SOOAD

UNIT 3

OBJECT ORIENTED ANALYSIS & DESIGN

UNIT -3 OOAD

- Introduction
- Object-Oriented Modelling
- Object-Oriented Approach
- The Constituents of OOAD
- Pillars of Object-Oriented Analysis and Design
- The Language of OOAD – Unified Modelling Language

PILLARS OF OOAD

PLLARS OF OOAD

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism
- Coupling
- Cohesion
- Components
- Interface

Abstraction

- Abstraction is specifying the framework and hiding the implementation level information.
- Concreteness will be built on top of the abstraction.
- It gives you a blueprint to follow to while implementing the details.
- Abstraction reduces the complexity by hiding low level details.
- Act of representing essential features without including the background details or explanations.

Encapsulation

- Acts like a black box
- Encapsulation is the process of binding both attributes and methods together within a class.
- Through encapsulation, the internal details of a class can be hidden from outside.
- It permits the elements of the class to be accessed from outside only through the interface provided by the class.
- Allows degree of visibility of elements

Inheritance

- Inheritance is the mechanism that permits new classes to be created out of existing classes by extending and refining its capabilities.
- The existing classes are called the base classes/parent classes/super-classes, and the new classes are called the derived classes/child classes/subclasses.
- The subclass can inherit or derive the attributes and methods of the super-class(es) provided that the super-class allows so.
- Besides, the subclass may add its own attributes and methods and may modify any of the super-class methods. Inheritance defines an “is – a” relationship.

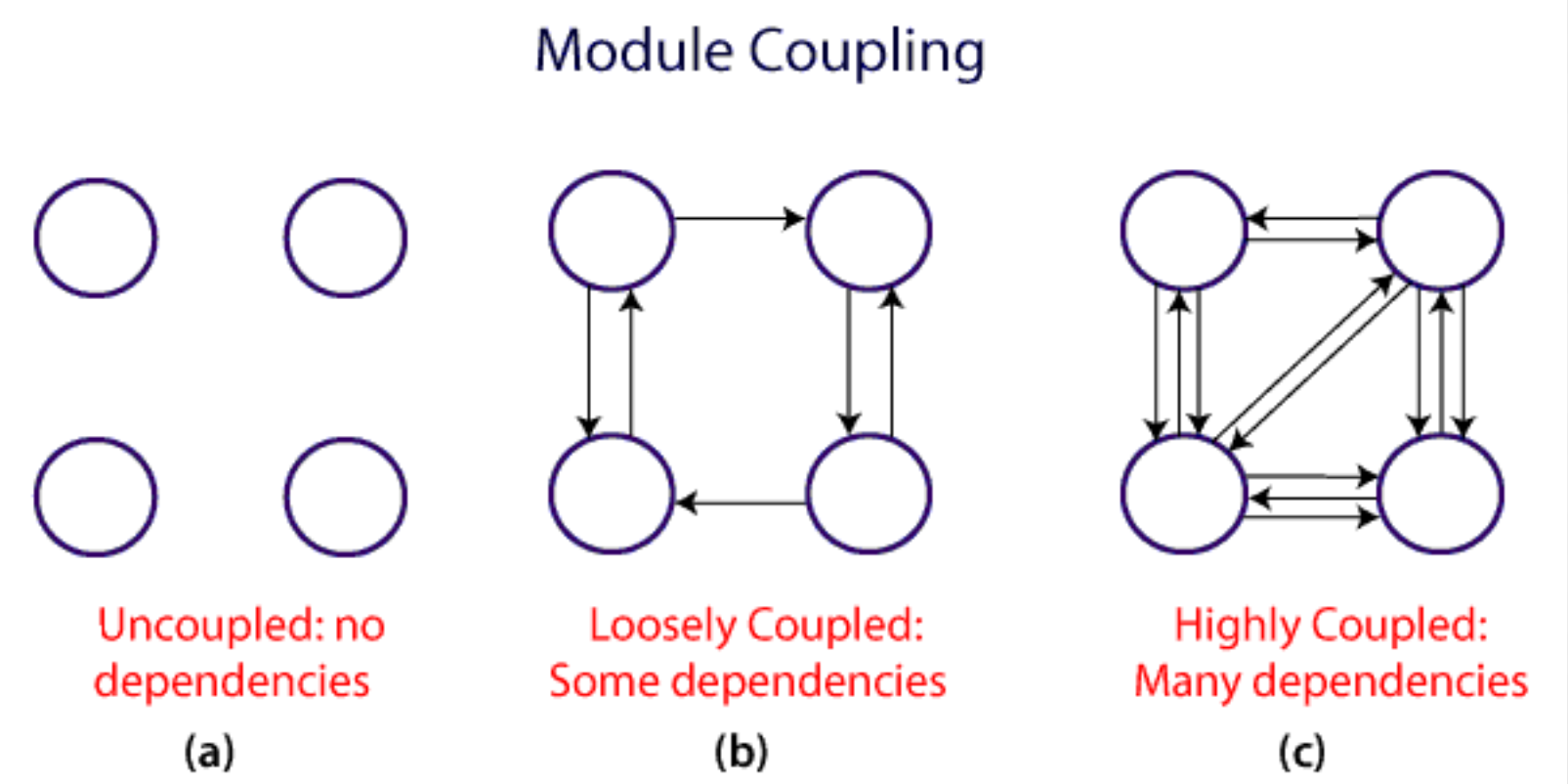
Polymorphism

- Polymorphism is originally a Greek word that means the ability to take multiple forms.
- In object-oriented paradigm, polymorphism implies using operations in different ways, depending upon the instance they are operating upon.
- Polymorphism allows objects with different internal structures to have a common external interface.
- Polymorphism is particularly effective while implementing inheritance.

Coupling

- The coupling is the degree of interdependence between software modules.
- Refers to the ways, in which and degrees, to which one part of the system relies on the details of another part.
- The tighter the coupling, the more changes in one part of the system will ripple throughout the system.
- With loose coupling, the interfaces between subsystems are well defined and restricted.
- What lies beyond those interfaces can change without any changes needed in the client subsystems.
- OOP supports loose coupling by allowing us to define and publish a class's method without publishing how those methods are carried out.

Coupling

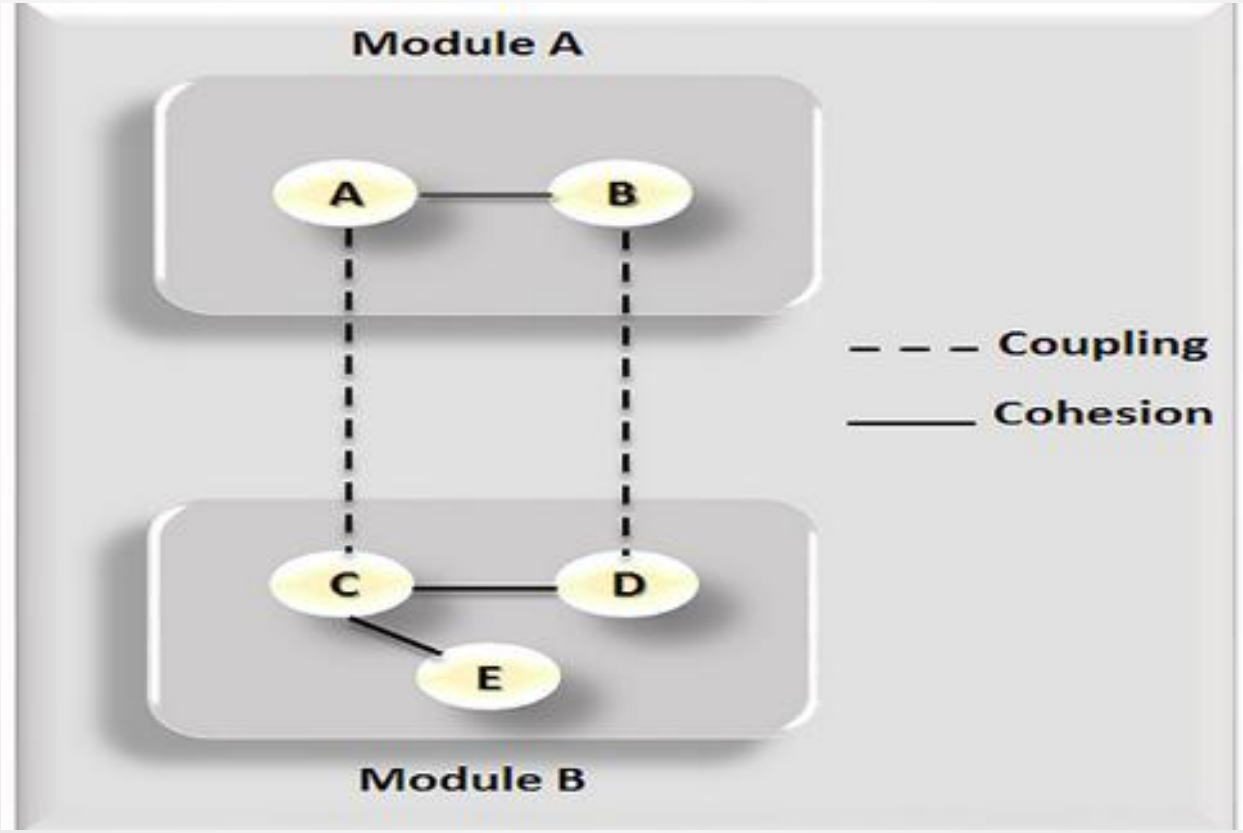

Module Coupling
Uncoupled: no dependencies
(a)
Loosely Coupled: Some dependencies
(b)
Highly Coupled: Many dependencies
(c)

