

**CSE213 - OBJECT ORIENTED ANALYSIS
AND DESIGN**

AY-2020-2021

SEMESTER-VI

JANUARY-MAY



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



MODULE-1

COURSE OUTCOMES:

- Define the basic concepts of Object Oriented approaches**



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



MODULE I

INTRODUCTION TO OBJECT ORIENTED SYSTEM

Object Basics-Object Oriented System Development Life Cycle- Use case driven approach-Rumbaugh Object Model- Booch Methodology-Jacobson Methodology- Unified Approach



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



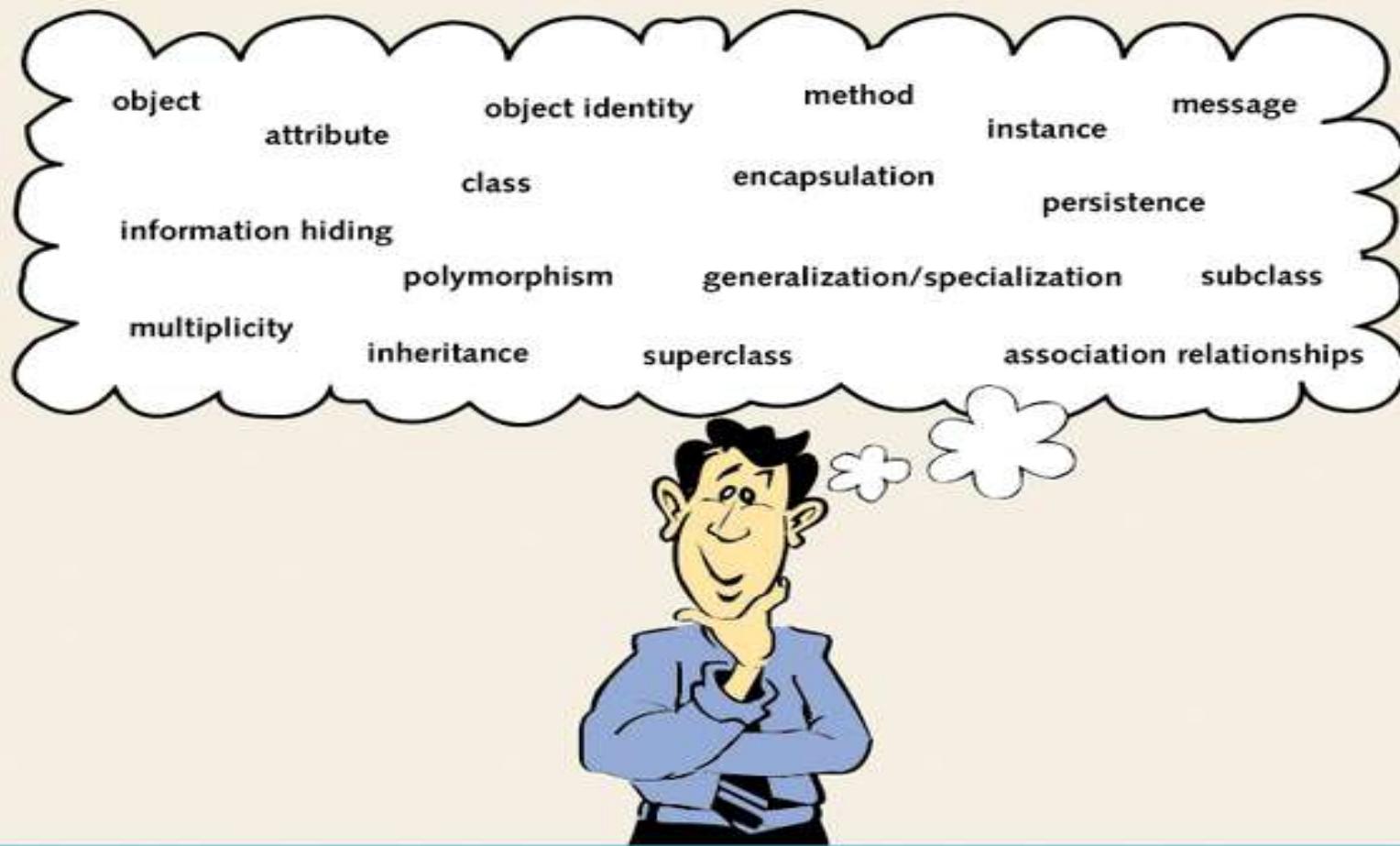


Figure 1-5 Key OO concepts



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Objects

- An object is a **real-world element** in an object-oriented environment that may have a **physical or a conceptual existence**. Each object has –
- **Identity** that distinguishes it from other objects in the system.
- State that **determines the characteristic properties** of an object as well as the values of the properties that the object holds.

Objects

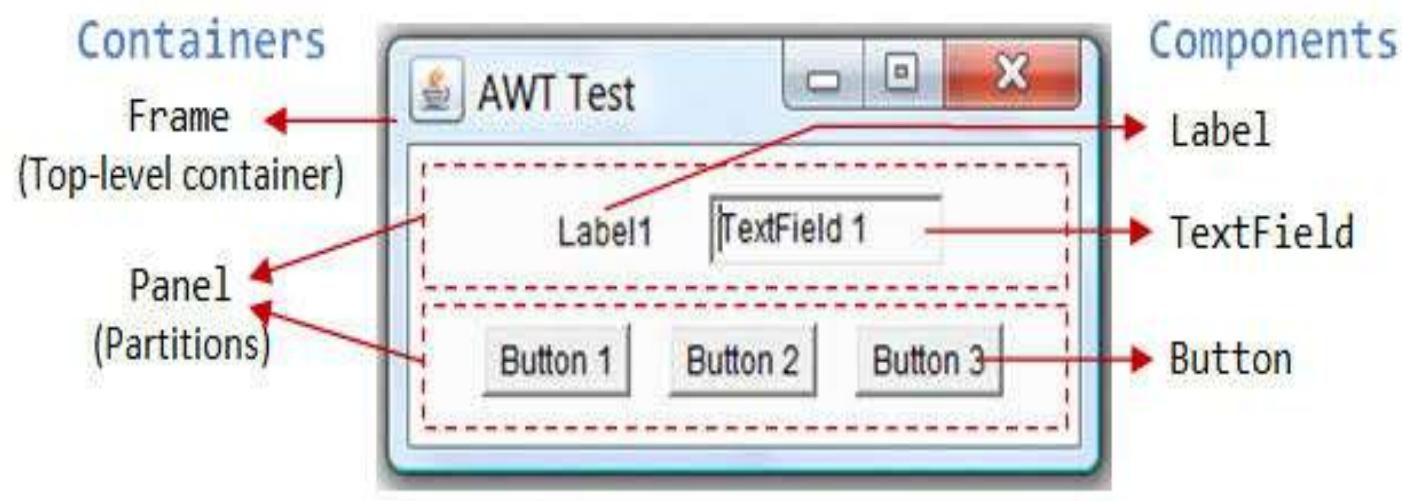
- Behavior that represents **externally visible activities** performed by an object in terms of changes in its state.
- Objects can be modelled according to the needs of the application.
- An object may have a physical existence, like a **customer, a car, etc.**; or an intangible conceptual existence, like a project, a process, etc.

Objects

- Most basic component of OO design. Objects are designed to do a *small, specific piece of work.*
- Objects represent the various components of a business system

Examples of Different Object Types

- GUI objects
 - objects that make up the user interface
 - e.g. buttons, labels, windows, etc.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Examples of Different Object Types

- Problem Domain objects
 - Objects that represent a business application
 - A problem domain is the scope of what the system to be built will solve. It is the business application.
 - e.g. An order-entry system
 - A payroll system
 - A student system

Sample Problem Domain

Company ABC needs to implement an order-entry system that takes orders from customers for a variety of products.

What are the objects in this problem domain?

Hint: Objects are usually described as nouns.

Possible Objects for this Problem Domain

Customer

Order

Product

Classes and Objects

Class

- A class represents a **collection of objects having same characteristic properties** that exhibit common behavior.
- It gives the blueprint or description of the objects that can be created from it. Creation of an object as a member of a class is called instantiation. Thus, object is an instance of a class.

Classes and Objects

- **Classes**
 - Define what all objects of the class represent
 - It is like a blueprint. It describes what the objects look like
 - They are a way for programs to model the real world
- **Objects**
 - Are the instances of the class

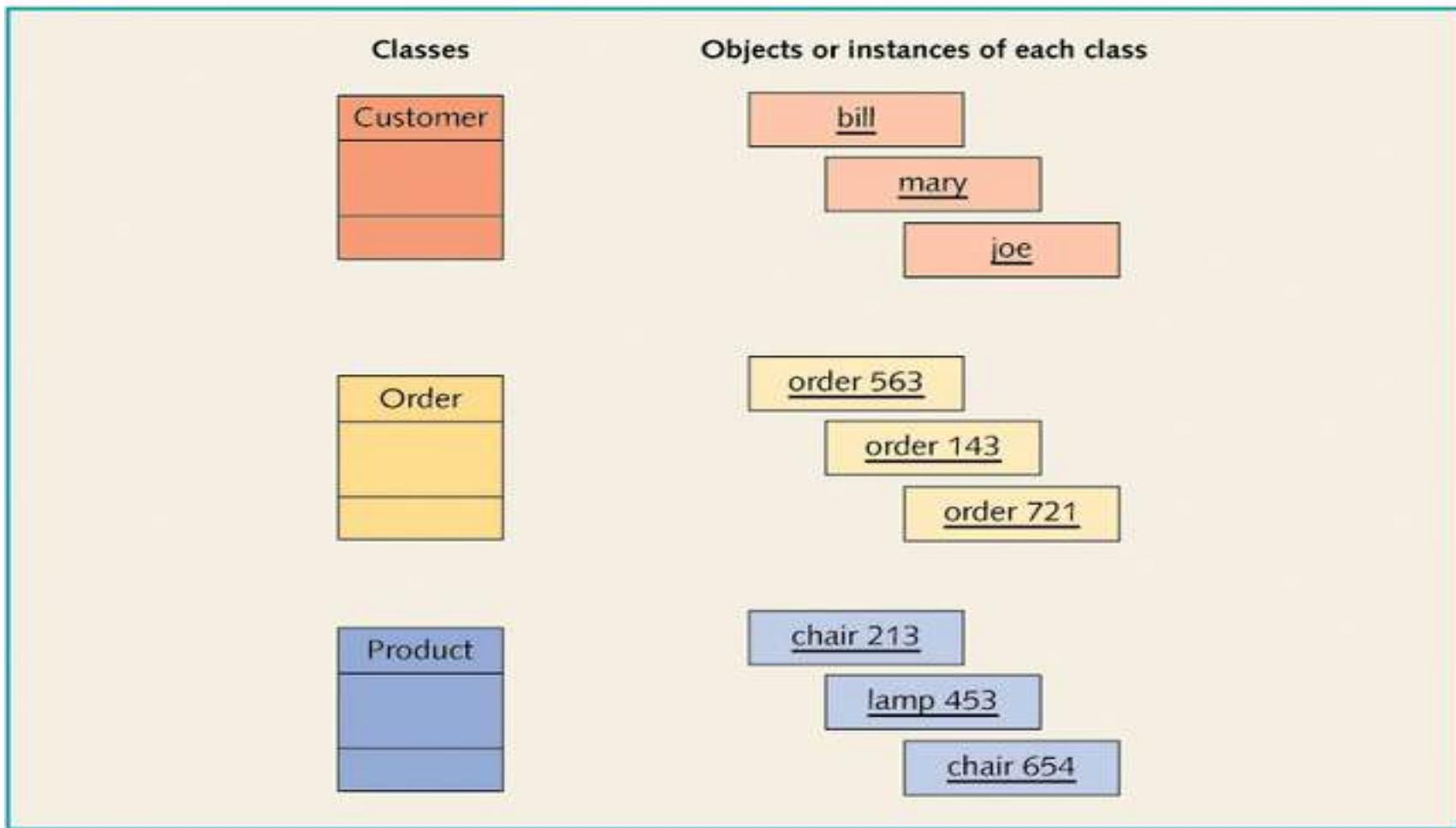


Figure 1-9 Class versus objects or instances of the class

Object Attributes and Methods

- **Classes contain two things:**
 - **Fields** (attributes, data members, class variables):
 - Data items that differentiate one object of the class from another. e.g. employee name, student number
 - Characteristics of an object that have values
 - What are some possible attributes for the customer object?
 - **Methods** (behaviors):
 - Named, self-contained blocks of code that typically operate on the fields
 - Describe what an object can do
 - Can be thought of as the verbs in a problem domain
 - What are some possible methods for the customer object?

Problem Domain Objects	Attributes	Methods
Customer	name, address, phone number	set name, set address, add new order for customer
Order	order number, date, amount	set order date, calculate order, amount, add product to order, schedule order shipment
Product	product number, description, price	add to order, set description, get price

Figure 1-7 Attributes and methods in problem domain objects

GUI Objects	Attributes	Methods
Button	size, shape, color, location, caption	click, enable, disable, hide, show
Label	size, shape, color, location, text	set text, get text, hide, show
Form	width, height, border style, background color	change size, minimize, maximize, appear, disappear

Figure 1-6 Attributes and methods of GUI objects

Class member:

1. data member

2. Behavior

Class bicycle{

 int gear;

 Void braking()

 {sop("I stop");}

}

}

 Bicycle b1= new Bicycle();

 Bicycle b2= new Bicycle();

Class SportsBicycle extends bicycle

{

 Int tyre=0;

 Void sporting(){}

}

Object Interactions and Messages

- **Objects interact with other objects in a variety of relationships**
 - e.g. one-to-one, one-to-many
- **Messages**
 - The means by which objects interact
 - Example:
 - User initiates interaction via messages to GUI objects
 - GUI objects interact with problem domain objects via messages
 - Problem domain objects interact with each other and GUI objects via messages
 - GUI objects respond to user via messages



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



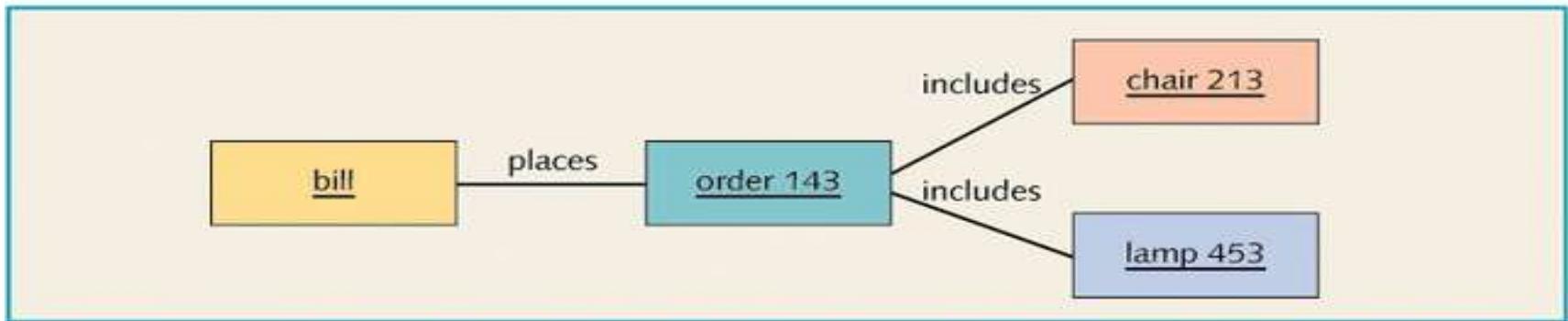


Figure 1-10 Associating objects with other objects

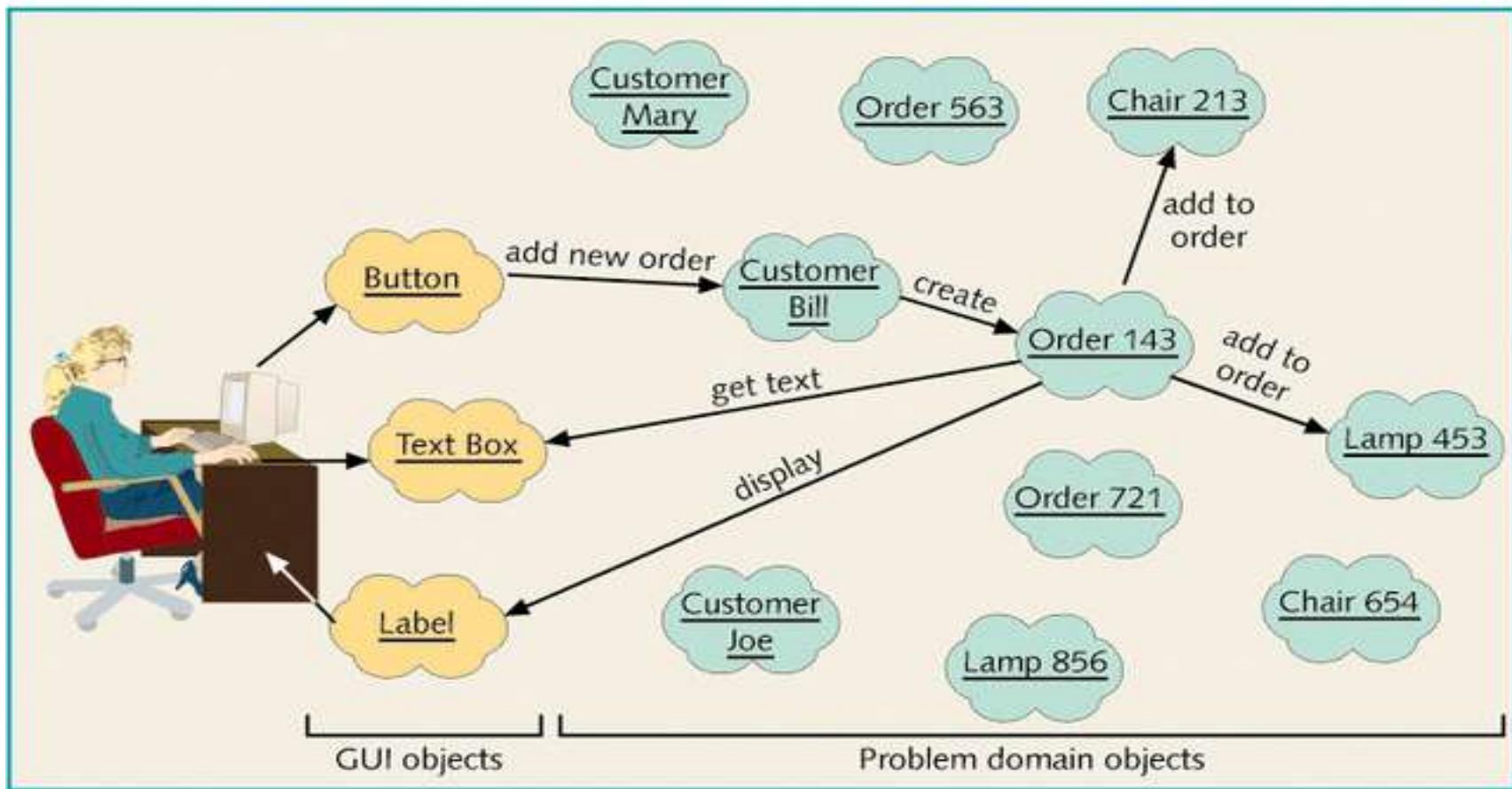


Figure 1-8 Order-processing system where objects interact by sending messages

Inheritance and Polymorphism

- 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.

Inheritance and Polymorphism

- 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.

Example

- From a class Mammal, a number of classes can be derived such as Human, Cat, Dog, Cow, etc. Humans, cats, dogs, and cows all have the distinct characteristics of mammals. In addition, each has its own particular characteristics.
- It can be said that a cow “is – a” mammal.

Types of Inheritance

- **Single Inheritance** – A subclass derives from a single super-class.
- **Multiple Inheritance** – A subclass derives from more than one super-classes.
- **Multilevel Inheritance** – A subclass derives from a super-class which in turn is derived from another class and so on.
- **Hierarchical Inheritance** – A class has a number of subclasses each of which may have subsequent subclasses, continuing for a number of levels, so as to form a tree structure.
- **Hybrid Inheritance** – A combination of multiple and multilevel inheritance so as to form a lattice structure.

Vehicle

Car

Car

Bus

Base Class 1

Base Class 2

Vehicle

Single Inheritance

Multiple Inheritance

Derive Class

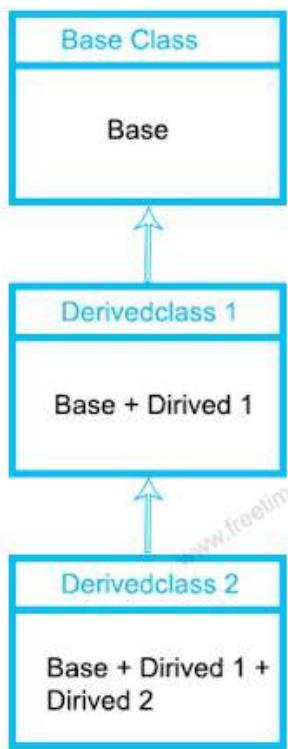
Multiple Inheritance



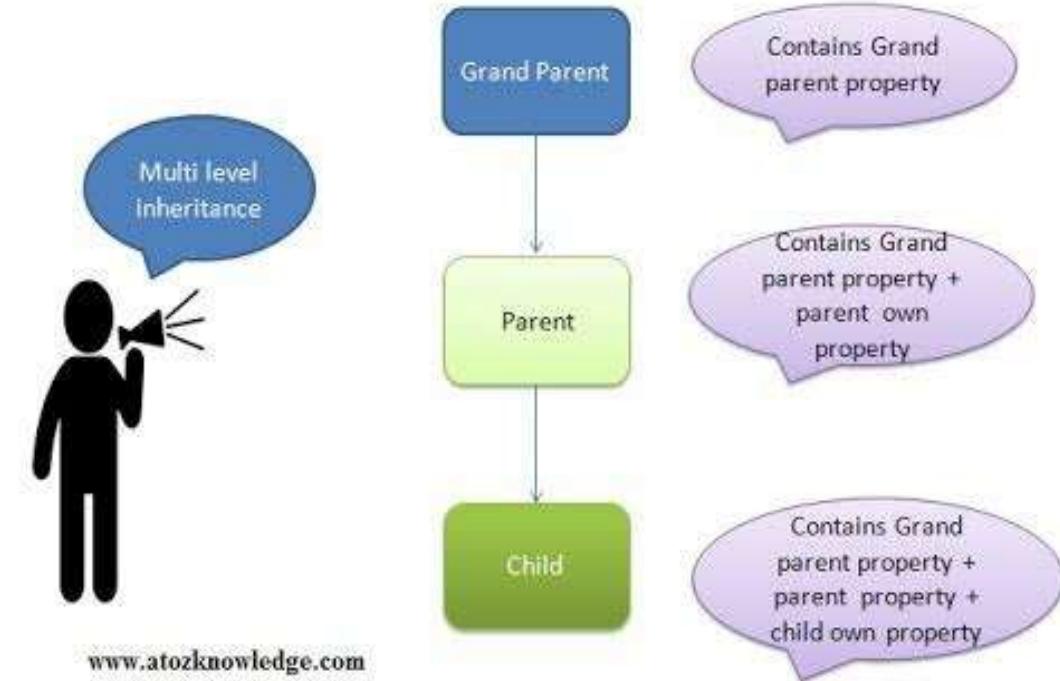
PRESIDENCY
UNIVERSITY

Private University Estd. in Karnataka State by Act No. 41 of 2013





Multilevel Inheritance



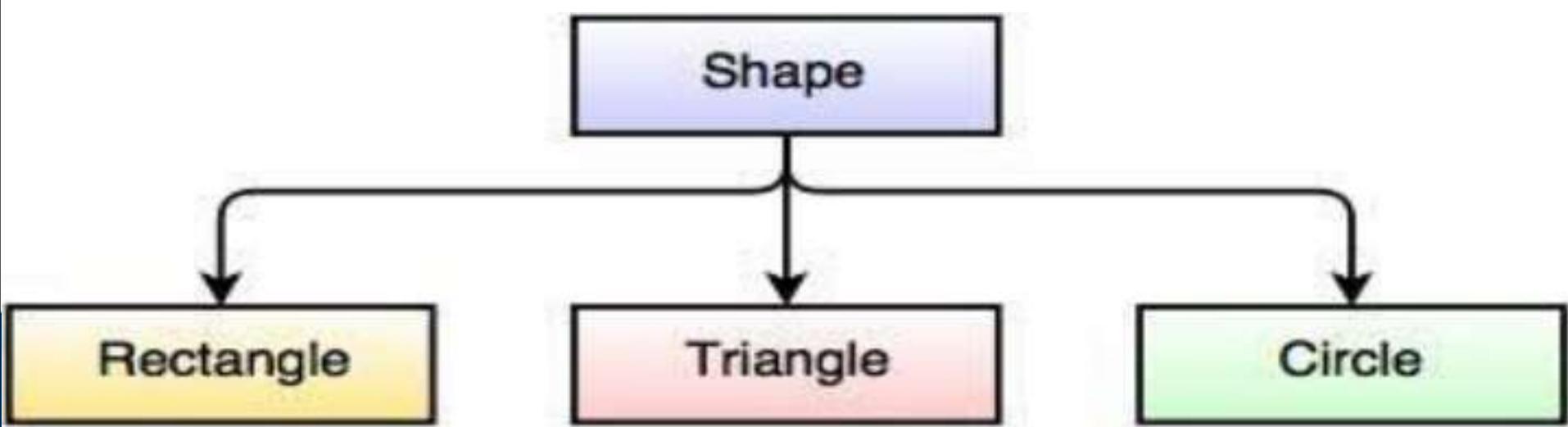
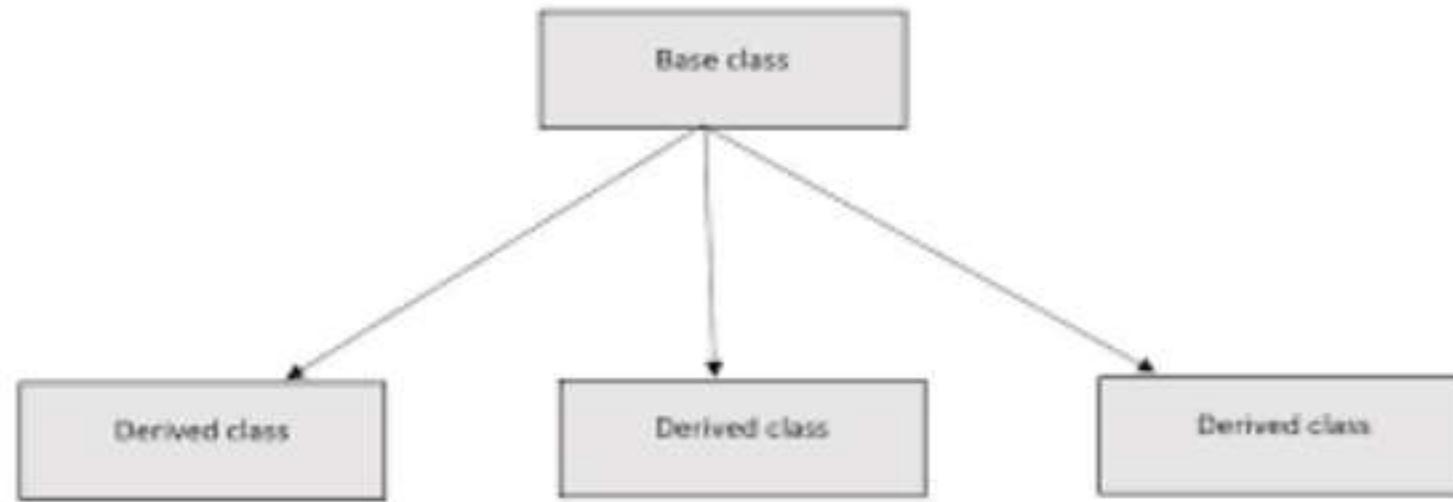
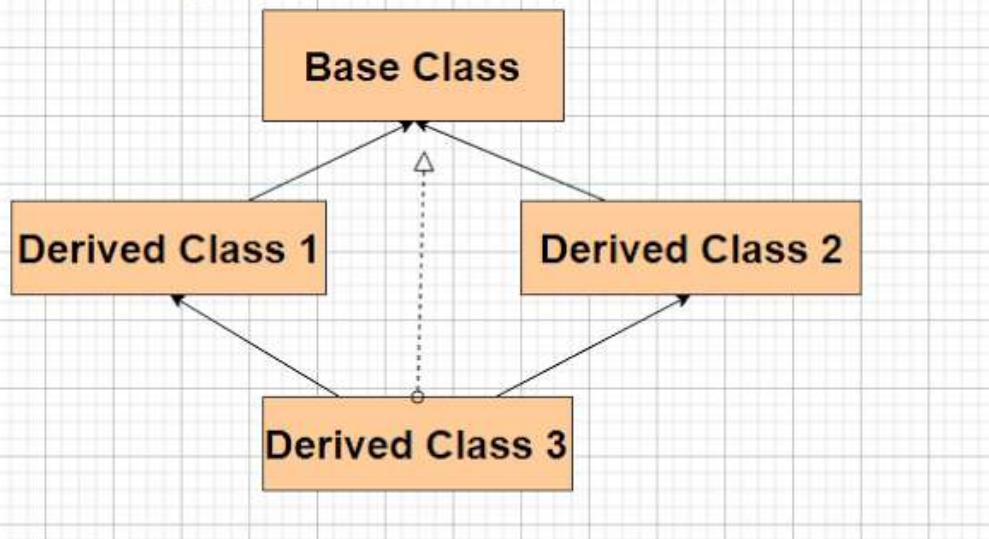


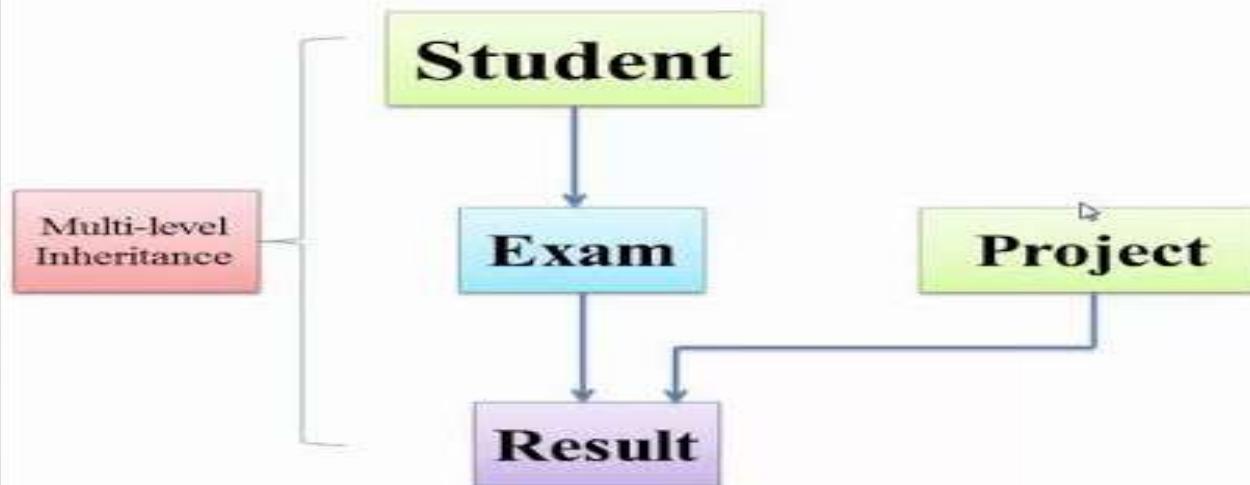
Fig. Hierarchical Inheritance with Class Shape

Hybrid Inheritance



Hybrid Inheritance

Combination of two or more type of inheritance to design a program.



Inheritance and Polymorphism

- **Inheritance**

- One class of objects takes on characteristics of another class and extends them
- Superclass → subclass
- Generalization/specialization hierarchy
 - Also called an inheritance hierarchy
 - Result of extending class into more specific subclasses
- **This is an important concept!!**

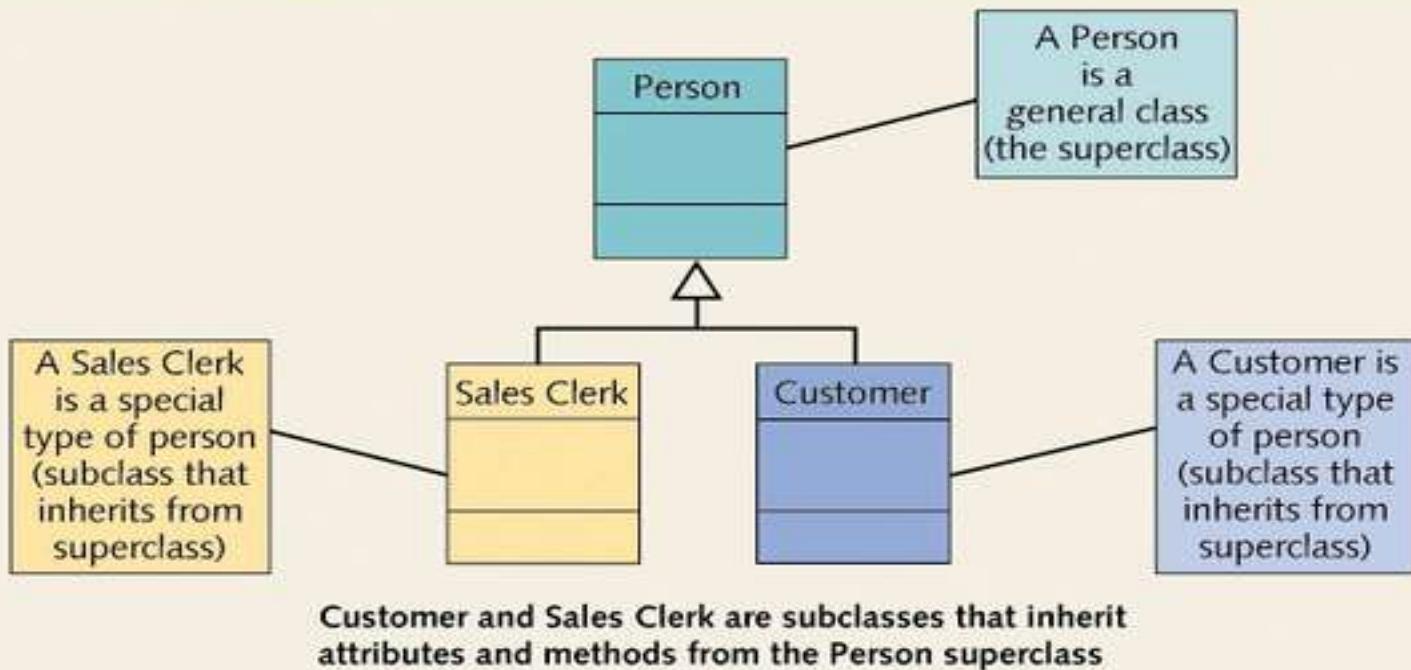


Figure 1-11 Superclass and subclass

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.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Inheritance and Polymorphism

- **Polymorphism**

- literally means “many forms”
- in Java means using the same message (or method name) with different classes
 - different objects can respond in their own way to the same message
 - e.g. `toString()`



Figure 1-12 Polymorphism for two different types of bank accounts

Association

- Association is a **group of links having common structure and common behavior**. Association depicts the **relationship between objects of one or more classes**. A link can be defined as an instance of an association.

Degree of an Association

- Degree of an association denotes the number of classes involved in a connection. Degree may be unary, binary, or ternary.
- A **unary relationship** connects objects of the same class.
- A **binary relationship** connects objects of two classes.
- A **ternary relationship** connects objects of three or more classes.

Cardinality Ratios of Associations

- Cardinality of a binary association denotes the **number of instances participating in an association**. There are three types of cardinality ratios, namely –
- **One-to-One** – A single object of class A is associated with a single object of class B.
- **One-to-Many** – A single object of class A is associated with many objects of class B.
- **Many-to-Many** – An object of class A may be associated with many objects of class B and conversely an object of class B may be associated with many objects of class A.



one-to-one

Each Department must be managed by one Manager.
Each Manager may manage one Department.

Information Engineering Style

one to one



one to many (mandatory)



many



one or more (mandatory)



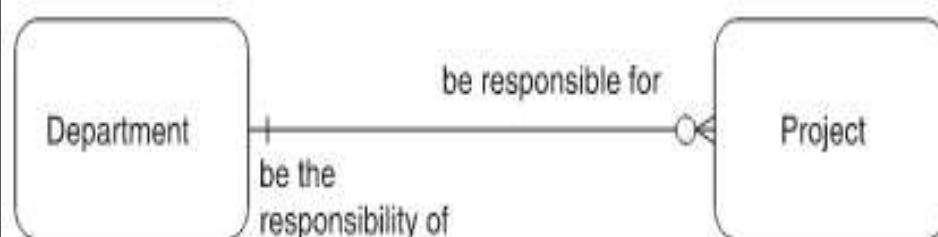
one and only one (mandatory)



zero or one (optional)

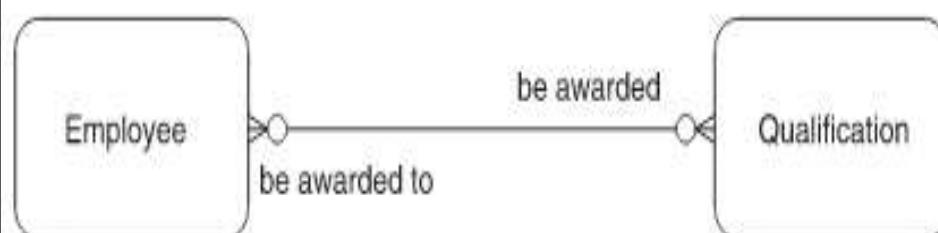


zero or many (optional)



one-to-many

Each Department may be responsible for one or more Projects.
Each Project must be the responsibility of one Department.



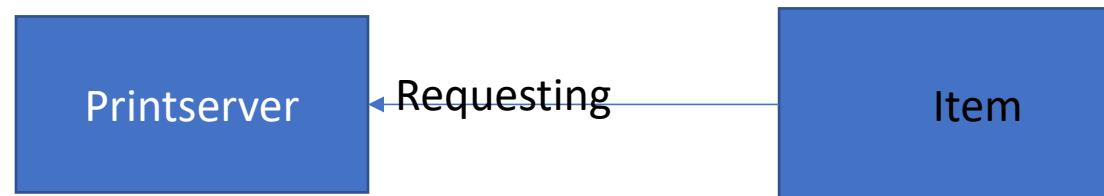
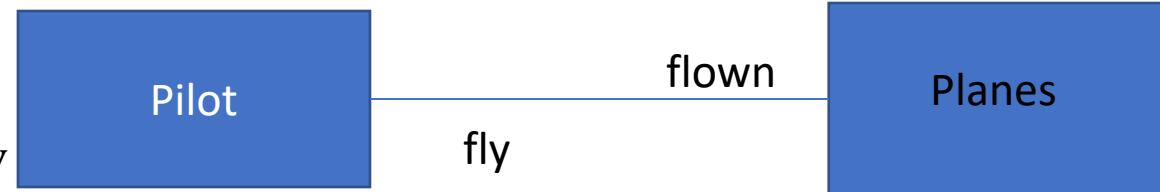
many-to-many

Each Employee may be awarded one or more Qualifications.
Each Qualification may be awarded to one or more Employees.



Objects relationships: Association

- Associations:
 - Bidirectional
 - Cardinality
 - One to one
 - One to many
 - Many to many
- Consumer-Producer Association



Aggregation or Composition

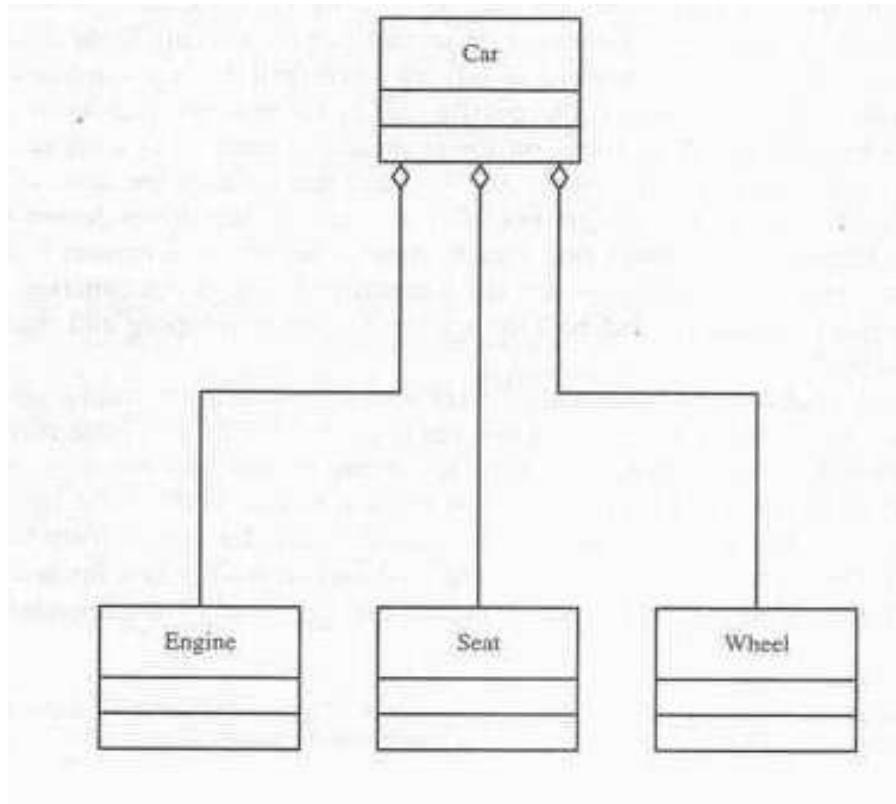
- Aggregation or composition is a relationship among classes by which a class can be made up of any combination of objects of other classes.
- It allows objects to be placed directly within the body of other classes.
- Aggregation is referred as a “part-of” or “has-a” relationship, with the ability to navigate from the whole to its parts. An aggregate object is an object that is composed of one or more other objects.

Aggregation or Composition

Example

- In the relationship, “a car has-a motor”, car is the whole object or the aggregate, and the motor is a “part-of” the car. Aggregation may denote –
- **Physical containment** – Example, a computer is composed of monitor, CPU, mouse, keyboard, and so on.
- **Conceptual containment** – Example, shareholder has-a share.

Objects relationships: Aggregation



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Aggregation

- All objects except the most basic ones, composed of and may contain other objects.
- eg: Spread sheets composed of cells and cells are objects contains texts, mathematical formulas,videos,etc..
- Each objects has an identity, one object can refer other objects

Static vs Dynamic Binding

- **Static Binding:** The binding which can be resolved at compile time by compiler is known as static or early binding. Binding of all the static, private and final methods is done at compile-time .
- **Dynamic Binding:** In Dynamic binding compiler doesn't decide the method to be called. Overriding is a perfect example of dynamic binding. In overriding both parent and child classes have same method .

- Main()

```
{...
```

```
..
```

```
..
```

```
Add()
```

```
}
```

```
Add()
```

```
{  
}
```



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Overloading occurs when two or more methods in one class have the same method name but different parameters.

```
class Cat{  
  
    public void Sound(){  
        System.out.println("meow");  
    }  
  
    //overloading method  
    public void Sound(int num){  
        for(int i=0; i<2;i++){  
            System.out.println("meow");  
        }  
    }  
}
```

OVERLOADING

Same method
name but
different
parameters



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Overriding means having two methods with the same method name and parameters (i.e., method signature). One of the methods is in the parent class and the other is in the child class.

```
class Cat{  
    public void Sound(){  
        System.out.println("meow");  
    }  
}  
  
class Lion extends Cat{  
    public void sniff(){  
        System.out.println("sniff");  
    }  
  
    public void Sound(){  
        System.out.println("roar");  
    }  
}
```

Overriding

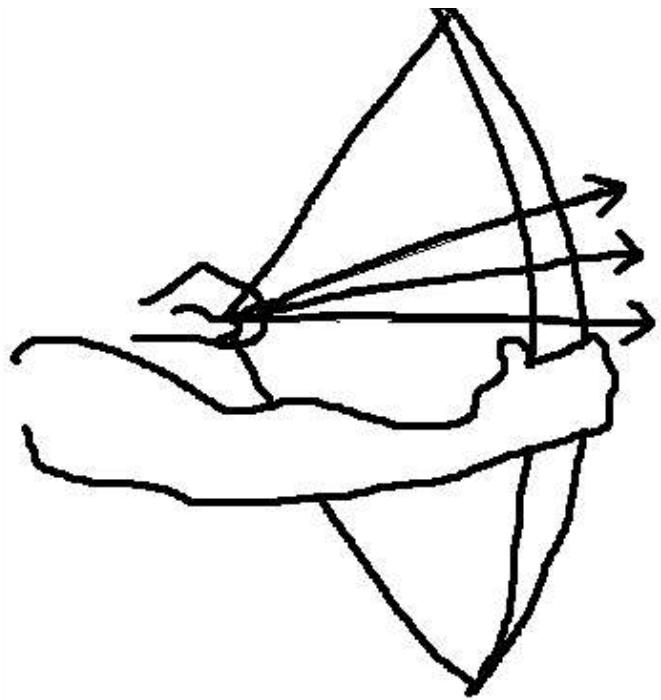
**Same method
name and same
parameters**



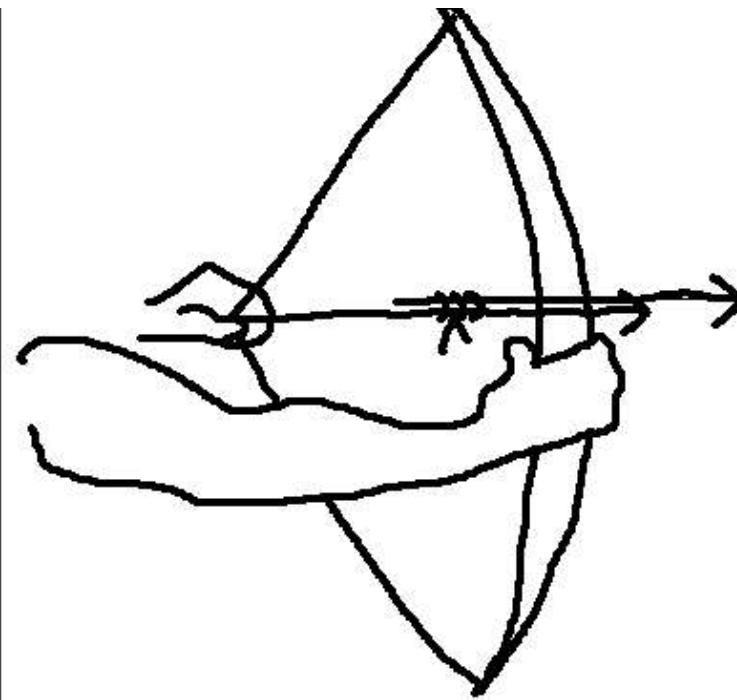
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





Overloading



Overriding



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013

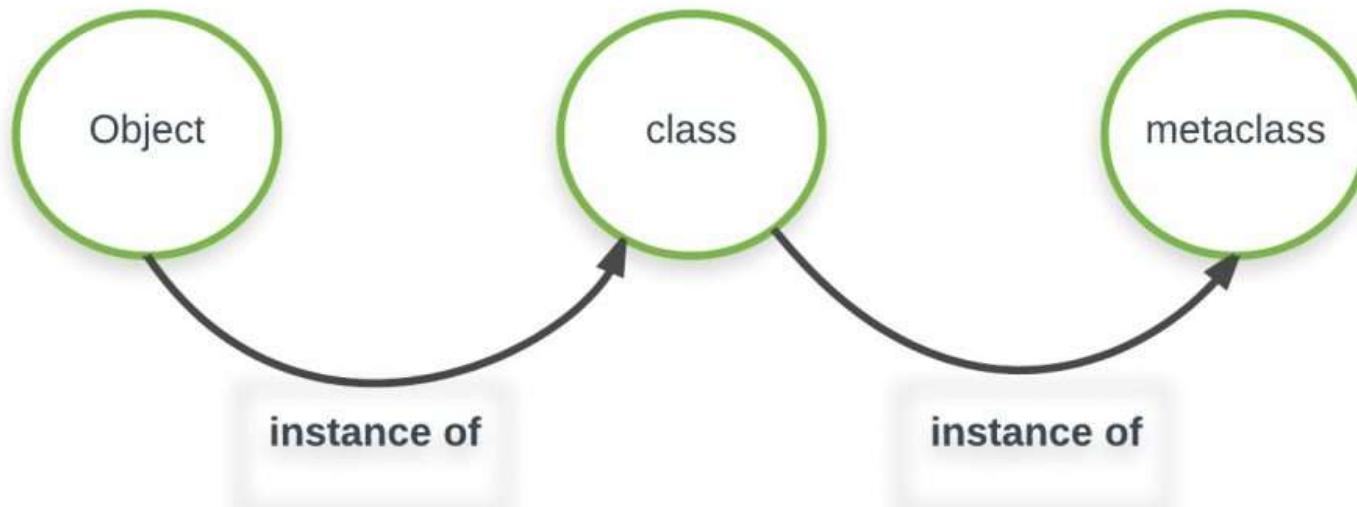


Differences Between Static Binding And Dynamic Binding In Java

The above findings can be summarized like below.

Static Binding	Dynamic Binding
It is a binding that happens at compile time.	It is a binding that happens at run time.
Actual object is not used for binding.	Actual object is used for binding.
It is also called early binding because binding happens during compilation.	It is also called late binding because binding happens at run time.
Method overloading is the best example of static binding.	Method overriding is the best example of dynamic binding.
Private, static and final methods show static binding. Because, they can not be overridden.	Other than private, static and final methods show dynamic binding. Because, they can be overridden.

Meta-Classes



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- **A Class is also an object**, and just like any other object it's a instance of something called **Metaclass**.
- A special class **type** creates these *Class* object.
- The **type** class is default **metaclass** which is responsible for making classes.

Reference

- “Object Oriented Systems Development using the unified modeling language” Ali Bahrami, TATA McGraw-Hill Edition.

General Objective

- Students will be able to understand the **Object oriented systems development life cycle.**



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Specific Objectives

Students will be able to :

1. List the three transformation of **software development process** [R,C,T].
2. Examine the need of **building high quality software** [An,C,T].
3. Discuss a **Use case driven approach** [U,C,T].

Introduction

- Analysis, design, implementation, testing & refinement to transform users' need into software solution that satisfies those needs
- Object-oriented approach
 - ✓ more rigorous process to do things right
 - ✓ more time spent on gathering requirements, developing requirements model & analysis model, then turn into design model
 - ✓ need not see code until after 25% development time

Software Development Life Cycle (SDLC)

- Software Development Life Cycle (SDLC) is a process used by the **software industry** to design, develop and test high quality softwares.
- The SDLC aims to produce a **high-quality software** that meets or exceeds customer expectations, reaches completion within times and cost estimates.
- SDLC is a framework defining tasks performed at each step in the software development process.

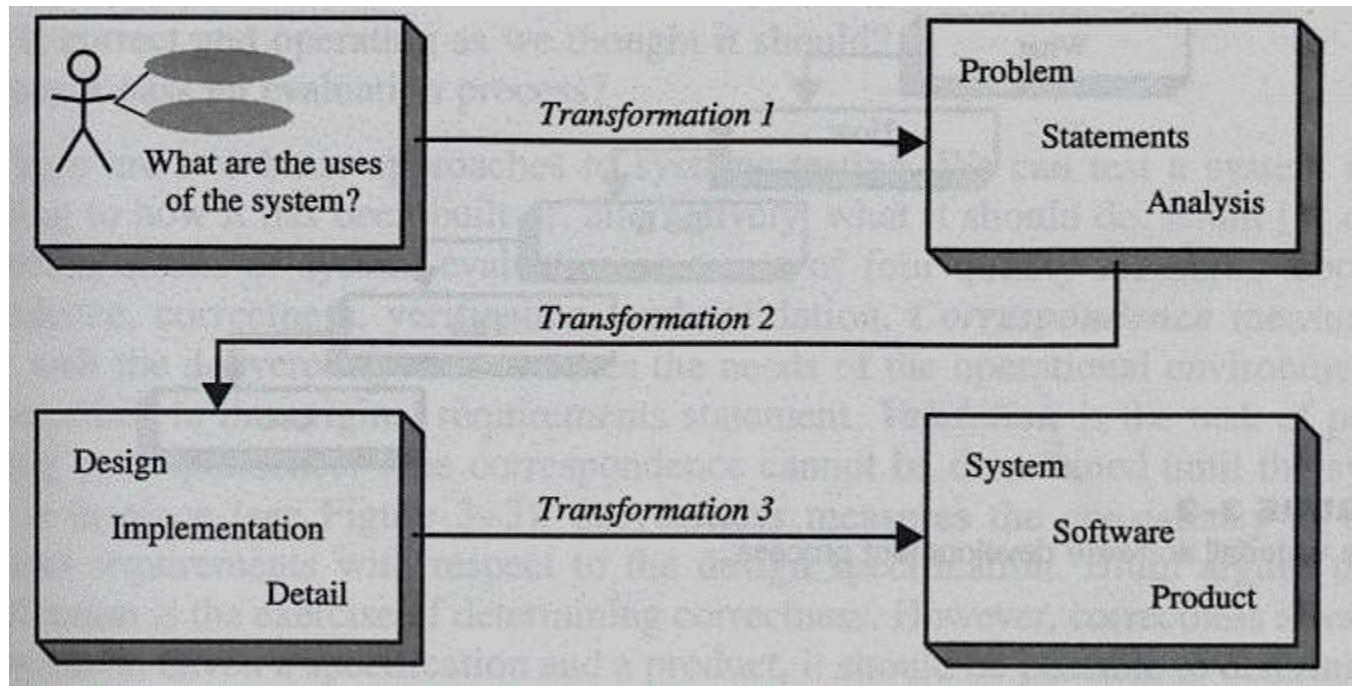


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



SO1: List the three transformation of software development process [U][C][T]



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



1. **transformation 1 (analysis)** - translates user's need into system's requirements & responsibilities
2. **transformation 2 (design)** - begins with problem statement, ends with detailed design that can be transformed into operational system
3. **transformation 3 (implementation)** - refines detailed design into system deployment that will satisfy user's needs

SDLC Models

- Waterfall Model
- Iterative Model
- Spiral Model
- V-Model
- Big Bang Model

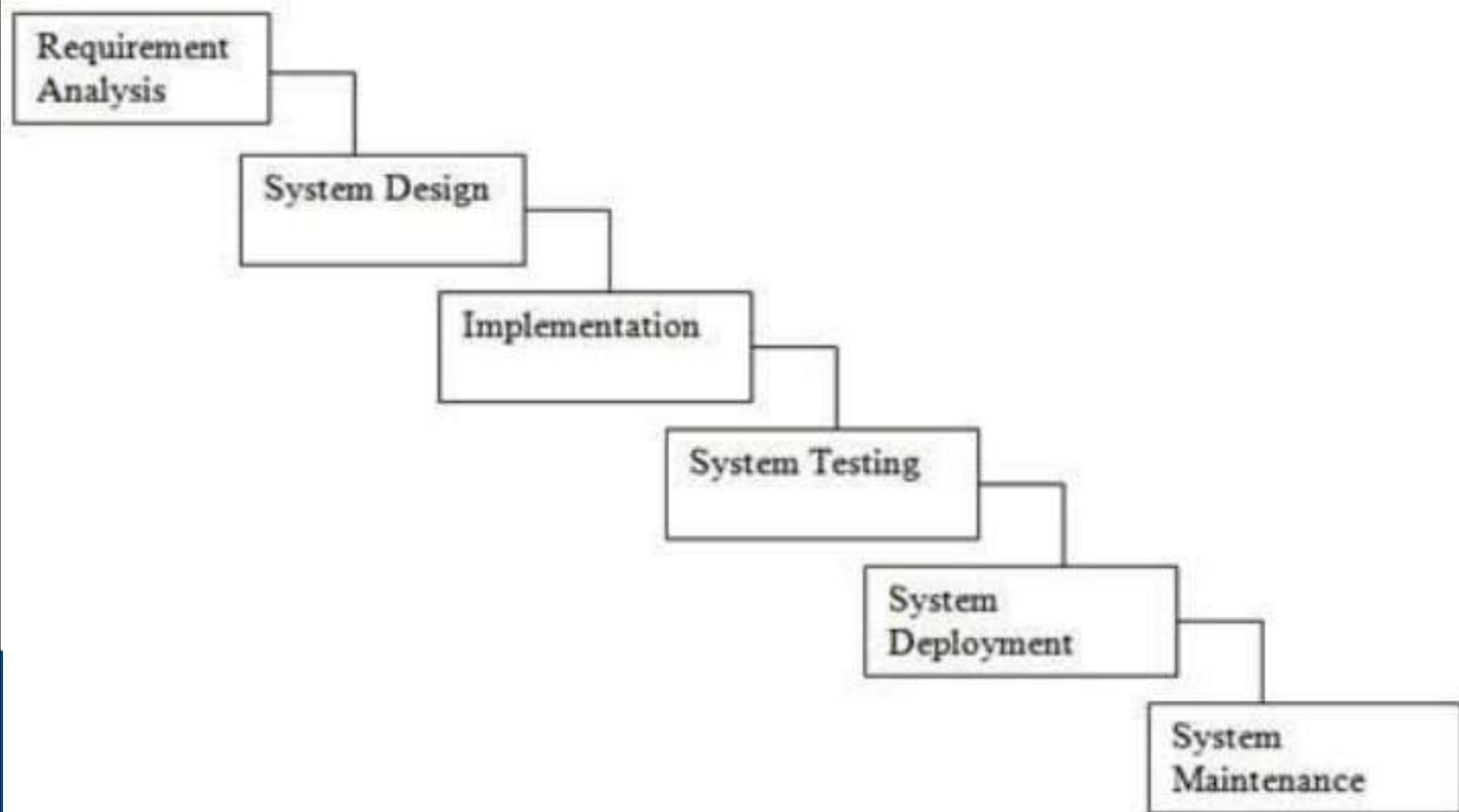


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Waterfall software development process



Why waterfall model fails?

1. When **there is uncertainty** regarding what's required or how it can be built
2. Assumes **requirements are known** before design begins
3. Assumes **requirements remain static** over development cycle
4. Assumes **sufficient design knowledge** to build product
5. Problem if **environment changes**

SO2:Examine the need of building high quality software [An,C,T]

- Goal is user satisfaction
 - To achieve high quality in software we need to be able to answer the following questions
1. how do we determine system is ready for delivery?
 2. Is it now an operational system that satisfies user ' s needs?
 3. Is it correct and operating as we thought it should ?
 4. Does it pass an evaluation process ?

Approaches to systems testing

Test according to

- ✓ how it has been built
- ✓ what it should do

4 quality measures

1.correspondence

measures how well delivered system matches needs of operational environment, as described in original requirements statement

2.validation

task of predicting correspondence (true correspondence only determined after system is in place)



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013

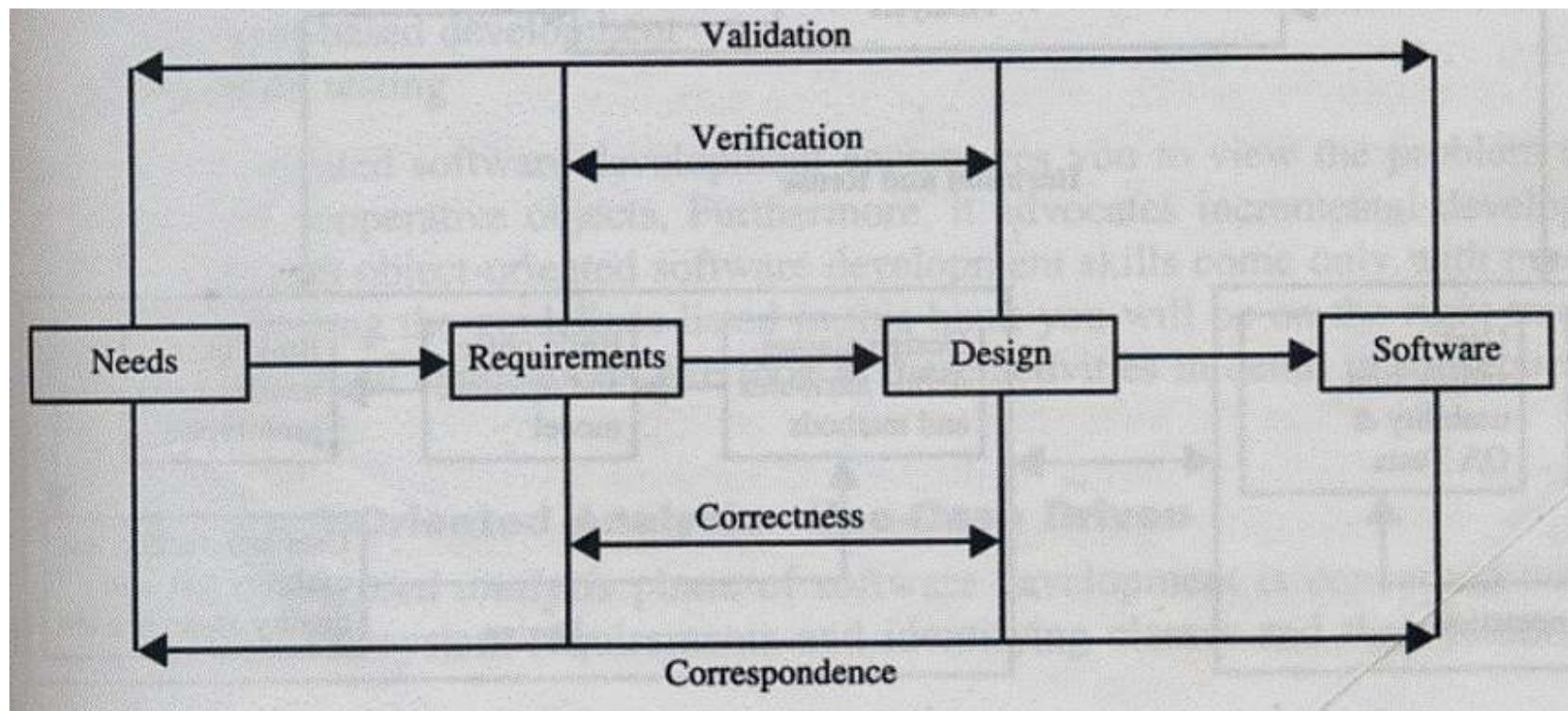


3. correctness

measures consistency of product requirements with respect to design specification

4. Verification

Exercise of determining correctness (correctness objective
=> always possible to determine if product precisely satisfies requirements of specification)



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Verification vs Validation

Verification

- am I building the product right ?
- Begin after specification accepted

Validation

- am I building the right product ?
- Begins as soon as project starts

Verification & validation independent of each other

SO3: Object-Oriented approach: A use-case driven approach[U][C][T]

Object-oriented software development life cycle consists of

- Object-oriented analysis
- Object-oriented design
- Object-oriented implementation

Object-Oriented Analysis

- In this stage, the problem is formulated, user requirements are identified, and then a model is built based upon real-world objects.
- The analysis produces models on how the desired system should function and how it must be developed.
- The models do not include any implementation details so that it can be understood and examined by any non-technical application expert.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Object-Oriented Design

System Design

- In this stage, the **complete architecture of the desired system is designed.**
- System design is done according to both the **system analysis model and the proposed system architecture.**

Object Design

- In this phase, a design model is developed based on both the models developed in the system **analysis phase and the architecture designed in the system design phase.** All the classes required are identified.

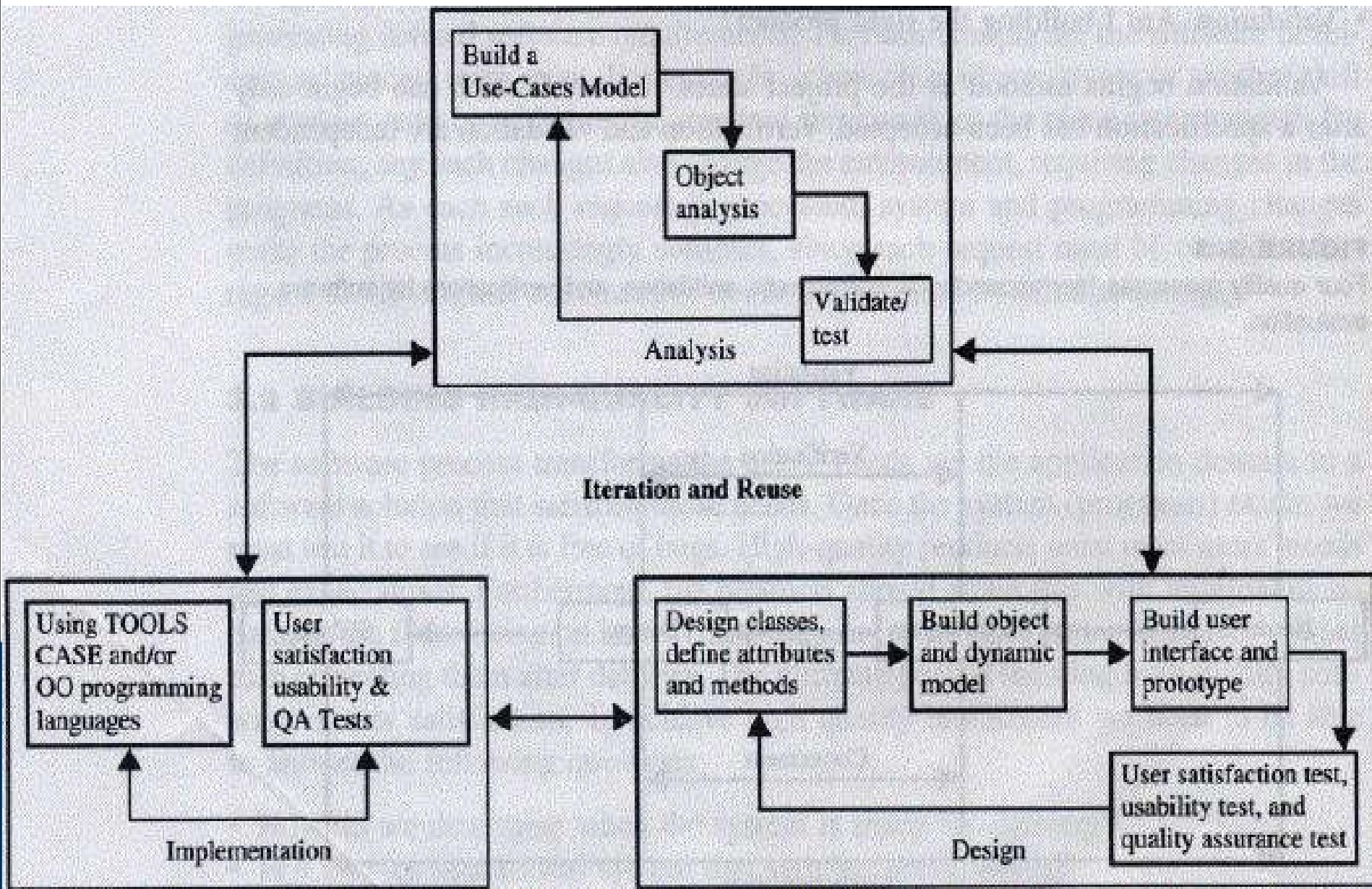
The designer decides whether –

- new classes are to be **created from scratch**,
- any **existing classes can be used in their original form**, or
- new classes should be inherited from the existing classes.
- The **associations** between the identified classes are established and the hierarchies of classes are identified.
- Besides, the developer designs the internal details of the classes and their associations, i.e., the data structure for each attribute and the algorithms for the operations.

Object–Oriented Implementation and Testing

- In this stage, the design model developed in the object design is **translated into code in an appropriate programming language or software tool.**
- The **databases are created and the specific hardware requirements are ascertained.**
- Once the code is in shape, it is tested using specialized techniques to identify and remove the errors in the code.

Object-oriented Systems Development Approach



Object-oriented software development activities

- Object-oriented analysis - use case driven
- Object-oriented design
- Prototyping
- Component-based development
- Incremental testing



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Summary

- Software development process
- High quality software
- Verification
- Validation

References

1. Ali Bahrami, Object Oriented Systems Development, Third Edition, Tata McGraw-Hill, 2012

Stimulating Questions

1. How is software verification different from validation?



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Thank You !!!



