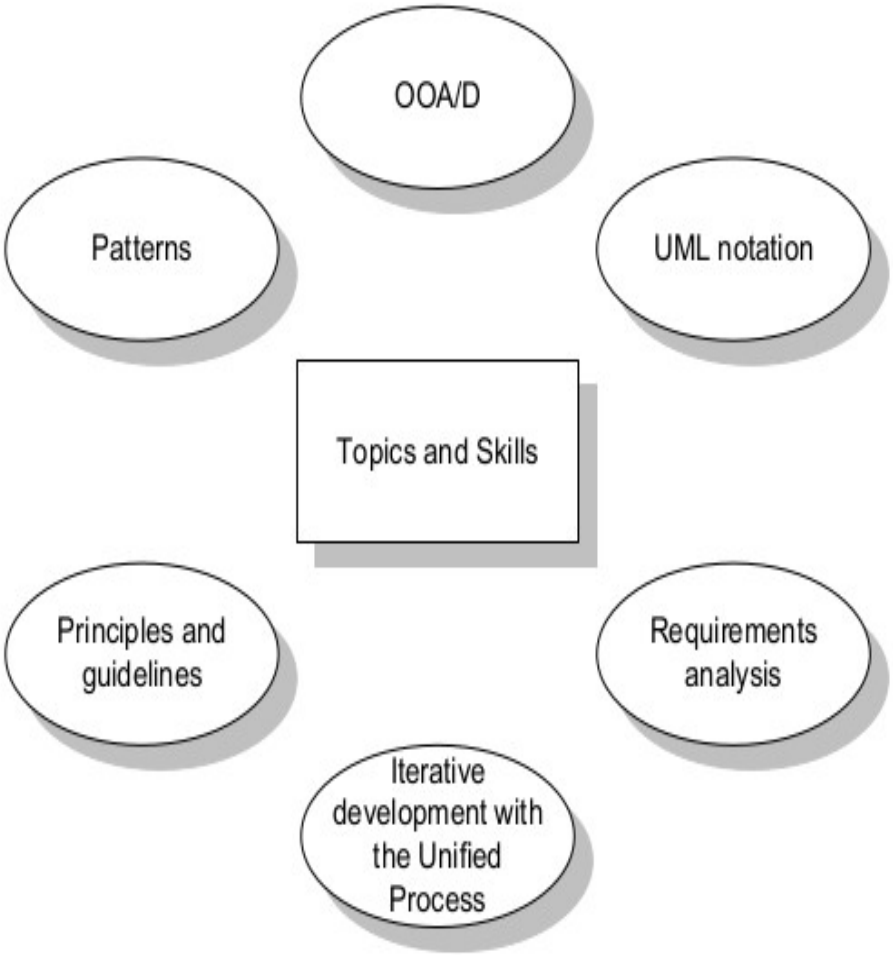
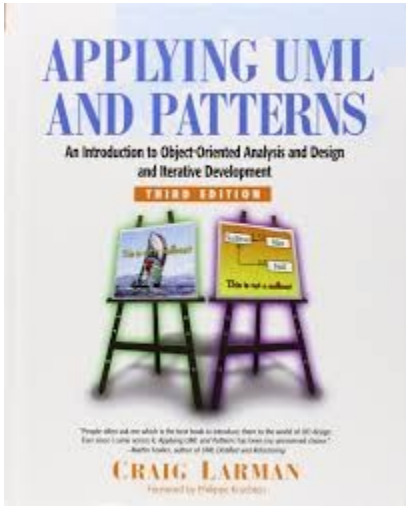
OO Concepts, OOAD and UML

Rajeev Gupta

Rgupta.mtech@gmail.com

Java Trainer & consultant


OOA/D
Patterns
UML notation
Topics and Skills
Principles and guidelines
Requirements analysis
Iterative development with the Unified Process

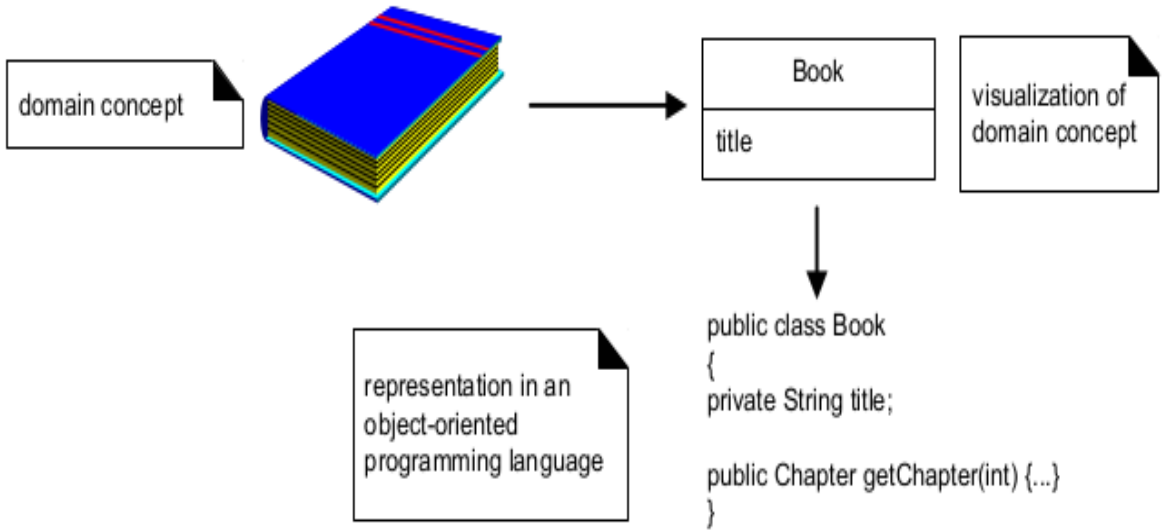

APPLYING UML AND PATTERNS
An Introduction to Object-Oriented Analysis and Design and Iterative Development
THIRD EDITION
CRAIG LARMAN

What Is Object-Oriented Analysis and Design?

During **object-oriented analysis,** there is an emphasis on finding and describing the objects—or concepts—in the problem domain. For example, in the case of the library information system, some of the concepts include *Book, Library,* and *Patron.*

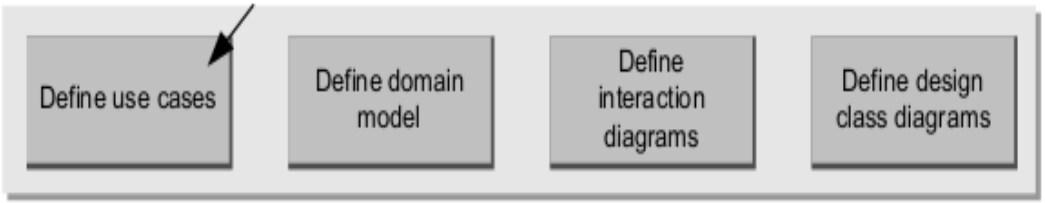
During **object-oriented design,** there is an emphasis on defining software objects and how they collaborate to fulfill the requirements. For example, in the library system, a *Book* software object may have a *title* attribute and a *getChap-ter* method (see Figure 1.2).

Finally, during implementation or object-oriented programming, design objects are implemented, such as a *Book* class in Java.



Define Use Cases

Requirements analysis may include a description of related domain processes; these can be written as **use cases.**

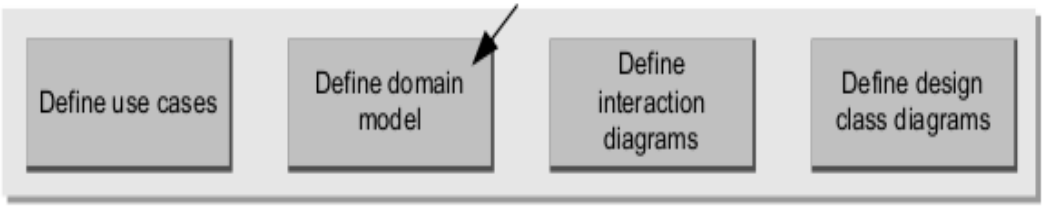


Use cases are not an object-oriented artifact—they are simply written stories. However, they are a popular tool in requirements analysis and are an important part of the Unified Process. For example, here is a brief version of the *Play a Dice Game* use case:

> **Play a Dice Game:** A player picks up and rolls the dice. If the dice face value total seven, they win; otherwise, they lose.

Define a Domain Model

Object-oriented analysis is concerned with creating a description of the domain from the perspective of classification by objects. A decomposition of the domain involves an identification of the concepts, attributes, and associations that are considered noteworthy. The result can be expressed in a **domain model,** which is illustrated in a set of diagrams that show domain concepts or objects.



For example, a partial domain model is shown in Figure 1.3.

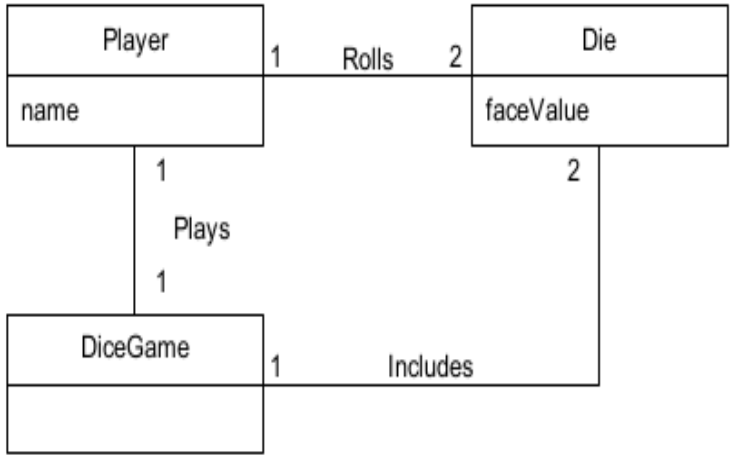
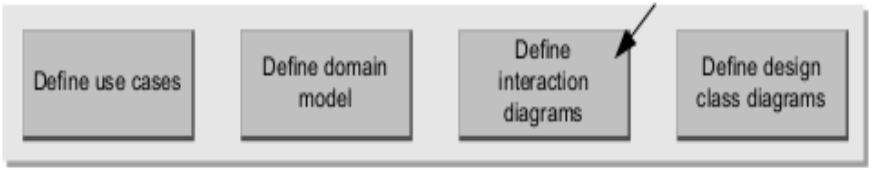


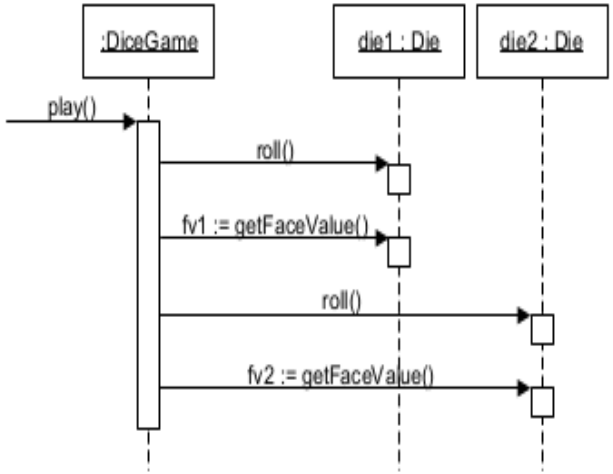
Figure 1.3 Partial domain model of the dice game.

Define Interaction Diagrams

Object-oriented design is concerned with defining software objects and their collaborations. A common notation to illustrate these collaborations is the **interaction diagram.** It shows the flow of messages between software objects, and thus the invocation of methods.

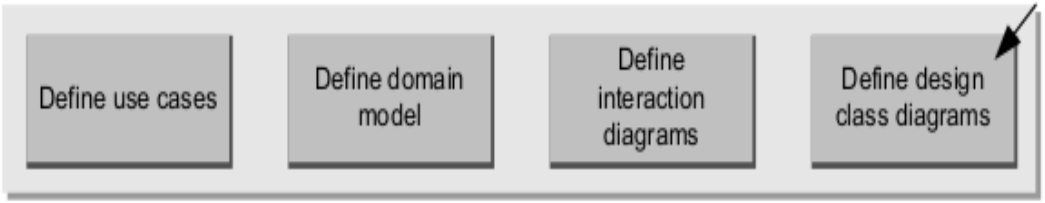


For example, assume that a software implementation of the dice game is desired. The interaction diagram in Figure 1.4 illustrates the essential step of playing, by sending messages to instances of the *DiceGame* and *Die* classes.



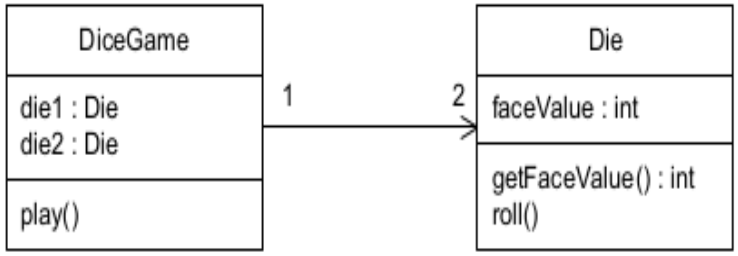
Define Design Class Diagrams

In addition to a *dynamic* view of collaborating objects shown in interaction diagrams, it is useful to create a *static* view of the class definitions with a **design class diagram.** This illustrates the attributes and methods of the classes.



For example, in the dice game, an inspection of the interaction diagram leads to the partial design class diagram shown in Figure 1.5. Since a *play* message is sent to a *DiceGame* object, the *DiceGame* class requires a *play* method, while class *Die* requires a *roll* and *getFaceValue* method.

In contrast to the domain model, this diagram does not illustrate real-world concepts; rather, it shows software classes.



Rational Unified Process (RUP)

Early iterative process ideas were known as spiral development and evolutionary development [Boehm.88, Gilb88].

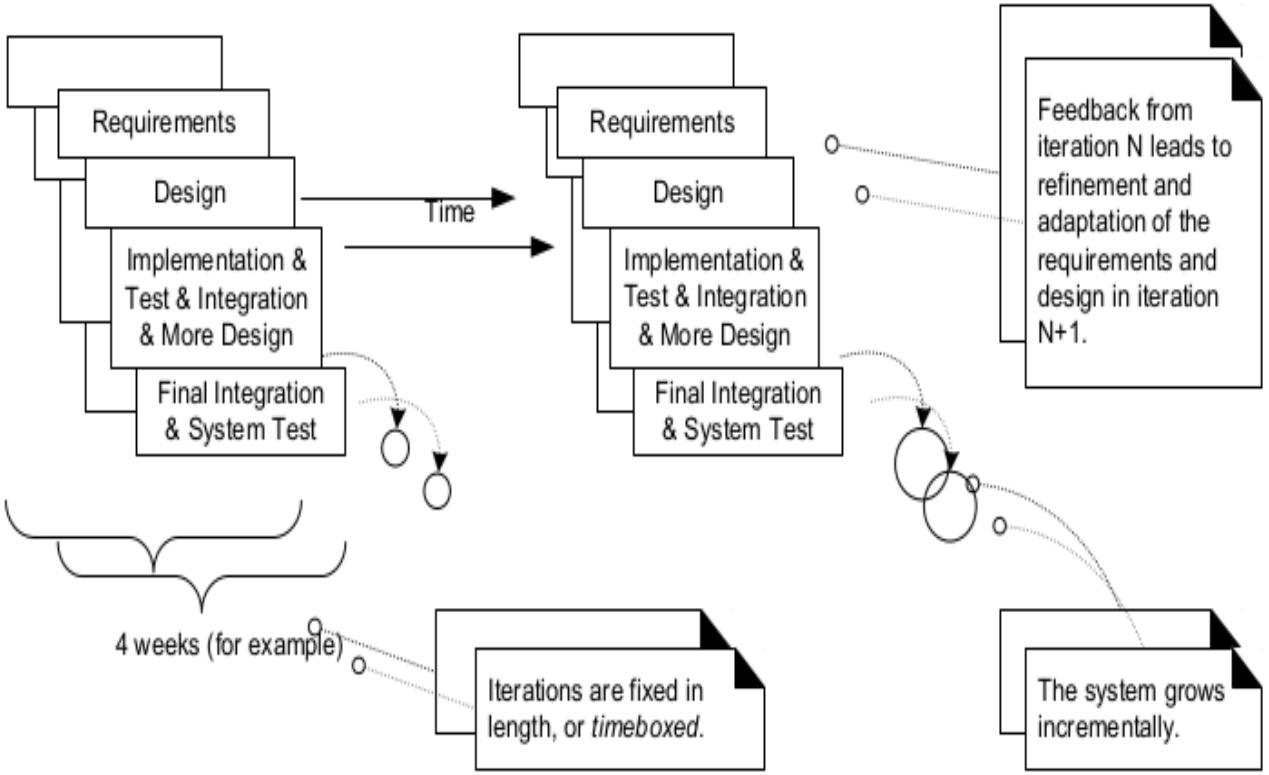


Figure 2.1 Iterative and incremental development.

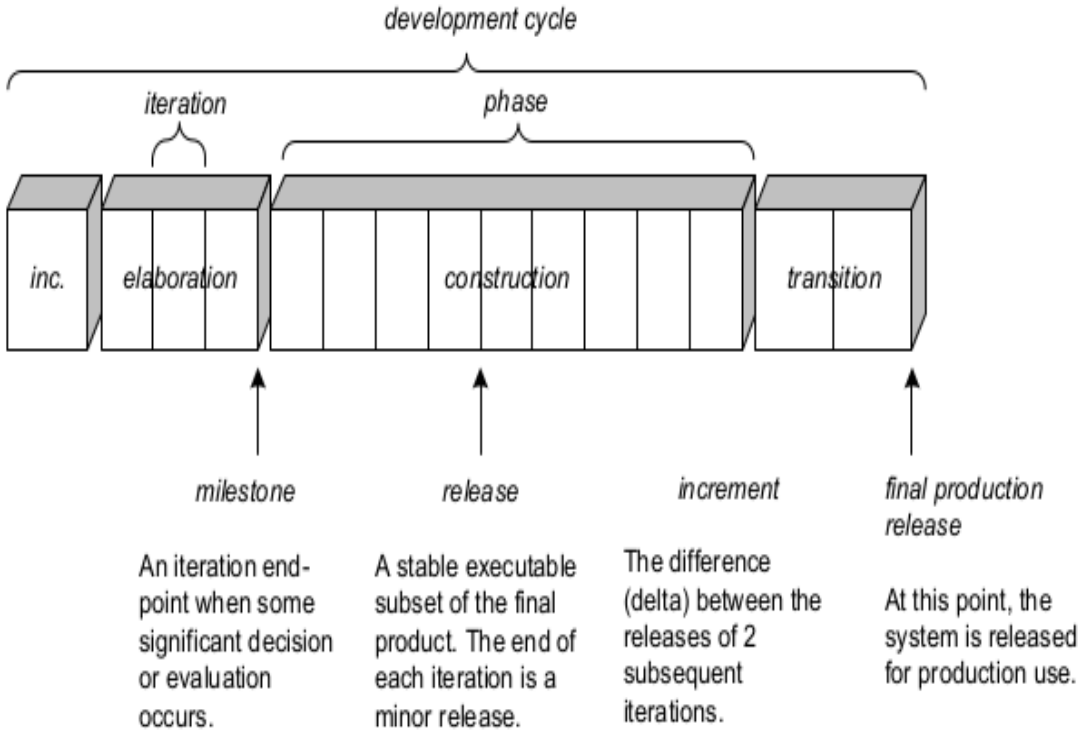


Figure 2.3 Schedule-oriented terms in the UP.

Types of Requirements

In the UP, requirements are categorized according to the FURPS+ model [Grady92], a useful mnemonic with the following meaning:[1]

- **Functional**—features, capabilities, security.
- **Usability**—human factors, help, documentation.
- **Reliability**—frequency of failure, recoverability, predictability.
- **Performance**—response times, throughput, accuracy, availability, resource usage.
- **Supportability**—adaptability, maintainability, internationalization, con
figurability.

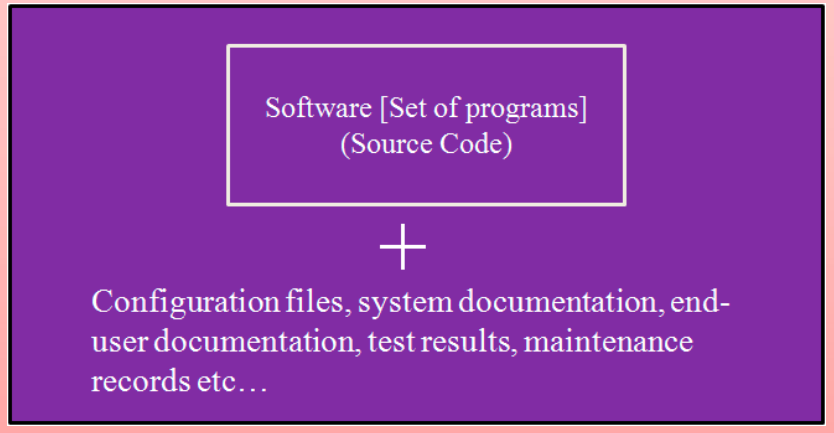
The "+" in FURPS+ indicates ancillary and sub-factors, such as:

- **Implementation**—resource limitations, languages and tools, hardware, ...
- **Interface**—constraints imposed by interfacing with external systems.
- **Operations**—system management in its operational setting.
- **Packaging**
- **Legal**—licensing and so forth.

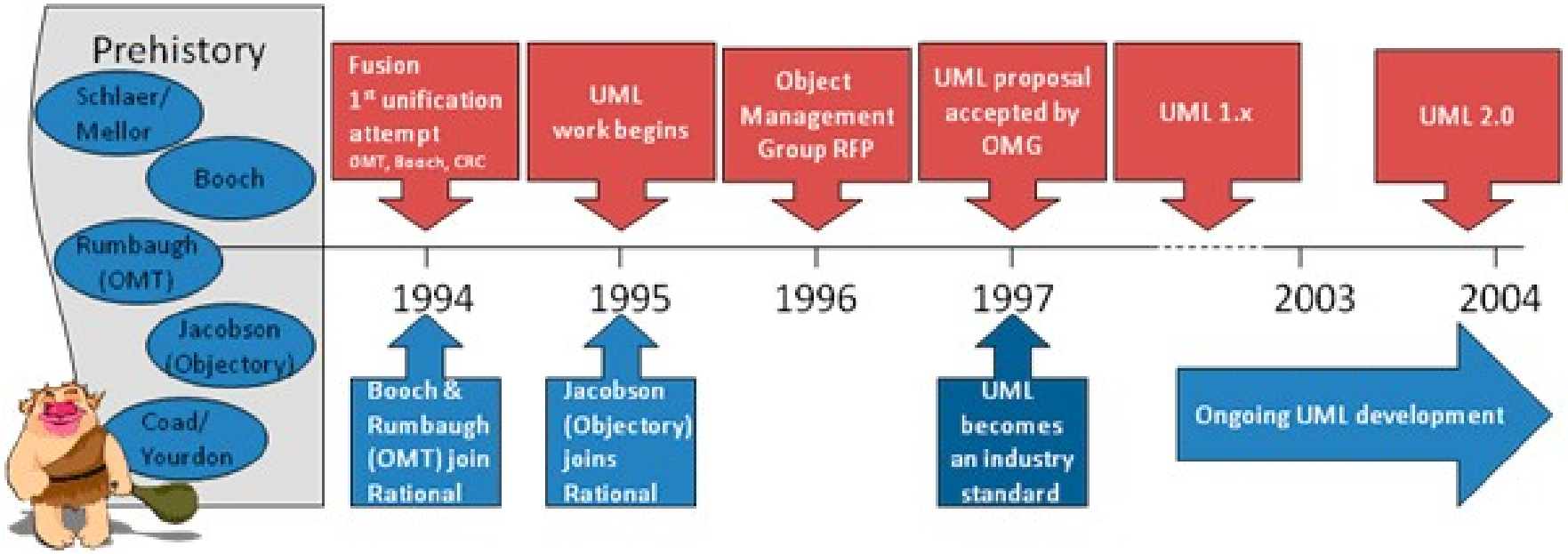
What is a model?