Cohesion

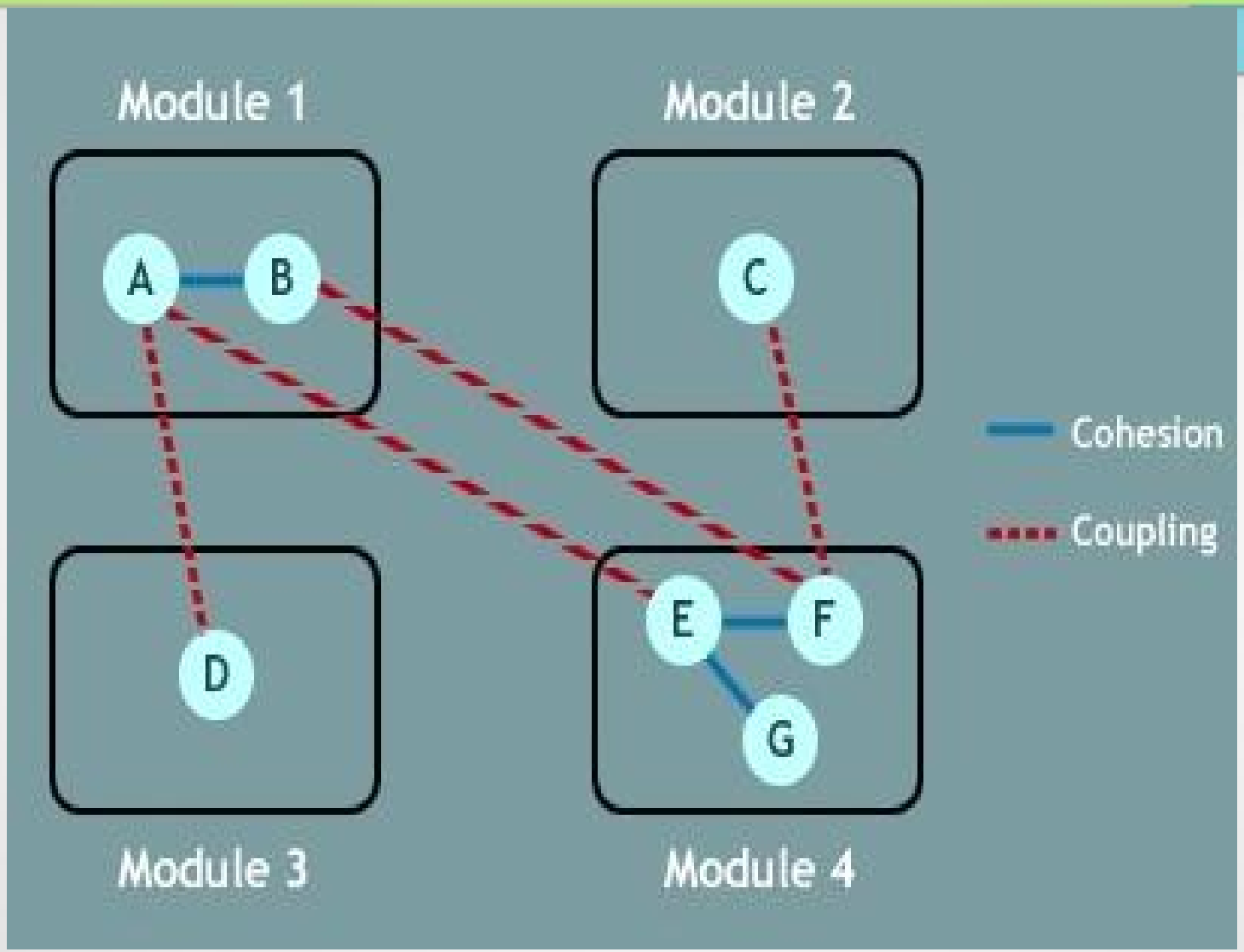
- Refers to the degree in which elements within a subsystem form a single, unified concept, with no excess elements.
- Cohesion is used to indicate the degree to which a class has a single, well-focused purpose.
- Coupling is all about how classes interact with each other, on the other hand cohesion focuses on how single class is designed.
- Higher the cohesiveness of the class, better is the OO design.
- Where there is high cohesion, there is easier comprehension and thus more reliable code.
- OOP supports strong cohesion by allowing one to design classes in which the data and the fuctions that operate on them are tightly bound together.

Coupling & Cohesion

- Cohesion and Coupling deal with the quality of an OOAD design.



Coupling & Cohesion

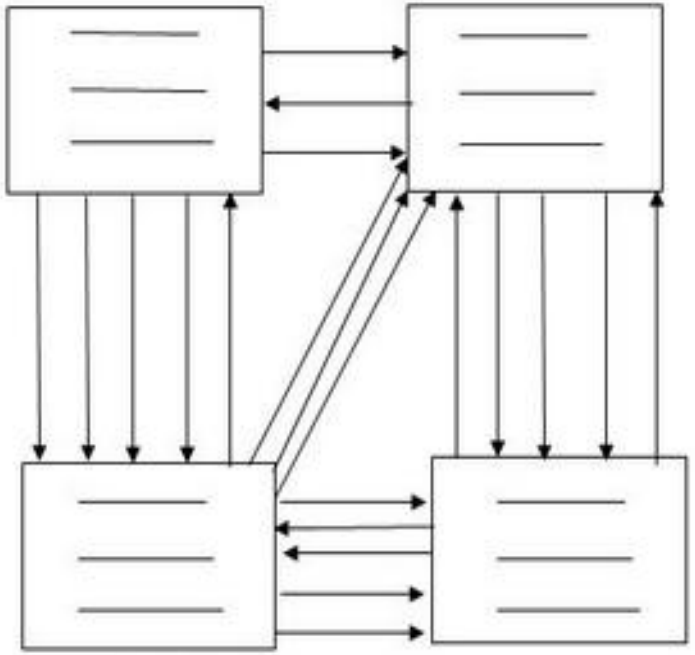


Coupling & Cohesion

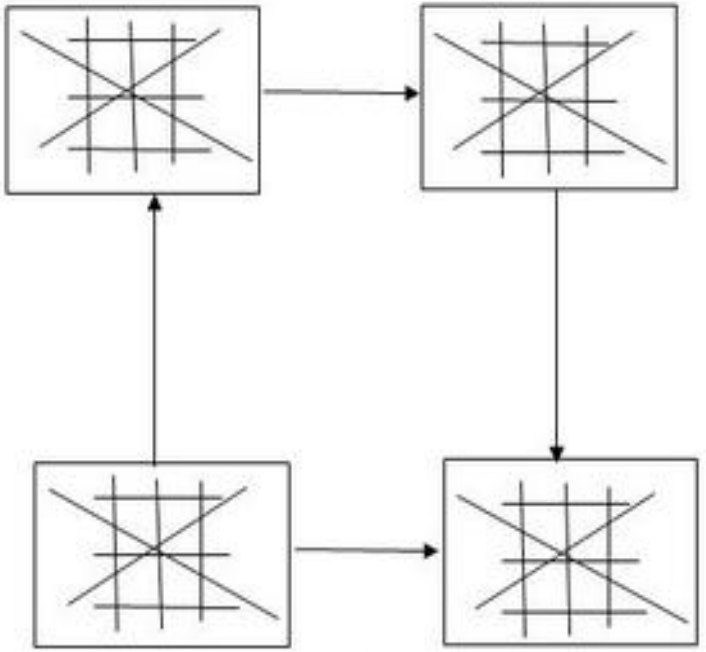
- Cohesion is the indication of the relationship within module.
- Coupling is the indication of the relationships between modules.
- Cohesion shows the module's relative functional strength.
- Coupling shows the relative independence among the modules.
- Cohesion is a degree (quality) to which a component / module focuses on the single thing.
- Coupling is a degree to which a component / module is connected to the other modules.