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Object-Oriented Systems Development : A Use-Case Driven Approach

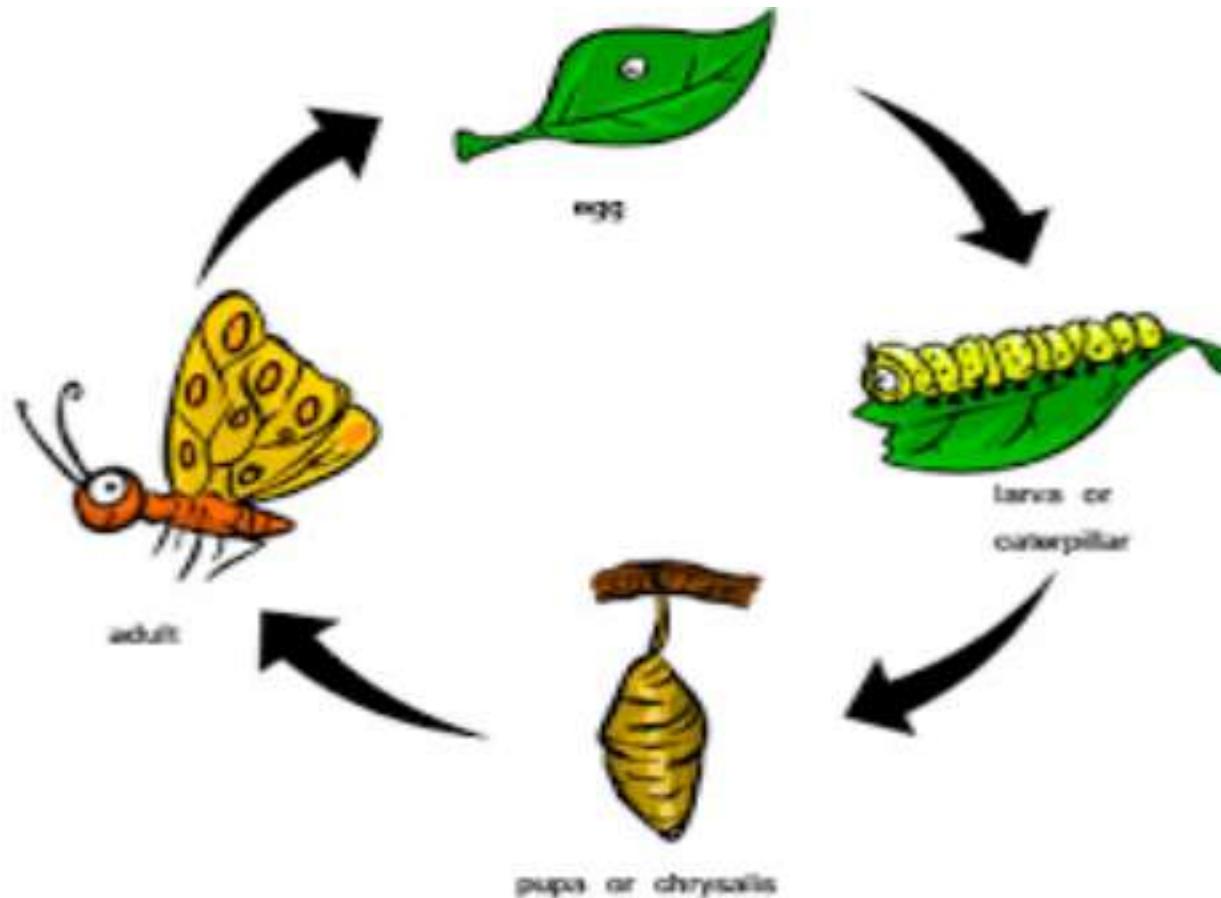


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Evocation



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





Goals (Con't)

- Use-case driven systems development
- Prototyping
- Rapid application development
- Component-based development
- Continuous testing and reusability

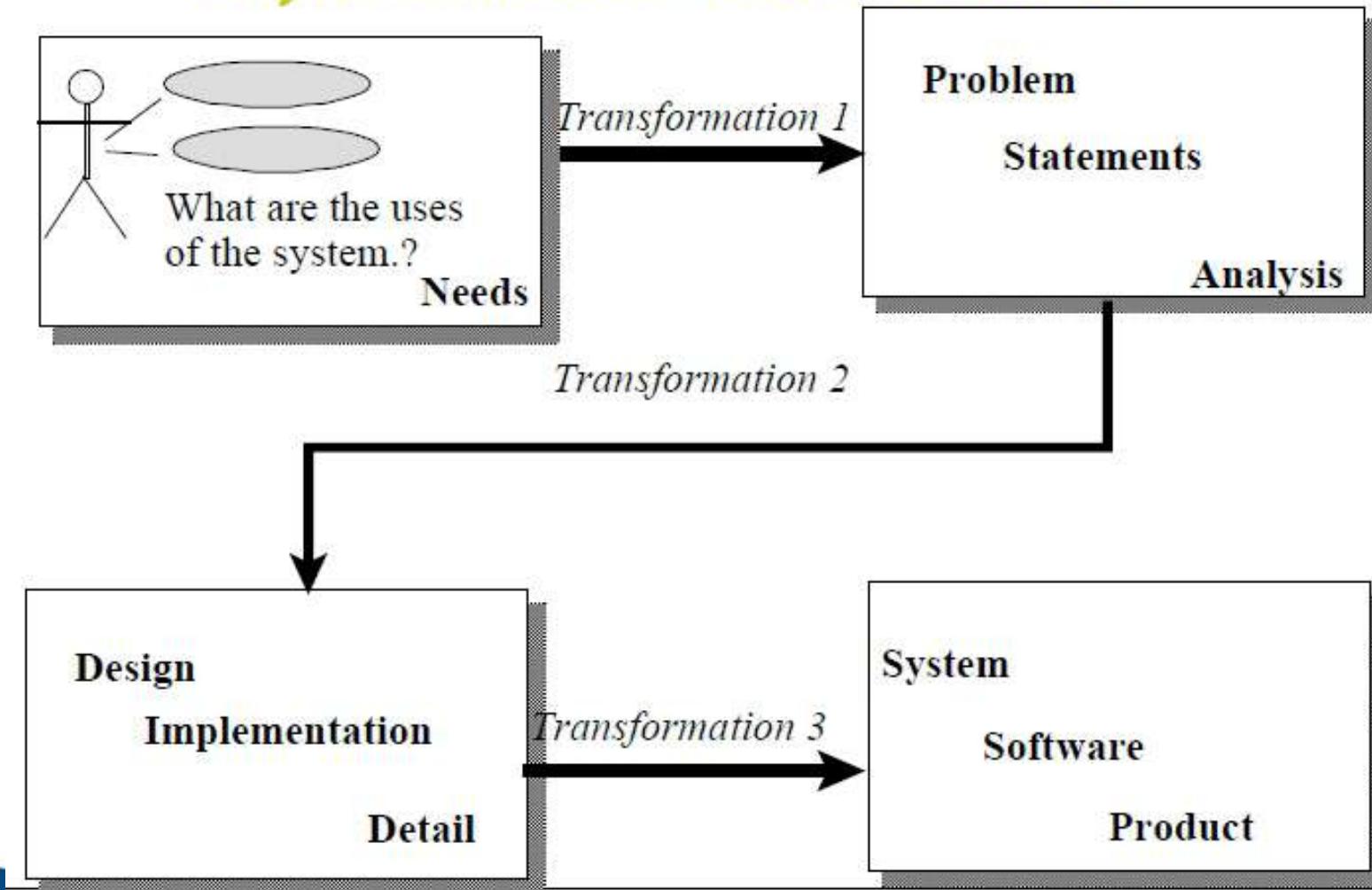


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Software Process (Con't)



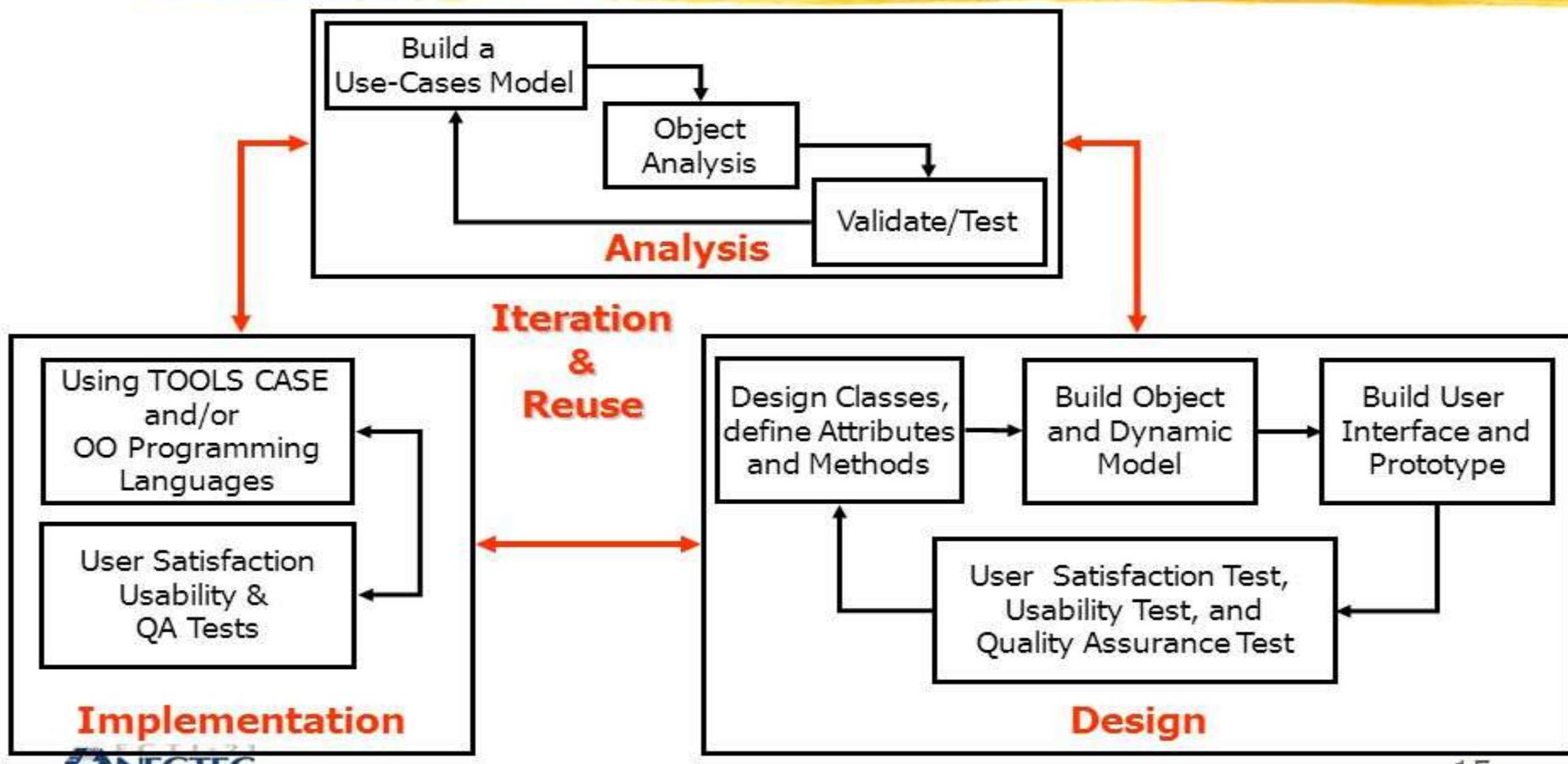
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



1. **transformation 1 (analysis)** - translates user's need into system's requirements & responsibilities
2. **transformation 2 (design)** - begins with problem statement, ends with detailed design that can be transformed into operational system
3. **transformation 3 (implementation)** - refines detailed design into system deployment that will satisfy user's needs

Object-Oriented Systems Development Life Cycle



Object-Oriented Systems

Development activities

- Object-oriented analysis.
- Object-oriented design.
- Prototyping.
- Component-based development.
- Incremental testing.



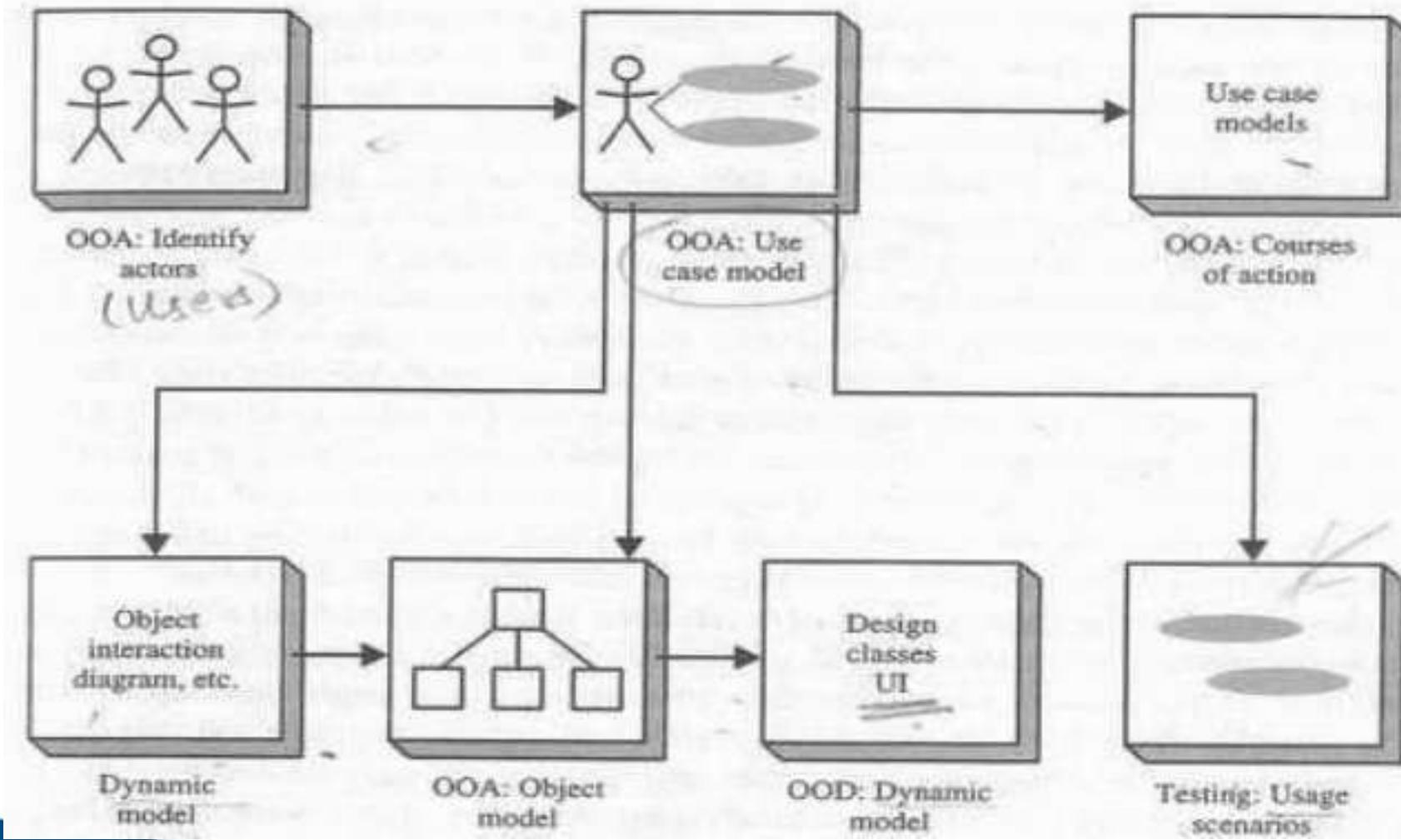
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



OO Analysis

- Use case [Ivar Jacobson] – user computer interaction
- Collaborations
- Use Case Modeling
 - Identify the Actors
 - Clarify their interaction
 - Analyze their action
- Documentation
 - [80 -20 Rule]



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



OO Design

- Identify the object model [objects & their relationships]
 - Design & refine classes
 - Design & refine attributes
 - Design & refine methods
 - Design & refine structures
 - Design & refine associations

Contd,...

- Guidelines for Design
 - Reuse rather than build a new class
 - Design large number of simple classes
 - Design methods
 - Critique what you have proposed. [Go back and refine if possible]

Prototyping



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Contd,...

- Types of Prototype:-
 - Horizontal prototype: **Simulation of the interface**
 - Vertical prototype: subset of the system features with complete Functionality
 - Analysis prototype: aid for exploring the problem domain
 - Domain prototype: aid for the incremental development of the ultimate software solution

Implementation

- Custom development [target audience]
- Component Based Development [CBD]
- Rapid Application Development
 - Assemble pre-built, pretested & reusable components.

Component-based development (CBD)

- CBD is an industrialized approach to the software development process.
- Application development moves from custom development to assembly of pre-built, pre-tested, reusable software components that operate with each other.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



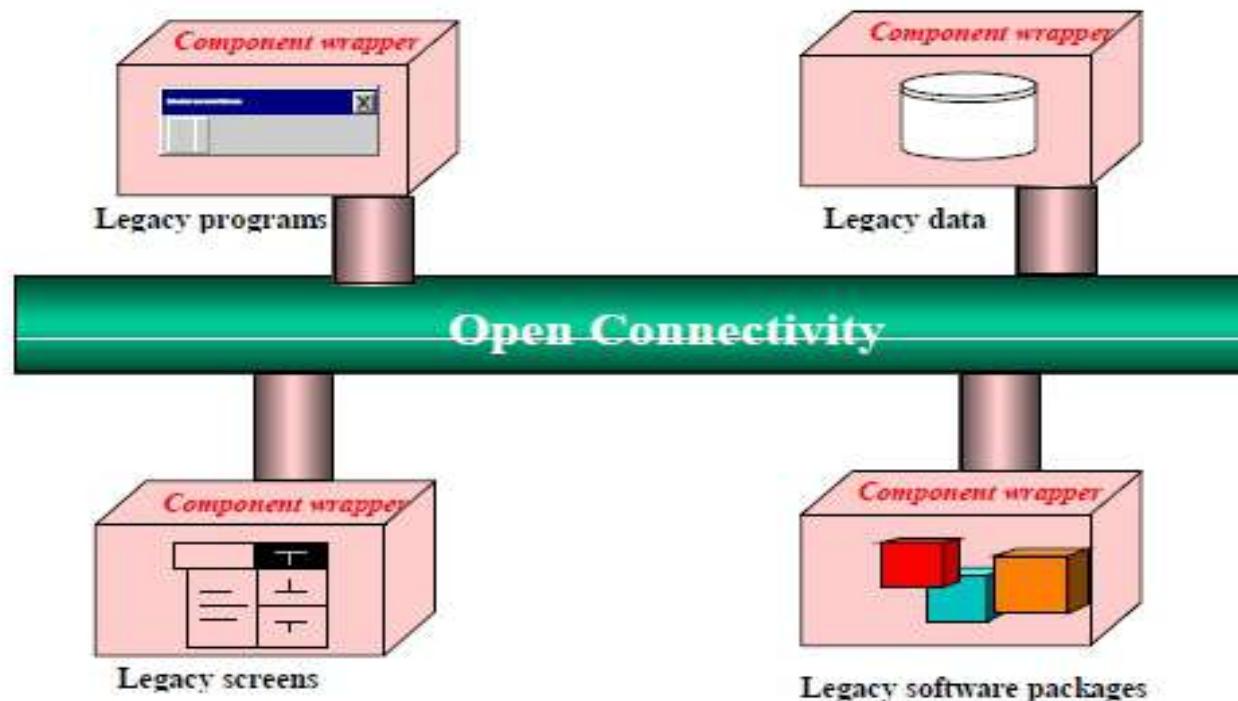


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Component-based development (CBD) Con't)



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Rapid Application Development (RAD)

- RAD is a set of tools and techniques that can be used to build an application faster than typically possible with traditional methods.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Rapid Application Development (RAD) (Con't)

- RAD does not replace SDLC but complements it, since it focuses more on process description and can be combined perfectly with the object-oriented approach.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013

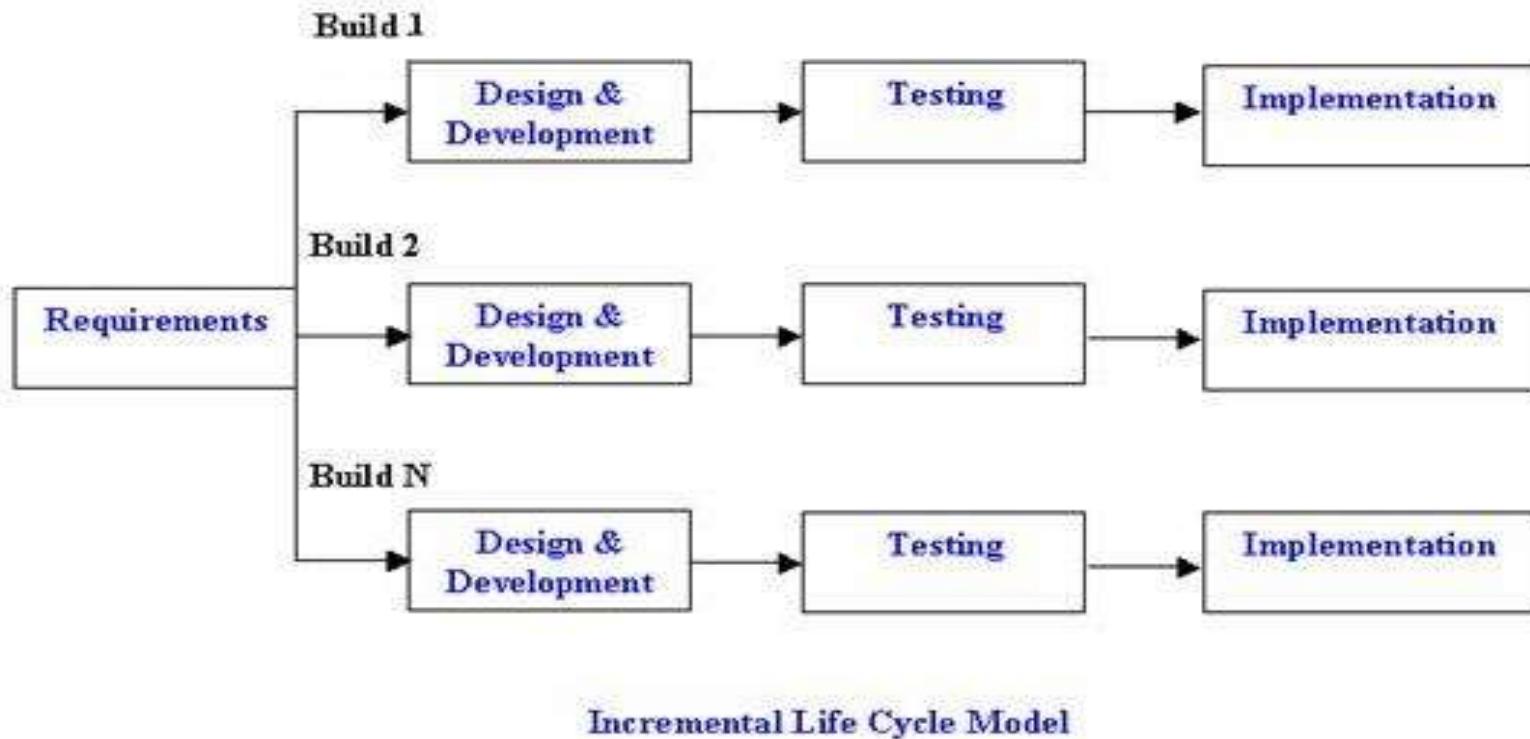


RAD (cont)

- Task of RAD is to build application quickly and incrementally implement the design and user requirements
- Eg :through Delphi,Visualage,Visual Basic and Power Builder
- RAD involves in number of iterations.

Incremental Testing

- Test an application for bugs and performance.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Incremental Integration Testing:

- Integration testing is performed after unit testing.
- It is the process of verifying the interfaces and interaction between modules.
- As the developers integrate modules one by one this type of testing is called as Incremental Integration testing.
- In this testing, the developer integrate the modules one by one using stubs or drivers to uncover the defects.

Reusability

- Difficult promise to deliver on.
- Evaluation for reusability :-
 - Has my problem already been solved?
 - Has my problem been partially solved?
 - What has been done before to solve a problem similar to this one ?



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



References

1. Ali Bahrami, Object Oriented Systems Development, Third Edition, Tata McGraw-Hill, 2012

Object-Oriented Methodologies



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



What is Object Oriented Methodology?

- It is a new **system development approach**, encouraging and facilitating re-use of software components.
- It employs international standard **Unified Modeling Language** (UML) from the **Object Management Group** (OMG).
- Using this methodology, a system can be developed on a component basis, which enables the effective re-use of existing components, it facilitates the sharing of its other system components.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



What is Object Oriented Methodology?

- Object Oriented Methodology asks the analyst to determine **what the objects of the system are?**,
- What **responsibilities and relationships** an object has to do with the other objects? and
- How they **behave over time?**



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Types

- **There are three types of Object Oriented Methodologies**
 1. Object Modeling Techniques (OMT)
 2. Object Process Methodology (OPM)
 3. Rational Unified Process (RUP)



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Method & its Strength

- Rumbaugh Methodology
 - Describe the object model or the static structure of the system.
- Booch Methodology
 - Produces detailed design models
- Jacobson Methodology
 - Produces user-driven analysis models



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- **1. Object Modeling Techniques (OMT)**
- It was one of the **first object oriented methodologies** and was introduced by Rumbaugh in 1991.
- OMT uses **three different models** that are combined in a way that is analogous to the older structured methodologies.
- A method for **analysis, design and implementation** by an object oriented technique.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- **Fast and intuitive approach** for identifying and modeling all objects making up a system.
- **Class attributes, methods, inheritance and association can be expressed easily.**
- Dynamic behavior of objects can be described using the OMT dynamic model.
- Detailed specification of state transitions and their descriptions within a system



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



a. Analysis

- The main goal of the analysis is to **build models of the world**.
- The requirements of the users, developers and managers provide the information needed to develop the **initial problem statement**.



b. OMT Models

I. Object Model

- It depicts the object classes and their relationships **as a class diagram**, which represents the static structure of the system.
- It observes all the objects as static and does not pay any attention to their dynamic nature.

II. Dynamic Model

- It captures the behavior of the system over time and the flow control and events in the **Event-Trace Diagrams and State Transition Diagrams**.
- It portrays the changes occurring in the states of various objects with the events that might occur in the system.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



III. Functional Model

- It describes the **data transformations of the system.**
- It describes the **flow of data and the changes that occur to the data throughout the system.**



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



c. Design

- It specifies **all of the details needed to describe how the system will be implemented.**
- In this phase, the details of the system **analysis and system design are implemented.**
- The objects identified in the system design phase are designed.



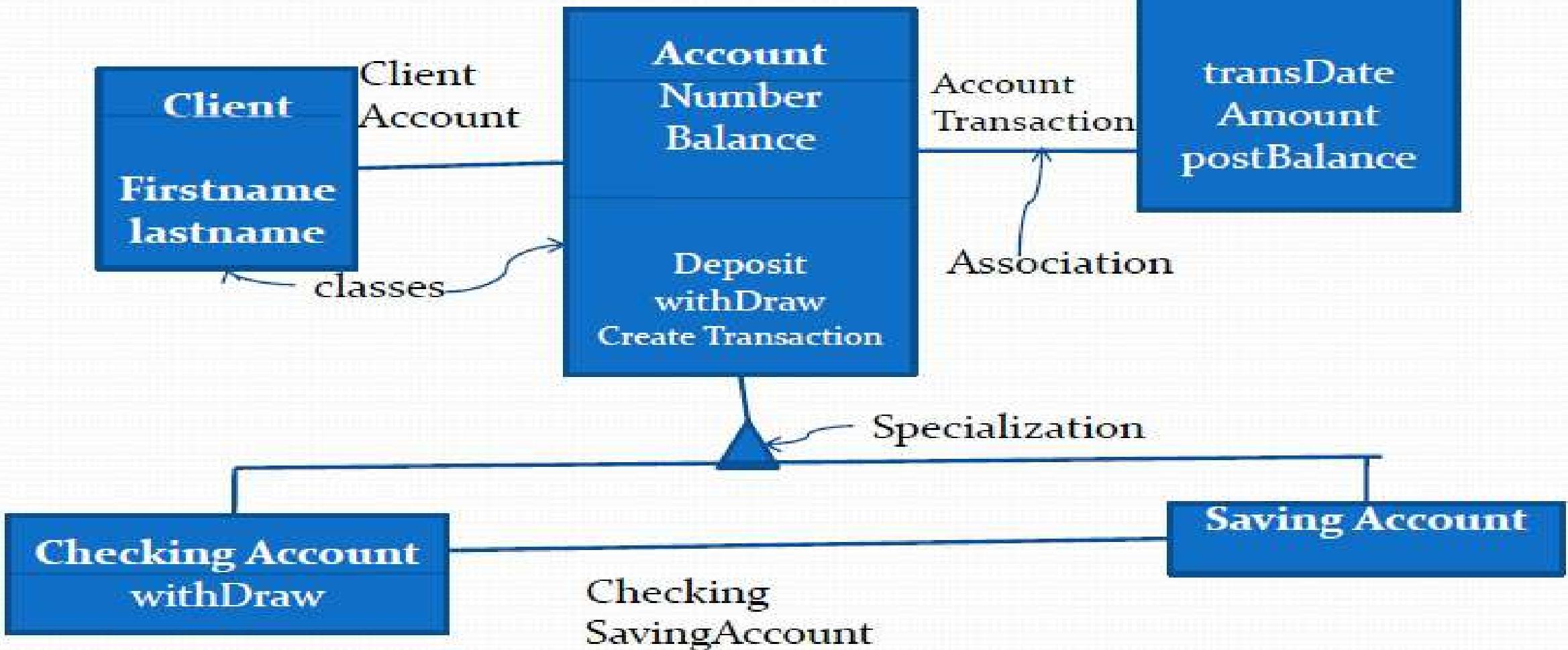
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



RUMBAUGH ETAL'S OBJECT MODELING TECHNIQUE

- An Object model: bank system



The OMT Dynamic Model

- Lets you depict **states, transitions, events and actions.**
- State Transition Diagram **is a network of states and events.**
- States receives **1 or more events** at which they make the **transition to next state.**



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



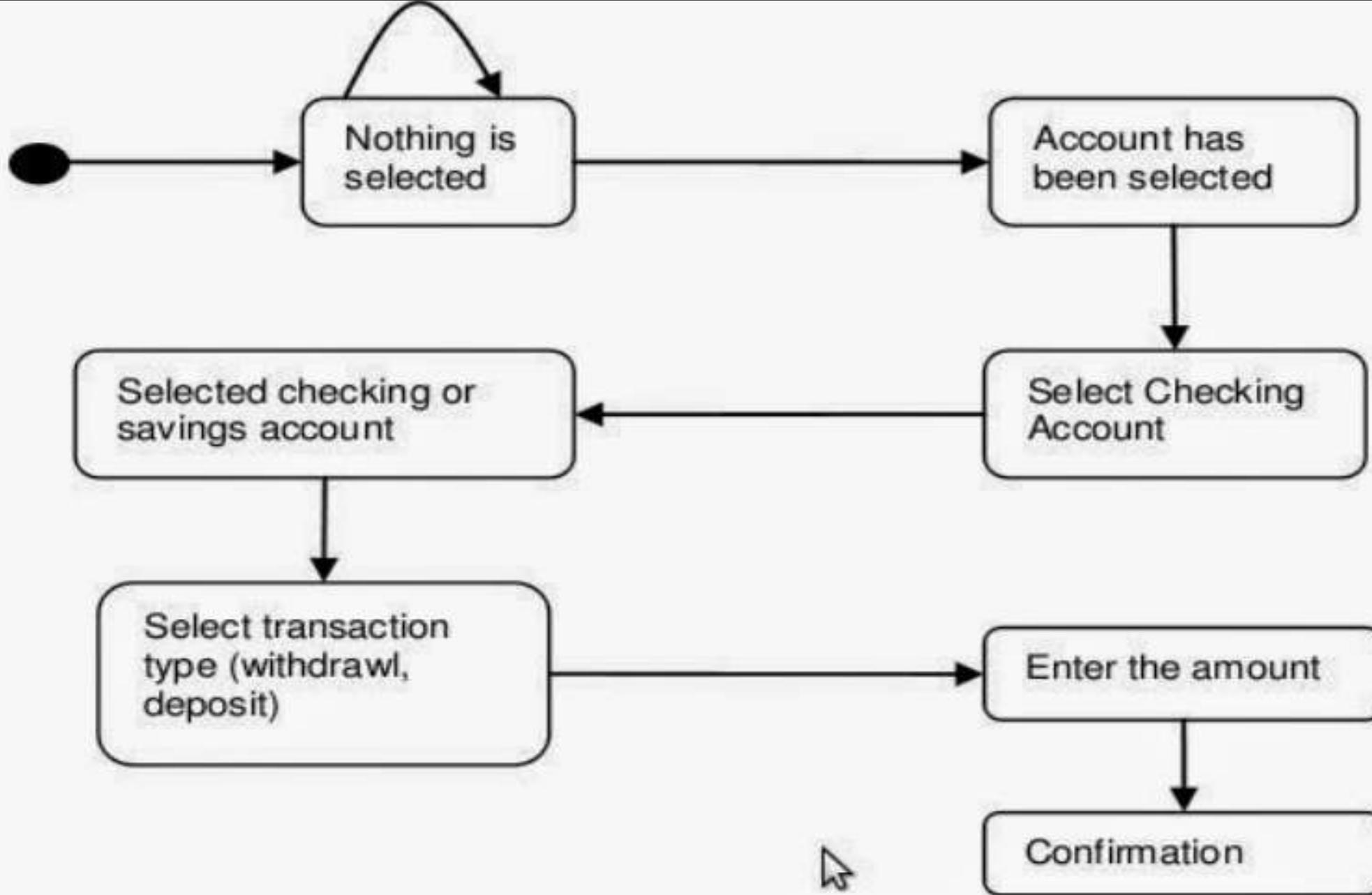


Fig. 4.2: State transition diagram for the bank application user interface. The round boxes represent states and the arrows represent transitions.

The OMT Functional Model

- Data Flow Diagram **shows the flow of data between different processes** in a business.
- OMT DFD provides simple and intuitive approach for describing business processes.
- **Four Primary Symbols used in DFD are :-**
 1. **Process** → Any function being performed
 2. **Dataflow** → Shows the direction of data element
 3. **Data Store** → Location where data is stored
 4. **External Entity** → source/destination of data element



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



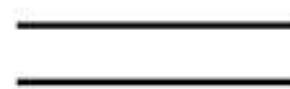
Data Flow Diagram

- Four primary symbols



Process- any function being performed

Data Flow- Direction of data element movement



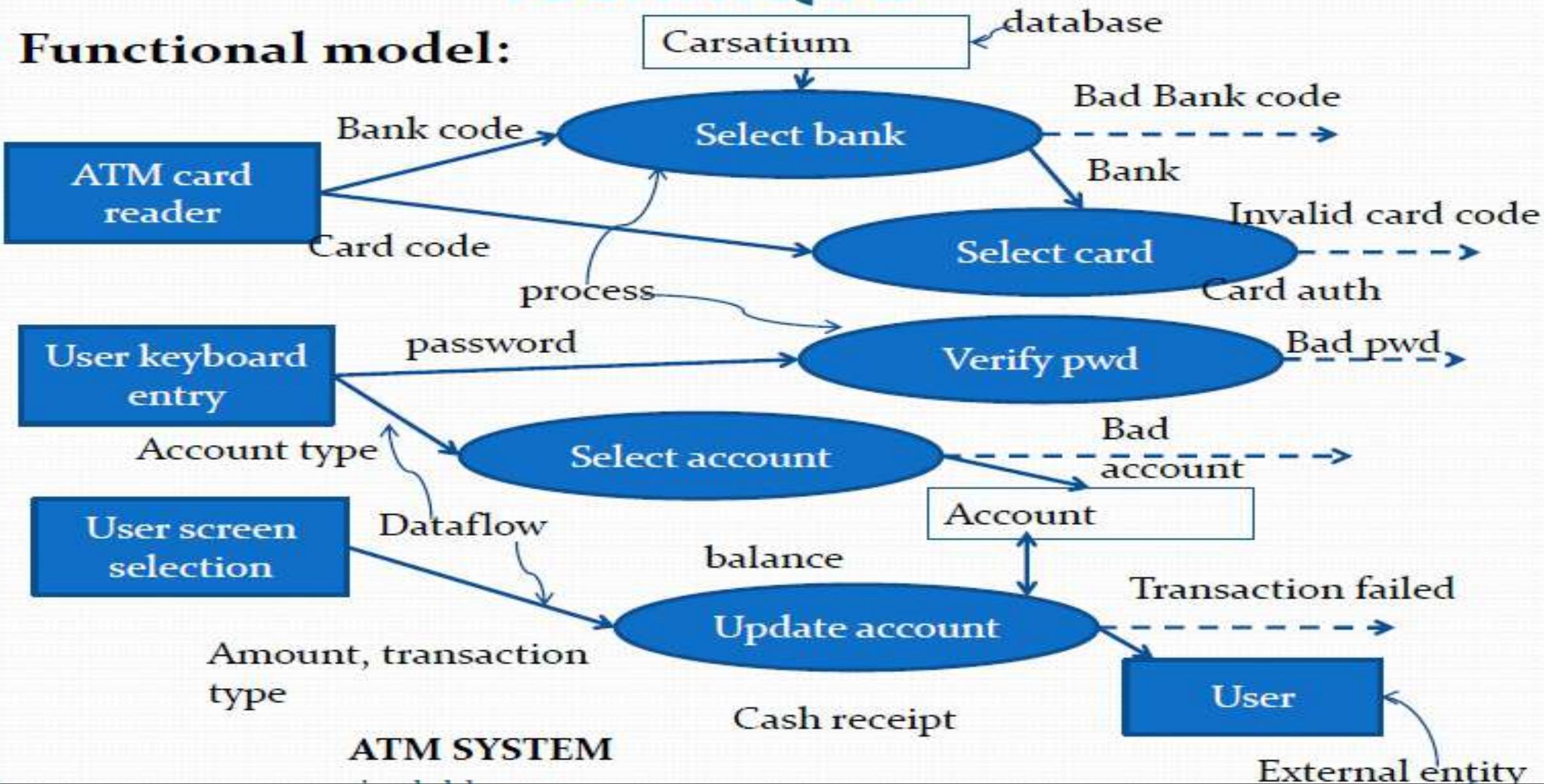
Data Store – Location where data is stored



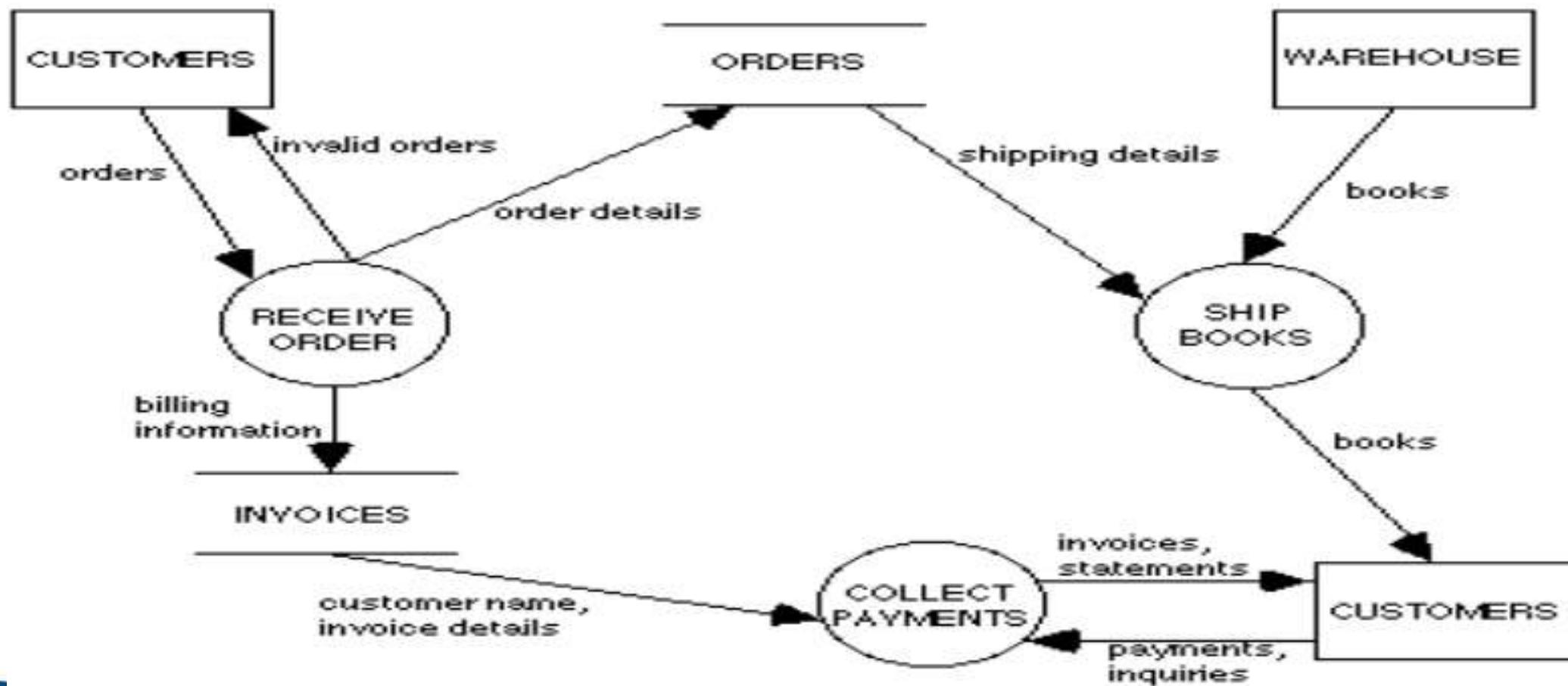
External Entity-Source or Destination
of a data element

RUMBAUGH ETAL'S OBJECT MODELING TECHNIQUE

- Functional model:



Example for Data Flow Diagram



- Case study: Apply Rumbaugh Model for Online ticket reservation System

Booch Methodology



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- The Booch method is one well-known OO-method
- Design the systems using the **object paradigm**
- Covers the **analysis and design phases** of an OO-system implementation
- Booch defines a **lot of symbols** to document almost every design decision



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- This method starts with **class and object diagrams** (a discovering activity) in the analysis phase
- These diagrams to be **refines** through various steps
- Refinement process continuous till the problem domain gets more and more understood following an ***evolutionary*** approach



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- Design symbols are to be added when ready to generate code
- Usually it represents very final implementation decisions
- Booch notation are larger sets and appears to prove beneficial: it is possible to fully document the OO-code

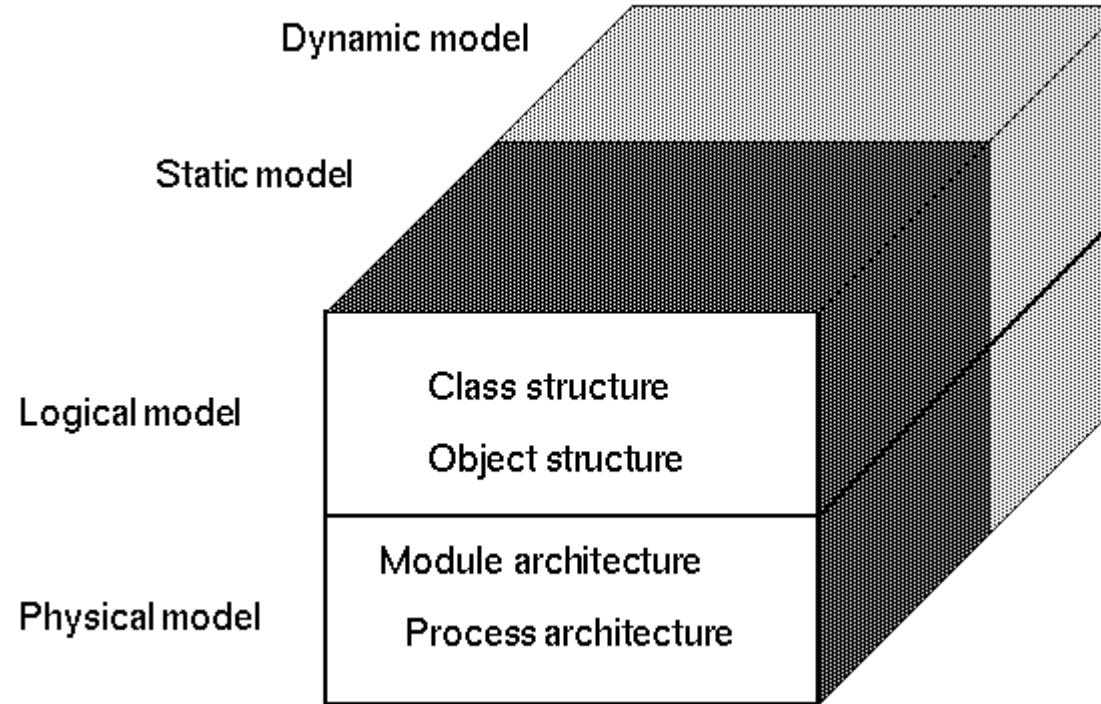


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Overview



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- Booch proposes different views to describe an OO system
 1. Physical model
 2. Logical model
 3. Static model
 4. Dynamic model



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



1. Physical model

- It describes the **concrete hardware with respect to the software components of a system.**
 - ✓ Module and process architecture
 - ✓ Processors
 - ✓ Devices and communication connections between them



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



2.Logical model (i.e., the problem domain) :

- Represented in the **class and object structure**
- In the class diagram one builds **up the architecture**, or the **static model**
- To deal with complex diagrams
 - ✓ the notation allows ***class categories*** to group classes into namespaces, each category being itself a class diagram.

3 Static Model:

- ✓ class diagrams (and the relationships therein) are mostly static

4 Dynamic Model :

- ✓ Object diagrams (and the relationships therein) describe the dynamic behavior of the system
- ✓ In this instance relationship means *message exchanges* between objects



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



It consists of the following Six diagrams:

- Class diagrams
- Object diagrams
- State transition diagrams
- Module diagrams
- Process diagrams
- Interaction diagrams

Class Diagrams

- Class:
 - The modular unit of OO software decomposition
 - As a type, **a class represents a set of similar run-time data elements, or *objects***
 - Share a common structure and a common behavior
 - As a decomposition unit
 - a class is a group of **related services packed together under a single representative name**



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Notations for Class

```
class name
{
    attributes
    operations()
    [constraints]
}
```



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- class Class Name {
public:
 // constructors
 // destructors
 // operators
 // modifiers
 // selectors
protected:
 // member objects
private:
 // friends
};

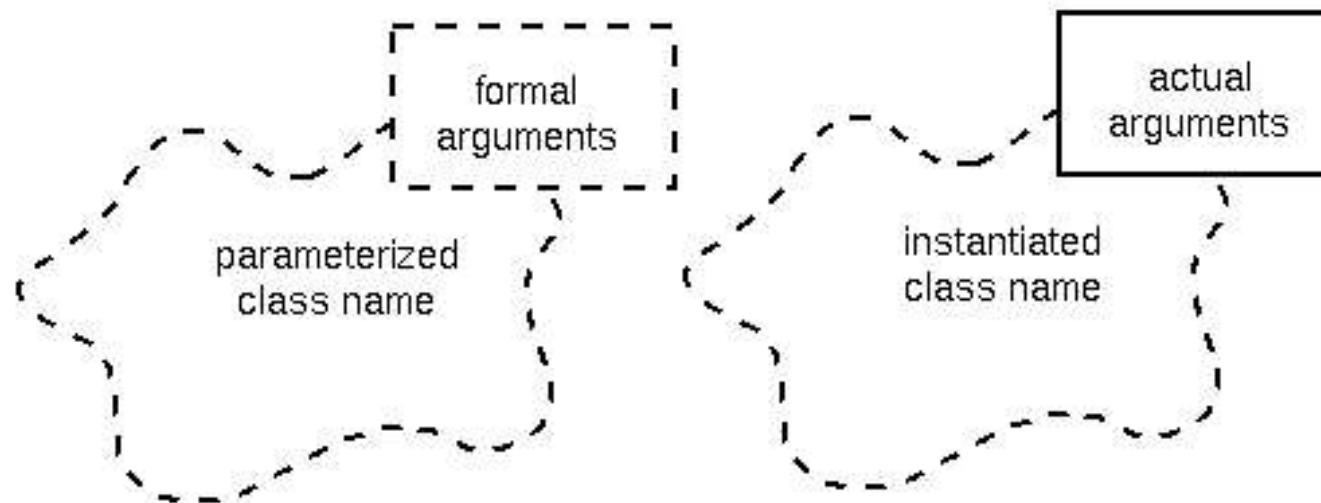


PRESIDENCY
UNIVERSITY

Private University Estd. in Karnataka State by Act No. 41 of 2013



Parameterized class:



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- Public class Box<T>{

```
    private T t;}
```

Box – generic class

T – generic type parameter passed to generic class

.t- instance of generic type T

Class category:

class category name

- A logical collection of classes
- Some of which are visible to other class categories, and others are hidden
- The classes in a class category collaborate to provide a set of services.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- Class categories allow us to represent an important architectural element of the system to be implemented:
 - Clusters of classes that are cohesive
 - But loosely coupled relative to other class aggregates
 - Categories are a way to partition the logical model of a system
 - They represent an encapsulated name space

Class adornment and Properties (for relationships)

 abstract class

 friend

 static

 virtual



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Export control (for attributes/operations and relationships):

public

private

protected

implementation



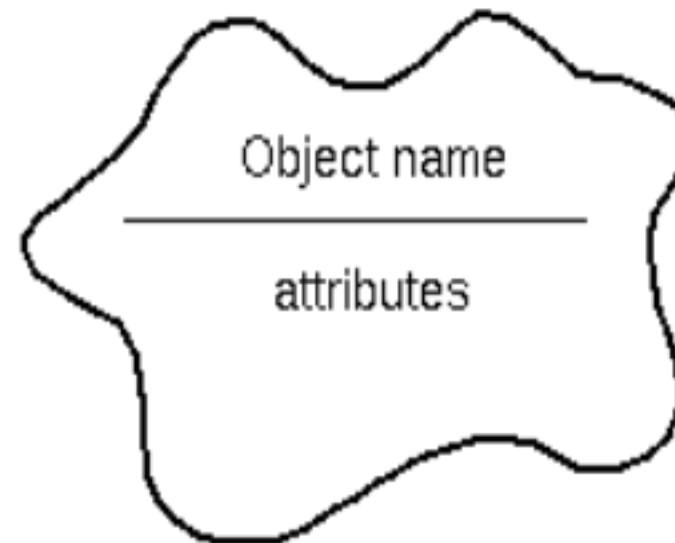
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Object Diagram

- Object
- Object link
- Synchronization
- Visibility



- **State Transition Diagram**
 - Shows the dynamic behavior of classes
- **Module Diagram**
 - Module diagrams are used to show the allocation of classes and objects to modules in the physical design of a system
- **Process Diagram**
 - A process diagram is used to show the allocation of processes to processors in a physical design of a system
- **Interaction Diagram**
 - Interaction diagrams trace the execution of a scenario in the same context as an object diagram



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Two processes:

- BOOCH Methodology prescribes 2 different process
 1. Macro Development Process
 2. Micro Development Process



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



1. Macro development process

- Primary concern – **technical management of the system**
- Steps involved:
 - **Conceptualization.**
 - Establish the core requirements and develop a prototype
 - **Analysis and development of the model**
 - Use the class diagram to describe the roles
 - Responsibilities of objects
 - Use the object diagram to describe the desired behavior of the system



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



➤ Design or create the system architecture

- Use the class diagram to decide what classes exist and how they relate to each other
- The object diagram to decide what mechanisms are used
- The module diagram to map out where each class and object should be declared
- The process diagram to determine to which processor to allocate a process.

➤ Evolution or implementation

- Refine the system through much iteration

➤ Maintenance

- Make localized changes to the system to add new requirements and eliminate bugs



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



2. Micro development process

- The micro development process is a description of the day-to-day activities.
- Steps involved:
 - Identify classes and objects
 - Identify classes and object semantics
 - Identify classes and object relationships
 - Identify classes and object interfaces and implementation



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



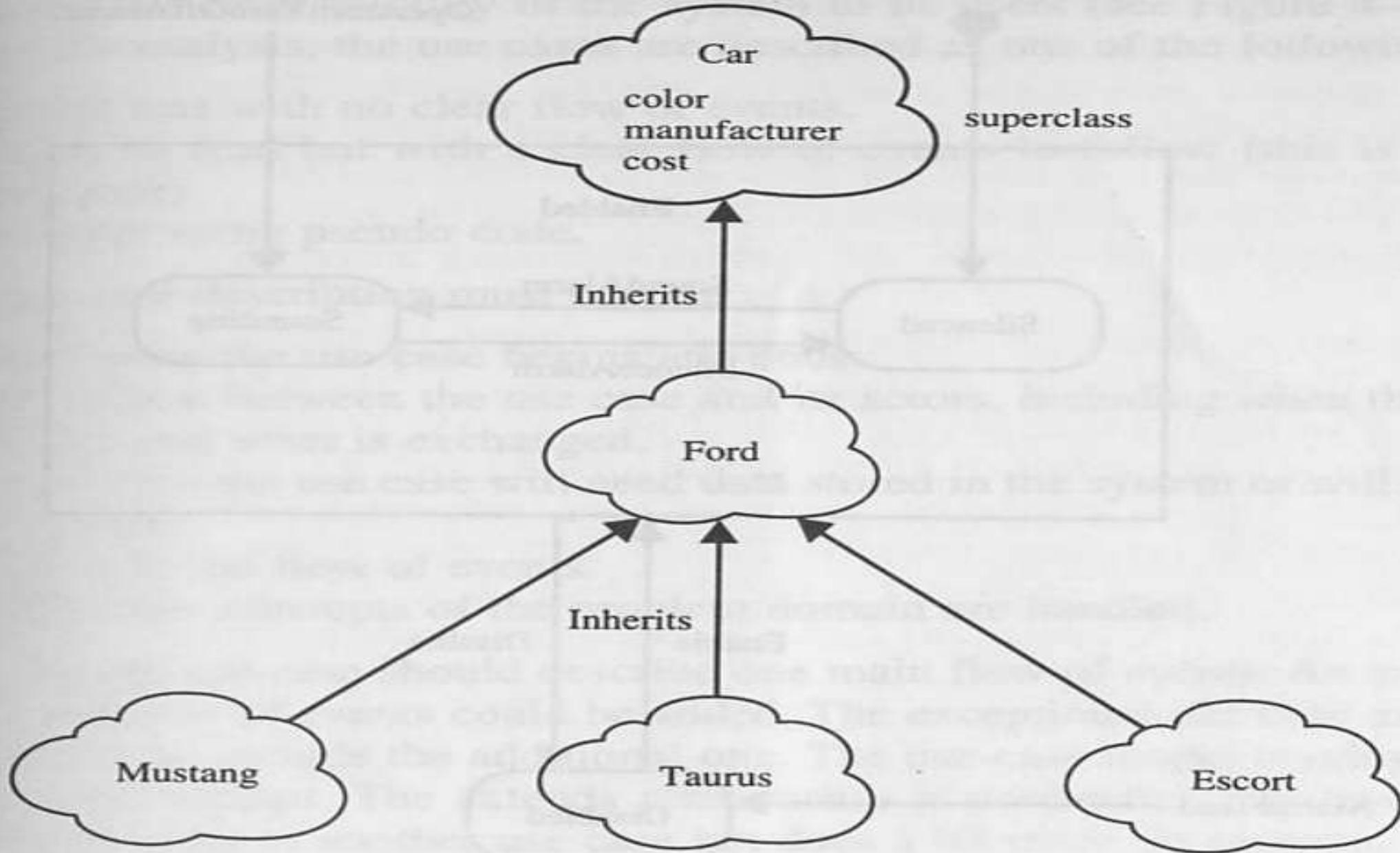
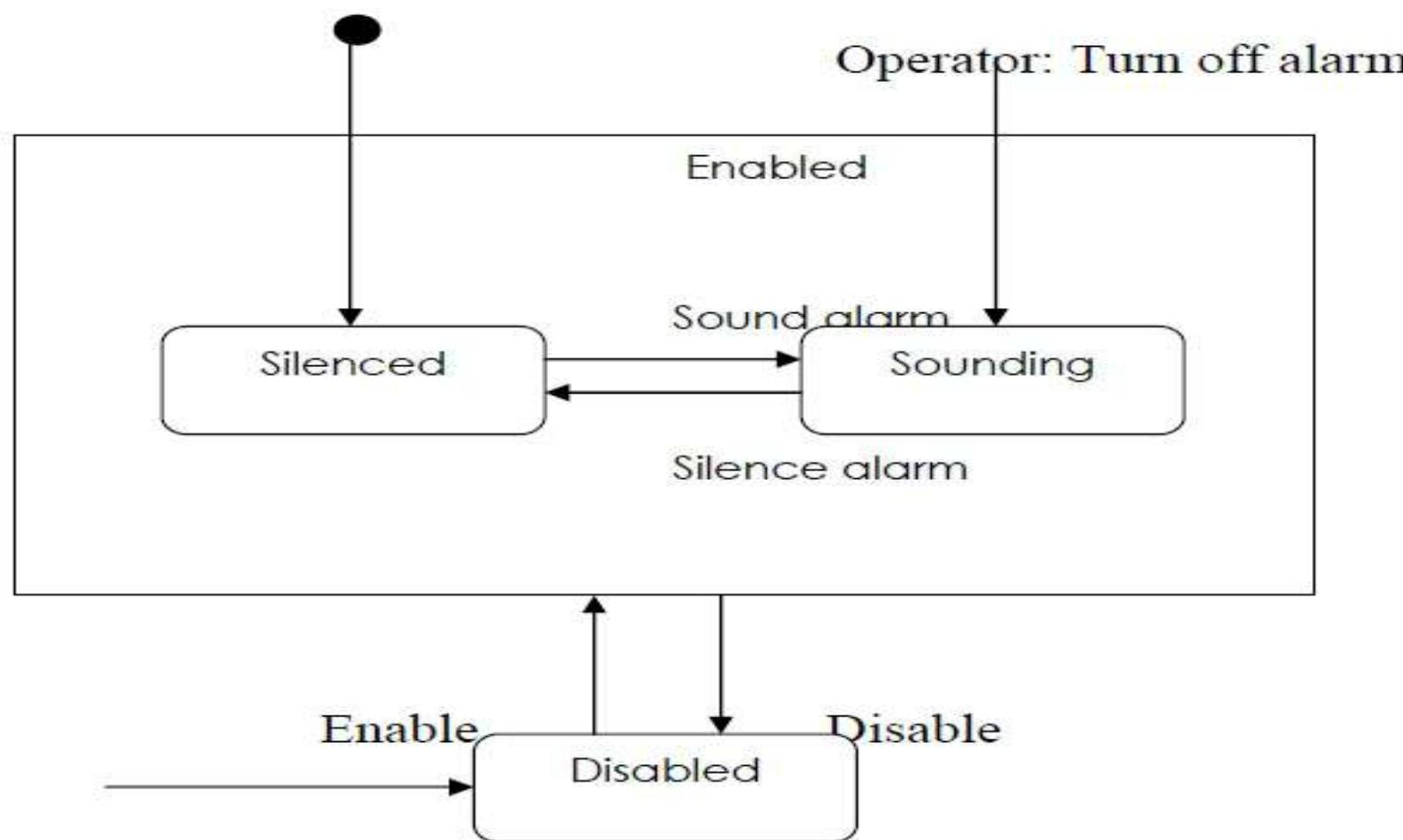


FIGURE 4-4

Object modeling using Booch notation. The arrows represent specialization; for example, the class Taurus is subclass of the class Ford.

An alarm state transition diagram with Booch



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Reference:

- Ali Behrami, “Object Oriented Systems Development using Unified Modeling Language”, McGraw Hill International Edition.
- Grady Booch, “*Object Oriented Analysis and Design with Applications*”, Addison-Wesley.
- https://www.slac.stanford.edu/BFROOT/www/doc/workbook_kiwi/coding/booch/method.html



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Jacobson Methodology



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Jacobson Methodology

- OOSE is developed by **Ivar Jacobson** in 1992. OOSE is the first object-oriented design methodology that employs use cases in software design.
- It covers **entire life cycle and stress traceability** between different phases
- Advantage :-
 - Reduction of development time
 - Reuse of code
 - Reuse of analysis and design work

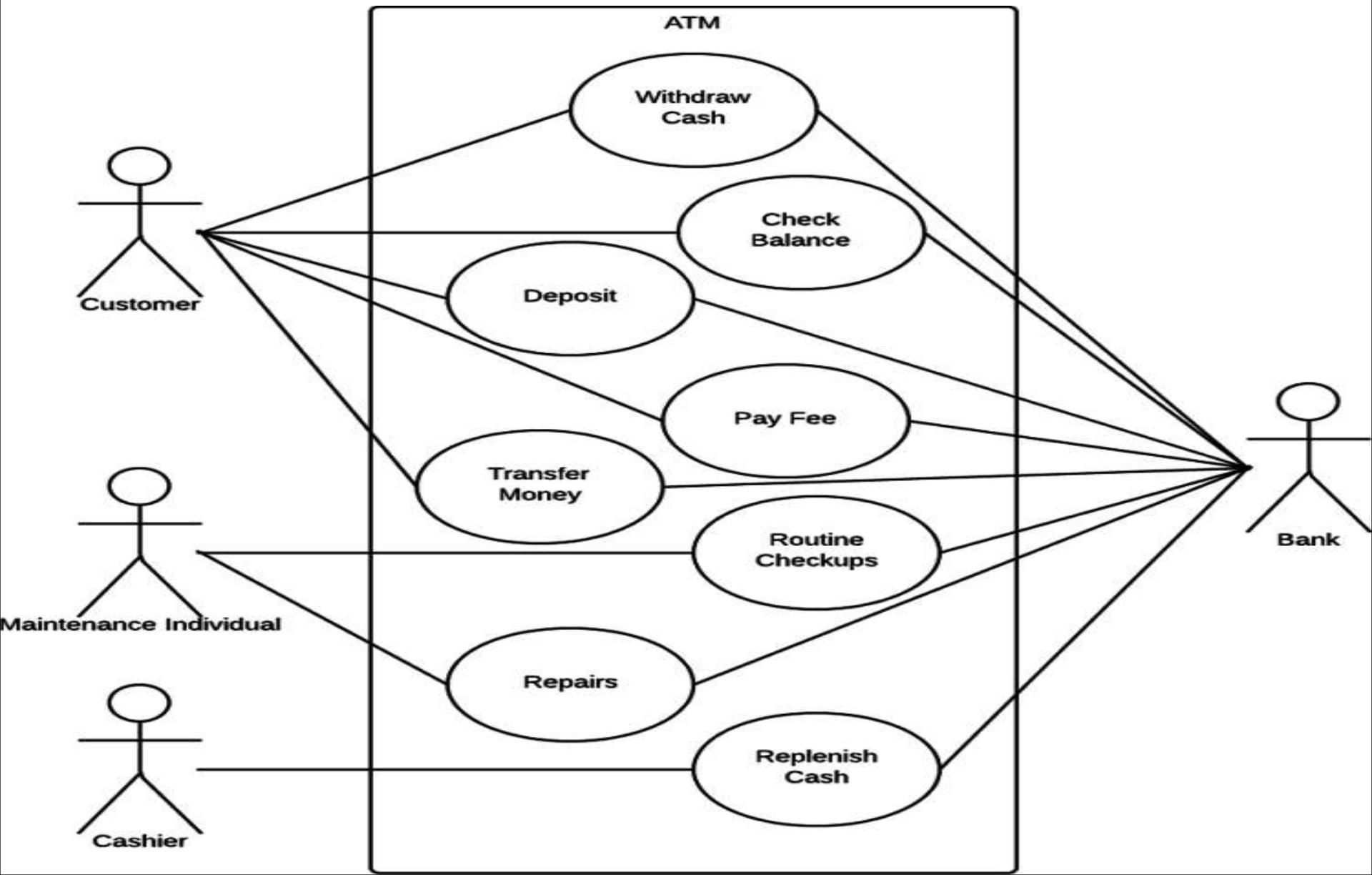
Use-Case

- Use-Case is **an interaction between user and the system.**
- Scenarios for understanding system requirements.
- Use-case model **captures the goal of the user and the responsibility of the system to its users.**
- Use Cases are described as :-
 - Nonformal text with no clear flow of events.
 - Text, easy to read but with a clear flow of events to follow
 - Formal style using Pseudo code.

Use- Case Description must contain the following :-

- *How and when* the use case begins and ends
- The interaction between the use case and its actors, including *when* the interaction occurs and *what* is exchanged.
- *How and when* the usecase will need data stored in the system or will store data in the system.
- *Exceptions* to the flow of events
- *How and when* concepts of the problem domain are handled.

Sample Use-Case Diagram



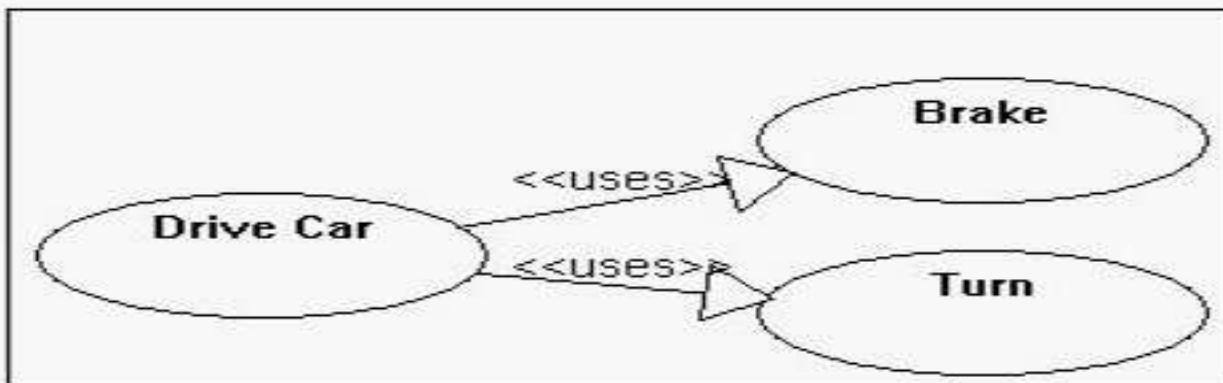
Extends and Uses Relationship

- **Extends** – used when you have one use case similar to another use case but does a bit more (Extends the functionality of original use case---sub class).
- **Uses** – Reuses the common behavior in different use case.

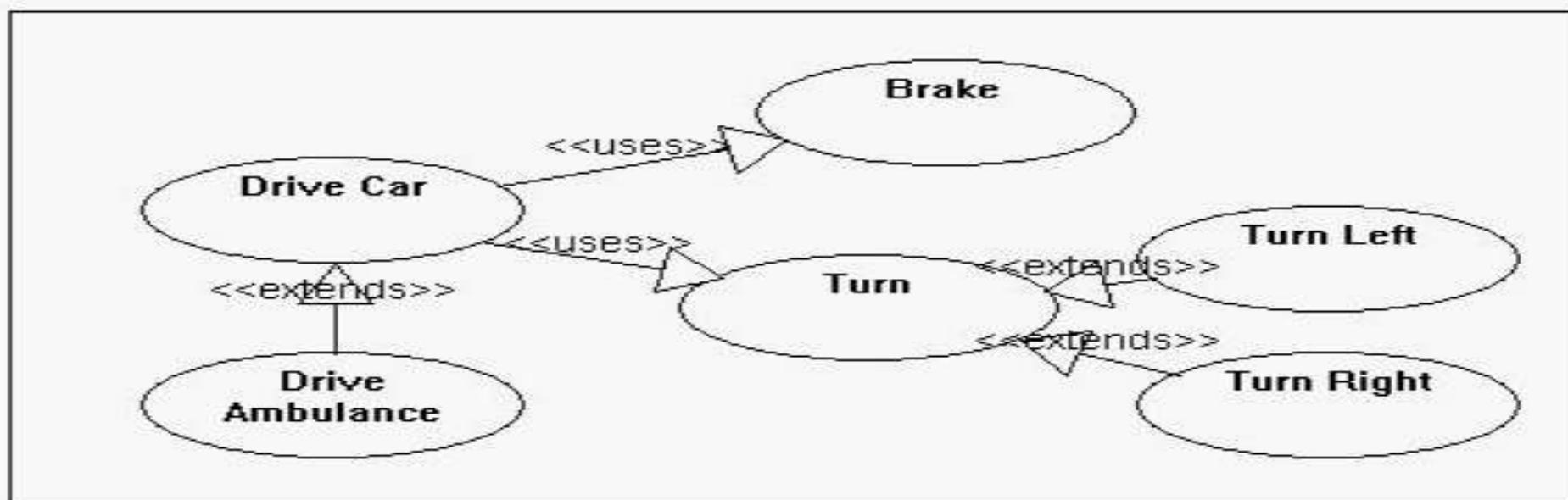
Evolution of a UML Use Case Diagram



... Becomes ...

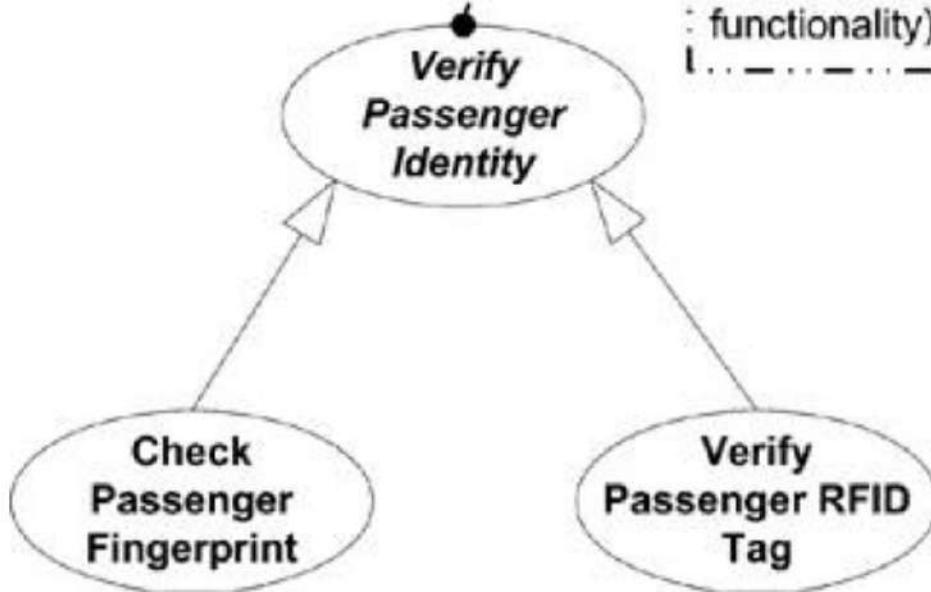


... Which Becomes ...



Use-Case Viewed as,

- Use Case Viewed as **Concrete or Abstract**.
- **Abstract Use Case is:-**
 - Not complete
 - Has no actors to initiate
 - But used by other use-case
 - It uses extends or uses relationship



The general use case is abstract (it doesn't describe a particular functionality), so the title is in italics.

Object Oriented Software Engineering

- Also called **Objectory**.
- Objectory is built around several different models :-
 1. **Use case model** → Defines the inside & outside of the system
 2. **Domain object model** → Objects of “real world” are mapped into domain object
 3. **Analysis object model** → how source code should be carried out & written
 4. **Implementation model** → represent the implementation of the system
 5. **Test Model** → constitute test plan, specification & report

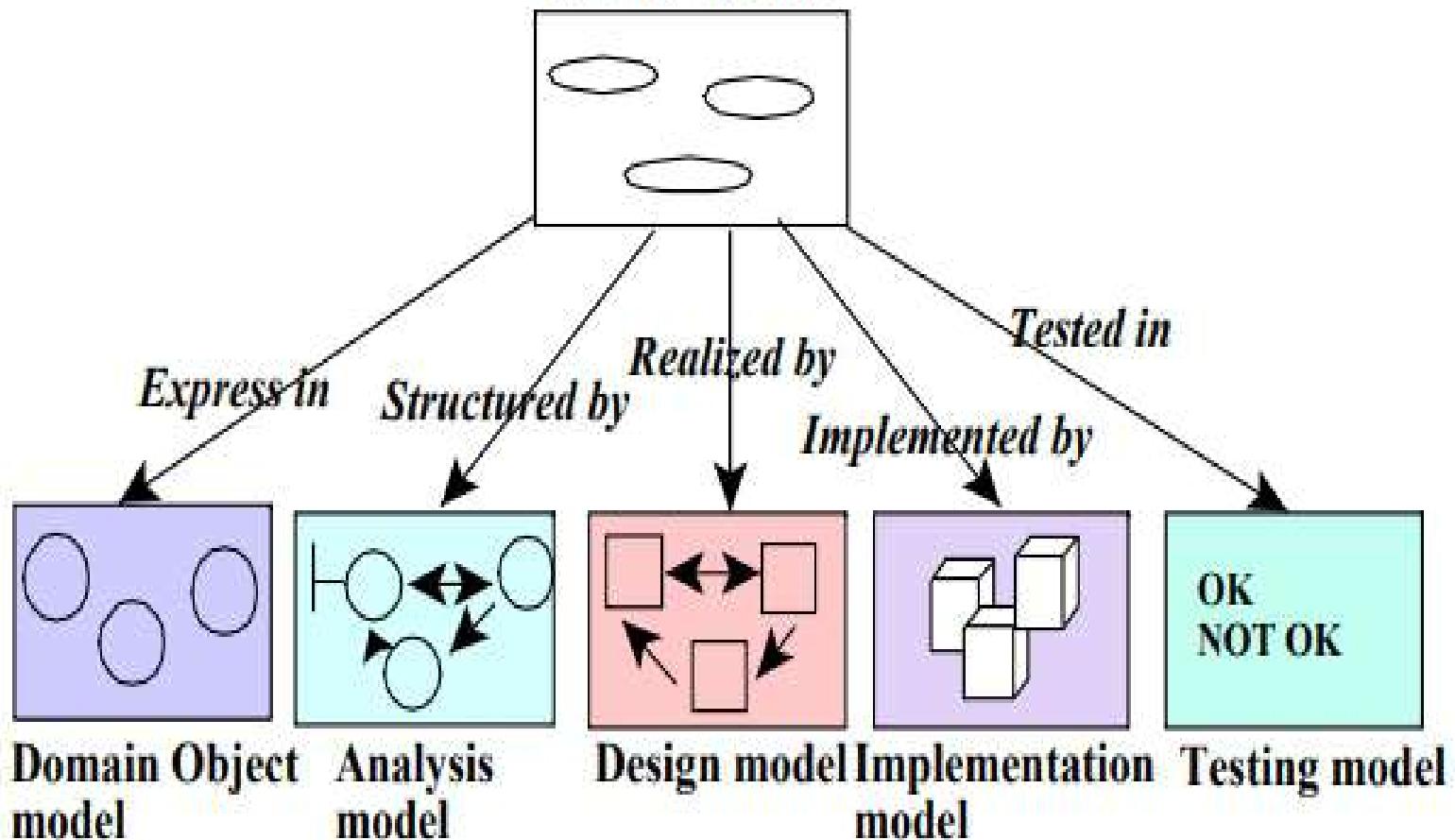


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Use-case model



Object Oriented Business Engineering

- Object **Modeling at the enterprise level**
- Use Case serves as **central vehicle for modeling**, providing traceability throughout the software engineering process.
- Software Engineering Process includes the following phases :-
 - *Analysis phase*
 - *Design and Implementation phase*
 - *Testing phase*

Assessment Questions

- What are the object oriented methodologies available ?
- Which is the widely used OO Methodology ?
- Object-oriented Software Engineering is also known as _____.
- Requirement should be finalized before which phase in development ?
- Use case can be viewed as _____.

Assessment Questions & Answers

1. What are the object oriented methodologies available ?
Rumbaugh, Booch, Jacobson
2. Which is the widely used OO Methodology ? Booch
3. Object- oriented Software Engineering is also known as
Objectory
4. Requirement should be finalized before which phase in development ? Design Phase
5. Use case can be viewed as Concrete or Abstract.

1980s

C pointers

Emacs

Math library

Ad hoc programming

Waterfall

Flowcharts

Write your own sort

Computer room

Hard disk

Text terminals

Email

No regulation

2010s

Java garbage collection

Eclipse

Frameworks

Agile methodology

Evolution/continuous integration

UML

Copy from Stack Overflow

Computer in your pocket

Cloud

Touch screens

Internet of Things

No regulation



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



The Unified Approach



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



The Unified Approach

- Unified Approach “establishes a **unifying framework** by utilizing the **UML diagram** to describe, model and document the software development process”.
- Idea :-
 - To stop evolution of another methodology.



**PRESIDENCY
UNIVERSITY**

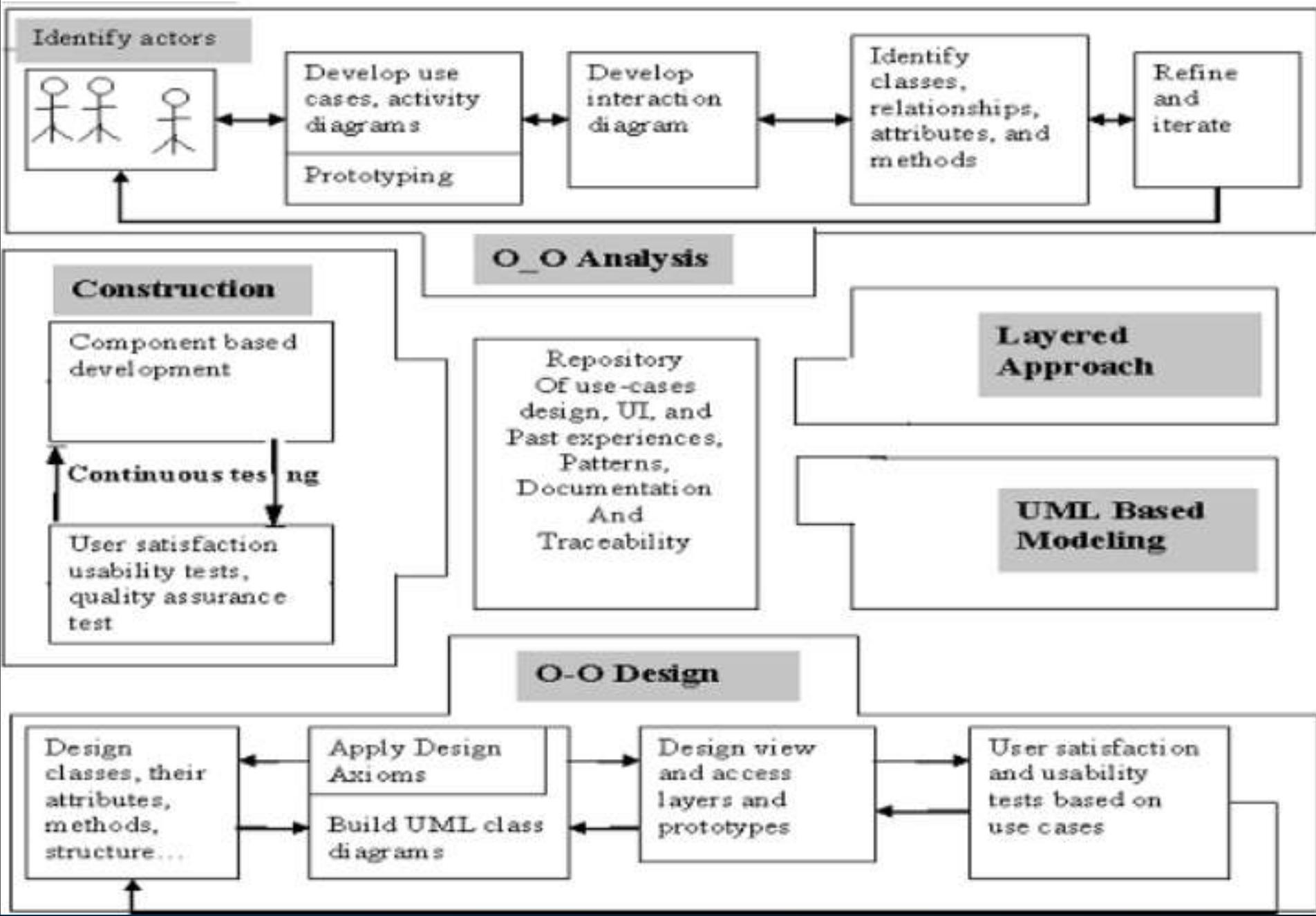
Private University Estd. in Karnataka State by Act No. 41 of 2013



- **Motivation :-**

- Combine best practices, methodologies, processes and guidelines along with UML diagrams for better understanding of OOPS concept & System development

Fig : Process and Components of Unified Approach



Processes involved in Unified Approach

- Use-case driven development
- OO Analysis
- OO design
- Incremental Development and Prototyping
- Continuous Testing

Methods and Technologies include

- Unified Modeling Language for Modeling
- Layered Approach
- Repository OO System Development
- Component-based development

It deviates from waterfall model since backtracking done between analysis and design phase [Iterative Development].

