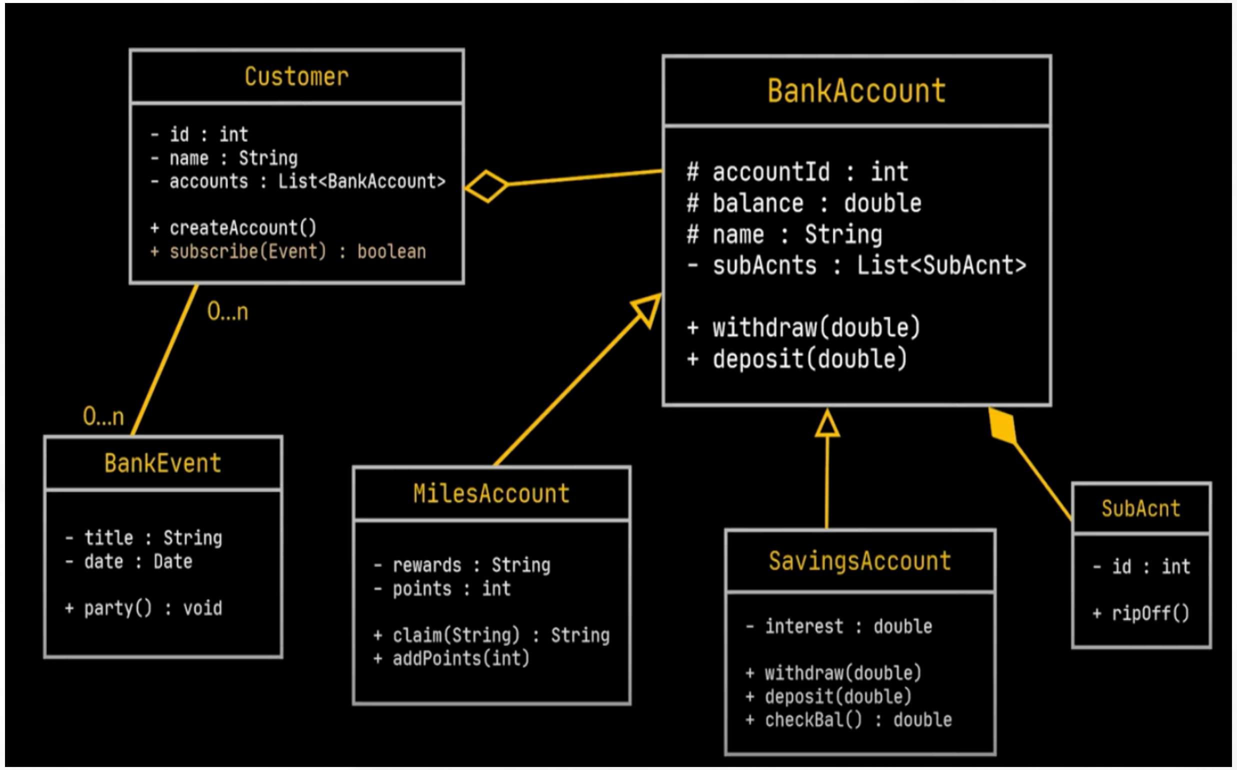


UML object Diagram

object Diagram



Class Diagram



UML State Machine Diagrams

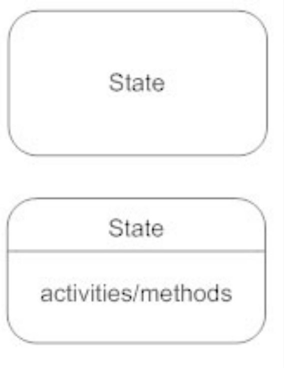
- State machine diagrams depict the dynamic behavior of an entity based on its response to events, showing how the entity reacts to various events based on its current state.
- Create a UML state machine diagram to explore the complex behavior of a class, actor, subsystem, or component.
- State Machine Diagrams, also known as state diagrams or statecharts.
- They are particularly useful for modeling the dynamic behavior of objects or systems that have a finite number of states.

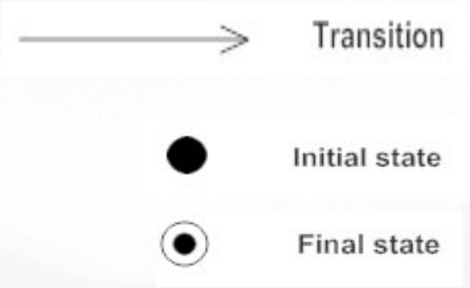
UML State Machine Diagrams

- State machine diagrams can also show how an entity reacts to various events, moving from one state to another.
- Each class has objects that may have status conditions or "states”.
- Object behavior consists of the various states and the movement between these states.
- *State Machine Diagram is a diagram which shows the life of an object in states and transitions*