Coupling & Cohesion

- While designing you should strive for high cohesion i.e. a cohesive component/ module focus on a single task (i.e., single-mindedness) with little interaction with other modules of the system.
- While designing you should strive for low coupling i.e. dependency between modules should be less.
- Cohesion is the kind of natural extension of data hiding for example, class having all members visible with a package having default visibility.
- Making private fields, private methods and non public classes provides loose coupling.
- Cohesion is Intra – Module Concept
- Coupling is Inter – Module Concept



High Coupling
Low Cohesion

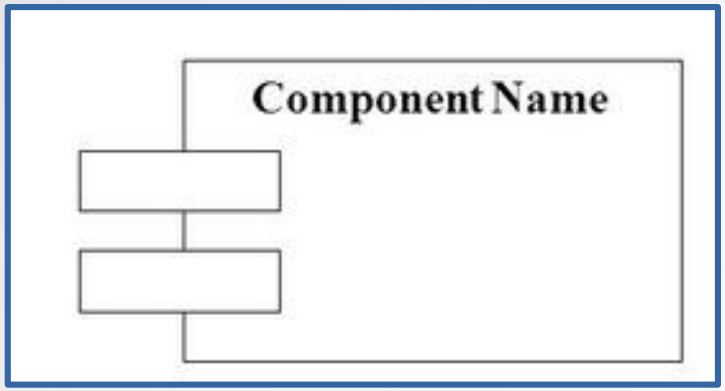


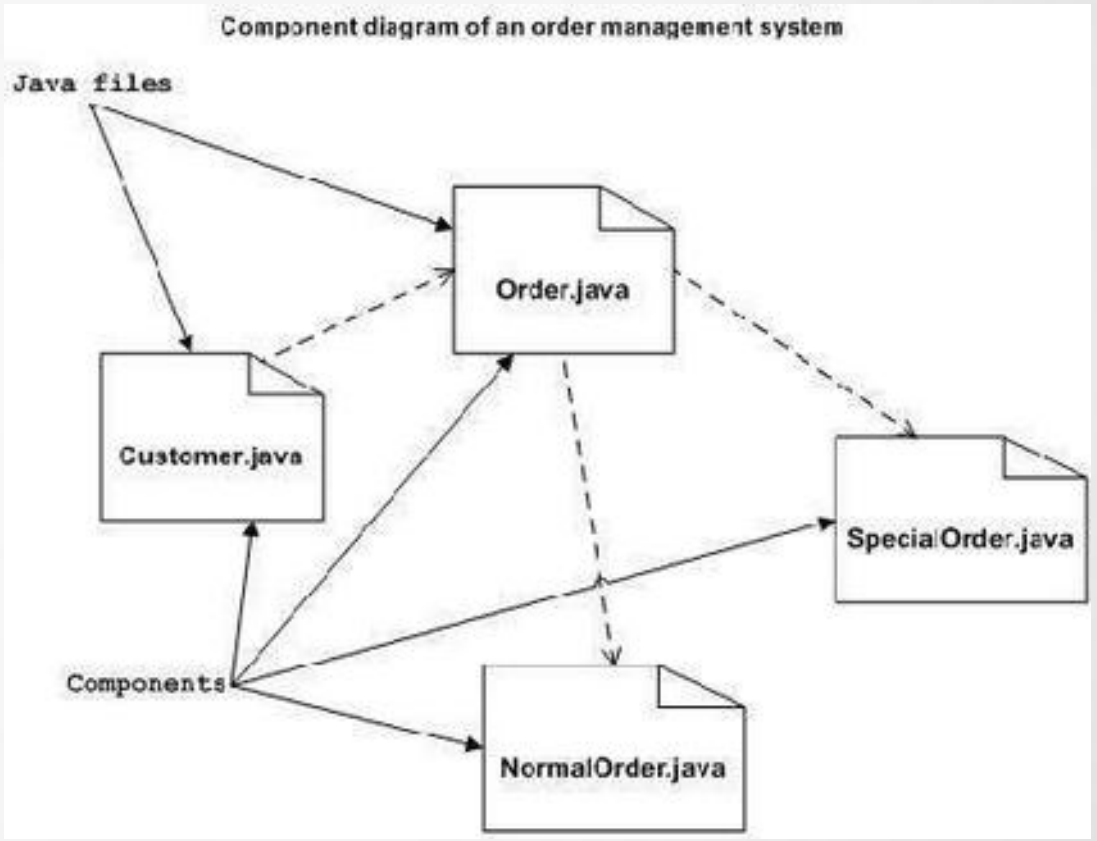
Low Coupling
High Cohesion

Components

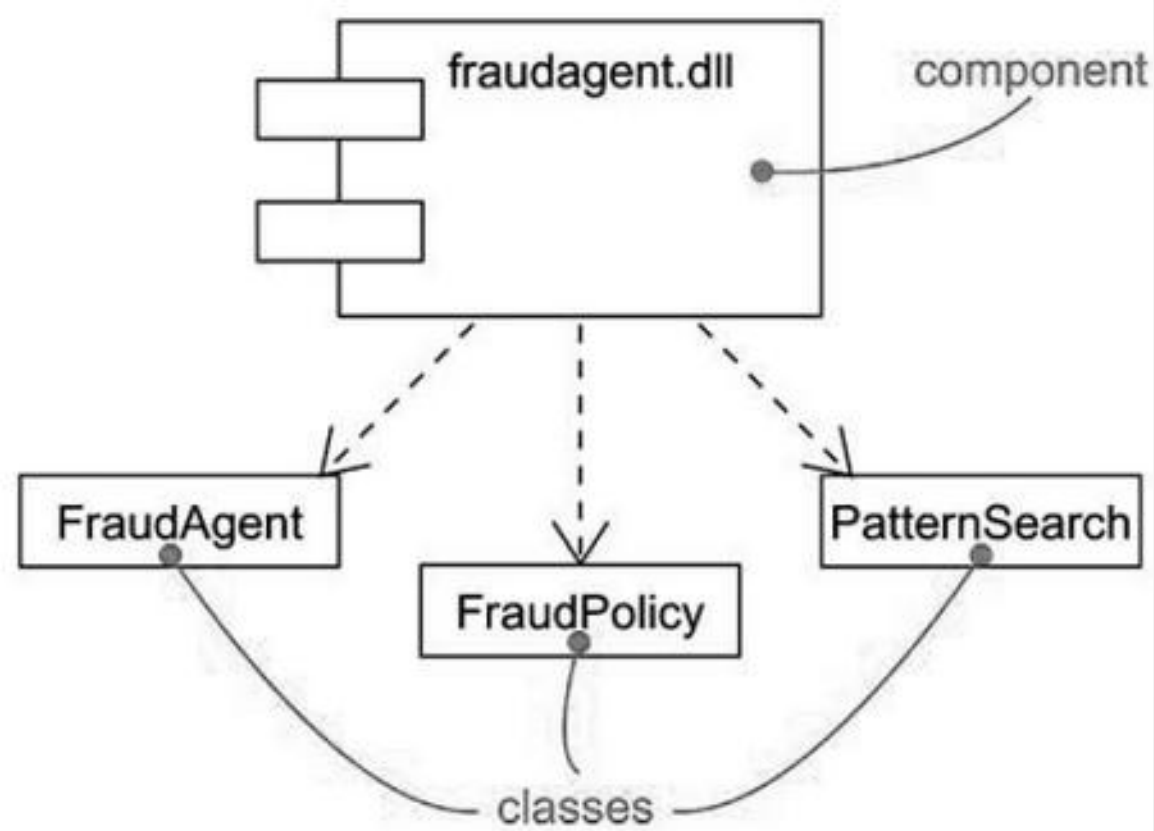
- In programming and engineering disciplines, a component is an identifiable part of a larger program or construction.
- Usually, a component provides a particular function or group of related functions.
- In programming design, a system is divided into components that in turn are made up of modules.
- Component test means testing all related modules that form a component as a group to make sure they work together.
- Component is a collection of related classes that together provide a larger set of services.
- Component in a system might include applications, libraries, Activex controls, javaBean daemons and services.