Object-Oriented Analysis

- Analysis is the **process of extracting the needs of a system and what the system must do to satisfy the users' requirements.**
- The goal of object-oriented analysis is to first **understand the domain of the problem and the system's responsibilities by understanding how the users use or will use the system.**
- It concentrates on describing *what the system does rather than how it does it.*
- View the system from the *user's perspective* rather than that of the machine.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Contd,...

OOA process consists of the following steps:

1. Identify the Actors.
2. Develop a simple business process model using UML Activity diagram.
3. Develop the Use Case.
4. Develop interaction diagrams.
5. Identify classes.

Object-Oriented Design

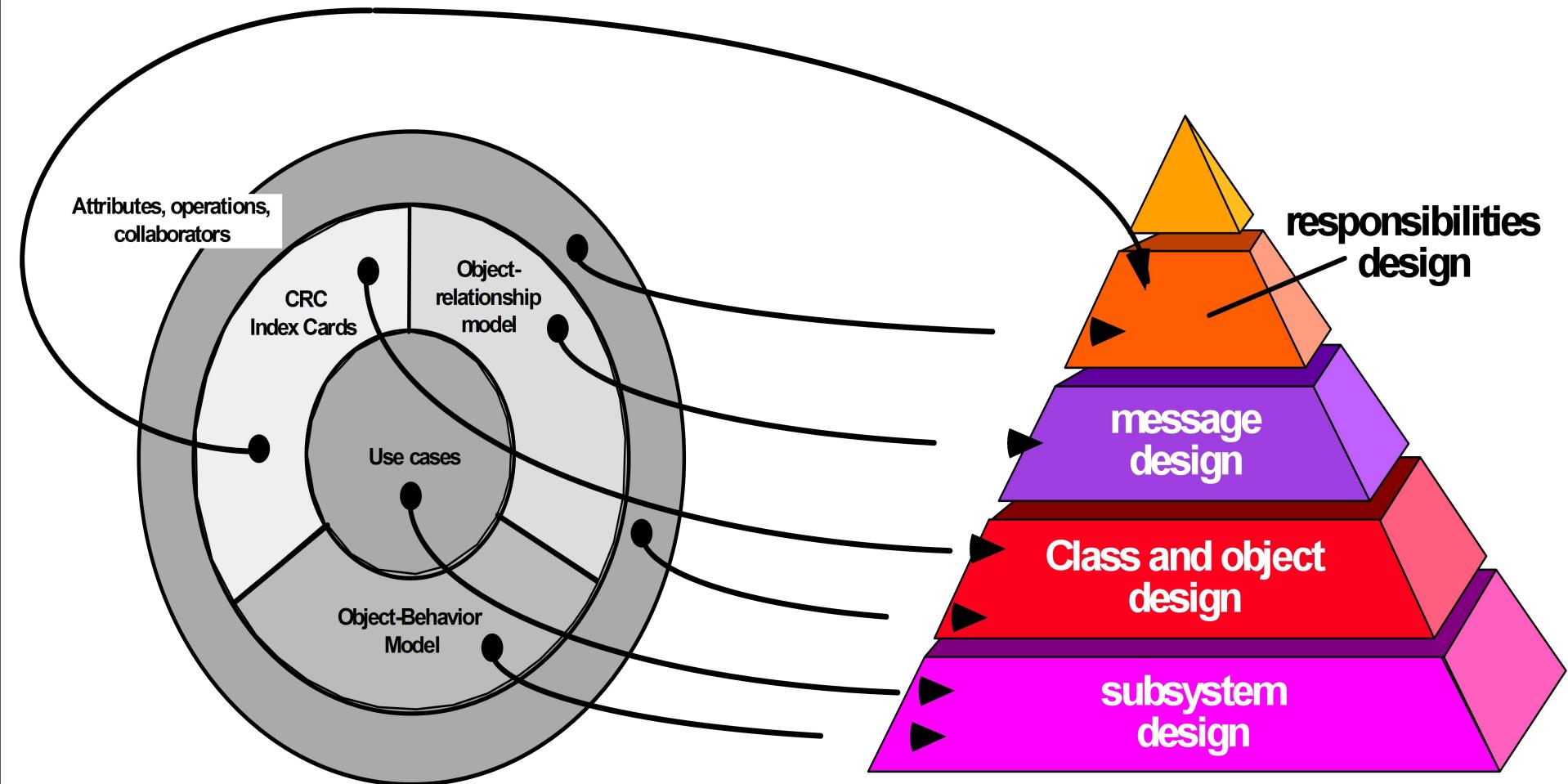
- Booch, provides the **most comprehensive** object-oriented design method.
- Rumbaugh et al.'s and Jacobson et al.'s high-level models provide **good avenues for getting started**.
- UA combines these by utilizing **Jacobson et al.'s *analysis and interaction diagrams***, Booch's ***object diagrams***, and Rumbaugh et al.'s ***domain models***.
- Furthermore, by following Jacobson et al.'s life cycle model, we can produce designs that are **traceable** across requirements, analysis, design, coding, and testing

Contd,...

OOD Process consists of:

1. Designing classes, their attributes, methods, associations, structures and protocols, apply design axioms.
2. Design the Access Layer
3. Design and prototype User interface
4. User Satisfaction and Usability Tests based on the Usage/Use Cases
5. Iterated and refine the design

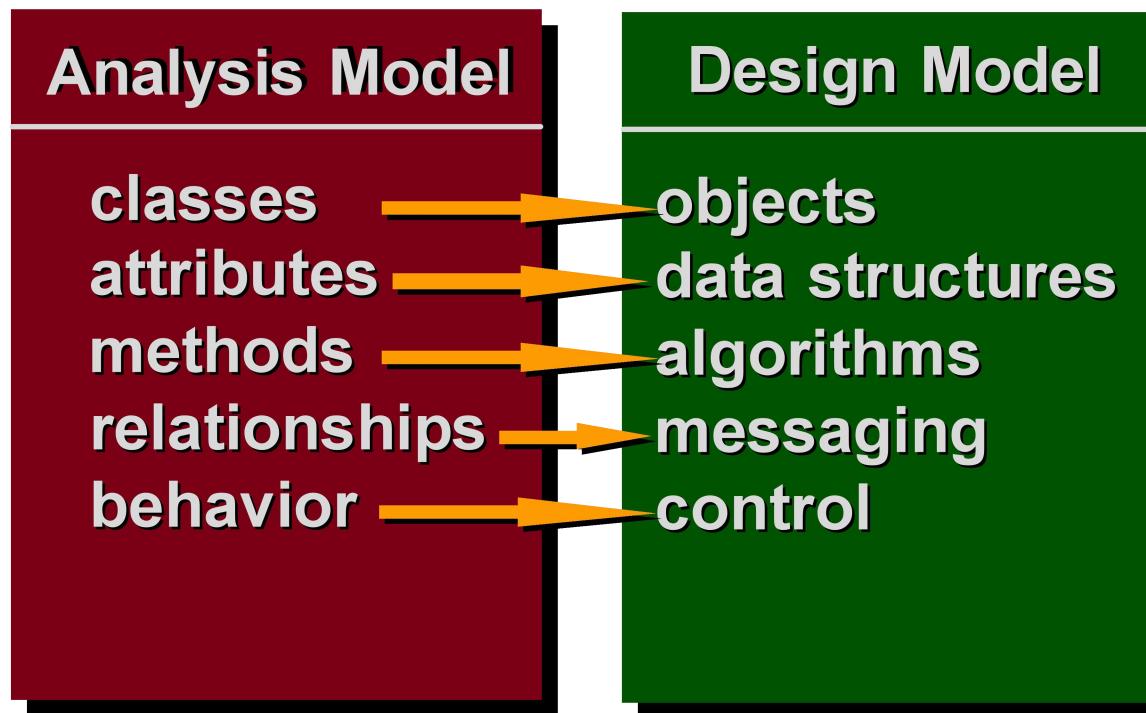
OOA to OOD



THE ANALYSIS MODEL

THE DESIGN MODEL

OOA to OOD



Iterative Development And Continuous Testing

- You must **iterate and reiterate until, eventually, you are satisfied with the system.**
- *“Testing uncovers design weaknesses”*
- During this iterative process, your **prototypes will be incrementally transformed into the actual application.**
- The UA encourages the integration of testing plans from **day 1 of the project.**

Modeling Based on UML

- The UA uses **UML to describe and model the analysis and design phases** of system development.
- The Unified Modeling Language (**UML**) is, as its name implies, a **modeling language and not a method or process**.
- UML is made up of a very specific **notation** and the related grammatical **rules** for constructing software models.

The UA Proposed Repository

- Sharing eliminates duplication of Problem Solving.
- Reusability.
- Assembling Components from the Library.
- Reduce the cost & development time.

The Layered Approach to Software Development



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Two-Layered Architecture

- **LAYER 1** -- User Interface [User Interface + Business Logic]
- **LAYER 2** -- Database

2-Tier Architecture

Client Computers

Client Tier



Database Tier



Database Server

Why Three-Layered Approach ?

- **Business Logic Resides** in the screen.
- **Routines to access database** too resides in the screen.
- Here, Objects are specialized and are not reusable.

[Objects are tangible elements of your business]

- Hence, Isolate the function of the interface from the function of the business – called *three-layered approach.*

Three-Layered Approach

OOD & Prototyping

Access Layer

OOA

Business Layer

OOA, OOD & Prototyping

View Layer



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



The User Interface (View) Layer

- **Consists of,**
 - Objects with which the user interacts
 - Objects needed to manage or control the interface
- **Responsible for:-**
 - Responding to user interaction → translate actions of user into business objects
 - Displaying business objects → paint the best possible picture of business object to user
- **View Layer Objects Created in OOA & OOD.**

Business Layer

- **Responsibility :-**
 1. Model the objects of the business &
 2. How the objects interact to accomplish the business processes.
- **Not Responsible for :-**
 1. Displaying details
 - a) No Knowledge of “how data will be displayed”. [Work of View Layer]
 2. Data Access Details
 - a) No Knowledge of “where the data come from”.
- **Business Model** Captures Static and Dynamic Relationships among a collection of business objects.

Access Layer

- Access layer objects that know **How to Communicate with the place where data actually resides.**
- **Responsibilities:-**
 - **Translate Request** → translate any data related requests from the business layer into appropriate protocol for data access
 - **Translate Response** → translate the data retrieved back into the appropriate business objects and pass onto business layer.
- **Access Layer Objects are identified in OOD Phase.**

END OF MODULE

Thank You !!!



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Module-2

OBJECT ORIENTED ANALYSIS



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- **MODULE-2**
- **COURSE OUTCOMES:**
- **Classify the various object oriented analysis techniques**



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



•**MODULE II**

OBJECT ORIENTED ANALYSIS

- Identifying use cases-Object Analysis-Classification:
Theory-Approaches for Identifying Classes: Noun Phrase approach, Common Class pattern approach, Use case driven approach, Classes, Responsibilities and Collaborators-Identifying Object relationships: Associations, Super-sub class relationships, Aggregation.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Object Oriented Analysis Process: Identifying Use Cases



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Objectives

At the end of this chapter, students should be able to

1. Define and understand the object-oriented analysis process
2. Explain the use case modelling process
3. Identify actors
4. Identify use cases
5. Developing Effective Documentation

Introduction

- Analysis is the **process of extracting the needs of a system and what the system must do to satisfy the users' requirements**
- Analysis focus on understanding the problem and its domain
- Analysis involves a great deal of interaction with the people who will be affected by the system.
- By constructing models of the system that concentrate on describing **WHAT the system does, rather than HOW it does**

Introduction

- **Analysis** = “process of transforming a problem definition from a fuzzy set of facts and myths (into a) → coherent statement of system’s requirements”
- **Objective:-**
 1. To capture a complete, unambiguous and consistent picture of the requirements of the system &
 2. What the system must do to satisfy the user’s requirements.

Tools to extract information about a system

- 1) Examination of existing system documentation
- 2) Interviews
- 3) Questionnaire
- 4) Observation
- 5) And Also, Literature Survey

Why Analysis is a difficult activity?

- Analysis is a **creative process**, that involves understanding problem domain, its associated constraints, and methods to overcome those constraints
- Three most common **sources of requirements difficulties** :-
 1. Fuzzy Description
 2. Incomplete Requirements
 3. Unnecessary Features

Business object analysis

- Business object analysis is a process of,
 - **understanding the system's requirements and**
 - **establishing the goals of an application**
- The outcome of the business object analysis is to,
 - identify classes that make up the business layer and
 - the relationships that play a role in achieving system goals
 - **To understand users' requirements :**
Find out how they “use” the system, by developing use cases

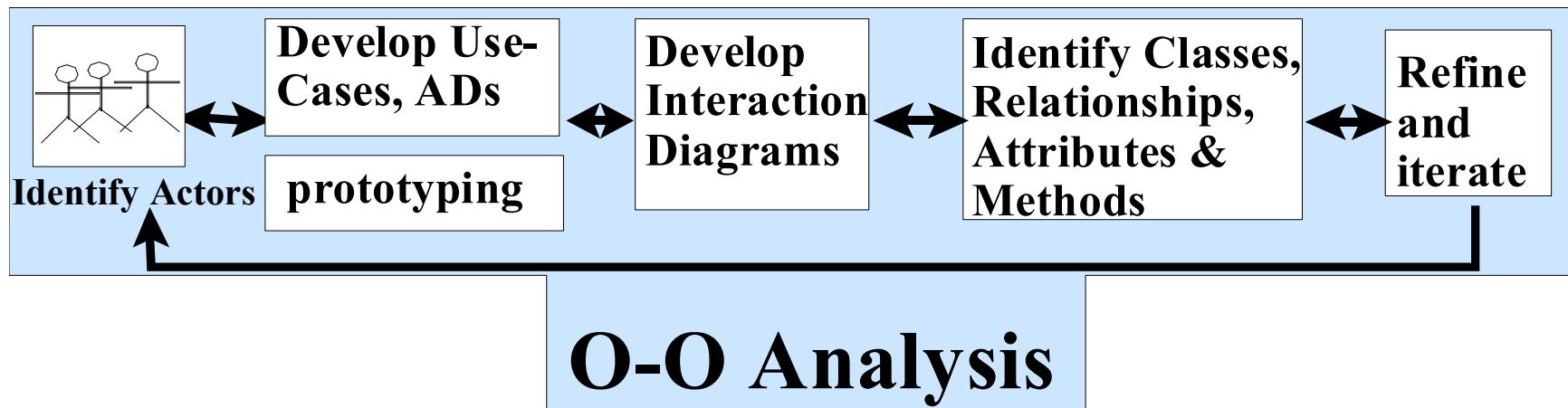


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



OOA phase of the unified approach uses actors and use cases to describe the system from users' perspective



OOA process consists of the following steps :

1. Identify the actors :

- Who is using the system
- Who will be using the system (in case of a new system)

2. Develop a simple business process model using UML activity diagram

3. Develop the use case :

- What are the users doing with the system
- What will users be doing with the new system (in case of a new system)
- Use cases provide comprehensive documentation of the system under study

4. Prepare interaction diagrams

- Determine the sequence
- Develop collaboration diagrams

5. Classification – develop a static UML class diagram :

- Identify classes
- Identify relationships
- Identify attributes
- Identify methods

6. Iterate and refine : if needed, repeat the preceding steps

Business process modelling

- Not necessary for all project
- When required business process and requirements can be modelled and recorded to any level of detail
- Activity diagram support this modelling
- Disadvantages:-
 - Time consuming process
- Advantages:-
 - familiarity

Use case model

- 1) Scenario for understanding the system
- 2) Interaction between user and system
- 3) Captures users goal and systems responsibility
- 4) Used to discover classes and relationship
- 5) Developed by talking to users

- Use case model – Provides external view of the system
- Object model (UML class diagram) – Provides internal view

Use cases and microscope

“ A use case is a sequence of transaction in a system whose task is to yield results of measurable value to an individual actor of the system ”

Use-Case model

- A use-case model is a model of the system's intended functions (use cases) and its surroundings (actors)
- The same use-case model is used in requirement analysis, design, & test
- Primary purpose is to communicate the system's functionality and behavior to the customer or end user
- Benefits of a use-case model : To communicate with the end users and domain experts

Actor & Use-Case

Actor

- Role played by the user with respect to the system
- Single actor may perform many use cases
- Can be external system
- Can be one get value from the system, or just participate in the use case

“an actor represents anything that interacts with the system”

“a user may play more than one role”

“a use case is a sequence of actions a system performs that yields an observable result of value to a particular actor”

- **Uses Association**
 - common sub flows are extracted and separate use case is created
 - Relationship between use-case and extracted one is called uses relationships
- **Extends Association**
 - Used when use case is similar to other, but do bit more or more Specialized
- **Abstract use case**
 - No initiating actor
 - Used by concrete use cases
- **Concrete use cases**
 - Interacts with actors

Identifying actor (cont..)

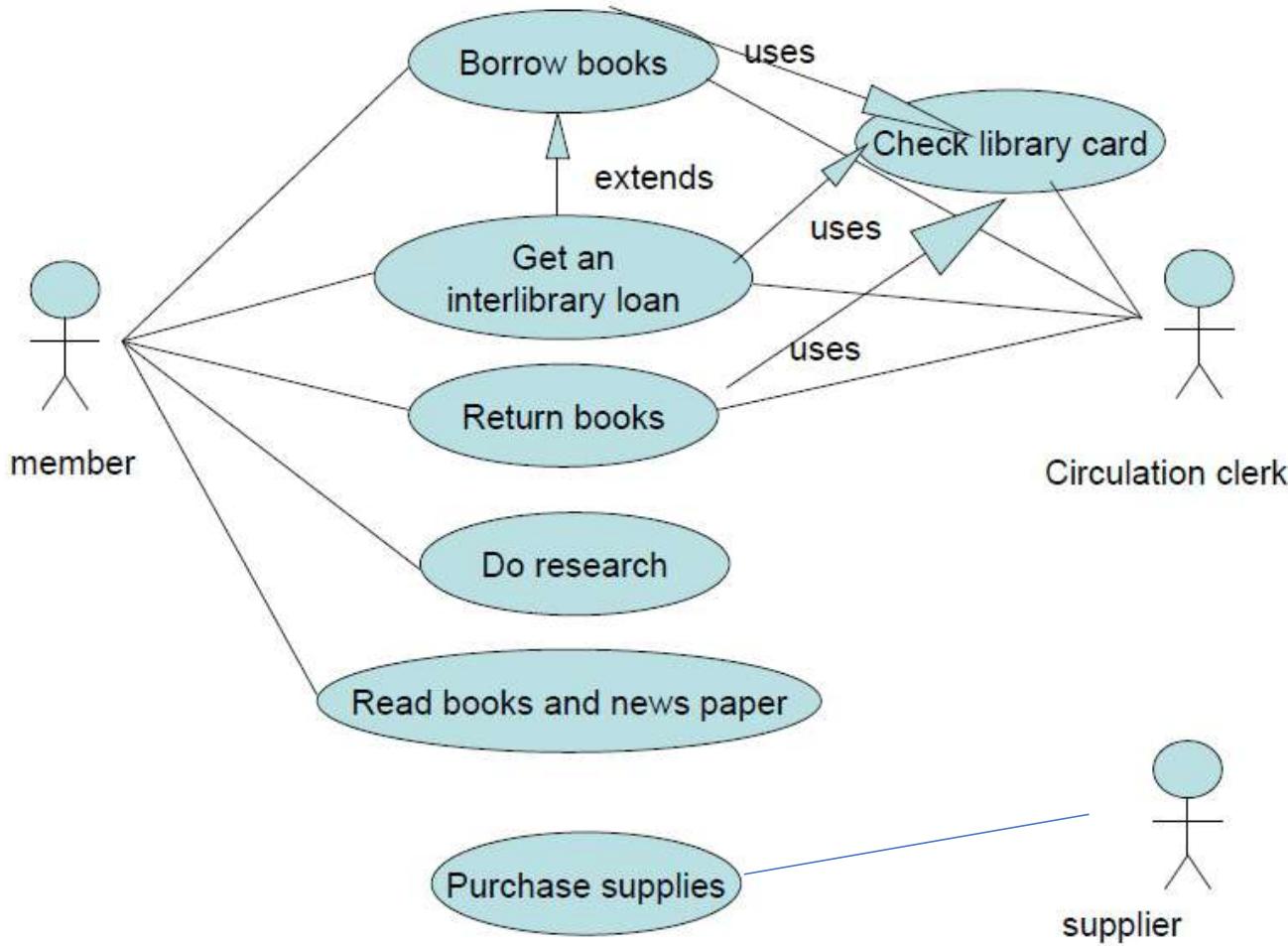
- Two-three rule
 - Used to identify the actors
 - Start with naming at least 2 or 3 , people who could serve as the actor in the system.other actor can be identified in the subsequent iteration



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



How detailed must a use case be? When to stop decomposing it and when to continue

- Develop system use case diagram
- Draw package
- Prepare at least one scenario for each use case
- When the lowest use case level is arrived, which can't be broken further, sequence and collaboration diagram is drawn

Identifying actors

- Who is using the system? Or,
- Who affects the system? Or
- which user groups are needed by the system to perform its function?
- Which external hardware or other systems use the system to perform tasks?
- Which problem does this application solve?
- How do users use the system ? What are they doing with the system?



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Guidelines for Finding Use Cases

- For each actor, find the tasks and functions that the actor should be able to perform or that the system needs the actor to perform.
- Name the use cases
- Describe the use cases briefly by applying terms with which the user is familiar.

Dividing Use Case into Packages

- Whole system is divided into many packages
- Each package encompasses multiple use cases

Naming a Use Case

- Provide a general description of the use-case function
- Name should express what happens when an instance of use case is performed

Developing effective documentation

- An effective document can serve as a **communication vehicle among the project's team members**, or it can **serve as initial understanding of the requirements**.
- Important factor in making a decision about committing (assigning, handover, giving) resources
- Mainly depends on organization's rules and regulations.

Guidelines for developing effective documentation:

According to Bell and Evans:

- Common cover [*common cover sheet*]
- 80-20 rule [*80% work – 20% document*]
- Familiar vocabulary [*don't use buzz words*]
- Make the document as short as possible [*eliminate all repetition*]
- Organize the document [*use rules of good organization*]

objectives

- Classification
- Noun Phrase Approach
- Common class pattern approach
- Use-Case Driven Approach
- Class, Responsibilities and collaborators



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Classification and Noun phrase approach



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Classification

- Classification is the process of checking to see if an **object belongs to a category or a class** and it is regarded as a basic attribute of human nature.
- A class is a specification **of structure, behaviour, and the description of an object.**

... Intelligent classification is intellectually hard work, and it best comes about through an incremental and iterative process

Booch



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



..There is no such thing as the perfect class structure, nor the right set of objects. As in any engineering discipline, our design choice is compromisingly shaped by many competing factors.

Booch



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



The Challenge of Classification

- Intelligent classification is intellectually hard work and may seem rather arbitrary.
- Martin and Odell have observed in object-oriented analysis and design, that

“In fact, an object can be categorized in more than one way.”



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013

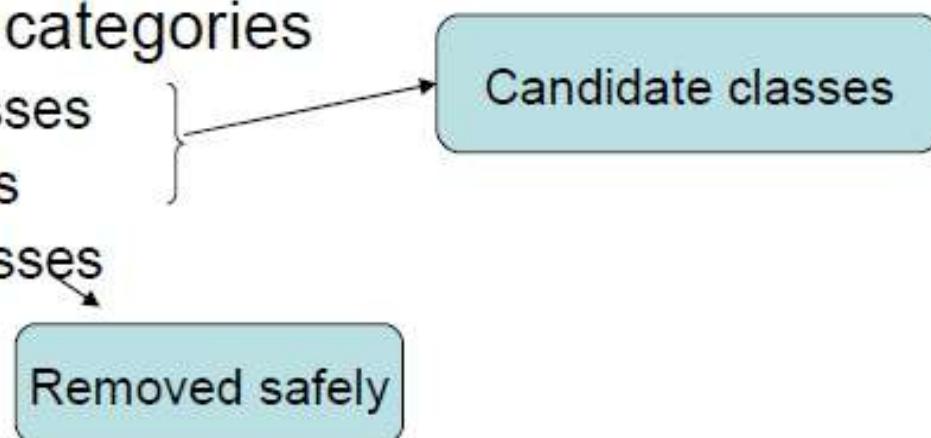


Approaches for Identifying Classes

- The noun phrase approach.
- The common class patterns approach.
- The use-case driven approach.
- Class, Responsibilities and Collaborators (CRC)

Noun phrase approach

- Identify Noun phrases from requirements or use cases
- Nouns - classes
- Verbs - methods
- All plurals → singular
- Create a List of nouns
 - Divided into 3 categories
 - Relevant classes
 - Fuzzy classes
 - Irrelevant classes



Relevent Classes

Fuzzy Classes

Irrelevent Classes



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- Identifying tentative classes
- Guidelines
 - Look for nouns and noun phrases in the use case
 - Some classes are implicit and taken from knowledge
 - Avoid computer implementation classes (defer them to design phase). application domain related classes makes sense
 - Carefully choose and define class name



UNIVERSITY

Private University Estd. in Karnataka State by Act No. 41 of 2013



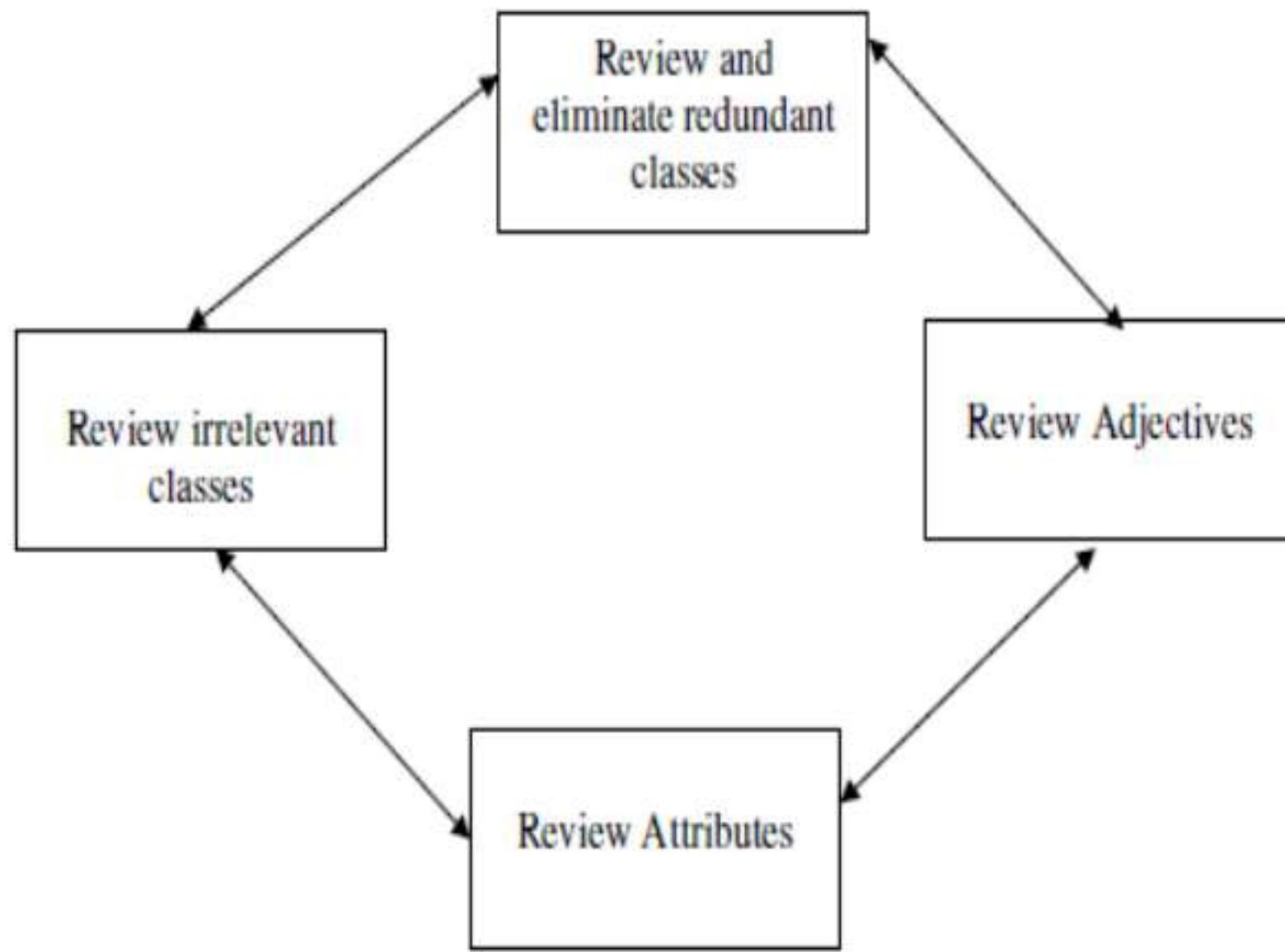
Guidelines :Selecting classes from relevant and fuzzy category

- **Redundant classes**
 - Avoid
 - Choose more meaningful name and name used by user
- **Adjective classes**
 - Adjective can suggest
 - Different kind of object
 - Different use of same object
 - Utterly irrelevant

eg: adult member and youth member
- **Attribute classes**
 - Objects used only as value can be treated as attribute instead of classes
- **Irrelevant classes**
 - Relevant class have statement of purpose.
 - Irrelevant classes - have no statement of purpose

Eliminating redundant classes and refining

- 1) Review redundant class
- 2) Review irrelevant class
- 3) Review adjectives
- 4) Review attributes



Initial list of noun classes : in vianet bank

- Account
- Account balance
- Amount
- Approval process
- Atm card
- Atm machine
- Bank
- Bank client
- Card
- Cash
- Check
- Checking
- Checking account
- Client
- Client's account
- Currency
- Dollar
- Envelope
- Four digits
- Fund
- Invalid pin
- Message
- Money
- Password
- PIN
- Pin code
- Record
- Savings
- Savings account
- Step
- System
- Transaction
- transaction history

Removing irrelevant classes

- Account
- Account balance
- Amount
- Approval process
- Atm card
- Atm machine
- Bank
- Bank client
- Card
- Cash
- Check
- Checking
- Checking account
- Client
- Client's account
- Currency
- Dollar
- Envelope
- Four digits
- Fund
- Invalid pin
- Message
- Money
- Password
- PIN
- Pin code
- Record
- Savings
- Savings account
- Stop
- System
- Transaction
- transaction history

Removing redundant classe and building common vocabulary

- Account
- Account balance
- Amount
- Approval process
- Atm card
- Atm machine
- Bank
- Bank client
- ~~Card~~
- Cash
- Check
- ~~Checking~~
- Checking account
- Client
- ~~Client's~~ account
- Currency
- Dollar
- ~~Envelope~~
- ~~Four digits~~
- Fund
- Invalid pin
- Message
- ~~Money~~
- Password
- PIN
- ~~Pin code~~
- Record
- ~~Savings~~
- ~~Savings~~ account
- ~~Stop~~
- System
- Transaction
- transaction history

Reviewing the classes containing adjectives

- When class represented by noun behaves differently when adjective is applied to it, then separate class has to be created
- In this ex no such classes

Reviewing the possible attributes

- Noun phrases used only as values should be treated as attributes



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Reviewing possible attributes

- Account
- ~~Account balance~~
- ~~Amount~~
- Approval process
- Atm card
- Atm machine
- Bank
- Bank client
- ~~Card~~
- Cash
- Check
- ~~Checking~~
- Checking account
- ~~Client~~
- ~~Client's~~ account
- Currency
- Dollar
- ~~Envelope~~
- ~~Four digits~~
- Fund
- ~~Invalid pin~~
- Message
- ~~Money~~
- ~~Password~~
- ~~PIN~~
- Pin code
- Record
- ~~Savings~~
- Savings account
- ~~Stop~~
- System
- Transaction
- ~~transaction history~~

Reviewing the class purpose

- Include classes with
 - Purpose
 - Clear definition
 - Necessary in achieving system goal
- Eliminate classes with no purpose
- Ex: Candidate class with purpose are
 - ATM machine class
 - ATM card class
 - Bankclient class
 - Bank class
 - Account class
 - Checking account class
 - Saving account class
 - Transaction class

Home work

- Apply noun phrase approach for Grocery store problem.

- A store wants to automate its inventory. It has point-of-sale terminals that can record all of the items and quantities that a customer purchases. Another terminal is also available for the customer service desk to handle returns. It has a similar terminal in the loading dock to handle arriving shipments from suppliers. The meat department and produce department have terminals to enter losses/discounts due to spoilage.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- **Step.1:**
- **Identify nouns**
- Store
- Inventory
- Point-of-sale terminal
- Terminals
- Items
- Quantity
- Purchase
- Customer
- Customer service desk
- Handle returns
- Returns
- Loading dock
- Shipment
- Handle shipment
- Suppliers
- Meat department
- Produce department
- Department
- Enter losses
- Enter discount
- Spoilage

- **Step.2:**
- **Eliminate irrelevant nouns**
- Store
 - Point-of-sale terminal
 - inventory
 - Item
 - Customer
 - Customer service desk
 - Handle returns
 - Returns
 - Handle shipment
 - Shipment
- Enter losses
- Enter discount
- Meat department
- Produce department
- Department

- **Step.3**
- **Eliminate redundancies**
- Store
- Point-of-sale terminal
- Item
- Customer service desk
- Handle returns
- Handle shipment
- Meat department
- Produce department
- Enter losses
- Enter discount



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- **Step.4:**
- The final set of classes and objects after the elimination process.
- Point-of-sale terminal
- Item
- Handling return
- Handling shipment
- Enter losses
- Enter discount
- Meat department
- Produce department
- Store



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Hospital Management System

a comprehensive example

- Using Noun approach



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



System Domain

- The hospital has several wards divided into male wards & female wards.
- Each ward has a fixed number of beds. When a patient is admitted they are placed onto an appropriate ward.
- The Doctors in the hospital are organized into teams such as Orthopedics A, or Pediatrics, and each team is headed by a consultant doctor. There must be at least one grade 1 junior doctor in each team.

System Domain

- The Administration department keeps a record of these teams and the doctors allocated to each team.
- A patient on award is under the care of a single team of doctors, with the consultant being the person who is responsible for patient.
- A record is maintained of who has treated the patient, and it must be possible to list the patients on award and the patients treated by a Particular team.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Noun/ Noun phrase:

Hospital → System general

Ward → class

Male wards Female wards } Property (attribute)

Fixed number of beds → Att(fixed → constraint)

Admitted → relation

Patient → class

Appropriate ward

They (patient)

Doctors → class

Organized → relation

Teams → class

Orthopedics A

Prediatrics } language

Headed by → relation

Consultant doctor } subclass
 Property

One grade → attribute

1 junior → subclass

Administrator department

Record

Under the care → relation

Single team of doctors

person → class

Responsible for → relation

List the patient

Patient treated by → relation
 particular

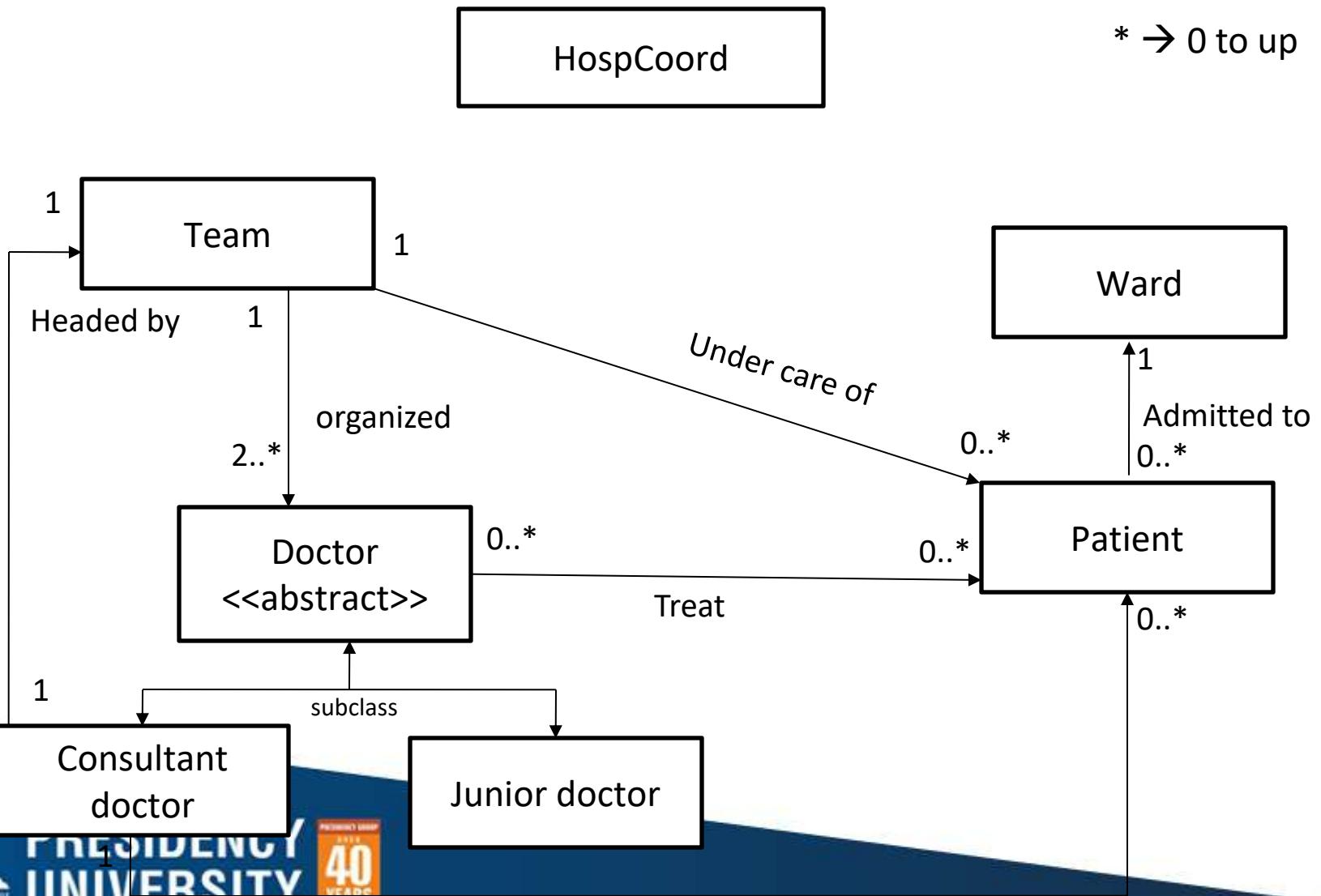


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Class Diagram



COMMON CLASS PATTERNS APPROACH

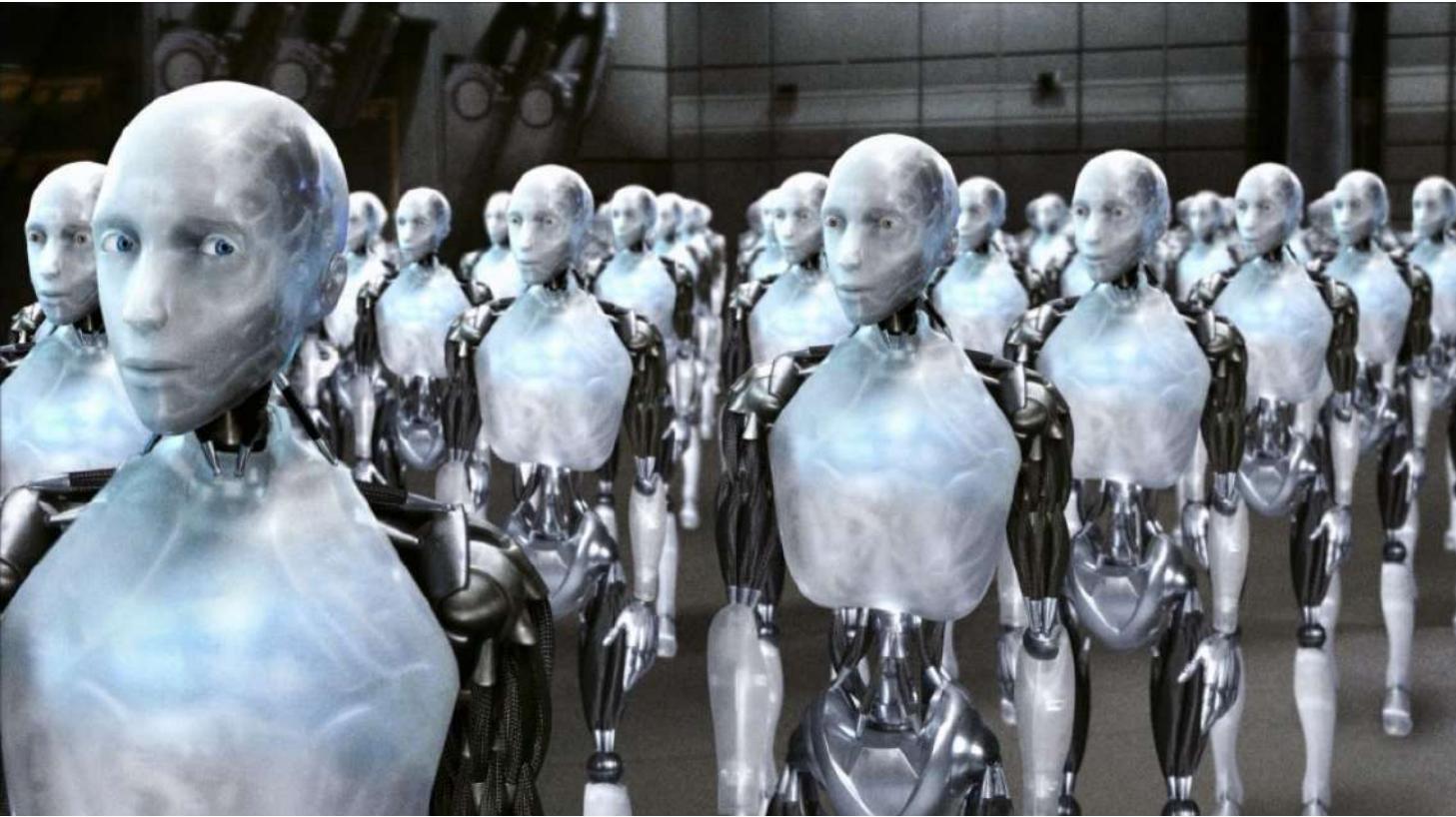


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Pattern



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Common Class Pattern Approach

- This approach is based on the knowledge – base of the common classes that have been proposed by the various researchers.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Patterns for finding the candidate class and objects :-

- Concept Class [Idea or Understanding]
- Event Class [points in time to be recorded]
- Organization Class [collection of people, resources, groups]
- People Class [roles user play to interact with the system]
- Places Class [physical locations the system has info about]
- Tangible things and devices [physical objects]

Candidate Classes-Events

- There are points in time that must be recorded and remembered.
- Things happen, usually to do something else, at a given date and time, or as step in an ordered sequence.
- For example
 - Order placed time which is an event must be remembered



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



In ATM SYSTEM

Events?



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



List of event in ATM SYSTEM

- Account class: is formal class it defines common behavior inherited
 - checkingAccount** class- model checking clients accounts
 - savingAccount** class-models for clients savings accounts
 - Transaction** Class-tracking transaction,date,time



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Candidate Classes- Organization

- The organizational units that peoples belongs to,
- For example accounting department might be considered as a potential class.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Organization in ATM?

?



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Organization

- Bank class: Bank clients belong to Bank class
- Its repository of **accounts** and processes the accounts transaction



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Candidate Classes-people and person(Roles and roles played)

- **The different roles users play in interacting with the application.**



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Candidate Classes-People (Can't)

- It can be divided into two types (Coad & Yourdon):
- Those representing users of the system, such as an operator, or a clerk;



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Candidate Classes-People (Can't)

- **Those people who do not use the system but about whom information is kept.**
 - Some examples are Client, Employee, Teacher, Manager.



People

- Bank clients
- ATM OPERATORS



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



For ATM system

Event

- Account

Organization

- Bank

People

- Bank clients
- ATM OPERATORS



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Candidate Classes-Places

- These are physical locations, such as buildings, stores, sites or offices that the system must keep information about.



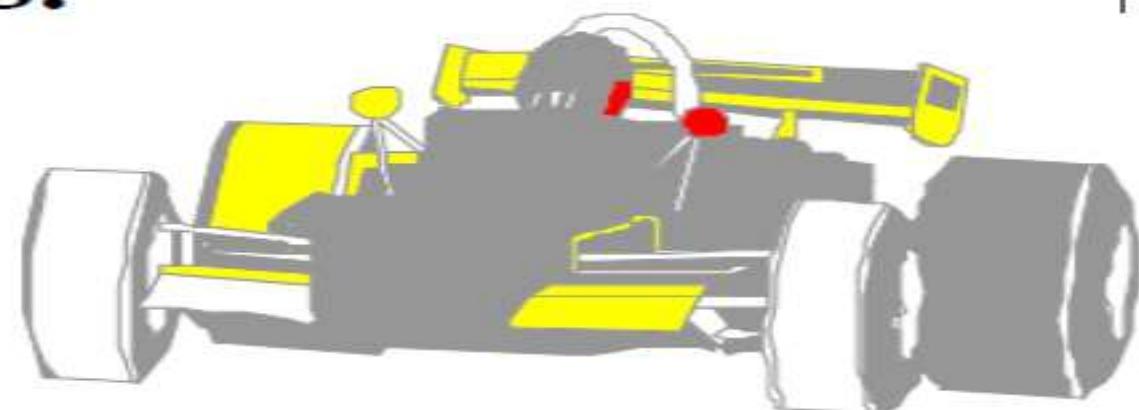
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Candidate Classes-Tangible Things and Devices

- **Physical objects, or group of objects, that are tangible, and devices with which the application interacts.**
- **For example, cars, pressure sensors.**



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Candidate Classes-Concepts

- Concepts are principles or ideas not tangible but used to organize or keep track of business activities and/or communications.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Work out Example

- Apply common class pattern approach for Elevator system



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Identifying Object relationships



**PRESIDENCY
UNIVERSITY**

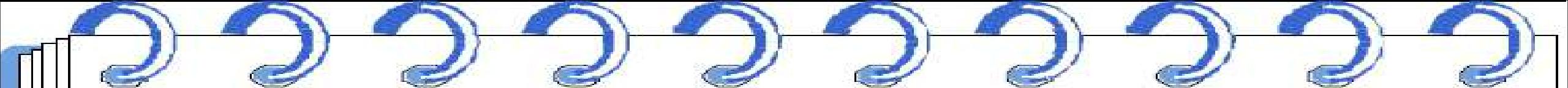
Private University Estd. in Karnataka State by Act No. 41 of 2013





Goals

- Analyzing relationships among classes.
- Identifying association.
- Association patterns.
- Identifying super- and subclass hierarchies.



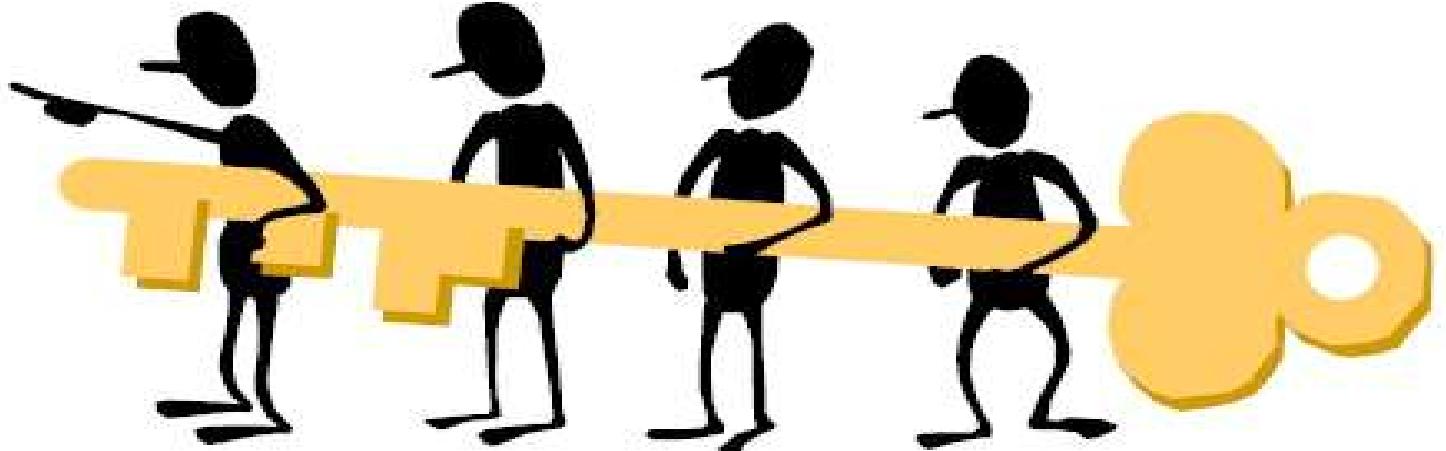
Introduction

- Identifying aggregation or a-part-of compositions.
- Class responsibilities.
- Identifying attributes and methods by analyzing use cases and other UML diagrams.



*Objects contribute to the behavior of
the system by collaborating with
one another.*

—Grady Booch

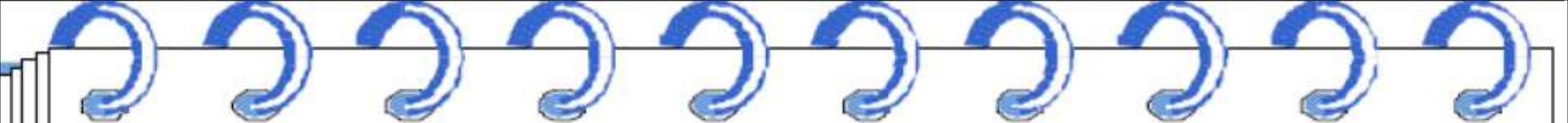




In OO environment, an application is the interactions and relationships among its domain objects.

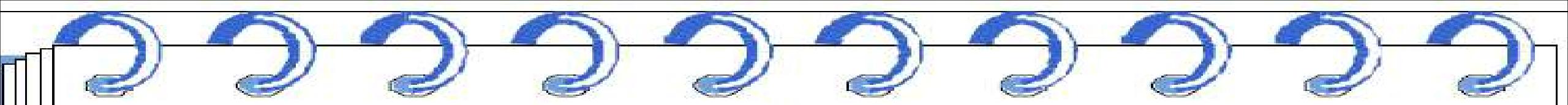
All objects stand in relationship to others, on whom they rely for services and controls.





Objects Relationships

- Three types of relationships among objects are:
 - *Association.*
 - *Super-sub structure* (also known as *generalization hierarchy*).
 - Aggregation and a-part-of structure.



Associations

- A reference from one class to another is an association.
- Basically a dependency between two or more classes is an association.
- For example, Jackie *works for* John.



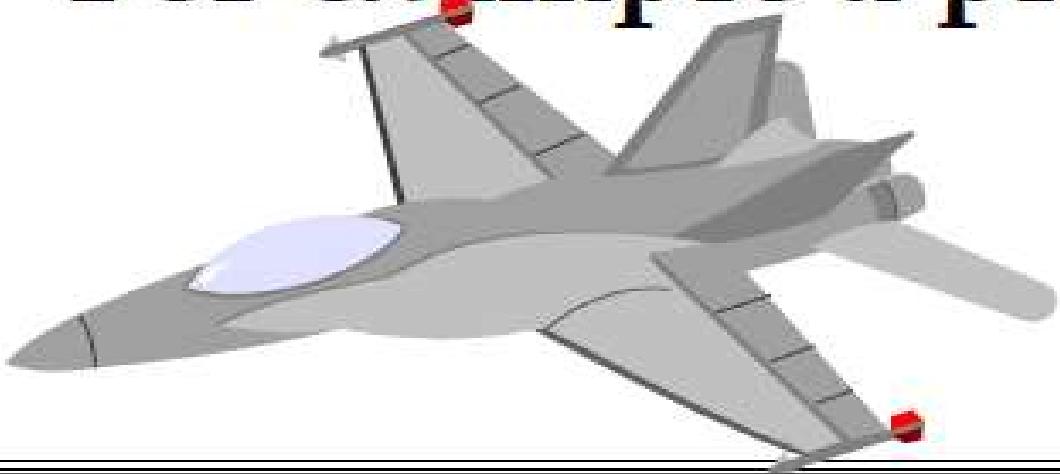
Associations (Con't)

- Some associations are implicit or taken from general knowledge.



Guidelines For Identifying Associations

- Association often appears as a verb in a problem statement and represents relationships between classes.
- For example a pilot *can fly* planes.



Guidelines For Identifying Associations (Con't)

- Association often corresponds to verb or prepositional phrases such as *part of*, *next to*, *works for*, *contained in*, etc.



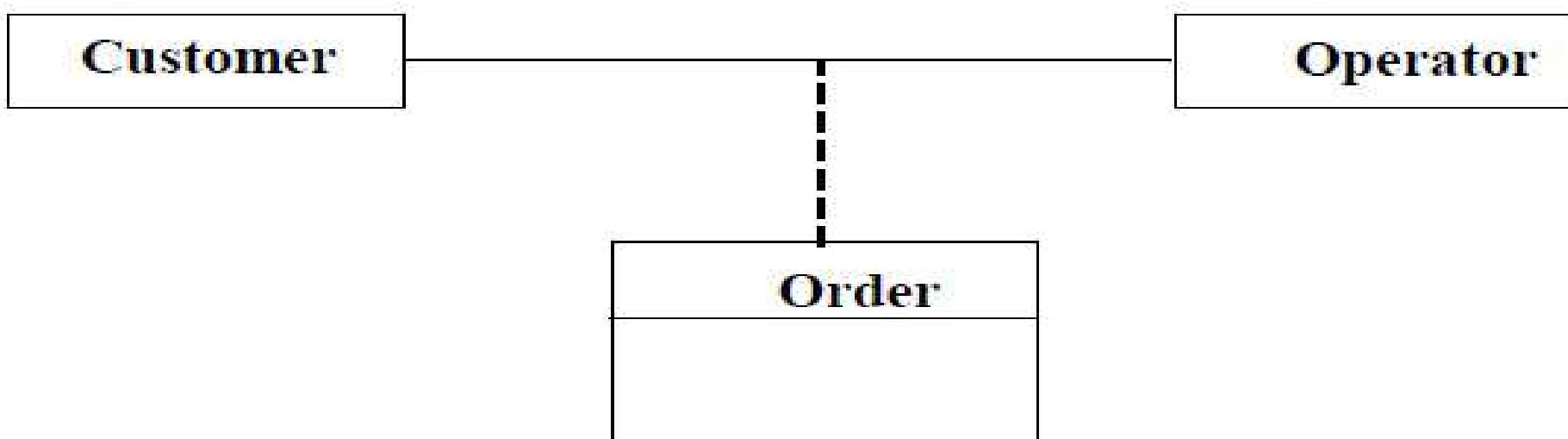
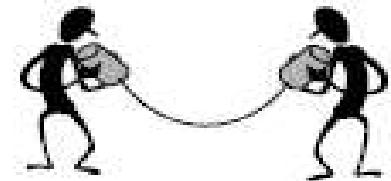
Common Association Patterns

- Common association patterns include:
- Location Association: *next To, part of, contained in, ingredient of etc.* :
- For example cheddar cheese is an *ingredient of* the French soup.

Common Association Patterns

(Con't)

- **Communication association – *talk to, order to.***
- For example, a customer places an order with an operator person.



Eliminate Unnecessary Associations

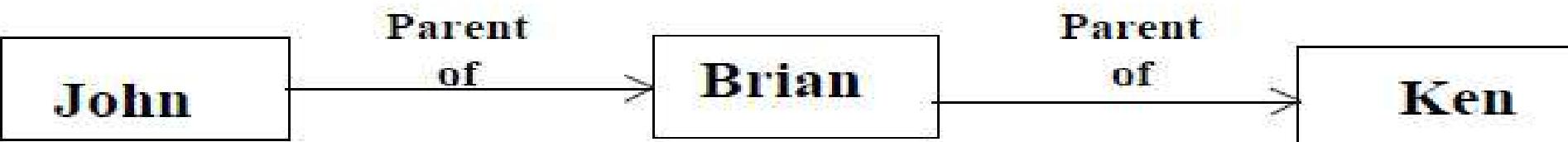
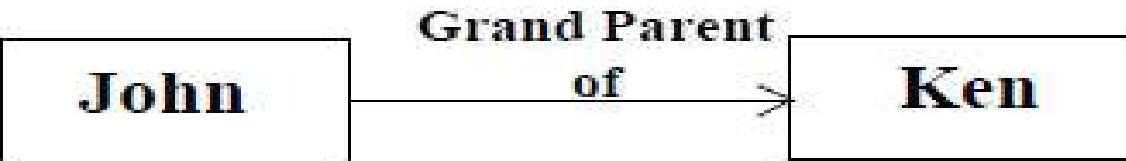
- *Implementation association.* Defer implementation-specific associations to the design phase.
- *Ternary associations.* Ternary or n-ary association is an association among more than two classes

Eliminate Unnecessary Associations (Con't)

- *Directed actions* (derived) *associations* can be defined in terms of other associations.
- Since they are redundant you should avoid these types of association.

Eliminate Unnecessary Associations (Con't)

- Grandparent of Ken can be defined in terms of the parent association.



Super-Class and Sub-Class



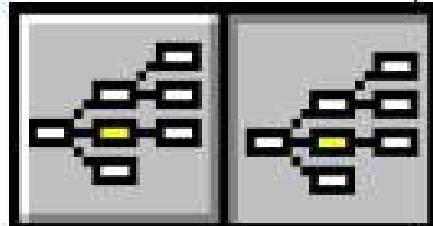
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



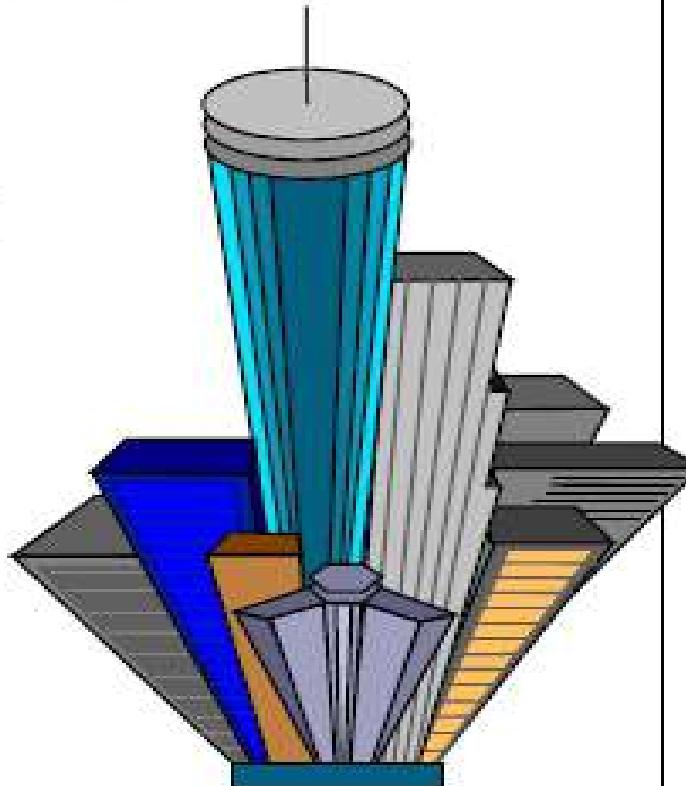
Superclass-Subclass Relationships

- Recall that at the top of the class hierarchy is the most general class, and from it descend all other, more specialized classes.
- Sub-classes are more specialized versions of their super-classes.



Guidelines For Identifying Super-sub Relationships: Top-down

- Look for noun phrases composed of various adjectives on class name.
- Example, Military Aircraft and Civilian Aircraft.
- Only specialize when the sub classes have significant behavior.



Guidelines For Identifying Super-sub Relationships: Bottom-up

- Look for classes with similar attributes or methods.
- Group them by moving the common attributes and methods to super class.
- Do not force classes to fit a preconceived generalization structure.



Guidelines For Identifying Super-sub Relationships: Reusability

- Move attributes and methods as high as possible in the hierarchy.
- At the same time do not create very specialized classes at the top of hierarchy.
- This balancing act can be achieved through several iterations.



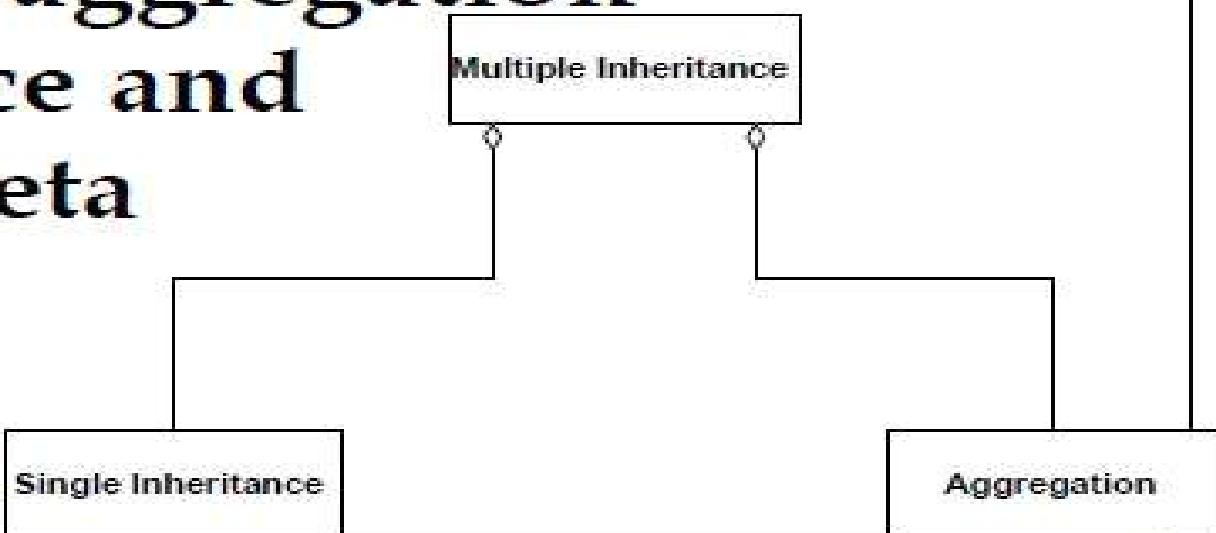
Guidelines For Identifying Super-sub Relationships: Multiple inheritance

- Avoid excessive use of multiple inheritance.
- It is also more difficult to understand programs written in multiple inheritance system.



Multiple inheritance (Con't)

- One way to achieve the benefits of multiple inheritance is to inherit from the most appropriate class and add an object of other class as an attribute.
- In essence, a multiple inheritance can be represented as an aggregation of a single inheritance and aggregation. This meta model reflects this situation.



a part of relationship

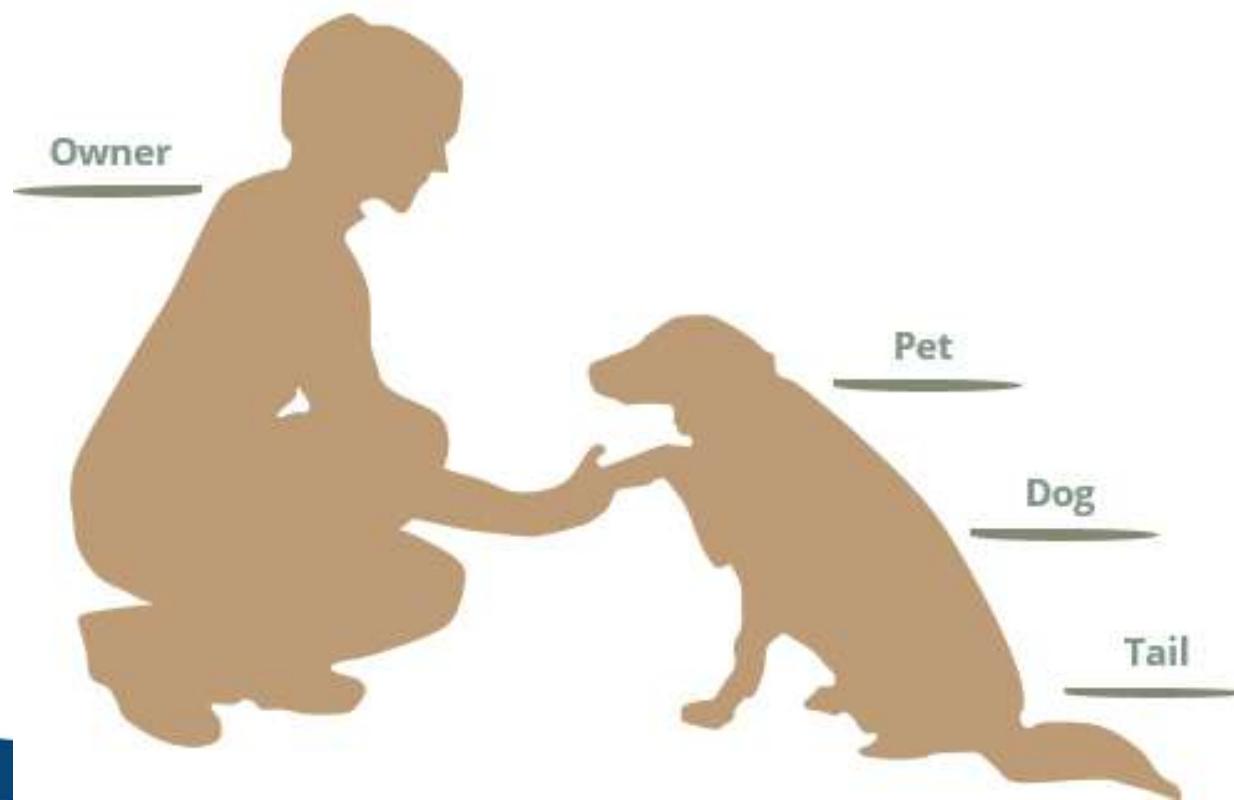


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Association • Aggregation • Composition



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



We see the following relationships:

- owners **feed** pets, pets **please** owners (**association**)
- a tail **is a part** of both dogs and cats (**aggregation / composition**)
- a cat **is a kind** of pet (**inheritance / generalization**)

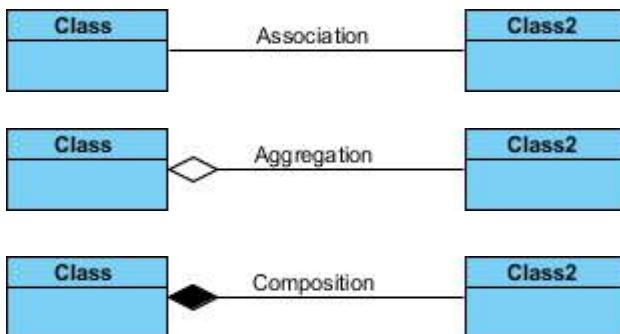


**PRESIDENCY
UNIVERSITY**

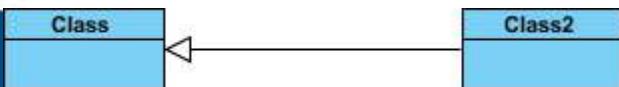
Private University Estd. in Karnataka State by Act No. 41 of 2013



The figure below shows the three types of association connectors: association, aggregation and composition.



The figure below shows a generalization.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





A-Part-of Relationship - Aggregation

- *A-part-of relationship, also called aggregation, represents the situation where a class consists of several component classes.*

A-Part-of Relationship - Aggregation (Con't)

- This does not mean that the class behaves like its parts.
- For example, a car consists of many other classes, one of them is a radio, but a car does not behave like a radio.



**PRESIDENCY
UNIVERSITY**

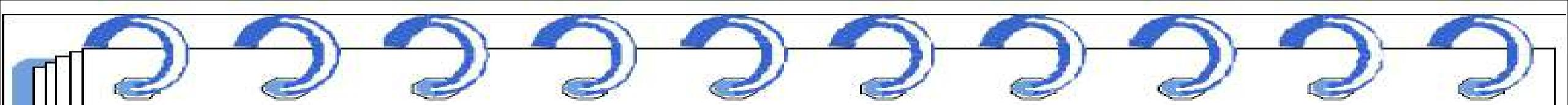
Private University Estd. in Karnataka State by Act No. 41 of 2013





A-Part-of Relationship - Aggregation (Con't)

- **Two major properties of a-part-of relationship are:**
 - transitivity
 - antisymmetry



Transitivity

- If A is part of B and B is part of C , then A is part of C .
- For example, a carburetor is part of an engine and an engine is part of a car; therefore, a carburetor is part of a car.



Antisymmetry

- If A is part of B , then B is not part of A .
- For example, an engine is part of a car, but a car is not part of an engine.

A-Part-of Relationship Patterns

Assembly

- An assembly-Part situation physically exists.
- For example, a French soup consists of onion, butter, flour, wine, French bread, cheddar cheese, etc.





A-Part-of Relationship Patterns Container

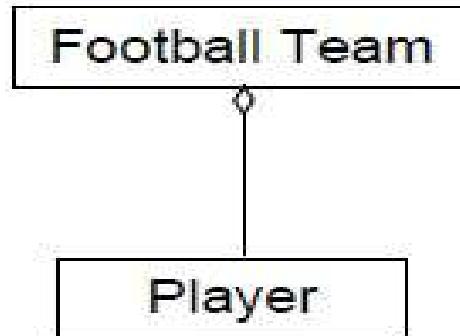
- A case such as course-teacher situation, where a course is considered as a container. Teachers are assigned to specific courses.



A-Part-of Relationship Patterns

Collection-Member

- A soccer team is a collection of players.



Class responsibility



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





Class Responsibility: Identifying Attributes and Methods

- Identifying attributes and methods, like finding classes, is a difficult activity.
- The use cases and other UML diagrams will be our guide for identifying attributes, methods, and relationships among classes.

Identifying Class Responsibility by Analyzing Use Cases and Other UML Diagrams

- **Attributes can be identified by analyzing the use cases, sequence/collaboration, activity, and state diagrams.**

Responsibility

- How am I going to be used?
- How am I going to collaborate with other classes?
- How am I described in the context of this system's responsibility?
- What do I need to know?
- What state information do I need to remember over time?
- What states can I be in?

Assign Each Responsibility To A Class

- Assign each responsibility to the class that it logically belongs to.
- This also aids us in determining the purpose and the role that each class plays in the application.