A model is a simplified representation of a thing. The model captures the important aspects of the thing being modeled and leaves out the rest of the details

Unified Modeling Language (UML)



OO Modeling History



Overview of UML

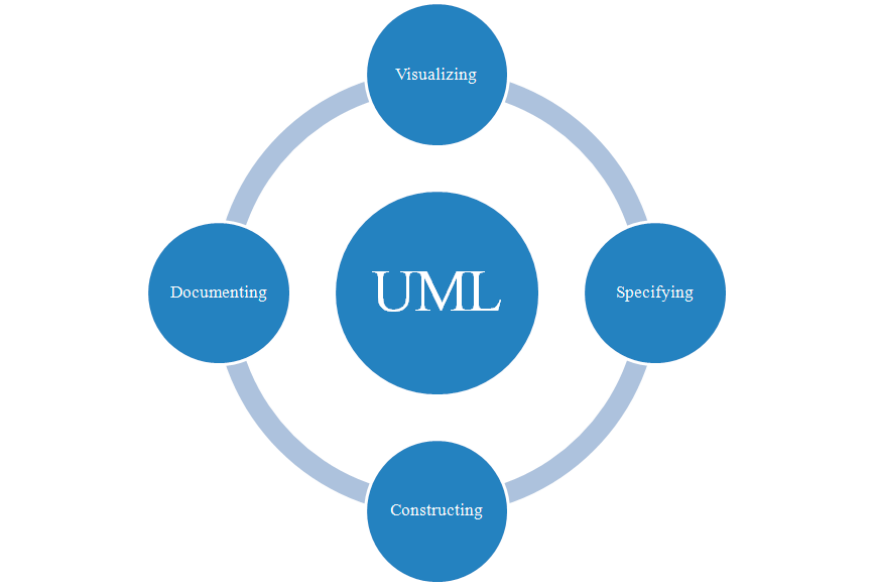
The UML is a language for (overview of UML):

Visualizing

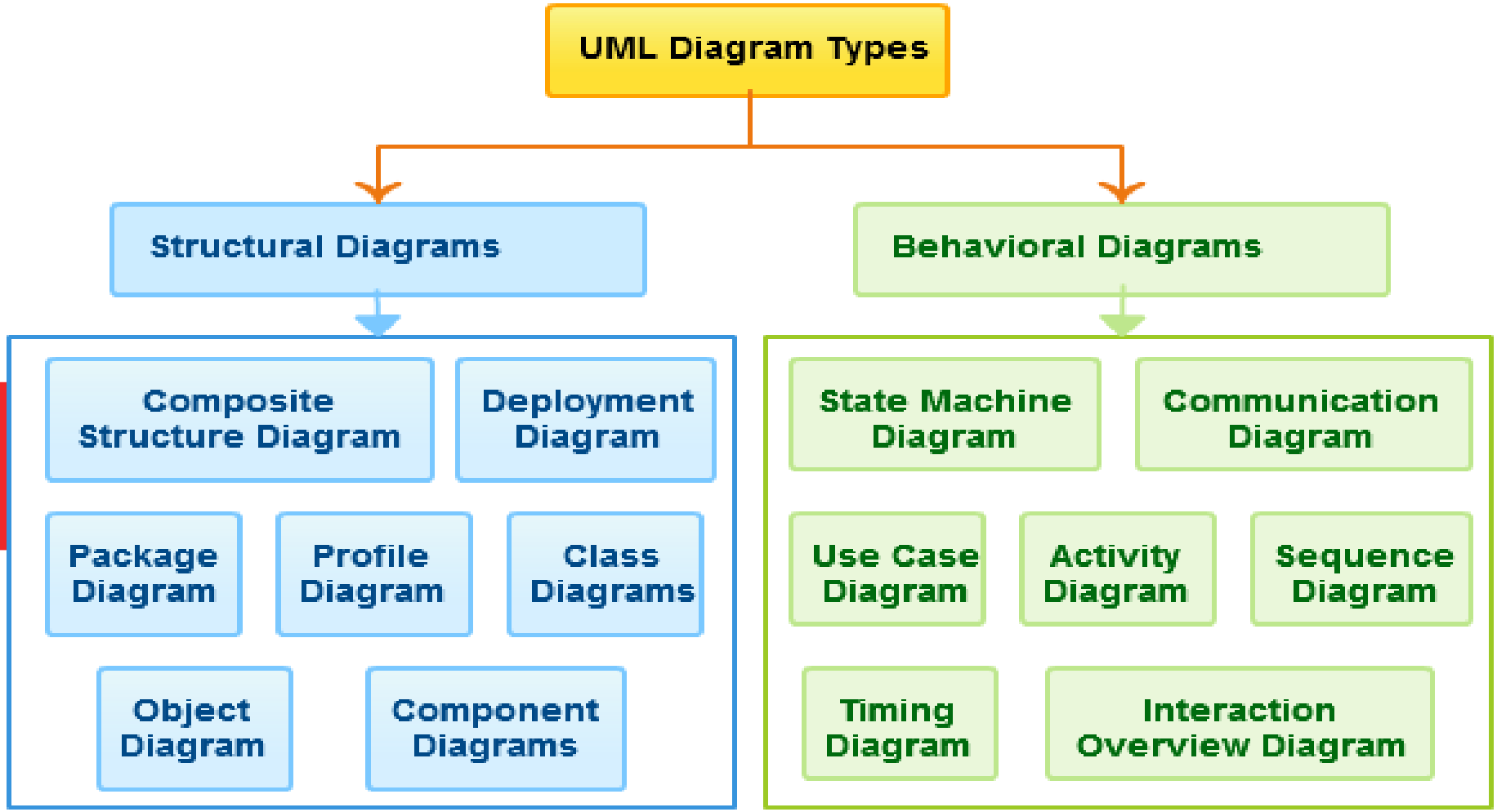
Specifying

Constructing

Documenting the artifacts of a software intensive system.



UML Diagrams Types



Conceptual model of UML

These are the fundamental elements in UML. Every diagram can be represented using these building blocks. The building blocks of UML contains three types of elements. They are:

1) Things (object oriented parts of uml)

2) Relationships (relational parts of uml)

3) Diagrams

Things

A diagram can be viewed as a graph containing vertices and edges. In UML, vertices are replaced by things, and the edges are replaced by relationships. There are four types of things in UML. They are:

1) Structural things (nouns of uml – static parts)

2) Behavioral things (verbs of uml – dynamic parts)

3) Grouping things (organizational parts)

4) Annotational things (explanatory parts)

Structural things

Represents the static aspects of a software system. There are seven structural things in UML. They are

Class

Interface

Use cases

Collaboration

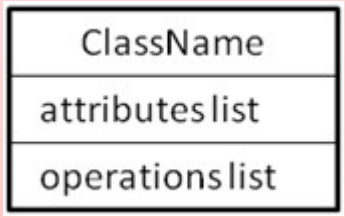
Component

Node

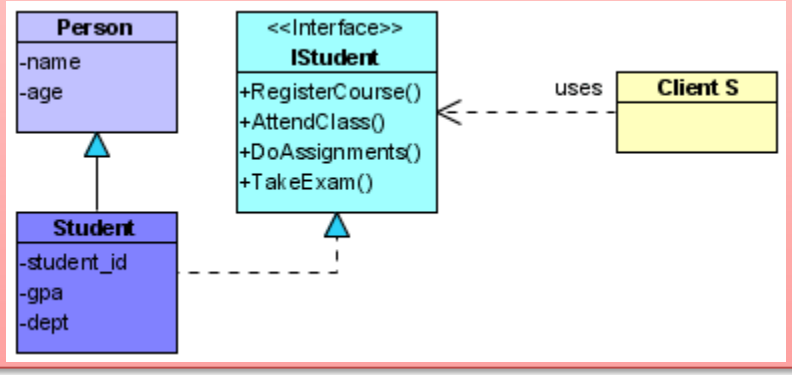
Active Class

Structural things

Class: A class is a collection of similar objects having similar attributes, behavior, relationships and semantics. Graphically class is represented as a rectangle with three compartments.

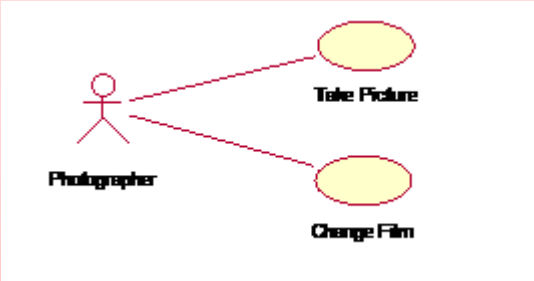


Interface: An interface is a collection of operation signatures and/or attribute definitions that ideally define a cohesive set of behavior.

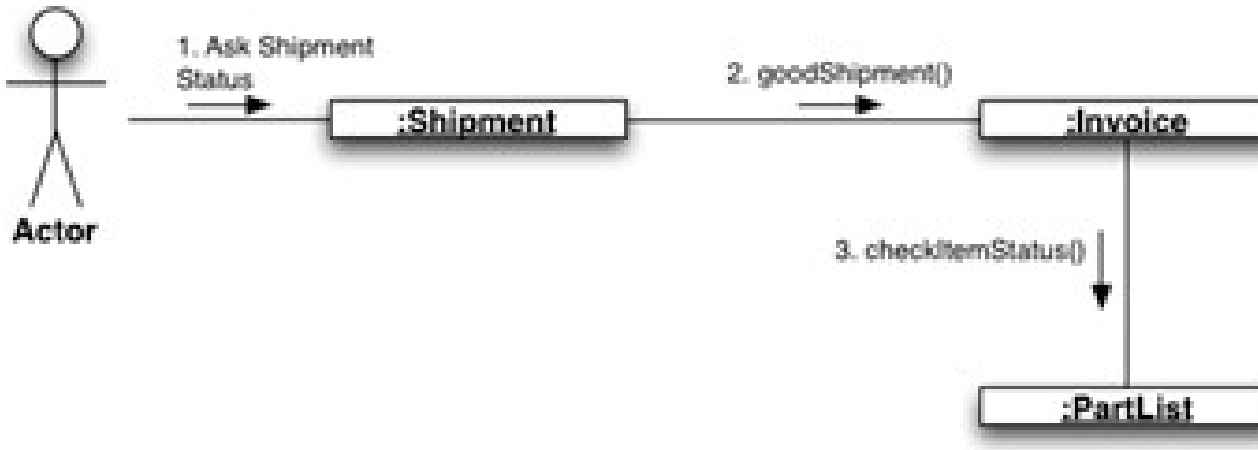


Structural things

Use Case: A use case is a collection of actions, defining the interactions between a role (actor) and the system.

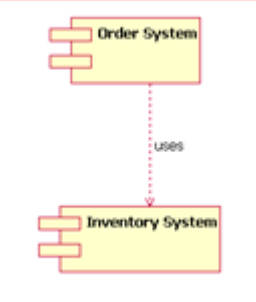


Collaboration: A collaboration is the collection of interactions among objects to achieve a goal.

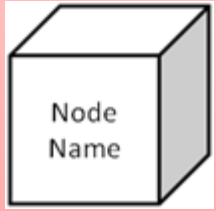


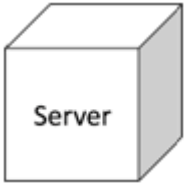
Structural things

Component: A component is a physical and replaceable part of a system. Graphically component is represented as a tabbed rectangle



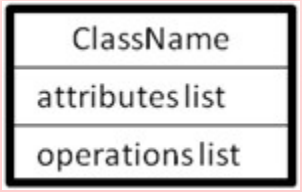
Node: A node is a physical element that exists at run time and represents a computational resource

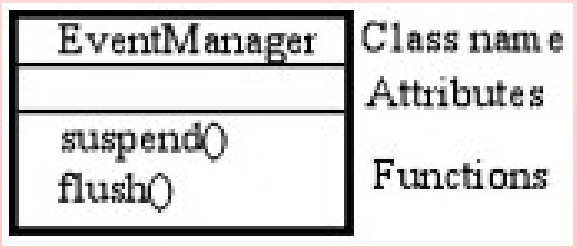




Structural things

Active Class: A class whose objects can initiate its own flow of control (threads) and work in parallel with other objects

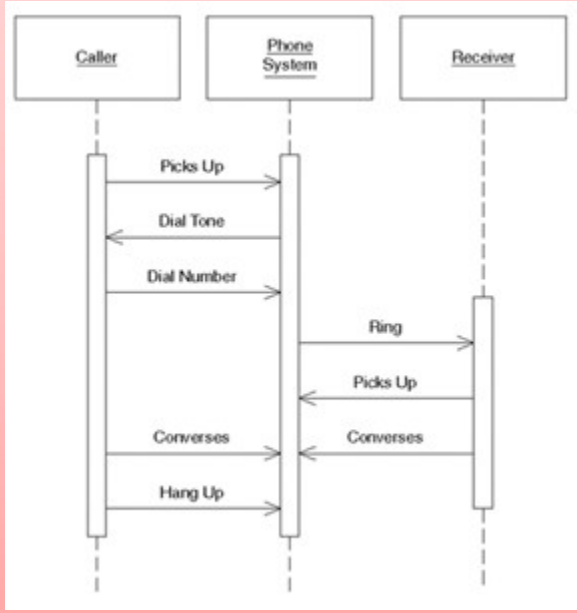




Behavioral Things

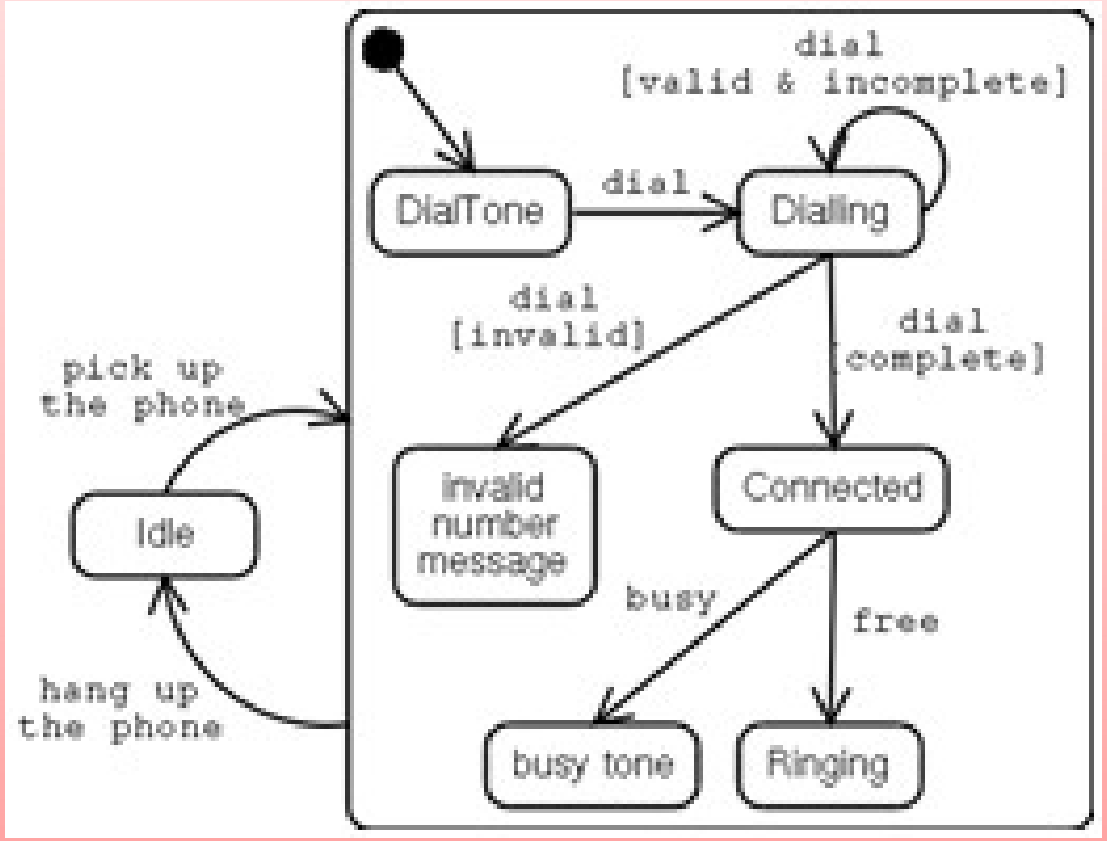
Represents the dynamic aspects of a software system. Behavior of a software system can be modeled as interactions or as a sequence of state changes.

Interaction: A behavior made up of a set of messages exchanged among a set of objects to perform a particular task.



Behavioral Things

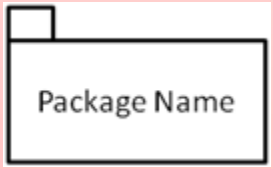
State Machine: A behavior that specifies the sequences of states an object or interaction goes through during its' lifetime in response to events. A state is represented as a rectangle with rounded corners



Grouping Things

Elements which are used for organizing related things and relationships in models.

Package: A general purpose mechanism for organizing elements into groups. Graphically package is represented as a tabbed folder. When the diagrams become large and cluttered, related are grouped into a package so that the diagram can become less complex and easy to understand



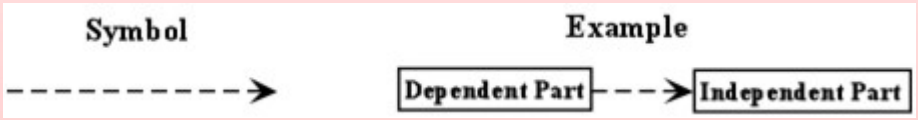
Annotational Things

Note: A symbol to display comments. Graphically note is represented as a rectangle with a dog ear at the top right corner.

Relationships

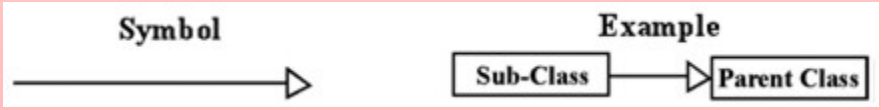
The things in a diagram are connected through relationships. So, a relationship is a connection between two or more things.

Dependency

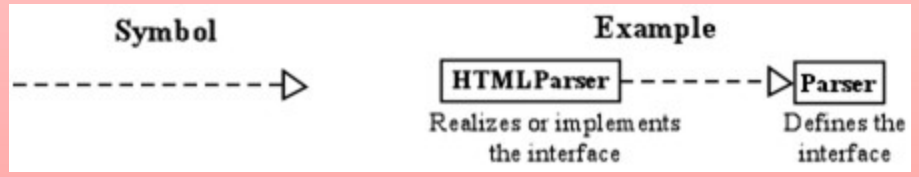


Association

Generalization



Realization



Diagrams

A diagram is a collection of elements often represented as a graph consisting of vertices and edges joining these vertices. These vertices in UML are things and the edges are relationships. UML includes nine diagrams:

1) Class diagram

2) Object diagram

3) Use case diagram

4) Component diagram

5) Deployment diagram

6) Sequence diagram

7) Collaboration diagram

8) Statechart diagram and

9) Activity diagram.

Rules (conceptual model of UML)

The rules of UML specify how the UMLs building blocks come together to develop diagrams. The rules enable the users to create well-formed models. A well-formed model is self-consistent and also consistent with the other models.

UML has rules for

Names – What elements can be called as things, relationships and diagrams

Scope – The context that gives a specific meaning to a name

Visibility – How these names are seen and can be used by the other names

Integrity – How things properly relate to one another

Execution – What it means to run or simulate a model

Architecture

A model is a simplified representation of the system. To visualize a system, we will build various models. The subset of these models is a view. Architecture is the collection of several views.

The stakeholders (end users, analysts, developers, system integrators, testers, technical writers and project managers) of a project will be interested in different views.

Architecture can be best represented as a collection five views:

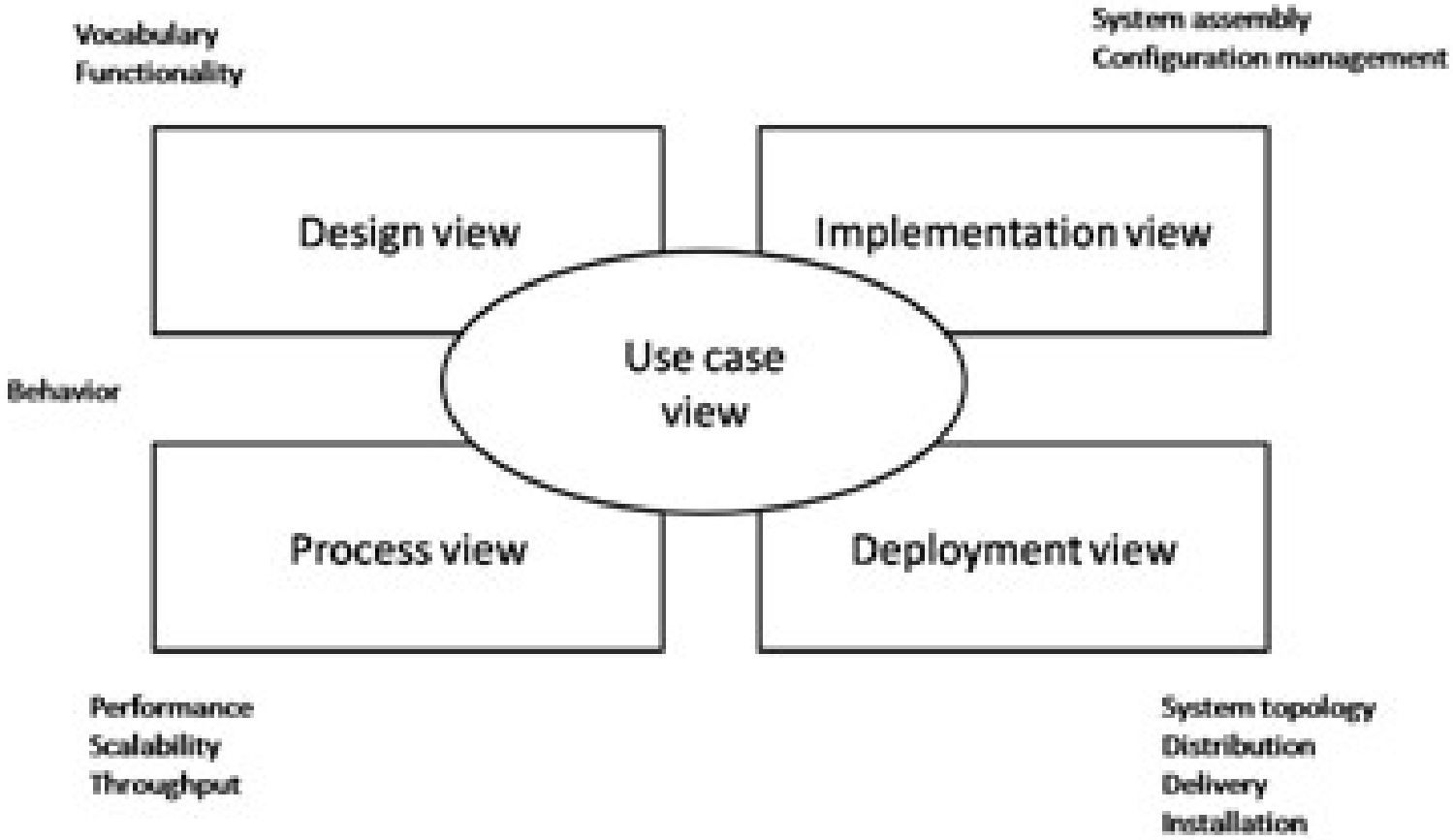
1) Use case view,

2) Design/logical view,

3) Implementation/development view,

4) Process view and

5) Deployment/physical view.