Components


Component Name


Component diagram of an order management system
Java files
Order.java
Customer.java
SpecialOrder.java
Components
NormalOrder.java

Components


fraudagent.dll
component
FraudAgent
FraudPolicy
PatternSearch
classes

Interfaces

- An interface is a definition of a set of services provided by a component or by a class.
- This alllows further encapsulation : the author of a component can pubish just the interfaces to the component, completely hiding any implementation details.

  Interface is a collection of methods of a class or component.
- It specifies the set of services that may be provided by the class or component.

THE LANGUAGE OF OOAD - UNIFIED MODELLING LANGUAGE AND BPMN

THE LANGUAGE OF OOAD

- The Unified Modeling Language is a graphical language for visualizing, specifying, constructing & documenting the artifacts of a software intensive system.
- Offers a standard way to write a system's blueprints, including conceptual things like business processes and system functions as well as concrete thing such as programming language statements, database schemas, and reusable software components.
- Provides a comprehensive notation for communicating the requirements, behavoiur, architecture, and realization of an oo design.
- Provides a way to create and document a model of a system.

`

THE LANGUAGE OF OOAD

- Purpose is communcication; to be specific, it is to provide a comprehensive notation for communcating the requirements, architecture, implementation, deployment and states of a system.
- Everything is decribed in terms of obejcts: the actions that objects take, the relationships between the objects, the deployment of the objects, and the way the states of objects change in response to extrenal events.
- Static diagram: shows the structure of the system
- Dynamic diagram: shows the behavior of the system

THE LANGUAGE OF OOAD

- With dynamic diagrams, the user can trace through the behaviour and analyze how various scenarios play out.
- With static diagrams, we ensure that each component or class has access to the interfaces and information that it needs to carry out its reponsibilities.

UML DIAGRAMS

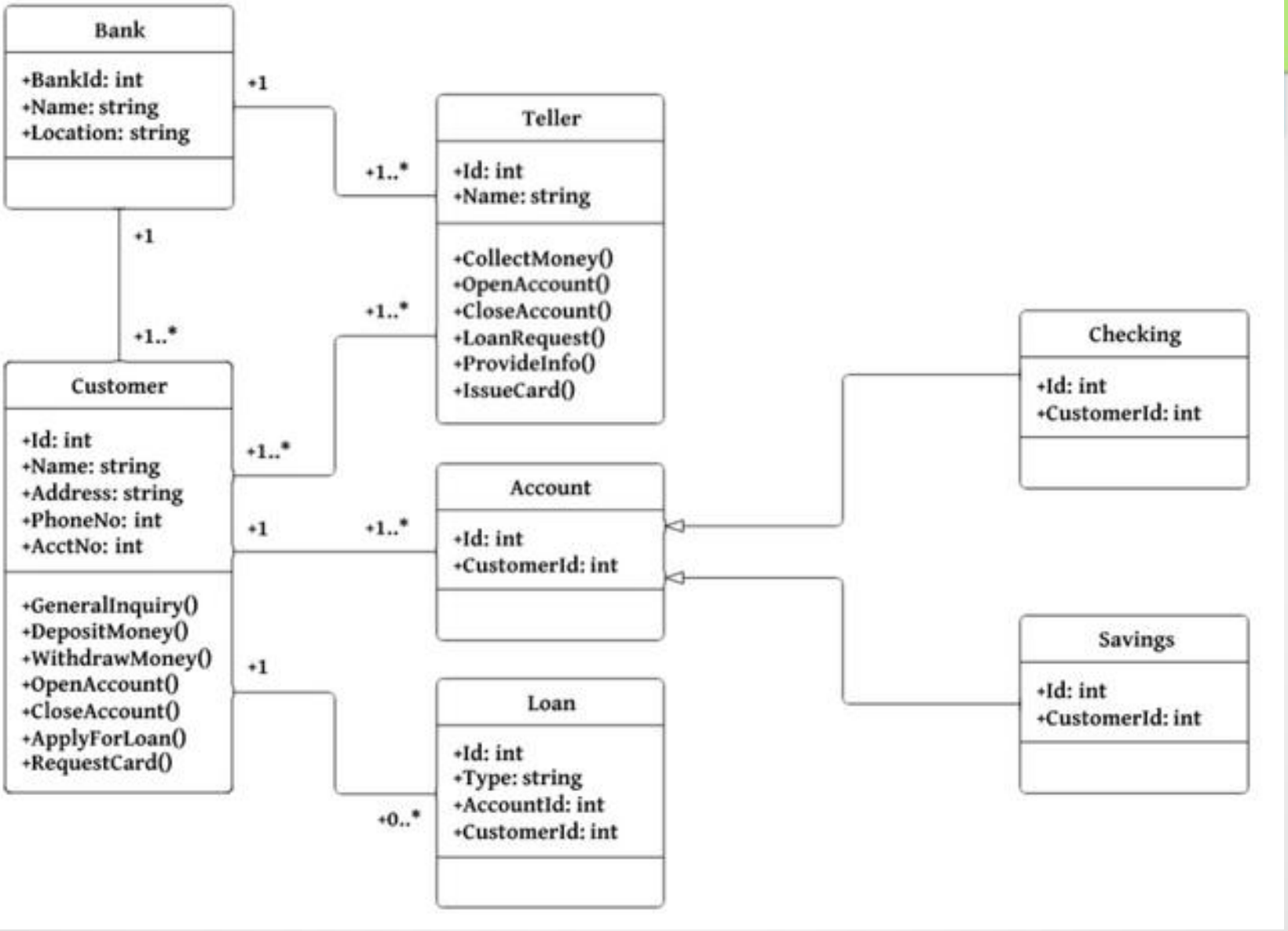
UML Diagrams

1) Class diagrams
2) Object diagrams
3) Use-case diagrams
4) Sequence diagrams
5) Collaboration diagrams
6) Statechart diagrams
7) Activity diagrams
8) Component diagrams
9) Deployment diagrams

UML Diagrams

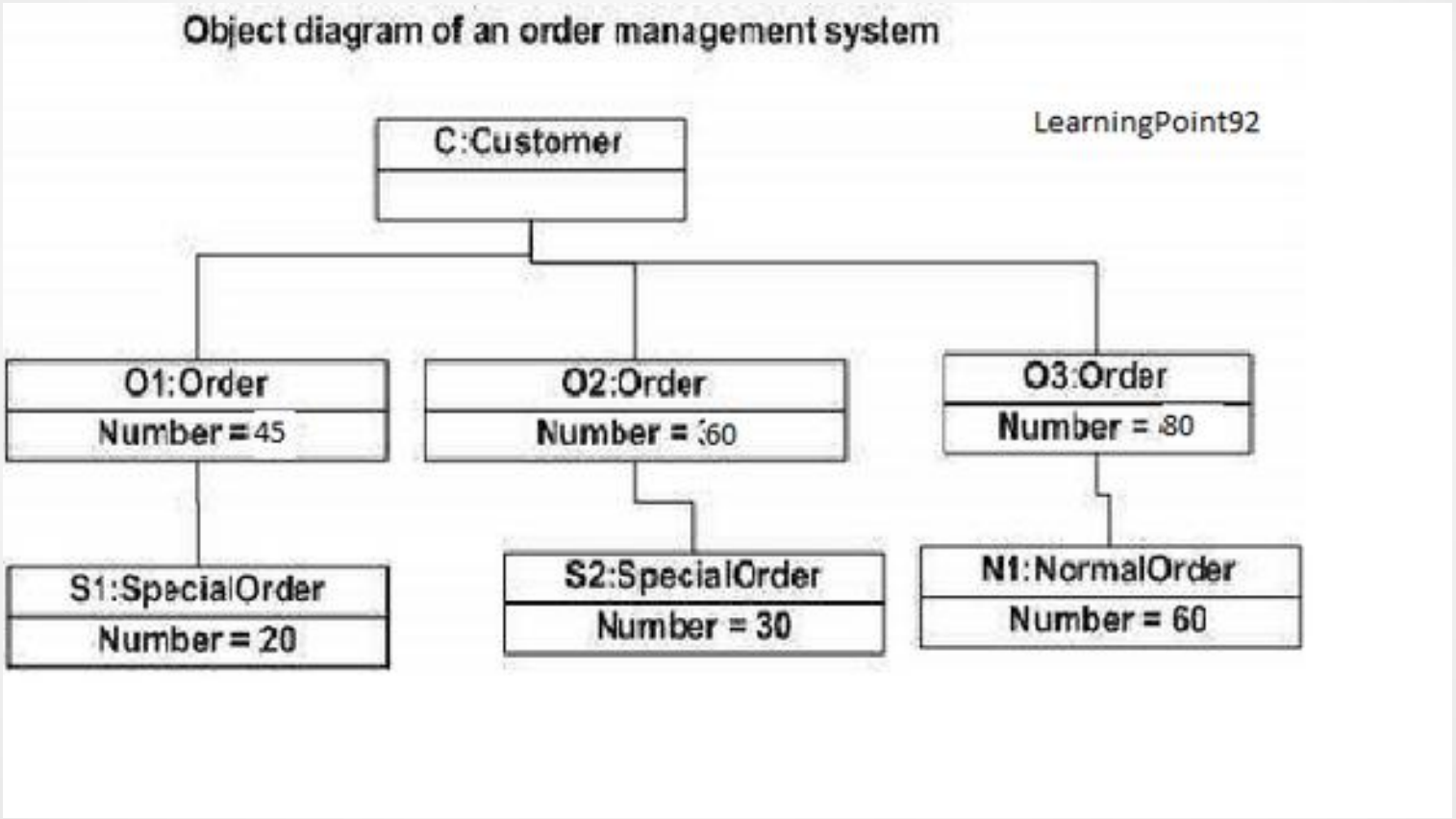
Class Diagram:

- A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.
- Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction.