Identifying attributes and methods by analyzing use cases and other UML diagrams



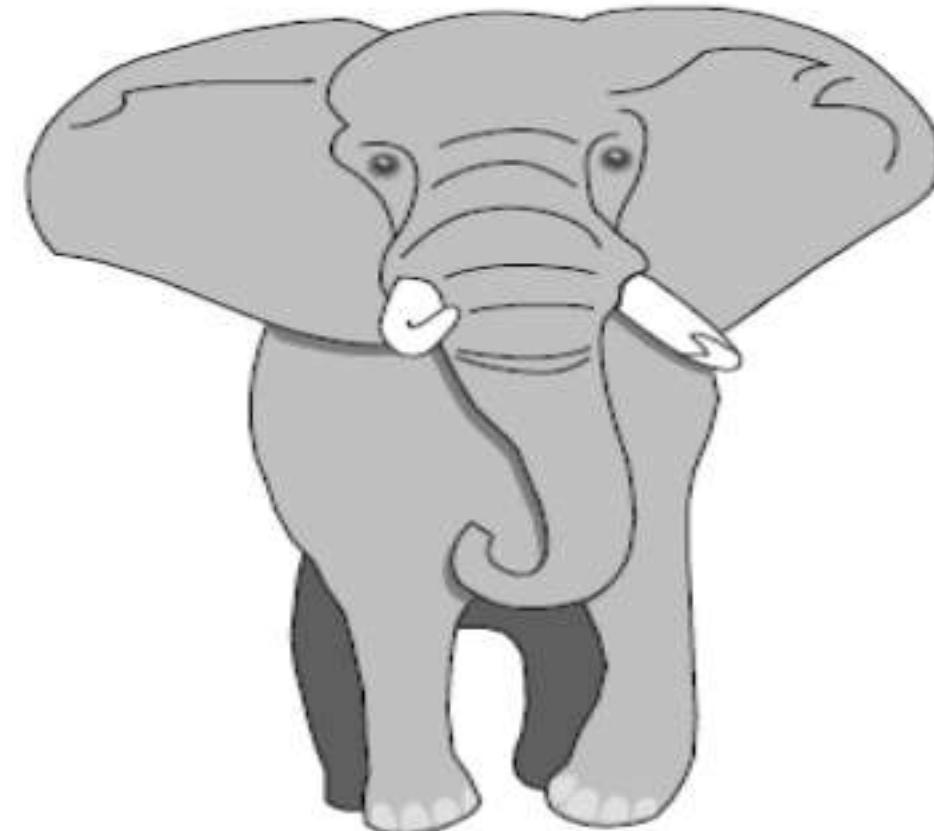
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Object Responsibility: Attributes

- **Information that the system needs to remember.**



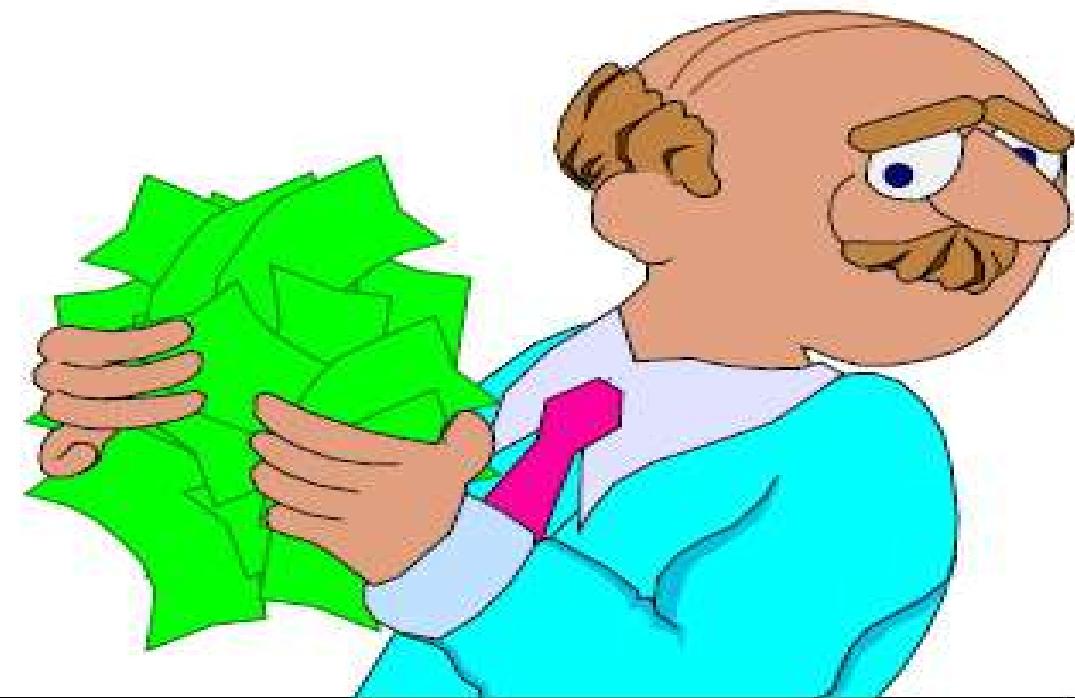
Guidelines For Identifying Attributes Of Classes

- Attributes usually correspond to nouns followed by possessive phrases such as *cost of* the soup.



Guidelines For Identifying Attributes Of Classes (Con't)

- Keep the class simple; only state enough attributes to define the object state.



Guidelines For Identifying Attributes Of Classes (Con't)

- **Attributes** are less likely to be fully described in the problem statement.
- You must draw on your knowledge of the application domain and the real world to find them.



Guidelines For Identifying Attributes Of Classes (Con't)

- Omit derived attributes.
- For example, don't use **age** as an attribute since it can be derived from date of birth.
- Drive attributes should be expressed as a method.

Guidelines For Identifying Attributes Of Classes (Con't)

- **Do not carry discovery of attributes to excess.**
- **You can always add more attributes in the subsequent iterations.**



Object Responsibility: Methods & Messages

- **Methods and messages are the work horses of object-oriented systems.**
- **In O-O environment, every piece of data, or object, is surrounded by a rich set of routines called methods.**

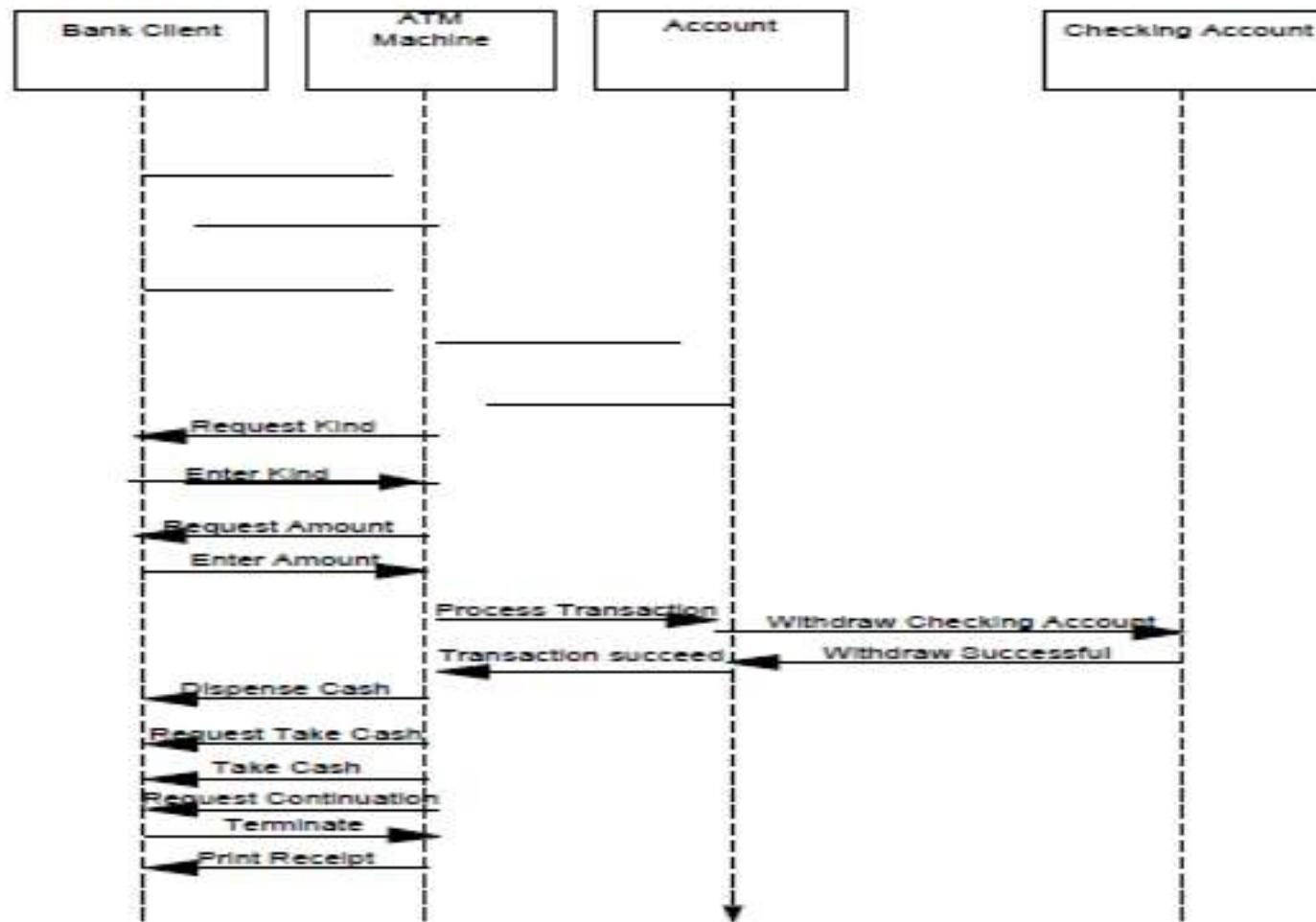




Identifying Methods by Analyzing UML Diagrams and Use Cases

- Sequence diagrams can assist us in defining the services the objects must provide.

Identifying Methods (Con't)





Identifying Methods (Con't)

- Methods usually correspond to queries about attributes (and sometimes association) of the objects.
- Methods are responsible for managing the value of attributes such as query, updating, reading and writing.

Identifying Methods (Con't)

- For example, we need to ask the following questions about soup class:
- What services must a soup class provide? And
- What information (from domain knowledge) is soup class responsible for storing?



Identifying Methods (Con't)

- Let's first take a look at its attributes which are:
 - name
 - preparation,
 - price,
 - preparation time and
 - oven temperature.



Identifying Methods (Con't)

- Now we need to add methods that can maintain these attributes.
- For example, we need a method to change a price of a soup and another operation to query about the price.

Identifying Methods (Con't)

- **setName**
- **getName**
- **setPreparation**
- **get Preparation**
- **setCost**
- **getCost**
- **setOvenTemperature**
- **getOvenTemperature**
- **setPreparationTime**
- **getPreparationTime**

Summary



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Summary

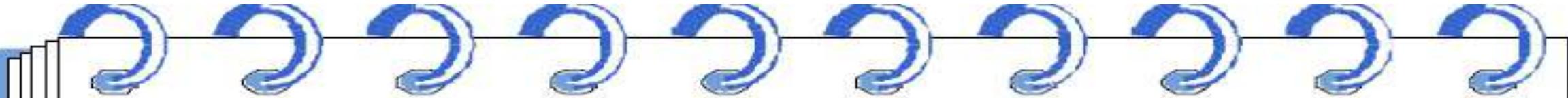
- We learned how to identify three types of object relationships:
- Association
- Super-sub Structure
(Generalization Hierarchy)
- A-part-of Structure





Summary (Con't)

- The **hierarchical relation** allows the sharing of properties or inheritance.
- A reference from one class to another is an **association**.
- The **A-Part-of Structure** is a special form of association.



Summary (Con't)

- Every class is responsible for storing certain information from domain knowledge .
- Every class is responsible for performing operations necessary upon that information.

- MODULE-3
- OBJECT ORIENTED DESIGN



- **MODULE-3**

- **COURSE OUTCOMES:**

Identify suitable object oriented design methodologies



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



OBJCT ORIENTED DESIGN

**Object Oriented Design Axioms-Designing Classes -Class visibility
-Redefining attributes -Designing methods and protocols -
Packages and managing classes -Access Layer- Object Storage
Persistence - Object oriented Database System-Designing view
layer classes -Macro level process -Micro level process.**



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



OO Design Process and Design Axioms

Goals

- The object-oriented design process.
- Object-oriented design axioms and corollaries.
- Design patterns.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Object-Oriented Design Process in the Unified Approach

- Objects discovered during the analysis can serve as the framework for design.
- During the design phase the **classes identified in object-oriented analysis must be revisited** with a shift in focus to their implementation.
- **New classes or attributes and methods must be added** for implementation purposes and user interfaces.
- The object-oriented design process consists of the following activities



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Object-Oriented Design Process in the Unified Approach

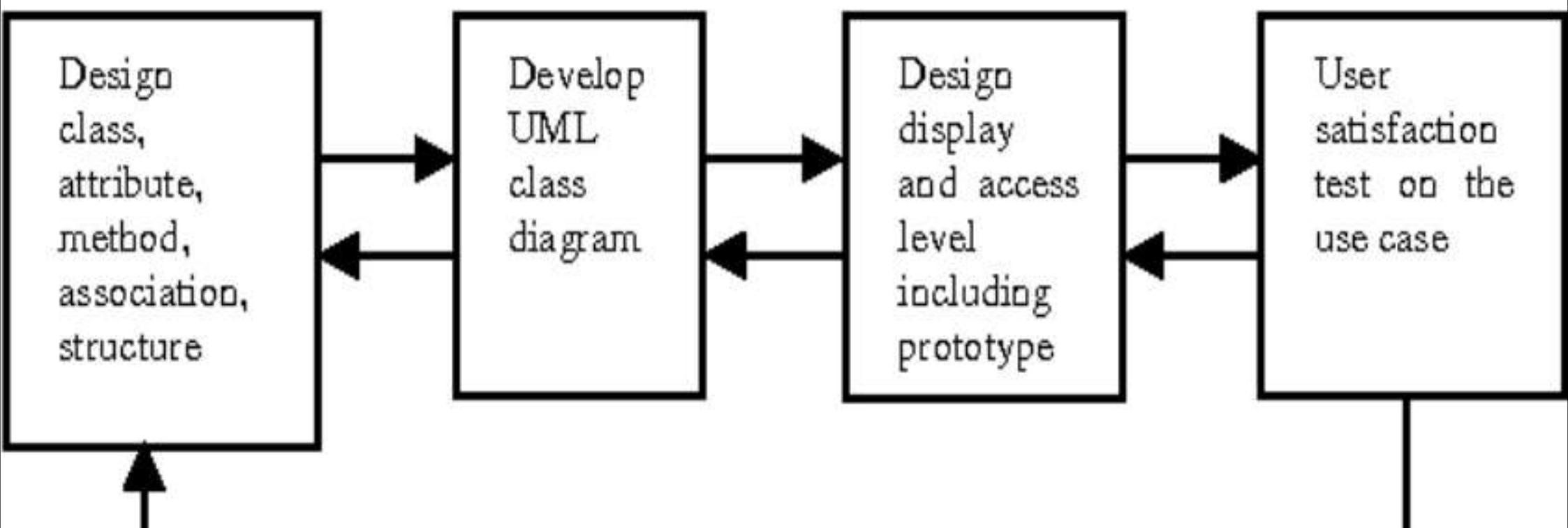


Fig. 2 Object-oriented Design (OOD) Process

OO Design Process

- 1. Apply design axioms**
- 2. Design the access layer**
- 3. Design the view layer classes.**
- 4. Iterate and refine the preceding steps.**



**PRESIDENCY
UNIVERSITY**

Private UN Object-Oriented Systems Development



OO Design Process

- 1. Apply design axioms to design classes, their attributes, methods, associations, structures, and protocols.



**PRESIDENCY
UNIVERSITY**

Private UN Object-Oriented Systems Development



OO Design Process

1. Apply Design Axioms

1.1. Refine and complete the static UML class diagram (object model) by adding details to the UML class diagram.



**PRESIDENCY
UNIVERSITY**

Private UN Object-Oriented Systems Development



OO Design Process

This step consists of the following activities:

1.1.1. **Refine attributes.**

1.1.2. **Design methods and protocols** by utilizing a UML activity diagram to represent the method's algorithm.

1.1.3. **Refine associations** between classes
(if required).

1.1.4. **Refine class hierarchy** and design with inheritance (if required).

• 1.2. **Iterate and refine again.**



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

OO Design Process (Con't)

- **2. Design the access layer**
- **2.1.** Create mirror classes. For every business class identified and created, create one access class.
- For example if there are 3 business class(class1, class2, class3) create 3 access class layers.(class 1DB,class2DB and class3DB)
 - **2.2.** define relationships among access layer classes.



**PRESIDENCY
UNIVERSITY**

Private UN Object-Oriented Systems Development



OO Design Process (Con't)

2.3. Simplify the class relationships. The main goal here is to eliminate redundant classes and structures.

- 2.3.1. **Redundant classes:** Do not keep two classes that perform similar *translate request* and *translate results* activities. Simply select one and eliminate the other.
- 2.3.2. **Method classes:** Revisit the classes that consist of only one or two methods to see if they can be eliminated or combined with existing classes.

2.4. Iterate and refine again.



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

OO Design Process (Con't)

3. Design the view layer classes.

3.1. Design the macro level user interface, identifying view layer objects.

3.2. Design the micro level user interface, which includes these activities:

- 3.2.1. Design the view layer objects by applying the design axioms and corollaries.
- 3.2.2. Build a prototype of the view layer interface.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



OO Design Process (Con't)

**4. Iterate and refine the preceding steps.
Reapply the design axioms and, if needed,
repeat the preceding steps.**



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Object-Oriented Design *Axioms,* *Theorems and Corollaries*

- **Axiom:**

An axiom is a **fundamental truth that always is observed to be valid** and for which there is no counterexample or exception.

- The axioms **cannot be proven** or derived but they cannot be invalidated by counterexamples or exceptions.



**PRESIDENCY
UNIVERSITY**

Private UNI Object-Oriented Systems Development



Object-Oriented Design *Axioms, Theorems and Corollaries*

- **Axiom:**

There are two design axioms applied to object-oriented design.

- Axiom 1 deals with relationships between system components and
- Axiom 2 deals with the complexity of design.



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Design Axioms

- **Axiom 1 deals with relationships between system components (such as classes, requirements, software components).**
- **Axiom 2 deals with the complexity of design.**



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Axioms

- Axiom 1. *The independence axiom.*
Maintain the independence of components.
- Axiom 2. *The information axiom.*
Minimize the information content of the design.



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Axioms, Theorems and *Corollaries (Con't)*

- A **Corollary** is a proposition that follows from an axiom or another proposition that has been proven.



**PRESIDENCY
UNIVERSITY**

Private UNI Object-Oriented Systems Development



Occam's Razor

- The best theory explains the known facts with a minimum amount of complexity and maximum simplicity and straightforwardness.

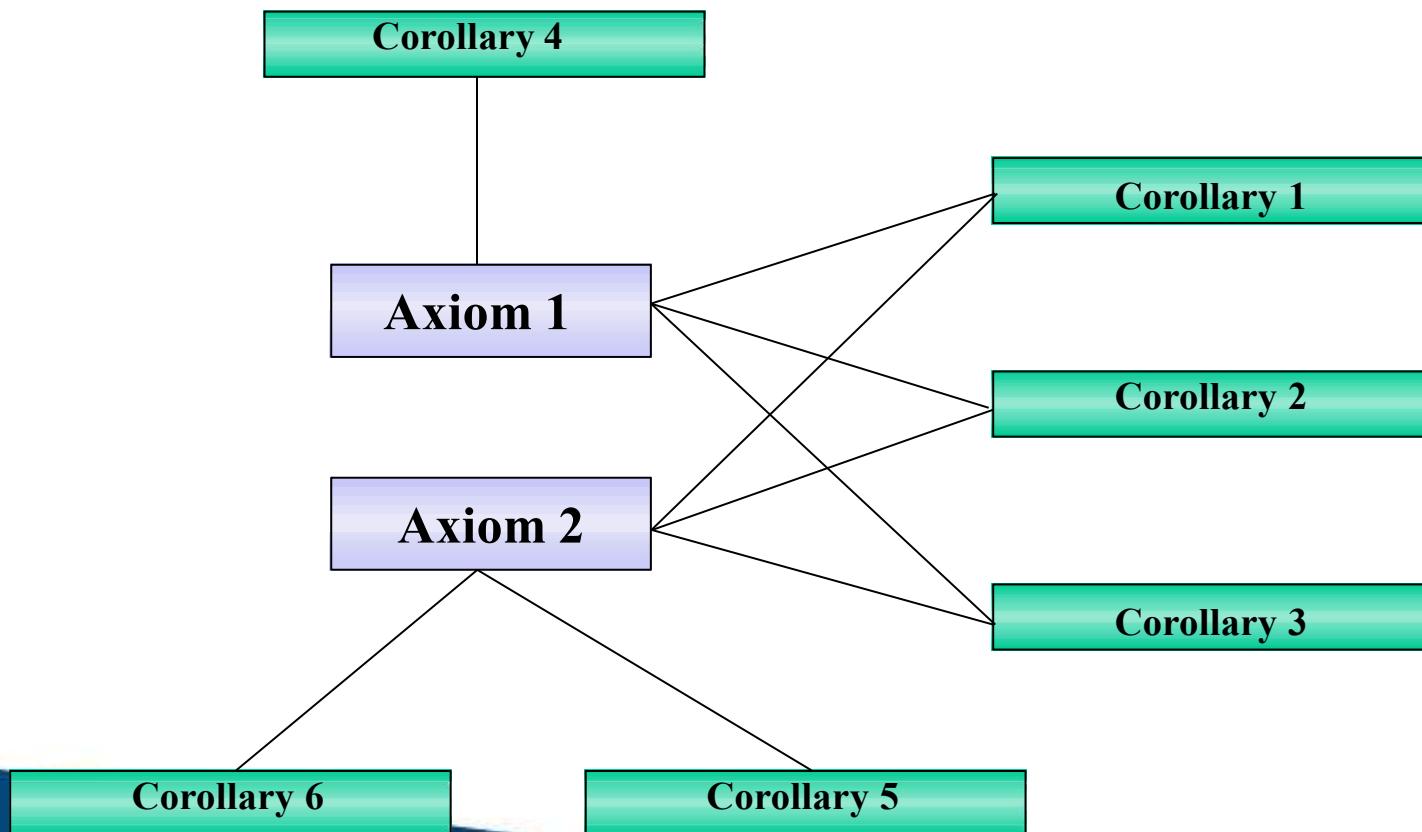


**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

The Origin of Corollaries



**PRESIDENCY
UNIVERSITY**



Private University Object-Oriented Systems Development

Corollaries

- Corollary 1. ***Uncoupled design with less information content.***
- Corollary 2. ***Single purpose.*** Each class must have single, clearly defined purpose.
- Corollary 3. ***Large number of simple classes.*** Keeping the classes simple allows reusability.



**PRESIDENCY
UNIVERSITY**

Private University Object-Oriented Systems Development



Corollaries (Con't)

- Corollary 4. ***Strong mapping.*** There must be a strong association between the analysis's object and design's object.
- Corollary 5. ***Standardization.*** Promote standardization by designing interchangeable components and reusing existing classes or components.



**PRESIDENCY
UNIVERSITY**

Private UNI Object-Oriented Systems Development



Corollaries (Con't)

- Corollary 6. ***Design with inheritance.*** Common behavior (methods) must be moved to superclasses.
- The superclass-subclass structure must make logical sense.



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Coupling and Cohesion

- Coupling is a measure of the strength of association among objects.
- Cohesion is interactions within a single object or software component.

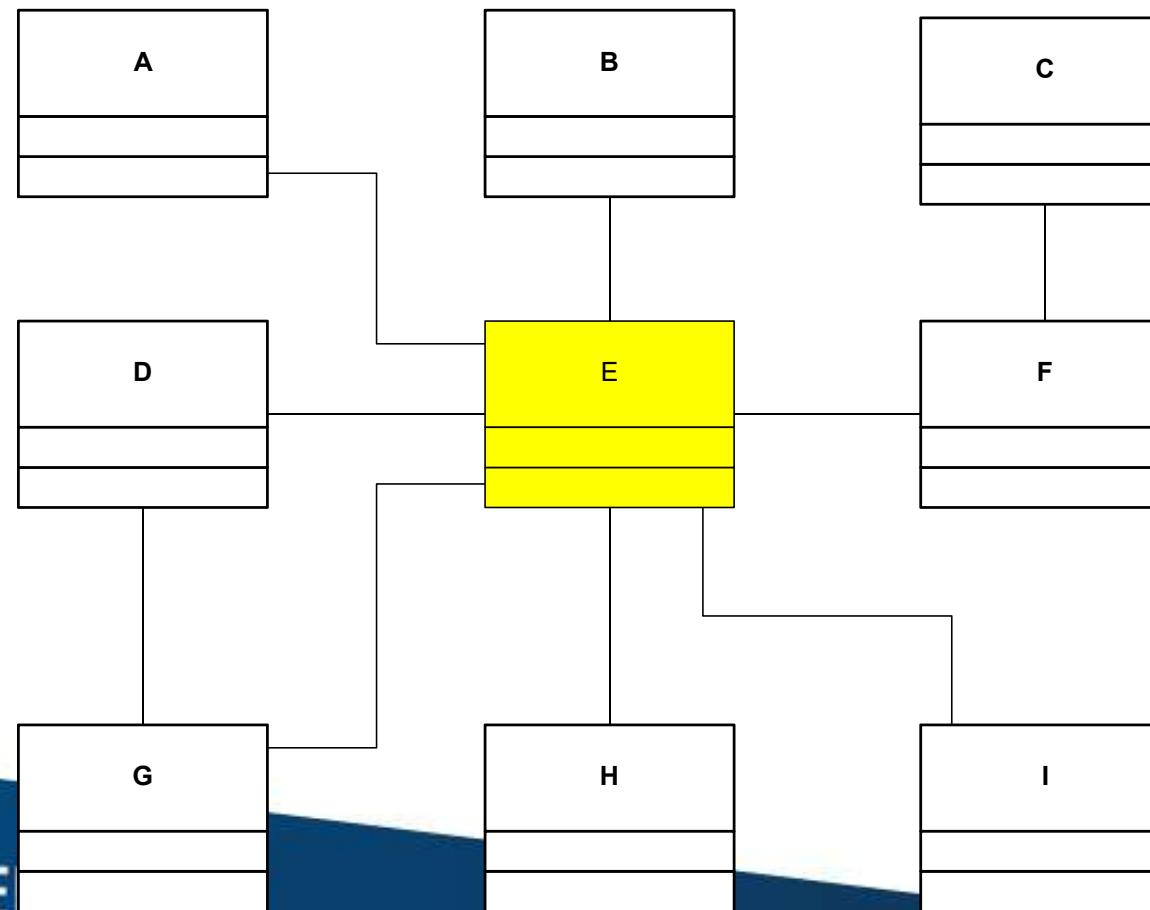


**PRESIDENCY
UNIVERSITY**



Private UNI Object-Oriented Systems Development

Tightly Coupled Object



PRESIDE
UNIVERSITY

Private University
Object-Oriented Systems Development

40
YEARS
OF EXCELLENCE
IN INNOVATION

Corollary 1- Uncoupled Design *with* *Less Information Content*

- The main goal here is to maximize objects (or software components) cohesiveness.



**PRESIDENCY
UNIVERSITY**

Private UNI Object-Oriented Systems Development



Corollary 2- Single Purpose

- Each class must have a purpose, as was explained in a previous topic.
- When you document a class, you should be able to easily explain its purpose in a sentence or two.



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Corollary 3- Large Number of *Simpler Classes, Reusability*

- A great benefit results from having a large number of simpler classes.
- The less specialized the classes are, the more likely they will be reused.



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Corollary 4.StrongMapping

- As the model progresses from analysis to implementation, more detail is added, but it remains essentially the same.
- A strong mapping links classes identified during analysis and classes designed during the design phase.



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Corollary 5. Standardization

- The concept of design patterns might provide a way for standardization by capturing the design knowledge, documenting it, and storing it in a repository that can be shared and reused in different applications.



**PRESIDENCY
UNIVERSITY**



Private University Object-Oriented Systems Development

Corollary 6. Designing with *Inheritance*

- Say we are developing an application for the government that manages the licensing procedure for a variety of regulated entities.
- Let us focus on just two types of entities: motor vehicles and restaurants.



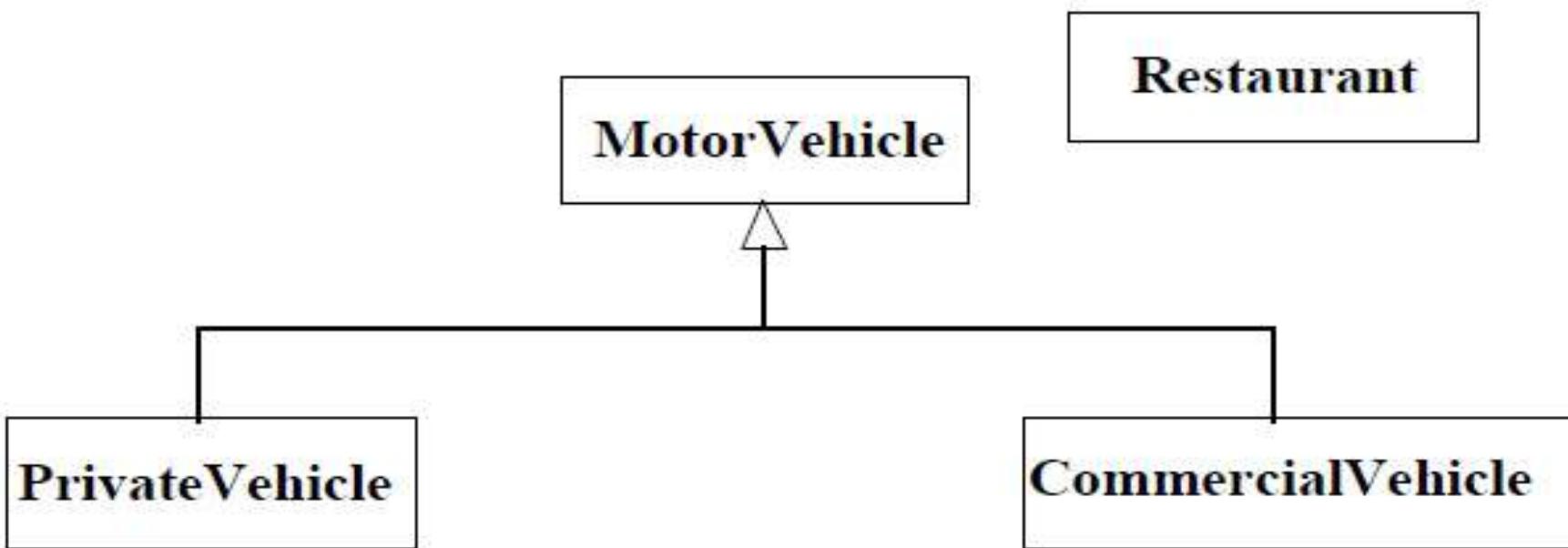
**PRESIDENCY
UNIVERSITY**



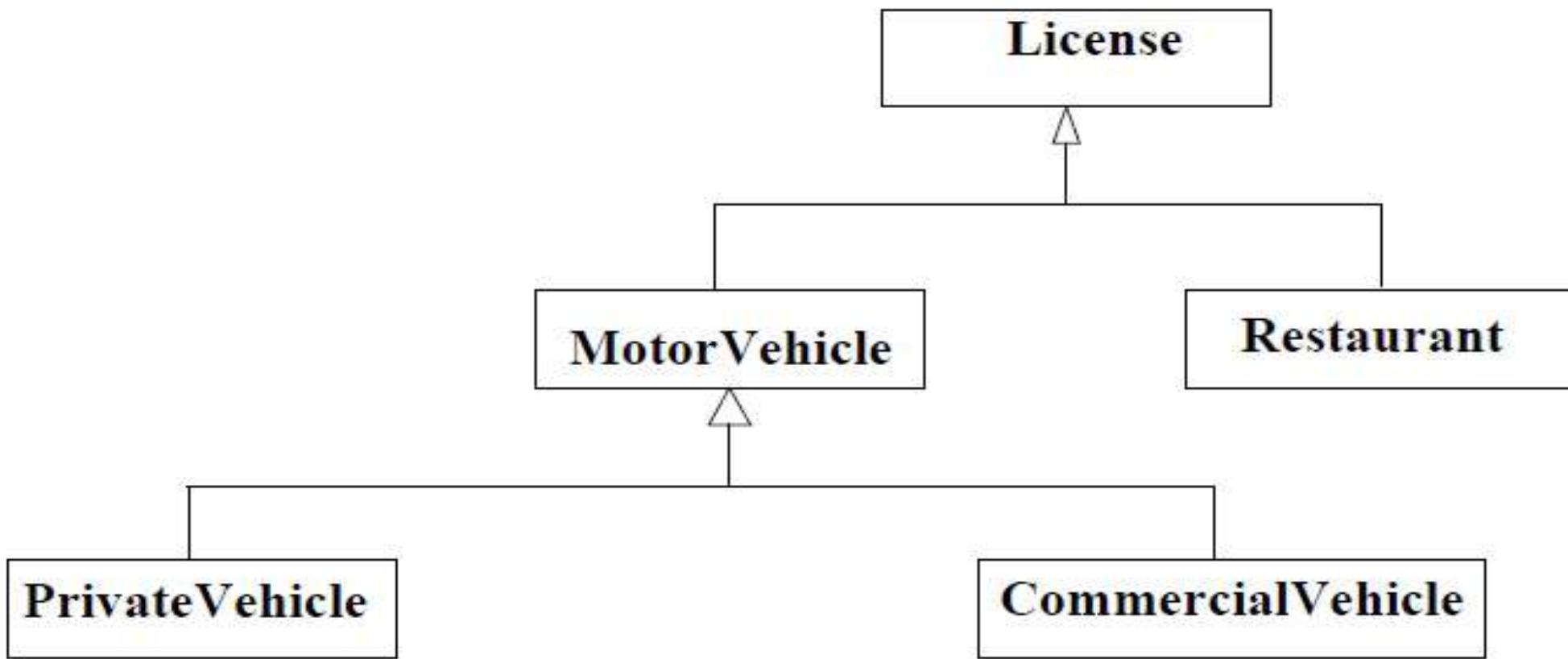
Private UN Object-Oriented Systems Development

Designing With Inheritance

(Con't)



Designing With Inheritance (Con't)



Designing With Inheritance *WeakFormal Class*

- MotorVehicle and Restaurant classes do not have much in common.
- For example, of what use is the gross weight of a diner or the address of a truck?

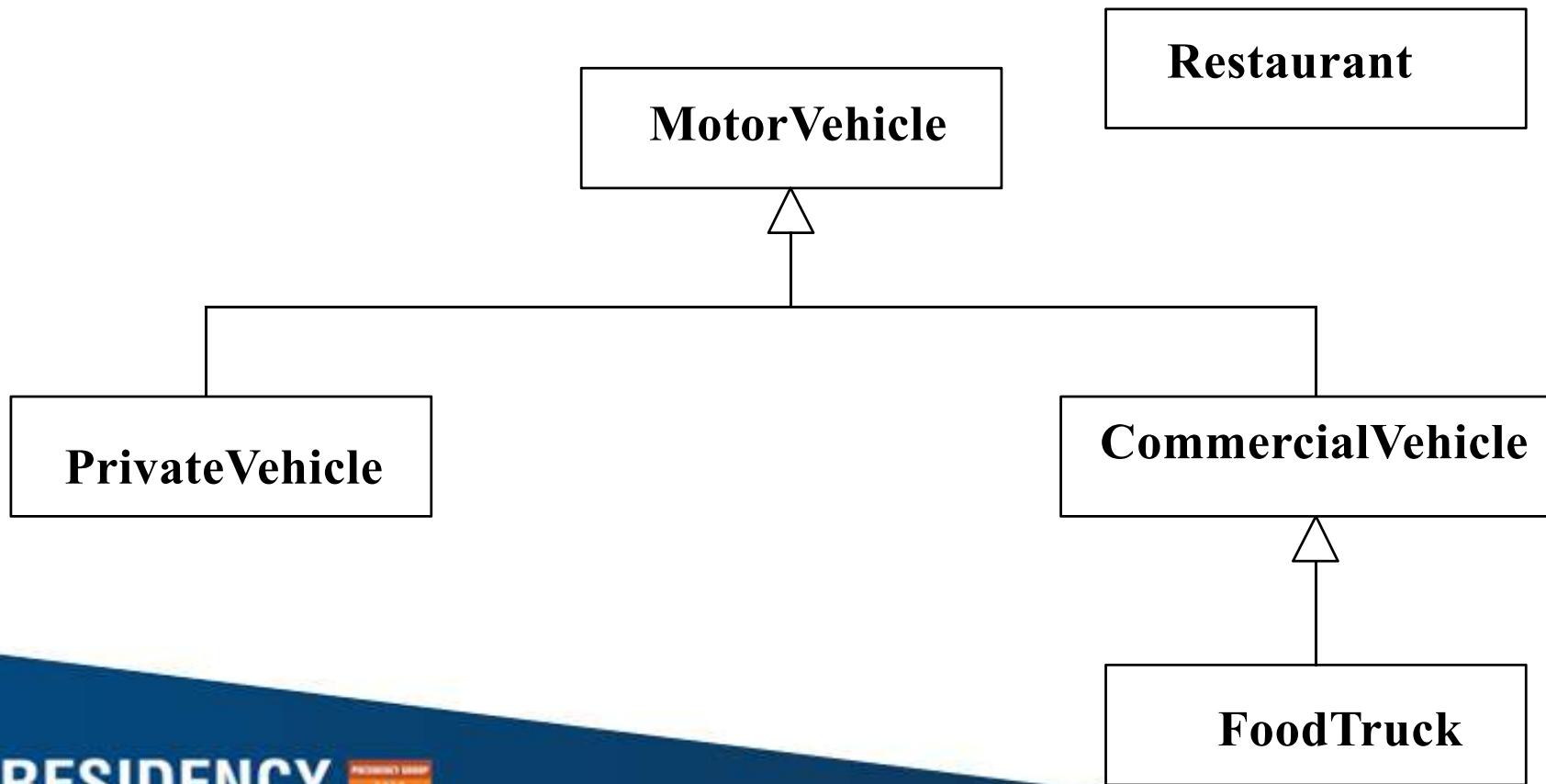


**PRESIDENCY
UNIVERSITY**

Private UN Object-Oriented Systems Development



Designing With Inheritance (*Con't*)



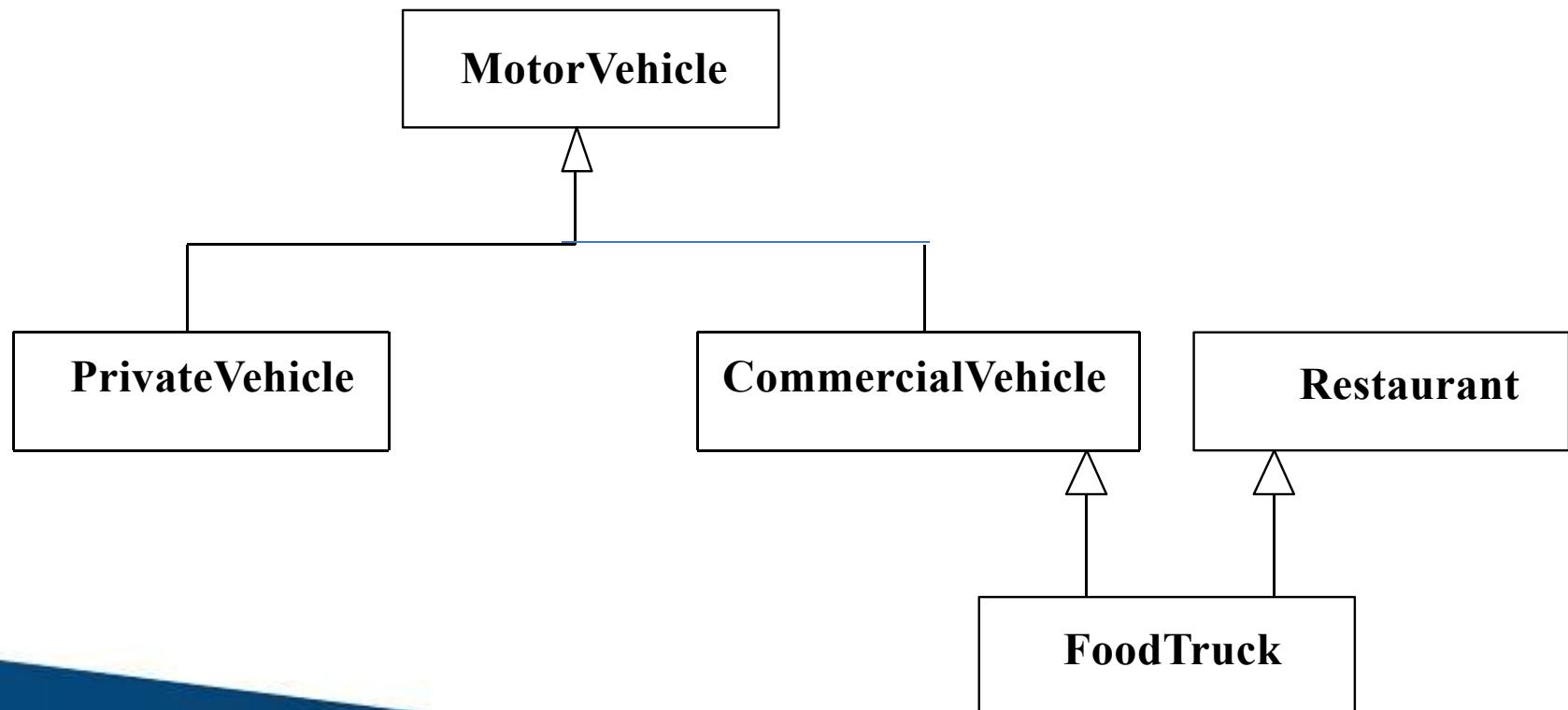
**PRESIDENCY
UNIVERSITY**



Object-Oriented Systems Development
? Irwin/ McGraw-Hill

Bahrami

Designing With Inheritance (*Con't*)

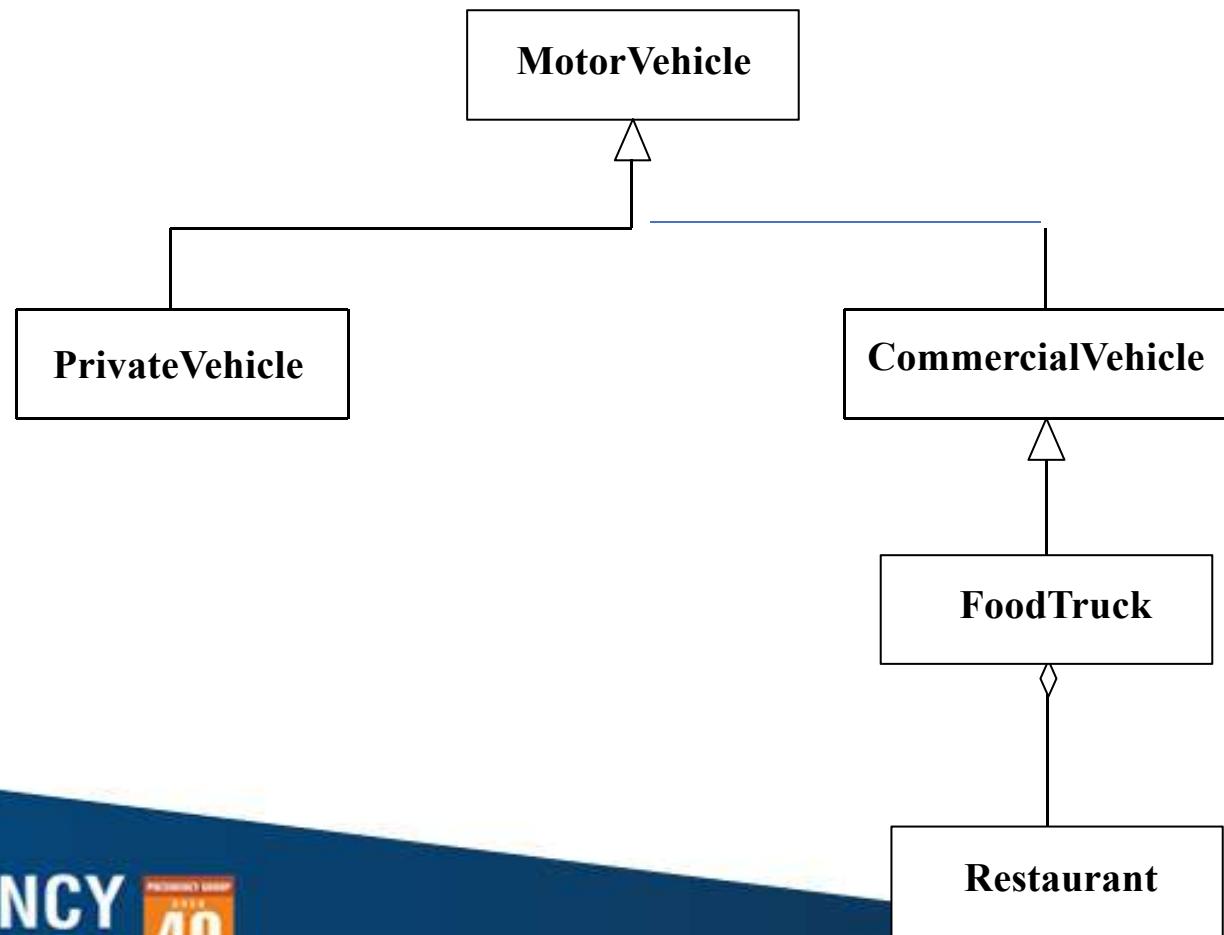


**PRESIDENCY
UNIVERSITY**



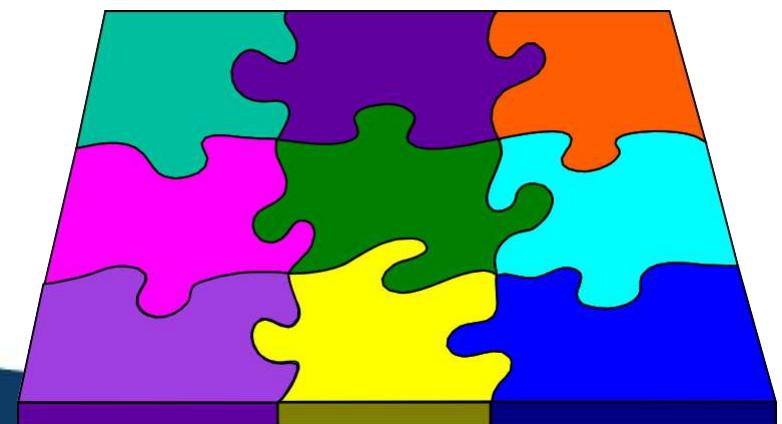
Private University Object-Oriented Systems Development

Achieving Multiple Inheritance *using Single Inheritance Approach*



Design Patterns

- Patterns provide a mechanism for capturing and describing commonly recurring design ideas that solve a general design problem.



**PRESIDENCY
UNIVERSITY**



<<abstract>>

Game

+initialize() : void
+startPlay() : void
+endPlay() : void
+play() : void

extends

Cricket

+initialize() : void
+startPlay() : void
+endPlay() : void
+play() : void

extends

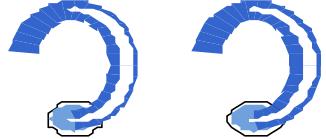
Football

+initialize() : void
+startPlay() : void
+endPlay() : void
+play() : void

uses

TemplatePatternDemo

+main() : void



Summary

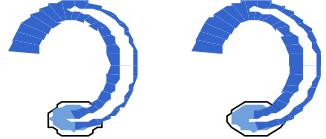
- We studied the object-oriented design process and axioms.
- The two design axioms are
 - Axiom 1. The independence axiom. Maintain the independence of components.
 - Axiom 2. *The information axiom. Minimize the information content of the design.*



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development



Summary (Con't)

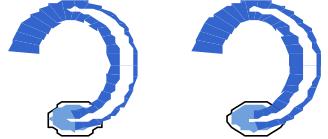
- The six design corollaries are
 - **Corollary 1. Uncoupled design with less information content.**
 - **Corollary 2. Single purpose.**
 - **Corollary 3. Large number of simple classes.**
 - **Corollary 4. Strong mapping.**
 - **Corollary 5. Standardization.**
 - **Corollary 6. Design with inheritance.**



**PRESIDENCY
UNIVERSITY**

Private University Object-Oriented Systems Development





Summary (Con't)

- We also studied the concept of design patterns, which allow systems to share knowledge about their design.



**PRESIDENCY
UNIVERSITY**

Private UNI Object-Oriented Systems Development



Designing Classes



**PRESIDENCY
UNIVERSITY**

Private UNI Object-Oriented Systems Development



Goals

- Designing classes.
- Designing protocols and class visibility.
- Defining attributes.
- Designing methods.
- Designing access layer objects.



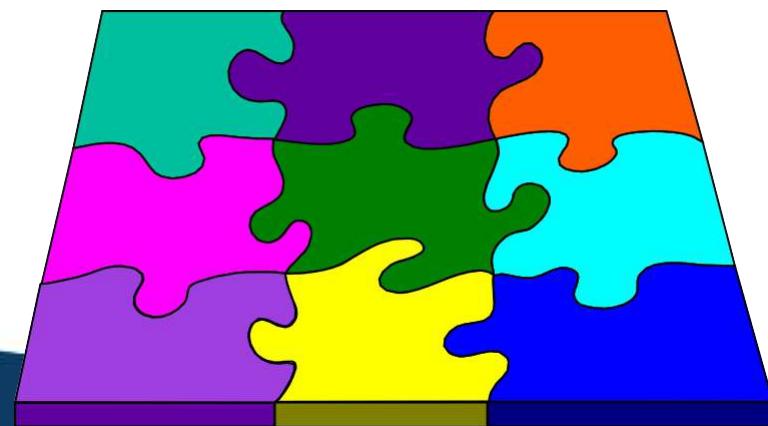
**PRESIDENCY
UNIVERSITY**



Private University Object-Oriented Systems Development

Object-Oriented Design *Philosophy*

- The first step in building an application should be to design a set of classes, each of which has a specific expertise and all of which can work together in useful ways.



**PRESIDENCY
UNIVERSITY**

Private UNI Object-Oriented Systems Development



- Class pen

{

Private int color;

Public char Company[10];

Public float Price;

Public void Write(){}

}

Pen p1,p2,p3;



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Class Visibility

- In designing methods or attributes for classes, you are confronted with two issues.
 - One is the *protocol*, or interface to the class operations and its visibility;
 - and how it should be implemented.



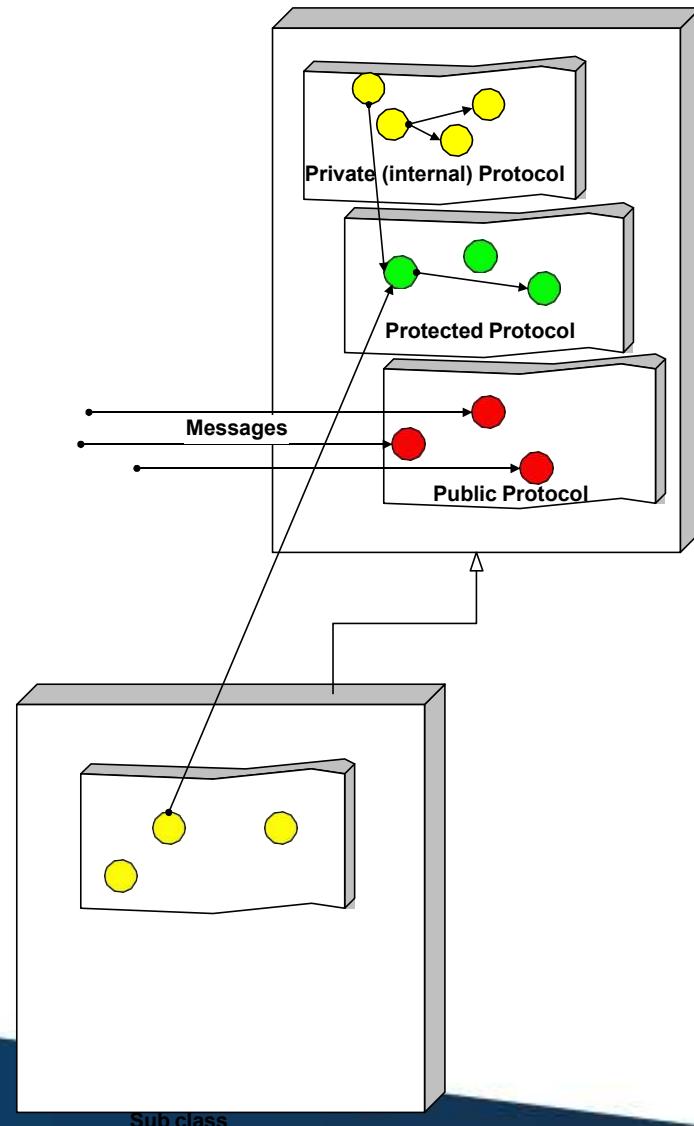
**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Class Visibility (Con't)

- Public protocols define the functionality and external messages of an object, while private protocols define the implementation of an object.



**PRESIDENCY
UNIVERSITY**



Object-Oriented Systems Development

Private Protocol (Visibility)

- A set of methods that are used only internally.
- Object messages to itself.
- Define the implementation of the object (Internal)
- **Issues are:** deciding what should be private.
 - What attributes
 - What methods



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Protected Protocol (Visibility)

- In a protected protocol, subclasses can use the method in addition to the class itself.
- In private protocols, only the class itself can use the method.



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Public Protocol (Visibility)

- Defines the functionality of the object
- Decide what should be public (External).



**PRESIDENCY
UNIVERSITY**



Private University Object-Oriented Systems Development

- Write using java class, a class for circle, with radius, x co ordinate and y co ordinate to be the attributes , calculate area, calculate perimeter and scale as methods in it.

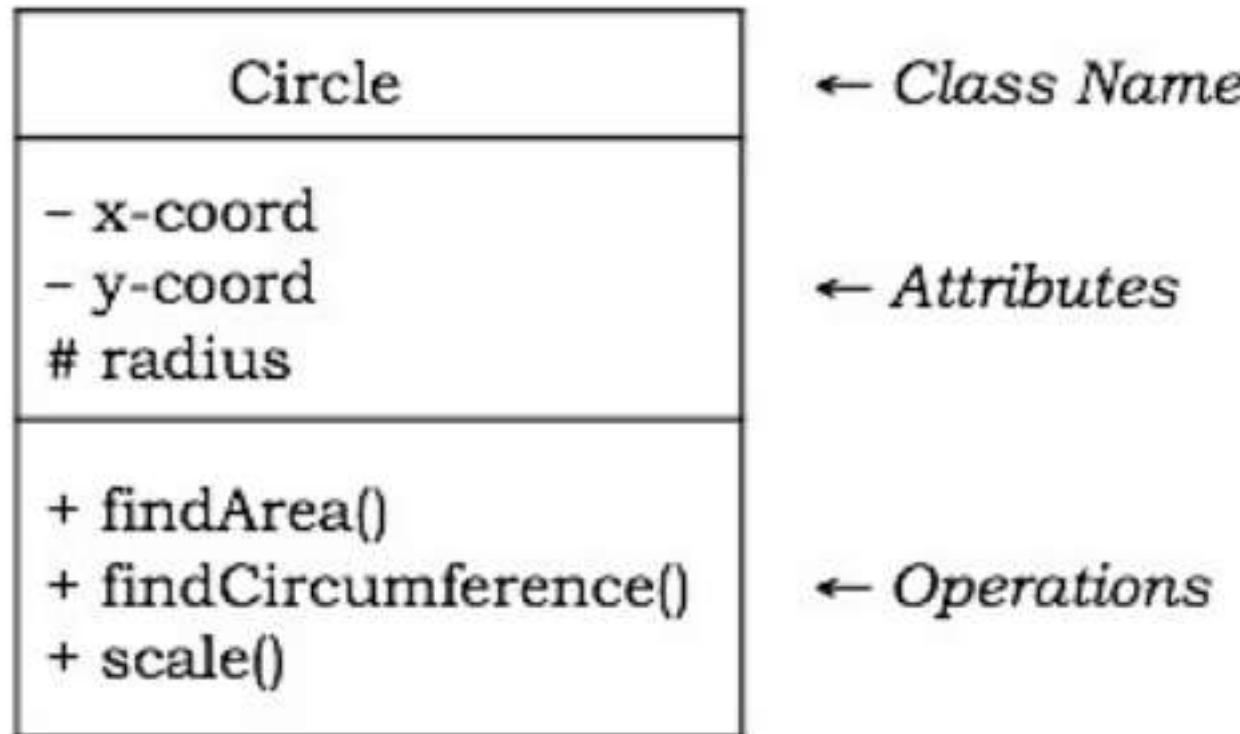


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- Write using UML notation, a class for circle, with radius, x co ordinate and y co ordinate to be the attributes , calculate area, calculate perimeter and scale as methods in it.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Guidelines for Designing *Protocols*

- Good design allows for polymorphism.
- Not all protocols should be public, again apply design axioms and corollaries.



**PRESIDENCY
UNIVERSITY**

Private UN Object-Oriented Systems Development



Guidelines for Designing *Protocols (Con't)*

- The following key questions must be answered:
 - What are the class interfaces and protocols?
 - What public (external) protocol will be used or what external messages must the system understand?



**PRESIDENCY
UNIVERSITY**



Private University Object-Oriented Systems Development

Questions (Con't)

- What private or protected (internal) protocol will be used or what internal messages or messages from a subclass must the system understand?



**PRESIDENCY
UNIVERSITY**



Private University Object-Oriented Systems Development

Attribute Types

- The three basic types of attributes are:
 - 1. Single-value attributes.
 - 2. Multiplicity or multivalue attributes.
 - 3. Reference to another object, or instance connection.



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Designing Methods and Protocols

- A class can provide several types of methods:
 - ***Constructor***. Method that creates instances (objects) of the class.
 - ***Destructor***. The method that destroys instances.
 - ***Conversion method***. The method that converts a value from one unit of measure to another.



**PRESIDENCY
UNIVERSITY**



Private UNI Object-Oriented Systems Development

Designing Methods and *Protocols* *(Con't)*

- Copy method.*** The method that copies the contents of one instance to another instance.
- Attribute set.*** The method that sets the values of one or more attributes.
- Attribute get.*** The method that returns the values of one or more attributes.

Designing Methods and *Protocols* *(Con't)*

- I/O methods.** The methods that provide or receive data to or from a device.
- Domain specific.** The method specific to the application.



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Five Rules For Identifying Bad *Design*

- **I.** If it looks messy then it's probably a bad design.



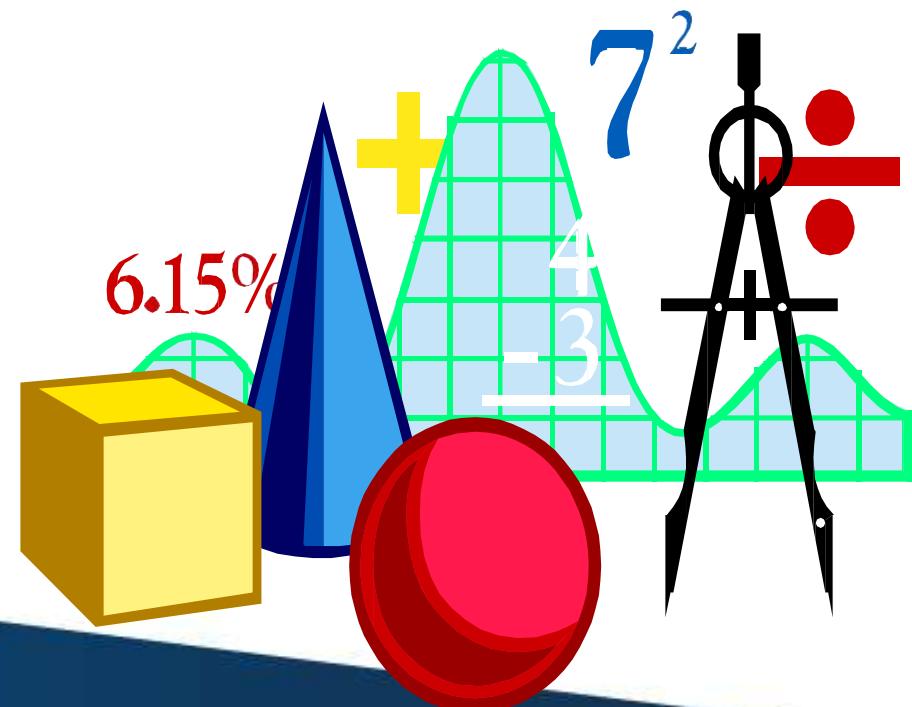
**PRESIDENCY
UNIVERSITY**

Private UNI Object-Oriented Systems Development



Five Rules For Identifying Bad Design (Con't)

- **II. If it is too complex then it's probably a bad design.**



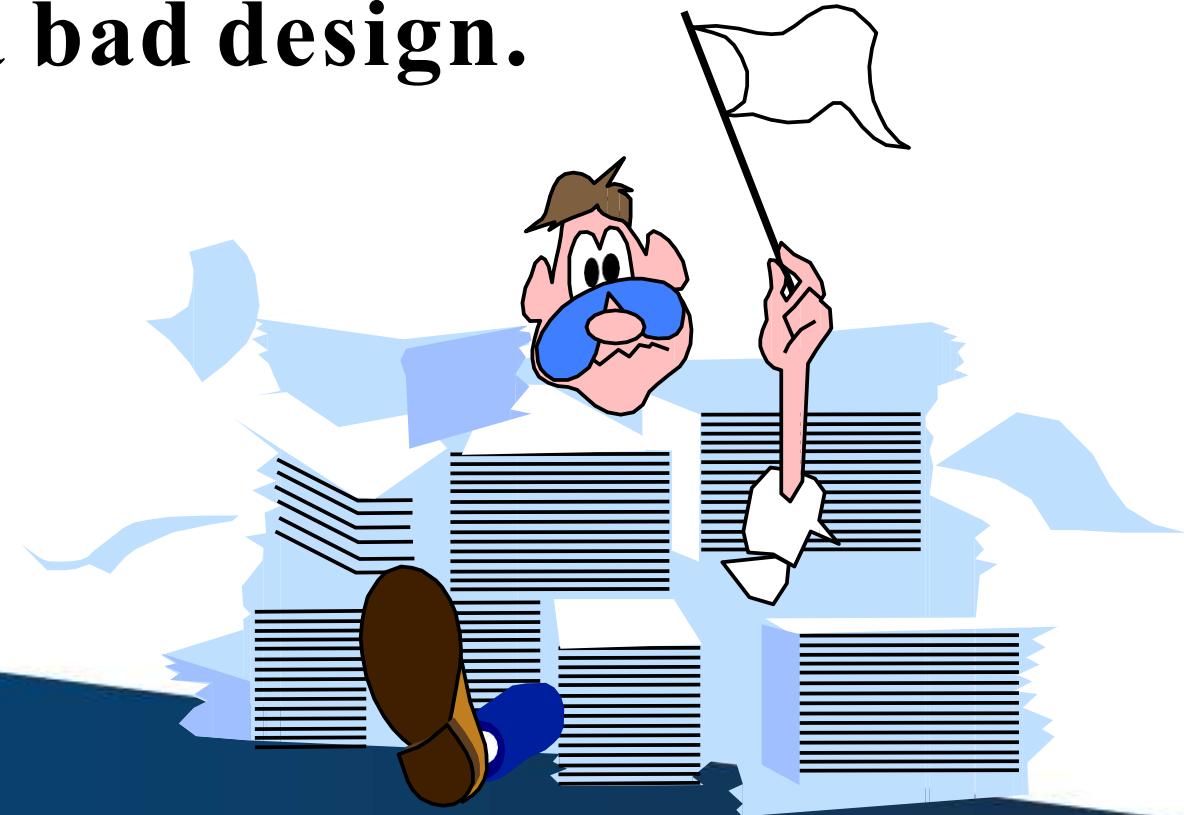
**PRESIDENCY
UNIVERSITY**

PRESIDENCY
UNIVERSITY
40
YEARS
OF EXCELLENCE, INNOVATION

Private UN Object-Oriented Systems Development

Five Rules For Identifying Bad Design (Con't)

- **III. If it is too big then it's probably a bad design.**



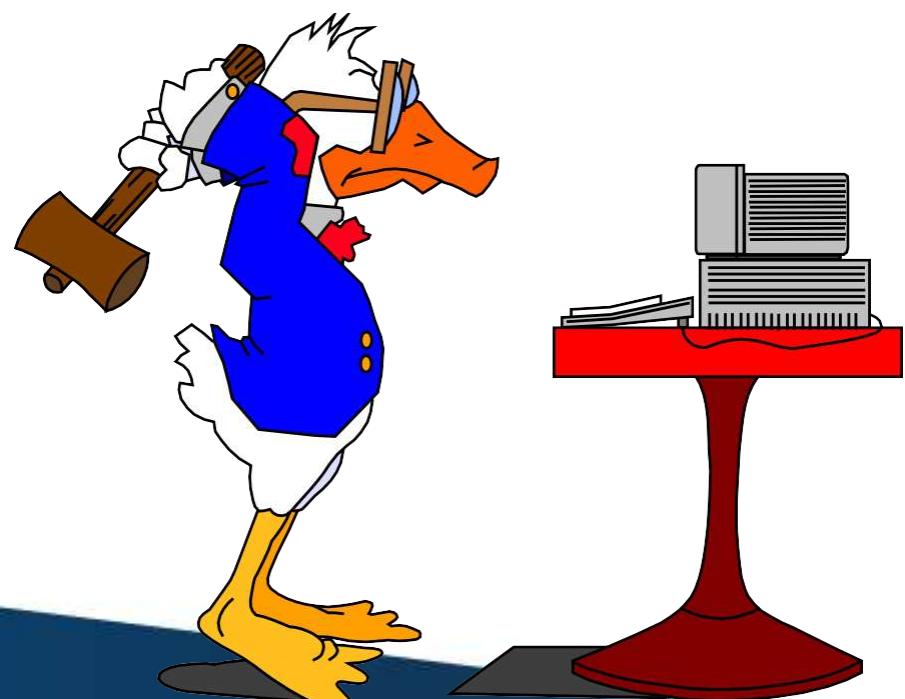
**PRESIDENCY
UNIVERSITY**



Private UNI Object-Oriented Systems Development

Five Rules For Identifying Bad *Design (Con't)*

- **IV. If people don't like it then it's probably a bad design.**



**PRESIDENCY
UNIVERSITY**



Private UNI Object-Oriented Systems Development

Five Rules For Identifying Bad Design (Con't)

- **V.** If it doesn't work then it's probably a bad design.



**PRESIDENCY
UNIVERSITY**

Private UNI Object-Oriented Systems Development



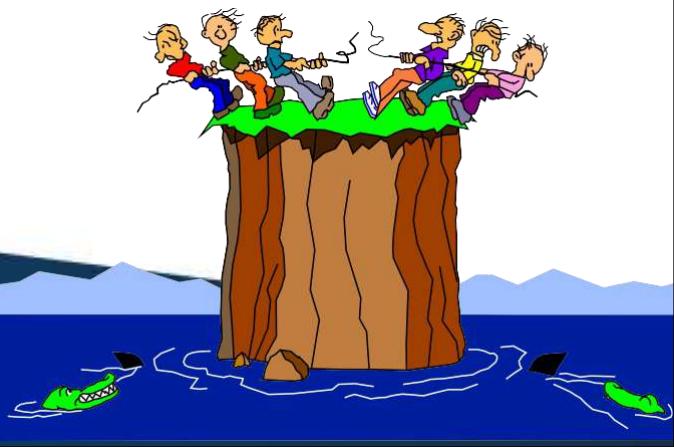
Avoiding Design Pitfalls

- Keep a careful eye on the class design and make sure that an object's role remains well defined.
- If an object loses focus, you need to modify the design.
- Apply Corollary 2 (single purpose).



**PRESIDENCY
UNIVERSITY**

Private UN Object-Oriented Systems Development



Avoiding Design Pitfalls (Con't)

- Move some functions into new classes that the object would use.
- Apply Corollary 1 (uncoupled design with less information content).
- Break up the class into two or more classes.
- Apply Corollary 3 (large number of simple classes).



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Designing Access Layer Classes

- The main idea behind creating an access layer is to create a set of classes that know how to communicate with data source, whether it be a file, relational database, mainframe, Internet, DCOM, or via ORB.
- The access classes must be able to translate any data-related requests from the business layer into the appropriate protocol for data access.



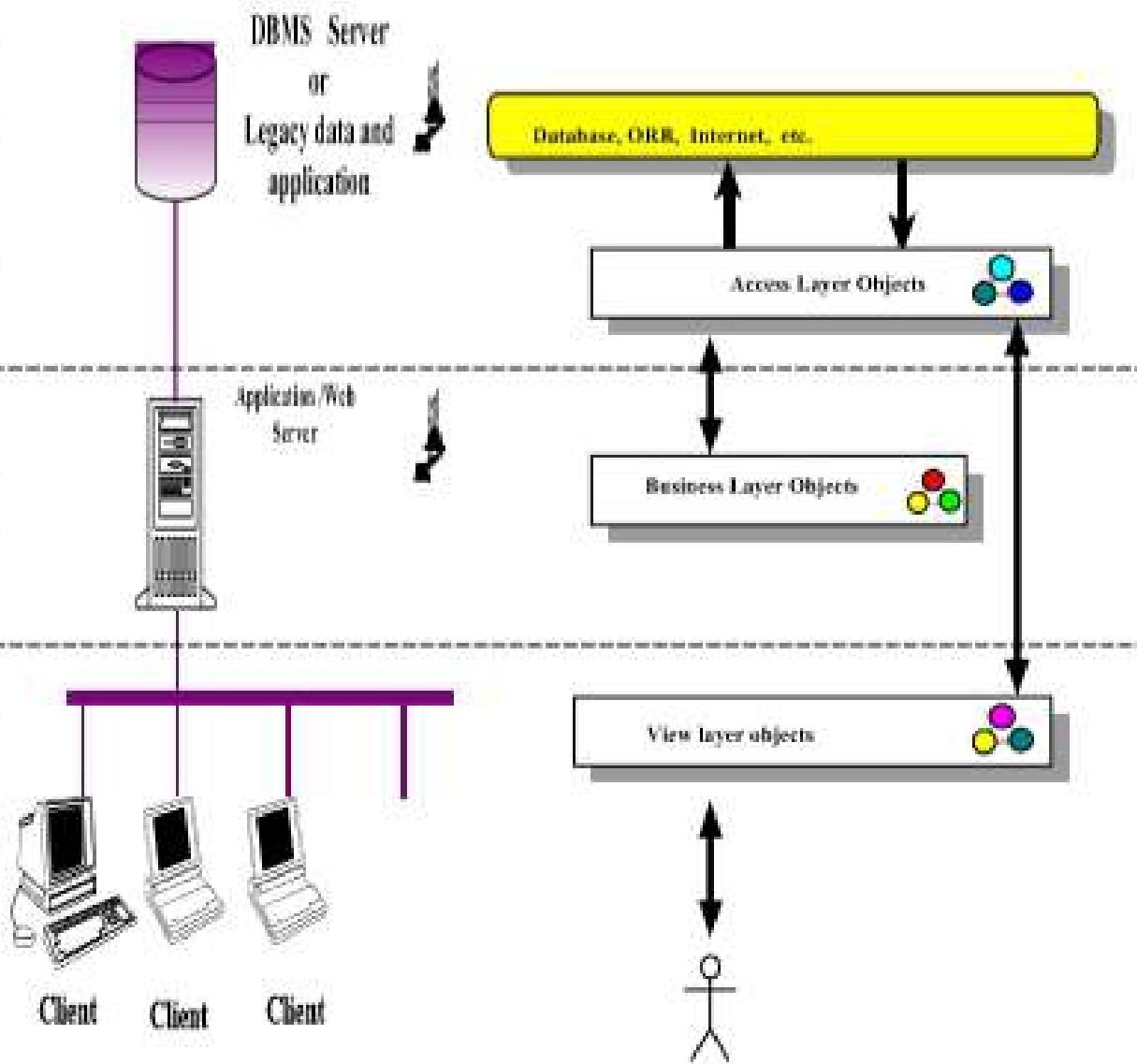
**PRESIDENCY
UNIVERSITY**

Private UNI Object-Oriented Systems Development



Access Layer Classes (Con't)

- The business layer objects and view layer objects should not directly access the database. Instead, they should consult with the access layer for all external system connectivity.



Benefits of Access Layer Classes

- Access layer classes provide easy migration to emerging distributed object technology, such as CORBA and DCOM.
- These classes should be able to address the (relatively) modest needs of two-tier client/server architectures as well as the difficult demands of fine-grained, peer-to-peer distributed object architectures.



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Process

- The access layer design process consists of these following activities:

1. If methods will be stored in a program then

For every business class identified, *determine if the class has persistent data.*

else

For every business class identified, *mirror the business class package.*

2. *Define relationships.* The same rule as applies among business class objects also applies among access classes.



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Process (Con't)

3. *Simplify classes and relationships.* The main goal here is to eliminate redundant or unnecessary classes or structures.
 1. *Redundant classes.* If you have more than one class that provides similar services, simply select one and eliminate the other(s).
 2. *Method classes.* Revisit the classes that consist of only one or two methods to see if they can be eliminated or combined with existing classes.
4. Iterate and refine.