Vocabulary
Functionality
System assembly
Configuration management
Design view
Implementation view
Use case view
Behavior
Process view
Deployment view
Performance
Scalability
Throughput
System topology
Distribution
Delivery
Installation

View	Stakeholders	Static Aspects	Dynamic Aspects
Use case view	End users Analysts Testers	Use case diagrams	Interaction diagrams Statechart diagrams Activity diagrams
Design view	End users	Class diagrams	Interaction diagrams Statechart diagrams Activity diagrams
Implementation view	Programmers Configuration managers	Component diagrams	Interaction diagrams Statechart diagrams Activity diagrams
Process view	Integrators	Class diagrams (active classes)	Interaction diagrams Statechart diagrams Activity diagrams
Deployment view	System Engineer	Deployment diagrams	Interaction diagrams Statechart diagrams Activity diagrams

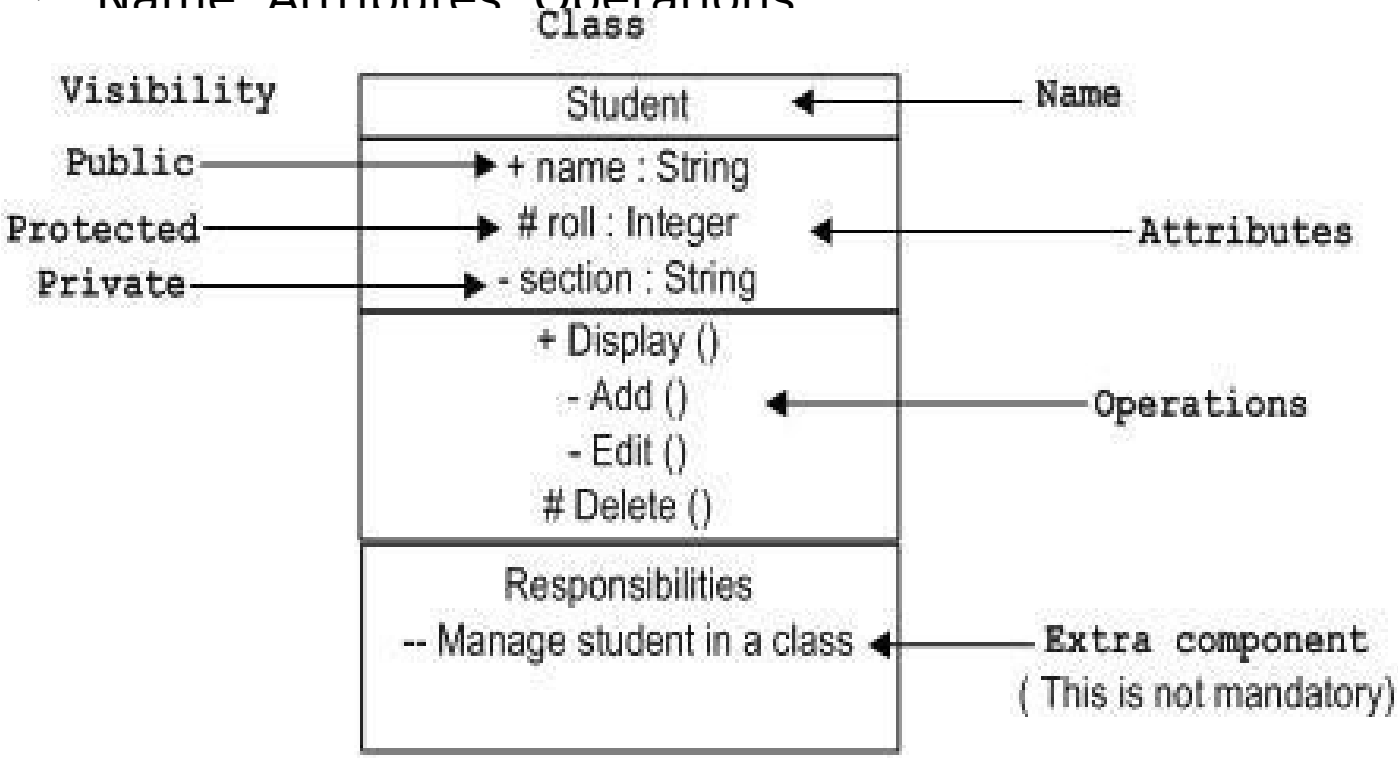
Class Diagram

Class diagram

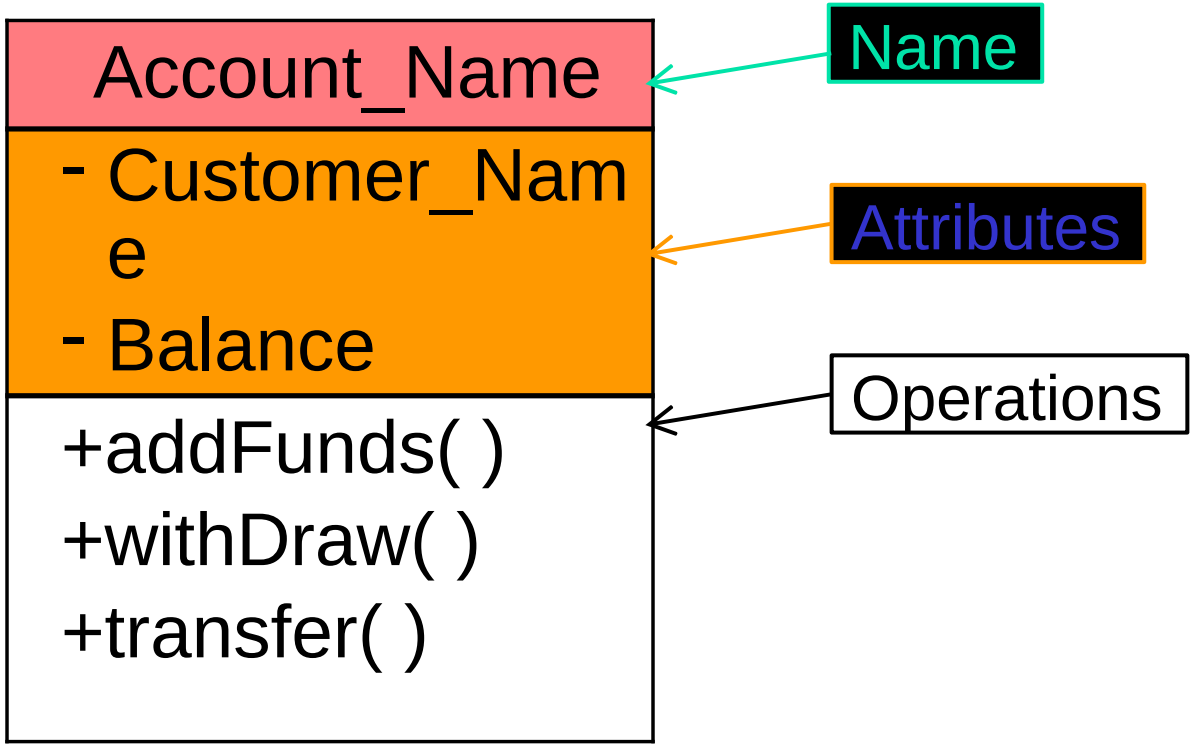
- A class diagram depicts classes and their interrelationships
- Used for describing structure and behavior in the use cases
- Provide a conceptual model of the system in terms of entities and their relationships– to translate the models into programming code.
- Represents the structural – static behavior of the System.
- Used for requirement capture, end-user interaction
- Detailed class diagrams are used for developers

Class diagram

- Each class is represented by a rectangle subdivided into three compartments
  - Name, Attributes, Operations



Class diagram



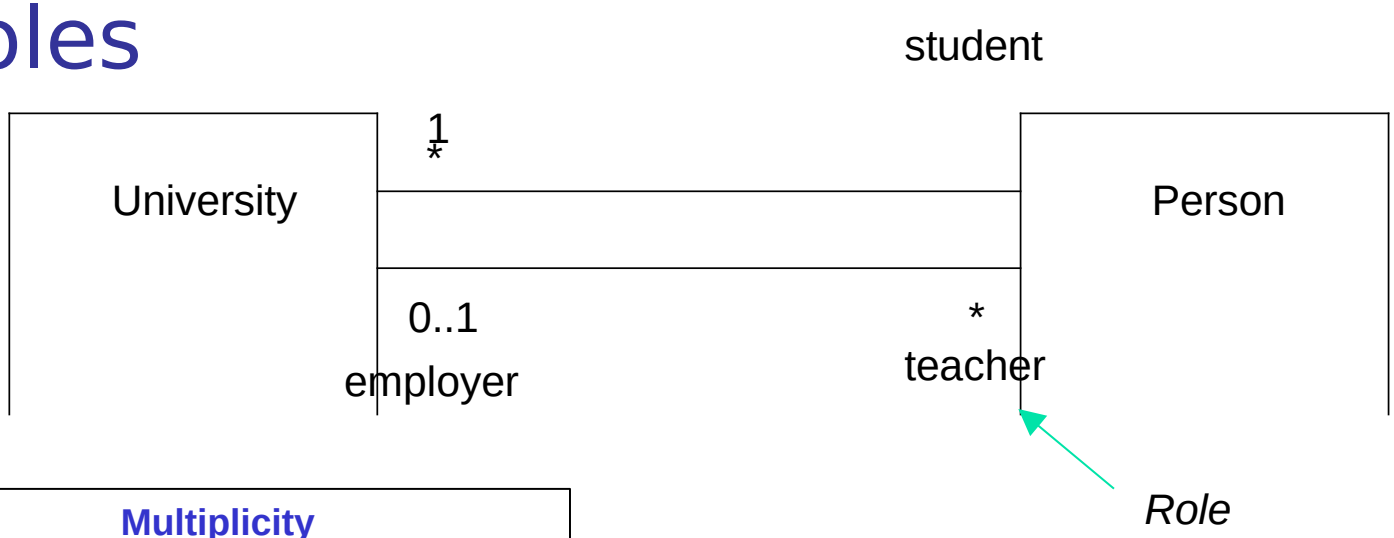
OO Relationships

- There are two kinds of Relationships
  - Generalization (parent-child relationship)
  - Association (student enrolls in course)
- Associations can be further classified as
  - Aggregation
  - Composition

OO Relationships: **Association**

- Represent relationship between instances of classes
  - Student enrolls in a course
  - Courses have students
  - Courses have exams
  - Etc.
- Association has two ends
  - Role names (e.g. enrolls)
  - Multiplicity (e.g. One course can have many students)
  - Navigability (unidirectional, bidirectional)

Association: Multiplicity and Roles



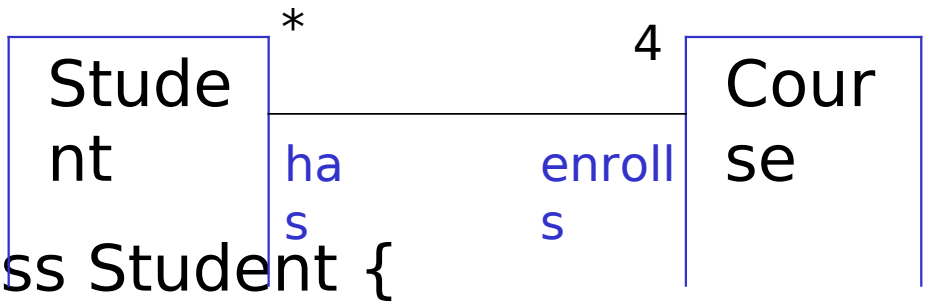
Multiplicity

Symbol	Meaning
1	One and only one
0..1	Zero or one
M..N	From M to N (natural language)
*	From zero to any positive integer
0..*	From zero to any positive integer
1..*	From one to any positive integer

Role

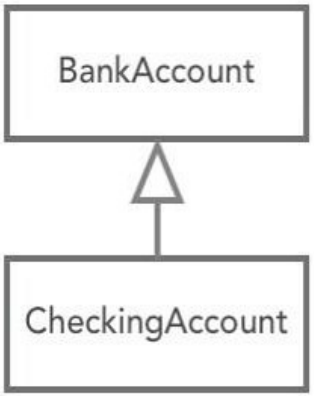
"A given university groups many people; some act as students, others as teachers. A given student belongs to a single university; a given teacher may or may not be working for the university at a particular time."

Association: Model to Implementation

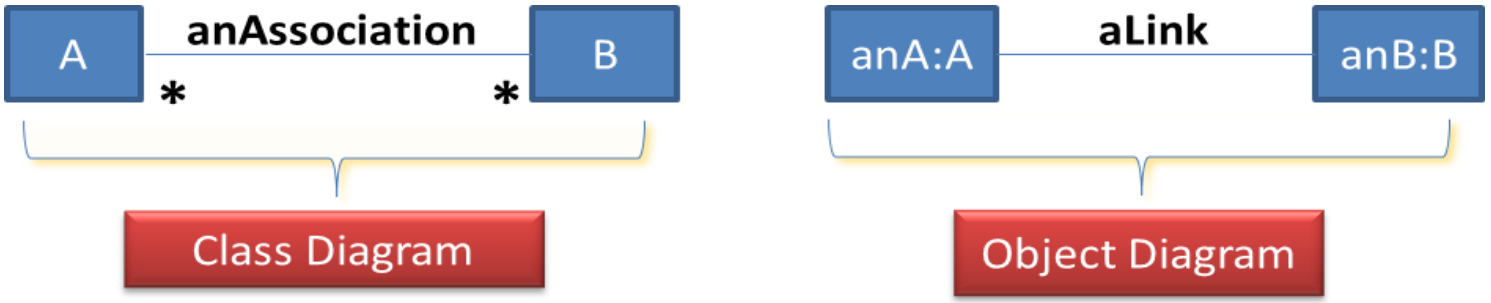



```
class Student {
    Course
    enrolls[4];
}

class Course
    {  Student
    have[];
}
```

ASSOCIATIONS



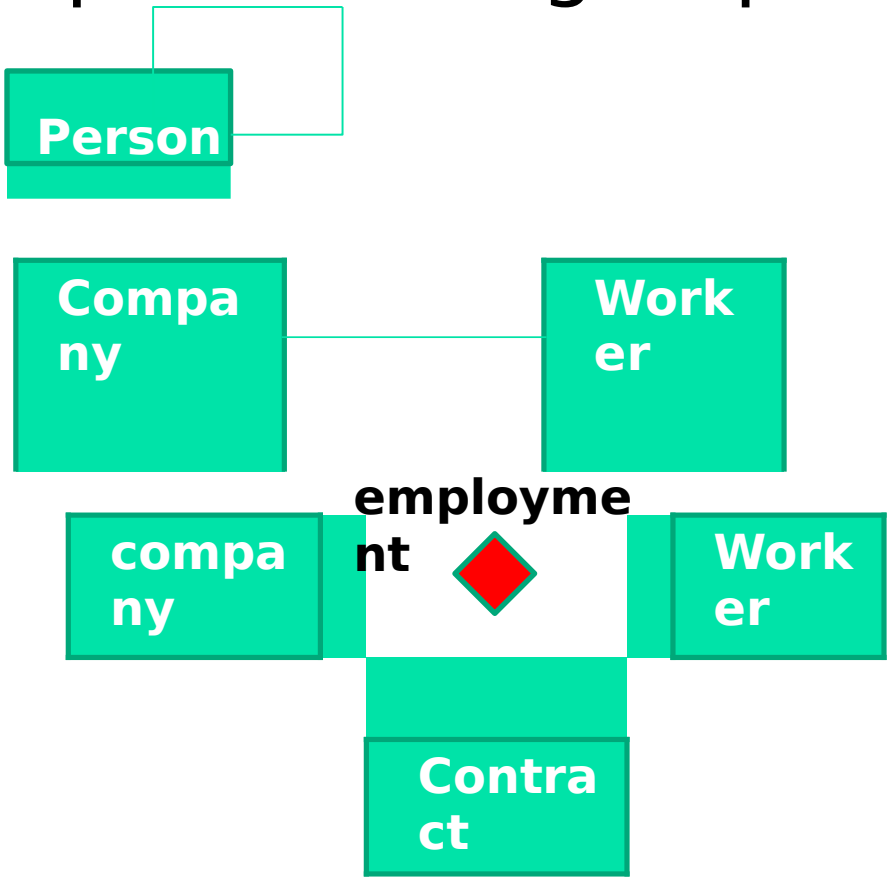
Link and Association



- Examples of class and object diagrams
  - Person and Company
  - Country and CapitalCity
  - Workstation and Window

Associations

- Association represents a group of links.
  - Self association:
  - Binary association:
  - N-array association:



Association End Names

- Use of association end names is optional, but it is often easier and less confusing to assign association end names instead of or in addition to, association names.
- Association end names are necessary for associations between two objects of the same class

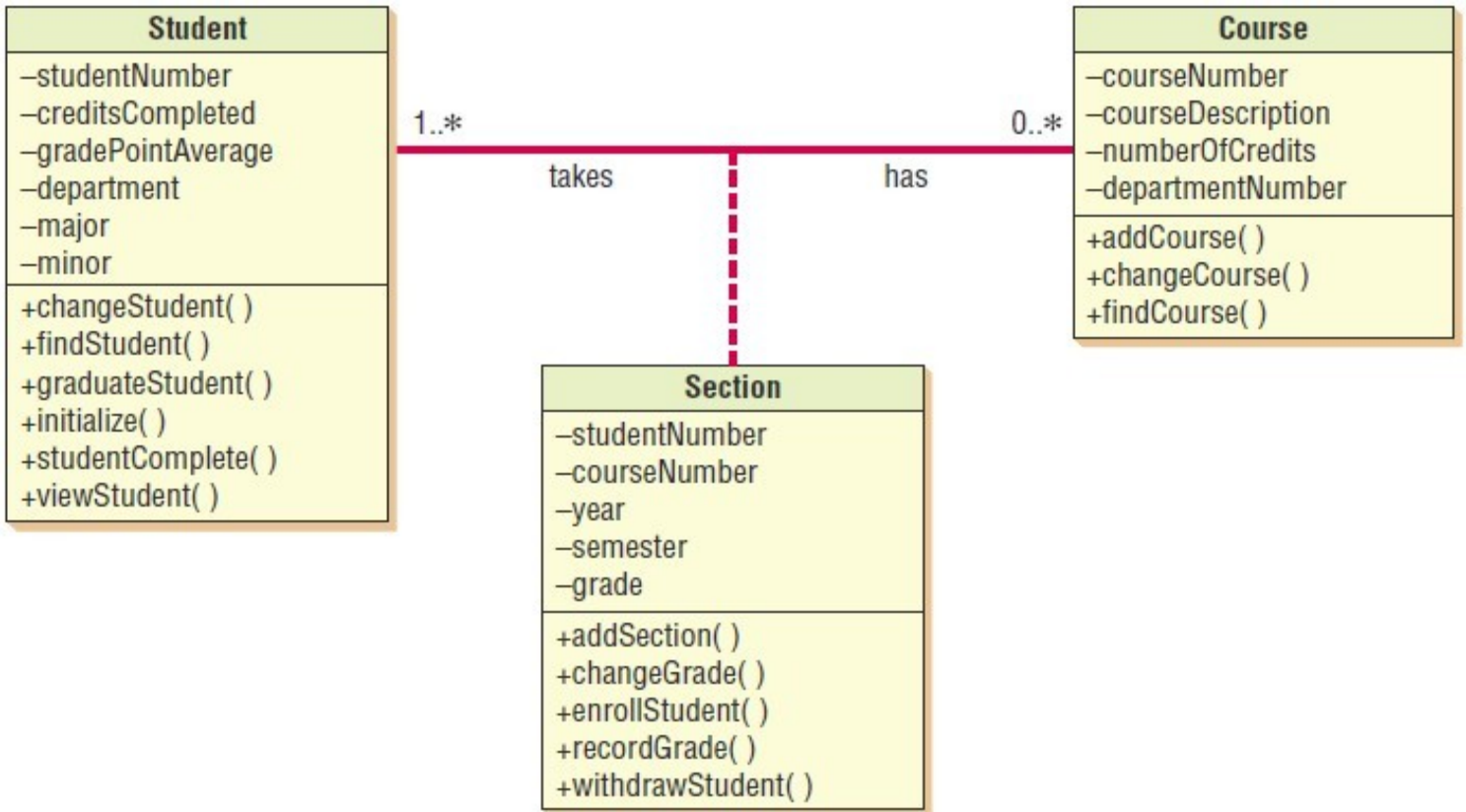
Association Classes

- An association class is used to model an association as a class. Association classes often occur in many-to-one and many-to- many associations where the association itself has attributes. It is used to break up a many-to-many association between classes.

- These are similar to associative entities on an entity-relationship
diagram.

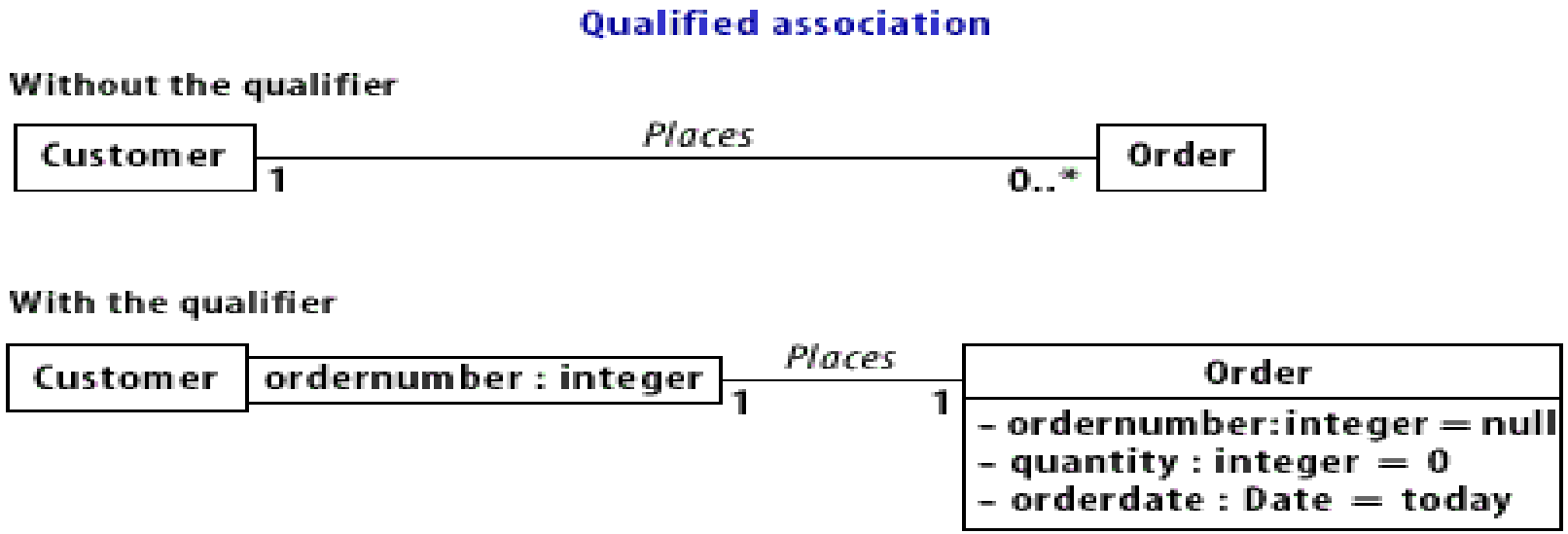
Association Classes

- **Student** and **Course** have a many-to-many relationship, which is resolved by adding an association class called **Section** between the classes of **Student** and **Course.** Figure illustrates an association class called **Section,** shown with a dotted line connected to the many-to-many relationship line.

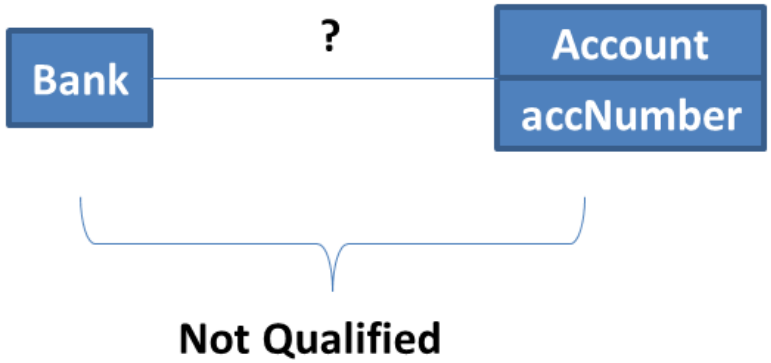


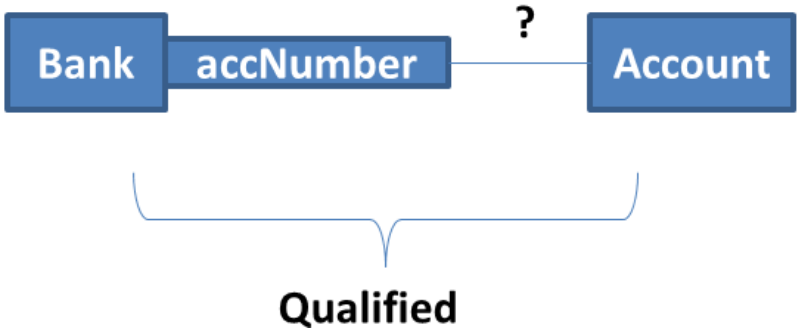
Qualified Associations

- It is an Association in which an attribute called qualifier disambiguates the objects for a “many” association end.
- A qualifier selects among target objects, reducing the effective multiplicity from “many” to “one” and specify a precise path for finding the target object from the source object.



Qualified Associations(2)





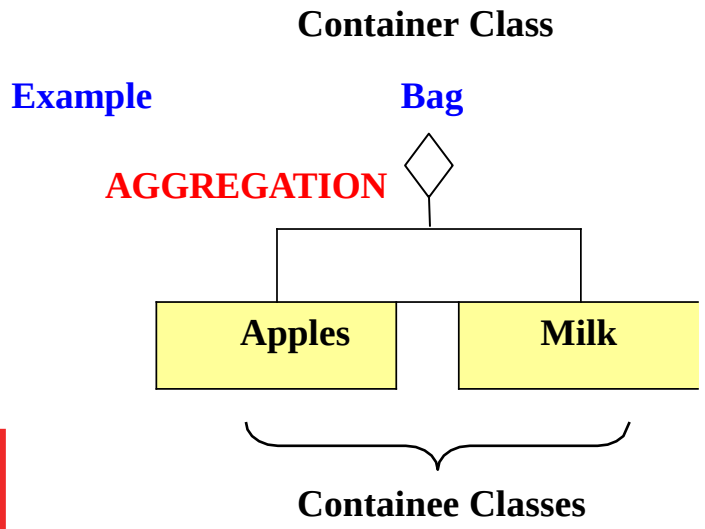
Whole/Part relationships

- Whole/part relationships are when one class represents the whole object and other classes represent parts.
- The whole acts as a container for the parts.
- These relationships are shown on a class diagram by a line with a diamond on one end.
- A whole/part relationship may be an entity object that has distinct parts, such as a computer system that includes the computer, printer, display, and so on, or an automobile that has an engine, brake system, transmission, and so on.
- Whole/part relationships have two categories: aggregation and composition.

AGGREGATION DESCRIBES A "HAS A" RELATIONSHIP

A customer has a address.
A car has a engine.
A bank has many bank accounts.
A university has many students.