UML State Machine Diagrams

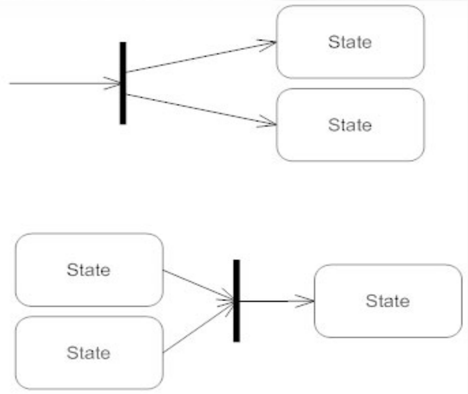
- key components and concepts of UML State Machine Diagrams:
  - State: a condition during an object's life when it satisfies some criterion, performs an action, or waits for an event.
  - Transitions: the movement of an object from one state to another
  - Origin state: the original state of an object before it begins a transition
  - Destination state: the state to which an object moves after completing a transition
  - Actions: some activity that must be completed as part of a transition.
  - guard-condition: a true/false test to see whether a transition can fire
  - Initial and Final States
  - Pseudostate: the starting point in a state machine diagram. Noted by a black circle.





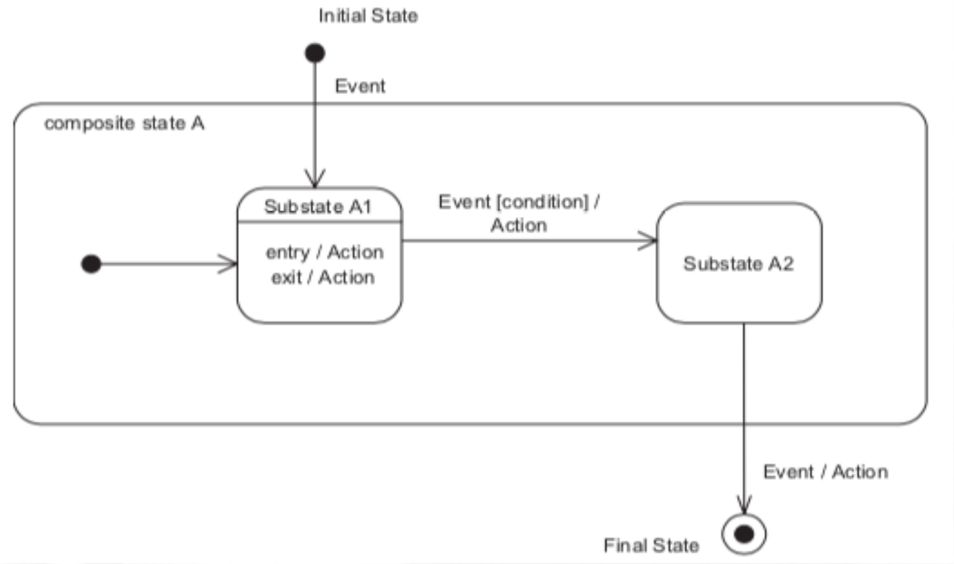
UML State Machine Diagrams

- key components and concepts of UML State Machine Diagrams:
  - Concurrent states: when an object is in one or more states at the same time
  - Concurrent paths: when multiple paths are being followed concurrently, i.e. when one or more states in one path are parallel to states in another path



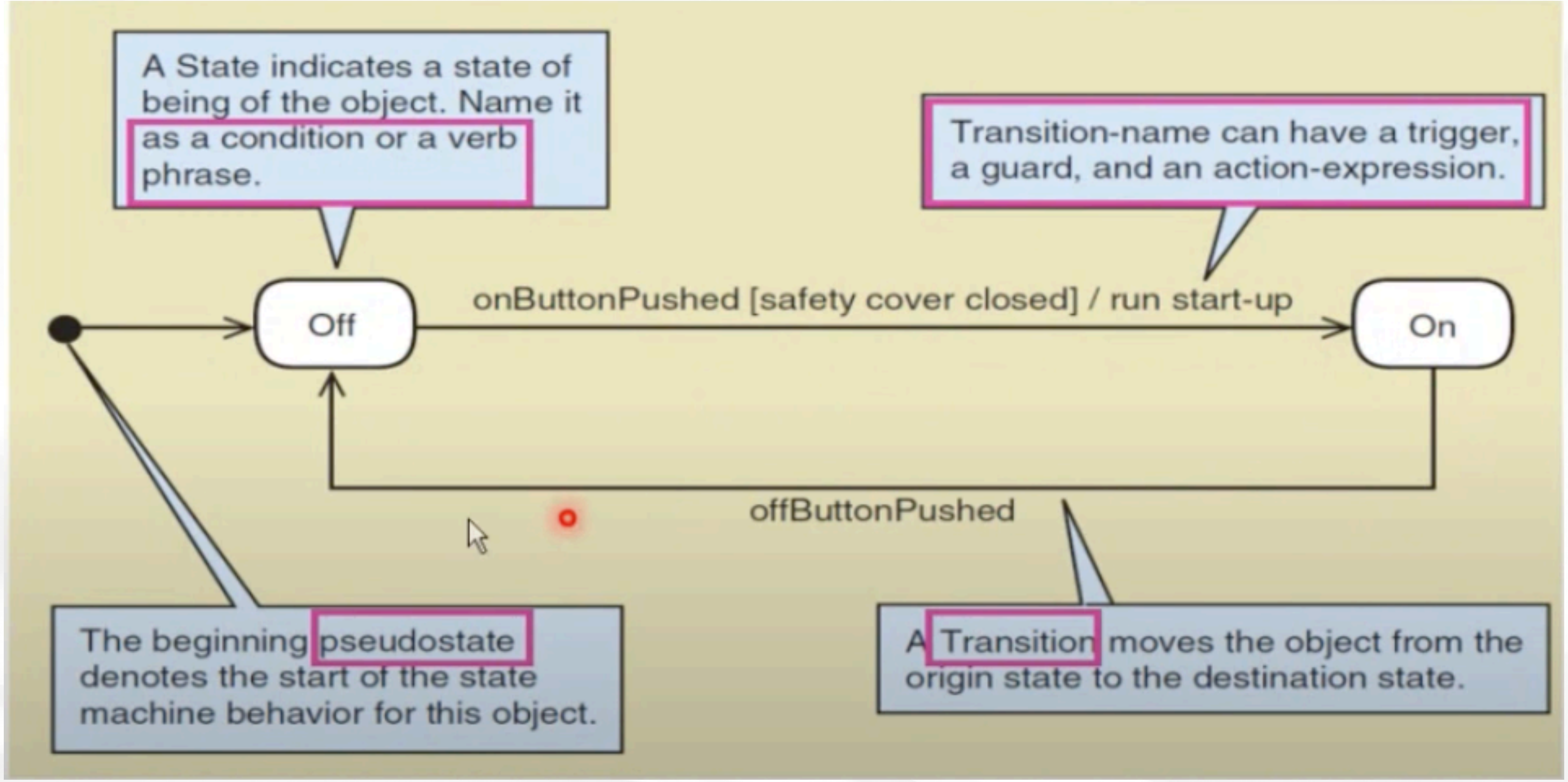
Notes:

- The **event** causes the state transition
- The optional **action** is performed as a result of the transition. Optionally, a state may have any of the following:
  - ✓ An **entry action**, performed when the state is entered
  - ✓ An **exit action**, performed on exit from the state



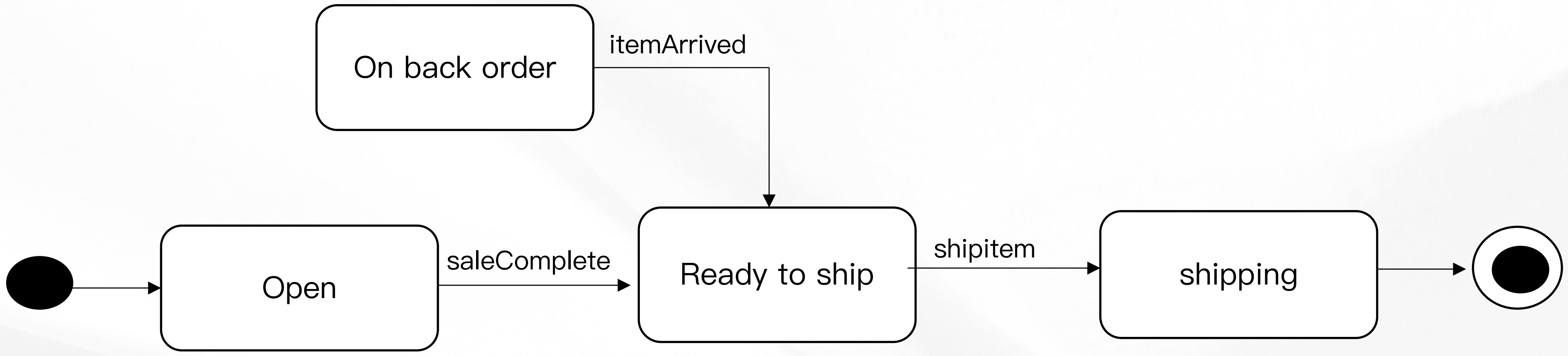
UML State Machine Diagrams

Example:



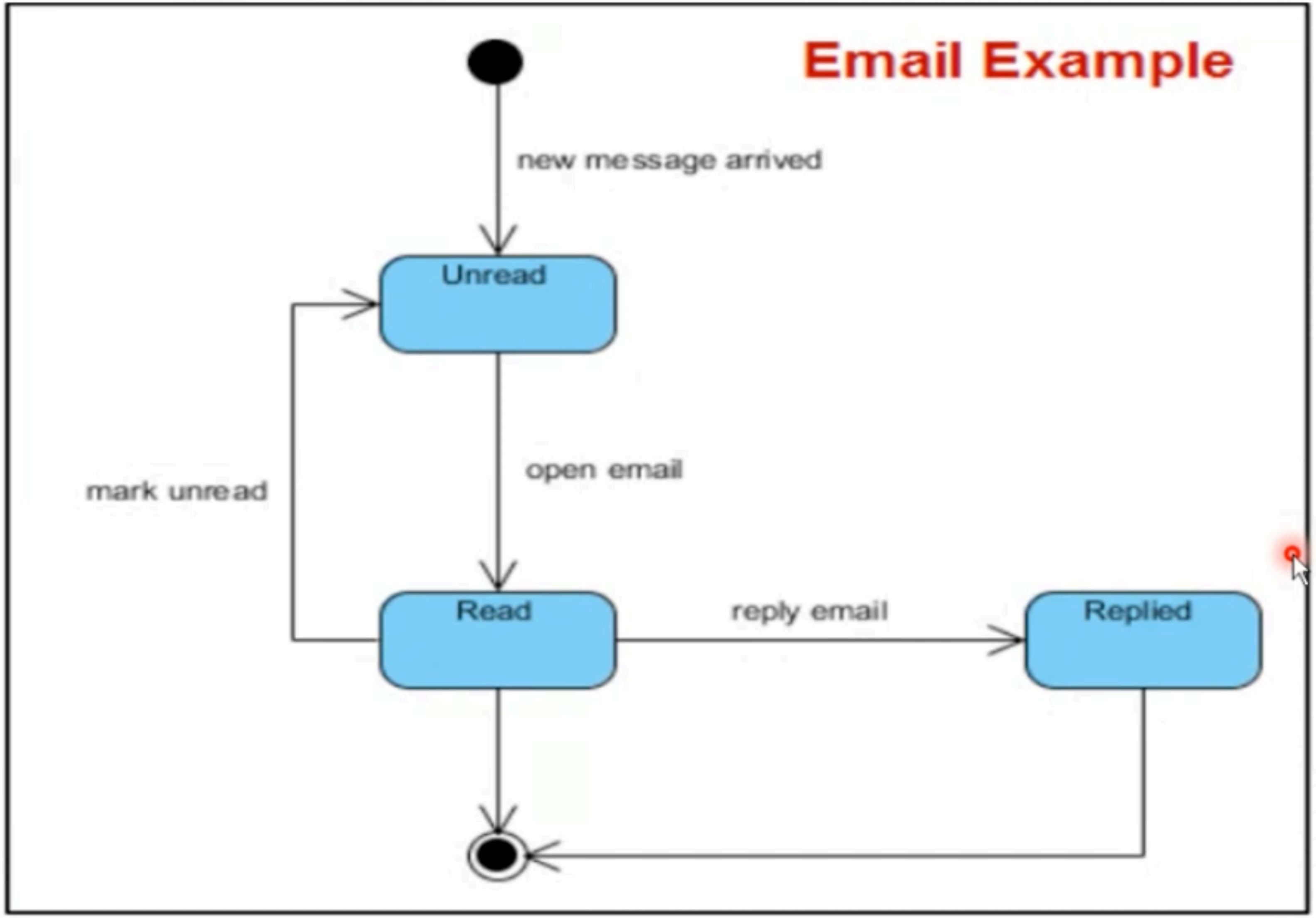
UML State Machine Diagrams

Example:



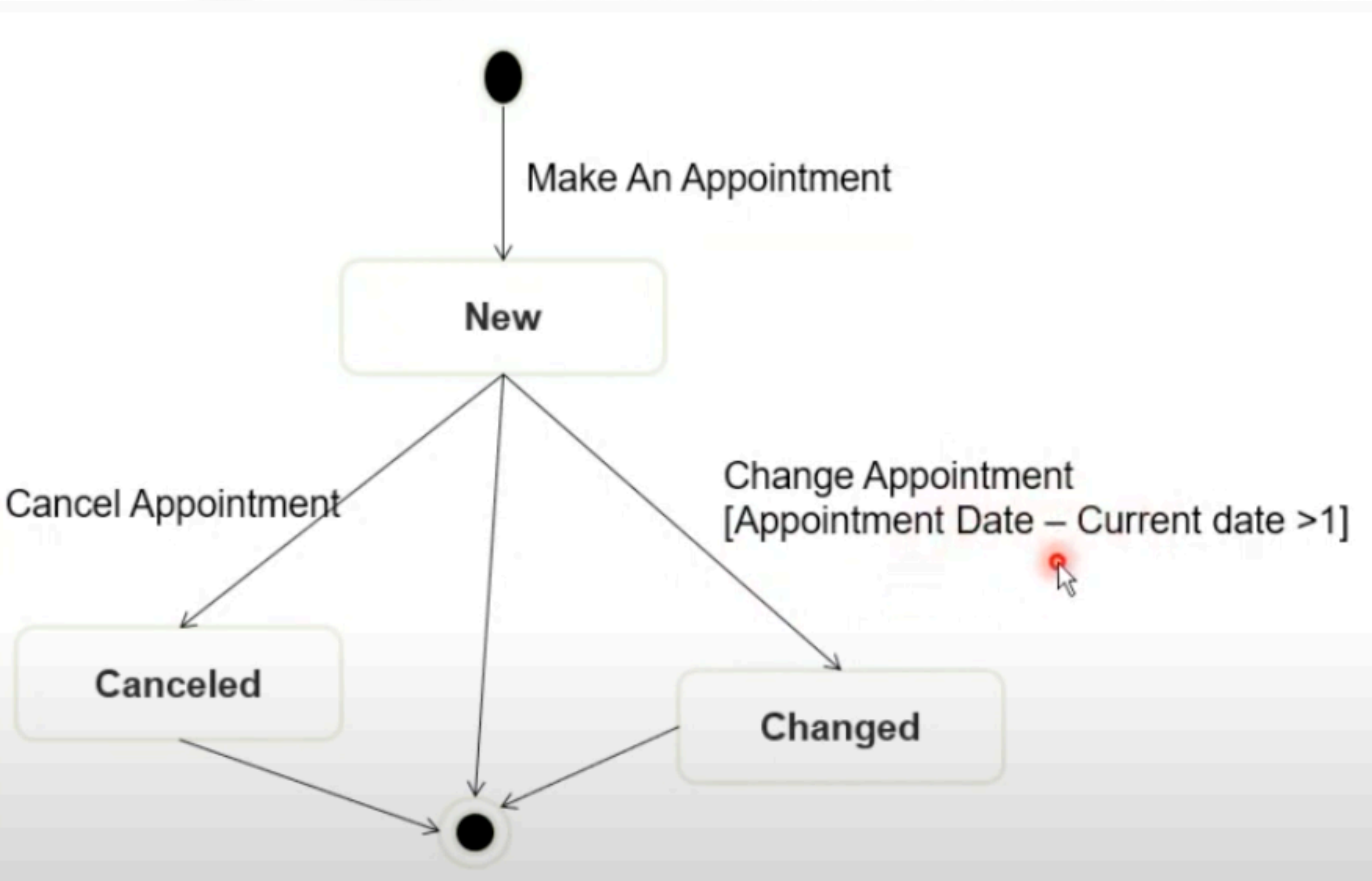
UML State Machine Diagrams

Example:



UML State Machine Diagrams

Example:



UML Activity Diagram

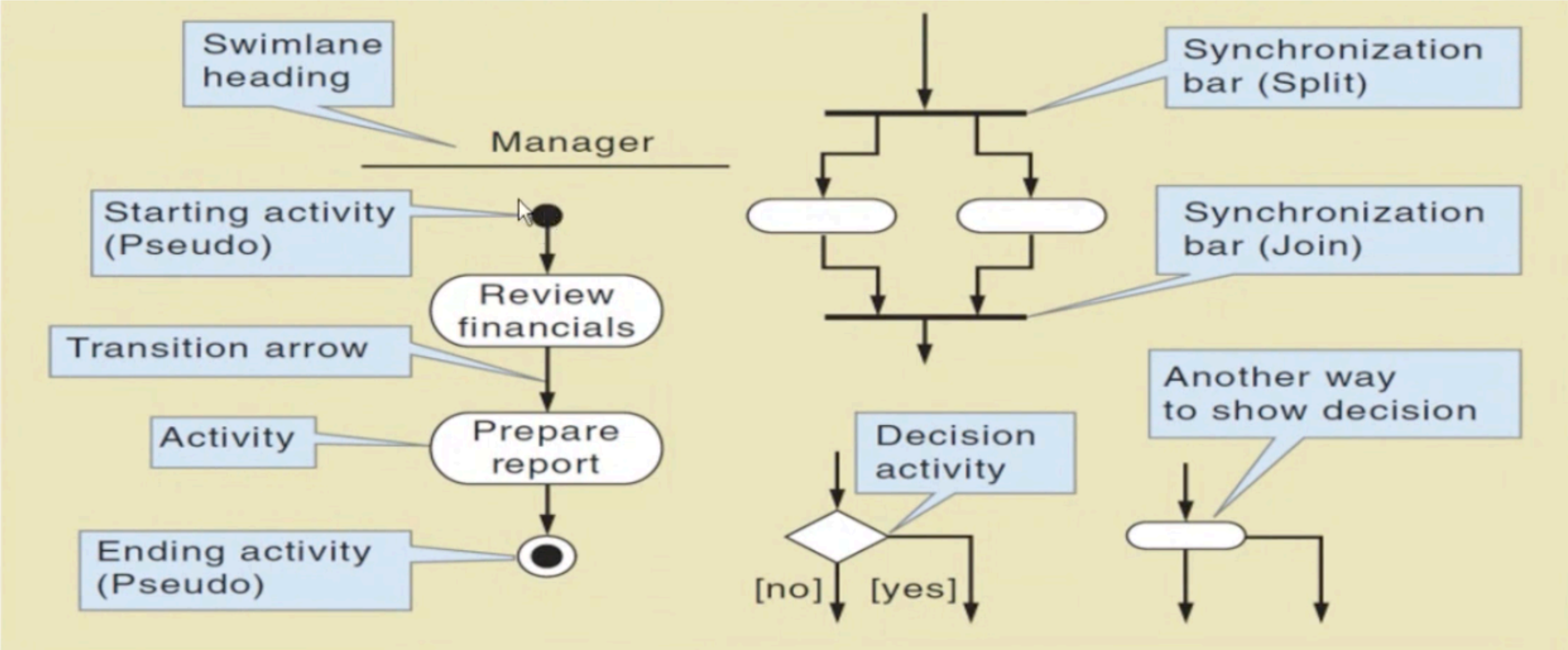
- activity diagram is used to describe A use case model. However, to depict a use case, a subset of the activity diagram capabilities is sufficient. In particular, it is not necessary to model concurrent activities for use cases.
- An activity diagram can be used to represent the sequential steps of a use case, including the main sequence and all the alternative sequences.
- Activity diagrams is something like the famous flow charts but it's much more powerful than flow charts. Flow charts are not part of UML diagrams.