**PRESIDENCY
UNIVERSITY**

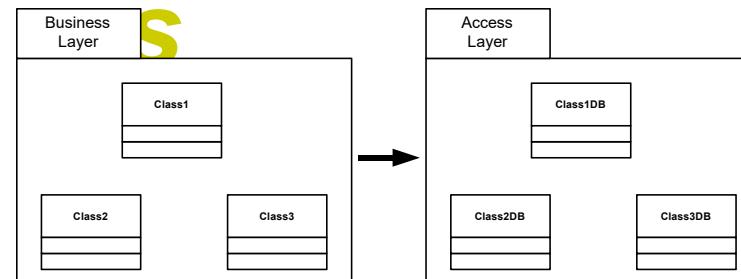
Private UNI Object-Oriented Systems Development



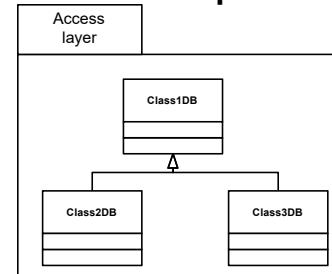
Process of Creating Access Layer

Classe

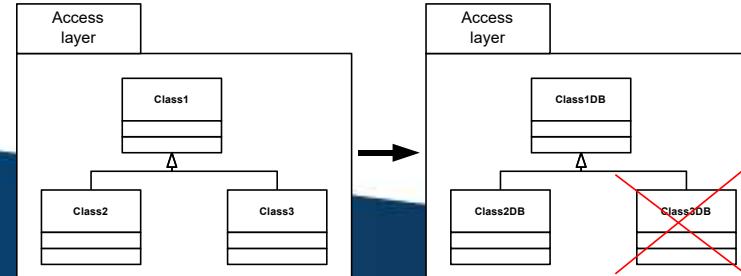
Step 1: Mirror business class package



Step 2: Define relationships



Step 3: Simplify classes and relationships



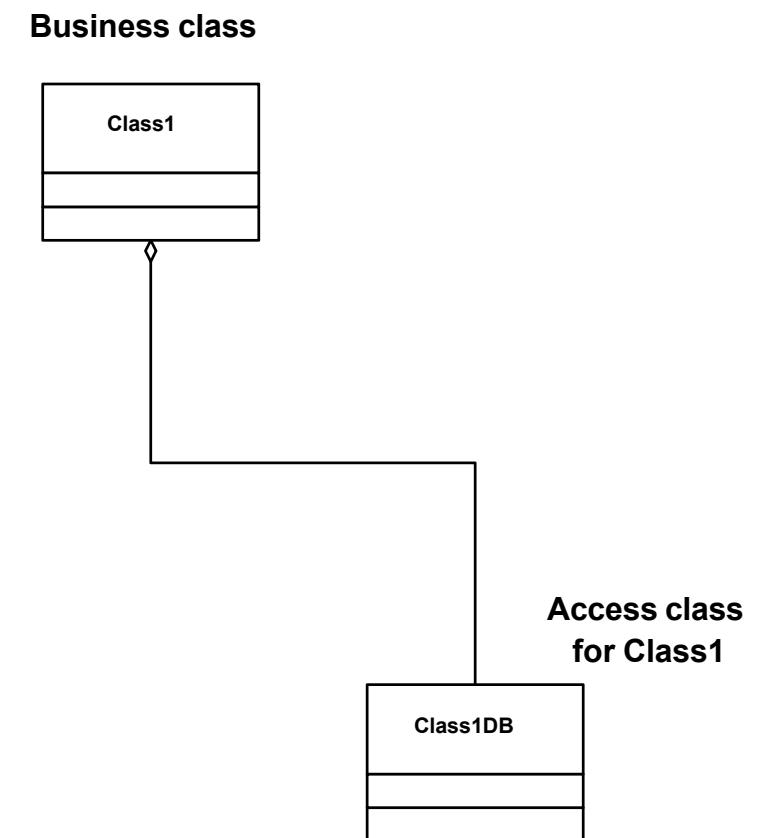
**PRESIDENCY
UNIVERSITY**



Private University Object-Oriented Systems Development

Access Layer Classes (Con't)

- The relation between a business class and its associated access class.



**PRESIDENCY
UNIVERSITY**

Private UN

GAIN MORE KNOWLEDGE
REACH GREATER HEIGHTS



Object-Oriented Systems Development

Summary

- This chapter concentrated on the first step of the object-oriented design process, which consists of applying the design axioms and corollaries to design classes, their attributes, methods, associations, structures, and protocols; then, iterating and refining.



**PRESIDENCY
UNIVERSITY**

Private UN Object-Oriented Systems Development



Summary (Con't)

- Object-oriented design is an iterative process.
- Designing is as much about discovery as construction.
- Do not be afraid to change a class design, based on experience gained, and do not be afraid to change it a second, third, or fourth time.



**PRESIDENCY
UNIVERSITY**



Private UNI Object-Oriented Systems Development

View Layer: Designing Interface Objects



**PRESIDENCY
UNIVERSITY**

Private University Object-Oriented Systems Development



Goals

- **Identifying View Classes**
- **Designing Interface Objects**
- **Guidelines to Graphical User Interface (GUI)**



**PRESIDENCY
UNIVERSITY**

Private UNI Object-Oriented Systems Development



*...The design of your software's *interface*,
more than anything else, affects how a user
interacts and therefore experiences your
application.*

Tandy Trower



**PRESIDENCY
UNIVERSITY**

Private UNI Object-Oriented Systems Development



Designing

View Layer Classes

The view layer classes are responsible for two major aspects of the applications:

- **Input-Responding to user interaction**
- **Output-Displaying business objects**



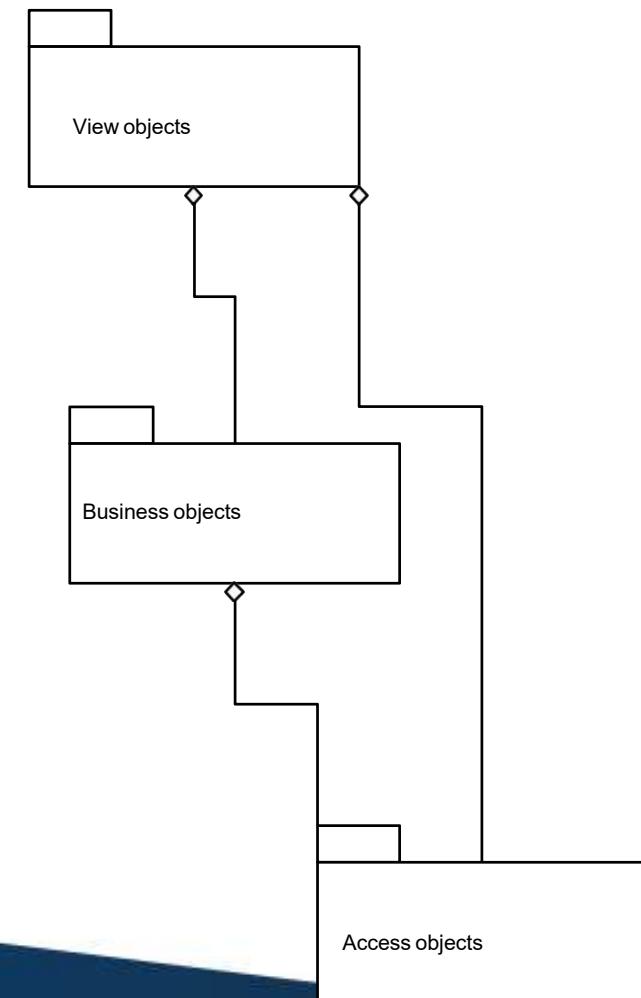
**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Relationships Among Business, Access and View Classes

- In some situations the view class can become a direct aggregate of the access object, as when designing a web interface that must communicate with application/Web server through access objects.



**PRESIDENCY
UNIVERSITY**



Private UNI Object-Oriented Systems Development

Designing

View Layer Classes *(Con't)*

- **Design of the view layer classes are divided into the following activities:**
 - **I. Macro Level UI Design Process- Identifying View Layer Objects.**
 - **II. Micro Level UI Design Activities.**
 - **III. Usability and User Satisfaction Testing.**
 - **IV. Refine and Iterate.**



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

View Layer Macro Level

1. For Every Class Identified

1. Determine If the Class Interacts With Human Actor: If **yes**, do next step **otherwise move to next class.**

1. Identified the View (Interface) Objects for The Class.

2. Define Relationships Among the View (Interface) Objects.

2. Iterate and refine.



**PRESIDENCY
UNIVERSITY**



Private UNI Object-Oriented Systems Development

View Layer Micro Level

- 1. For Every Interface Object Identified in the Macro UI Design Process.
 - 1.1 Apply Micro Level UI Design Rules and Corollaries to Develop the UI.

2. Iterate and refine.

Apply design rules and GUI guidelines to design the UI for the interface objects identified.



**PRESIDENCY
UNIVERSITY**

Private UNI Object-Oriented Systems Development



UI Design Rules

- Rule 1- Making the Interface Simple
- Rule 2- Making the Interface Transparent and Natural
- Rule 3- Allowing Users to Be in Control of the Software



**PRESIDENCY
UNIVERSITY**

Object-Oriented Systems Development
? Irwin/ McGraw-Hill



UI Design Rule 1

- Making the interface simple: application of **corollary 2.**
- KISS, Keep It Simple, Stupid.
- Simplicity is different from being simplistic.
- Making something simple requires a good deal of work and code.



**PRESIDENCY
UNIVERSITY**



Making The Interface Simple(Con't)

- **Every additional feature potentially affects performance, complexity, stability, maintenance, and support costs of an application.**



**PRESIDENCY
UNIVERSITY**

Private UNI Object-Oriented Systems Development



UI Design Rule 1 (Con't)

- A design problem is harder to fix after the release of a product because users may adapt, or even become dependent on, a peculiarity in the design.



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

UI Design Rule 2

- **Making the interface transparent and Natural: application of corollary 4.**
- **Corollary 4 implies that there should be strong mapping between the user's view of doing things and UI classes.**



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Making The Interface Natural

- **The user interface should be intuitive so users can anticipate what to do next by applying their previous knowledge of doing tasks without a computer.**



**PRESIDENCY
UNIVERSITY**

Private UN Object-Oriented Systems Development



Using Metaphors

- **Metaphors can assist the users to transfer their previous knowledge from their work environment to your application interface.**
- **For example, forms that users are accustomed to seeing.**



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

UI Design Rule 3

- Allowing users to be in control of the software: application of **corollary 1**.
- Users should always feel in control of the software, rather than feeling controlled by the
 - software.



**PRESIDENCY
UNIVERSITY**



Private University Object-Oriented Systems Development

Allowing Users Control of the Software

- **Some of the ways to put users in control are:**
 - Making the interface forgiving.
 - Making the interface visual.
 - Providing immediate feedback.
 - Avoiding Modes.
 - Making the interface consistent.



Making the Interface Forgiving

- **Users should be able to back up or undo their previous action.**
- **They should be able to explore without fear of causing an irreversible mistake.**



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Making the Interface Visual

- You should make your interface highly visual so users can see, rather than recall, how to proceed.
- Whenever possible, provide users with a list of items from which they can choose.



**PRESIDENCY
UNIVERSITY**

Private UN Object-Oriented Systems Development



Providing Immediate Feedback

- **Users should never press a key or select an action without receiving immediate visual feedback, audible feedback, or both.**



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Avoiding Modes

- **Users are in a mode whenever they must cancel what they are doing before they can do something else.**
- **Modes force users to focus on the way an application works, instead of on the task they want to complete.**



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Can Modes be useful?

Yes, however:

- You should make modes an exception and limit their use.
- Whenever users are in a mode, you should make it obvious by providing good visual cues.
- The method for ending the mode should be easy to learn and remember.



**PRESIDENCY
UNIVERSITY**

Private UN Object-Oriented Systems Development



Modes can be useful (Con't)

These are some of the modes that can be used in the user interface.

- **Modal Dialog**
- **Spring-Loaded Modes**
- **Tool-Driven Modes**



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Making the Interface Consistent

- User Interfaces should be consistent throughout the applications.
- For example, keeping button locations consistent make users feel in control.



**PRESIDENCY
UNIVERSITY**



Private University Object-Oriented Systems Development

Purpose of a User Interface

- **Data Entry Windows:** Provide access to data that users can retrieve, display, and change in the application.
- **Dialog Boxes:** Display status information or ask users to supply information.
- **Application Windows (Main Windows):** Contain an entire application that users can launch.



**PRESIDIUM
UNIVERSITY**



Private UN Object-Oriented Systems Development

Guidelines For Designing Data *Entry Windows*

- You can use an existing paper form such as a printed invoice form as the starting point for your design.



**PRESIDENCY
UNIVERSITY**

Private UNI Object-Oriented Systems Development



Guidelines For Designing Data *Entry Windows* (*Con't*)

If the printed form contains too much information to fit on a screen:

- **Use main window with optional smaller Windows that users can display on demand, or**
- **Use a window with multiple pages.**



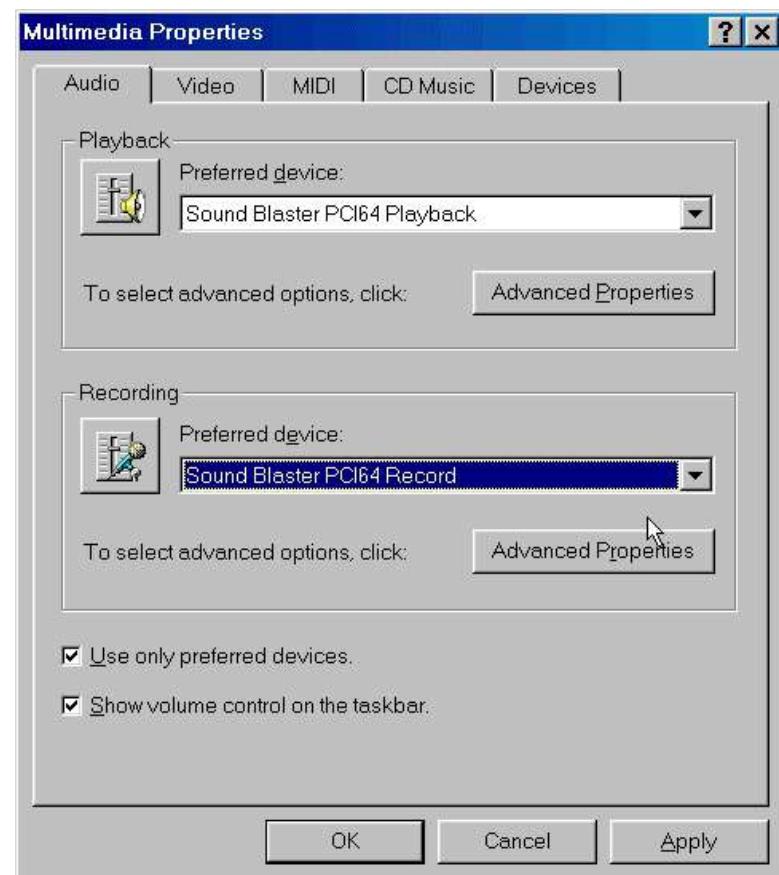
**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Guidelines For Designing Data Entry Windows (*Con't*)

- An example of a dialog box with multiple pages in the Microsoft multimedia setup.



**PRESIDENCY
UNIVERSITY**



Object-Oriented Systems Development

Guidelines For Designing Data *Entry Windows (Con't)*

Users scan a screen in the same way they read a page of a book, from left to right, and top to bottom.



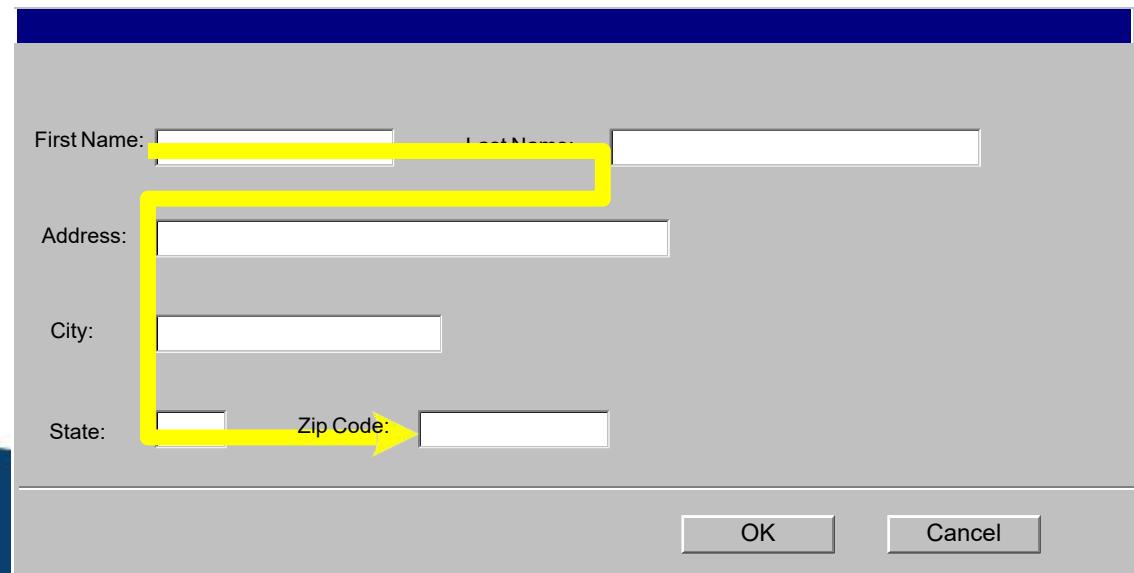
**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Guidelines For Designing Data *Entry Windows* (*Con't*)

- Orient the controls in the dialog box in the direction people read.
- In the Western world this usually means left
- to right, top to bottom.



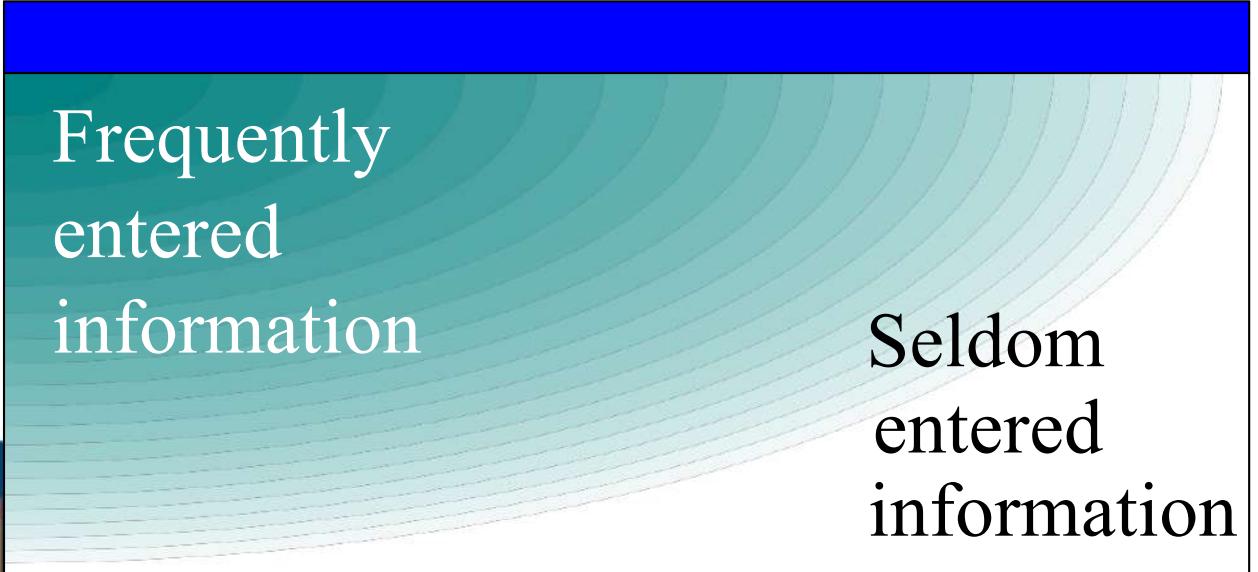
**PRESIDENCY
UNIVERSITY**



Guidelines For Designing Data

Entry Windows (Con't)

- Required information should be put toward the top and left side of the form, entering optional or seldom entered information toward the bottom.



Frequently
entered
information

Seldom
entered
information



PRESIDENCY
UNIVERSITY

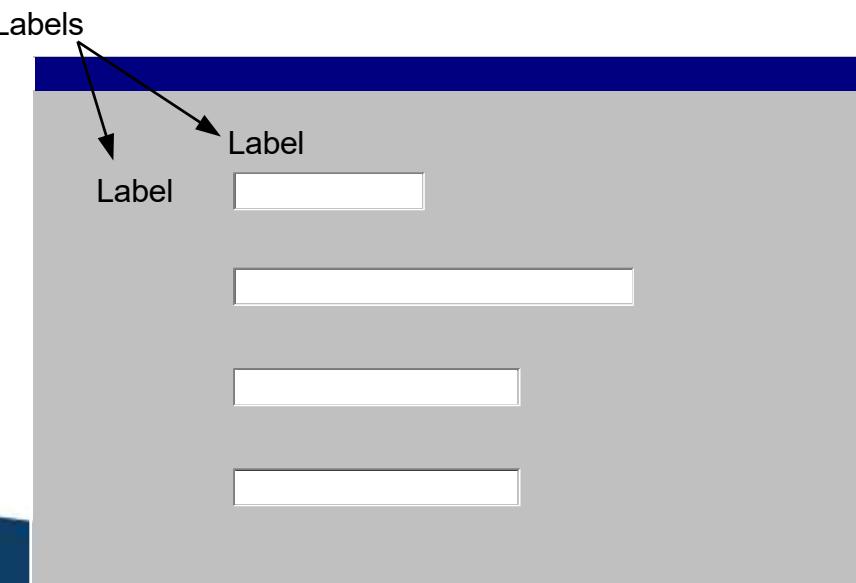


Private UN Object-Oriented Systems Development

Guidelines For Designing Data *Entry Windows* (*Con't*)

- Place text labels to the left of text box controls, align the height of the text with text displayed in the textbox.

Possible locations for text



**PRESIDENCY
UNIVERSITY**

Private UNI Object-Oriented Systems Development



Guidelines For Designing Dialog Boxes

- If the dialog box is for an error message, use the following guidelines:
 - Your error message should be positive.
 - For example instead of displaying “**You have typed an illegal date format,**” display this message “**Enter date format mm/dd/yyyy.**”



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

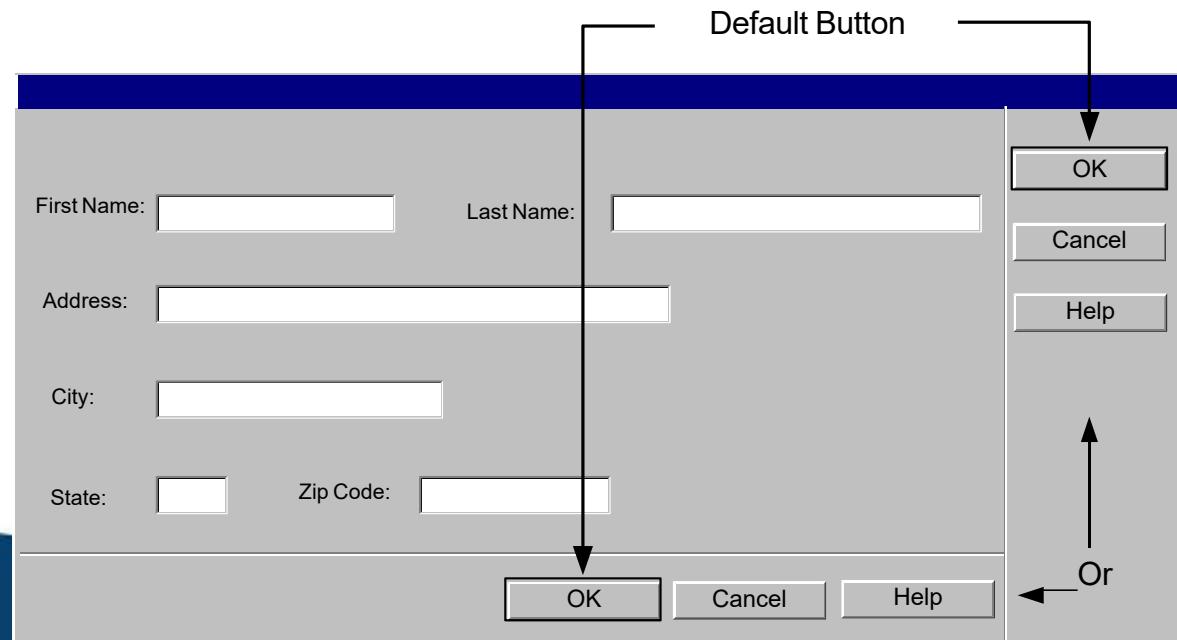
Guidelines For Designing Dialog Boxes(*Con't*)

- Your error message should be constructive, brief and meaningful.
- For example, avoid messages such as “**You should know better! Use the OK button**”
- instead display “**Press the Undo button and try again.**”



Guidelines For The Command Buttons Layout

- Arrange the command buttons either along the upper-right border of the form or dialog box or lined up across the bottom.



PRESIDENCY
UNIVERSITY



Private University Object-Oriented Systems Development

Buttons Layout(Con't)

- **Positioning buttons on the left or center is popular in Web interfaces.**

The screenshot shows a website layout with a vertical sidebar on the left containing links: home, what's new, corporate news and information, our businesses, news bureau, career opportunities, and FAQ. To the right, the main content area features a large title "Our Businesses" in a stylized font. Below it is a grid of twelve square icons, each representing a business unit: aviation (airplane), BROADCASTING (microphone), business & economics (woman at desk), COMMUNICATION (person on phone), COMPUTERS (computer monitor), CONSTRUCTION (construction worker), Consumer Products (remote control), Education (graduation student), FINANCIAL MARKETS (stock exchange), Government Solutions (building with American flag), Industry (hook), and Medical and Professional (stack of books). The McGraw-Hill logo is visible in the top right corner.

[Aviation](#) | [Broadcasting](#) | [Business & Economics](#) | [Computers/Communication](#) | [Construction](#) | [Consumer Products](#) | [Education](#)
[Financial Markets](#) | [Government Solutions](#) | [Industry](#) | [Medical & Professional](#)

[Home](#) | [What's New](#) | [Corporate News and Information](#) | [Our Businesses](#) | [Privacy Policy](#)
[Books](#) | [Career Opportunities](#) | [Directory](#) | [News Bureau](#) | [Contact Us](#) | [Terms of Use](#)

Copyright © 1998 The McGraw-Hill Companies, Inc.

Guidelines For Designing *Application Windows*

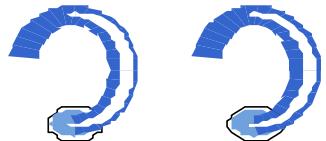
- A typical application window consists of a frame (or border) which defines its extent:
- title bar
- scroll bars
- menu bars,
- toolbars, and
- status bars.



**PRESIDENCY
UNIVERSITY**

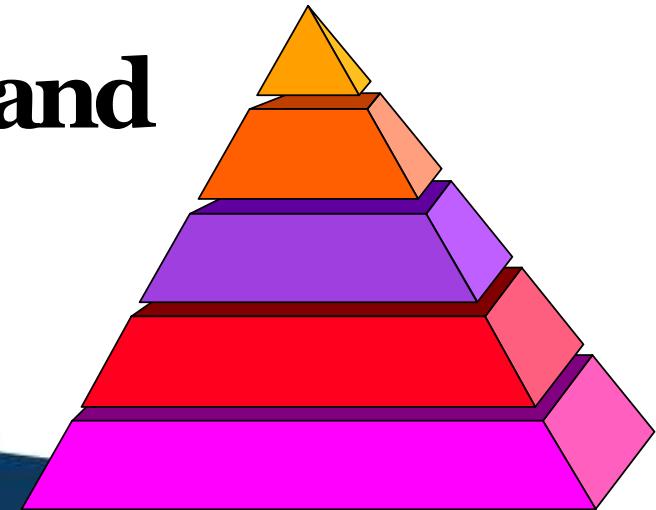


Private UNI Object-Oriented Systems Development



Guidelines For Using Colors

- Use identical or similar colors to indicate related information.
- Use different colors to distinguish groups of information from each other.
- For example, checkout and in-stock tapes could appear in different colors.



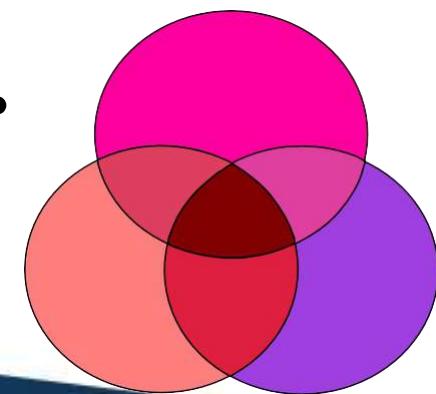
PRESIDENCY
UNIVERSITY

Private UNI Object-Oriented Systems Development



Guidelines For Using Colors (*Con't*)

- **For an object background, use a contrasting but complementary color.**
- **For example, in an entry field, make sure that the background color contrasts with the data color.**



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Guidelines For Using Colors (*Con't*)

- Use bright colors to call attention to certain elements on the screen.
- Use dim colors to make other elements less noticeable.
- For example, you might want to display the required field in a brighter color than optional fields.



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Guidelines For Using Colors (*Con't*)

- **Use colors consistently within each window and among all Windows in your application.**
- **For example the colors for Pushbuttons should be the same throughout.**



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Guidelines For Using Colors (*Con't*)

- **Using too many colors can be visually distracting, and will make your application less interesting.**

Guidelines For Using Colors (*Con't*)

- **Allow the user to modify the color configuration of your application.**



**PRESIDENCY
UNIVERSITY**



Private University Object-Oriented Systems Development

Guidelines For Using Fonts

- **Use commonly installed fonts, not specialized fonts that users might not have on their machines.**
- **Use bold for control labels so they will remain legible when the object is dimmed.**



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Guidelines For Using Fonts (Con't)

- **Use fonts consistently within each form and among all forms in your application.**
- **For example, the fonts for check box controls should be the same throughout.**
- **Consistency is reassuring to users, and psychologically makes users feel in control.**



**PRESIDENCY
UNIVERSITY**



Guidelines For Using Fonts (Con't)

- **Using too many font styles, sizes and colors can be visually distracting and should be avoided.**



**PRESIDENCY
UNIVERSITY**



Private UNI Object-Oriented Systems Development

Prototyping the User Interface

- **Rapid prototyping encourages the incremental development approach, “grow, don’t build.”**



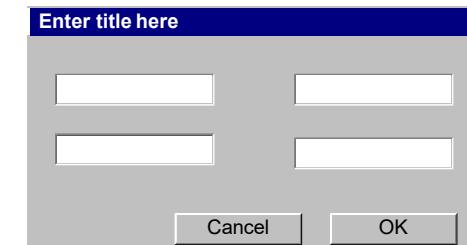
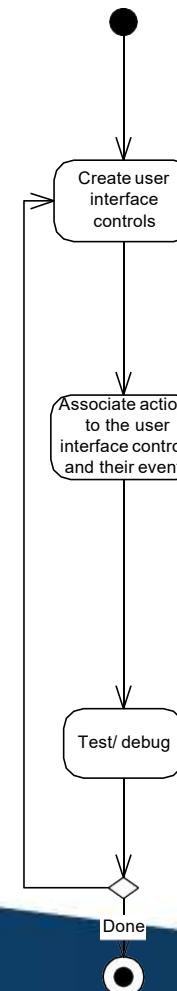
**PRESIDENCY
UNIVERSITY**

Private UN Object-Oriented Systems Development

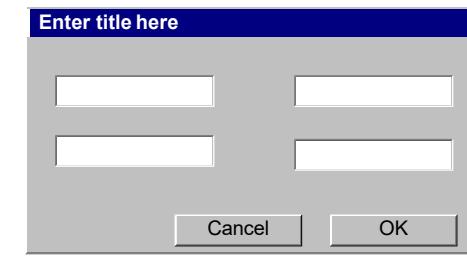


Three General Steps

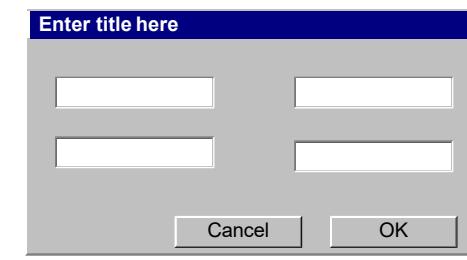
- **1. Create the user interface objects visually.**
- **2. Link or assign the appropriate behaviors or actions to these user interface objects and the ir events.**
- **3. Test, debug, then add more by going back to step 1.**



Create the forms and controls



Add actions



Test the UI

Make Users Feel in Charge

- Instead of using leading phrases like, "**we could do this ...**" or "**It would be easier if we ...**"
- Choose phrases that give the user the feeling that he/she is in charge:
 - “Do you think that if we did ... it would make it easier for the users?”**
 - “Do users ever complain about ...? We could add .. to make it easier.”**



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development

Summary

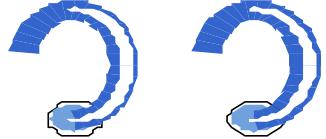
- **The main goal of UI is to display and obtain information you need in an accessible, efficient manner.**
- **The design of your software's interface, more than anything else, affects how a user interacts and therefore experiences your application.**



**PRESIDENCY
UNIVERSITY**



Private UN Object-Oriented Systems Development



Summary (Con't)

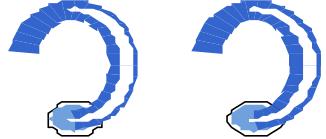
- **UI must provide users with the information they need and clearly tell them what they need to successfully complete a task.**
- **A well-designed UI has visual appeal that motivates users to use your application.**
- **UI should use limited screen space efficiently.**



**PRESIDENCY
UNIVERSITY**



Private UNI Object-Oriented Systems Development



Summary (Con't)

- **Designing View layer classes consists of the following steps:**

I. Macro Level UI Design Process- Identifying View Layer Objects

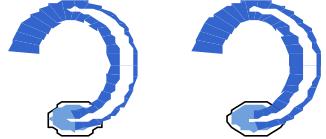
II. Micro Level UI Design Activities

II.1 Designing the View Layer Objects by applying Design Axioms and corollaries .

II. 2 Prototyping the View Layer Interface.

III. Usability and User Satisfaction Testing

IV. Refine and Iterate



Summary (Con't)

- Guidelines are not a standalone tool, and they cannot substitute for effective evaluation and iterative refinement within a design.
- However, they can provide helpful advice during the design process.



**PRESIDENCY
UNIVERSITY**

Private UNI Object-Oriented Systems Development



UNIT-4

UML FOR MODELING



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- **MODULE-4**

- **COURSE OUTCOMES:**

Apply UML Tools for Real World Applications.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- **OBJECT ORIENTED UML MODELING**

**Static and Dynamic Modeling-Unified Modeling Language -UML diagrams:
Class diagrams-Use case Diagram-UML Dynamic modeling: Interaction
diagram, Sequence diagram, Collaboration diagram, State-chart diagram,
Activity diagram, Experimental diagram-CASE STUDY using software
tools for UML(Star UML, Argo UML, Visio, Umbrello)**

Static and Dynamic Modeling-Unified Modeling Language



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Specific objectives

1. Why modeling?
2. What is UML and why we use UML?
3. List the UML diagrams.
4. Describe use-case diagram with examples(Self-Study)



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Modeling

- A model is an **abstract representation** of a system
- Constructed to understand the **system prior to building** or modifying it
- Several different models
 1. Use case model
 2. Domain object model
 3. Analysis object model
 4. Implementation model
 5. Test model

Static and Dynamic model

Static model

- snapshot of a system's parameters at rest
- represent the static aspect of a system
- Absence of change in data over time

Example: Class diagram

Dynamic model

- ✓ Behaviour of a system over time
- ✓ Dynamic relationships show how the business objects interact to perform tasks.

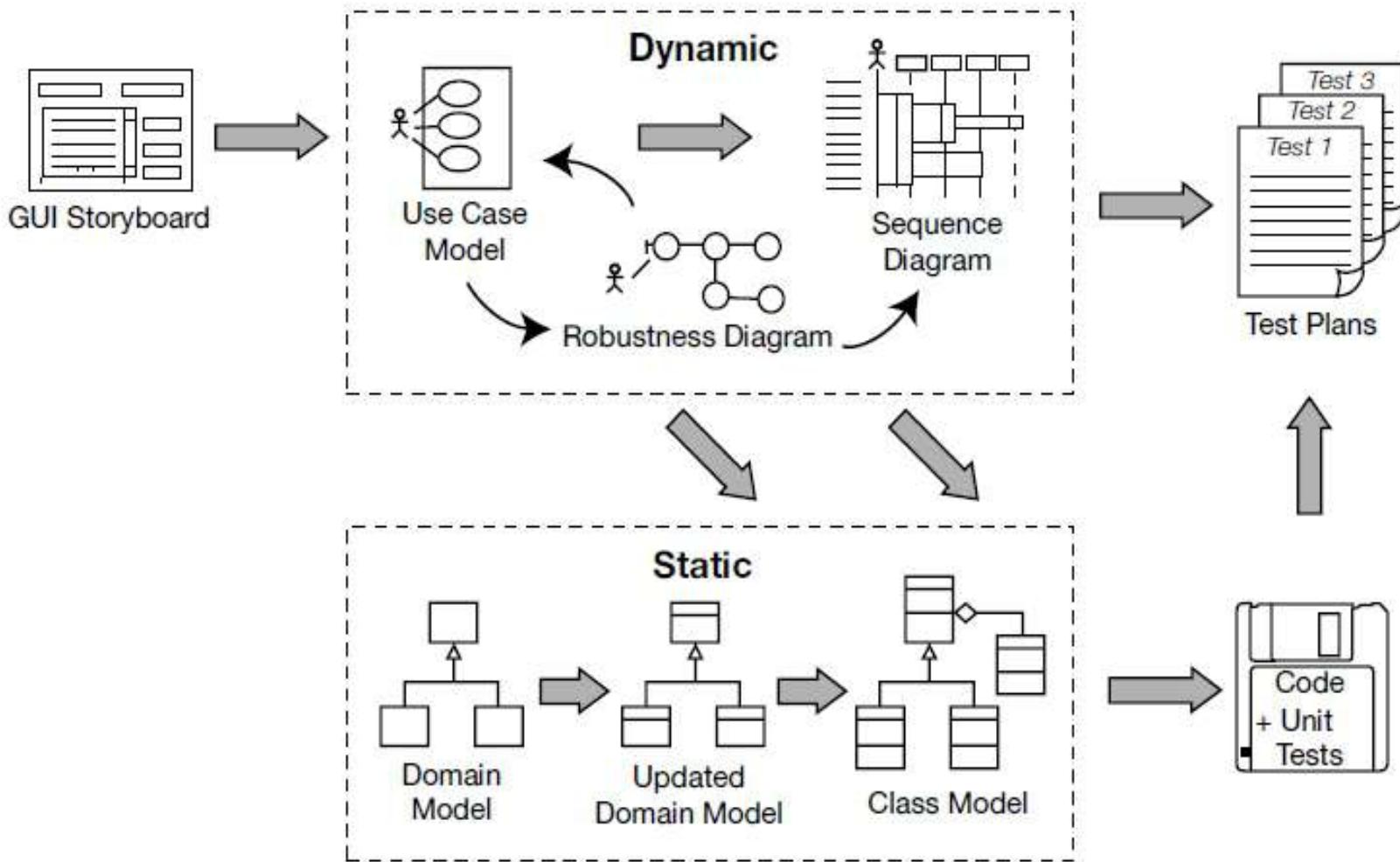
Example: Interaction diagram



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





**PRESIDENCY
UNIVERSITY**



Private University Estd. in Karnataka State by Act No. 41 of 2013

Static vs. Dynamic Design



- ❖ Static design describes code structure and object relations
 - Class relations
 - Objects at design time
 - Doesn't change
- ❖ Dynamic design shows communication between objects
 - Similarity to class relations
 - Can follow sequences of events
 - May change depending upon execution scenario
 - Called Object Diagrams



Why modeling?

- Building a model for a system prior to its construction is as essential as having a **blueprint** for a large building
- Modeling language must include
 - ✓ Model elements → [fundamental modeling concepts and semantics]
 - ✓ Notation → [visual rendering of model elements]
 - ✓ Guidelines → [expression of usage within the trade]
- Benefits
 - ✓ Clarity
 - ✓ Familiarity
 - ✓ Maintenance
 - ✓ Simplification

What is UML?

- UML → “Unified Modeling Language”
- Language: express idea, not a methodology
- Modeling: Describing a software system at a high level of abstraction
- Unified: UML has become a world standard
Object Management Group (OMG):
www.omg.org

What is UML ?

- It is a industry-standard graphical language for specifying, visualizing, constructing, and documenting the artifacts of software systems
- “ UML is a graphical language with set of rules and semantics ”
- The UML uses mostly graphical notations to express the OO analysis and design of software projects.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013

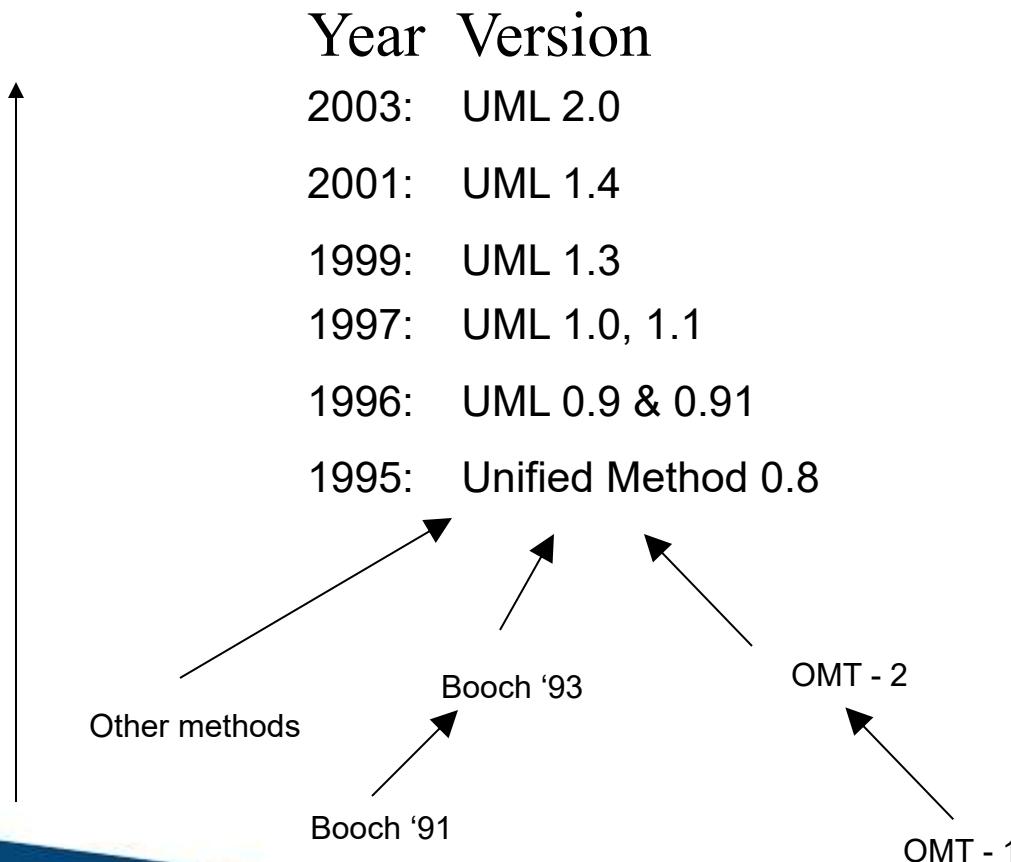


- Rules and semantics of a model expressed in the form of - Object Constraint Language(OCL)
- Simplifies the complex process of software design

Why we use UML?

- Use graphical notation: more clearly than natural language (imprecise) and code (too detailed).
- Help acquire an overall view of a system.
- UML is *not* dependent on any one language or technology.

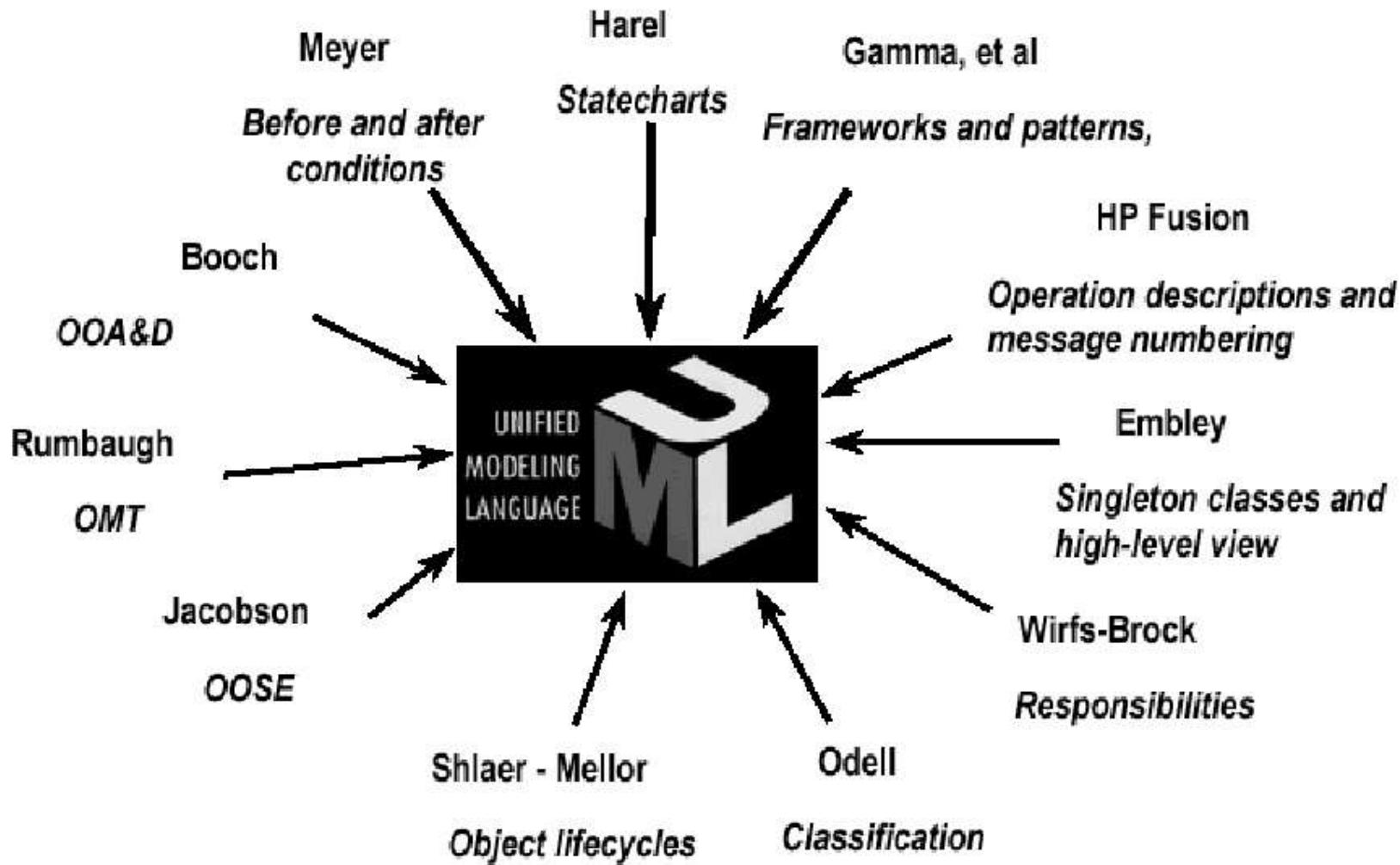
UML Versions



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



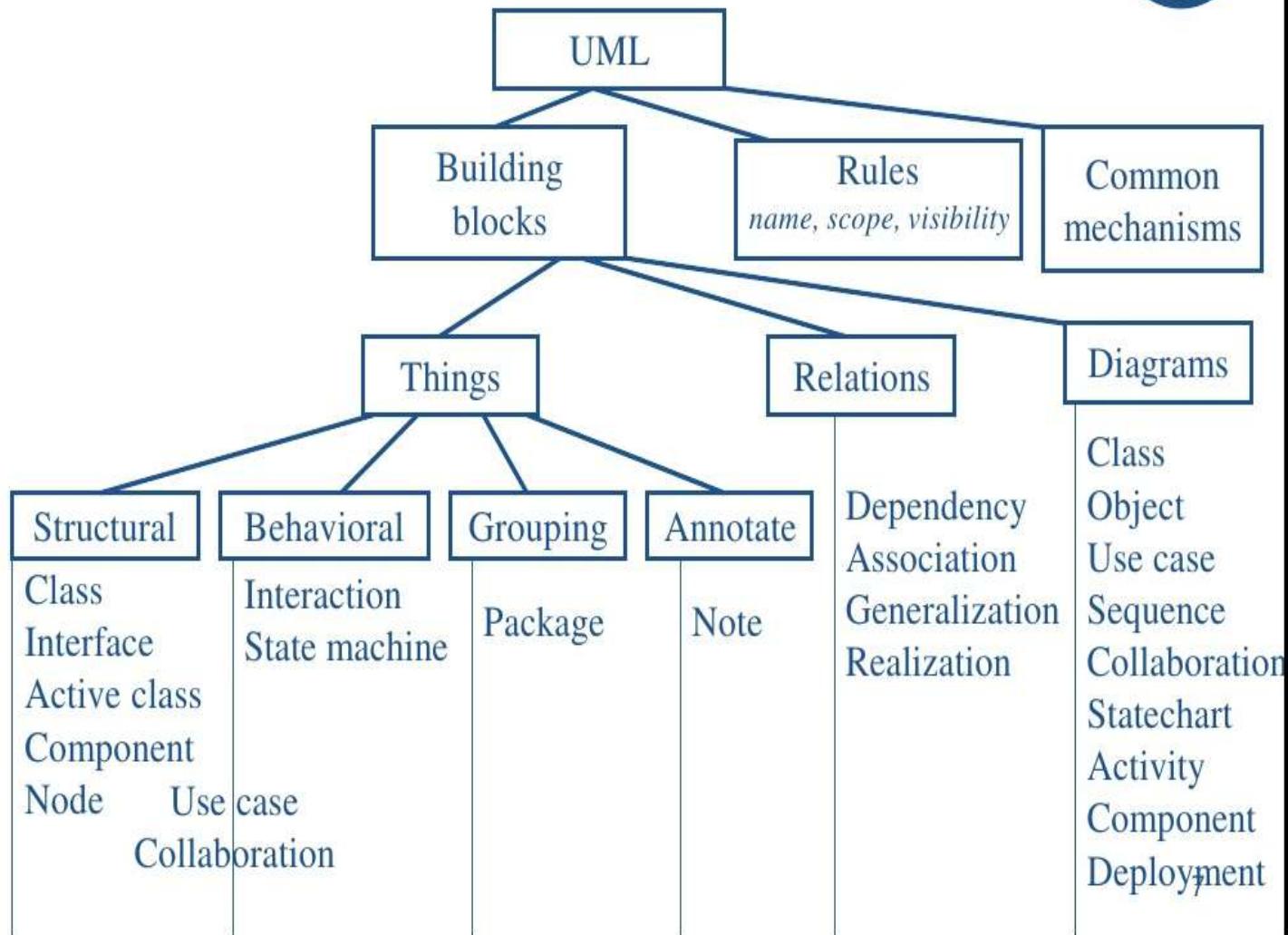
- ❖ Language can be used from general initial design to very specific detailed design across the entire software development lifecycle
- ❖ Increase understanding/communication of product to customers and developers
- ❖ Support for diverse application areas
- ❖ Support for UML in many software packages today (e.g. Rational, plugins for popular IDE's like JBuilder, NetBeans, Eclipse)
- ❖ Based upon experience and needs of the user community



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





Types of UML Diagrams

- Use Case Diagram
- Class Diagram (static)
- Behavior diagram (dynamic)
 - Interaction diagram
 - Sequence Diagram
 - Collaboration diagram
 - State chart diagram
 - Activity diagram
- Implementation Diagram
 - Component Diagram
 - Deployment Diagram



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Self-study

- Use Case Diagram

Example 1: Use case Diagram for a Website



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Library management system – Example2



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Example 3 : Hotel Management System



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Lesson Plan-2

Class diagram



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



UML class diagram:

- the classes in an OO system
- their fields and methods
- connections between the classes
 - that interact or inherit from each other

- Not represented in a UML class diagram:
 - details of how the classes interact with each other
 - algorithmic details; how a particular behavior is implemented



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Diagram of one class

- class name in top of box
- attributes (optional)
 - should include all fields of the object
- operations / methods (optional)
 - may omit trivial (get/set) methods

Rectangle

- width: int

- height: int

/ area: double

+ Rectangle(width: int, height: int)

+ distance(r: Rectangle): double



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Student

-name:String

-id:int

-total Students:int

#getID():int

+getName():String

-getEmailAddress():String

+ getTotal Students():int



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Class attributes (= fields)

- attributes (fields, instance variables)
 - *visibility name : type [count] = default_value*
 - visibility:
 - + public
 - # protected
 - private
 - ~ package (default)
 - / derived

- underline static attributes
- **derived attribute:** not stored, but can be computed from other attribute values
- attribute example:
 - balance : double = 0.00

Student

-name:String

-id:int

-totalStudents:int

#getID():int

+getName():String

-getEmailAddress():String

+getTotalStudents():int



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Relationships between classes

- **generalization:** an inheritance relationship
 - inheritance between classes
 - interface implementation
- **association:** a usage relationship
 - dependency
 - aggregation
 - composition



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Generalization

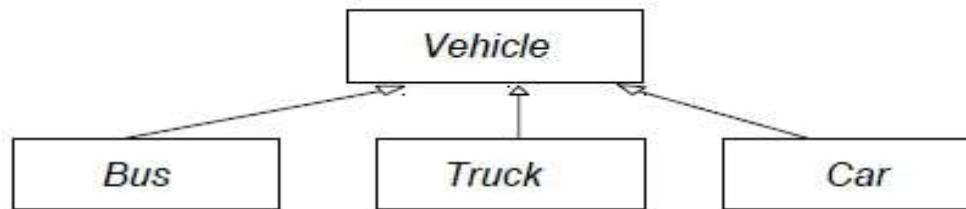
- Generalization is a relationship between two classes: a general class and a special class:



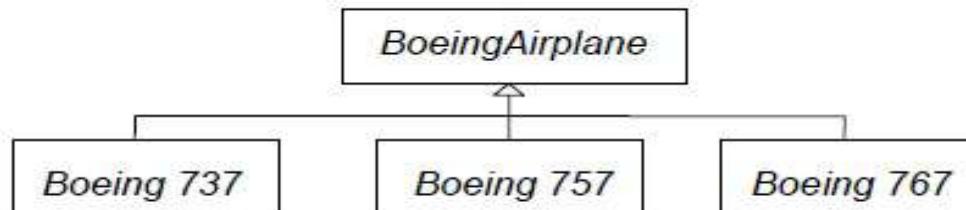
Generalization Relationships

- Generalization is a form of association.
- Sub-classes are specialized versions of their super-classes.

Separate target style



Shared target style



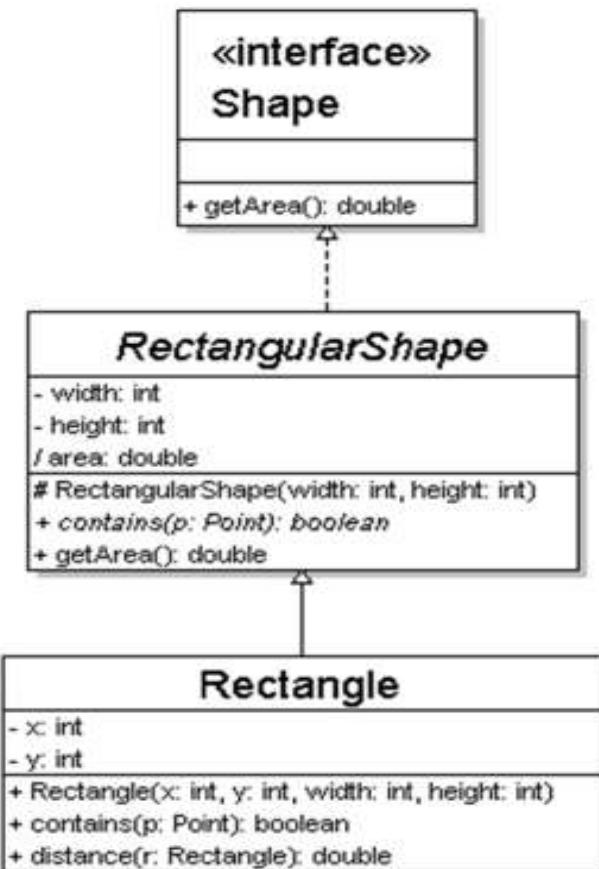
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Generalization (inheritance) relationships

- hierarchies drawn top-down
- arrows point upward to parent
- line/arrow styles indicate whether parent is a(n):
 - class: solid line, black arrow
 - abstract class: solid line, white arrow
 - interface: dashed line, white arrow
- often omit trivial / obvious generalization relationships, such as drawing the Object class as a parent



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Association

An association represents a relationship between two classes:

is flown with ►



**PRESIDENCY
UNIVERSITY**

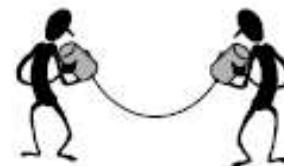
Private University Estd. in Karnataka State by Act No. 41 of 2013



- In the UML, a *navigable* association is represented by an open arrow.

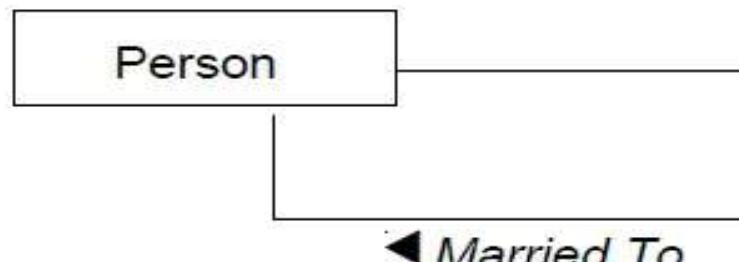


- A *bidirectional* association does not have an arrow.



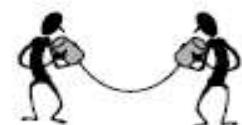
Binary Association

- A binary association is drawn as a solid path connecting two classes or both ends may be connected to the same class.

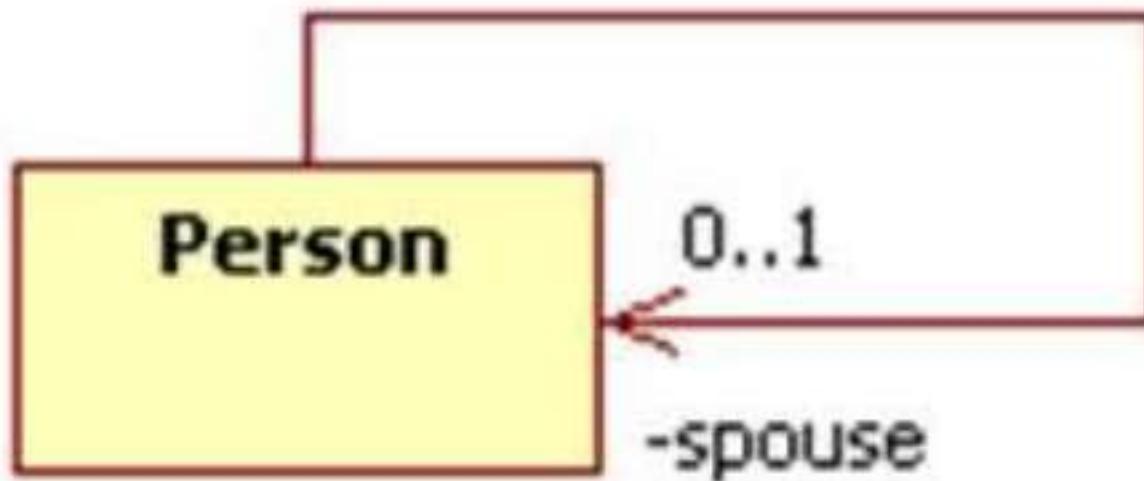


Note:

- Association Name
- Association Roles



isMarriedTo



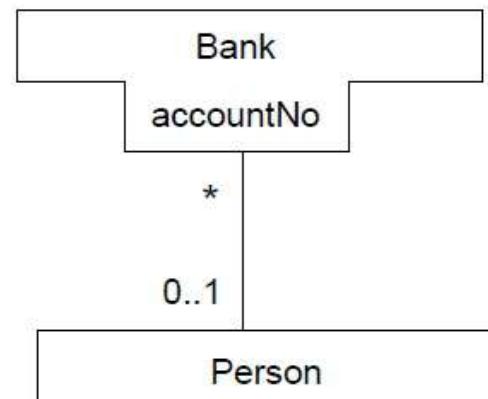
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



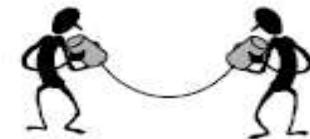
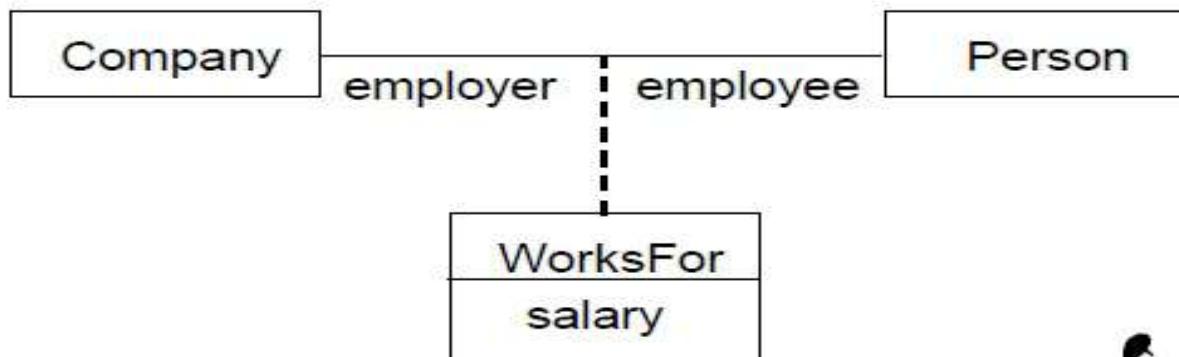
Qualifiers

- A *qualifier* is an association attribute. Account# is an attribute of Bank,
- but is important enough to note as the qualifier in the *association*.

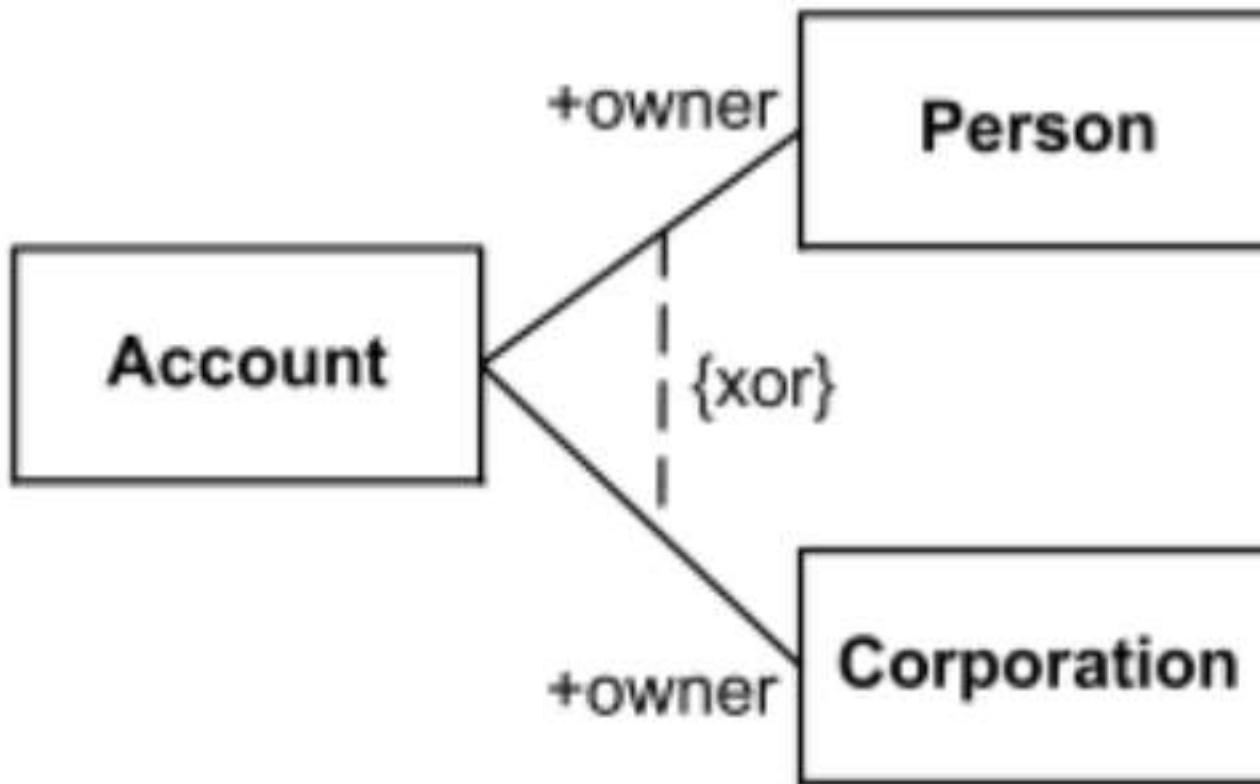


UML Association Class

- An *association class* is an association that also has class properties.
- An association class is shown as a class symbol attached by a dashed line to an association path.