Aggregation



[From Dr.David A. Workman]

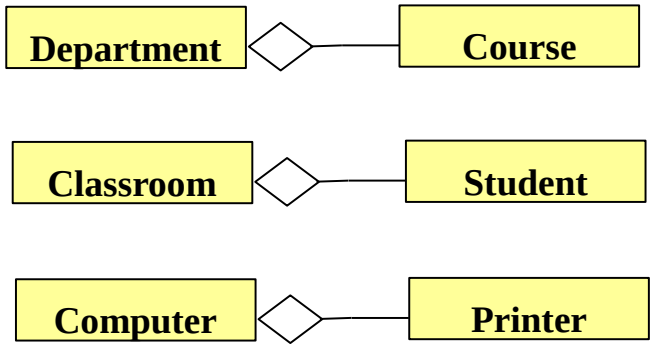
In the student enrollment example, the department *has a course* and the course is *for* a department. This is a weaker relationship, because a department may be changed or removed and the course may still exist, so independent from each other.

Aggregation:

described as a “has a” relationship..

Aggregation provides a means of showing that the whole object is composed of the sum of its parts (other objects).

Aggregation is appropriate when Container and Containees have no special access privileges to each other.

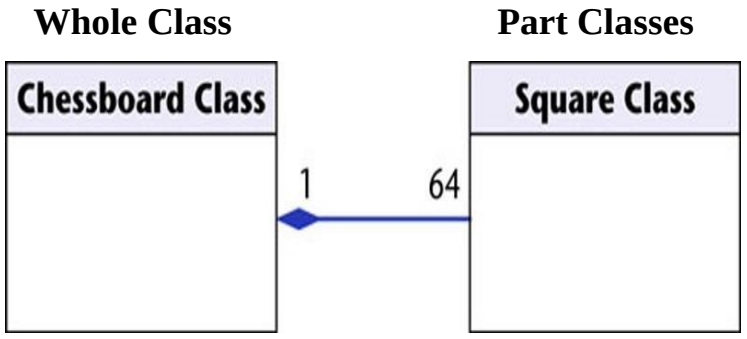


Whole/Part relationships

- **Composition**, a whole/part relationship in which the whole has a responsibility for the part, is a stronger relationship, shown with a filled-in diamond.
- Keywords for composition are one class “always contains” another class.
- If the whole is deleted, all parts are deleted.
- An example would be an insurance policy with riders. If the policy is canceled, the insurance riders are also canceled. In a database, the referential integrity would be set to delete cascading child records. In a university there is a composition relationship between a course and an assignment as well as between a course and an exam. If the course is deleted, assignments and exams are deleted as well.

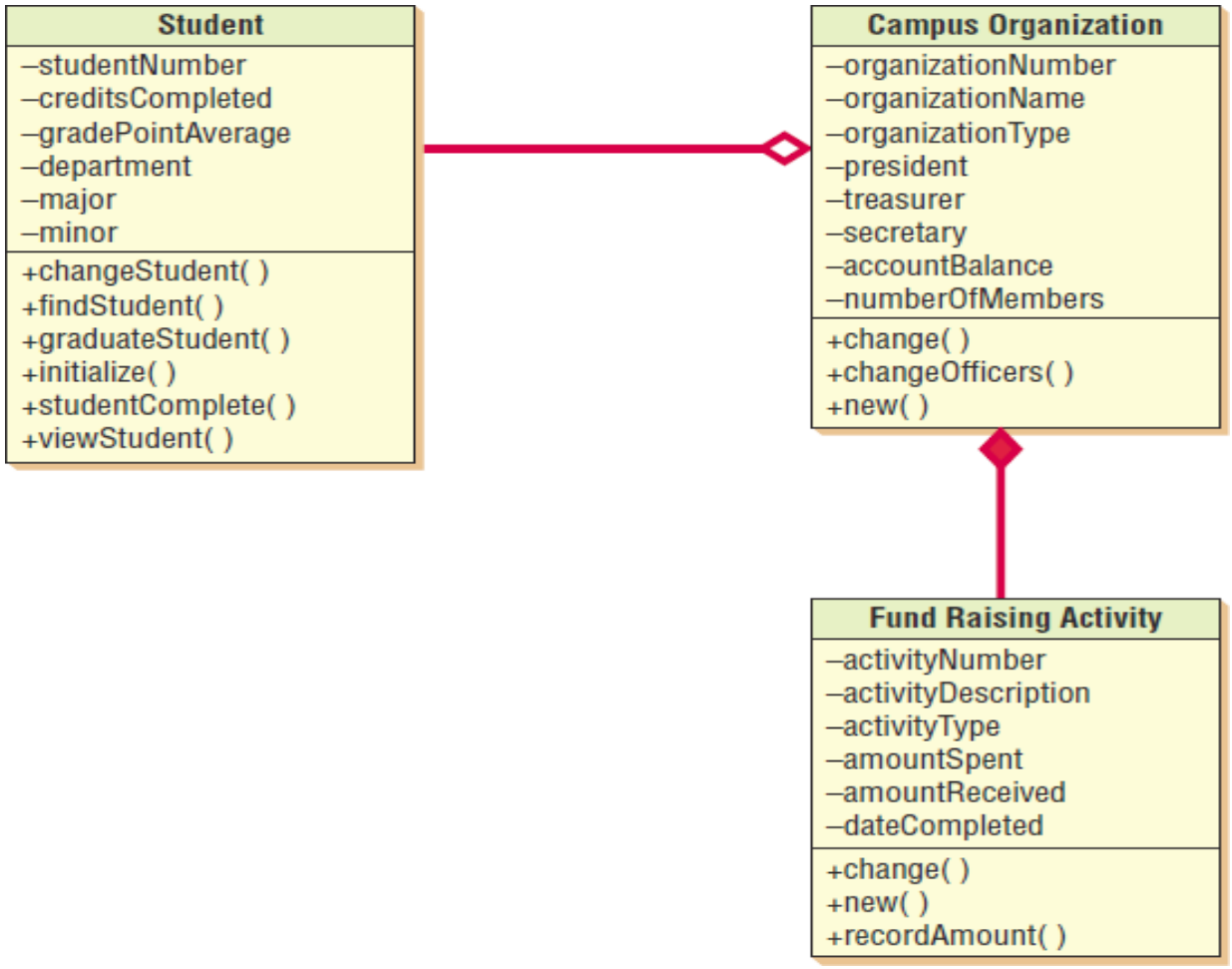
Example

A number of different chess boards: Each square belongs to only one board. If a chess board is thrown away, all 64 squares on that board go as well.



[From Dr.David A. Workman] The McGraw-Hill Companies,

OO Relationships



Aggregation vs. Composition

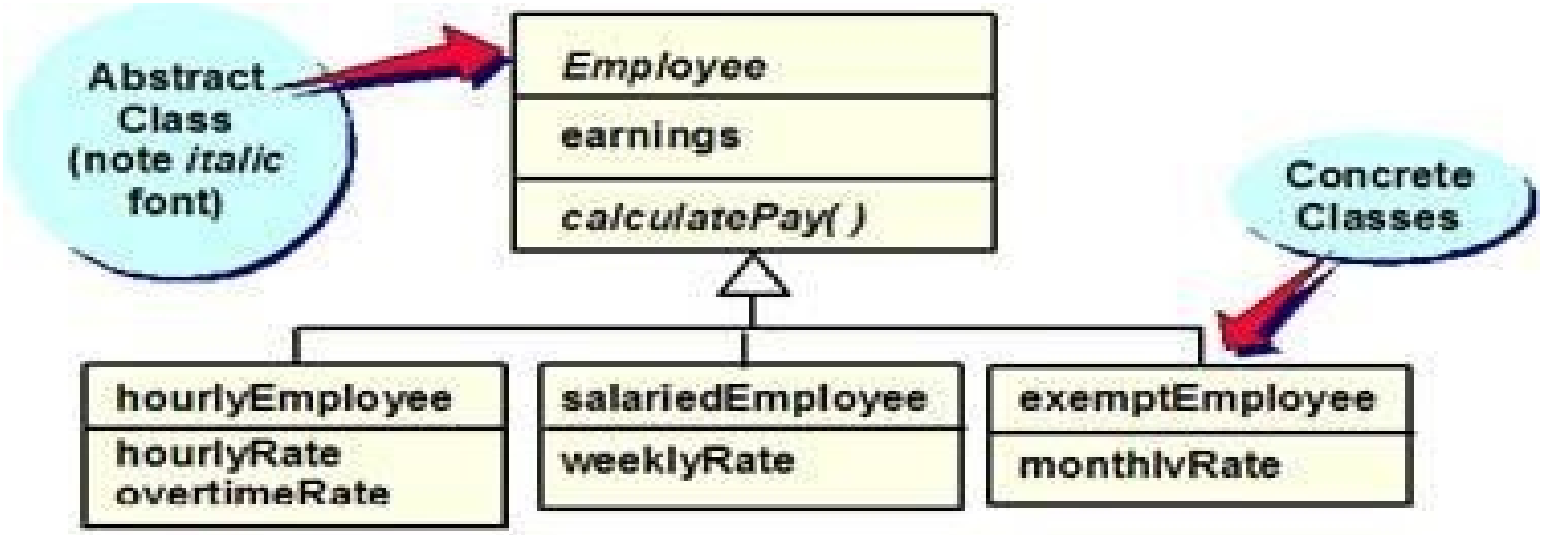
- **Composition** is really a strong form of **association**
  - components have only one owner
  - components cannot exist independent of their owner
  - components live or die with their owner
  - e.g. Each car has an engine that can not be shared with other cars.

- **Aggregations**

may form "part of" the association, but may not be essential to it. They may also exist independent of the aggregate. e.g. Apples may exist independent of the bag.

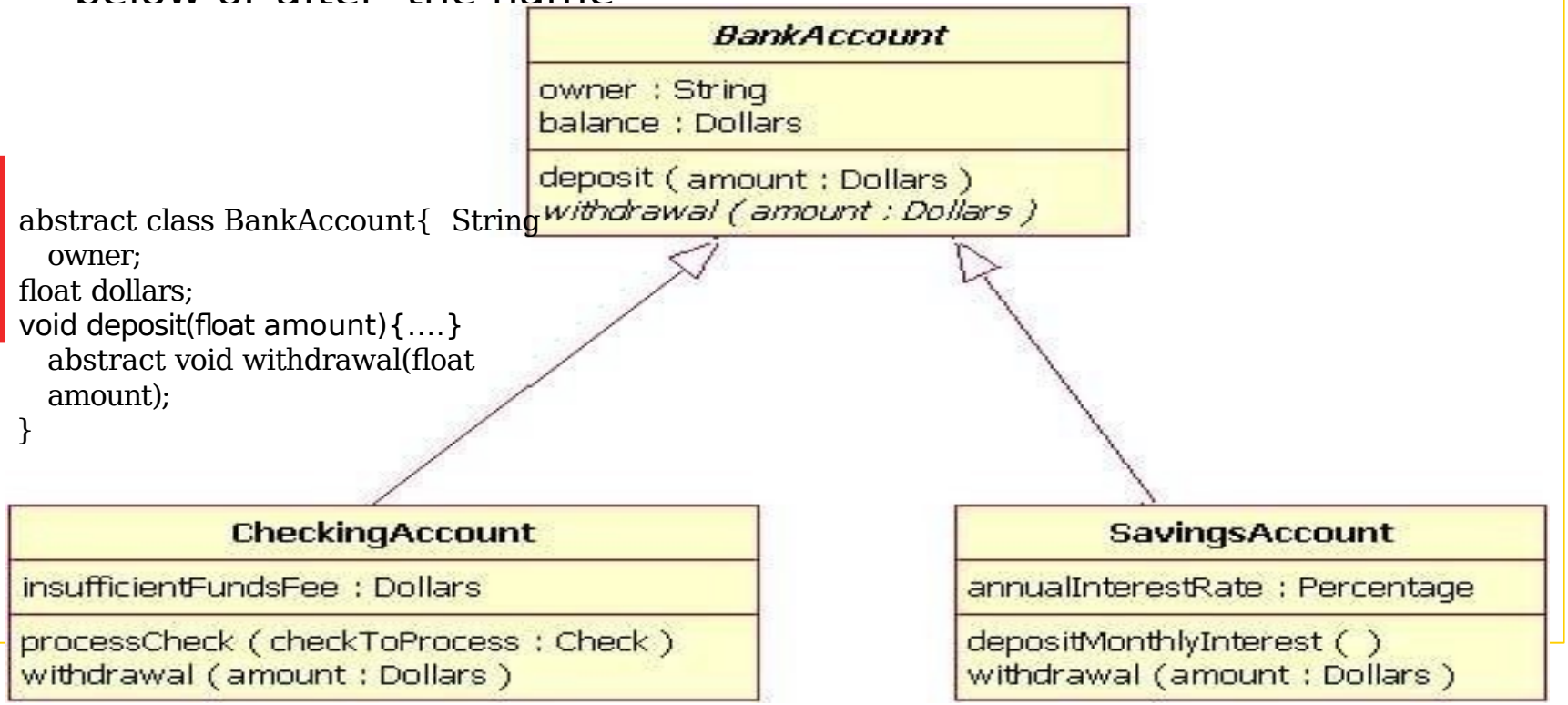
Concrete Class

- Instantiable – which can have direct instances.
- It may have abstract subclasses – But in turn must have concrete descendants
- Only concrete classes may be leaf classes in an inheritance tree.



Abstract Class

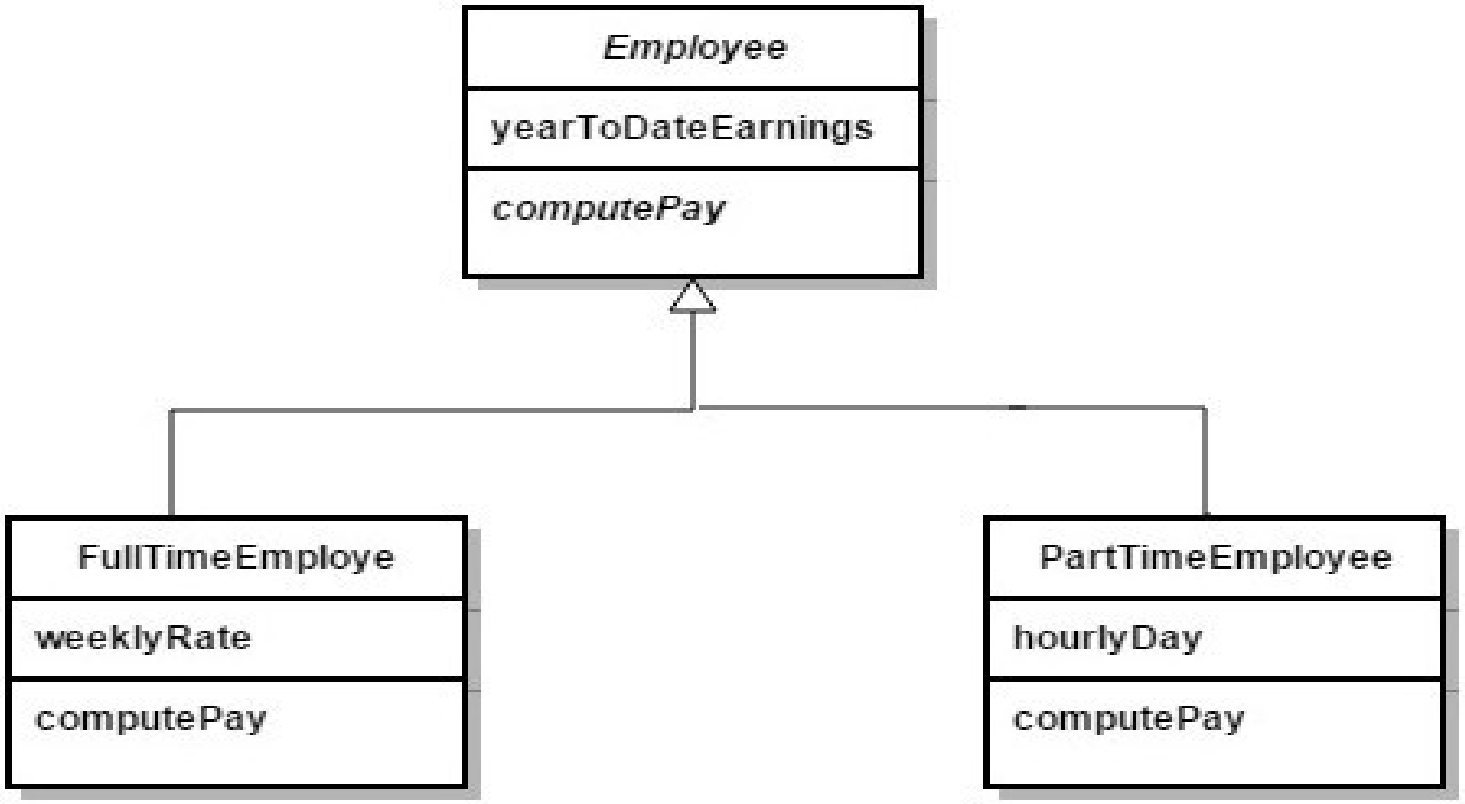
- No direct instances but whose descendant classes have direct instances.
- Represented by italicized text or place keyword {abstract} below or after the name




```
abstract class BankAccount{   String
   owner;
float dollars;
void deposit(float amount){….}
   abstract void withdrawal(float
   amount);
}
```

Abstract Class(2)

- Program Example :



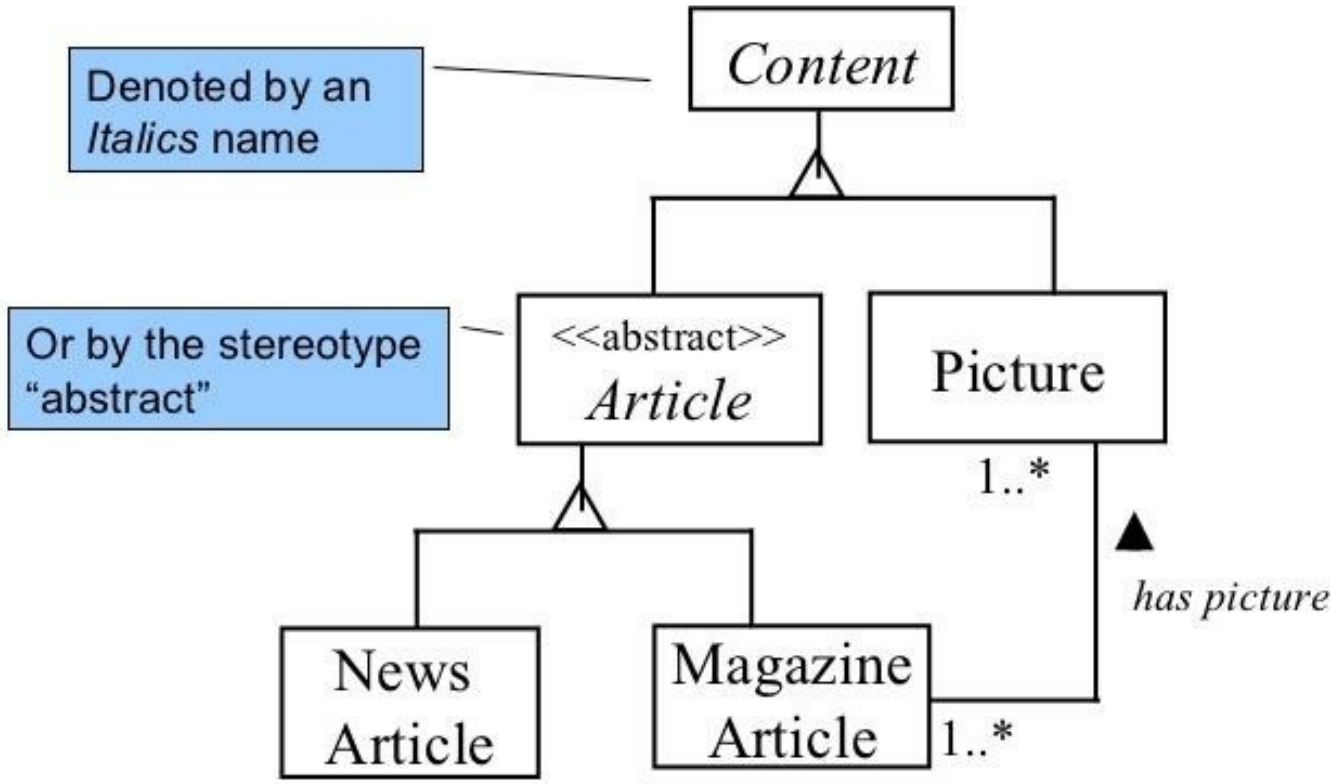
Abstract Class

setInterval.html | Hide_Show_mm.html | color_data.html | JQueryFadeIn.html | SlideDown.html | Animate.html | jquery-2.1.1.js | J1.html | String.txt | ABC.java

```
1  package javaapplication1;
2
3  abstract class Shape{  // Abstract class - can not be instantiated
4      abstract void draw();
5      void welcome(){
6          System.out.print("Welcome!!!");
7      }
8  }
9  class Rectangle extends Shape{   // Concrete class
10     @Override
11     void draw(){
12         System.out.println("drawing rectangle");}
13 }
14
15 abstract class Rect extends Shape{  // Abstract class as doesn't implement draw() method
16     void display(){
17         System.out.println("Display Info...");
18     }
19 }
20
21 class Circle extends Shape{  // Concrete class
22     @Override
23     void draw(){
24         System.out.println("drawing circle");}
25 }
26
27 class AbstractDemo{
28     public static void main(String args[]){
29         Shape s=new Circle();
30         s.draw();
31         s = new Rectangle();
32         s.draw();
33     }
34 }
```

Abstract Class

- A class that has no direct instances

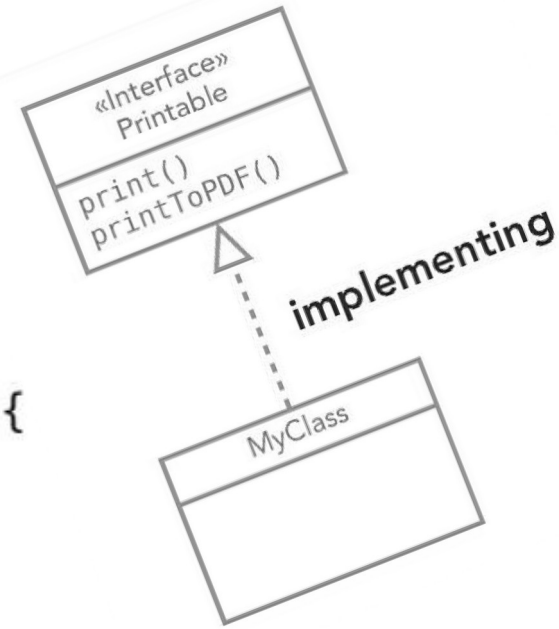


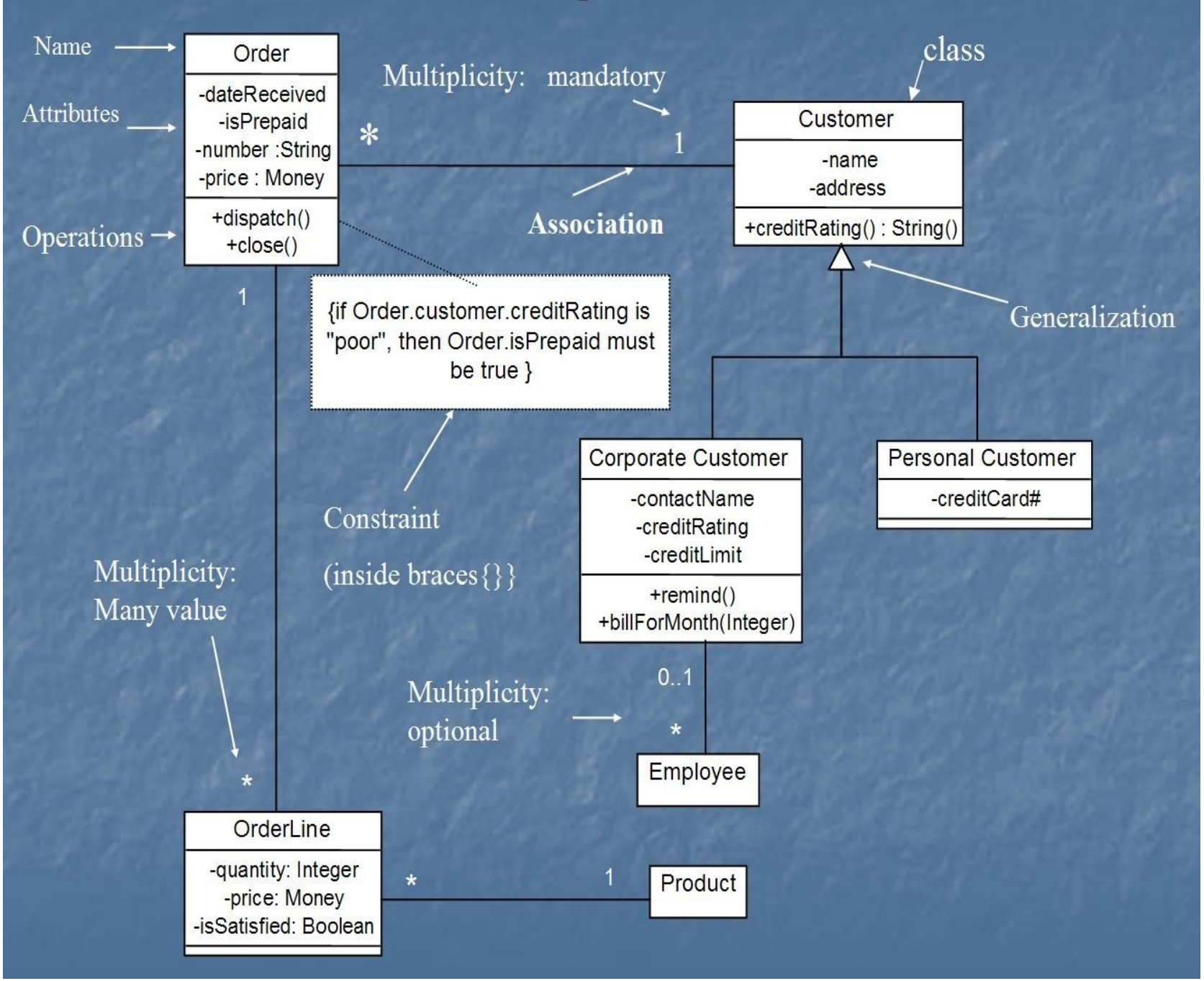
DEFINING INTERFACE CONTRACTS

```
interface Printable {

    // method signatures
    void print();
    void printToPDF(String filename);

}
```