UML Activity Diagram

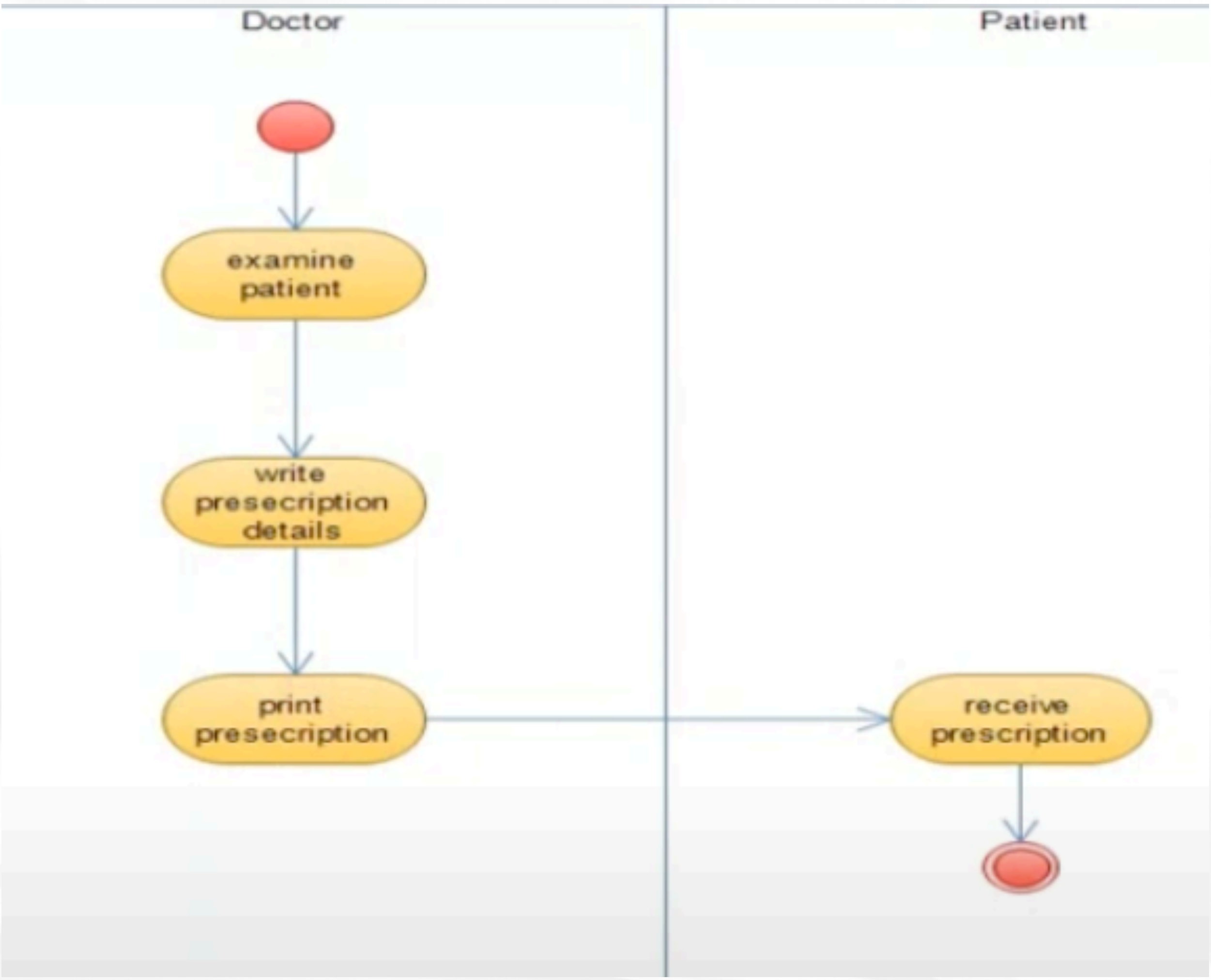
- key components and concepts of UML activity diagrams:

- **Initial and Final Nodes**: Initial nodes represent the start of the activity diagram, while final nodes represent the end.
- **Activities**: Activities represent tasks or actions that occur within the system.
- **Decisions**: Decisions, also known as decision nodes or decision points, represent points in the process where the flow of control can diverge based on conditions.
- **Merge Nodes**: Merge nodes are used to synchronize multiple incoming transitions into a single outgoing transition.
- **Forks and Joins**: Forks and joins are used to create parallel paths in the process flow. A fork splits the flow of control into multiple concurrent paths, while a join merges multiple concurrent paths back into a single path.