OR Association



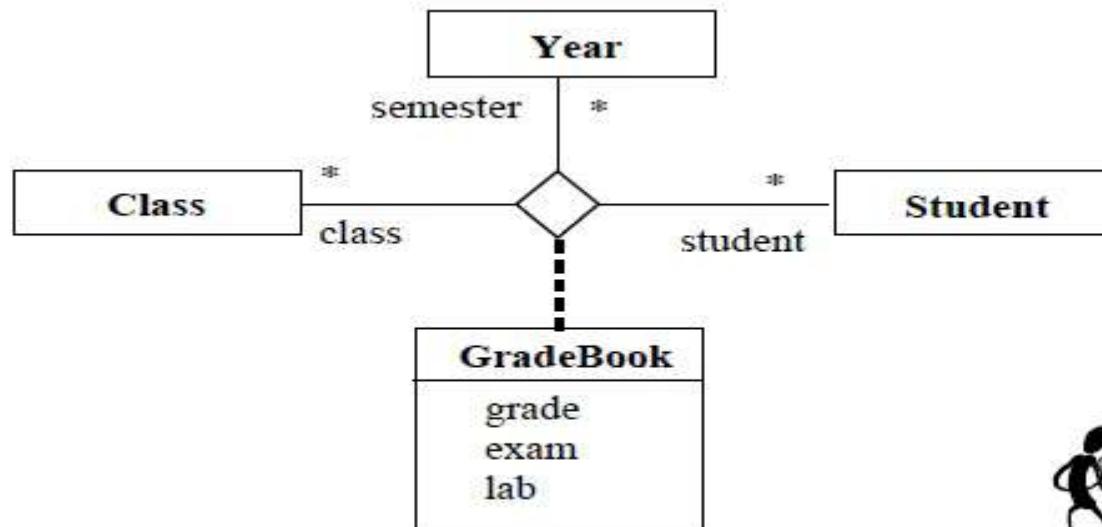
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



UML N-Ary Association

- An *n-ary association* is an association among more than two classes.
- The n-ary association is more difficult to understand. It is better to convert an n-ary association to binary association.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Multiplicity

A multiplicity allows for statements about the number of objects that are involved in an association:

*

0..1

0..1

Minimum 0, Maximum 1 =
One or None (Optional)

1 = 1..1

Minimum 1, Maximum 1 =
Exactly One

* = 0..*

Minimum 0, Maximum Unlimited =
Many, One, or None (Optional)

1..*

Minimum 1, Maximum Unlimited =
Many, Atleast One



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Multiplicity of associations

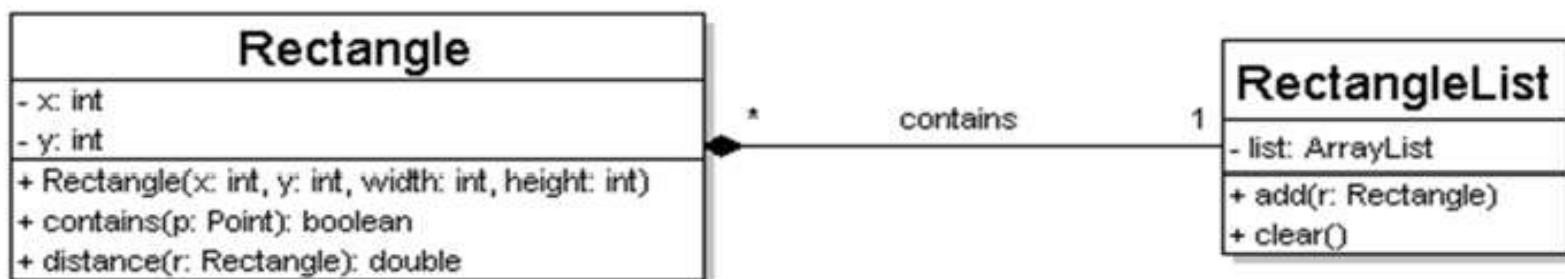
■ one-to-one

- each student must carry exactly one ID card



■ one-to-many

- one rectangle list can contain many rectangles



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Aggregation

An aggregation is a special case of an association (see above) meaning "consists of":



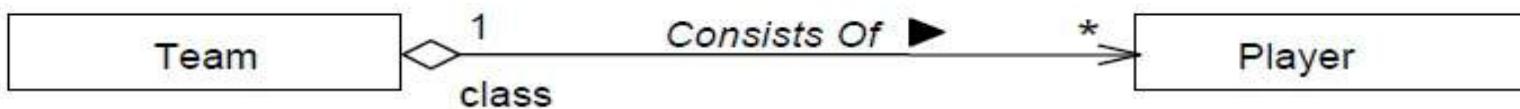
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Aggregation Relationships

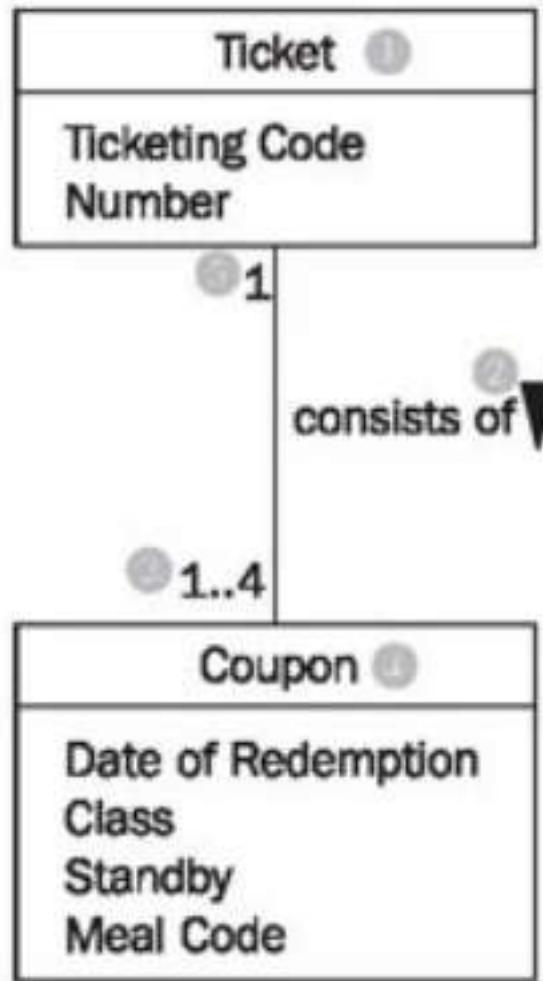
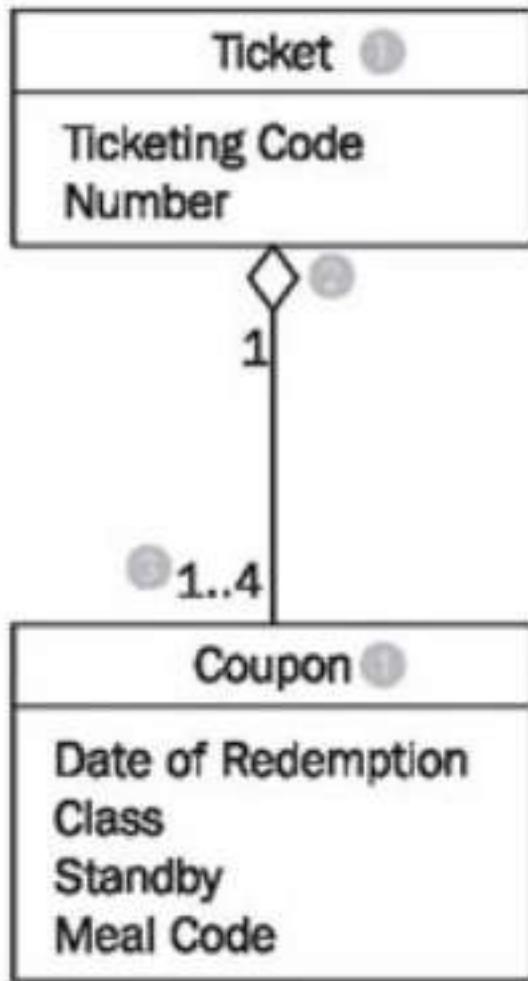
- Aggregations are *a-part-of* relationships, where a class consists of several component classes.
- Aggregation is a special form of association.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Composition

- Compositions are aggregations with strong ownership.
- They use solid diamonds.
- When the composition dies, all components die too

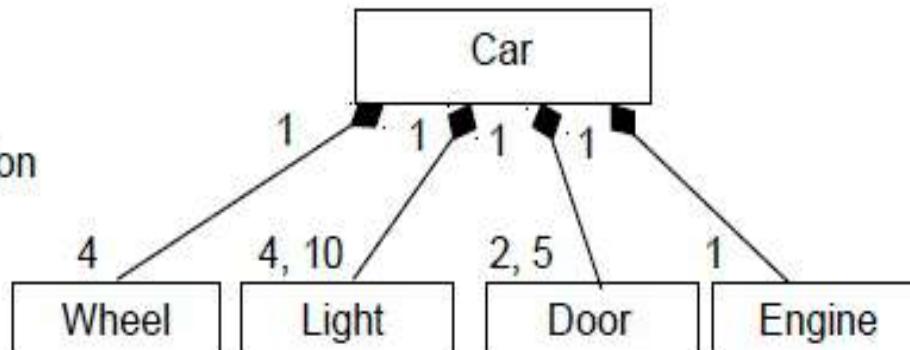


**PRESIDENCY
UNIVERSITY**

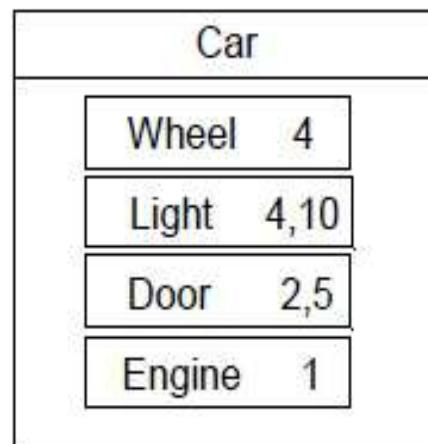
Private University Estd. in Karnataka State by Act No. 41 of 2013



graphical composition



nested composition



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



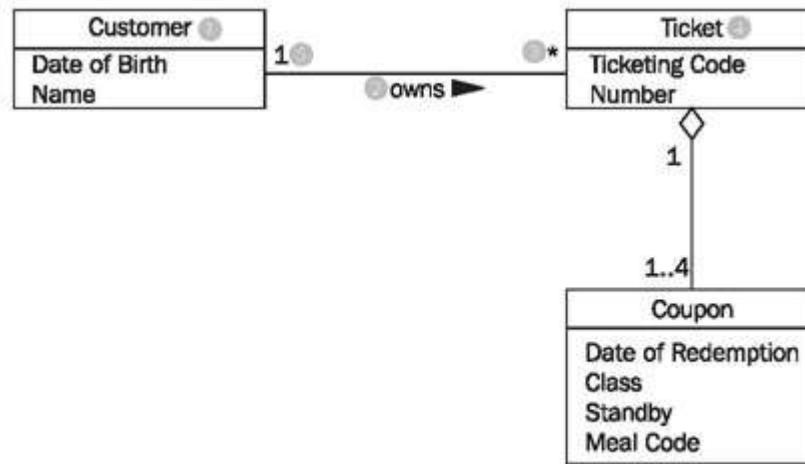
- Draw Class diagram with the classes customer, ticket, and coupon, their attributes, and their associations



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





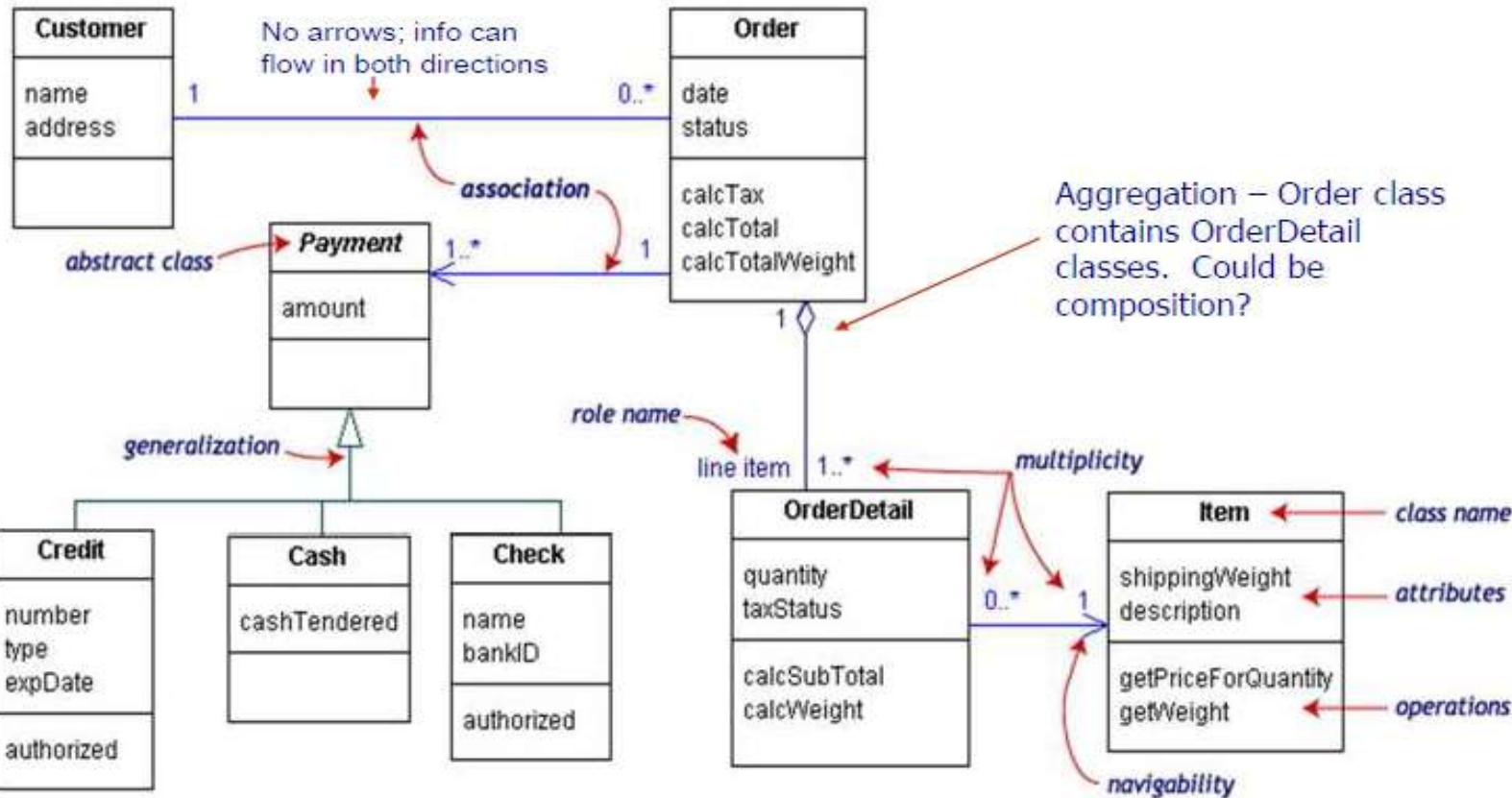
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



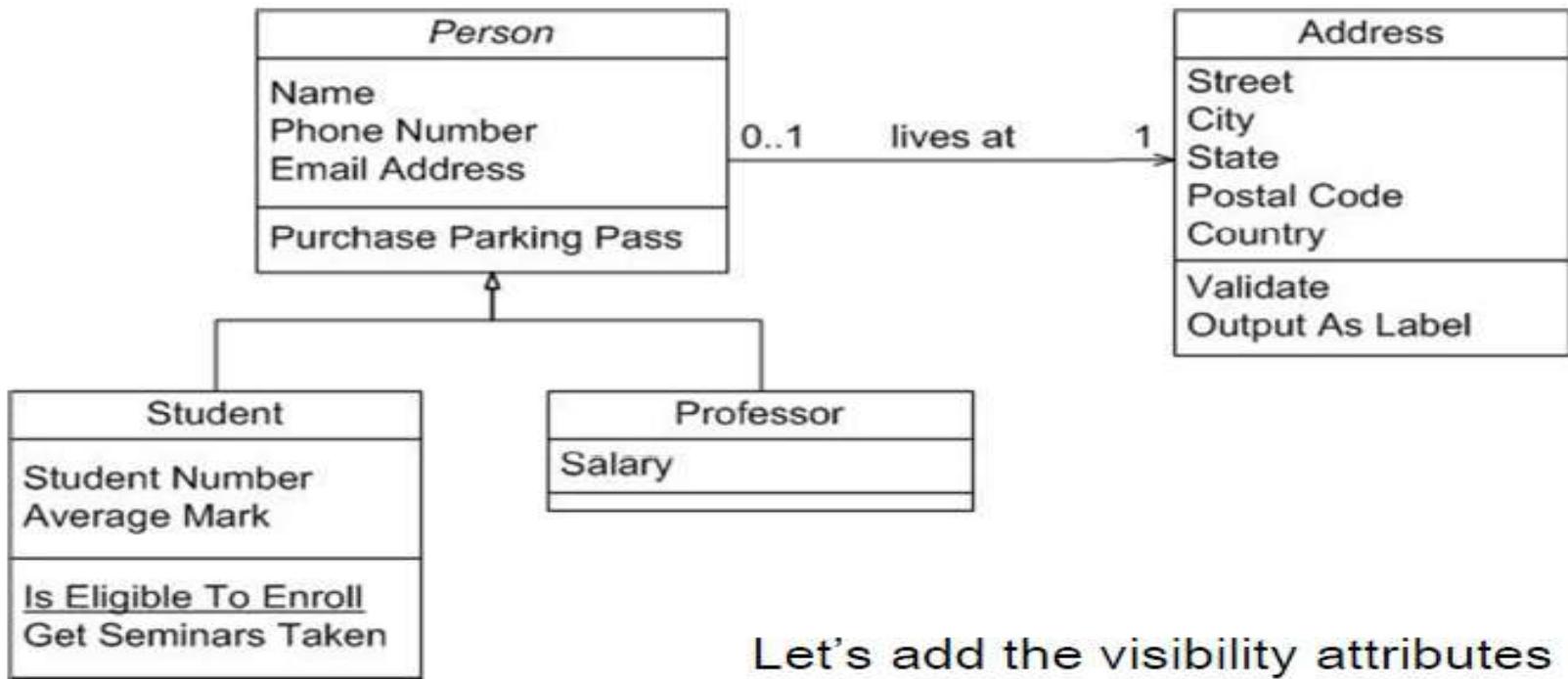
Example-2

Class diagram example



Example-3

UML example: people



Let's add the visibility attributes



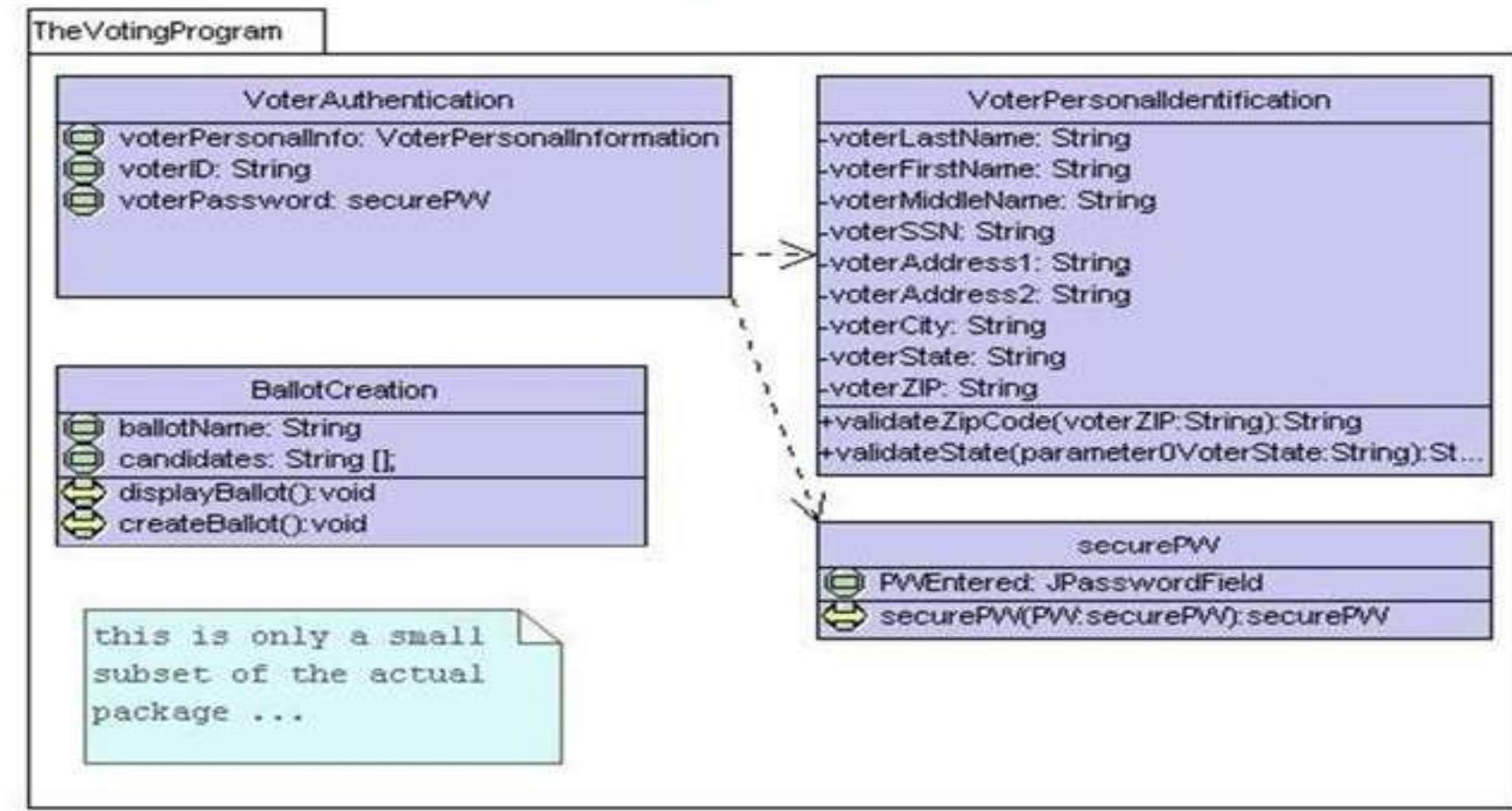
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013

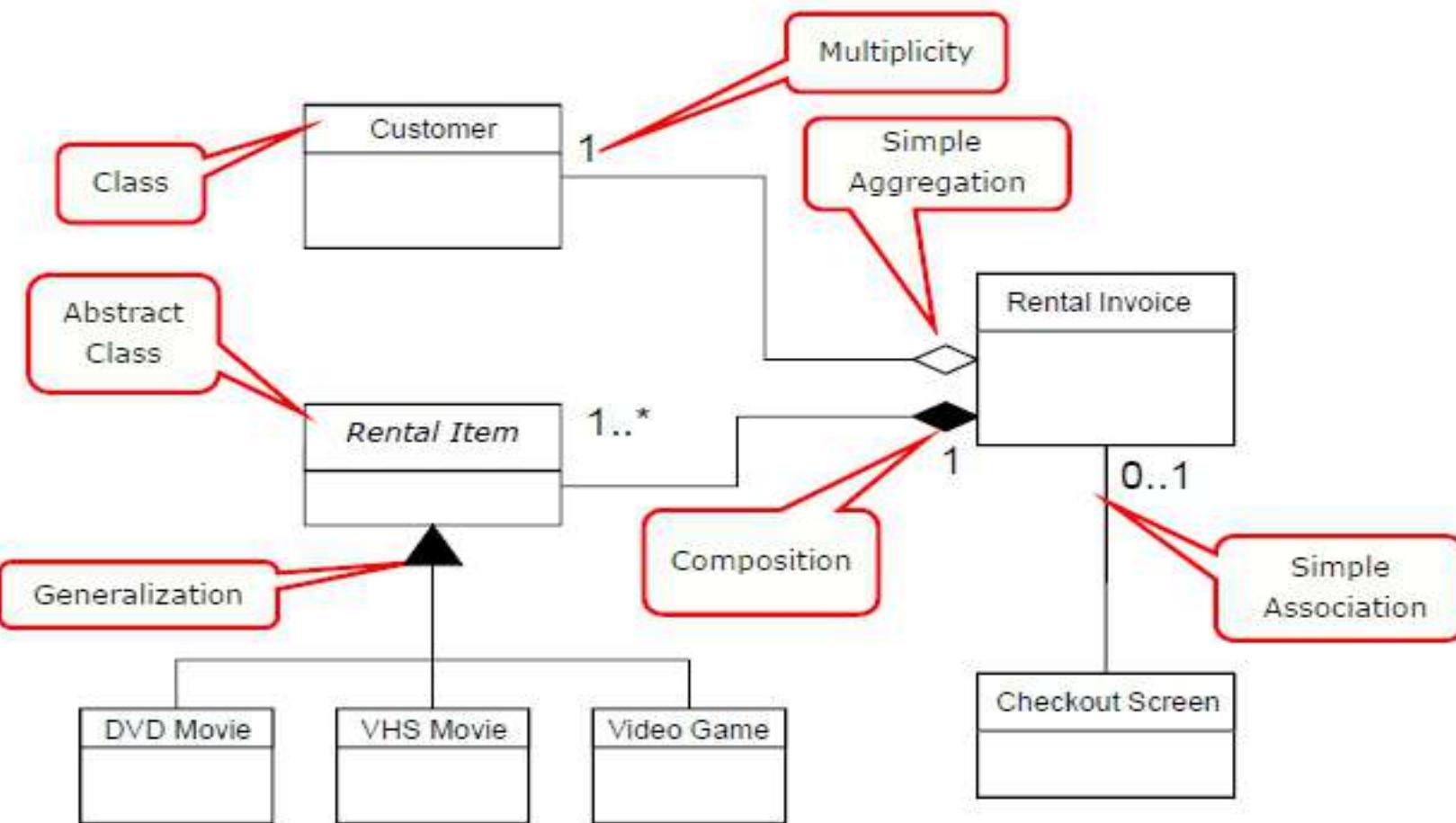


Example-4

Class diagram: voters



Class diagram example: video store

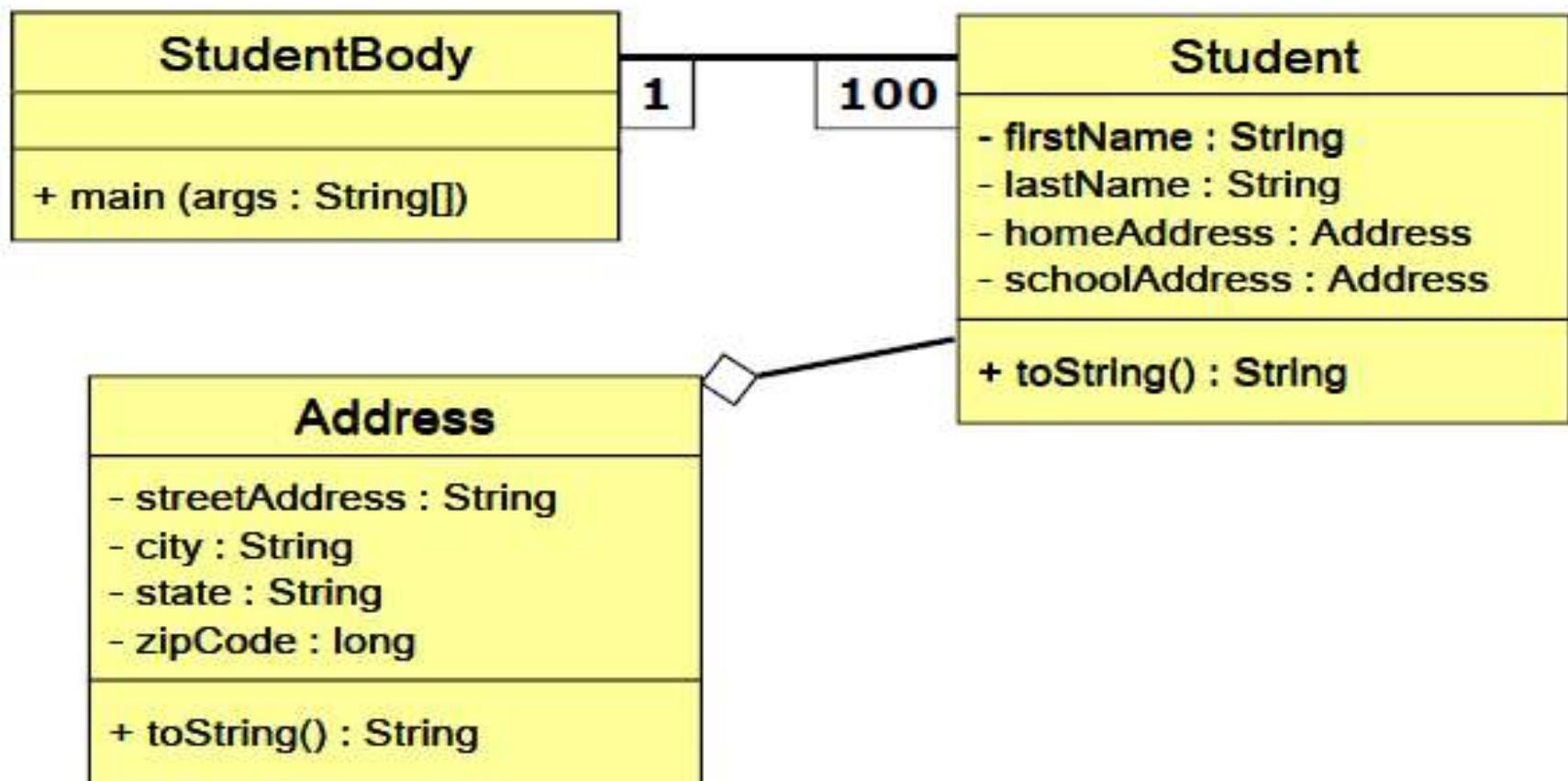


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Class diagram example: student

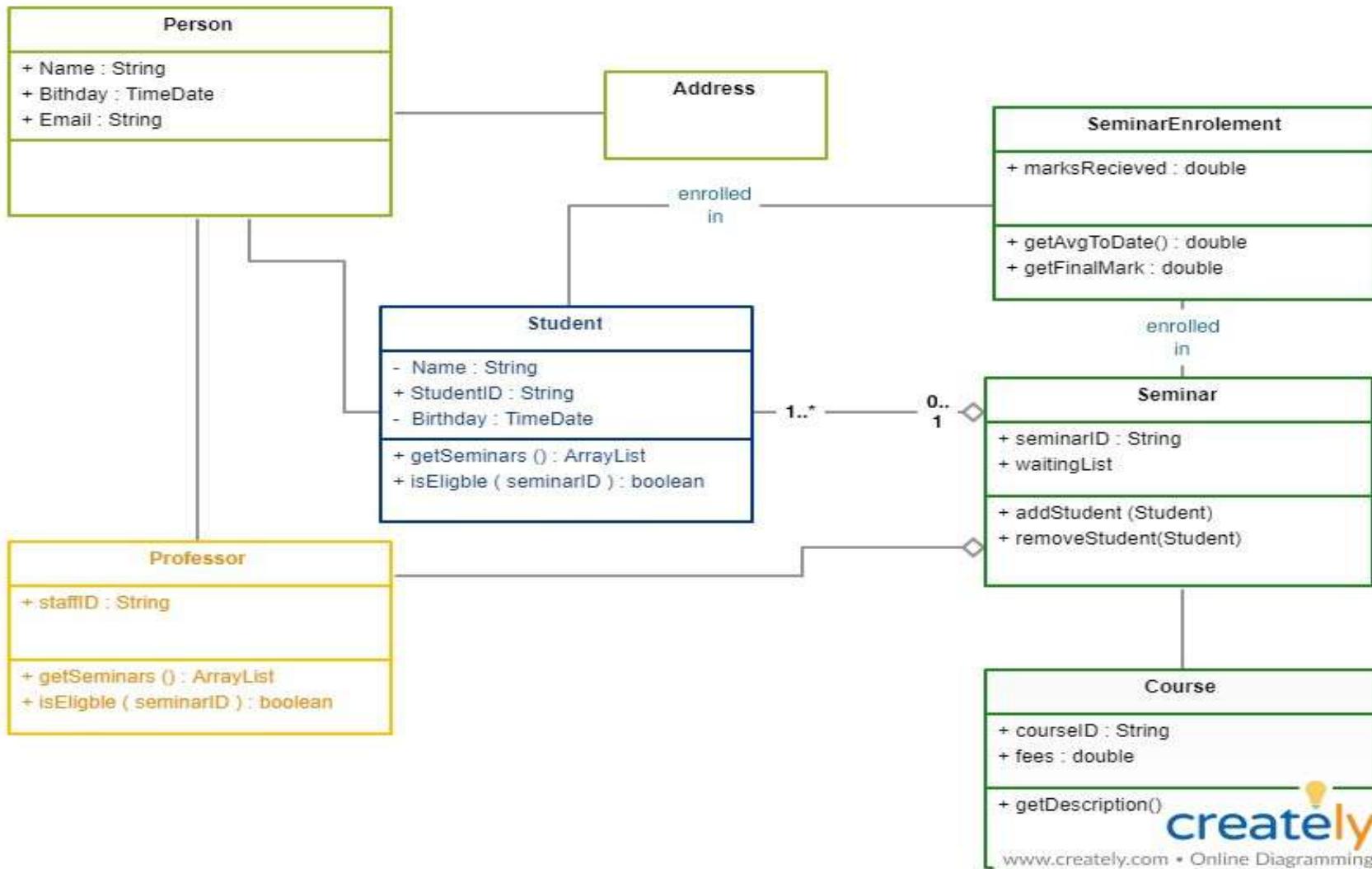


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



SEMINAR HALL Booking



PRESIDENCY
UNIVERSITY

Private University Estd. in Karnataka State by Act No. 41 of 2013



create 
www.createley.com • Online Diagramming

■ Tools for creating UML diagrams

- Umbrello(free)
<https://umbrello.kde.org/>
- Violet (free)
 - <http://horstmann.com/violet/>
- Rational Rose
 - <http://www.rational.com/>
- Visual Paradigm UML Suite (trial)
 - <http://www.visual-paradigm.com/>
 - (nearly) direct download link:

<http://www.visualparadigm.com/vp/download.jsp?product=vpuml&edition=ce>

- (there are many others, but most are commercial)

Interaction Diagram



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Interaction Diagram

- From the name *Interaction* it is clear that the diagram is used to describe some type of interactions among the different elements in the model.
- Interaction diagrams are models that describe how a group of objects interact / collaborate in some behavior - typically a single use-case.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Interaction Diagram

- **Purpose**
 - To capture dynamic behaviour of a system.
 - To describe the message flow in the system.
 - To describe structural organization of the objects.
 - To describe interaction among objects.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Drawing the interaction diagram

- The purpose of interaction diagrams are to capture the dynamic aspect of a system.
- Dynamic aspect can be defined as the snap shot of the running system at a particular moment.
- So the following things are to identified clearly before drawing the interaction diagram:
 - Objects taking part in the interaction.
 - Message flows among the objects.
 - The sequence in which the messages are flowing.
 - Object organization.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Forms of Interaction Diagram

- This interactive behaviour is represented in UML by two diagrams known as
 - *Sequence diagram* and
 - *Collaboration diagram.*



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Sequence Diagram

- Describe the flow of messages, events, actions between objects .
- An important characteristic of a sequence diagram is that time passes from top to bottom and model important runtime interactions between the parts that make up the system.
- Typically used to document and understand the logical flow of the system .



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Sequence Diagram Key Parts

- **participant:** object or entity that acts in the diagram
- **message:** communication between participant objects
- the **axes** in a sequence diagram:
 - –**horizontal:** which object/participant is acting
 - –**vertical:** time (down -> forward in time)



**PRESIDENCY
UNIVERSITY**

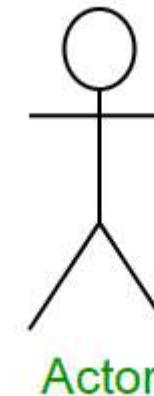
Private University Estd. in Karnataka State by Act No. 41 of 2013



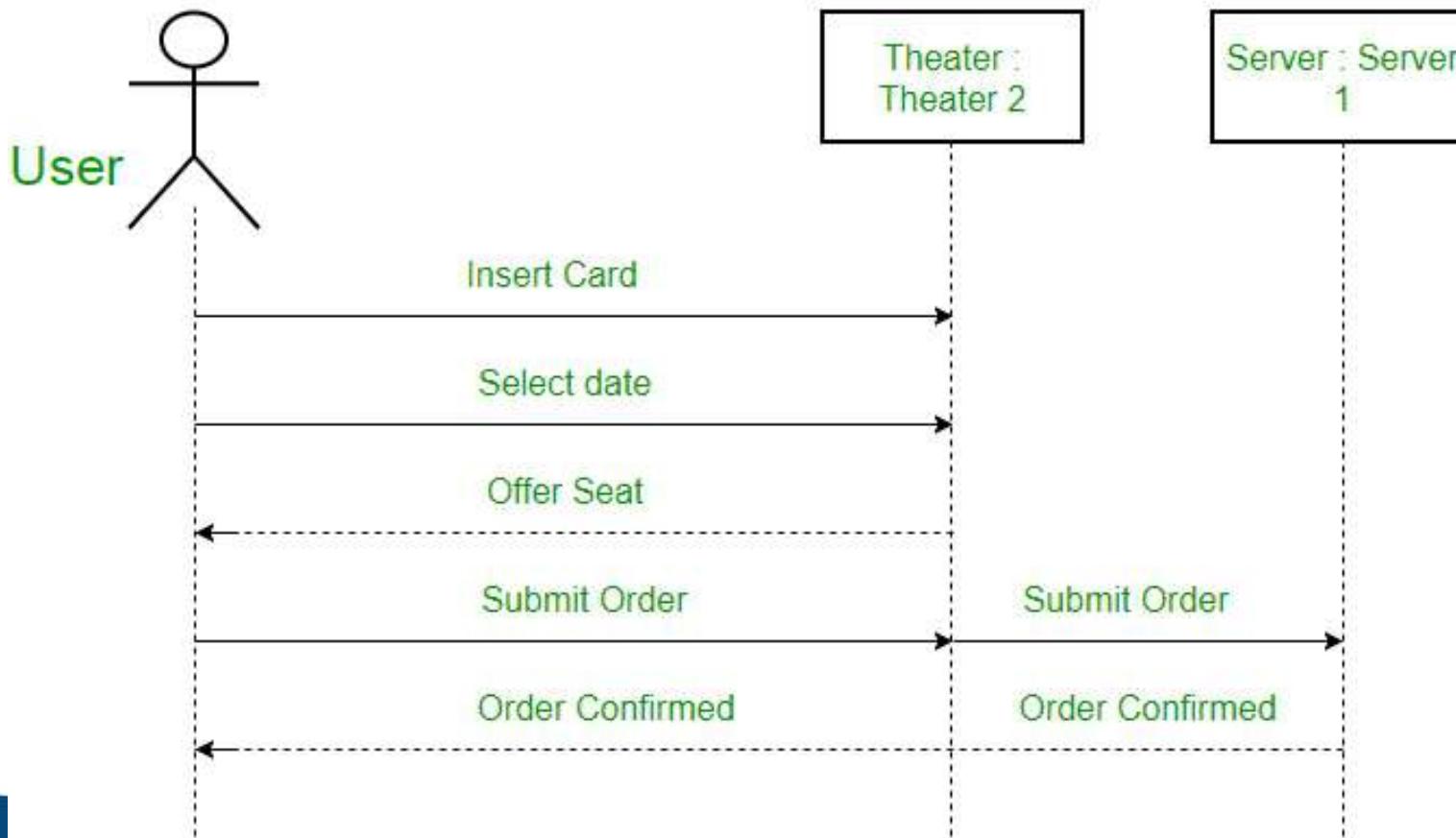
Sequence Diagram Notations –

1 Actors –

- An actor in a UML diagram represents a type of role where it interacts with the system and its objects.
- It is important to note here that an actor is always outside the scope of the system we aim to model using the UML diagram.



An actor interacting with a seat reservation system



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



2 Lifelines -

- A lifeline is a named element which depicts an individual participant in a sequence diagram.
- Each instance in a sequence diagram is represented by a lifeline. Lifeline elements are located at the top in a sequence diagram.
- The standard in UML for naming a lifeline follows the following format – Instance Name : Class Name



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



X : Class 1

Here X is the object or
instance name
Class 1 is the class
name

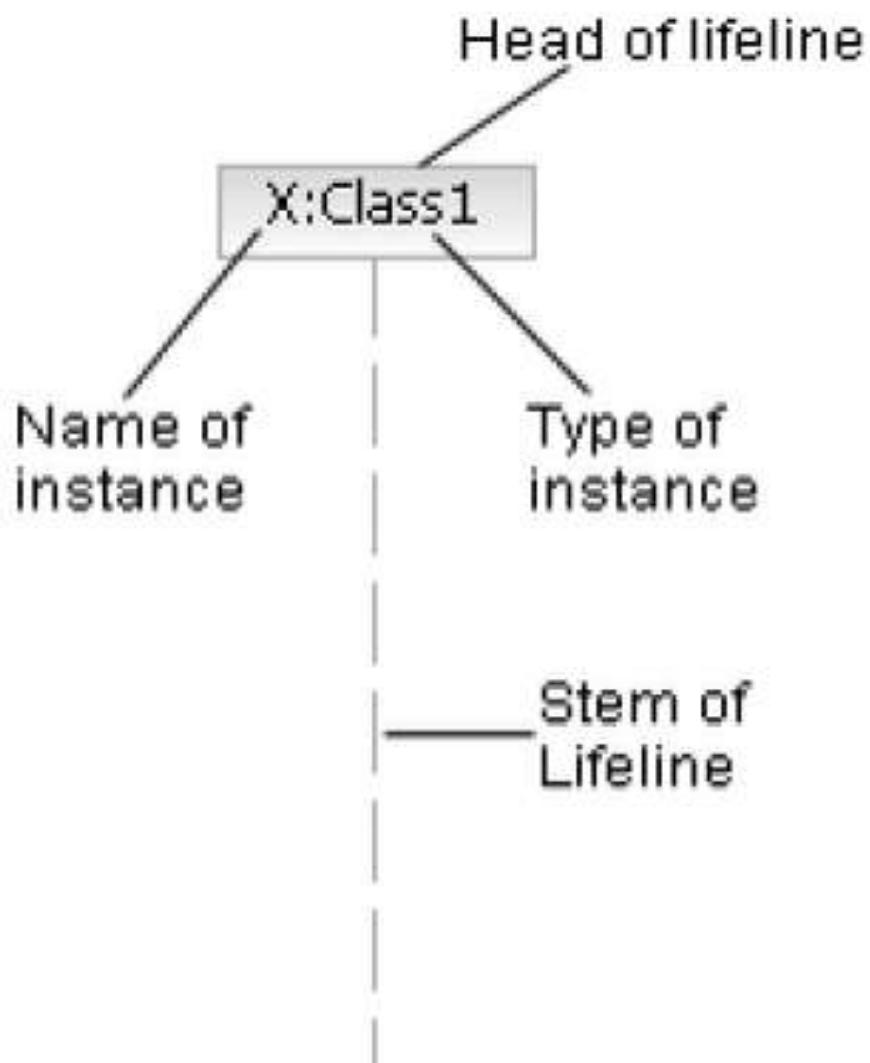
We display a lifeline in a rectangle called head with its name and type. The head is located on top of a vertical dashed line (referred to as the stem) as shown above.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





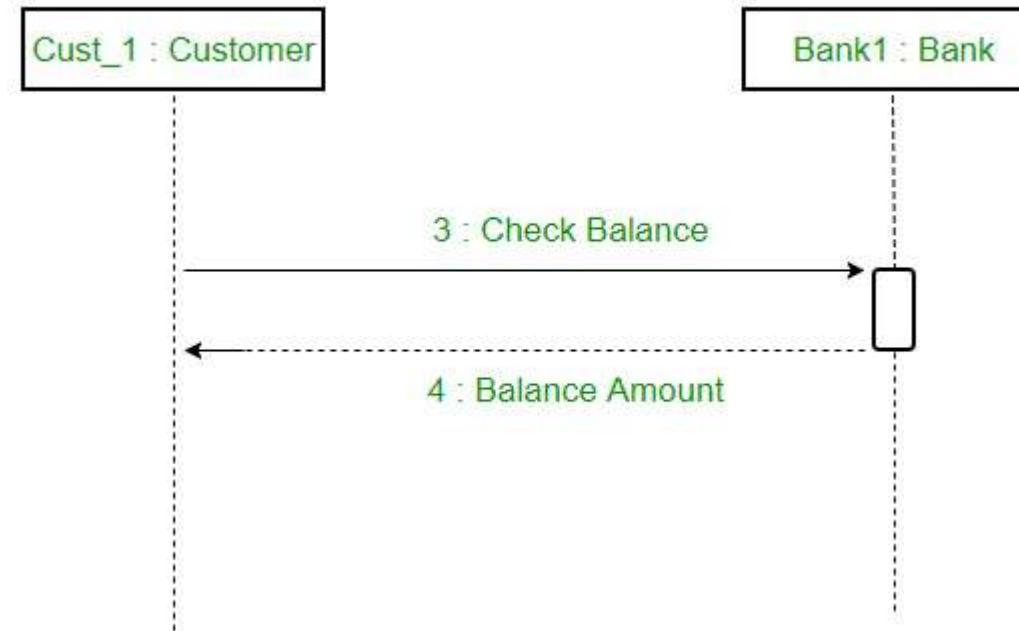
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Difference between a lifeline and an actor

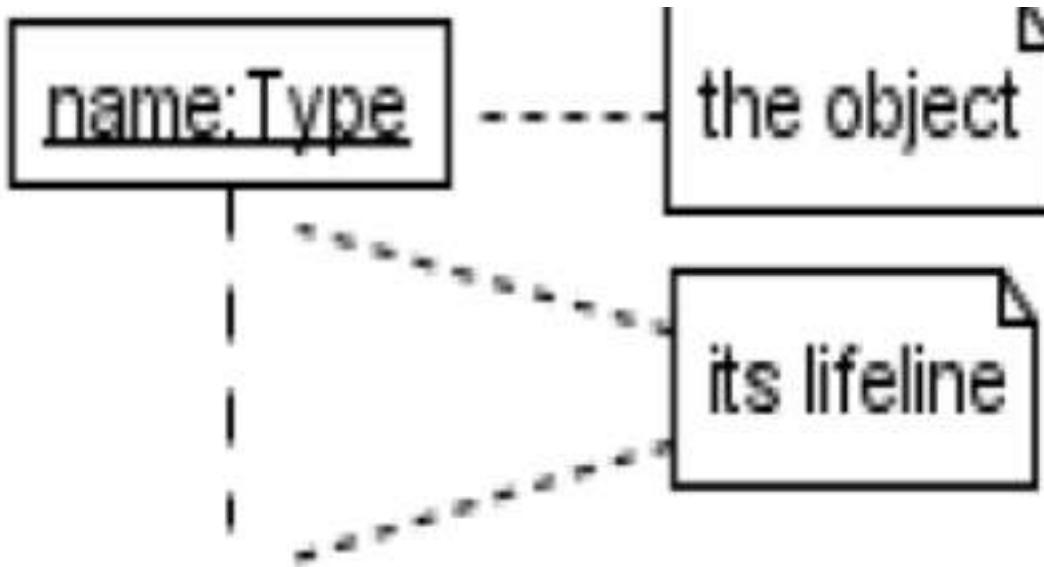
- A lifeline always portrays an object internal to the system
- Actors are used to depict objects external to the system.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





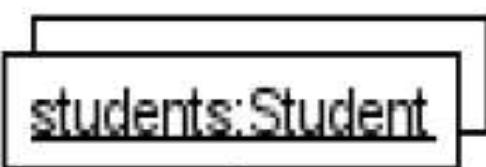
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



MultiObject

When we want to show how a client interacts with the elements of a collection, we can use a multiobject. Its basic notation is:



A collection of Student instances,
the collection is named 'students'



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



3 Messages –

Communication between objects is depicted using messages.

The messages appear in a sequential order on the lifeline.

We represent messages using arrows.

Lifelines and messages form the core of a sequence diagram.

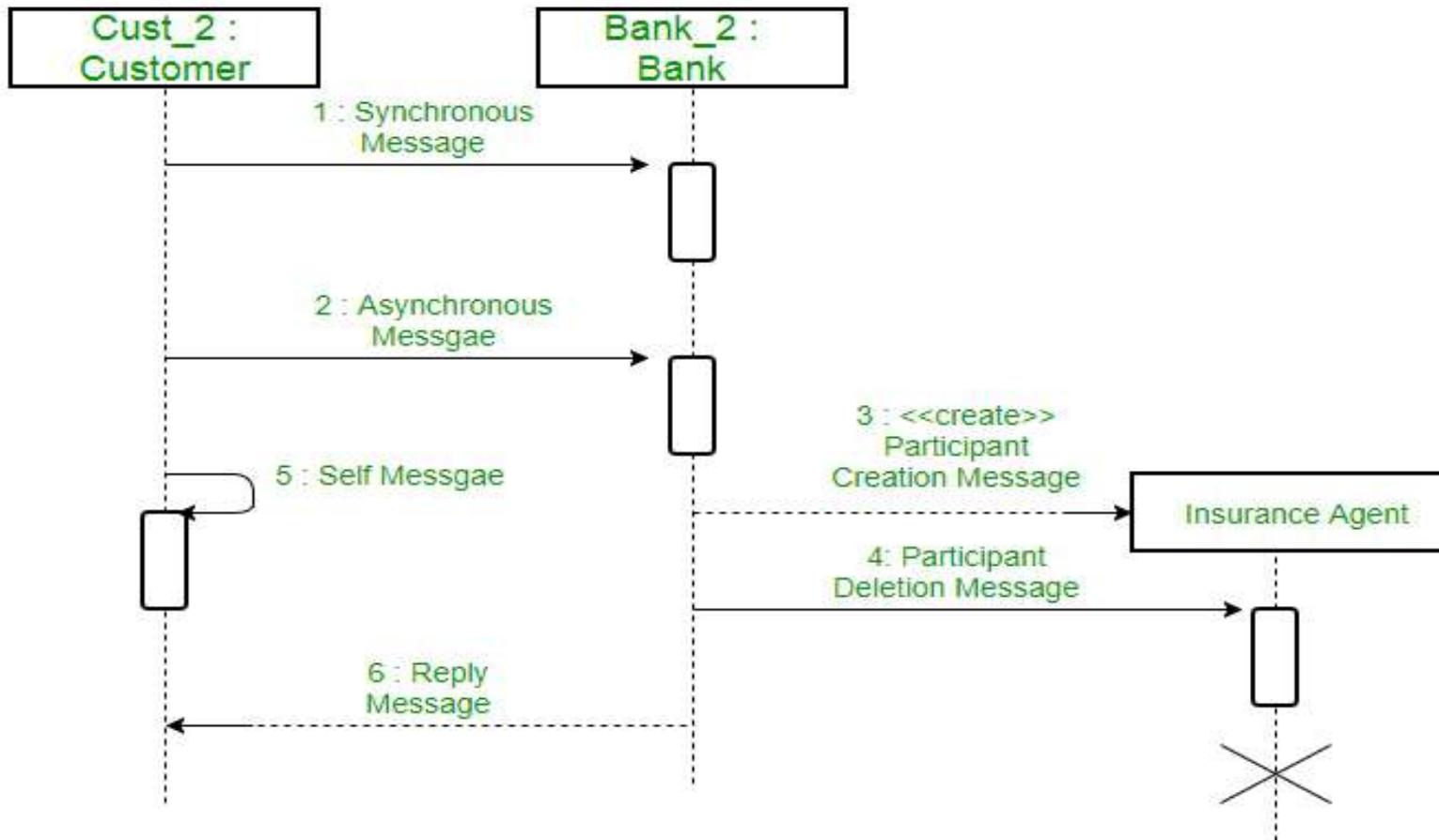


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Messages can be broadly classified into the following categories :



- **Synchronous messages –**

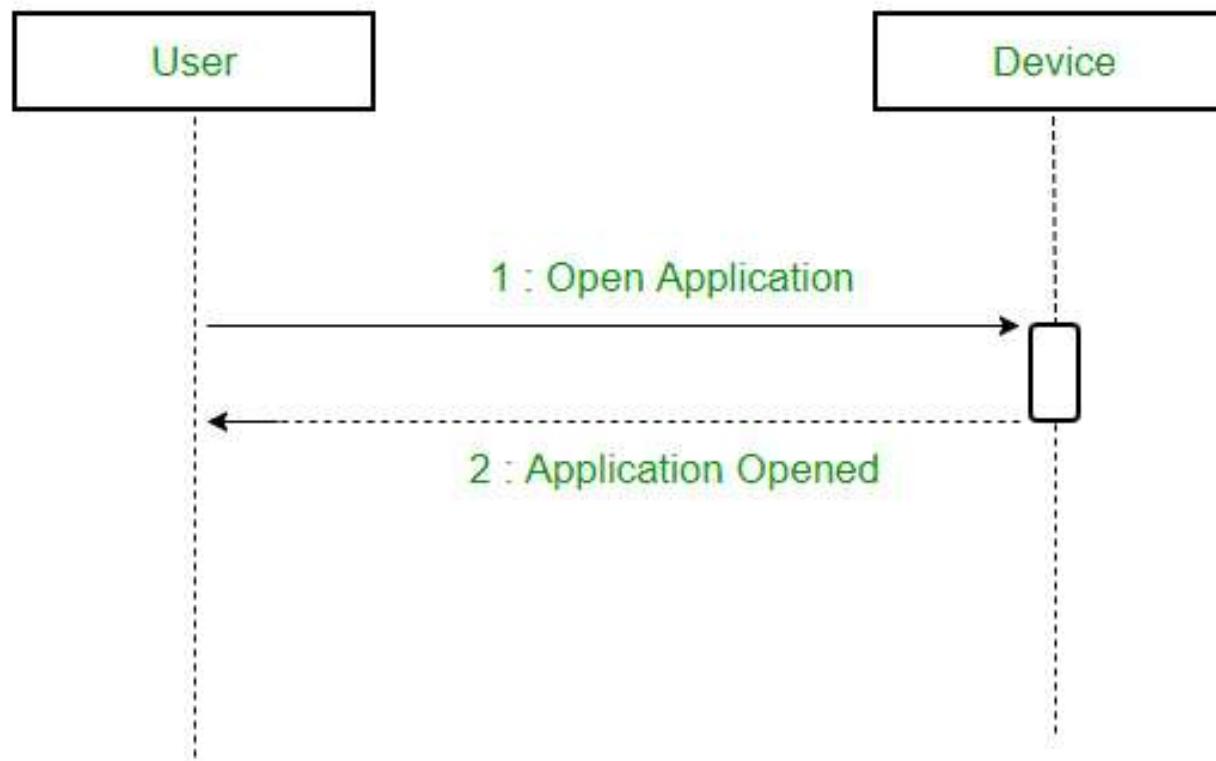
- A synchronous message waits for a reply before the interaction can move forward.
- The sender waits until the receiver has completed the processing of the message.
- The caller continues only when it knows that the receiver has processed the previous message i.e. it receives a reply message.
- We use a solid arrow head to represent a synchronous message.



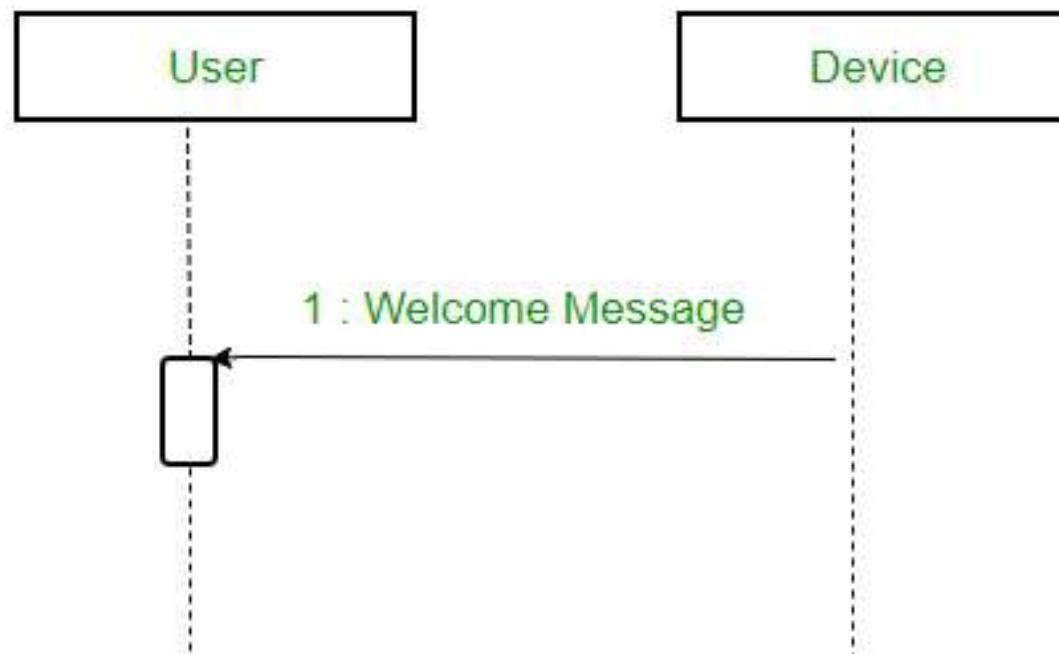
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





- **Asynchronous Messages** – An asynchronous message does not wait for a reply from the receiver. The interaction moves forward irrespective of the receiver processing the previous message or not. We use a lined arrow head to represent an asynchronous message.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- **Create message –**

- We use a Create message to instantiate a new object in the sequence diagram.
- There are situations when a particular message call requires the creation of an object.
- It is represented with a dotted arrow and create word labelled on it to specify that it is the create Message symbol.

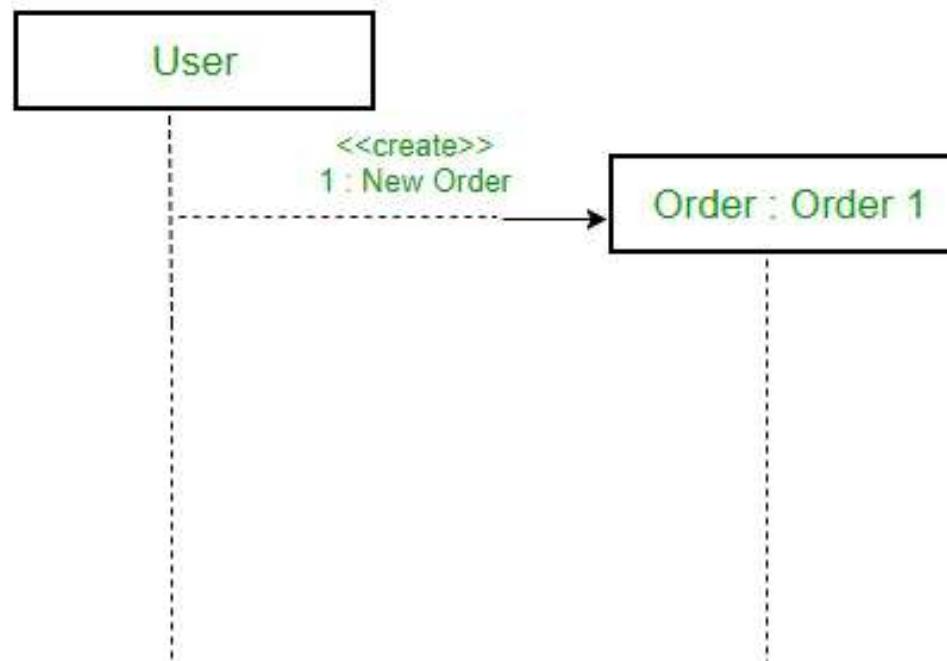


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



For example - The creation of a new order on a e-commerce website would require a new object of Order class to be created.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- **Delete Message –**

- We use a Delete Message to delete an object.
- When an object is deallocated memory or is destroyed within the system we use the Delete Message symbol.
- It destroys the occurrence of the object in the system.
- It is represented by an arrow terminating with a x.

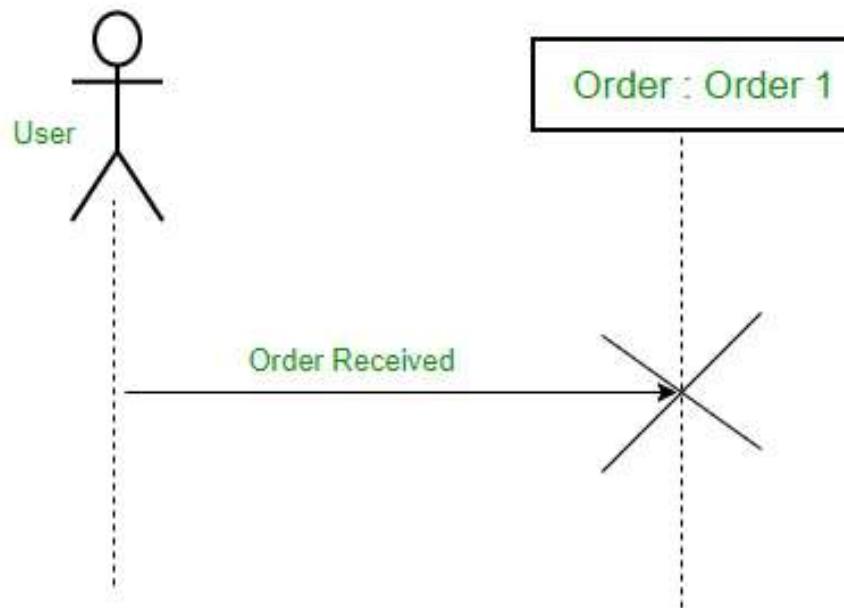


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



For example – In the scenario below when the order is received by the user, the object of order class can be destroyed.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- **Self Message –**

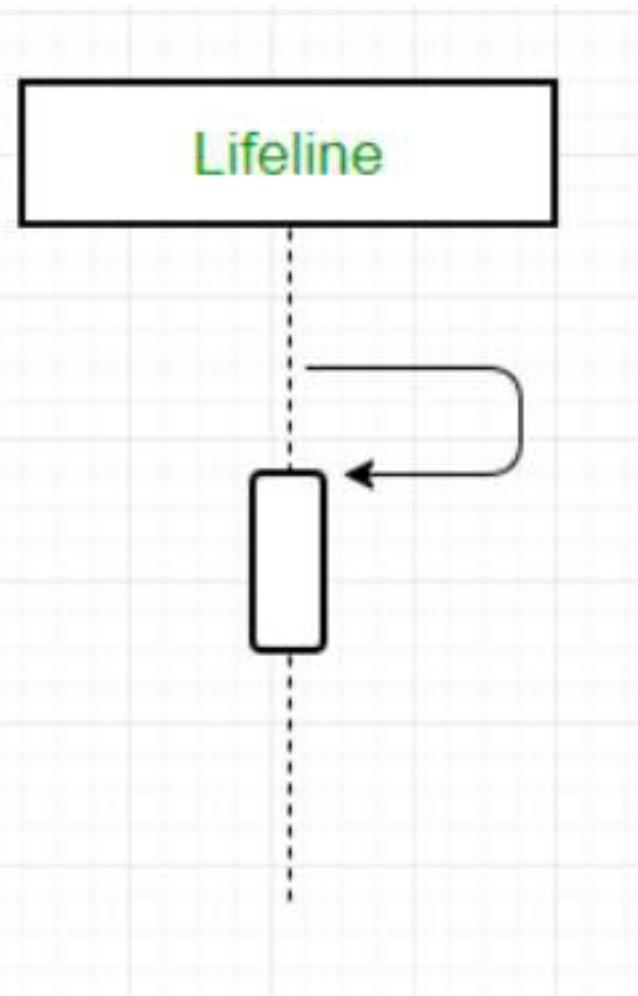
- Certain scenarios might arise where the object needs to send a message to itself.
- Such messages are called Self Messages and are represented with a U shaped arrow.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



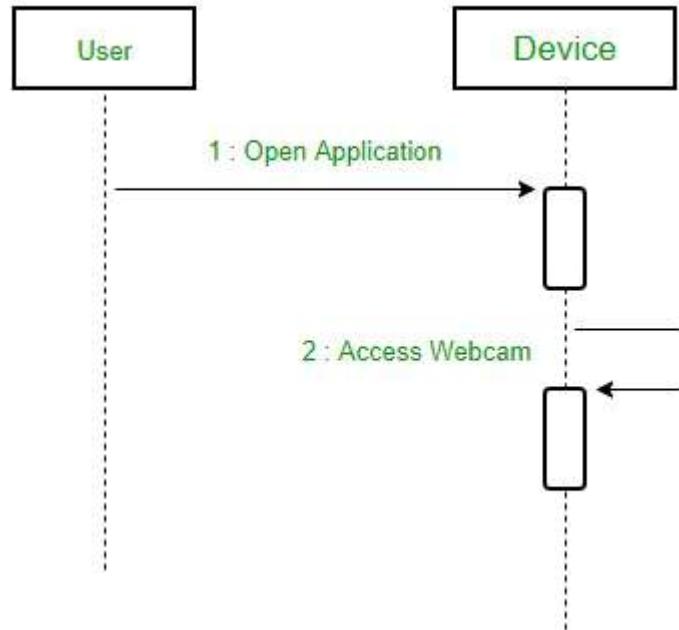


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



For example – Consider a scenario where the device wants to access its webcam. Such a scenario is represented using a self message.



- **Reply Message -**

- Reply messages are used to show the message being sent from the receiver to the sender.
- We represent a return/reply message using an open arrowhead with a dotted line.
- The interaction moves forward only when a reply message is sent by the receiver.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





Figure – reply message

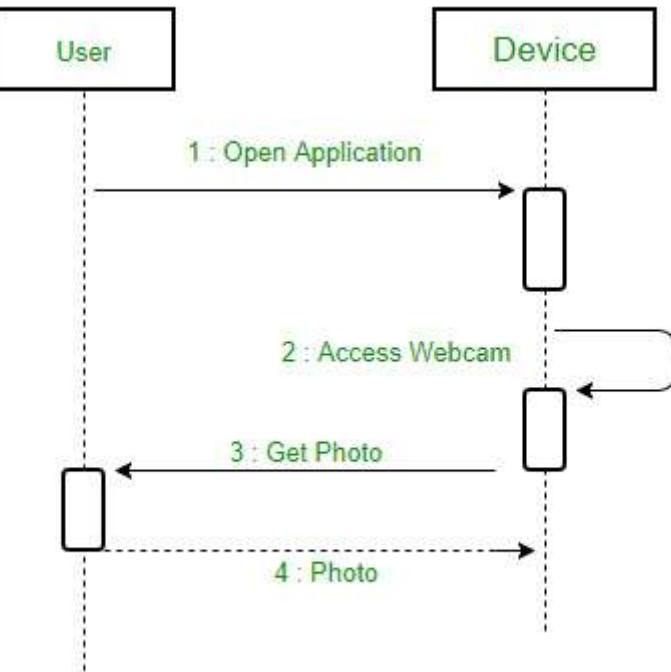


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



For example consider the scenario where the device requests a photo from the user. Here the message which shows the photo being sent is a reply message.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- **Found Message –**

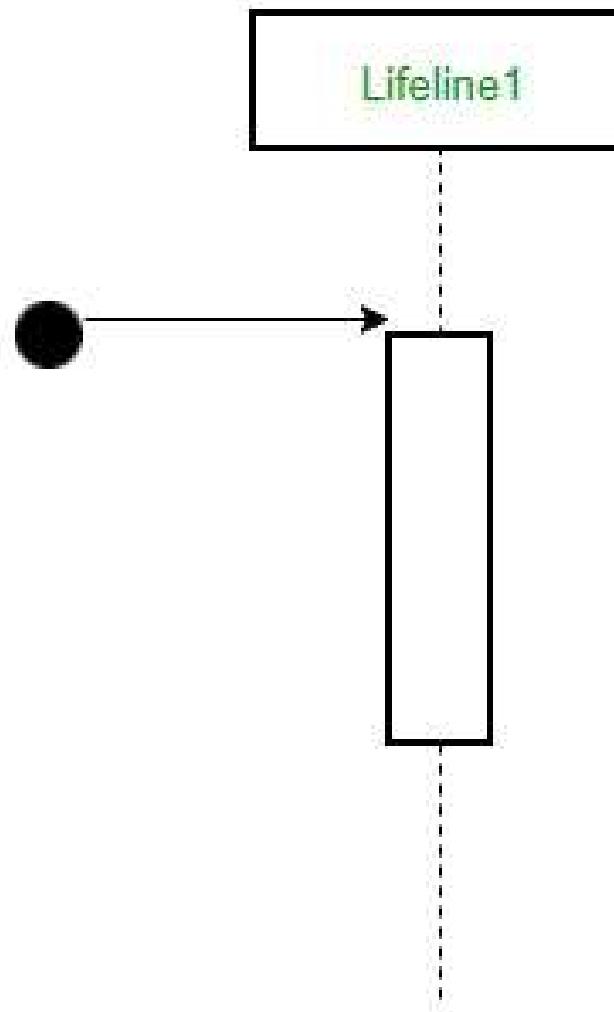
- A Found message is used to represent a scenario where an unknown source sends the message.
- It is represented using an arrow directed towards a lifeline from an end point.
- For example: Consider the scenario of a hardware failure.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



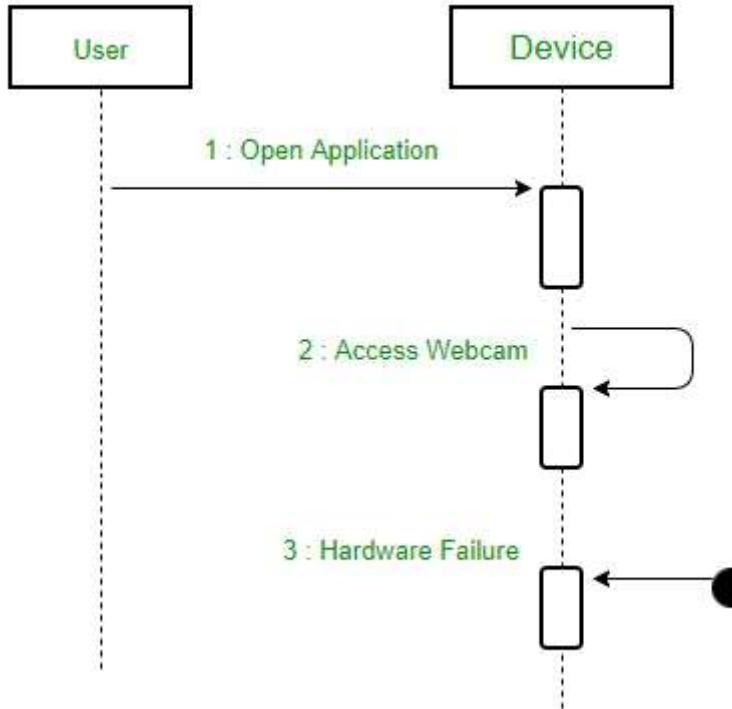


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



It can be due to multiple reasons and we are not certain as to what caused the hardware failure.



- **Lost Message -**

- A Lost message is used to represent a scenario where the recipient is not known to the system.
- It is represented using an arrow directed towards an end point from a lifeline.
- For example: Consider a scenario where a warning is generated.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



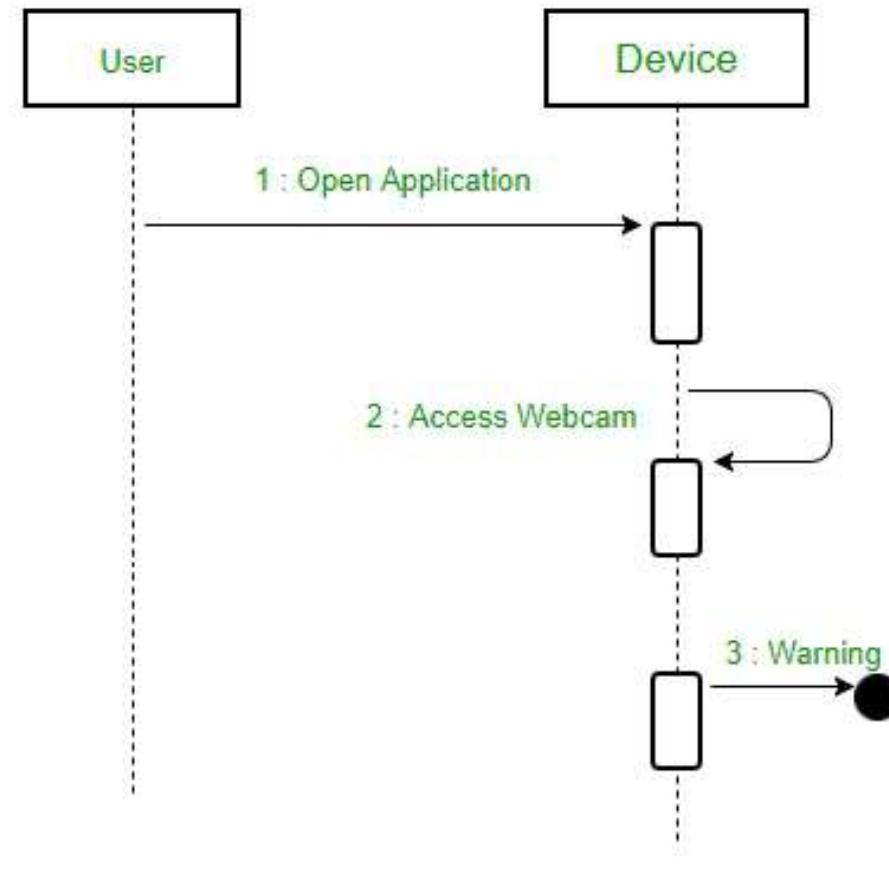


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- The warning might be generated for the user or other software/object that the lifeline is interacting with.
- Since the destination is not known before hand, we use the Lost Message symbol.