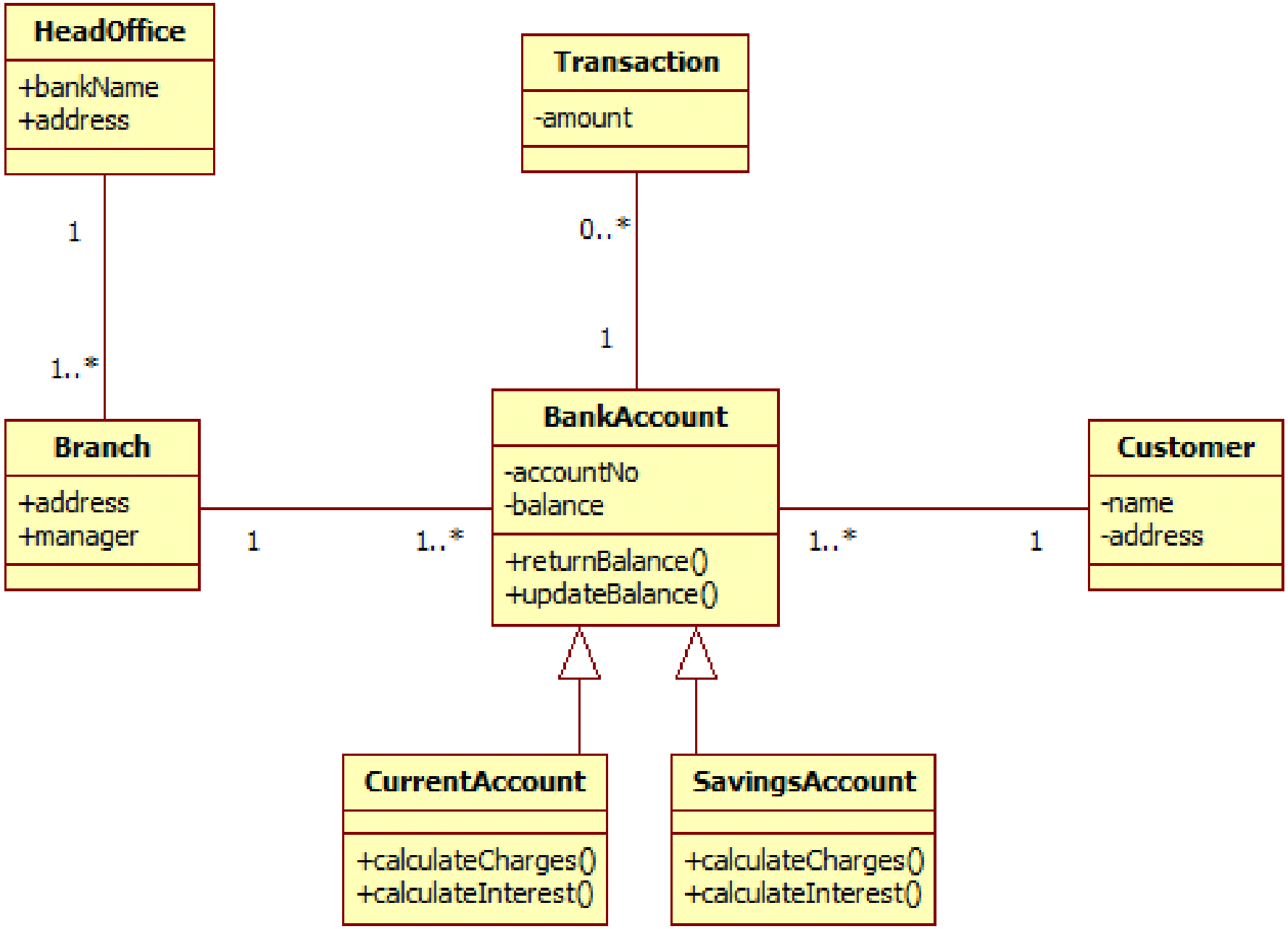
IMPLEMENTING INTERFACES

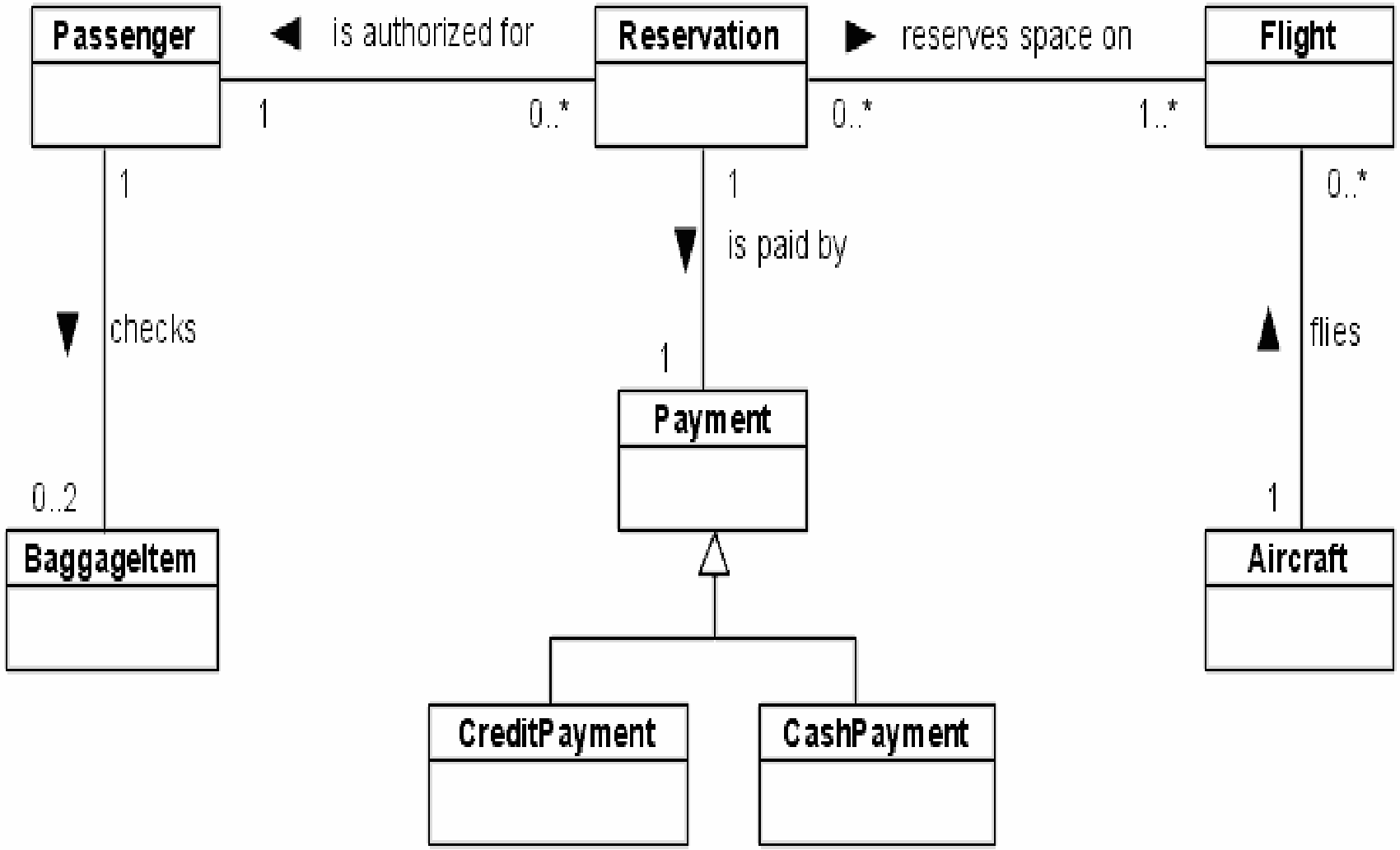
```
class MyClass implements Printable {


    // method bodies
    public void print() {
        // provide implementation
    }

    public void printToPDF(String filename) {
        // provide implementation
    }

   // additional functionality...

}
```

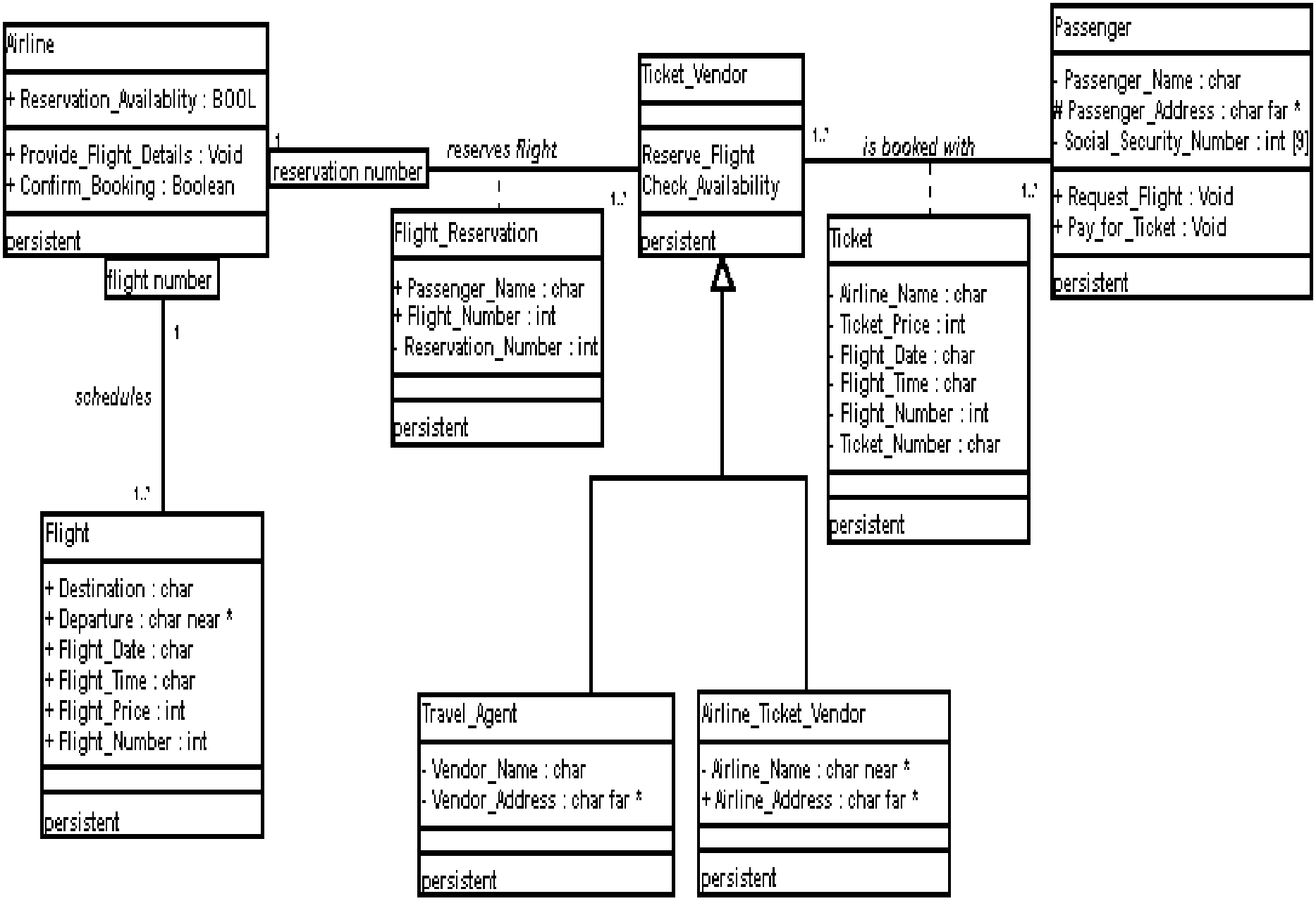




[from *UML Distilled Third Edition*]


HeadOffice
+bankName
+address
Transaction
-amount
1
0..*
1..*
1
Branch
+address
+manager
BankAccount
-accountNo
-balance
+returnBalance()
+updateBalance()
Customer
-name
-address
1
1..*
1..*
1
CurrentAccount
+calculateCharges()
+calculateInterest()
SavingsAccount
+calculateCharges()
+calculateInterest()


Passenger
is authorized for
Reservation
reserves space on
Flight
1
0..*
0..*
1..*
1
checks
0..2
BaggageItem
1
is paid by
1
Payment
0..*
flies
1
Aircraft
CreditPayment
CashPayment


Airline
+ Reservation_Availablity : BOOL
+ Provide_Flight_Details : Void
+ Confirm_Booking : Boolean
persistent
1
reservation number
reserves flight
1..*
Flight_Reservation
+ Passenger_Name : char
+ Flight_Number : int
- Reservation_Number : int
persistent
Ticket_Vendor
Reserve_Flight
Check_Availability
persistent
1..*
is booked with
1..*
Ticket
- Airline_Name : char
- Ticket_Price : int
- Flight_Date : char
- Flight_Time : char
- Flight_Number : int
- Ticket_Number : char
persistent
Passenger
- Passenger_Name : char
Passenger_Address : char far *
- Social_Security_Number : int [9]
+ Request_Flight : Void
+ Pay_for_Ticket : Void
persistent
flight number
1
schedules
1..*
Flight
+ Destination : char
+ Departure : char near *
+ Flight_Date : char
+ Flight_Time : char
+ Flight_Price : int
+ Flight_Number : int
persistent
Travel_Agent
- Vendor_Name : char
- Vendor_Address : char far *
persistent
Airline_Ticket_Vendor
- Airline_Name : char near *
+ Airline_Address : char far *
persistent

USE CASE Diagrams

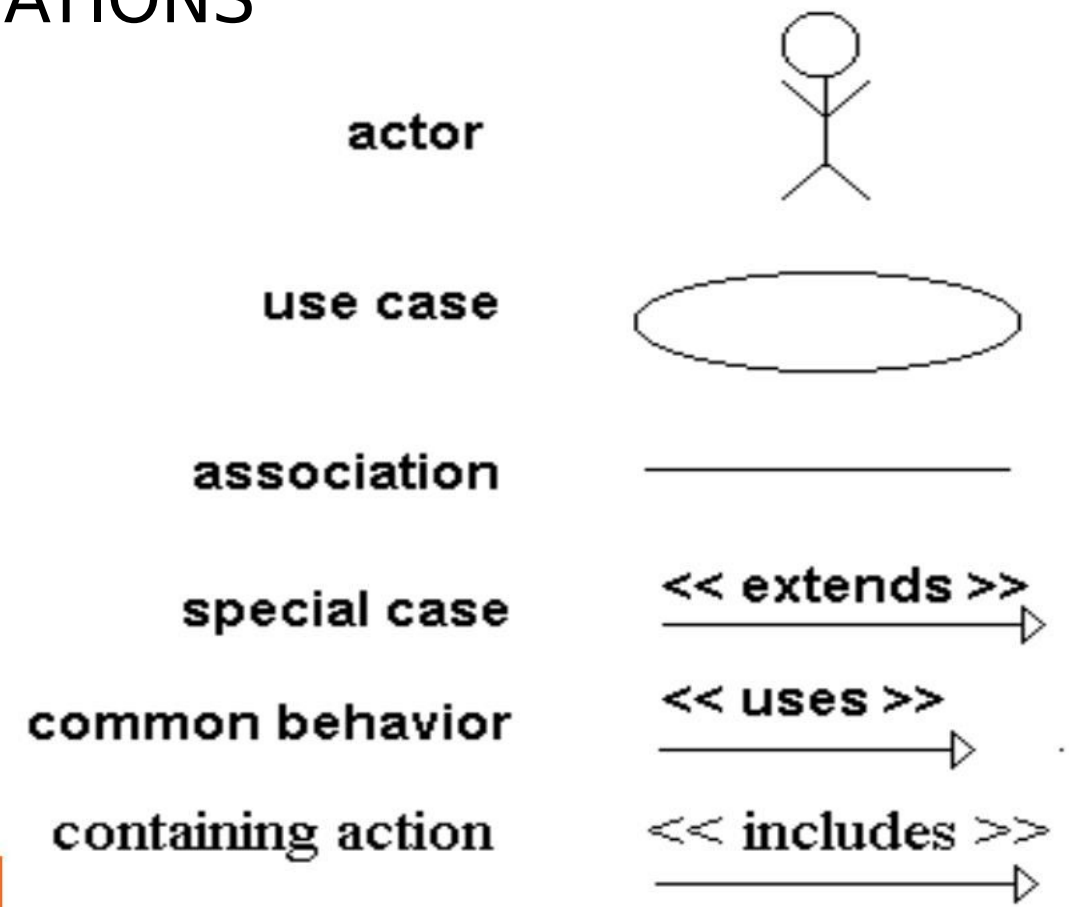
USE CASE

- Use case is a list of steps, typically defining interactions between a role (actor) and a system, to achieve a goal. The actor can be a human, an external system, or time.
- Actor: An actor is something with behavior, such as a person, computer system, or organization.
- Scenario: A scenario is a specific sequence of actions and interactions between actors and the system under discussion; it is also called a use case instance. It is one particular story of using a system, or one path through the use case; for example, the scenario of successfully using an ATM machine to withdraw cash.

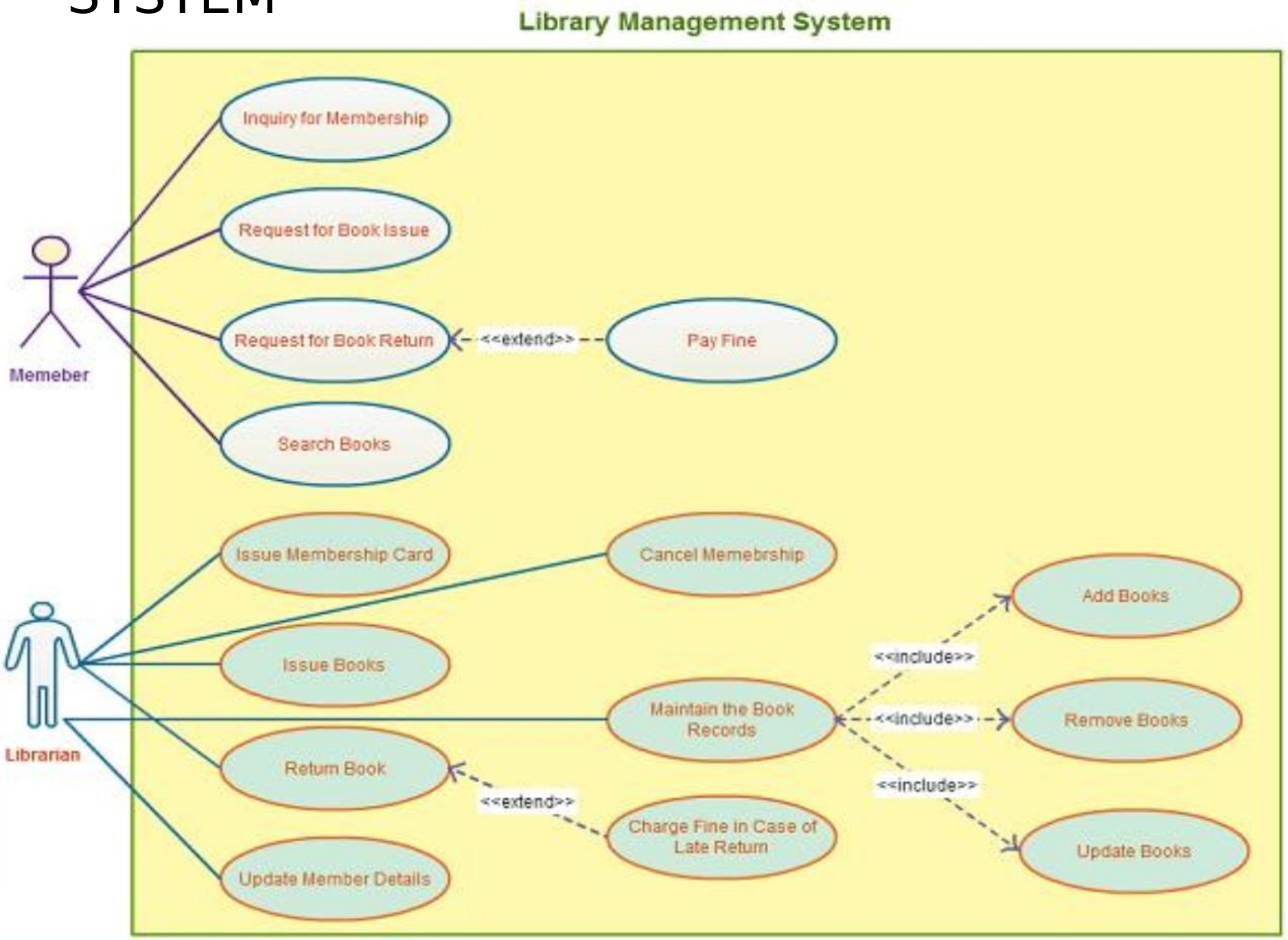
WHAT USE CASE CAN/CAN'T DO?

- What Use Cases Do?
  1. They hold Functional Requirements in an easy to read and tracking format.
  2. They represent the goal of an interaction between an actor and the system.
  3. They are multi-level, one use case can use/extent the functionality of another.
- What Use Cases Do Not Do?
  1. They don't specify user interface design. They specify the intent, not the action Detail.
  2. They don't specify implementation detail.