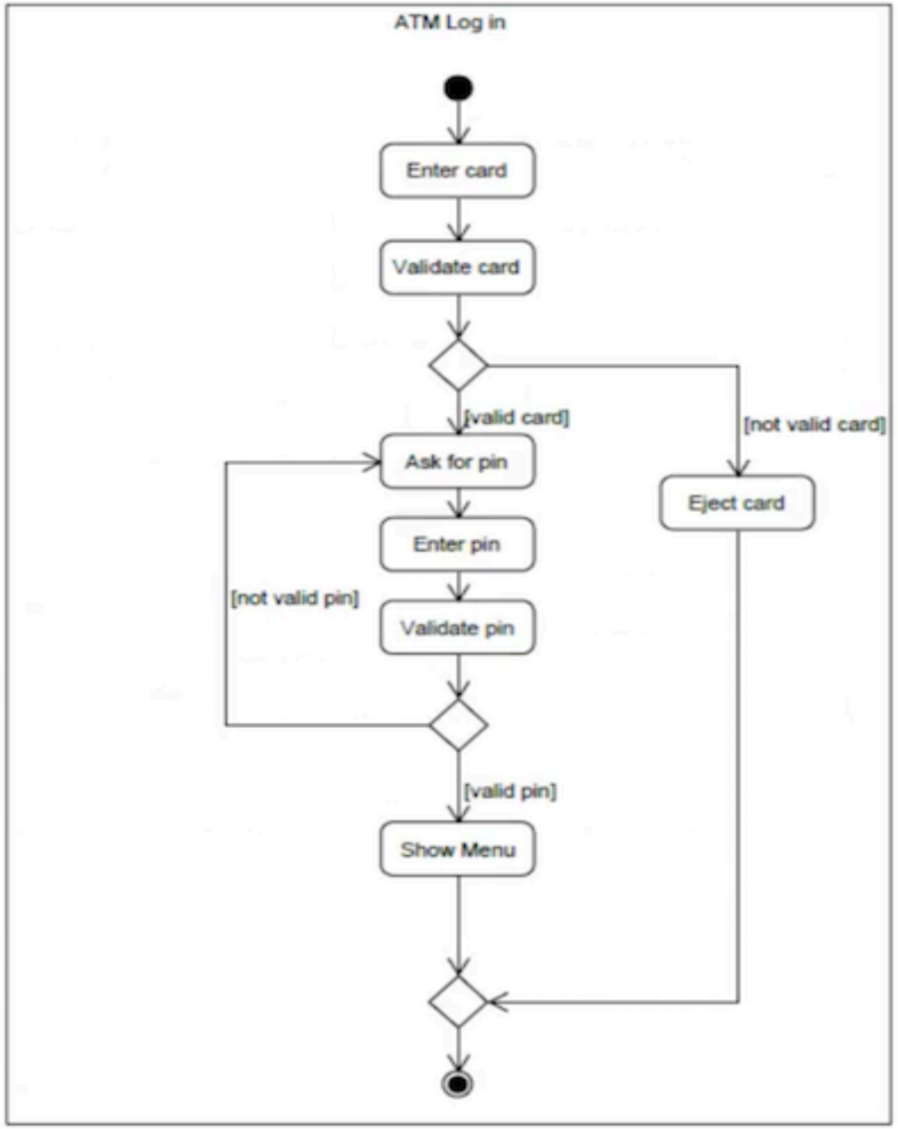
UML Activity Diagram



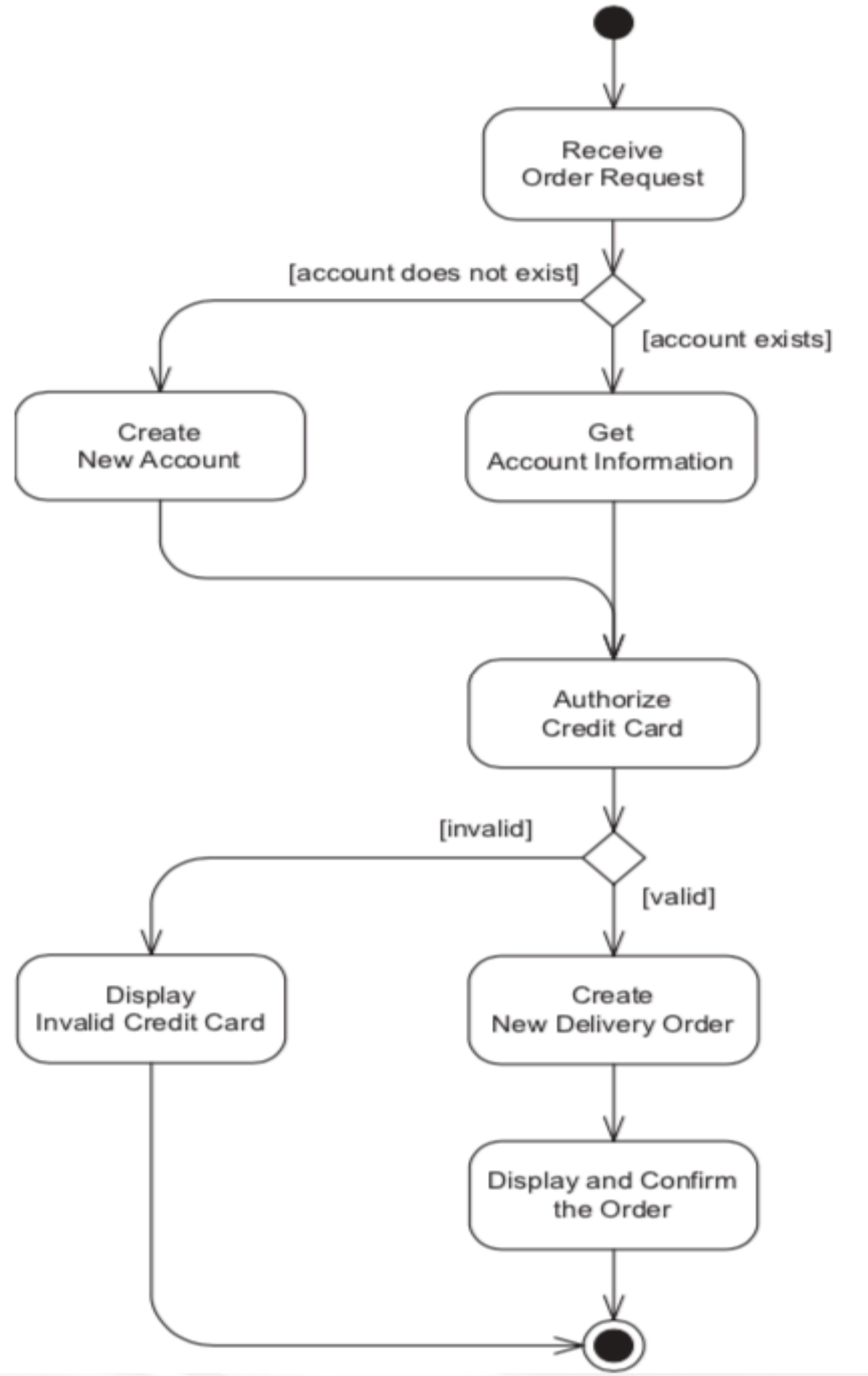
An example of an activity diagram for write prescription use case of the Clinic system