4 Guards -

- To model conditions we use guards in UML.
- They are used when we need to restrict the flow of messages on the pretext of a condition being met.
- Guards play an important role in letting software developers know the constraints attached to a system or a particular process.
- For example: In order to be able to withdraw cash, having a balance greater than zero is a condition that must be met as shown below.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



A sequence diagram for an emotion based music player -

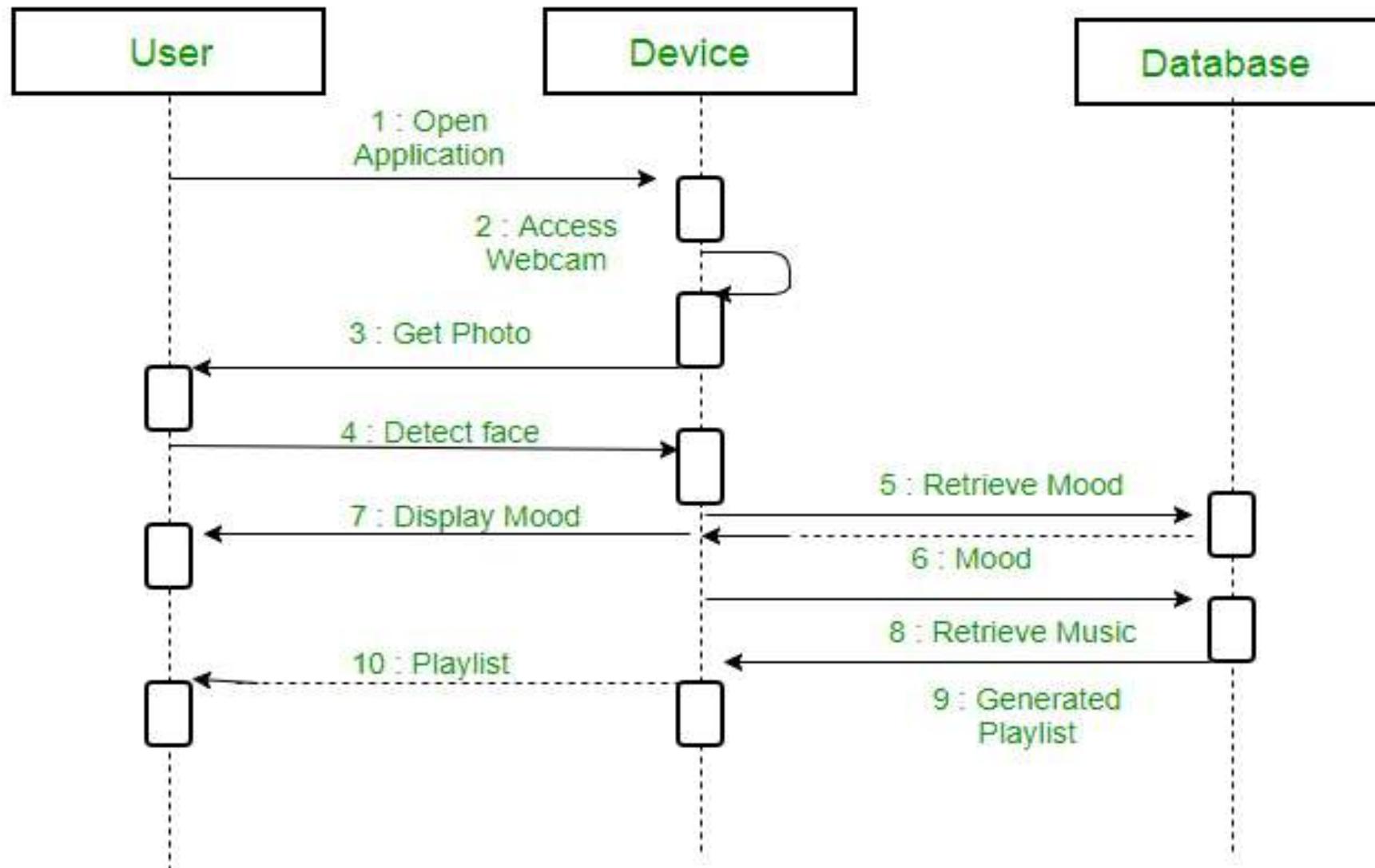
1. Firstly the application is opened by the user.
2. The device then gets access to the web cam.
3. The webcam captures the image of the user.
4. The device uses algorithms to detect the face and predict the mood.
5. It then requests database for dictionary of possible moods.
6. The mood is retrieved from the database.
7. The mood is displayed to the user.
8. The music is requested from the database.
9. The playlist is generated and finally shown to the user.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



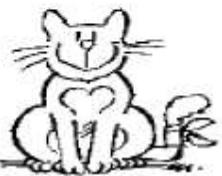


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Fun Example Objects



:Cat



:Policeman



:Person



:RSPCA Inspector



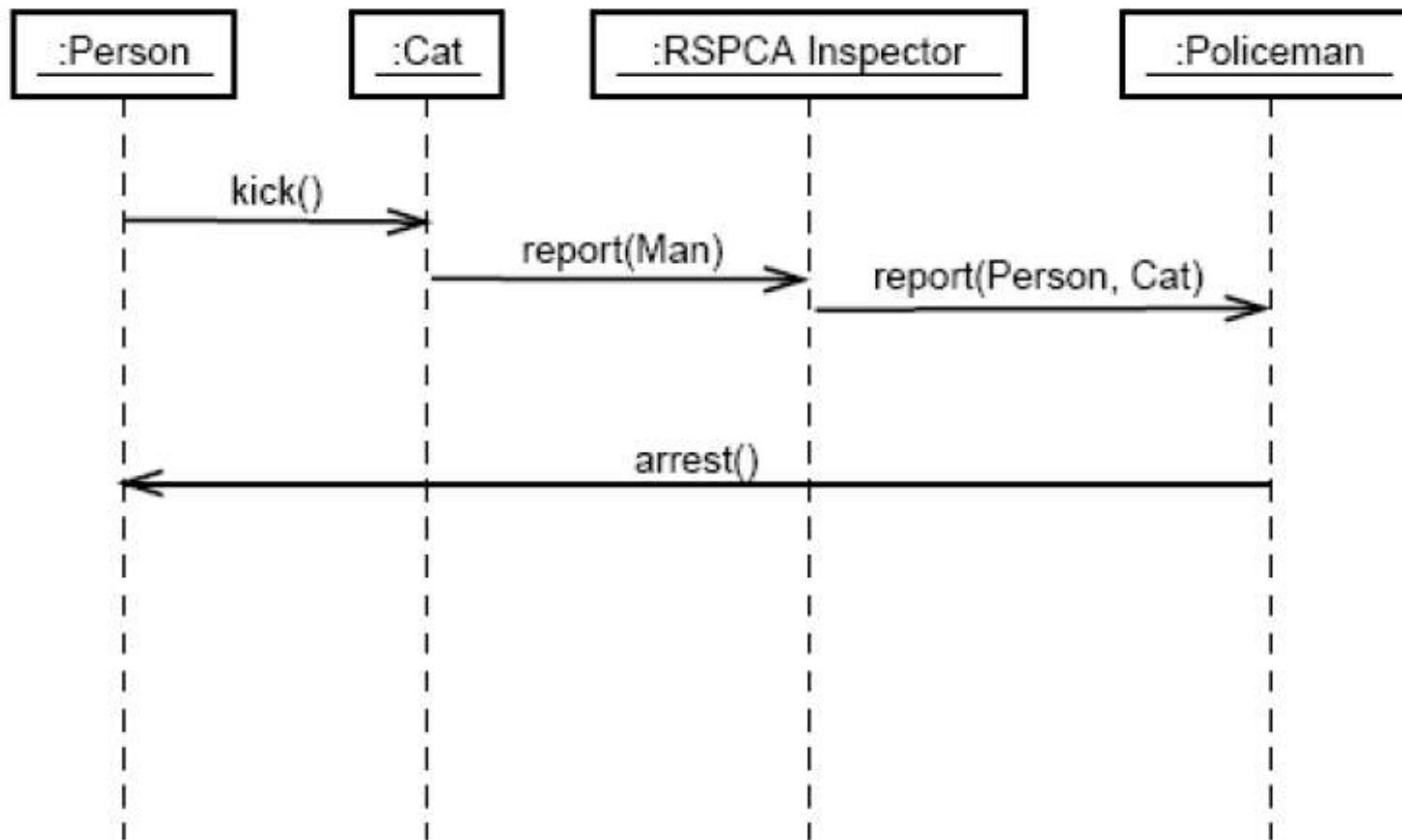
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Fun Example

Sequence diagram



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Online bookstore example

1. The customer begins the interaction by searching for a book by title.
2. The system will return all books with that title.
3. The customer can look at the book description.
4. The customer can place a book in the shopping cart.
5. The customer can repeat the interaction as many times as desired.
6. The customer can purchase the items in the cart by checking out.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Collaboration Diagrams



**PRESIDENCY
UNIVERSITY**

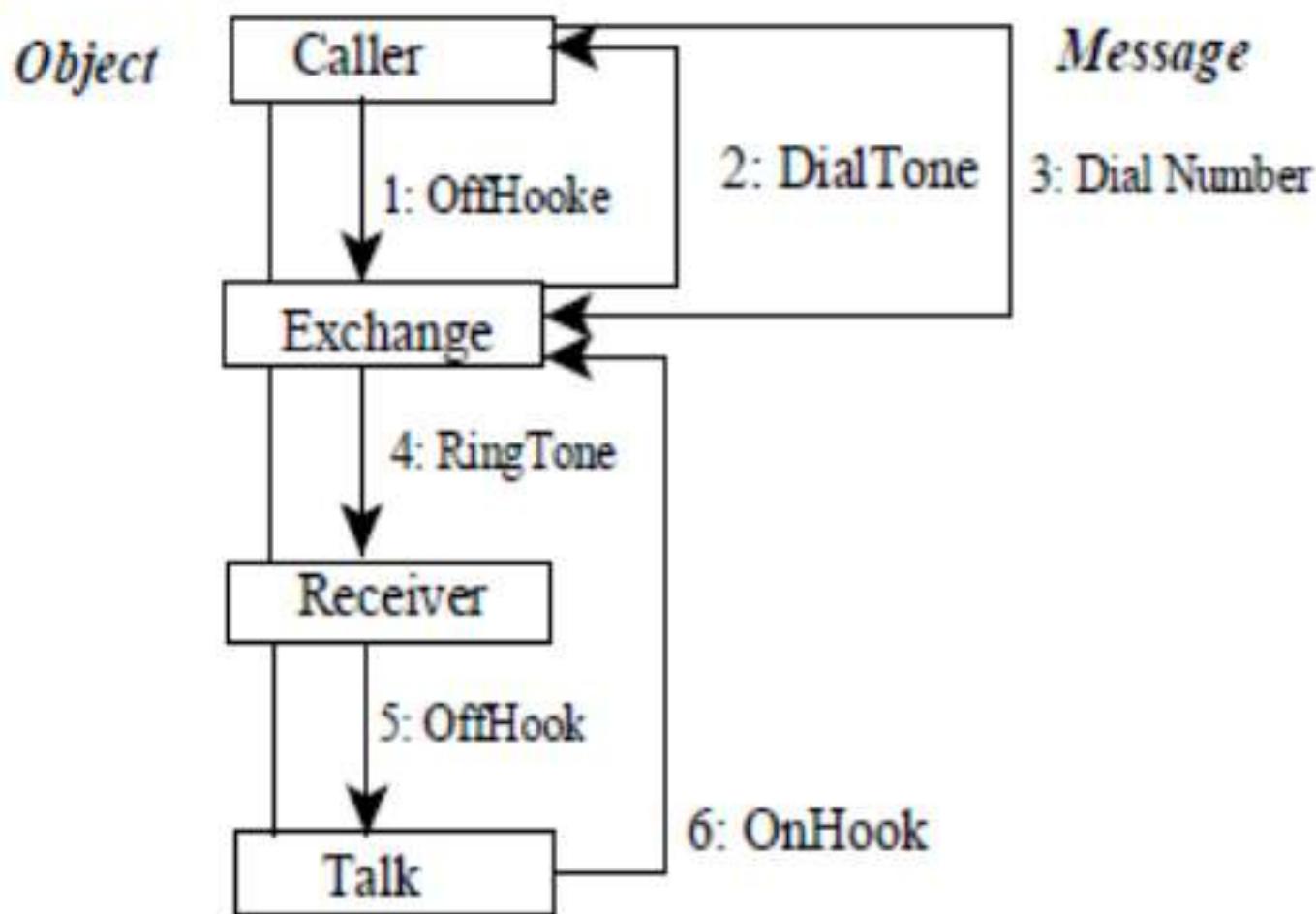
Private University Estd. in Karnataka State by Act No. 41 of 2013



What is a Collaboration Diagram

- Collaboration diagrams illustrate interactions between objects
- The collaboration diagram illustrates messages being sent between classes and objects (instances).
- Collaboration diagrams express both the context of a group of objects (through objects and links) and the interaction between these objects (by representing message broadcasts)

Telephone Call

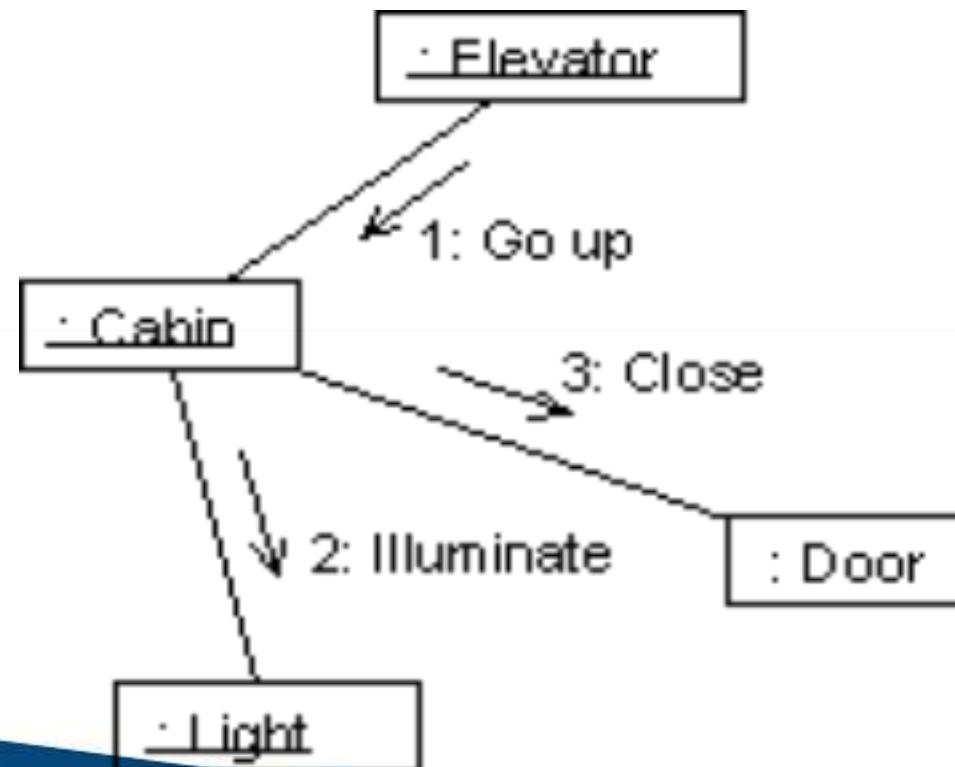


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Ex. Collaboration Diagram



Purpose of the Collaboration Diagram

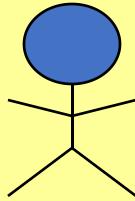
- They are very useful for visualizing the relationship between objects collaborating to perform a particular task
- They provide a good view – albeit static - view of interaction between objects which may be difficult to see at the class level

-
- Represents a Collaboration and Interaction
 - Collaboration-set of objects and their interactions in a specific context
 - Interaction-set of messages exchanged in a collaboration to produce a desired result

Collaboration Diagram Elements

- There are three primary elements of a collaboration diagram:
 - Objects
 - Links
 - Messages

Collaboration Diagram Syntax

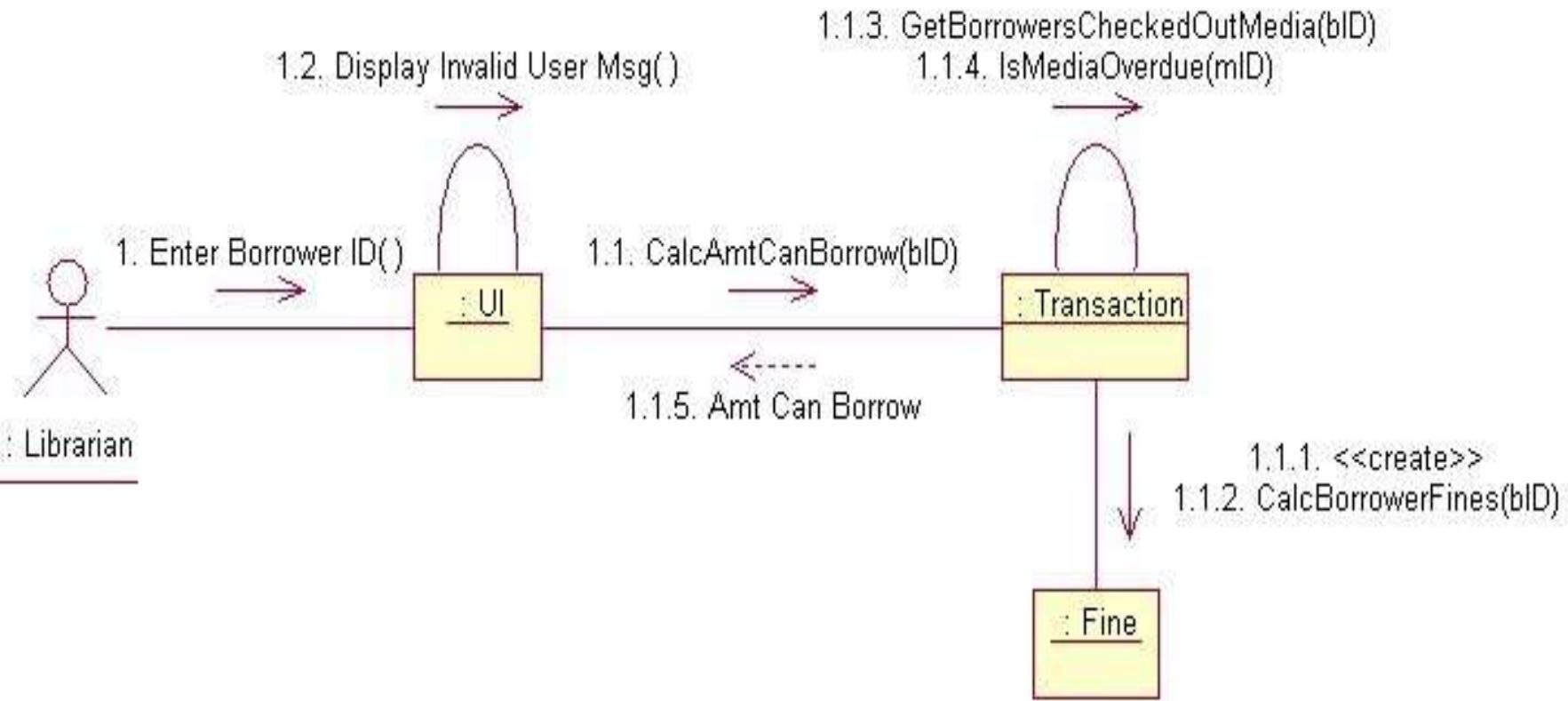
AN ACTOR	
AN OBJECT	<div style="background-color: orange; padding: 10px; text-align: center;">anObject:aClass</div>
AN ASSOCIATION	<hr/>
	<hr/> <u>aMessage()</u> →

Objects

- **Objects** -rectangles containing the object signature-
- object signature:
 - **object name : object Class**
 - object name (optional) - starts with lowercase letter
 - class name (mandatory) - starts with uppercase letter
- Objects connected by lines- actor can appear

Objects

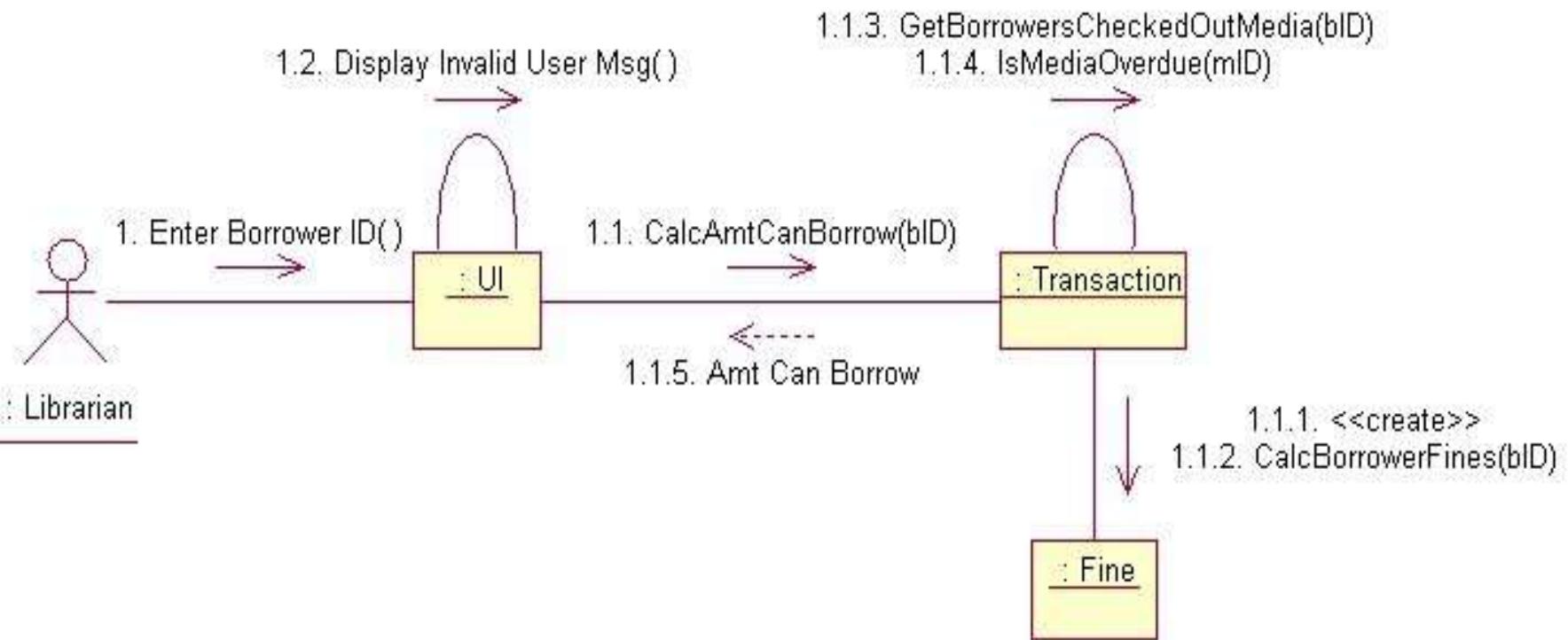
- Objects participating in a collaboration come in two flavors—supplier and client
- Supplier objects are the objects that supply the method that is being called, and therefore **receive** the message
- Client objects call methods on supplier objects, and therefore **send** messages



Transaction object acts as a Supplier to the UI (User Interface) Client object. In turn, the Fine object is a Supplier to the Transaction Client object.

Links

- The connecting lines drawn between objects are links
- They enable you to see the relationships between objects
- This symbolizes the ability of objects to send messages to each other
- A single link can support one or more messages sent between objects



The visual representation of a link is a straight line between two objects. If an object sends messages to itself, the link carrying these messages is represented as a loop icon. This loop can be seen on both the UI object and the Transaction object.

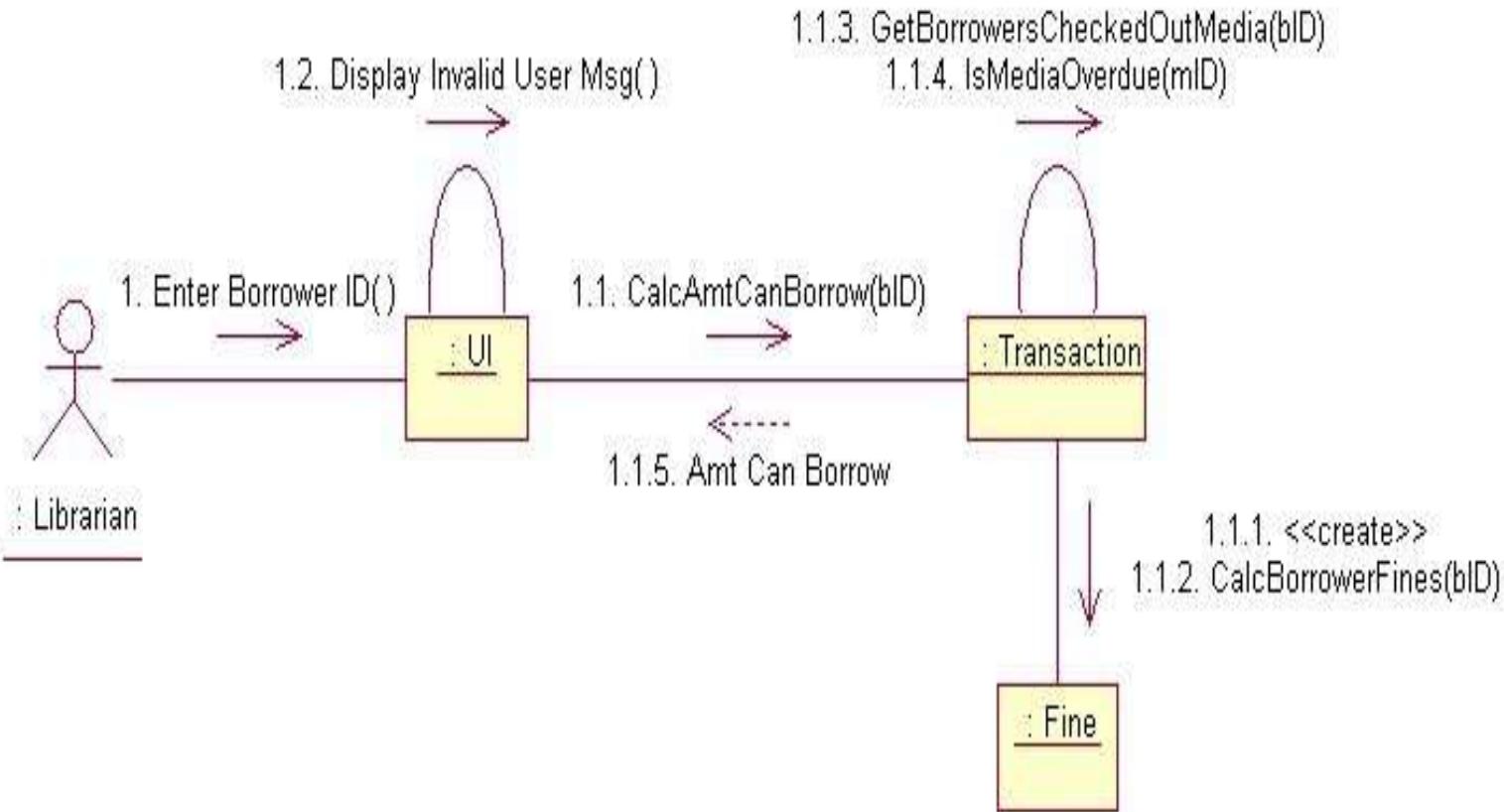
Messages

- An interaction is implemented by a group of objects that collaborate by exchanging messages
- Messages in collaboration diagrams are shown as arrows pointing from the Client object to the Supplier object.
- Typically, messages represent a client invoking an operation on a supplier object

Messages

- Message icons have one or more messages associated with them
- Messages are composed of message text prefixed by a sequence number
- Time is not represented explicitly in a collaboration diagram, and as a result the various messages are numbered to indicate the sending order

Flow by Number



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Flow by Numbers

1. Enter Borrower ID

1.1 CalcAmtCanBorrow

1.1.1 <<create>>

1.1.2 CalcBorrowerFines

1.1.3 GetBorrowersCheckedOutMedia

1.1.4 IsMediaOverdue

1.1.5 Amt Can Borrow

1.2 Display Invalid User Msg



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Message Flow Notation

Same as for sequence diagrams

Synchronous

Flow of control

Asynchronous

Return



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Iterating Messages

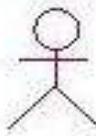
- Collaboration diagrams use syntax similar to sequence diagrams to indicate that either a message iterates (is run multiple times) or is run conditionally
 - An asterisk (*) indicates that a message runs more than once
 - Or the number of times a message is repeated can be shown by numbers (for example, 1..5)

Conditional Messages

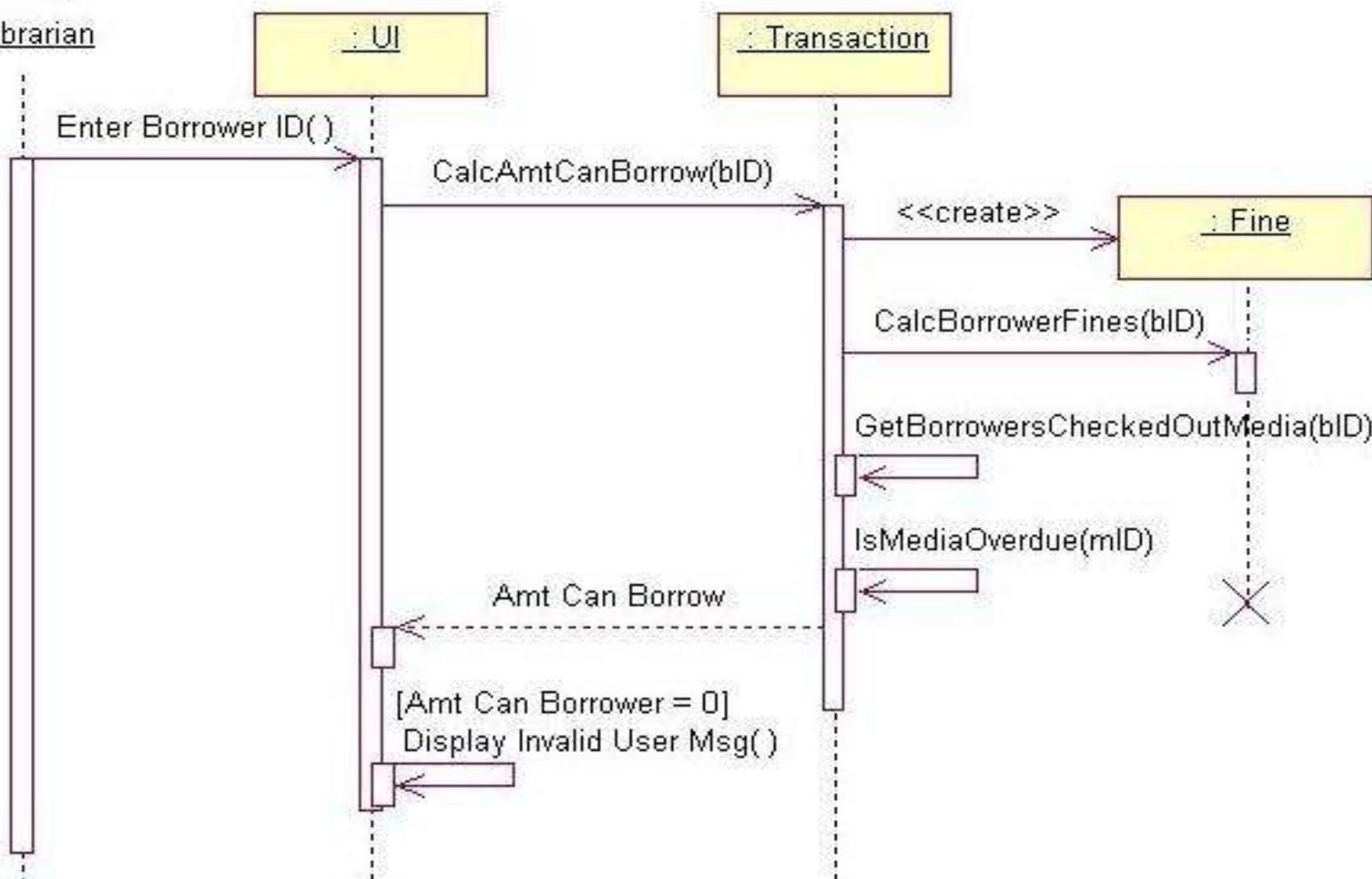
- To indicate that a message is run conditionally, prefix the message sequence number with a conditional [guard] clause in brackets [x = true]:
[IsMediaOverdue]
- This indicates that the message is sent only if the condition is met

Collaboration vs Sequence Diagram

- In reality, sequence diagrams and collaboration diagrams show the same information, but just present it differently
- Not used as often as sequence diagrams but are closely related



: Librarian



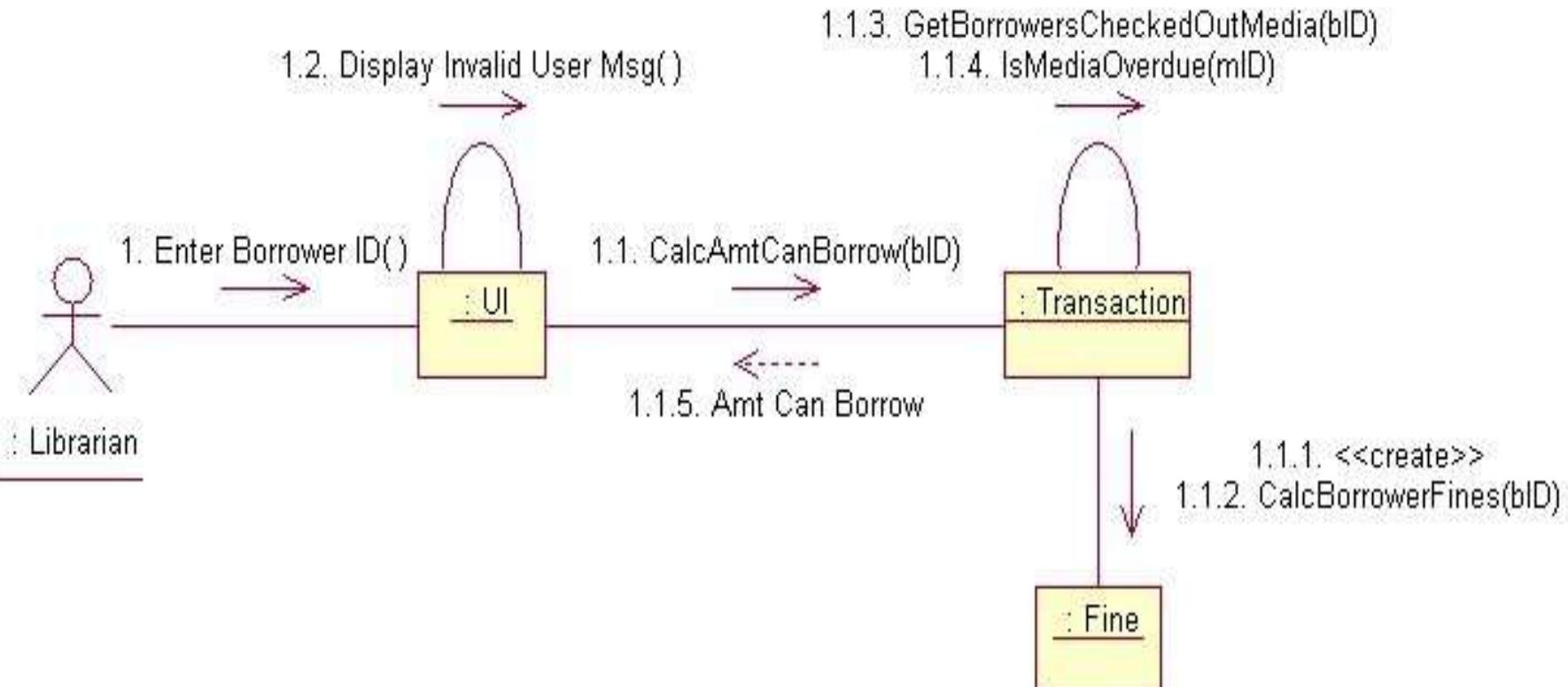
Sequence diagram is better at ‘time ordering’



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013

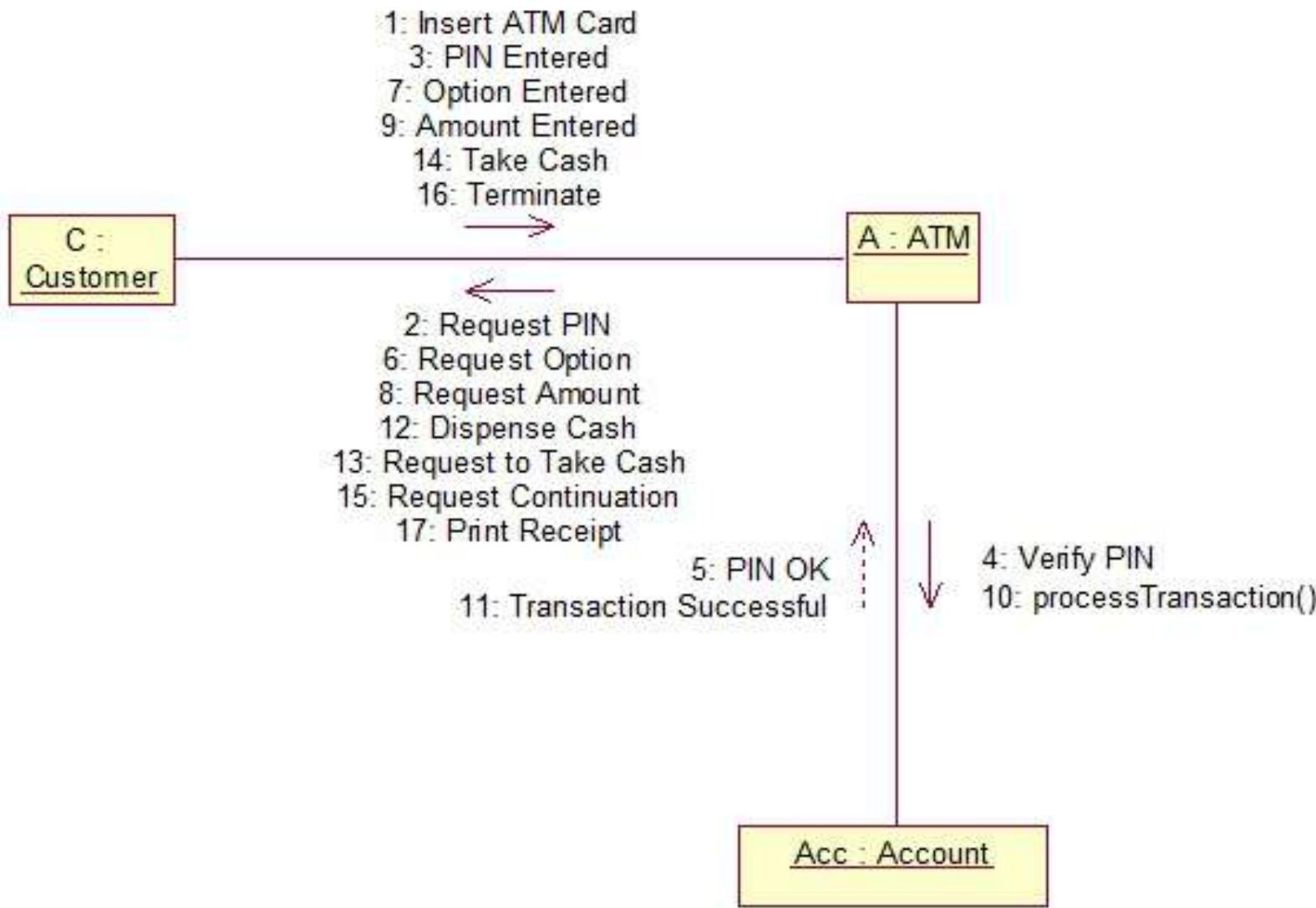




Collaboration diagram is better at showing the relationship between objects

Number of Messages

- In sequence diagrams, each message icon represents a single message.
- In collaboration diagrams, a message icon can represent one or more messages.
- Notice between the Transaction and Fine objects - there is a single message icon, but there are two messages (1.1.1 and 1.1.2) associated with the icon.

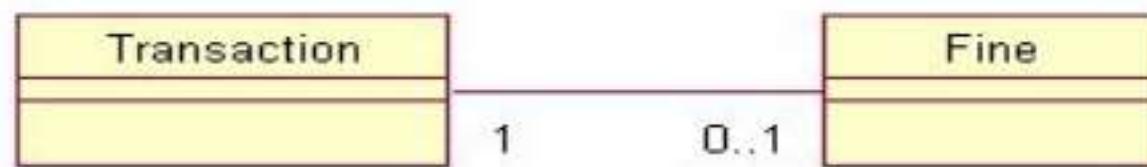


**PRESIDENCY
UNIVERSITY**

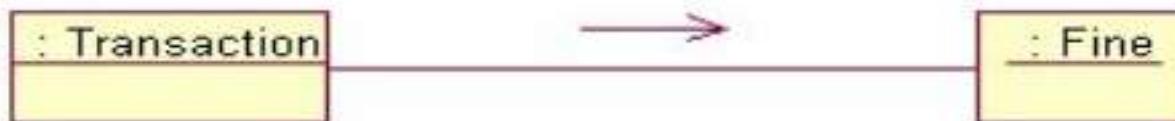
Private University Estd. in Karnataka State by Act No. 41 of 2013



Collaboration and Class Diagrams



1.1.1. CalcBorrowerFines(bID)



- Links in a collaboration diagram directly correlate to associations between classes in a class diagram

Steps to Creating a Collaboration Diagram

1. Determine the scope of the diagram- the use case it relates to
2. Place the objects that participate in the collaboration on the diagram
 - Remember to place the most important objects towards the center of the diagram.
3. If a particular object has a property or maintains a state that is important to the collaboration, set the initial value of the property or state

Steps to Creating a Collaboration Diagram

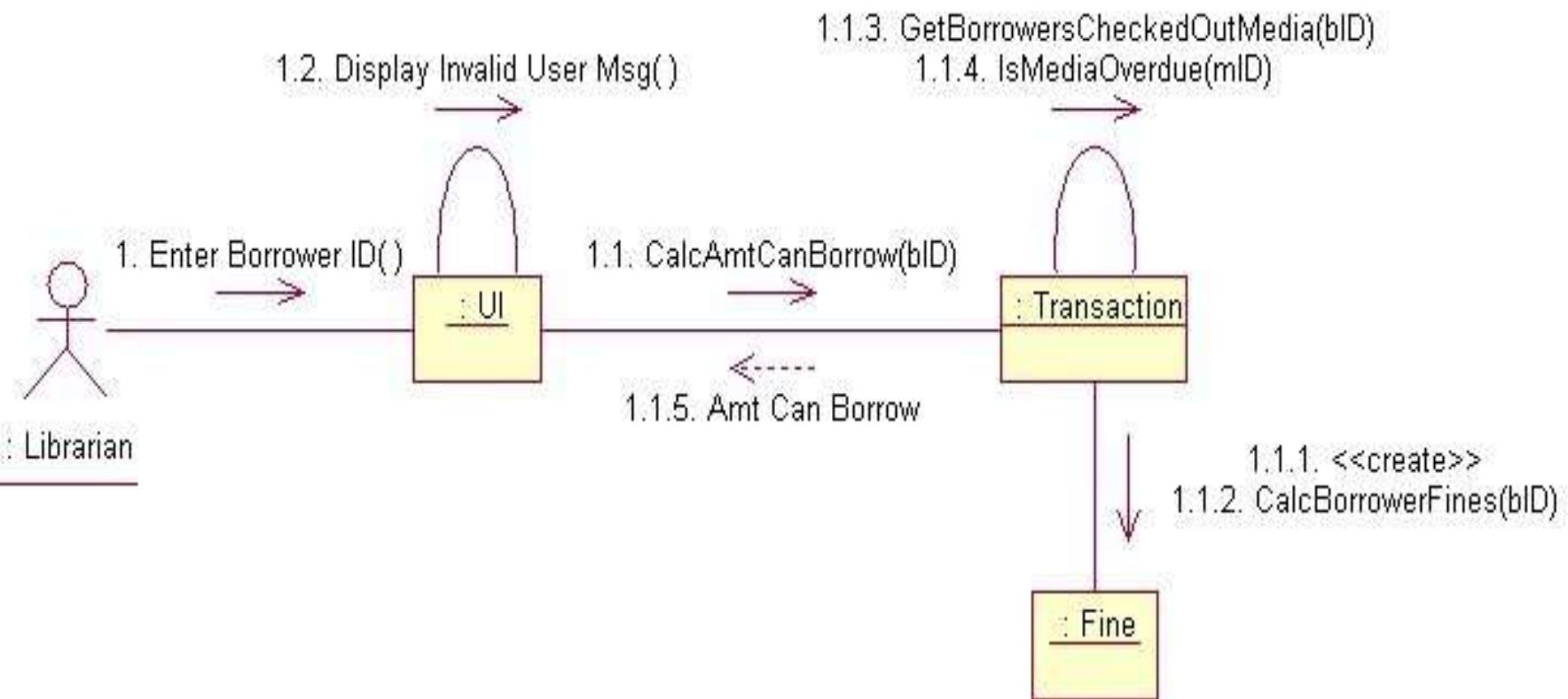
4. Create links between the objects
5. Create messages associated with each link
6. Add sequence numbers to each message corresponding to the time-ordering of messages in the collaboration

Creation and Deletion

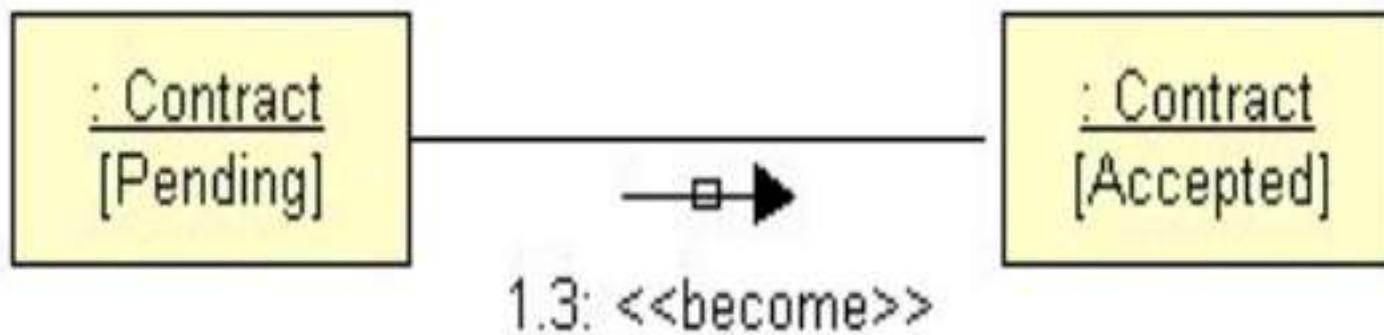
- Unlike sequence diagrams, you don't show an object's lifeline in a collaboration diagram
- If you want to indicate the lifespan of an object in a collaboration diagram, you can use create and destroy messages to show when an object is instantiated and destroyed

Objects Changing State

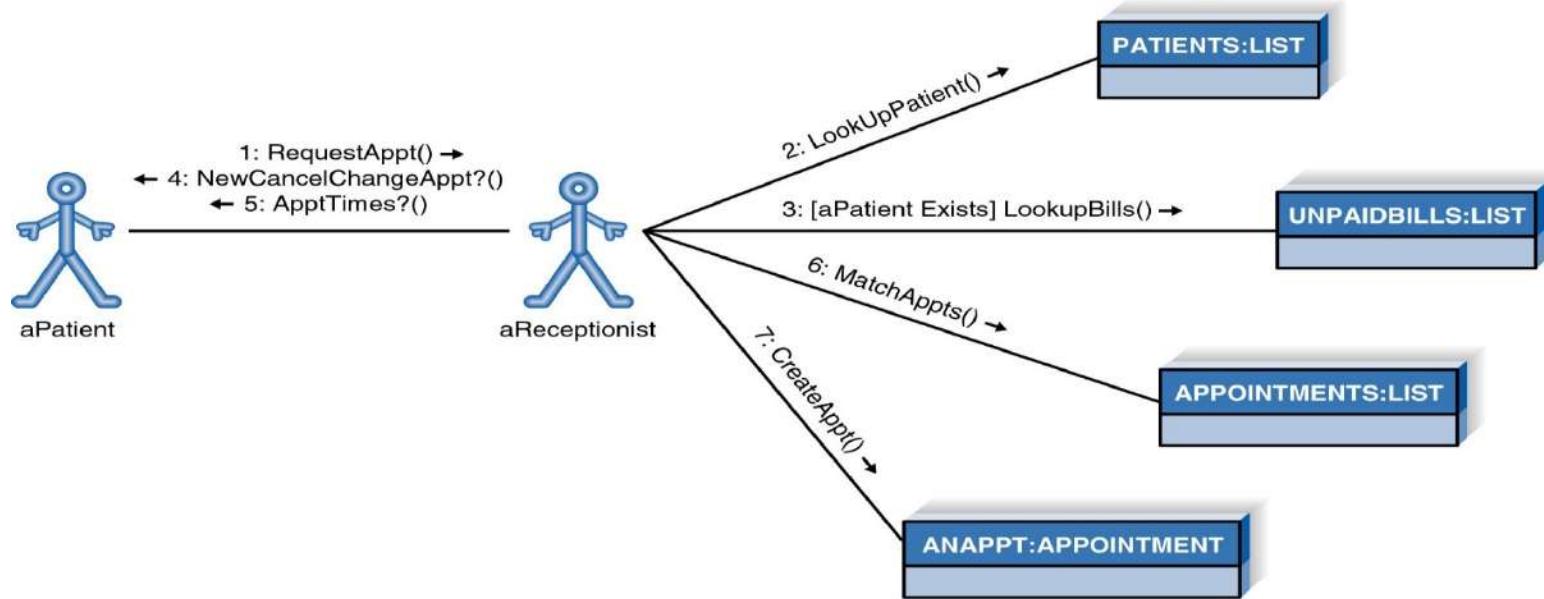
- State of an object can be indicated
- Initial state is indicated with <<create>>
- If an object changes significantly during an interaction, you can add a new instance of the object to the diagram, draw a link between them and add a message with the stereotype <<become>>



Change State of an Object



Example Collaboration Diagram



Dennis: SAD

Fig: 8-6 W-33 100% of size
Fine Line Illustrations (516) 501-0400

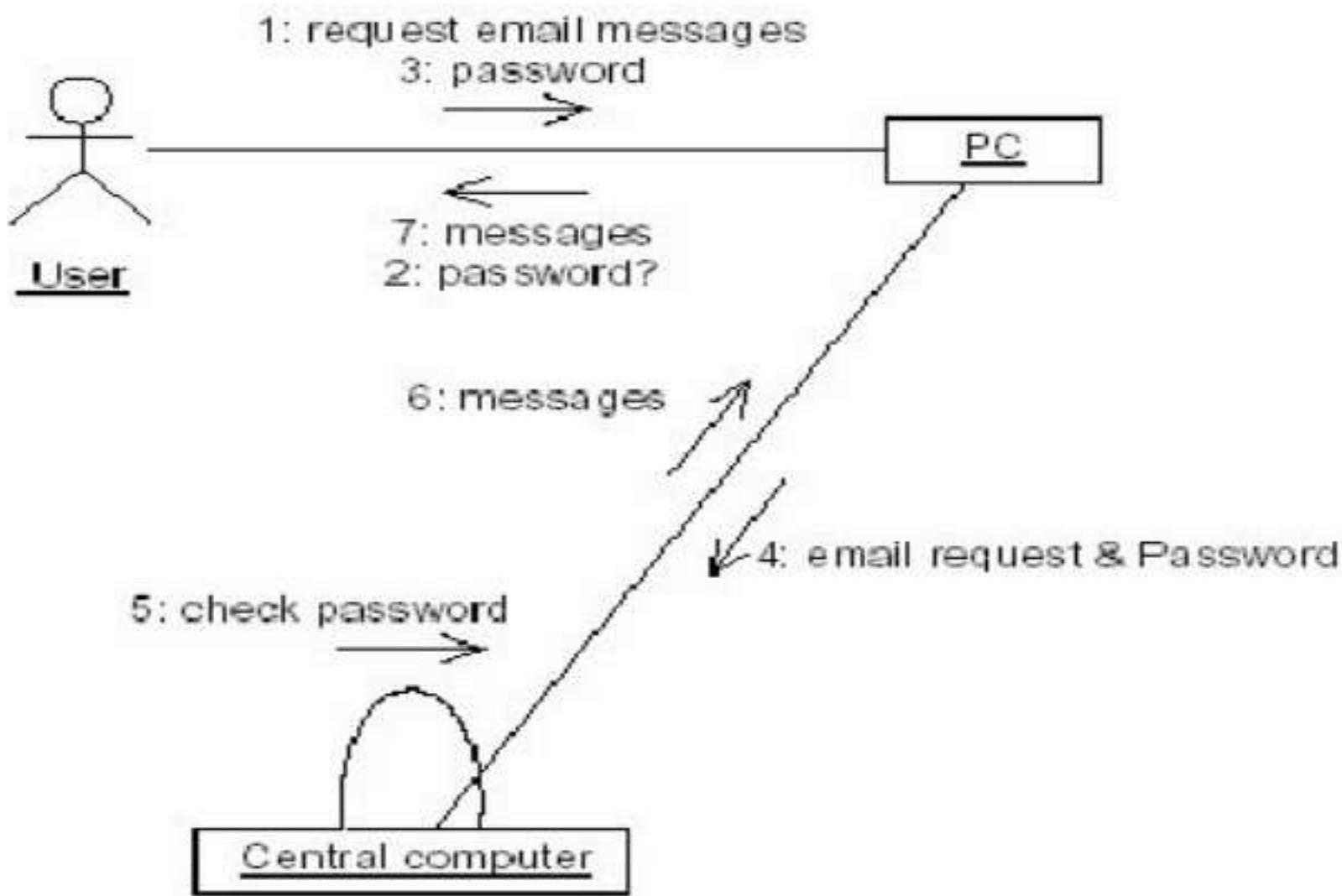
Draw Collaboration diagram for the scenario ‘reading an email message’.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Reference

- <http://www.devx.com/codemag/articles/2002/mayjune/collaborationdiagrams/codemag-2.asp>

Implementation Diagram



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





Implementation diagrams

- These diagrams show the implementation phase of systems development.
- Such as the source code structure and the run-time implementation structure.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





Implementation diagrams (Con't)

- **There are two types of implementation diagrams:**
 - Component diagrams show the structure of the code itself.
 - Deployment diagrams show the structure of the run-time system.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Strategies:

- Hardware
- Software
- Global control flow
- Data structures(persistent data storage)
- Access control policy
- Boundary conditions



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Deployment diagram

- UML Deployment Diagram Deployment diagrams are used to visualize the topology of the physical components of a system where the software components are deployed.
- So deployment diagrams are used to describe the static deployment view of a system.
- Deployment diagrams consist of nodes and their relationships.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013

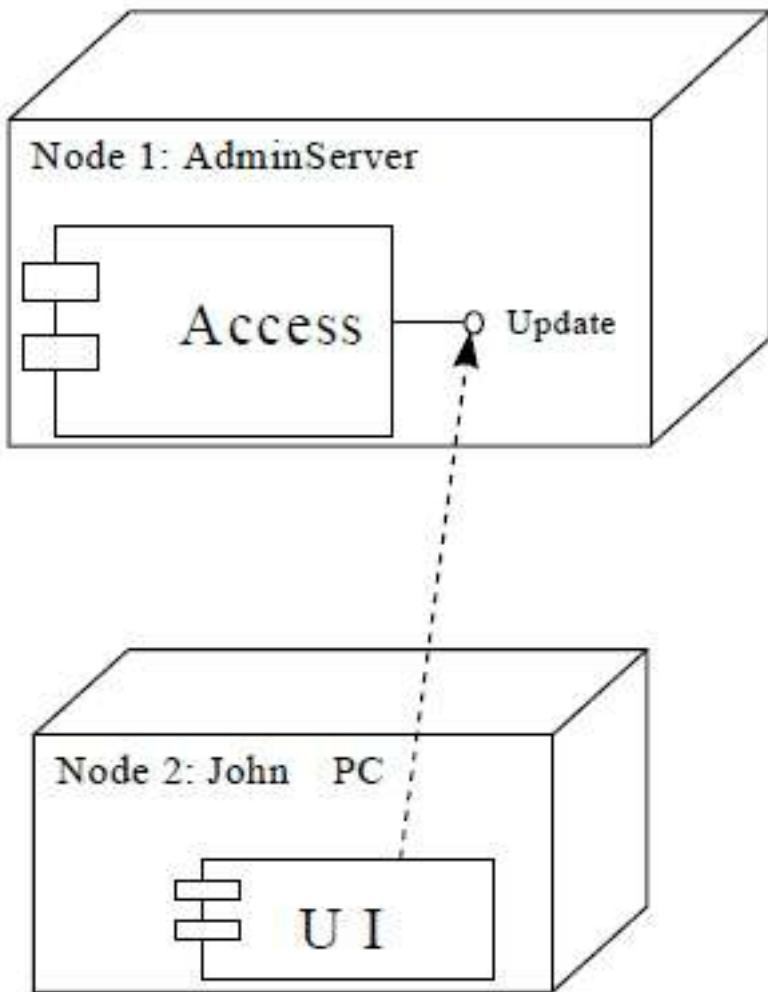


Purpose:

- Visualize hardware topology of a system.
- Describe the hardware components used to deploy software components.
- Describe runtime processing nodes.



Deployment Diagram



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



So the usage of deployment diagrams can be described as follows:

- To model the hardware topology of a system.
- To model embedded system.
- To model hardware details for a client/server system.
- To model hardware details of a distributed application.
- Forward and reverse engineering.

Deployment diagram commonly contains:

- Nodes
- Connections
- Artifacts



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Nodes:

- A Node represents hardware or software element of a system.
- Node is shown as a three dimensional box.
- Nested nodes are allowed in deployment diagram.
- There are two types of nodes.
 - Device Nodes
 - Execution Nodes



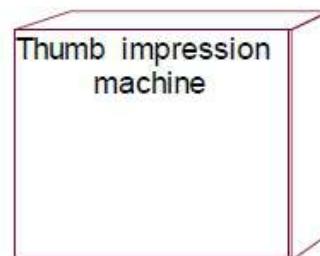
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Device Nodes

- Device Nodes are actual computing resources with processing memory and services for running software in them.
- For example mobile phones and PC's.
- They can be represented using stereotypes like <<server>>, <<storage>>, <<cd-rom>>, <<PC>> etc.



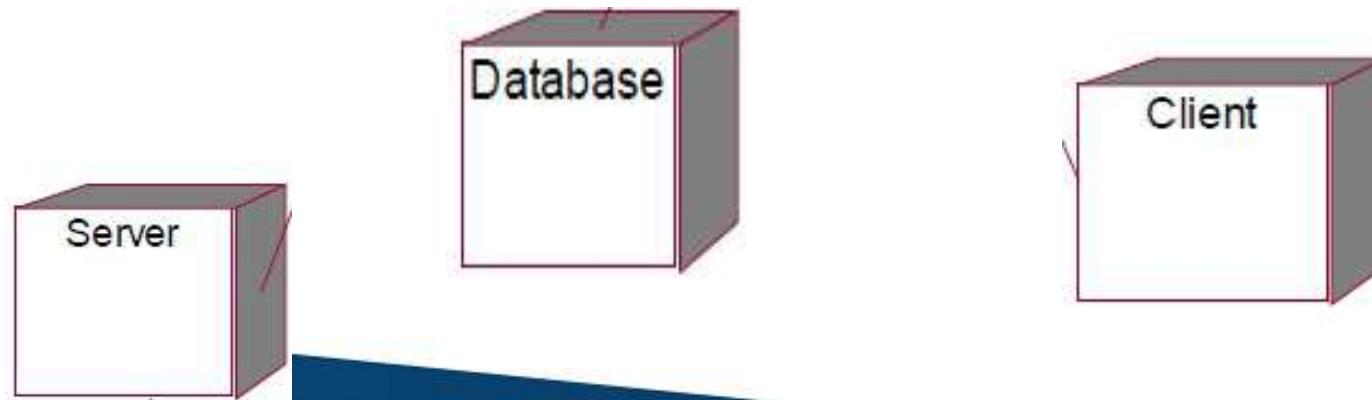
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Execution Nodes

- Execution Nodes are software computing resource which itself are present within the node.
- They may provide services for the host or are capable of executing other software resources.

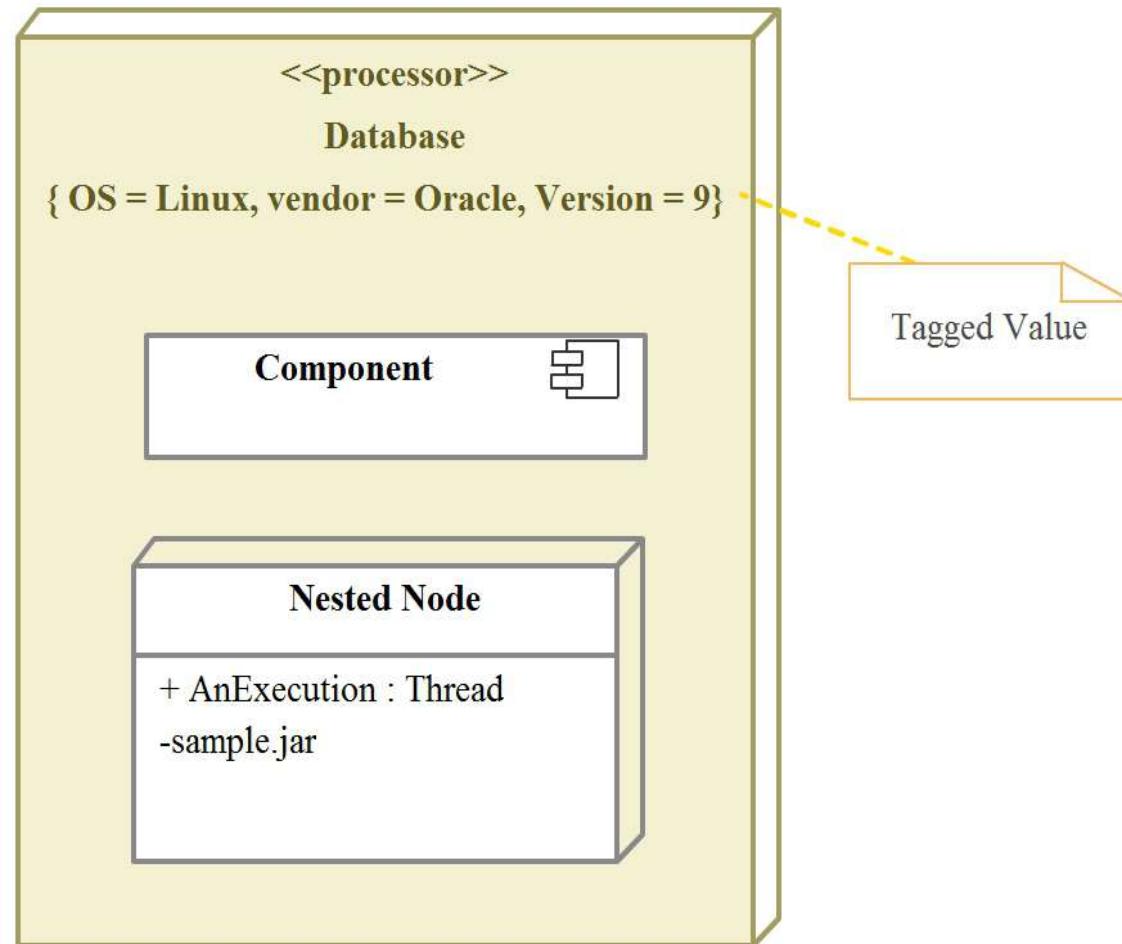


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Nested Nodes



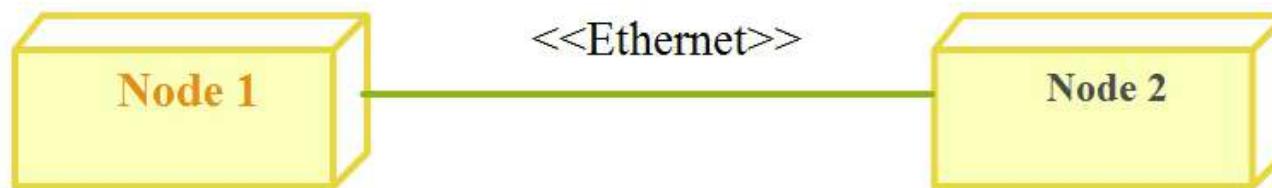
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Connectors

- Connectors are similar to associations and dependencies that are used in UML.
- Connectors are lines drawn between nodes, they are used as communication medium between nodes.
- Name of the physical connection can be represented using stereotypes. For Example <<Ethernet>>, <<message bus>> etc.

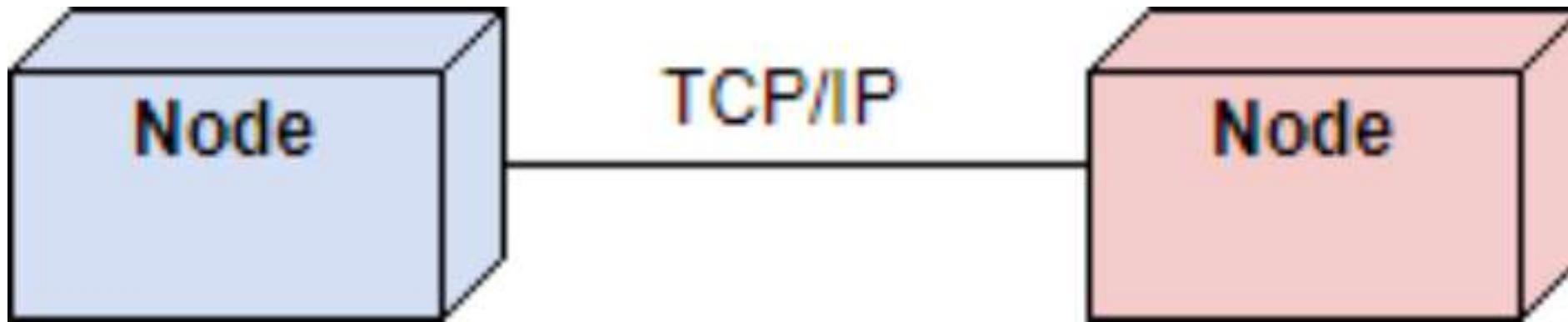


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Connectors



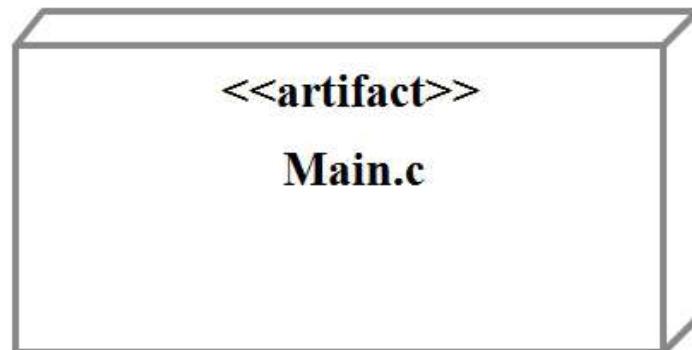
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Artifacts

- Artifacts are product of a software development process.
- They can range from models, files to documents.
- Artifacts are denoted by a rectangular box with <<artifact>> stereotype and a file icon in the box.

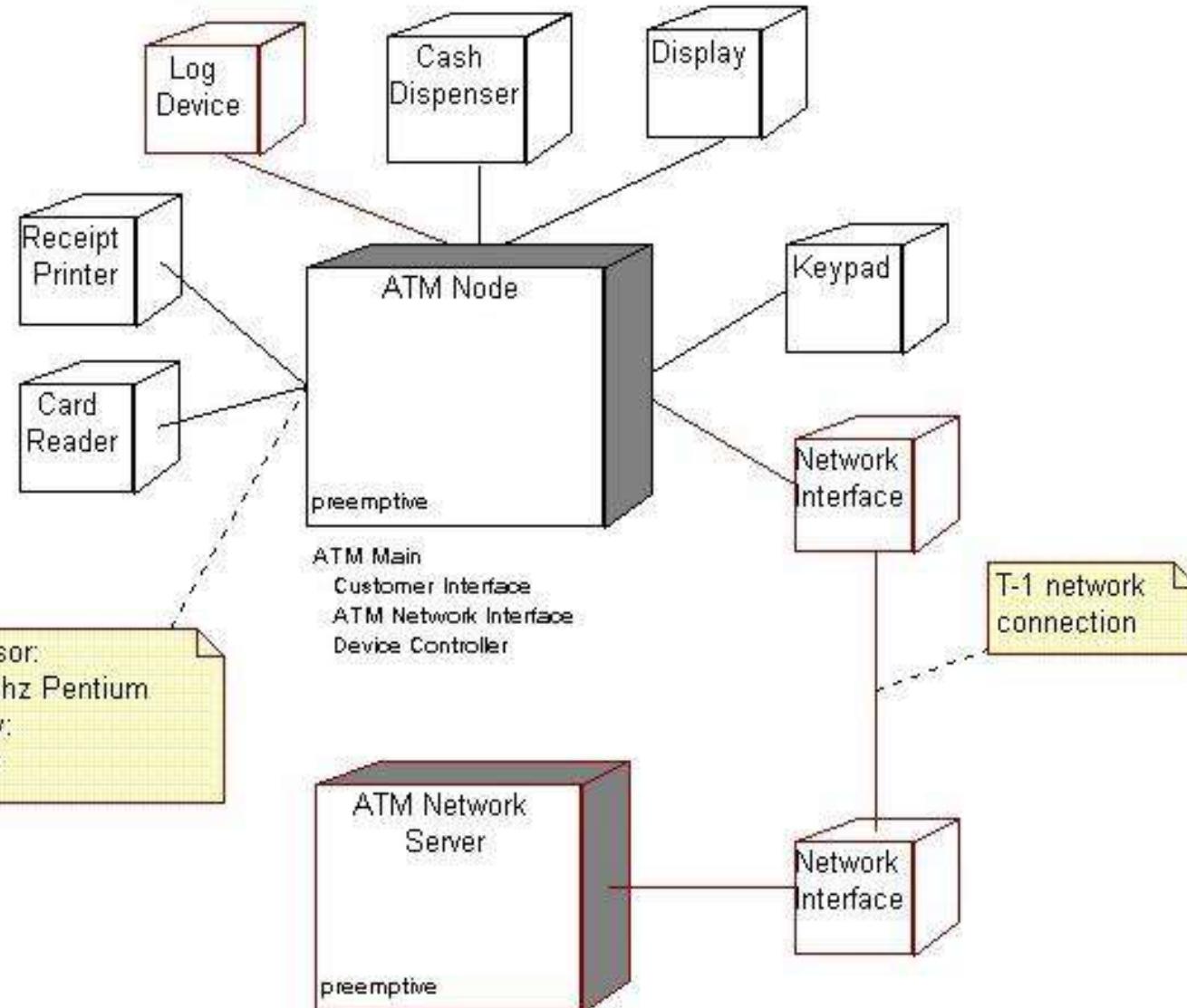


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



ATM

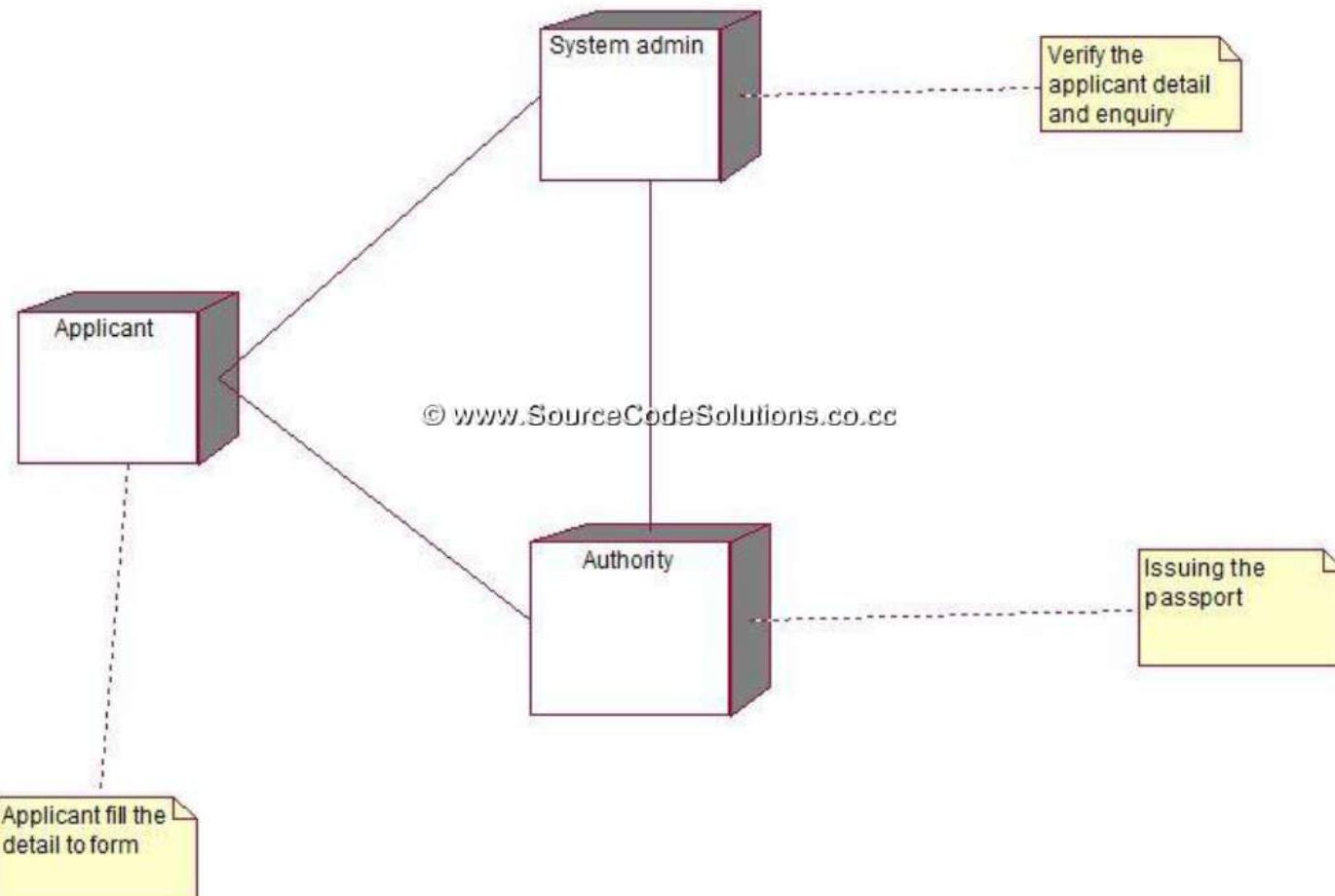


**PRESIDENCY
UNIVERSITY**



Private University Estd. in Karnataka State by Act No. 41 of 2013

Deployment diagram for Passport automation system

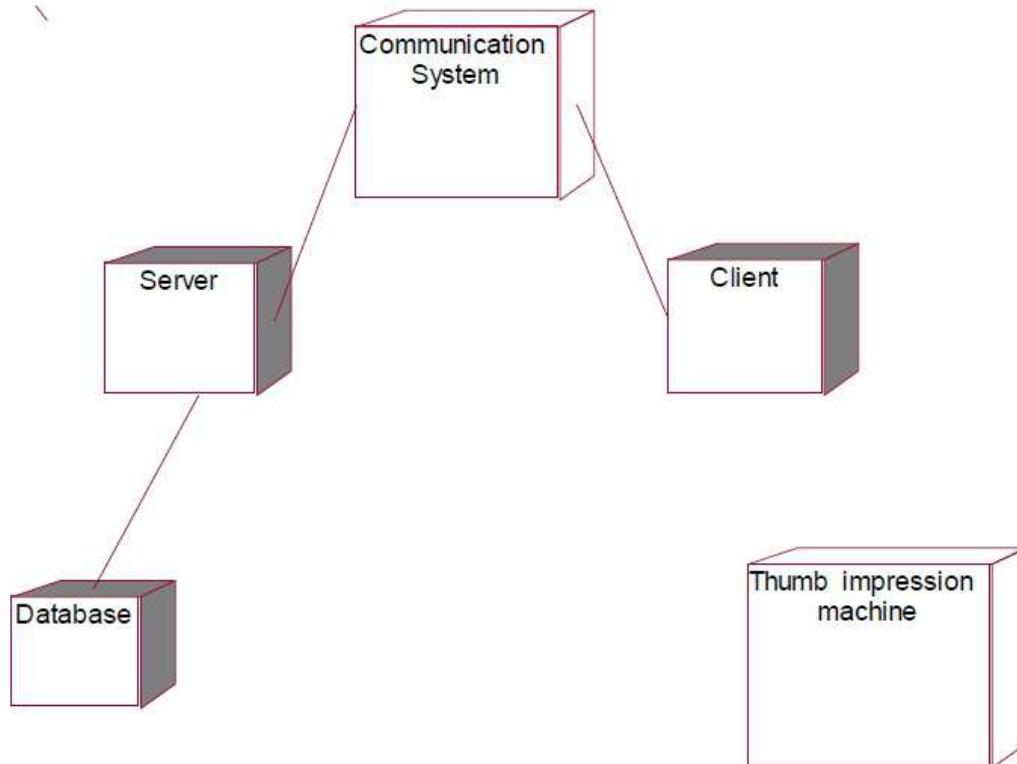


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Draw a Deployment diagram for online national polling

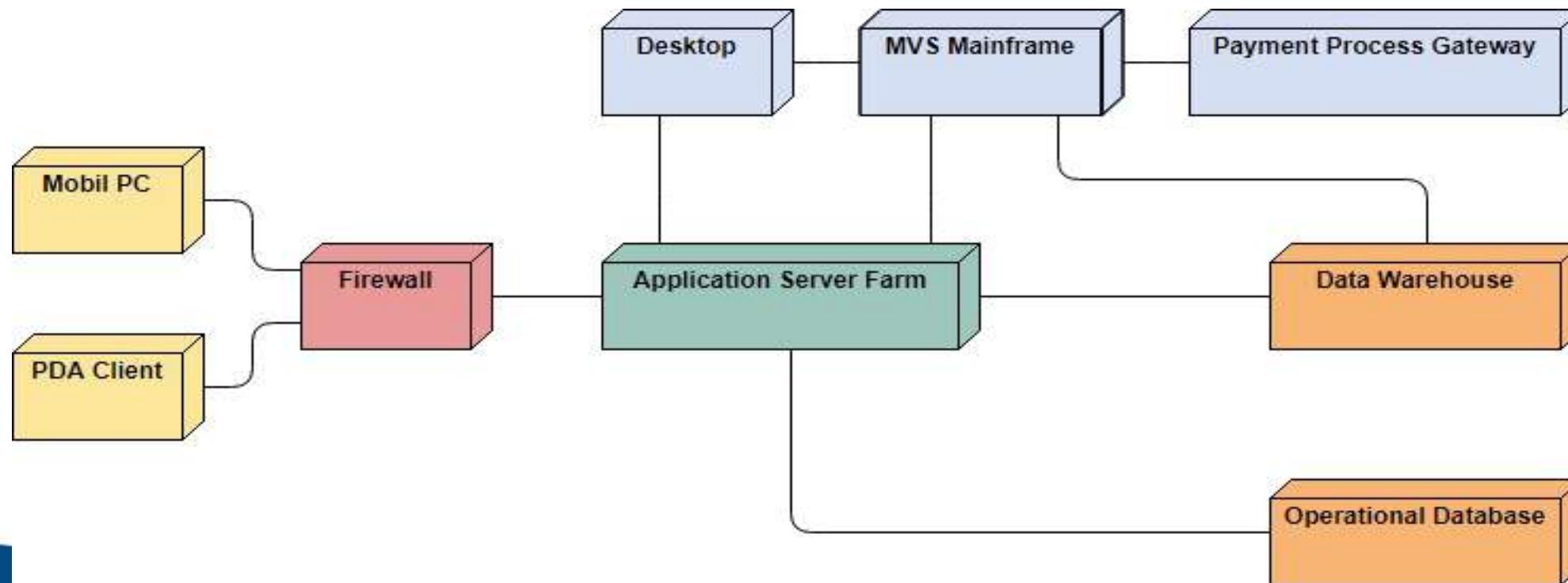


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Deployment Diagram Example - Corporate Distributed System

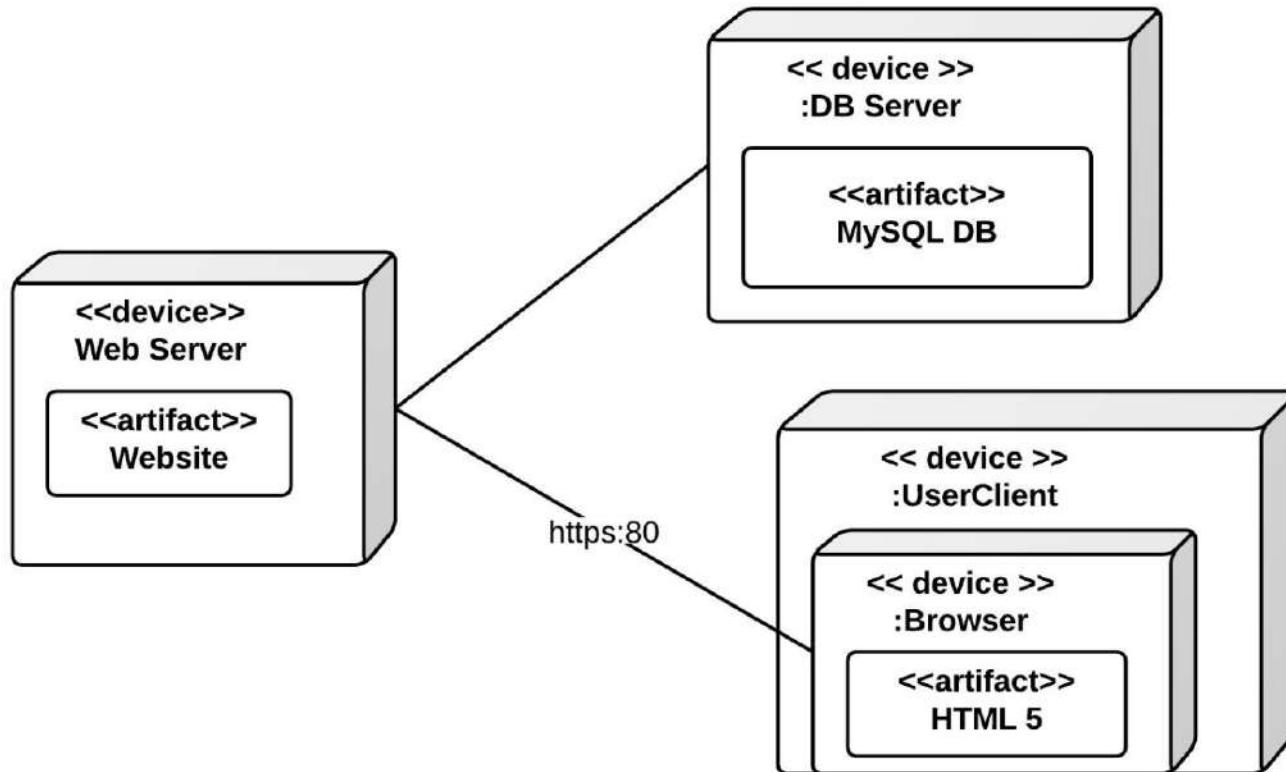


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Example for Deployment diagram

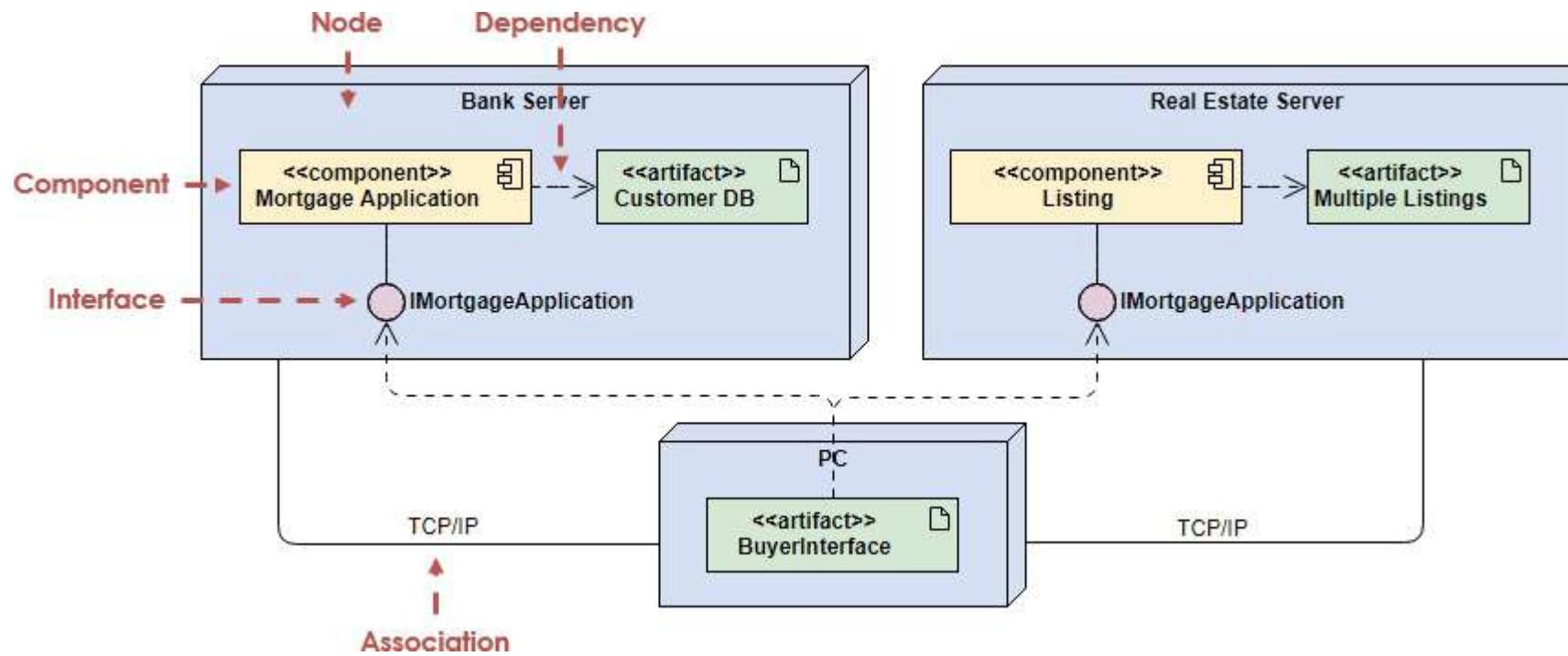


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Deployment diagram for Real estate transactions



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Component Diagrams



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



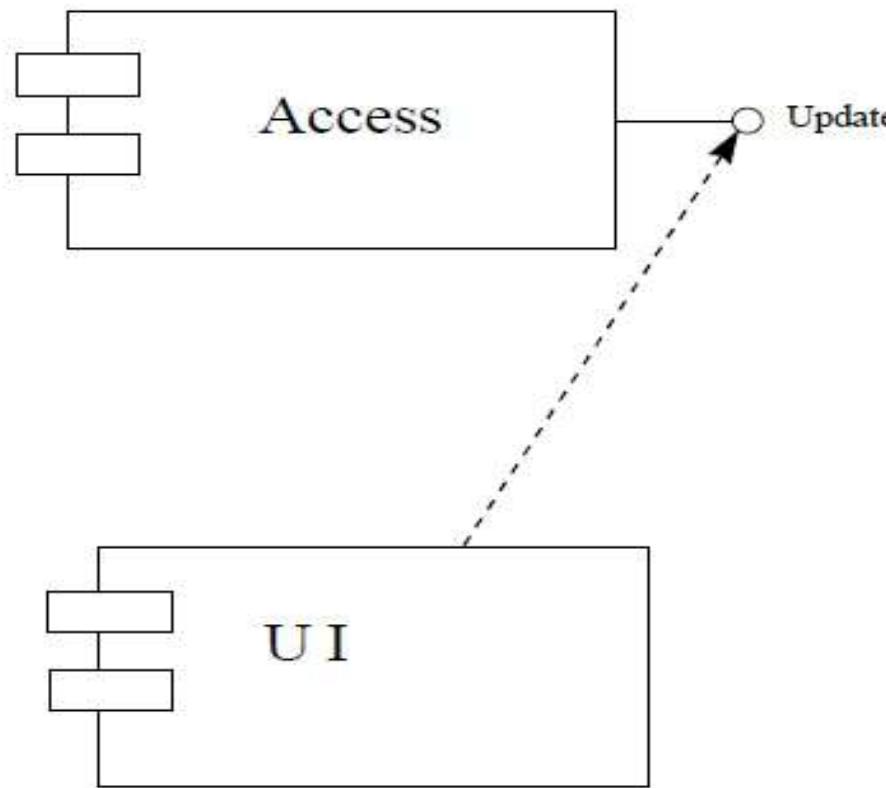
Component Diagrams

- A component is an encapsulated, reusable, and replaceable part of your software
- Reducing and defying coupling between software components
- Reusing existing components





Component diagrams



- The ability to identify software components (which are encapsulated, reusable and replaceable)
- Supports development strategies that use, e.g., COTS (Commercial-On-The-Shelf) components.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Component Diagrams

- Model physical software components and the interfaces between them
- Show the structure of the code itself
- Can be used to hide the specification detail (i.e., information hiding) and focus on the relationship between components
- Model the structure of software releases; show how components integrate with current system design
- Model source code and relationships between les
- Specify the les that are compiled into an executable



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- Components have interfaces and context dependencies
- i.e., implementation-specific shown on diagram; use-context may be described elsewhere
- for example, documentation, use-cases, interaction diagrams, etc.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Component Notation

- A Component is a physical piece of a system,
- Such as a compiled object file, piece of source code, shared library or Enterprise Java Bean (EJB).