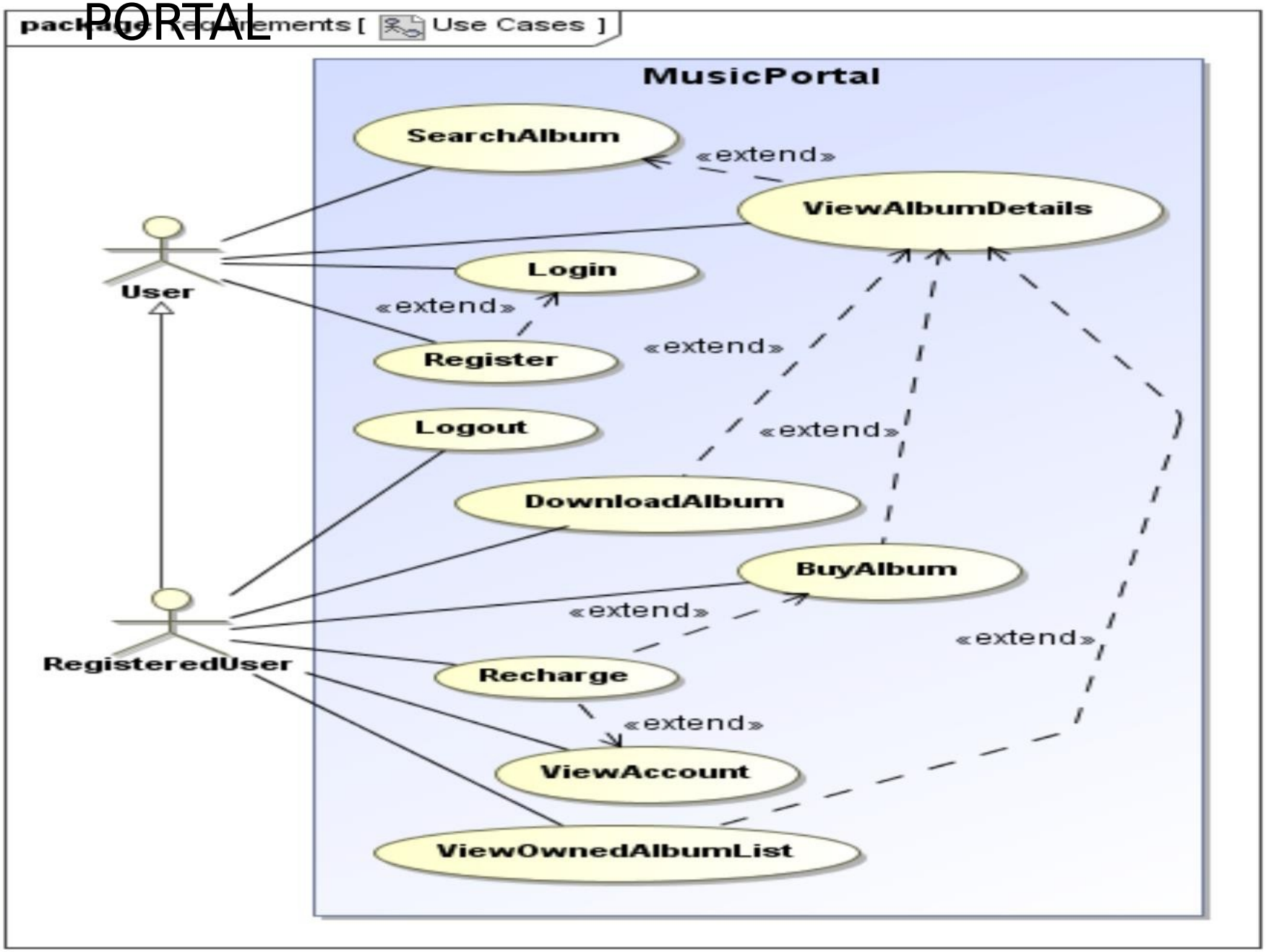
USE CASE: NOTATIONS



USE CASE: LIBRARY MANAGEMENT SYSTEM

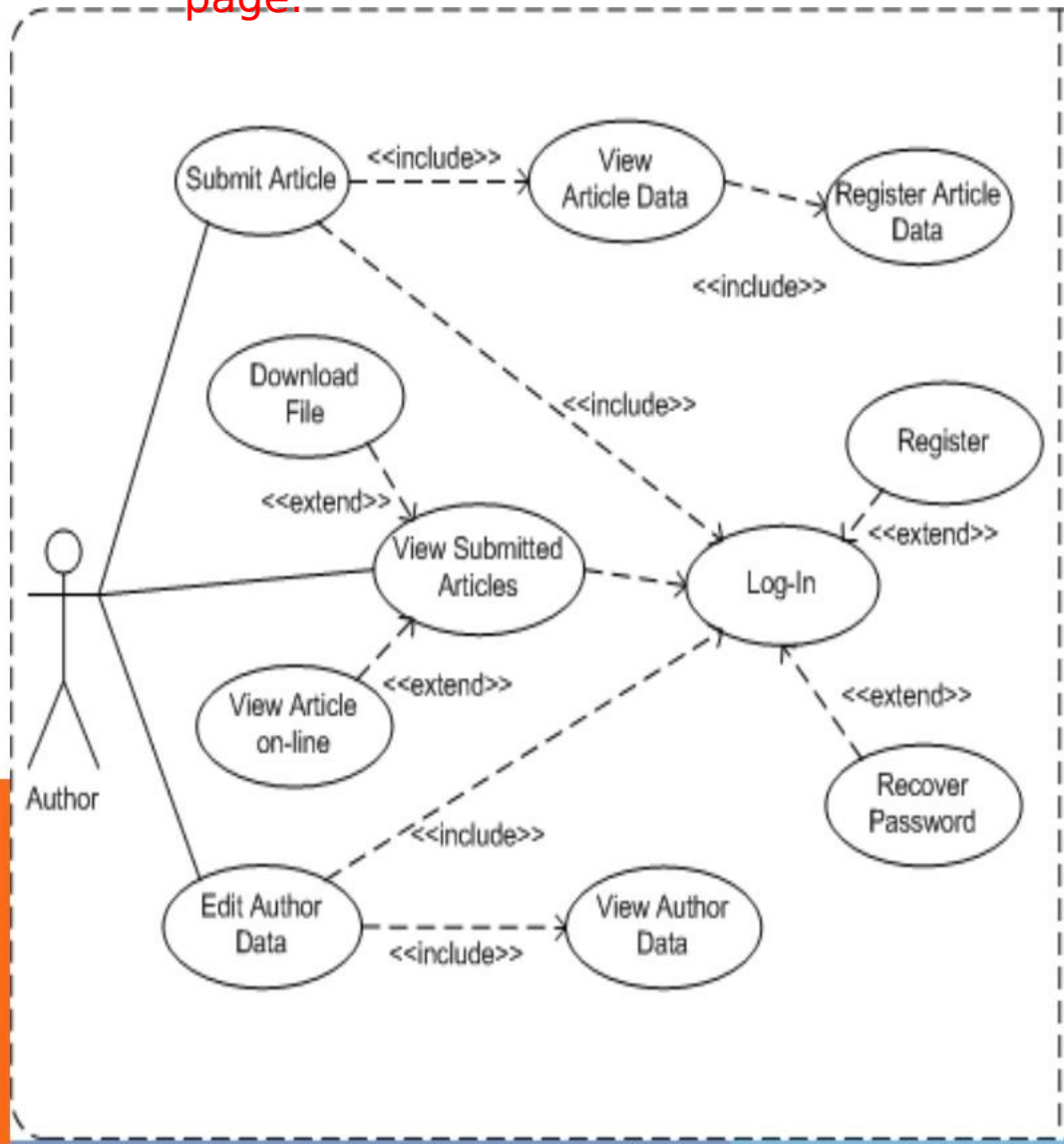


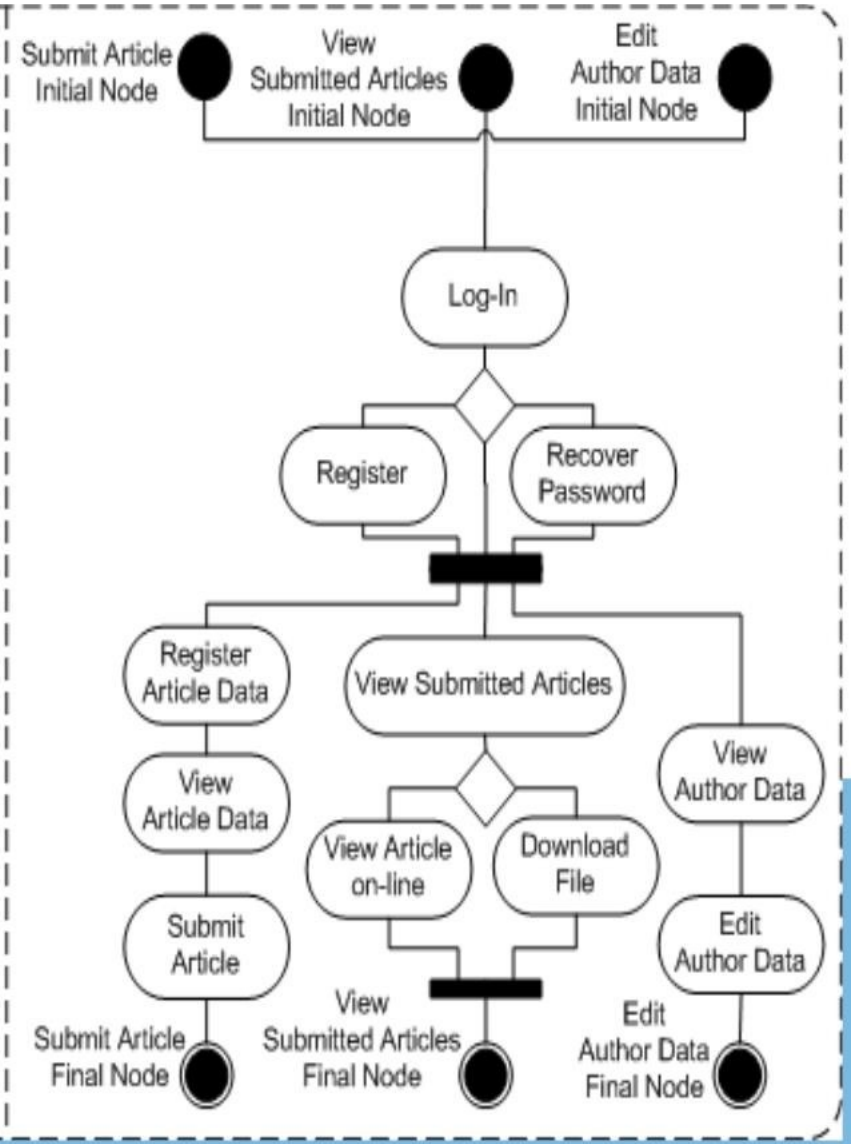
USE CASE: MUSIC PORTAL



USE CASE: CONFERENCE MANAGEMENT SYSTEM

Explanation is on the next page:



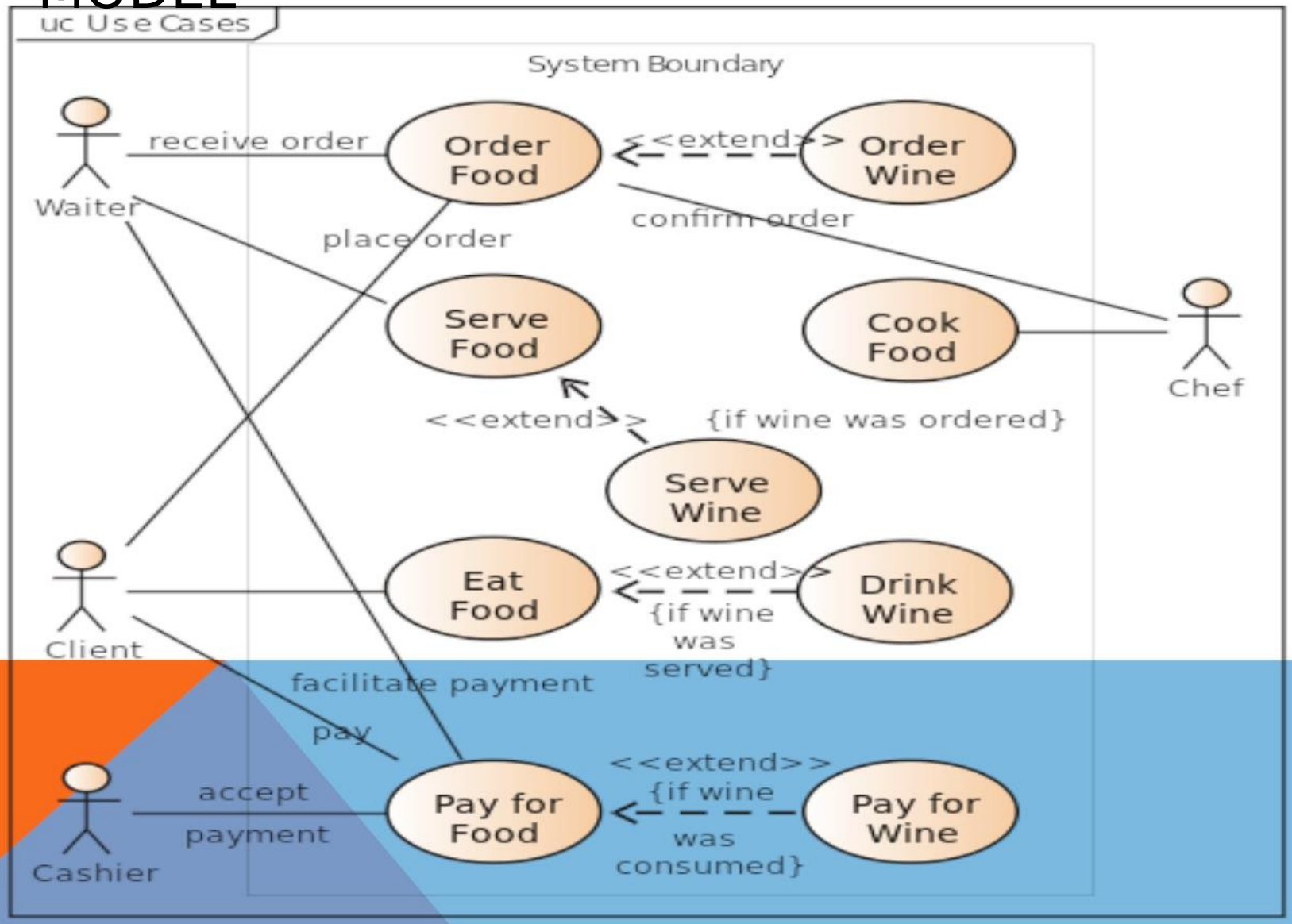


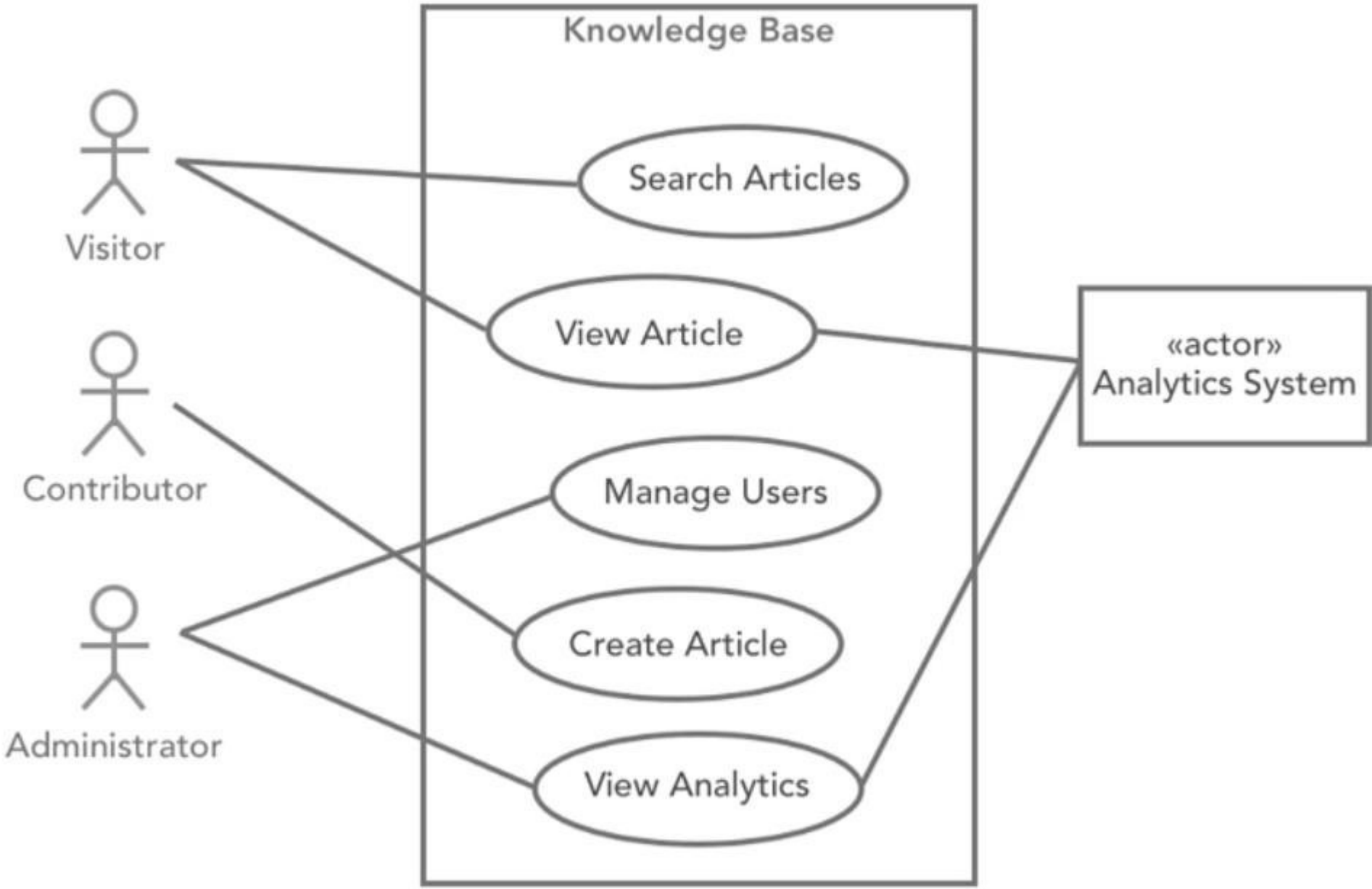
USE CASE: CONFERENCE MANAGEMENT SYSTEM

- In our example, the conference management system offers three different services: Submit an Article, Show the submitted articles and Edit the data of an author. In order to provide with this complex services, some basic services are needed, as the Log-In or the Register ones. To model the relation between the different Use Cases two types of associations are used: *<<include>>* and *<<extend>>*. The former implies that the behavior of the included Use Case is inserted in the including Use Case, while the later specifies how and when the behavior defined in the extending use case can be inserted into the behavior defined in the extended use case.

- On the other hand, in the Service Process model each complex service identified in the previous model is mapped to an activity and the basic as services that it uses are represented service activities. This way, back to the example we have three different activities that use a set of service activities. For instance the Log-In one is used by the three activities.

USE CASE: RESTAURANT MODEL




Knowledge Base
Visitor
Contributor
Administrator
Search Articles
View Article
Manage Users
Create Article
View Analytics
«actor»
Analytics System

SEQUENCE Diagrams

SEQUENCE MODELS

- **The Sequence models elaborates the themes of Use cases, it captures use-case behavior using the foundation of classes.**
- **Two kinds of sequence Models:**
  - Scenario
  - Sequence Diagrams
- **Scenario:** Scenario = sequence of Events
  - **A scenario is a sequence of events occurs during execution of a system-for a use case. It can be historical record of the system or a thought experiment. It can be displayed as a list of text statements.**
  - **Scenario contains messages between objects as well as activities performed by objects.**
  - **To write scenario:**
    - Identify the objects exchanging messages
    - Determine sender and receiver of message
    - Sequence of message

SEQUENCE MODELS - SCENARIO

- **Scenario Example: Online shopping System**

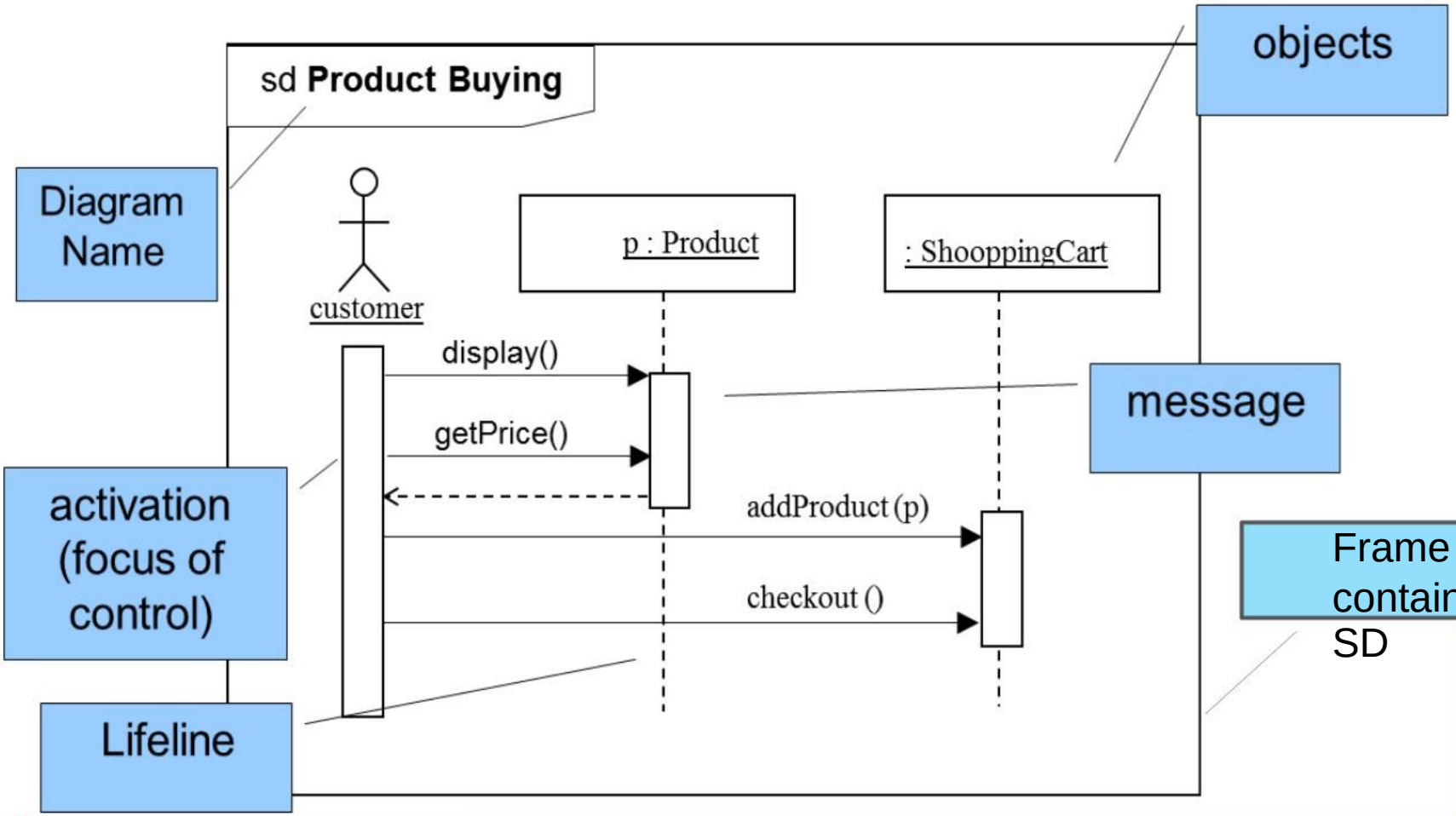
Viral logs in.
System establsihes secure connetion.
System displays different available products.
Viral enters purchase order for 5 books.
System verifies sufficient funds for purchase.
System displays confirmation screen with estimated cost.
Viral confirms order.
Viral enters transaction details.
System displays transaction tracking number.
Viral logs out.
System establishes insecure connection.
System displays good-bye screen.