Bank
+BankId: int
+Name: string
+Location: string
+1
+1..*
Teller
+Id: int
+Name: string
+CollectMoney()
+OpenAccount()
+CloseAccount()
+LoanRequest()
+ProvideInfo()
+IssueCard()
+1
+1..*
+1..*
Customer
+Id: int
+Name: string
+Address: string
+PhoneNo: int
+AcctNo: int
+GeneralInquiry()
+DepositMoney()
+WithdrawMoney()
+OpenAccount()
+CloseAccount()
+ApplyForLoan()
+RequestCard()
+1
+1..*
+1
Account
+Id: int
+CustomerId: int
+1
+0..*
Loan
+Id: int
+Type: string
+AccountId: int
+CustomerId: int
Checking
+Id: int
+CustomerId: int
Savings
+Id: int
+CustomerId: int

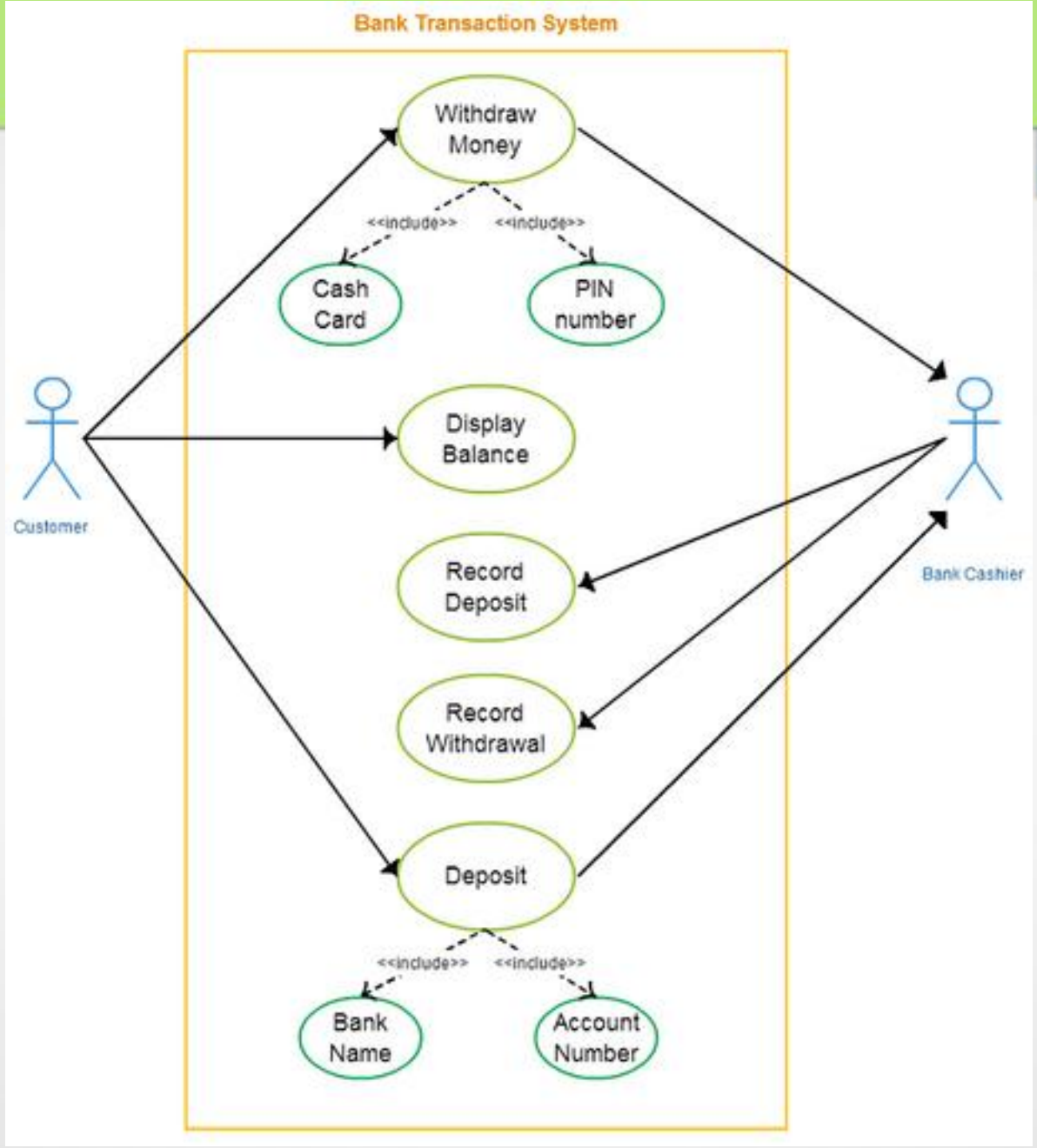
UML Diagrams

- Object Diagram:
- An object diagram in the Unified Modeling Language (UML), is a diagram that shows a complete or partial view of the structure of a modeled system at a specific time.
- Object diagrams represent an instance of a class diagram. The basic concepts are similar for class diagrams and object diagrams.
- Object diagrams also represent the static view of a system but this static view is a snapshot of the system at a particular moment.


Object diagram of an order management system
LearningPoint92
C:Customer
O1:Order
Number = 45
O2:Order
Number = 60
O3:Order
Number = 80
S1:SpecialOrder
Number = 20
S2:SpecialOrder
Number = 30
N1:NormalOrder
Number = 60