UML Activity Diagram



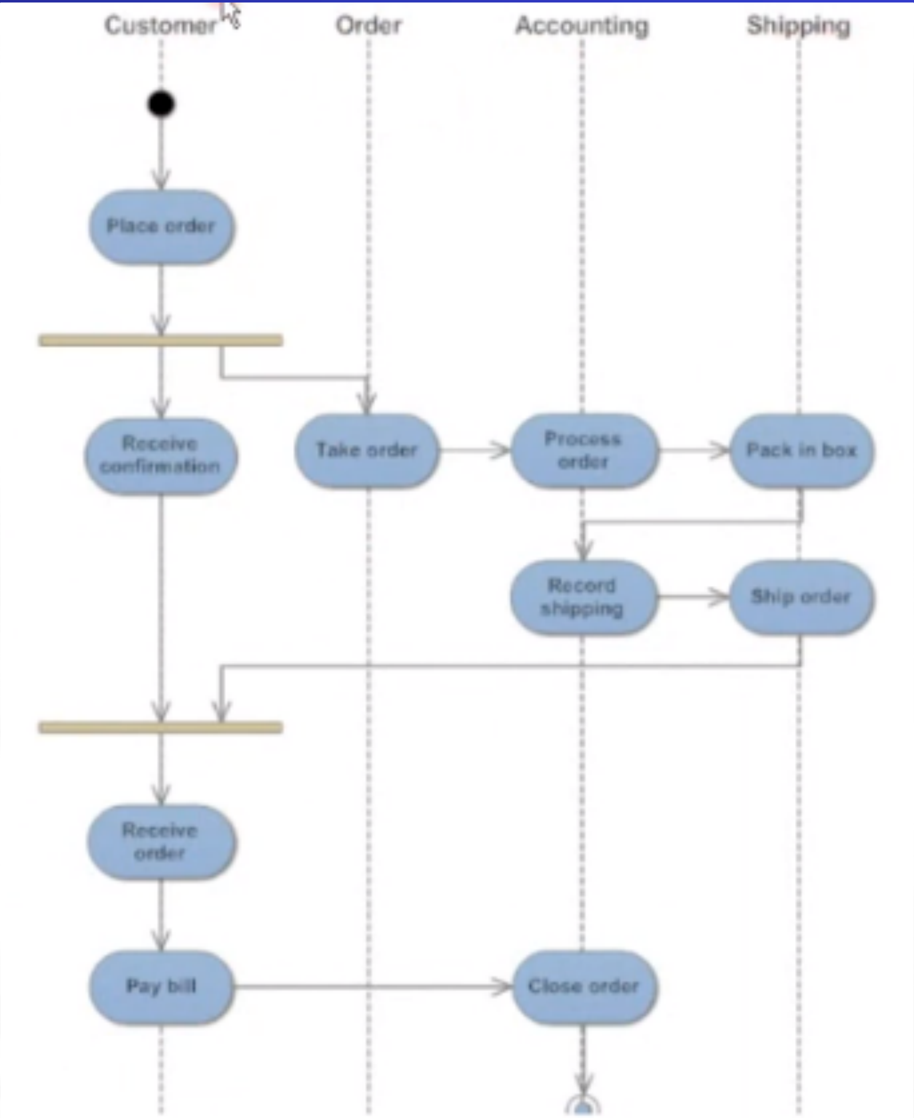
An example of an activity diagram for the Make Order Request use case of the Online Shopping System



UML Activity Diagram

Swimlane:
Swimlanes group related activates into one column.

Order processing



The End