Scenario for online shopping System

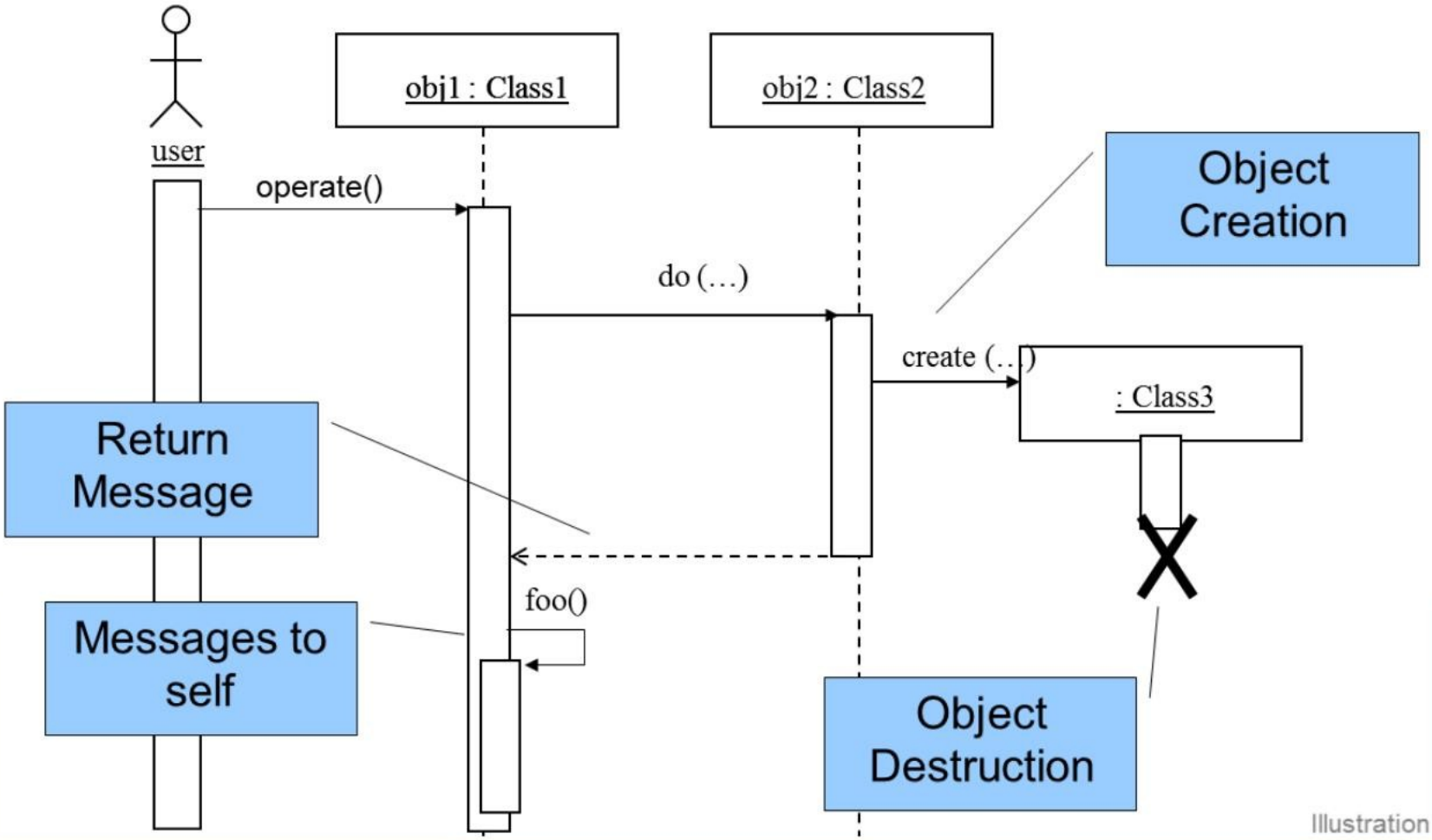
SEQUENCE MODELS : SEQUENCE DIAGRAM

- **If there are more than two objects in interaction, “Sequence Diagram” gives more clarity than scenario.**
- **A Sequence Diagram shows the interaction of a system with its actors to perform all parts of use case along with the sequence of messages among them.**
- **Sequence Diagram(make a phone call)**
- **Understand the overall flow of control in an object-oriented program, a good design has lots of small methods in different classes and at times it can be tricky to figure out the overall sequence of behavior.**

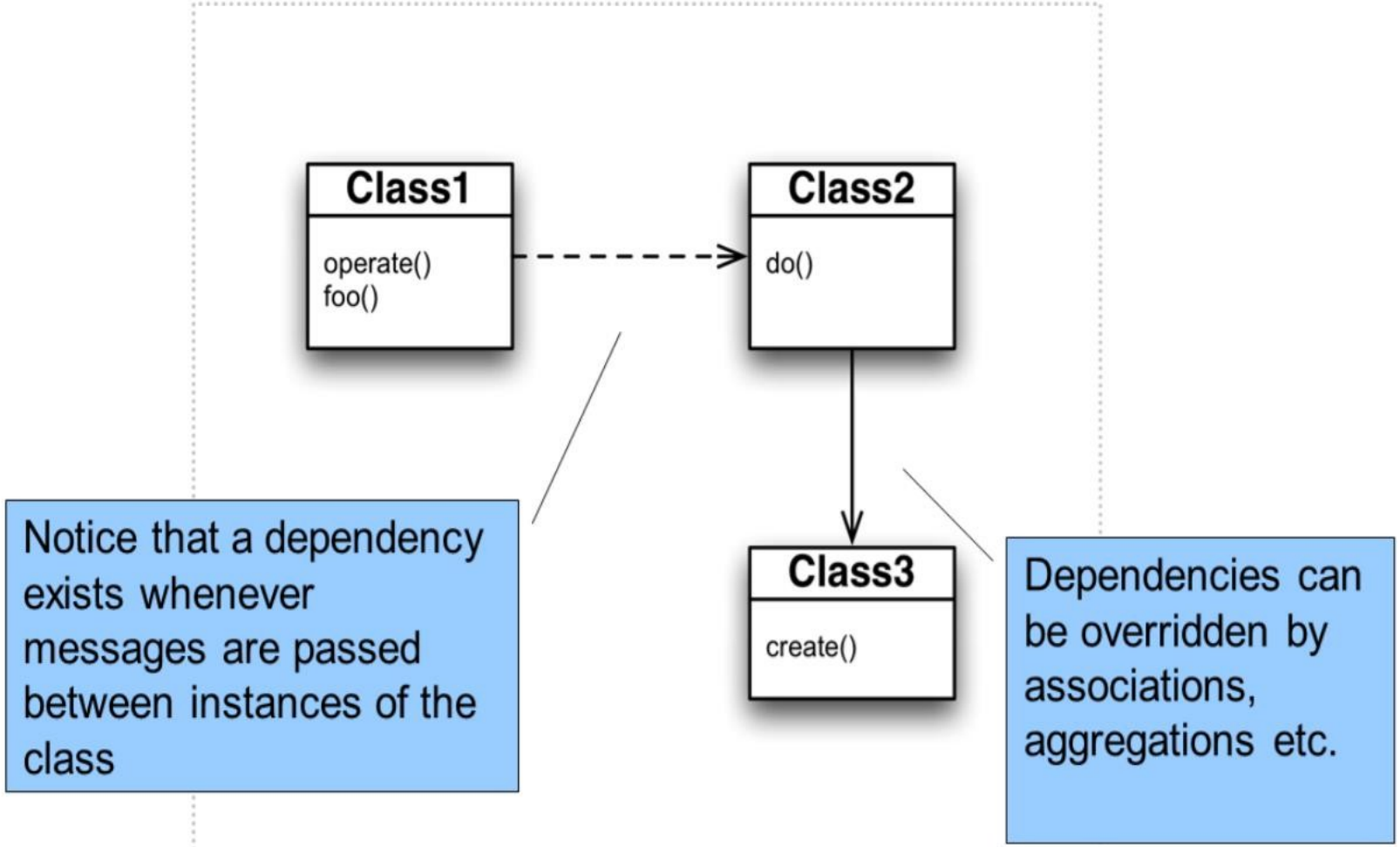
SEQUENCE DIAGRAM (BUYING A PRODUCT)



SEQUENCE DIAGRAM (OBJECT CONTROL)

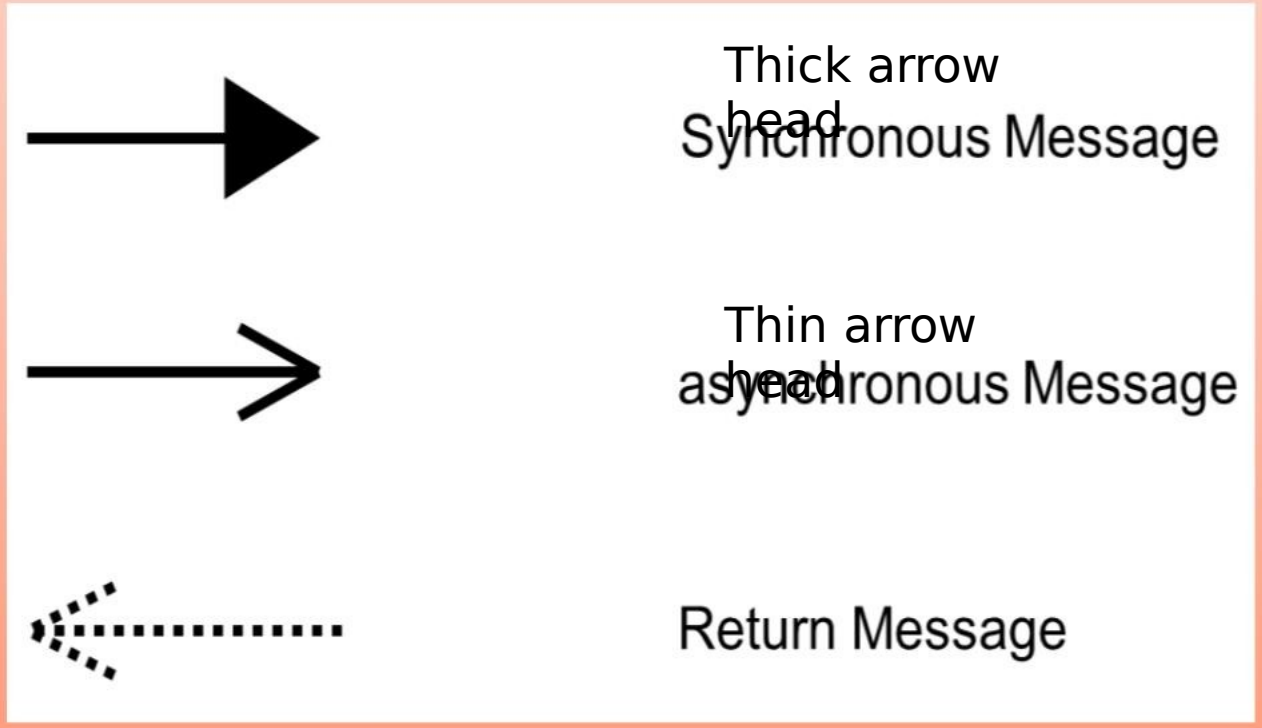


CORRESPONDING CLASS DIAGRAM

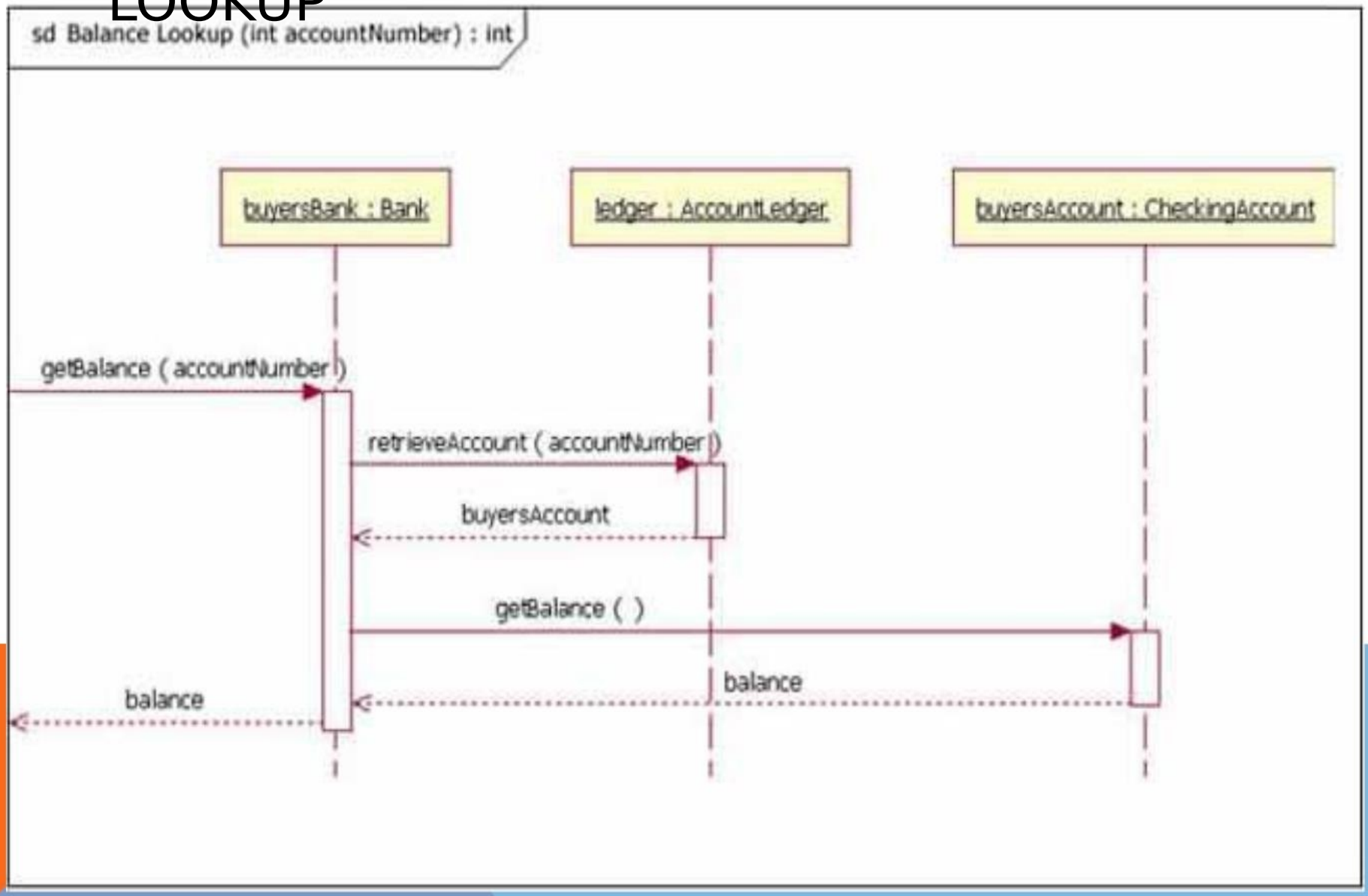


SEQUENCE DIAGRAM : OBJECT INTERACTION

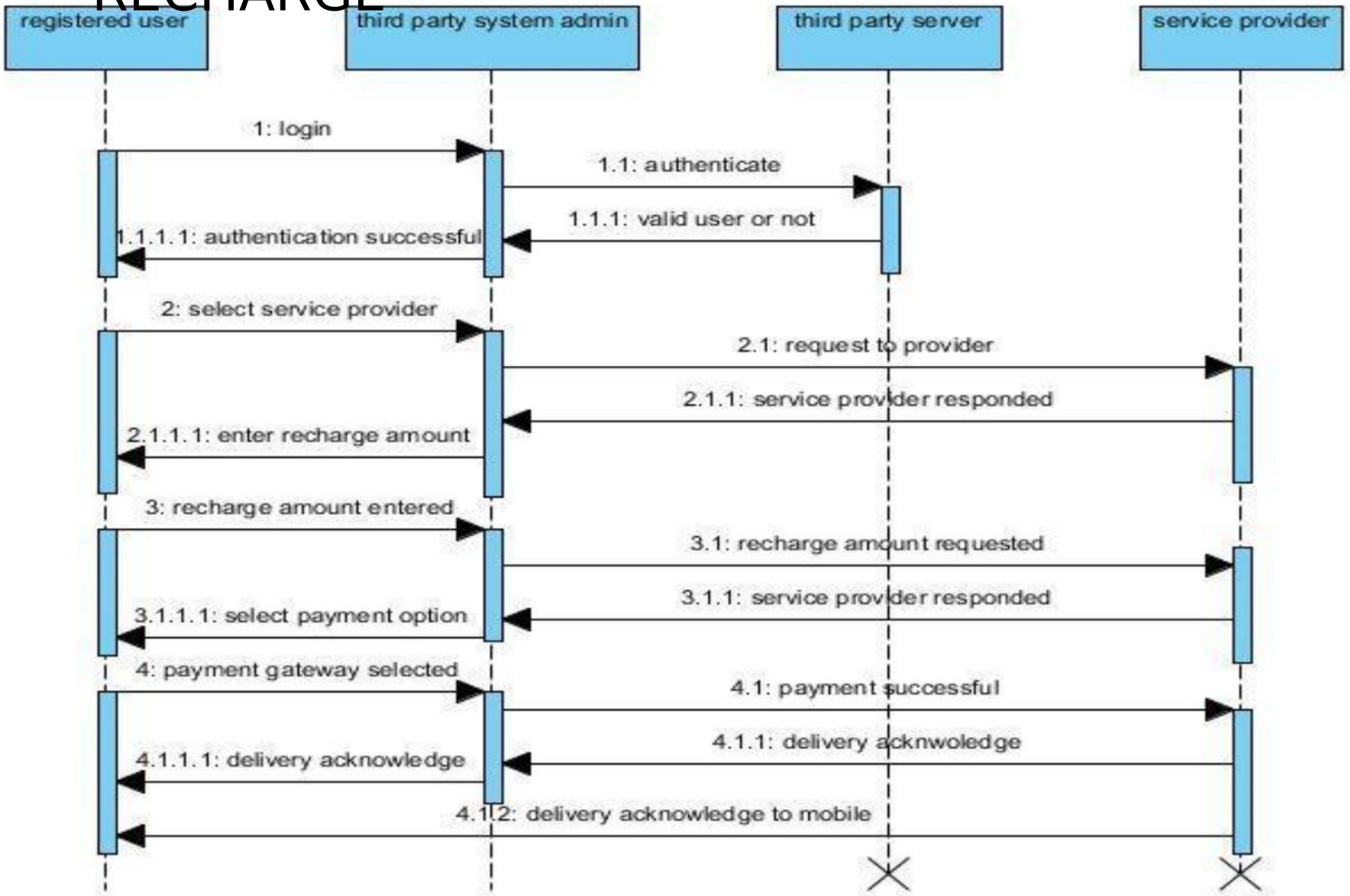
- **During interaction, three types of messages can be passed:**
  - **Synchronous : Sender will wait till receiver reply – Reply is required**
  - **Asynchronous : Sender continue with next activity to be performed**
  - **Recursive : Object calls itself : Factorial Example**



SEQUENCE DIAGRAM : BALANCE LOOKUP



SEQUENCE DIAGRAM : ONLINE MOBILE RECHARGE



SEQUENCE DIAGRAM : EXERCISE/ ASSIGNMENT

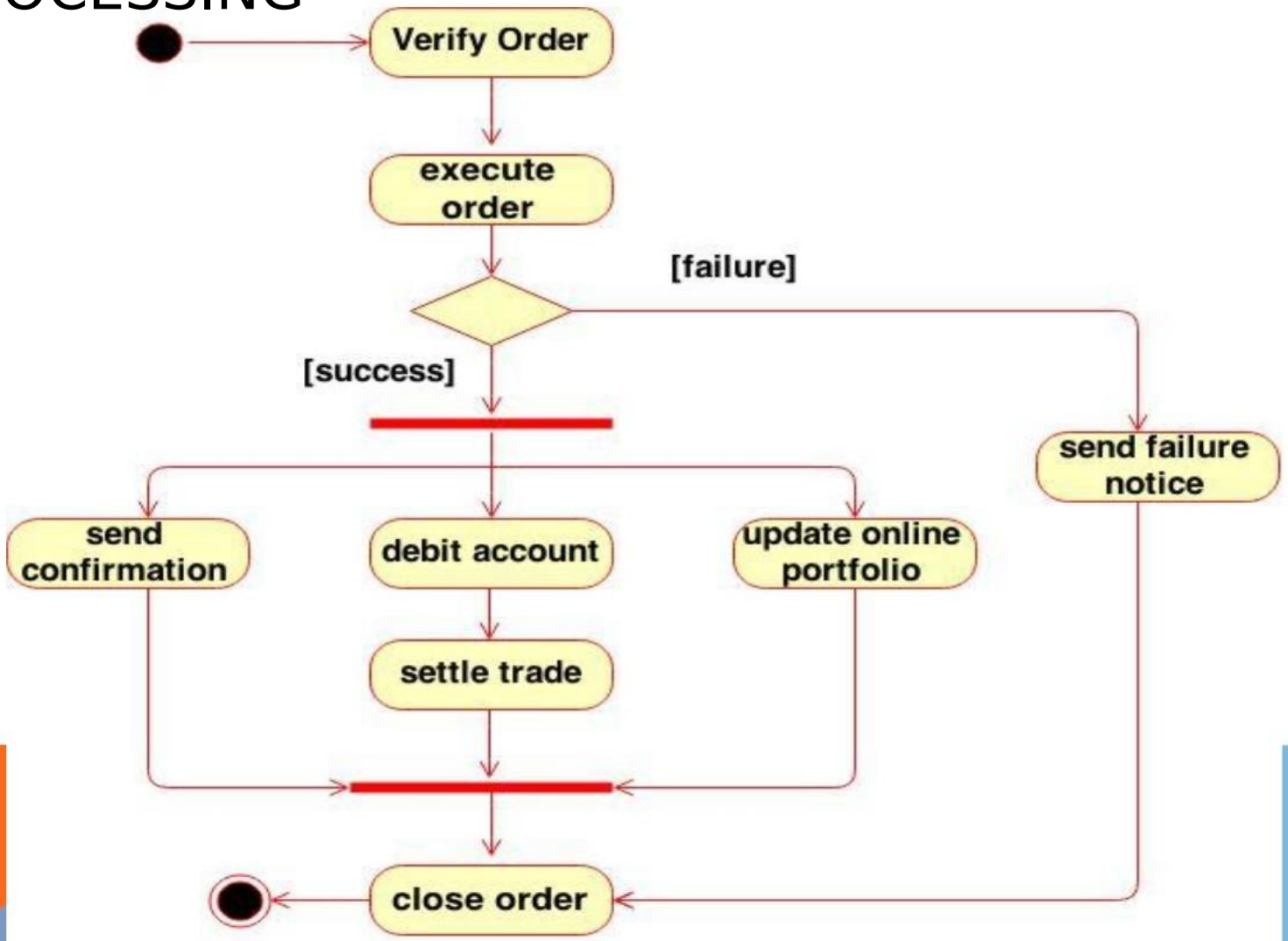
- **Prepare an sequence diagrams for the following Systems:**
  1. **Order food in Restaurant**
  2. **Deposit money in Bank Account**
  3. **Money Transfer from one Account to another Account**
  4. **Balance Inquiry of Savings Bank Account**
  5. **Random Password Generation**

ACTIVITY DIAGRAM

ACTIVITY MODELS: ACTIVITY DIAGRAM

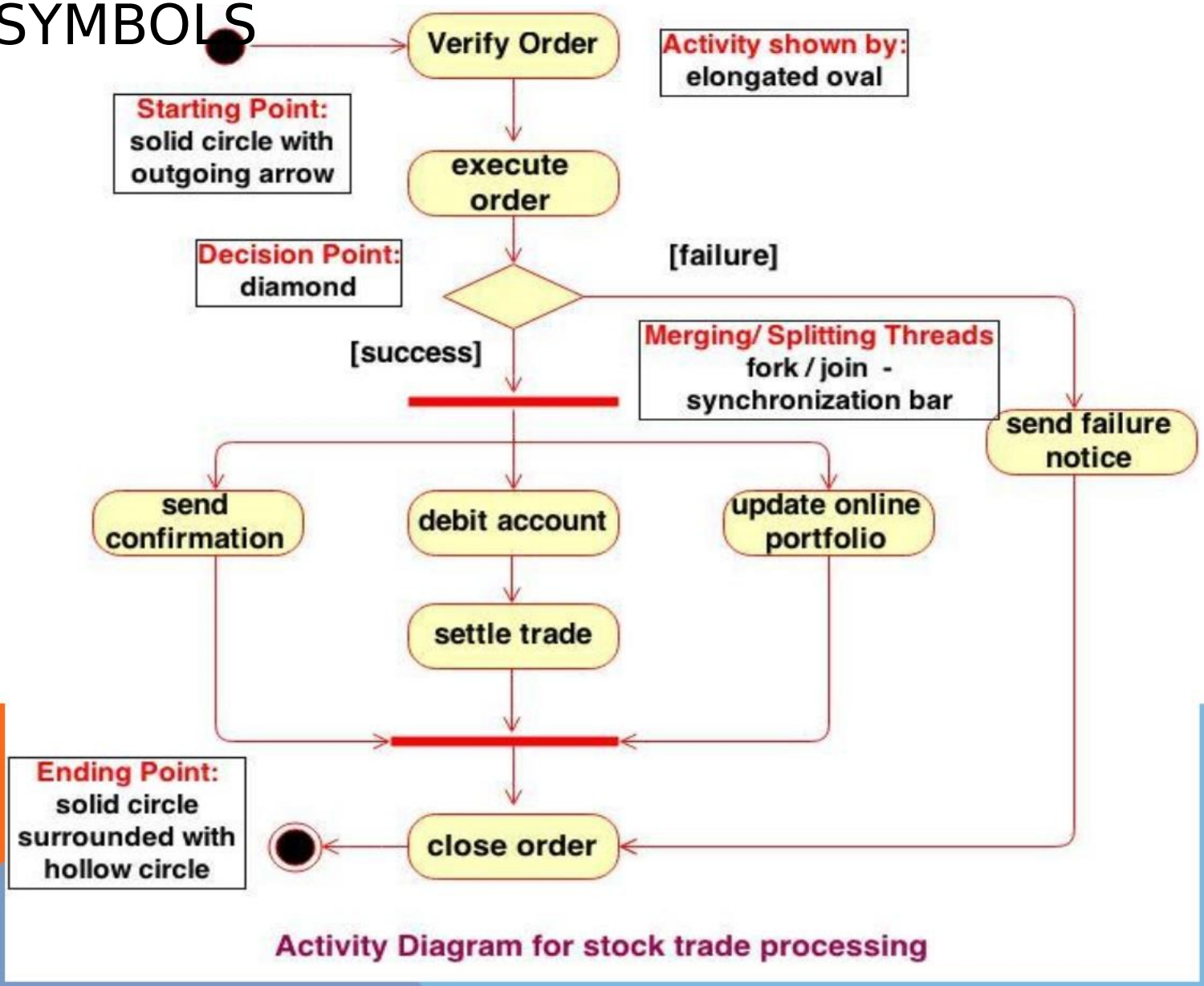
- **It shows the sequence of steps – flow of control - that make up a complex process**
- **Similar to sequence diagrams, but focuses on operations rather than objects**
- **Most useful during the early stages of designing workflows**
- **An unlabeled arrow from one activity to another in an activity diagram indicates – first activity must complete before the second activity can begin**
- **Similar to flowcharts but, Unlike traditional flow charts – computer/organizations can perform more than one activity at a time and pace of the activity can also change over time.**
- **Ex:**
  - One activity may be followed by another – **sequential control**
  - Split into several concurrent activities – **fork of control**
  - Combine into single activity – **join/merge of control**