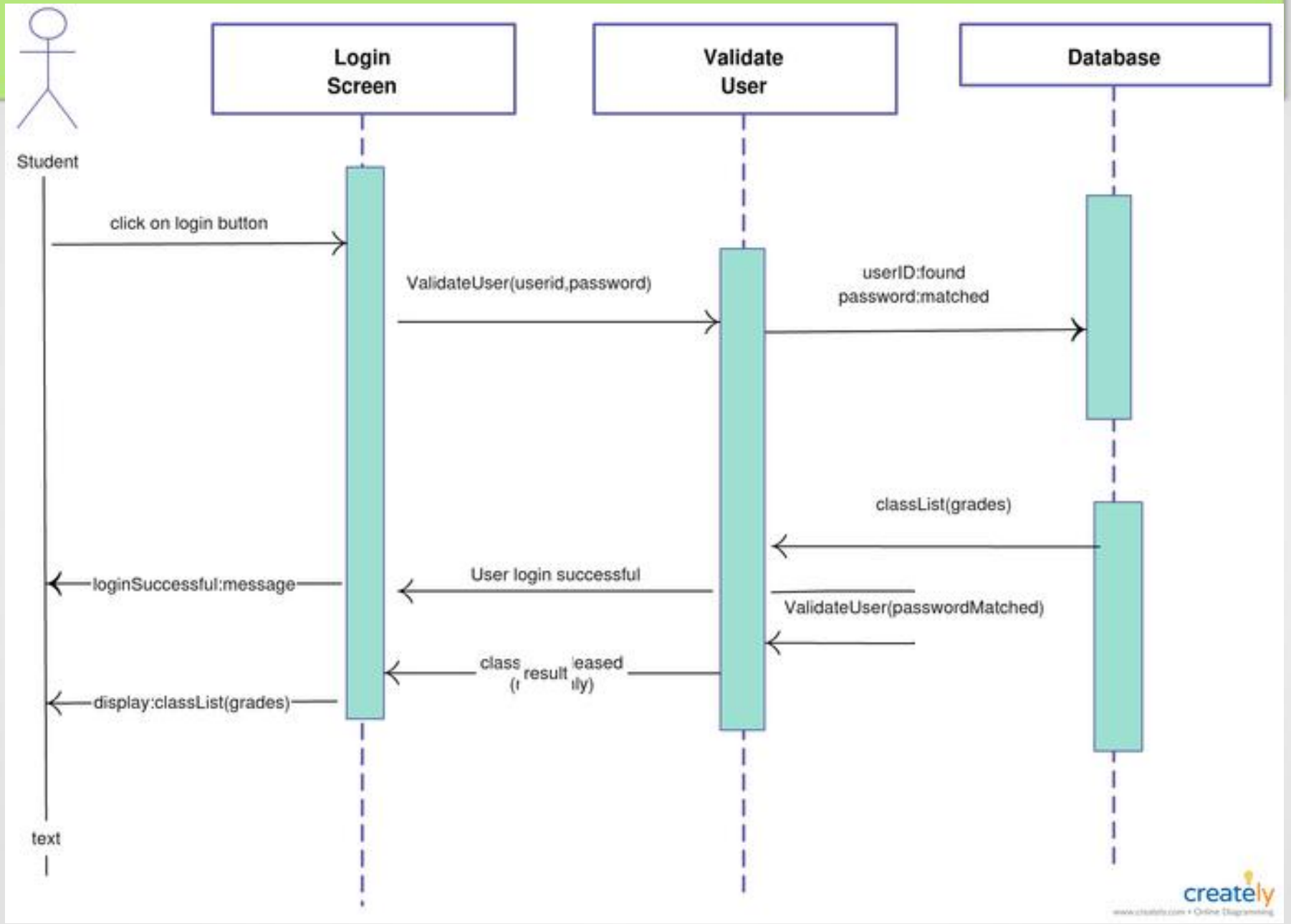
UML Diagrams

- Use-case Diagram:
- A use case diagram is a graphic depiction of the interactions among the elements of a system.
- A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved.
- Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors).
- Each use case should provide some observable and valuable result to the actors or other stakeholders of the system.


Bank Transaction System
Withdraw Money
<<include>>
<<include>>
Cash Card
PIN number
Display Balance
Record Deposit
Record Withdrawal
Deposit
<<include>>
<<include>>
Bank Name
Account Number
Customer
Bank Cashier

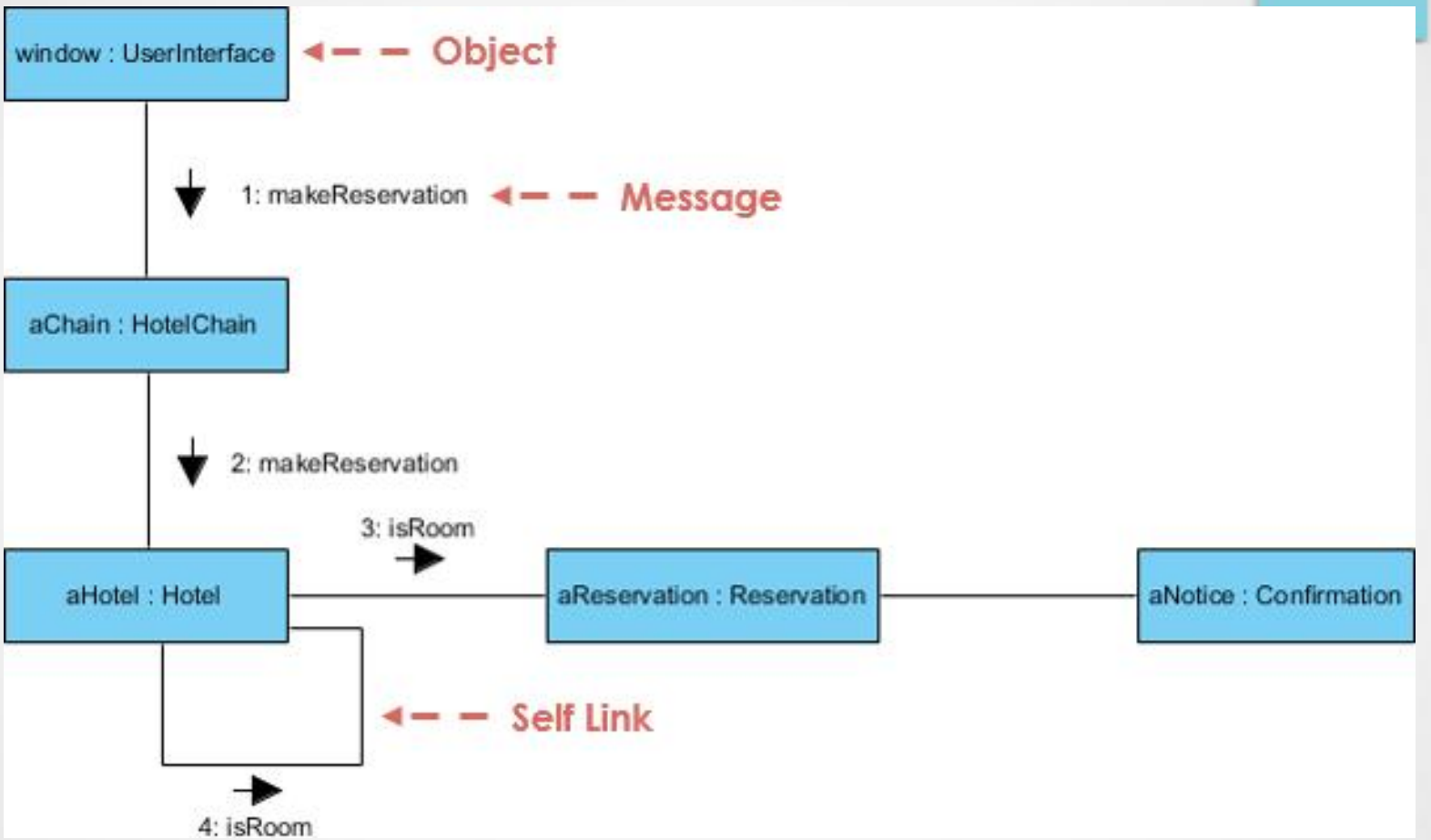
UML Diagrams

- Sequence Diagram:
- A sequence diagram is an interaction diagram that shows how objects operate with one another and in what order.
- It is a construct of a message sequence chart.
- A sequence diagram is a kind of interaction diagram, because they describe how—and in what order—a group of objects works together.
- Sequence diagrams are a popular dynamic modeling solution in UML because they specifically focus on the "lifelines" of an object and how they communicate with other objects to perform a function before the lifeline ends.


Login Screen
Validate User
Database
Student
click on login button
ValidateUser(userid,password)
userID:found
password:matched
classList(grades)
loginSuccessful:message
User login successful
ValidateUser(passwordMatched)
display:classList(grades)
text
creately

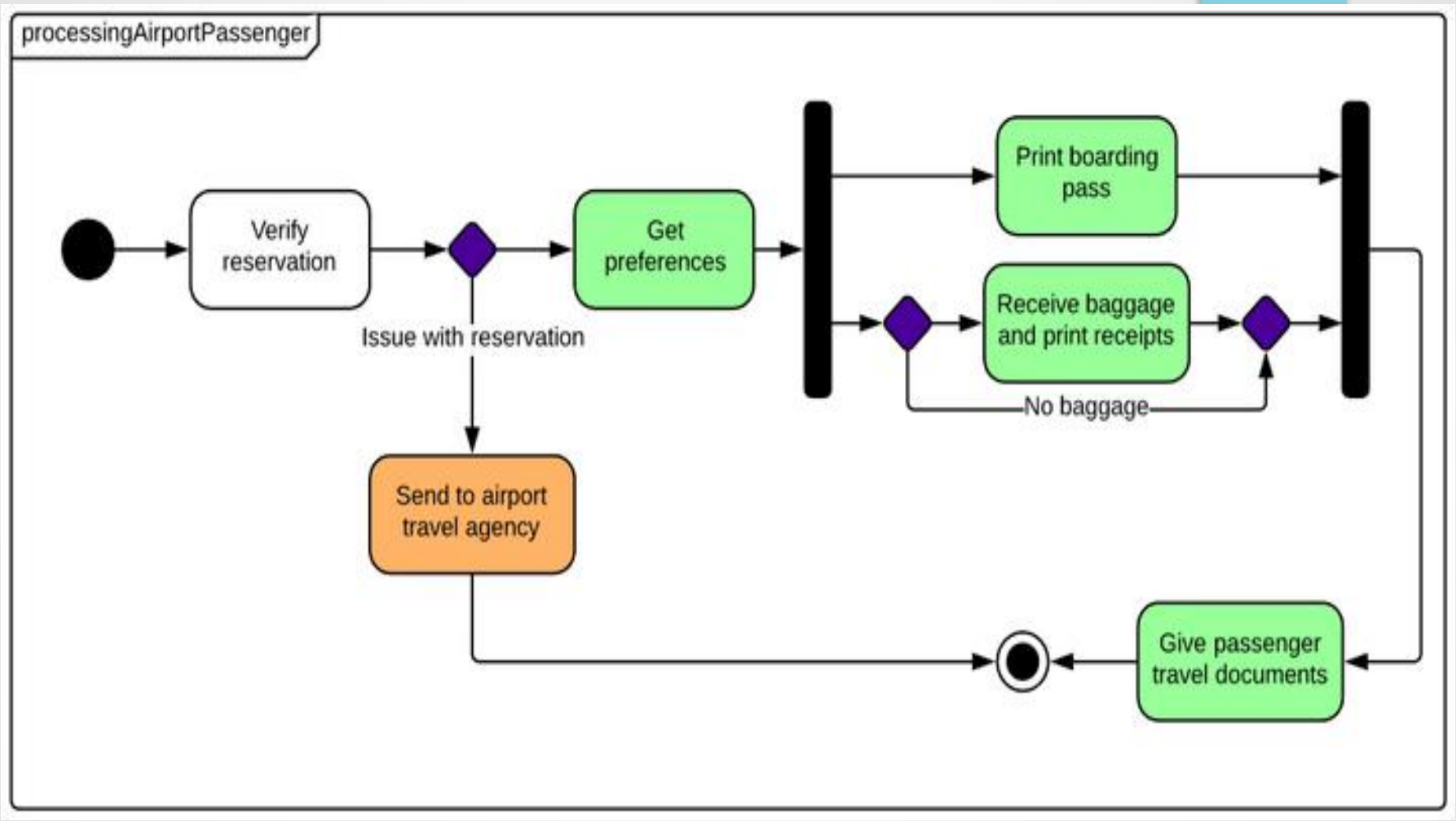
UML Diagrams

- Collaboration Diagram:
- The UML Collaboration diagram is used to model how objects involved in a scenario interact, with each object instantiating a particular class in the system.
- Objects are connected by links, each link representing an instance of an association between the respective classes involved.
- The link shows messages sent between the objects, and the type of message passed (synchronous, asynchronous, simple, balking, and timeout).
- A collaboration diagram resembles a flowchart that portrays the roles, functionality and behavior of individual objects as well as the overall operation of the system in real time.


window : UserInterface
Object
1: makeReservation
Message
aChain : HotelChain
2: makeReservation
3: isRoom
aHotel : Hotel
aReservation : Reservation
aNotice : Confirmation
Self Link
4: isRoom

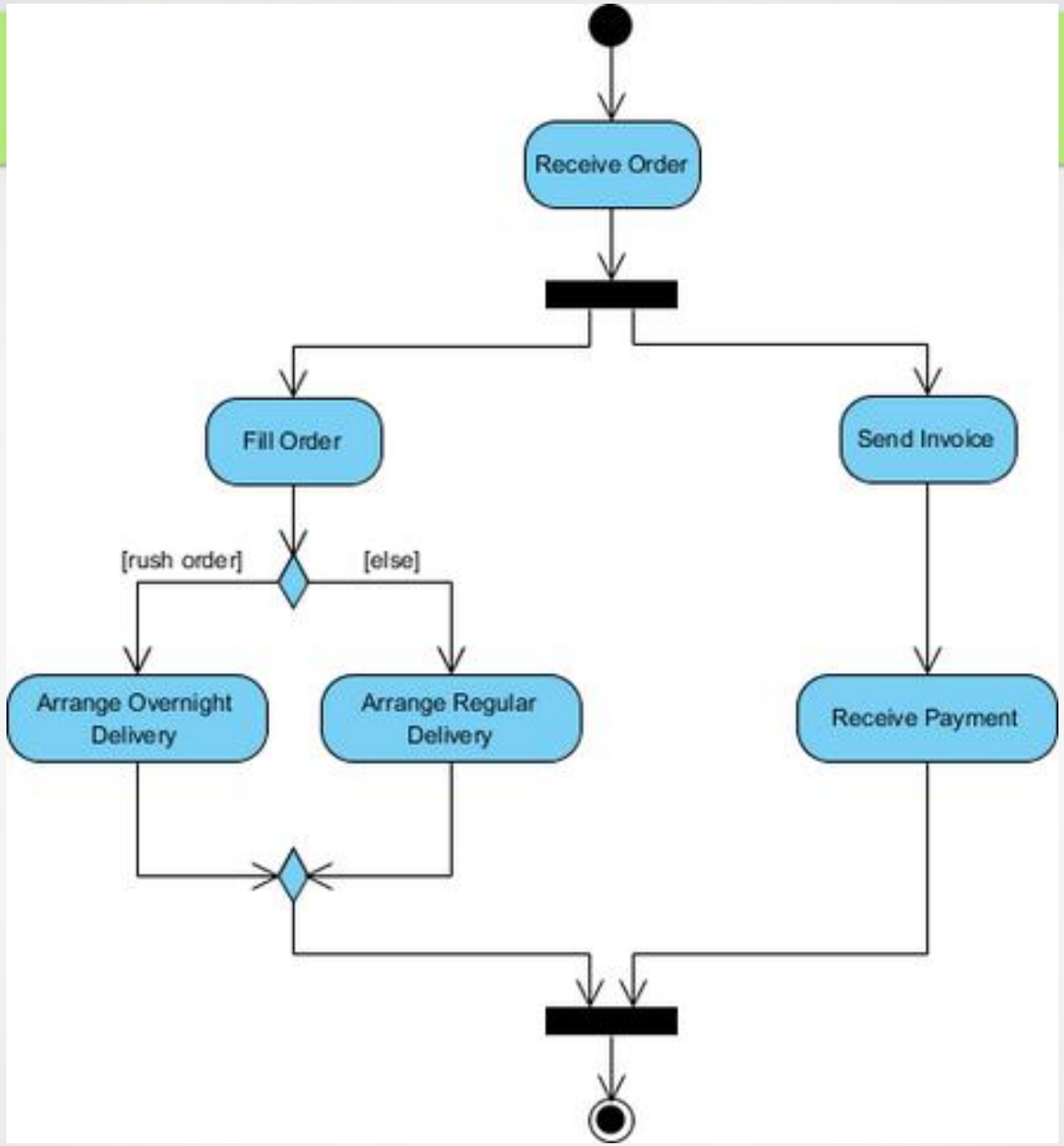
UML Diagrams

- Statechart Diagram:
- The name of the diagram itself clarifies the purpose of the diagram and other details.
- It describes different states of a component in a system. The states are specific to a component/object of a system.
- A Statechart diagram describes a state machine.
- State machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events.


processingAirportPassenger
Verify reservation
Issue with reservation
Get preferences
Print boarding pass
Receive baggage and print receipts
No baggage
Send to airport travel agency
Give passenger travel documents