ACTIVITY DIAGRAM : EX –STOCK TRADE PROCESSING

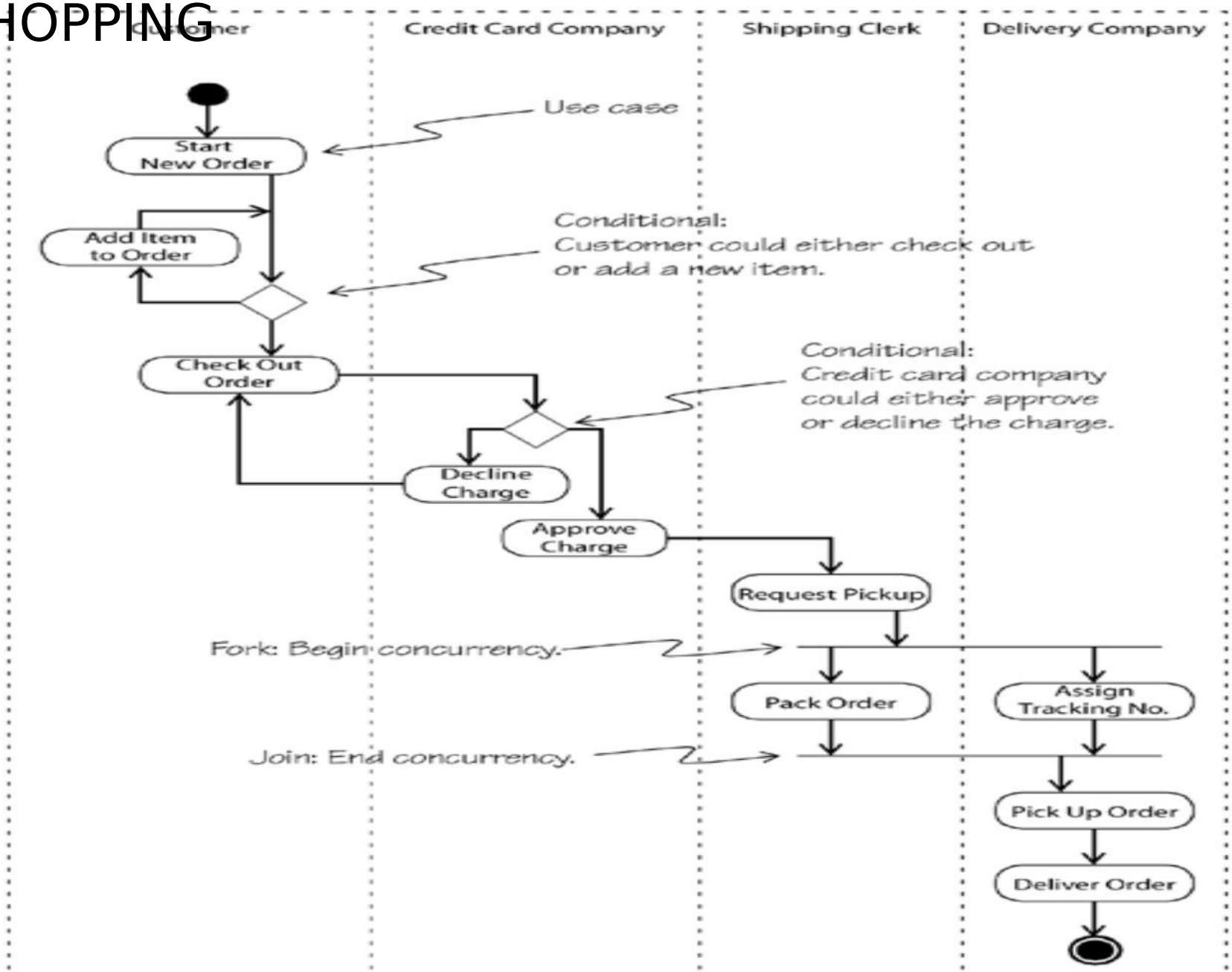


Activity Diagram for stock trade processing

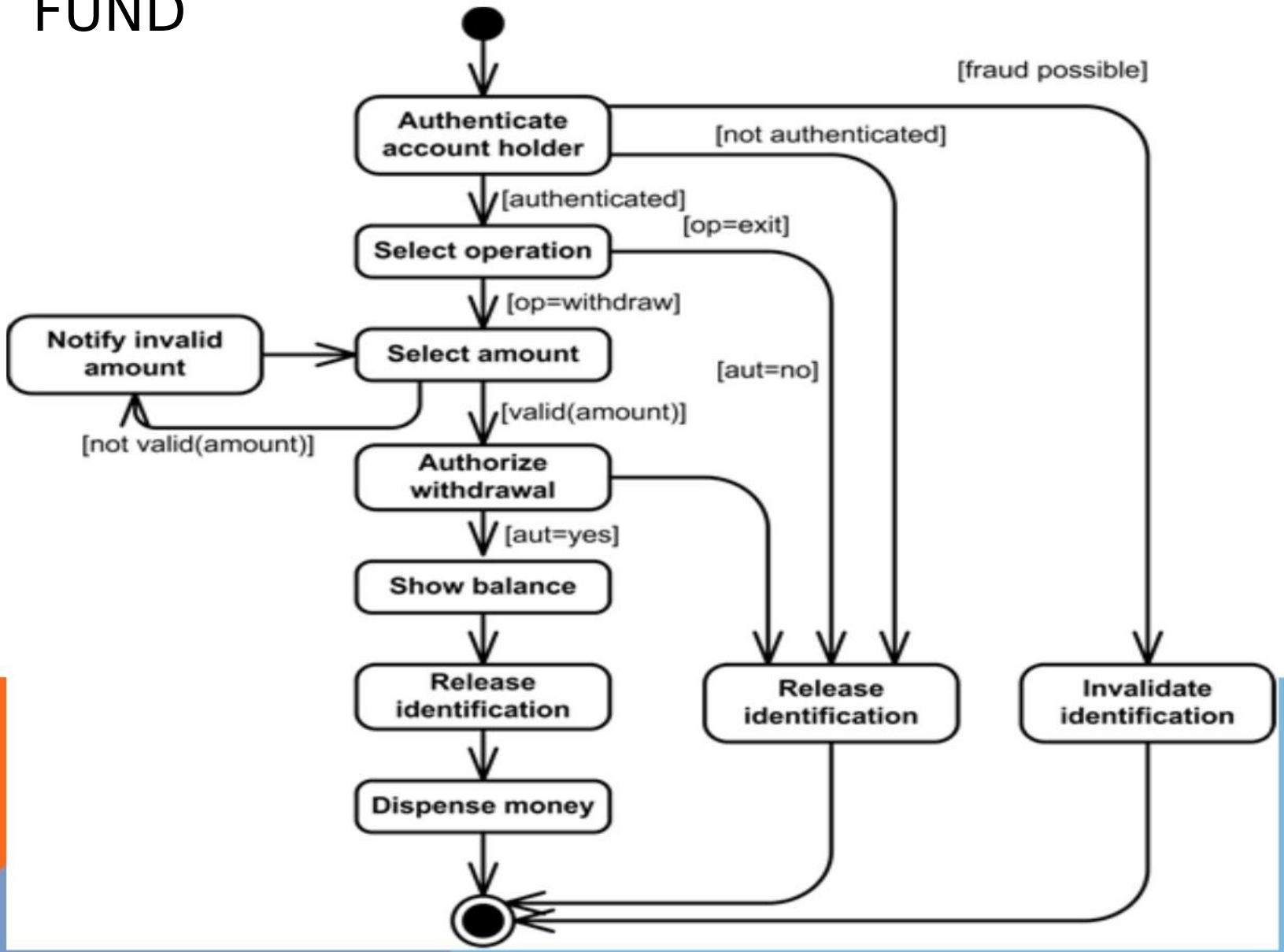
ACTIVITY DIAGRAM : EXAMPLE - SYMBOLS



ACTIVITY DIAGRAM : EXAMPLE – CREDIT CARD SHOPPING



ACTIVITY DIAGRAM : EXAMPLE – WITHDRAW FUND



EXECUTABLE ACTIVITY DIAGRAMS

- **Activity diagrams are not only useful for defining the steps in a complex process, but they can also be used to show the progression of control during execution.**
- **An *activity token* is placed on an activity symbol to indicate that it is executing.**
- **When an activity completes, the token is removed and placed on an outgoing arrow and then moves to next activity.**
- **Multiple tokens can arise through concurrency – split of control**

ACTIVITY DIAGRAM : EXERCISE/ ASSIGNMENT

- **Prepare an activity diagram for Library Management System.**
- **Prepare an activity diagram for Bug Tracking System for a software company - to create an ticket(issue) by tester and resolve ticket by an developer in a software company.**
- **Prepare an activity diagram for computing a restaurant bill. There should be a charge for each delivered item. The total amount should be subject to tax and a service charge of 18% for groups of six or more. For smaller groups, there should be a blank entry for a gratuity according to the customer's directions. Any coupons or gift certificate submitted by the customer should be subtracted.**
- **Prepare an activity diagram that elaborates the details of logging into an email system. Note that entry of the user name and password can occur in any order.**

STATE MODELING DIAGRAM

S TATE MODELING : INTRODUCTION

- **You can best understand a system by first examining its static structure – i.e. the structure of its objects and their relationships of each other at a single moment in time(the Class Model).**
- **You can examine the changes to the objects and their relationships over time(the State Model).**
- **The State Model describes the sequence of operations that occur in response to external stimuli.**
- **The State Model consists of multiple state diagrams, one for each class with temporal behavior that is important to an application.**
- **The State Diagram is a graphical representation of finite state machines that relates events and states.**
- **Events represent external stimuli and state represent values of objects.**

S TATE MODELING : EVENT AND S TATE

- **Events**

  An event is something that happens that affects the system. Event refers to the type of occurrence rather than to any concrete instance of that occurrence. For example, Keystroke is an event for the keyboard, but each press of a key is not an event but a concrete instance of the Keystroke event. Another event of interest for the keyboard might be Power-on, but turning the power on tomorrow at 10:05:36 will be just an instance of the Power-on event.

- **States**

  A state captures the relevant aspects of the system's history very efficiently. For example, when you strike a key on a keyboard, the character code generated will be either an uppercase or a lowercase character, depending on whether the Caps Lock is active. Therefore, the keyboard's behavior can be divided into two states: the "default" state and the "caps_locked" state.

S TATE MODELING : EVENT S

Kinds of Events

Signal Event: **It is an explicit one-way transmission (sending or receiving a signal) of information from one object to another.**

Change Event: **It is an event that is caused by the satisfaction of a boolean expression. *When* keyword is used followed by parenthesized boolean expression.**

Ex: when(room temperature < heating set point)

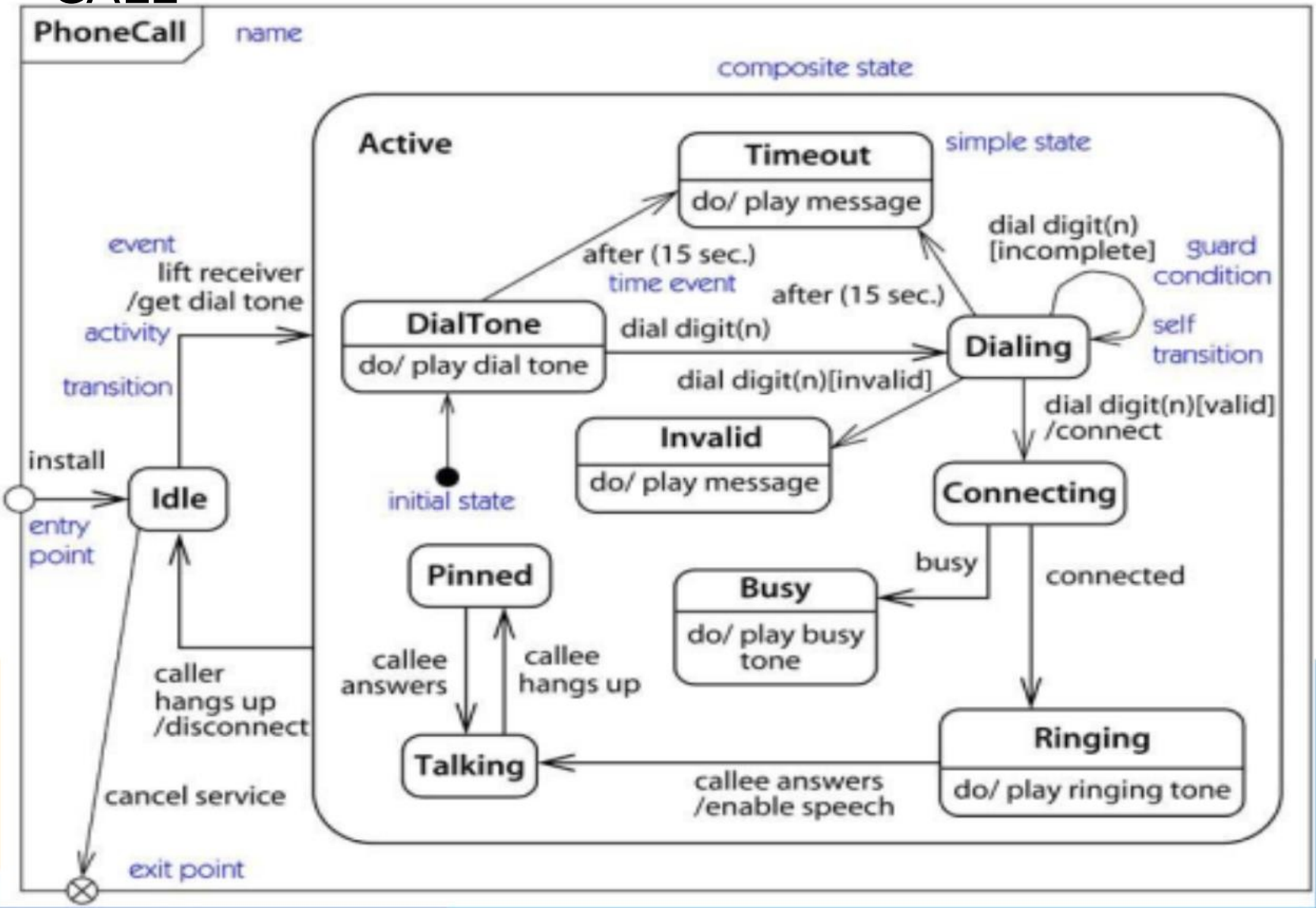
when(battery power < lower limit)

Time Event: **It is an event caused by the occurrence of an absolute time or the elapse of a time interval. *When or after* keyword is used followed by parenthesized expression involving time.**

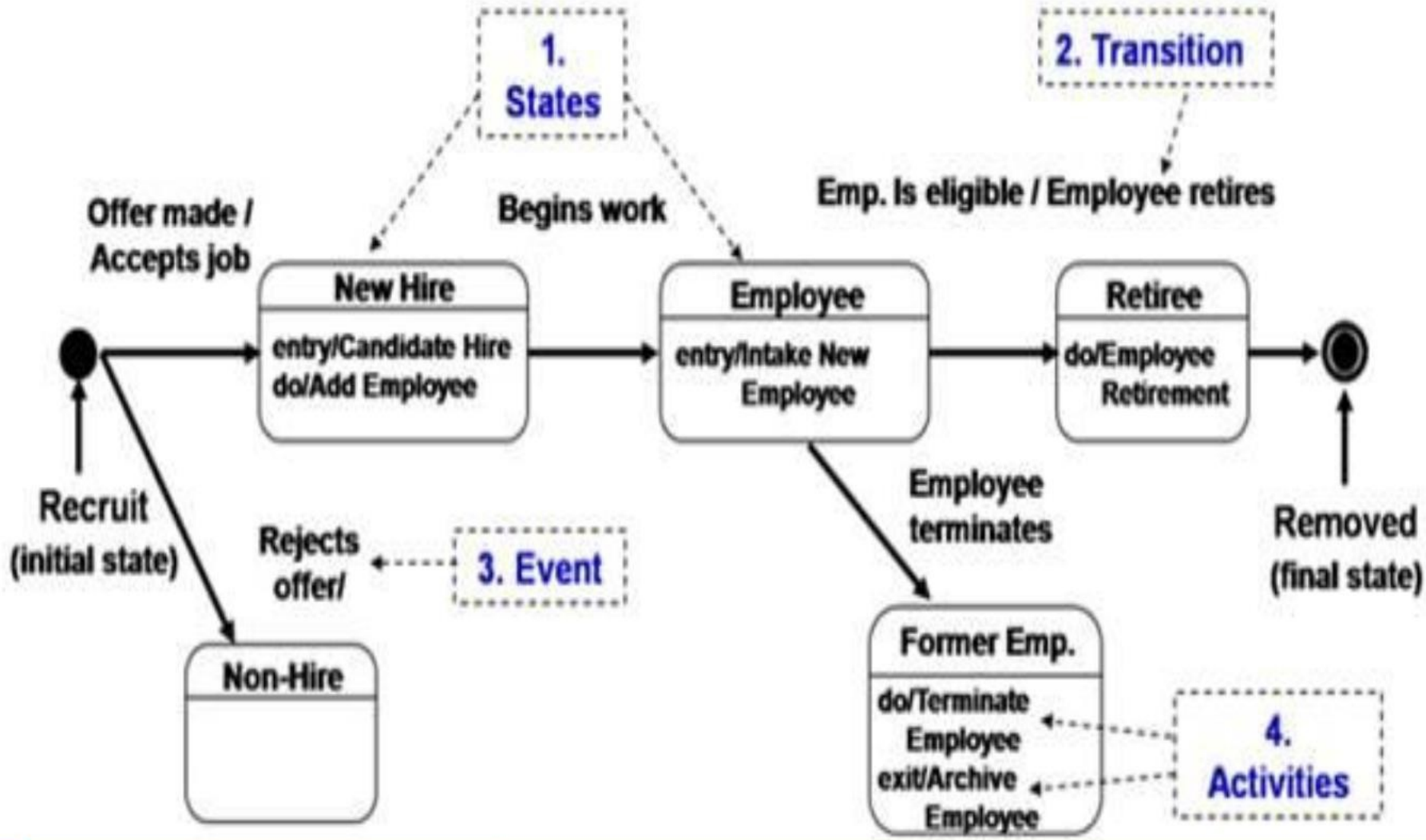
Ex: when(date=January 1, 2010)

after(10 seconds)

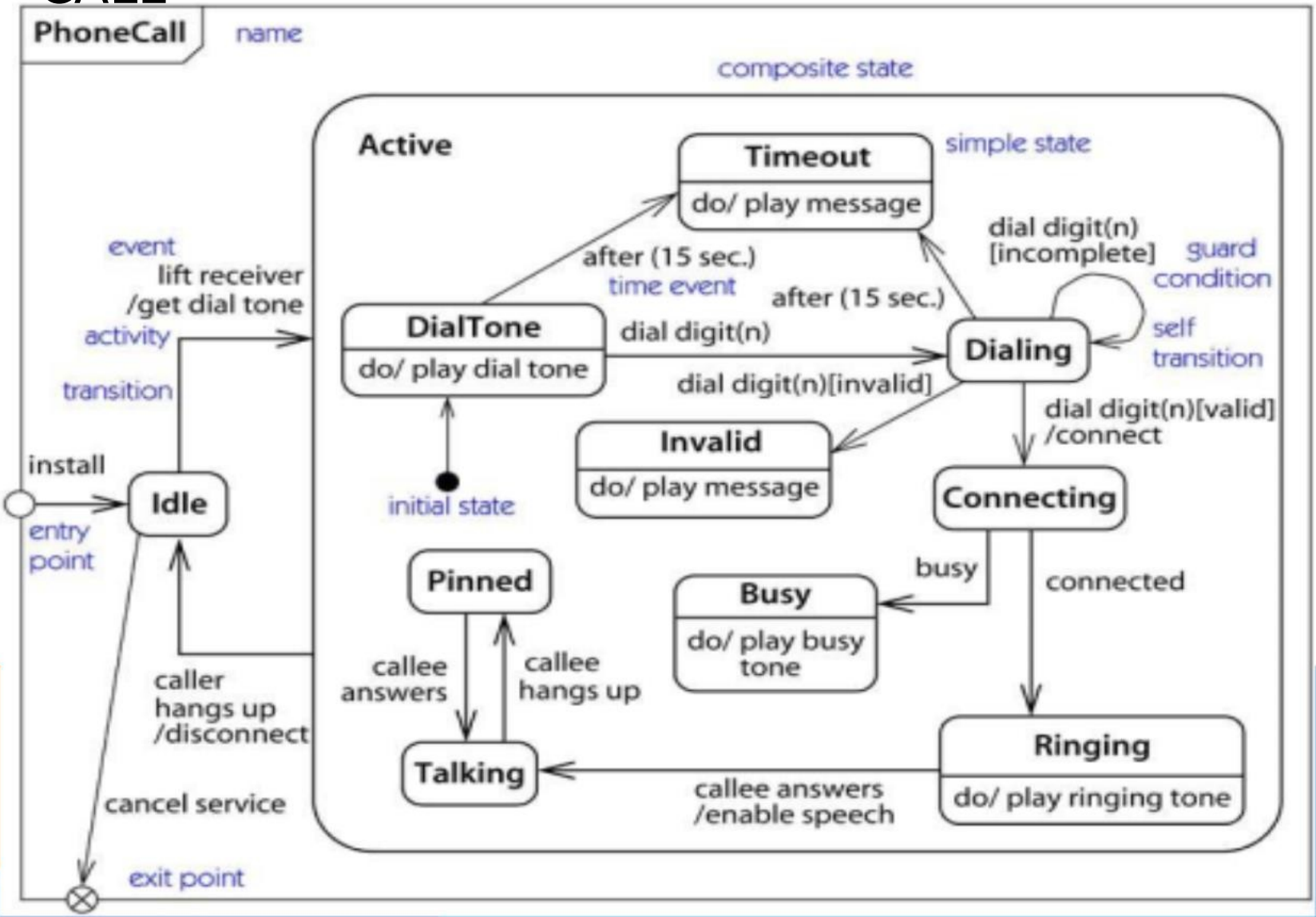
S TATE CHART DIAGRAM: PHONE CALL



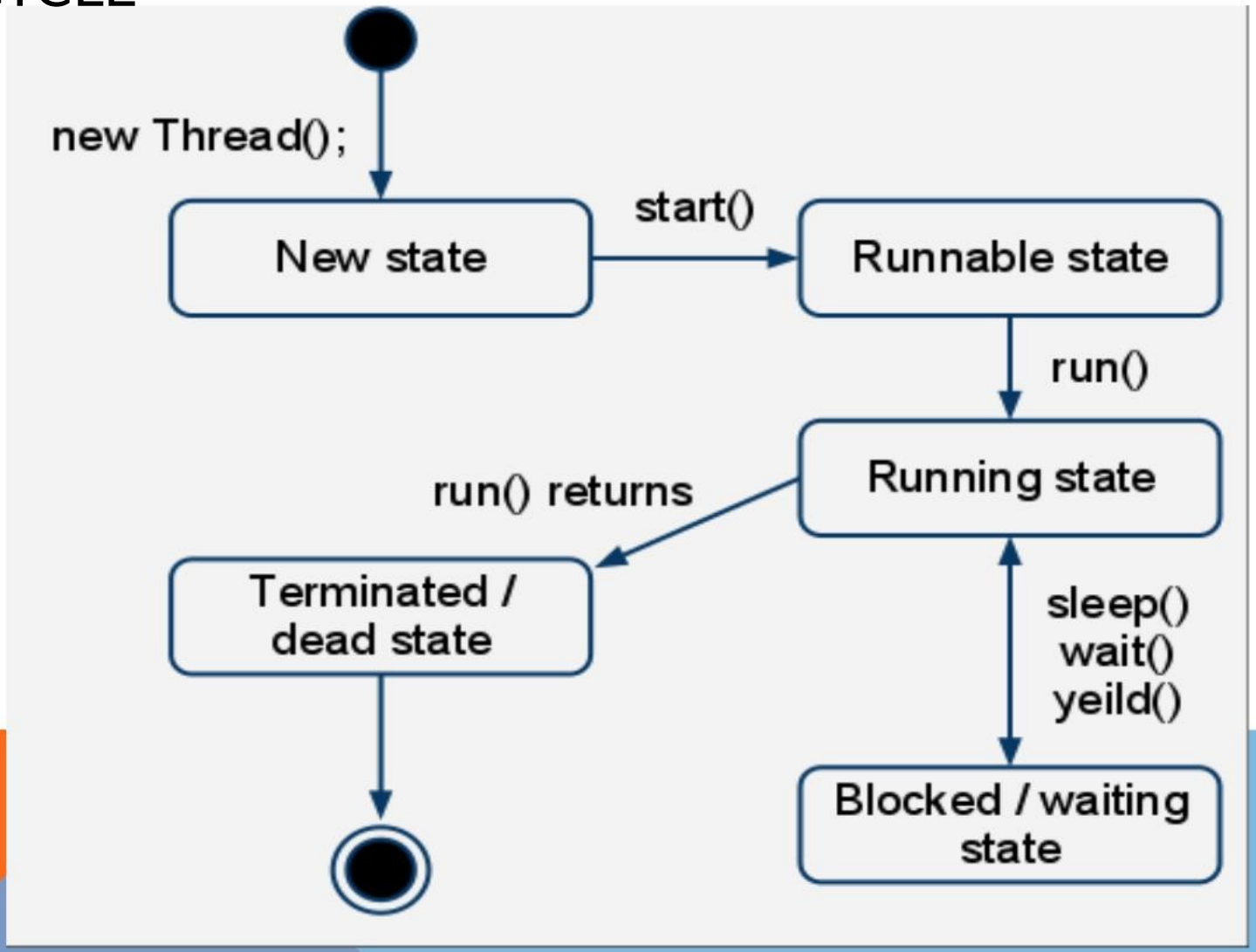
S TATE CHART DIAGRAM: EMPLOYEE HIRING



S TATE CHART DIAGRAM: PHONE CALL



S TATE CHART DIAGRAM: THREAD LIFE CYCLE



S TATE CHART DIAGRAM : EXERCISE/ ASSIGNMENT

- **Prepare a state chart diagram for all the possible states of Washing Machine.**
- **Prepare a State Chart diagram for Library Management System.**