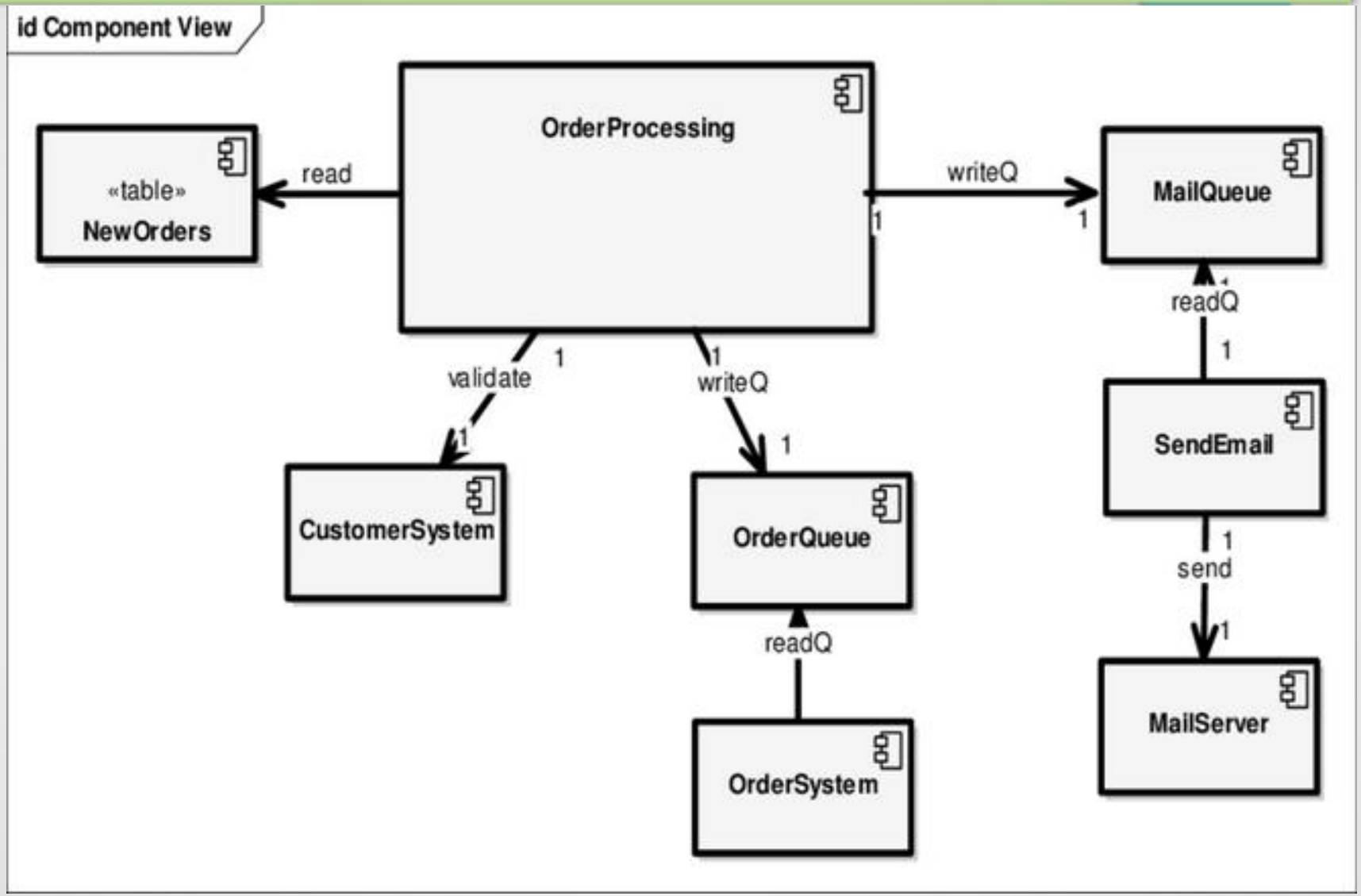
UML Diagrams

- Activity Diagram:
- Activity diagram is another important diagram in UML to describe the dynamic aspects of the system.
- Activity diagram is basically a flowchart to represent the flow from one activity to another activity.
- The activity can be described as an operation of the system.
- The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent.
- Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc


Receive Order
Fill Order
Send Invoice
[rush order]
[else]
Arrange Overnight Delivery
Arrange Regular Delivery
Receive Payment

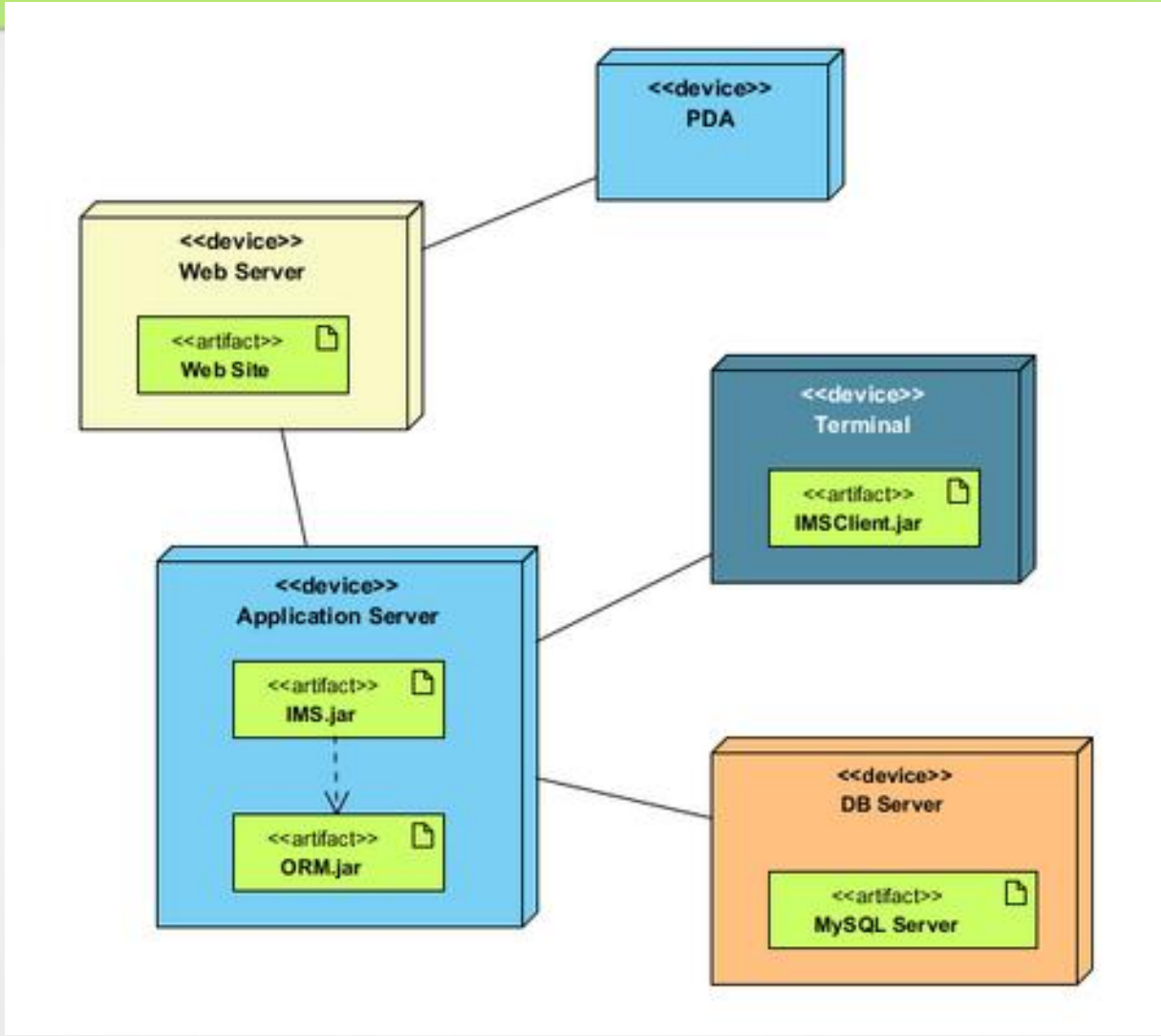
UML Diagrams

- Component Diagram:
- Component diagrams are different in terms of nature and behavior. Component diagrams are used to model the physical aspects of a system. Now the question is, what are these physical aspects? Physical aspects are the elements such as executables, libraries, files, documents, etc. which reside in a node.
- Component diagrams are used to visualize the organization and relationships among components in a system. These diagrams are also used to make executable systems.


id Component View
OrderProcessing
«table»
NewOrders
read
writeQ
MailQueue
1
1
readQ
1
validate
1
1
writeQ
1
1
SendEmail
CustomerSystem
OrderQueue
1
send
1
readQ
MailServer
OrderSystem

UML Diagrams

- Deployment Diagram: Deployment diagram is a structure diagram which shows architecture of the system as deployment (distribution) of software artifacts to deployment targets.
- Artifacts represent concrete elements in the physical world that are the result of a development process. Examples of artifacts are executable files, libraries, archives, database schemas, configuration files, etc.


<<device>>
PDA
<<device>>
Web Server
<<artifact>>
Web Site
<<device>>
Terminal
<<artifact>>
IMSClient.jar
<<device>>
Application Server
<<artifact>>
IMS.jar
<<artifact>>
ORM.jar
<<device>>
DB Server
<<artifact>>
MySQL Server

INTRODUCTION TO BPMN

BPMN

- The Business process Management Initiative(BPMI) has developed a standard Business Process Modeling Notation (BPMN).
- BPMN 1.0 specification was relased to the public in May 2004.
- To provide a notation that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those process.

BPMN

- The BPMN creates a standardized bridge for the gap between the business process design and process implementation
- It is an agreement between multiple modelling tools vendors, who had their own notations, to use a single notation for the benefit of end-user's understanding and training..
- BPMN defines a Business Process Diagram, which is based on a flowcharting technique tailored for creating graphical objects, which are activities and the flow controls that define their order of performance.