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



<<component>>
ComponentName

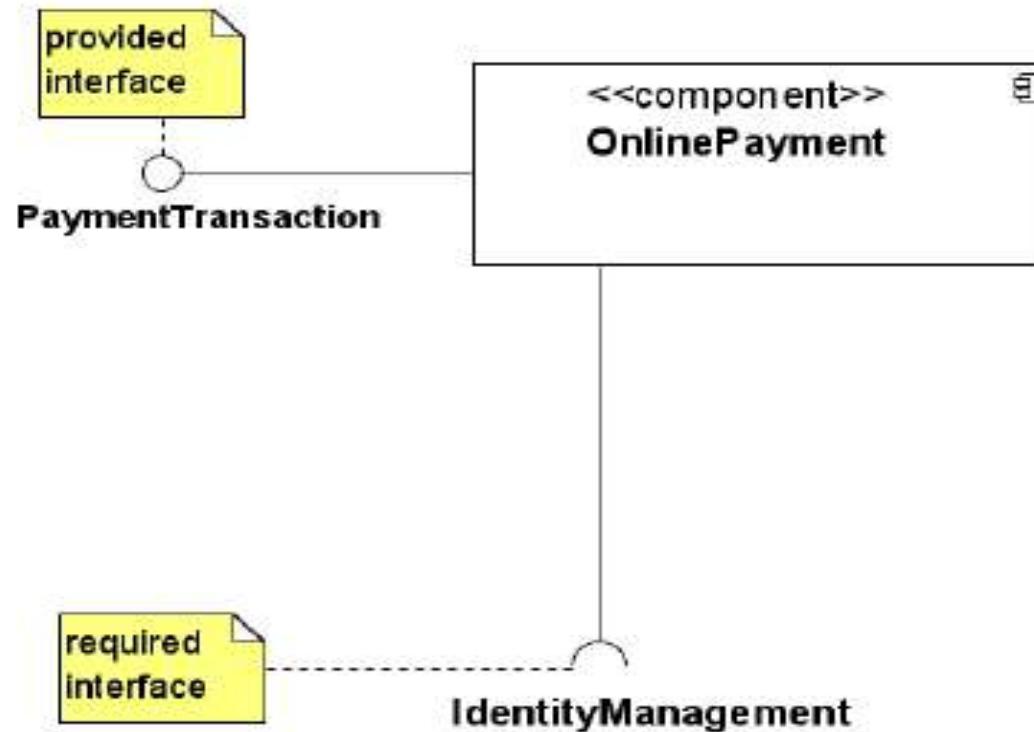


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Component Interfaces



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Component Interfaces

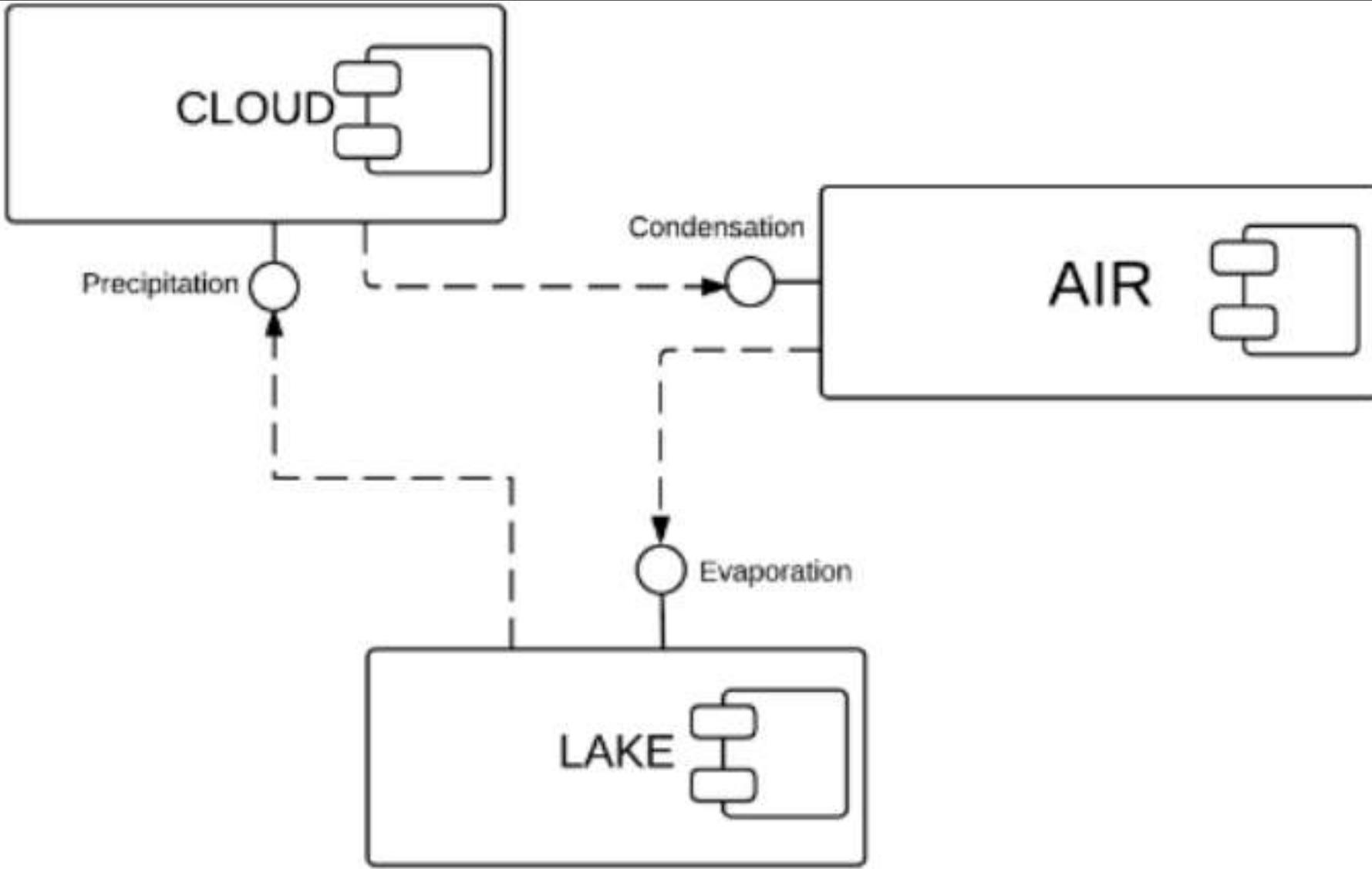
- A **provided interface** of a component is an interface that the component realizes
- A **required interface** of a component is an interface that the component needs to function



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



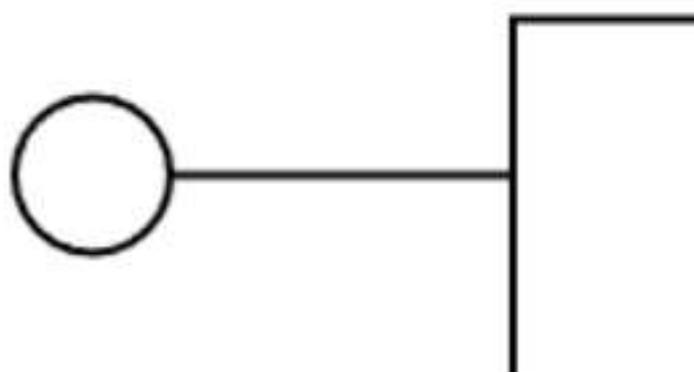


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Provided interface

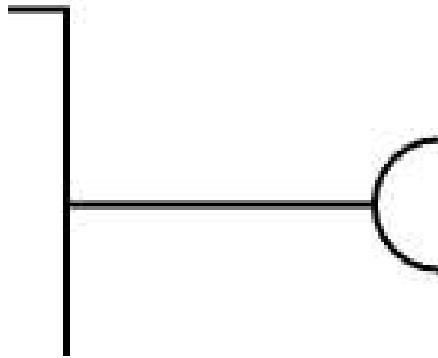


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Required interface

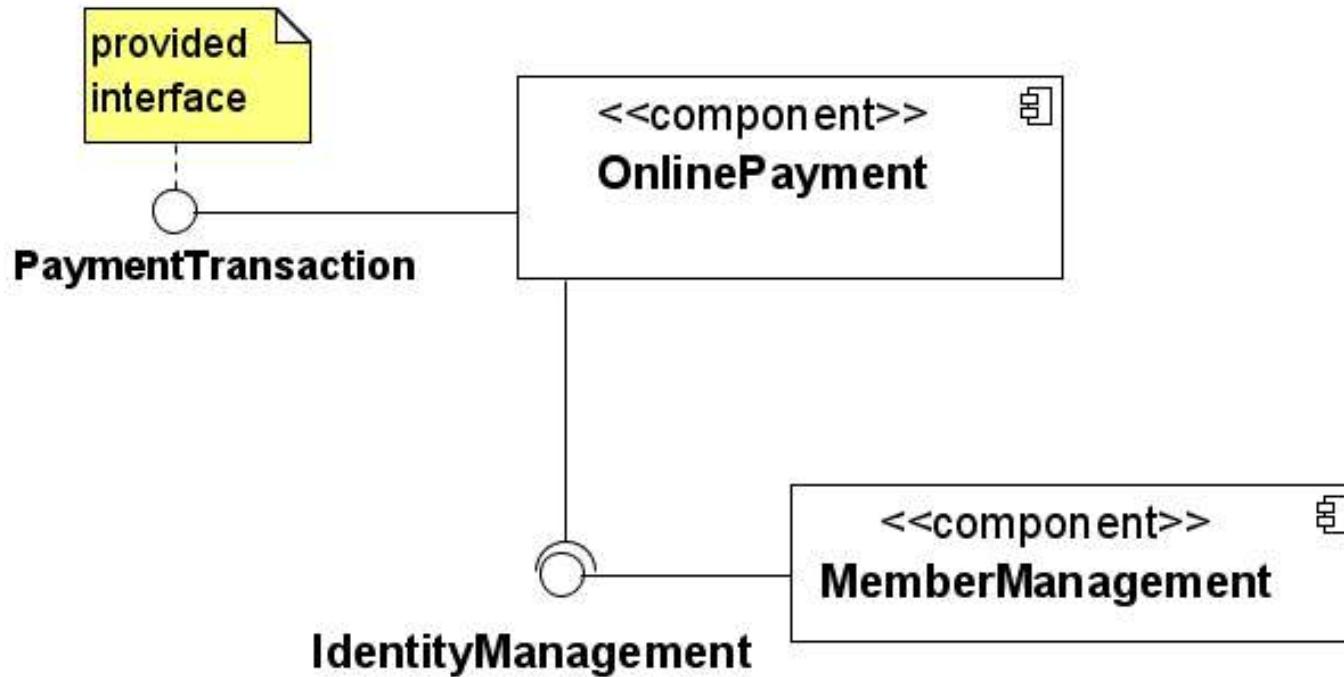


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



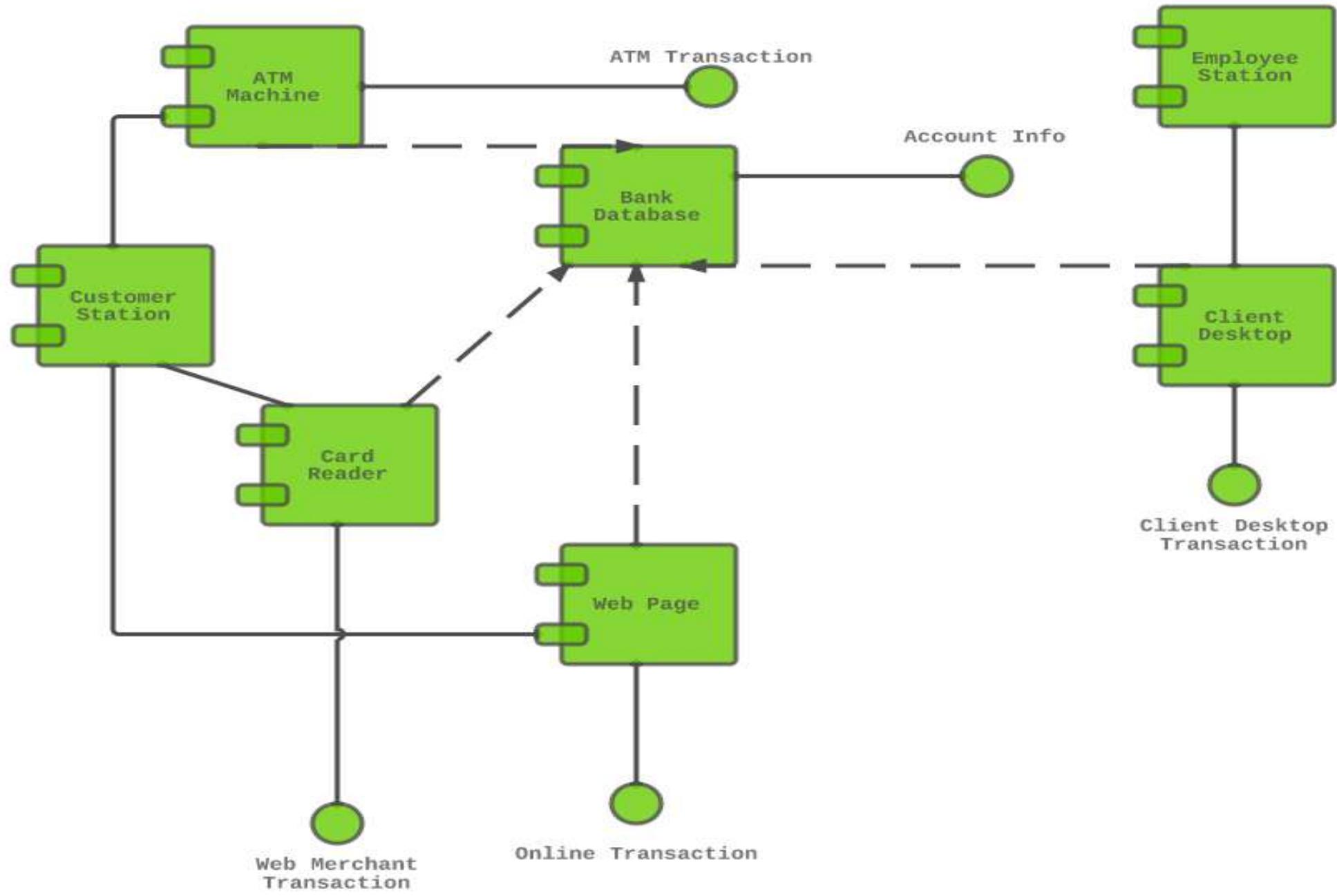
Component Interface



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





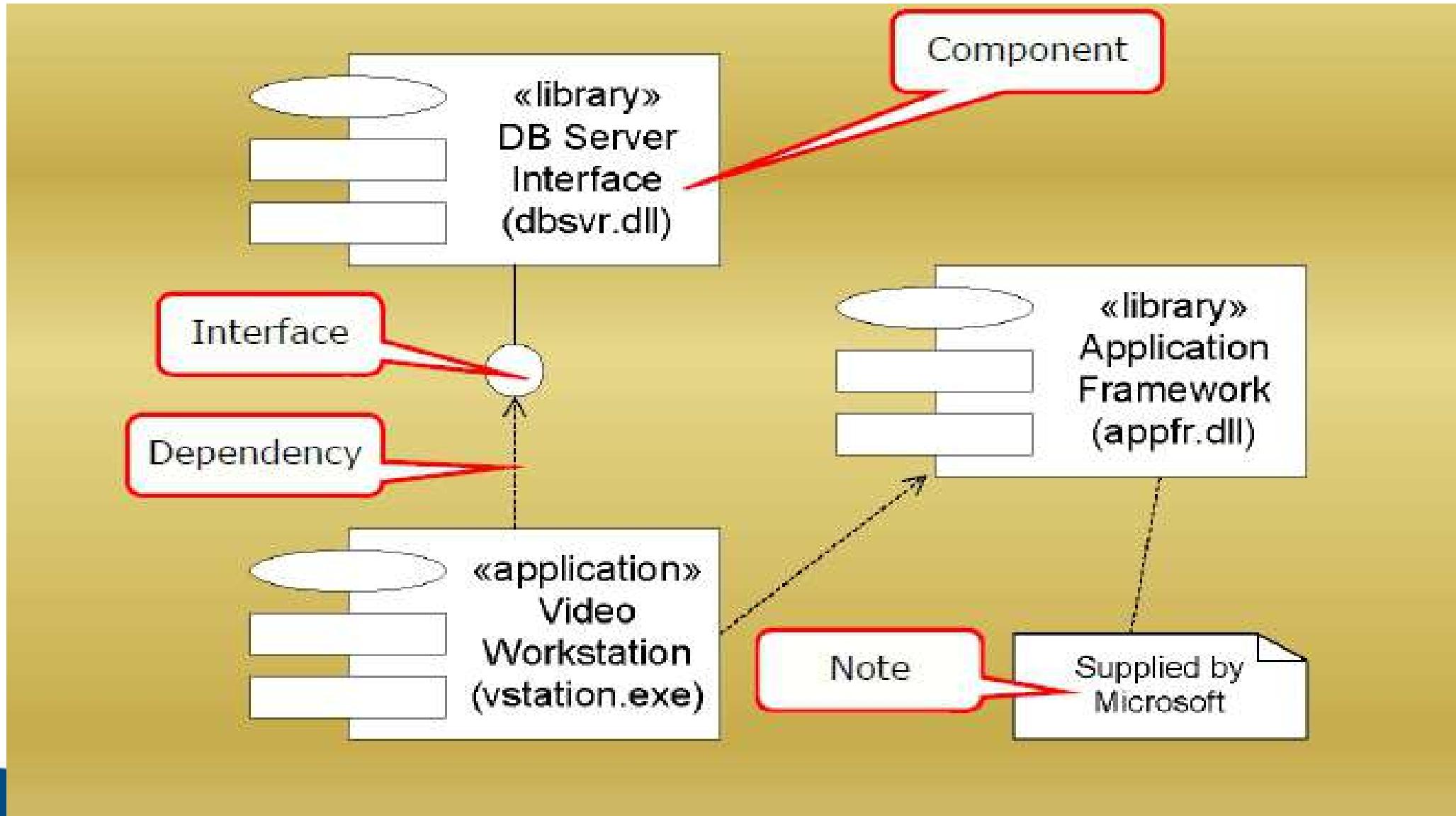
Video Workstation



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Online Bookshop

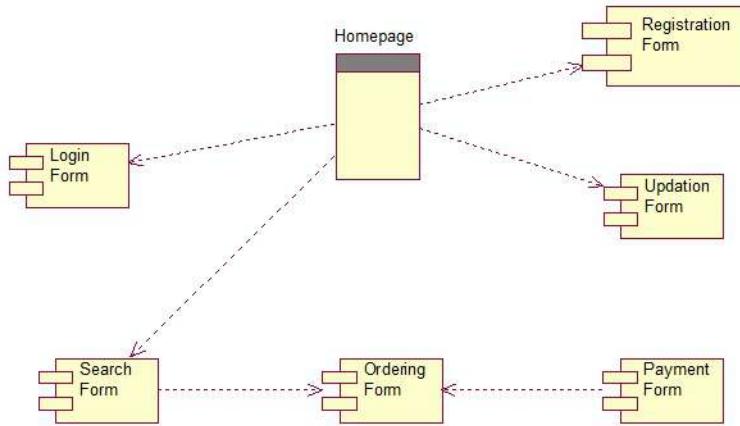


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Online bookshop



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Food Ordering System

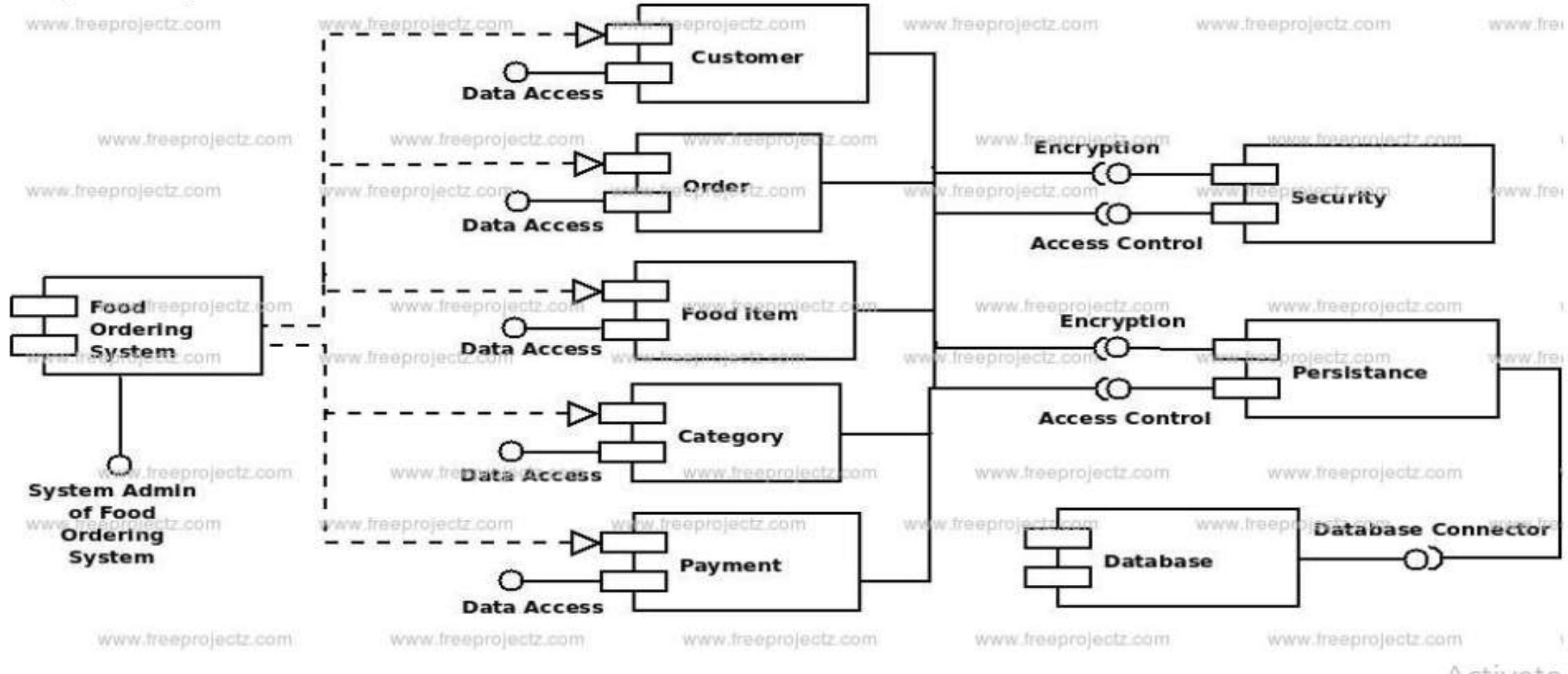


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Component Diagram:



Component Diagram of Food Ordering System



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Activate V
www.freeprojectz.com
Go to Setting

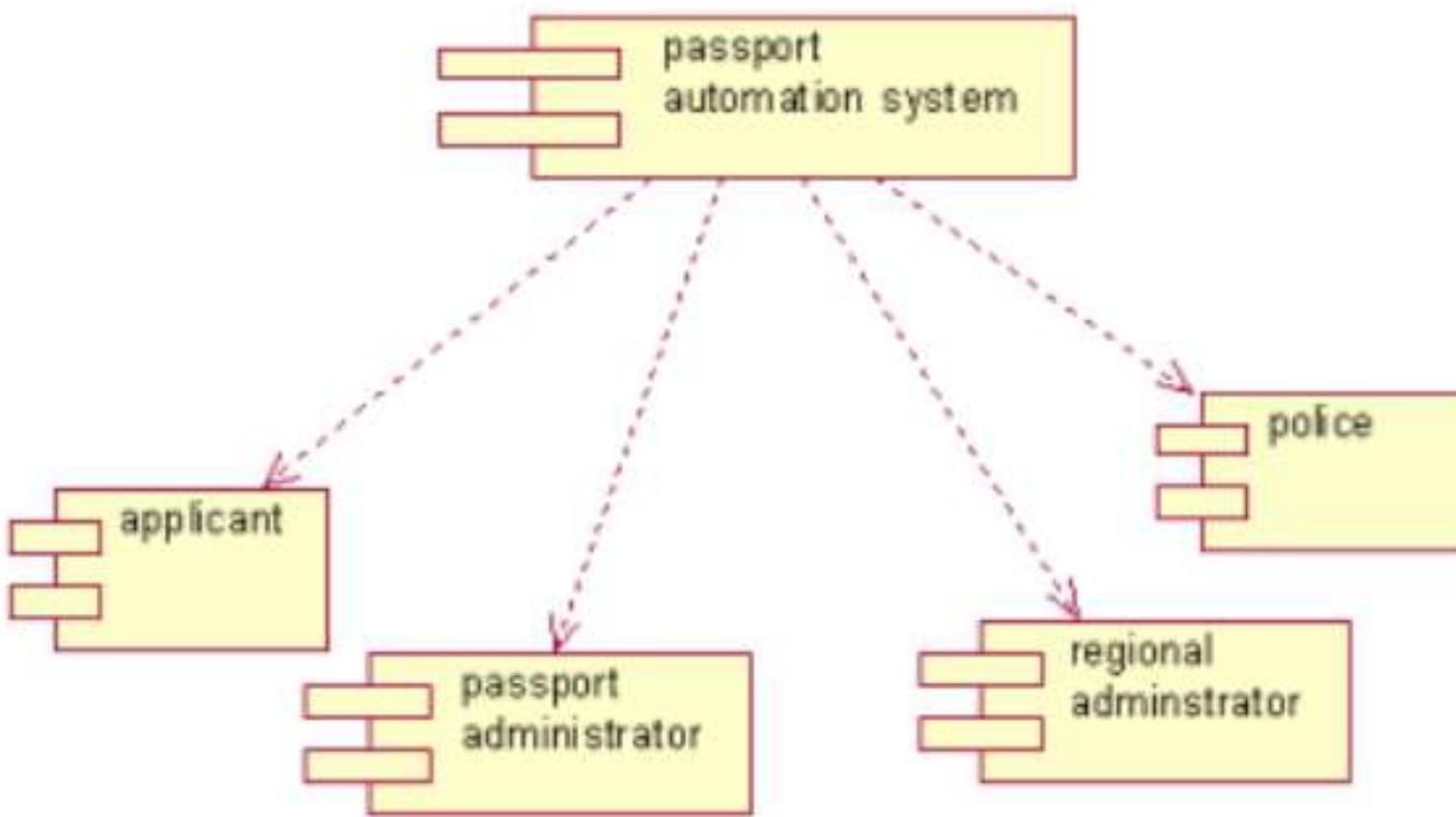
Passport Automation System



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



State Chart Diagram



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Introduction

- They define different states of an object during its lifetime. States are defined as a condition in which an object exists and it changes when some event is triggered
- These states are changed by events.
- So the most important purpose of Statechart diagram is to model life time of an object from creation to termination.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- ❑ Statechart diagrams are useful to model reactive systems
 - -Reactive systems can be defined as a system that
 - responds to external or internal events.
- ❑ Statechart diagram describes the flow of control from one state to another state.

Purpose

Following are the main purposes of using Statechart diagrams:

- To model dynamic aspect of a system.
 - To model life time of a reactive system.
 - To describe different states of an object during its life time.
- Define a state machine to model states of an object.

How to draw state charts

Before drawing a Statechart diagram we must have clarified the following points:

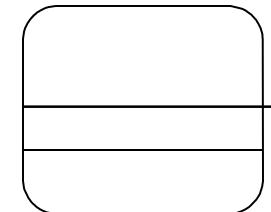
- Identify important objects to be analyzed. Identify
- the states.
- Identify the events.

Elements of state **chart** diagrams:

- **Initial State:** This shows the starting point of the state chart diagram that is where the activity starts.



- **State:** A state represents a condition of a modeled entity for which some action is performed. The state is indicated by using a rectangle with rounded corners and contains compartments.

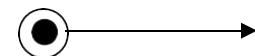


- **Transition:** It is indicated by an arrow. Transition is a relationship between two states which indicates that an object in the first state will enter the second state and performs certain specified actions.

Event/ Action



- **Final State:** The end of the state chart diagram is represented by a solid circle surrounded by a circle.



State Machine Diagrams - Notation



Notation for a State



Initial Pseudo State, Multiple Starts are Valid



Transition



Note



Decision



Constraint



Final PseudoState



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Business Excellence

State

A simple state

State

activities/methods

A state with internal activities

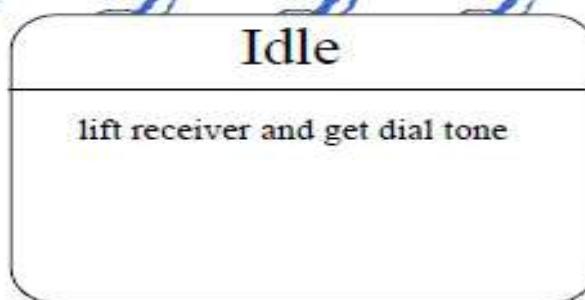


**PRESIDENCY
UNIVERSITY**

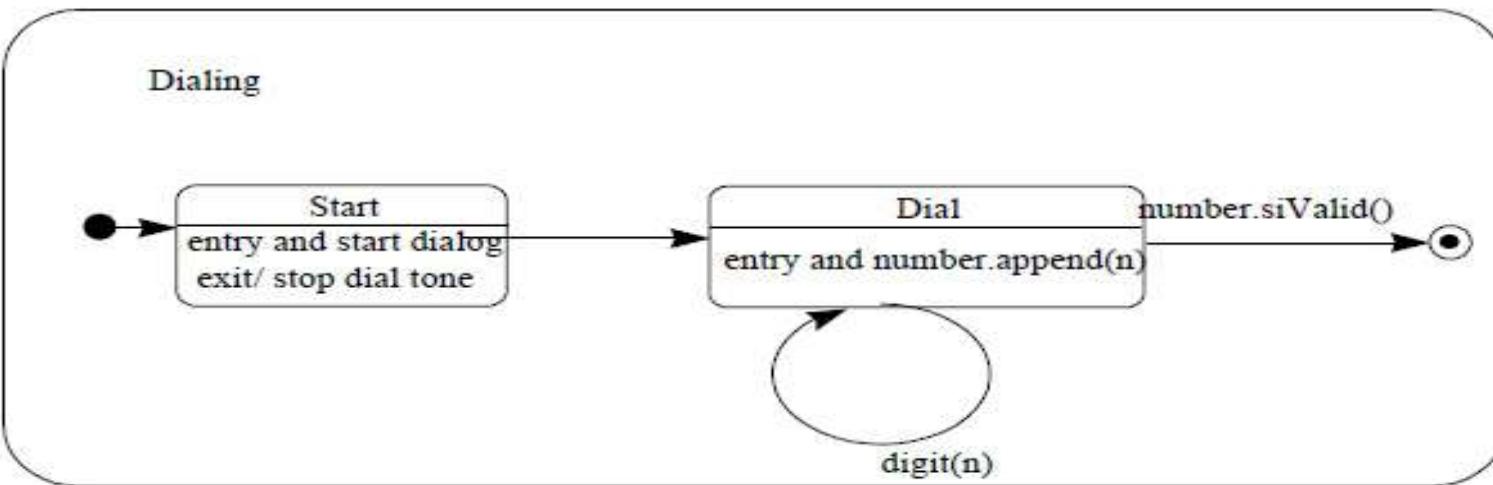
Private University Estd. in Karnataka State by Act No. 41 of 2013



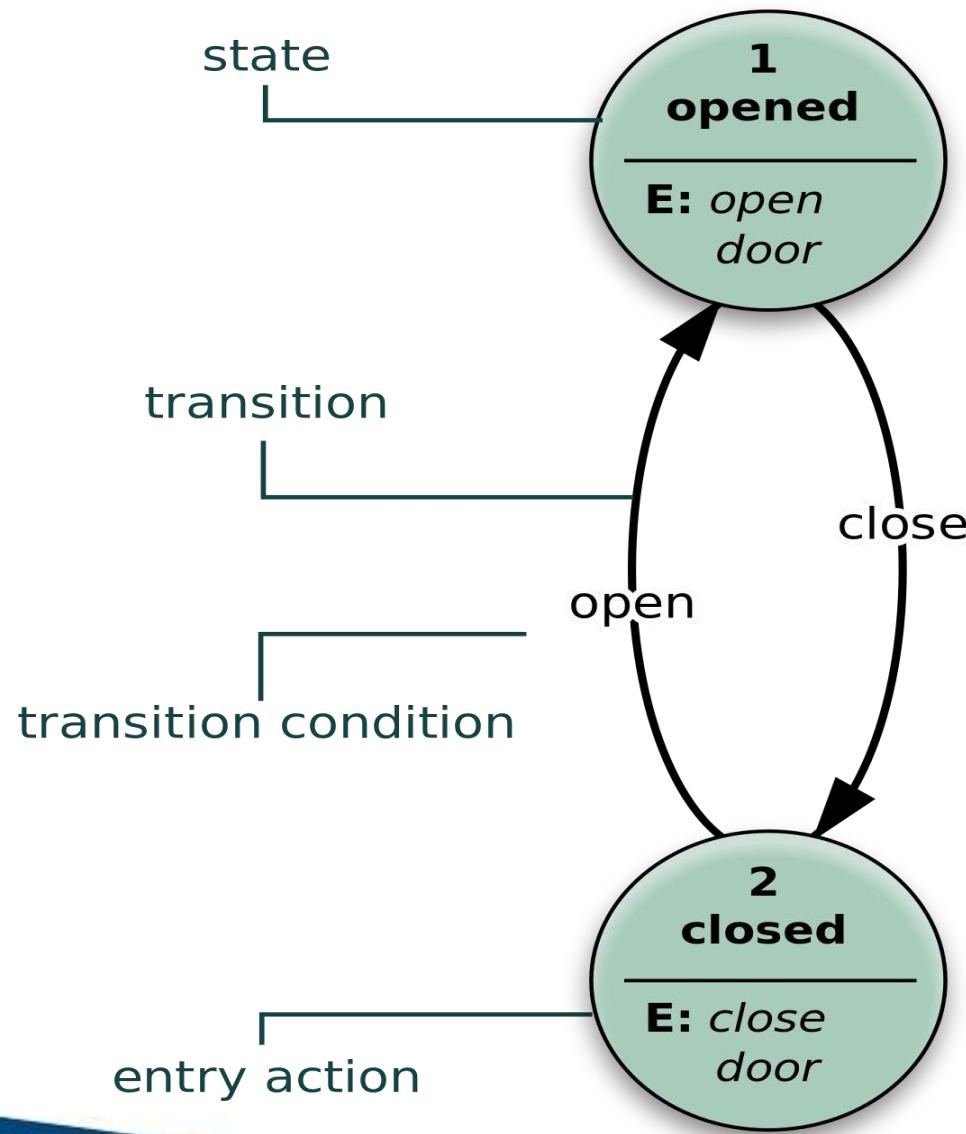
Example



State



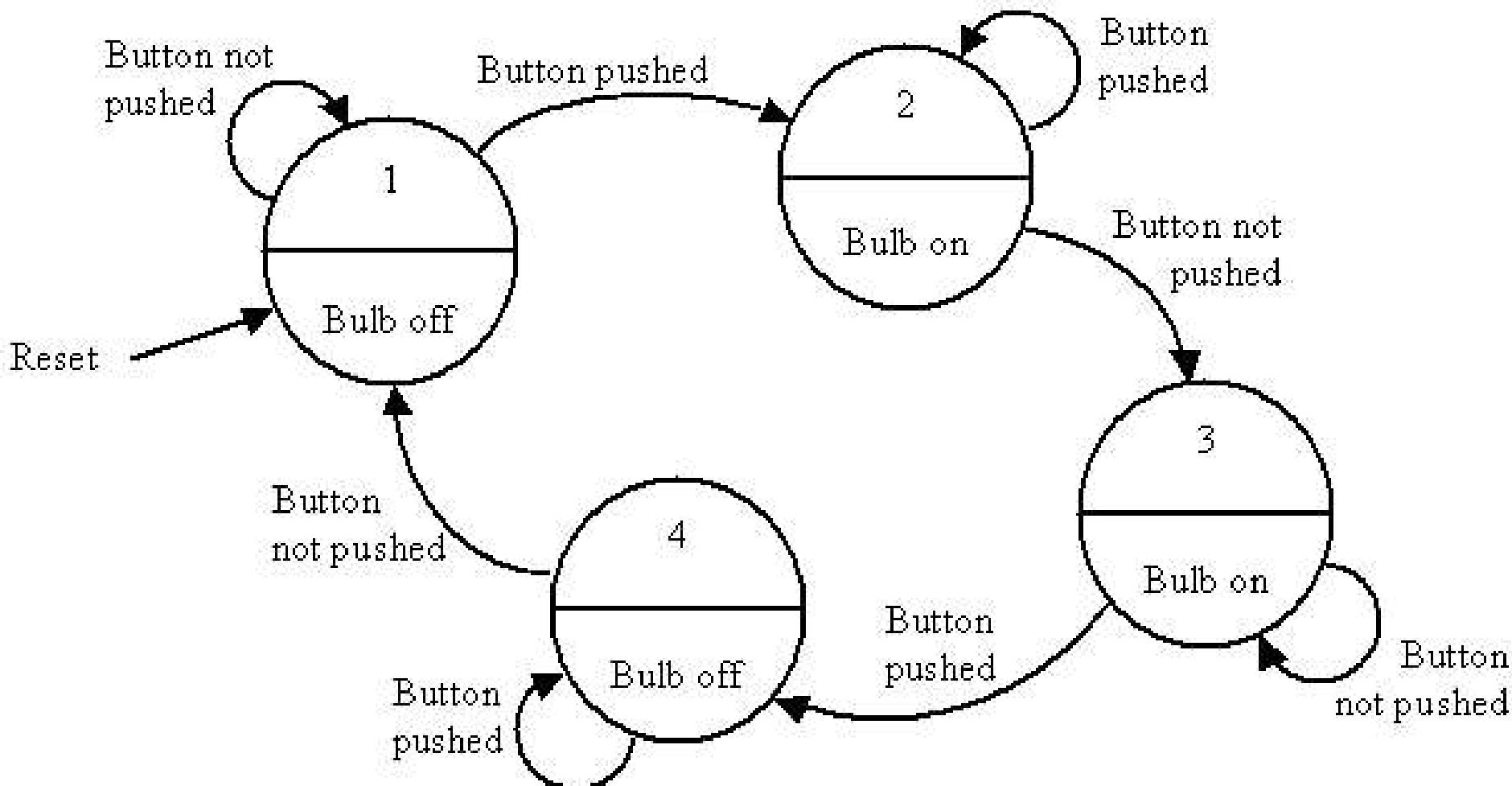
Substates



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013

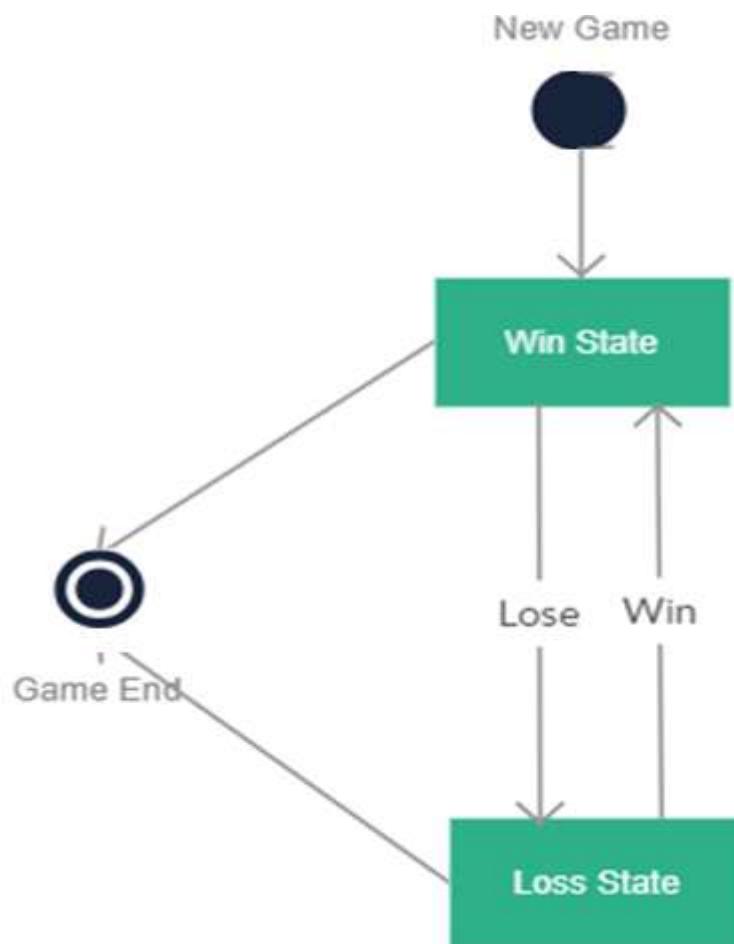




**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013

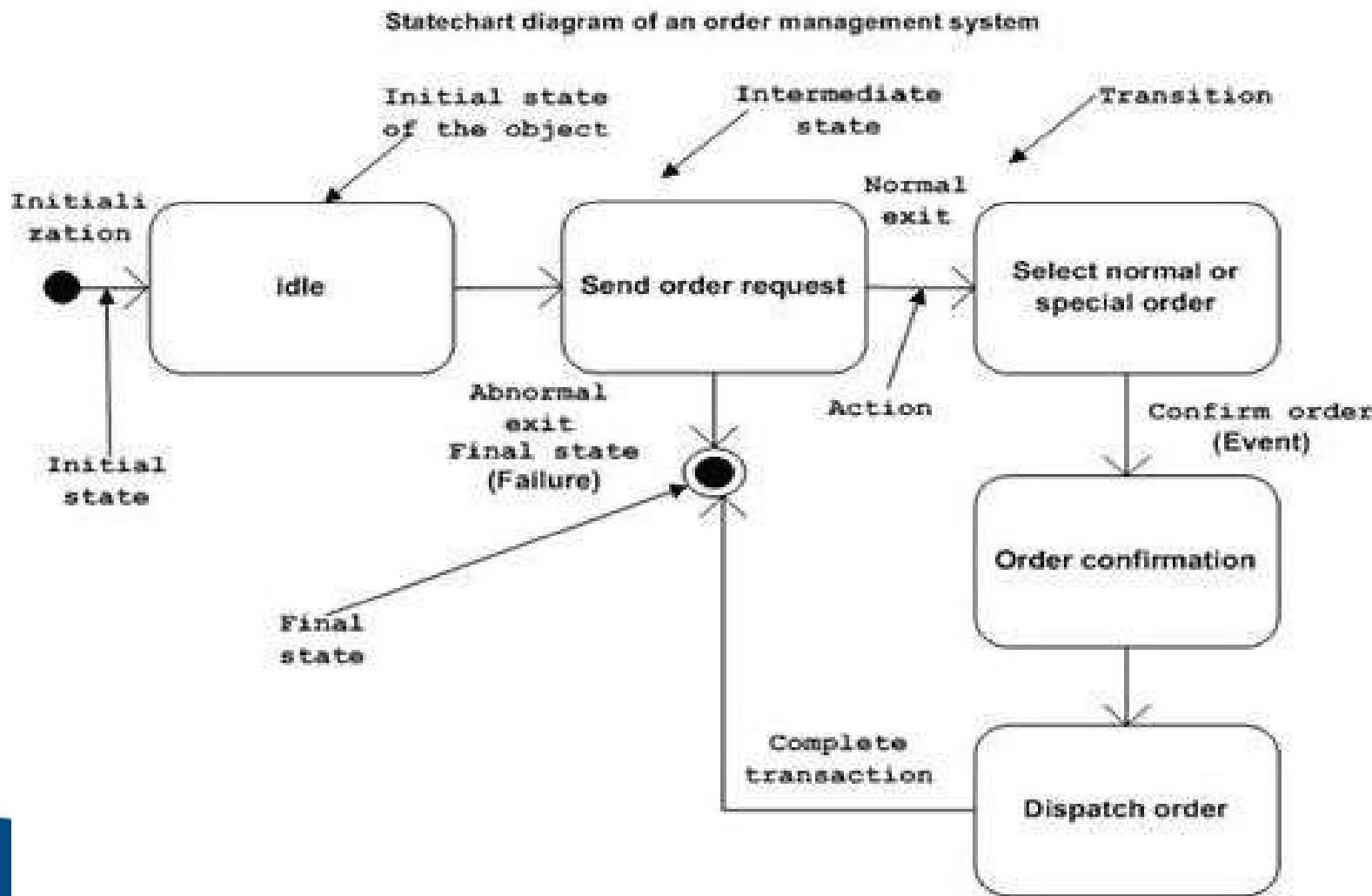




**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



State chart diagram

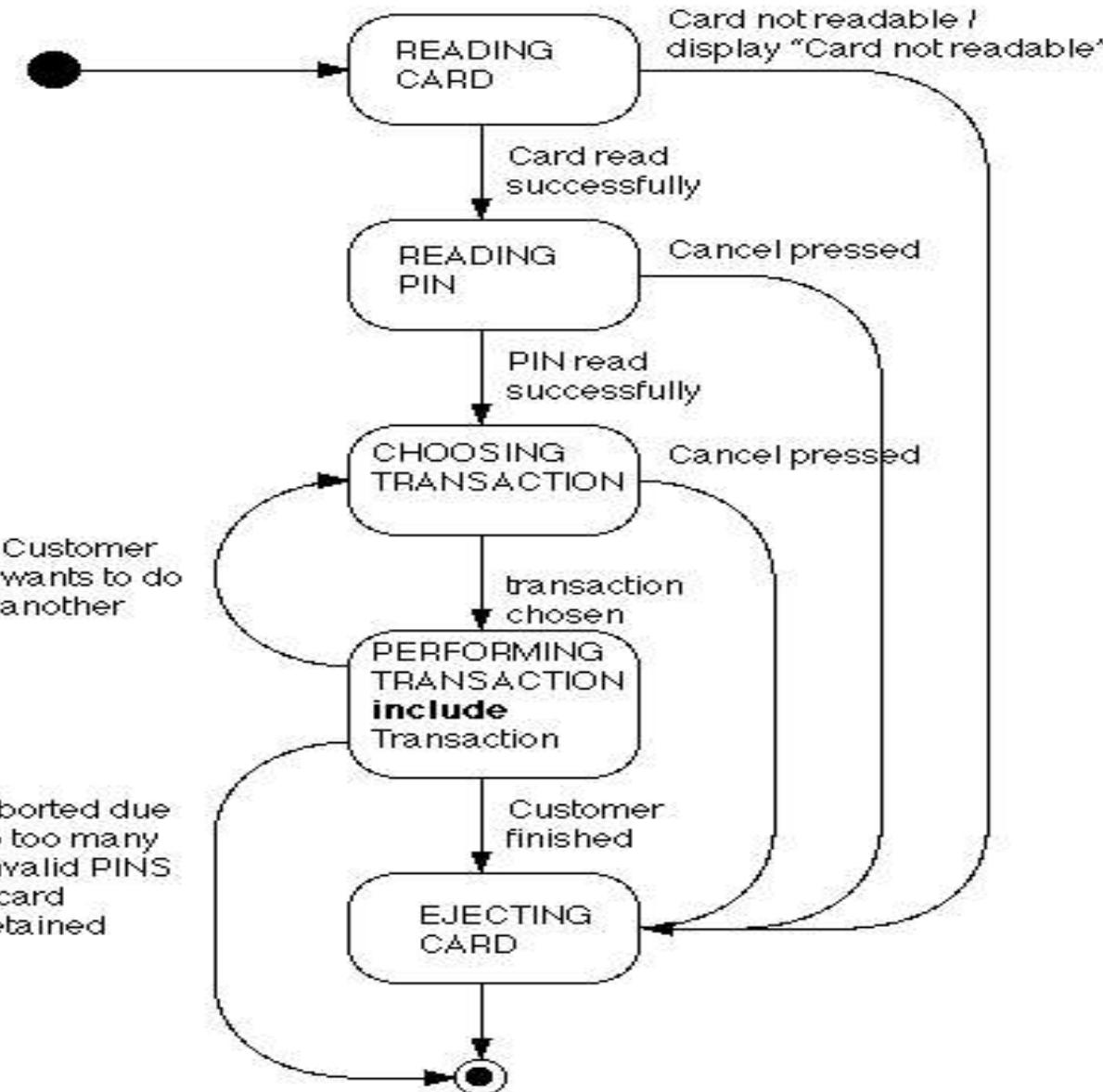
ATM OPERATION



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





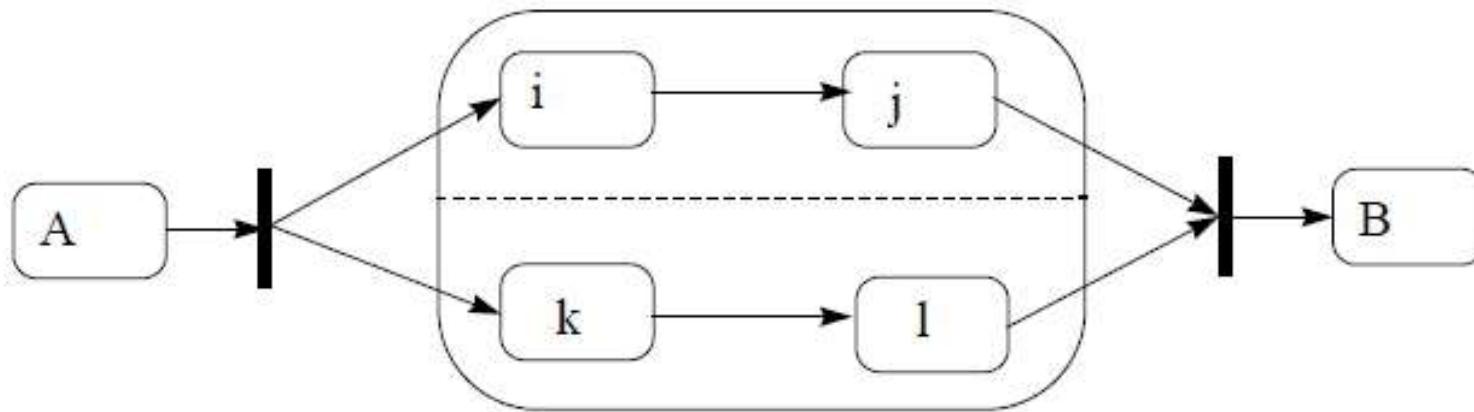
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





Complex Transition



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



When to use Statecharts

So the main usages can be described as:

- To model object states of a system.
- To model reactive system. Reactive system consists of reactive objects.
- To identify events responsible for state changes.
- Forward and reverse engineering.

Activity Diagrams



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Activity Diagram...

- ❖ Activity diagrams represent the dynamic (behavioral) view of a system.
- ❖ Activity diagrams are typically used for business (transaction) process modeling and modeling the logic captured by a single use-case or usage scenario.
- ❖ Activity diagram is used to represent the flow across use cases or to represent flow within a particular use case.
- ❖ UML activity diagrams are the object oriented equivalent of flow chart and data flow diagrams in function-oriented design approach.
- ❖ Activity diagram contains activities, transitions between activities, decision points, synchronization bars, swim lanes and many more...



Activity Diagram...

- ❖ Describes how activities are coordinated.
- ❖ Is particularly useful when you know that an operation has to achieve a number of different things, and you want to model what the essential dependencies between them are, before you decide in what order to do them.
- ❖ Records the dependencies between activities, such as which things can happen in parallel and what must be finished before something else can start.
- ❖ Represents the workflow of the process.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Activity Diagram...(Notations)

1. Activity

- ✓ The Core symbol is used for Activities.

Activity

- ✓ An activity is some task which needs to be done.
- ✓ Each activity can be followed by another activity (sequencing).
- ✓ An activity may be a manual thing, so that it's not necessarily in a program.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Activity Diagram...(Notations)

2. Transmission (Flow)

- ✓ When the action or activity of a state completes, flow of control passes immediately to the next action or activity state

- ✓ The flow of control is shown by arrow symbol.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Activity Diagram... (Notations)

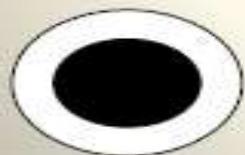
3. Starting and Ending Nodes

- ✓ The source of flow of control is known as '**Initial Node or Starting Node**'.



Starting Node(Mark)

- ✓ Destination of flow of control is called '**Ending Node or Final Node**'.



Ending Node



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013

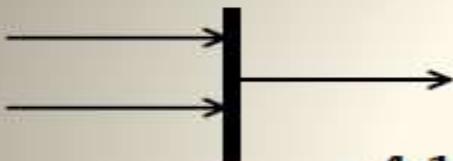


Activity Diagram... (Notations)

4. Join and Fork

✓ **Join**

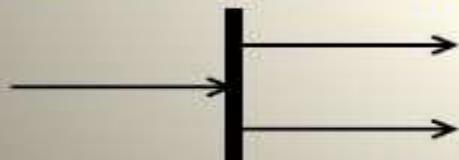
A block bar with several flows entering in it and one leaving from it. This denotes the end of parallel activities



4.1 Synch. Bar (Join)

✓ **Fork**

A black bar (horizontal/vertical) with one flow going into it and several leaving it. This denotes the beginning of parallel activities



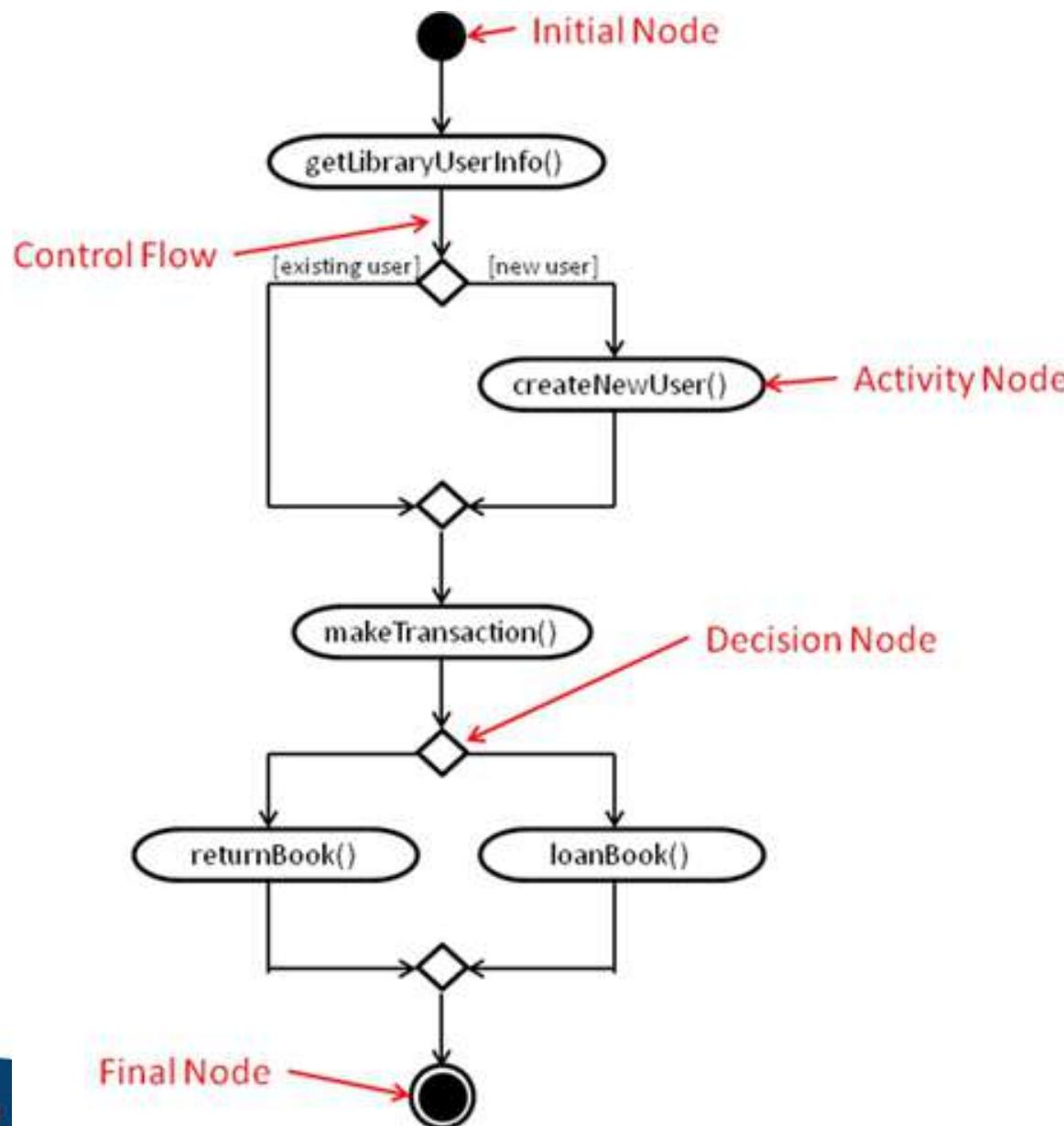
4.2 Splitting Bar (Fork)



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





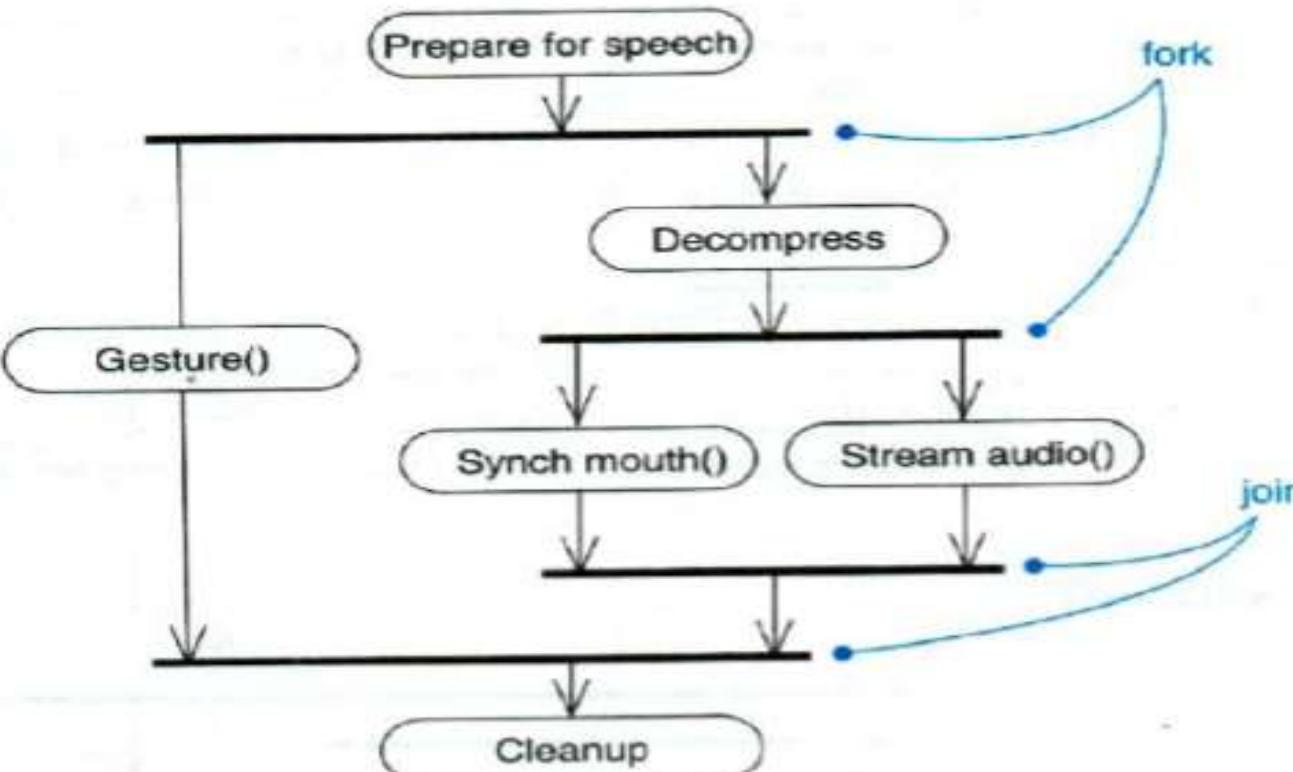
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013

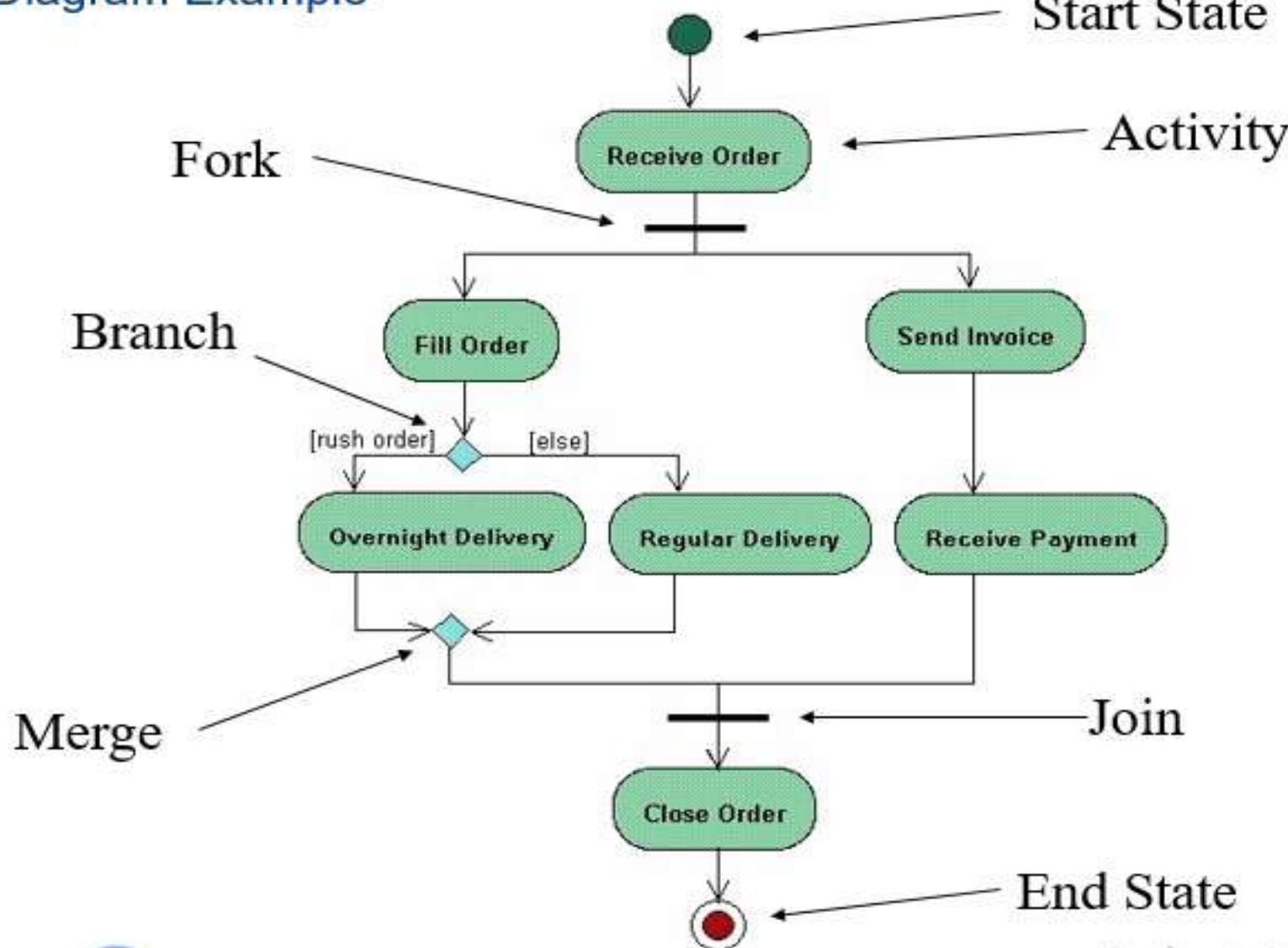


Activity Diagram... (Notations)

Example for Join and Fork



Activity Diagram Example



**PRESIDENCY
UNIVERSITY**



Private University Estd. in Karnataka State by Act No. 41 of 2013

Activity Diagram... (Notations)

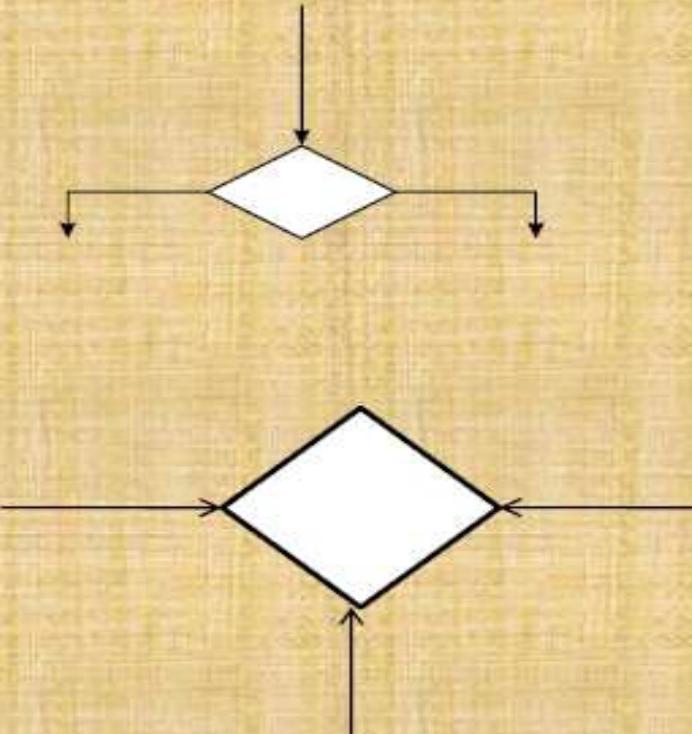
5. Decision and Merge

✓ Decision

- A diamond with one flow entering and several leaving. The flow leaving includes conditions as yes/ no state.

✓ Merge

- A diamond with several flows entering and one leaving. The implication is that all incoming flow to reach this point until processing continues



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Activity Diagram... (Notations)

Difference between Join and Merge

- A join is different from a merge in that the join synchronizes two inflows and produces a single outflow. The outflow from a join cannot execute until all inflows have been received

- A merge passes any control flows straight through it. If two or more inflows are received by a merge symbol, the action pointed to by its outflow is executed two or more times



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013

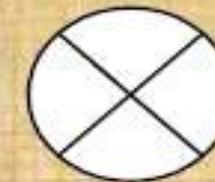


Activity Diagram... (Notations)

6. Flow Final and Swimlane

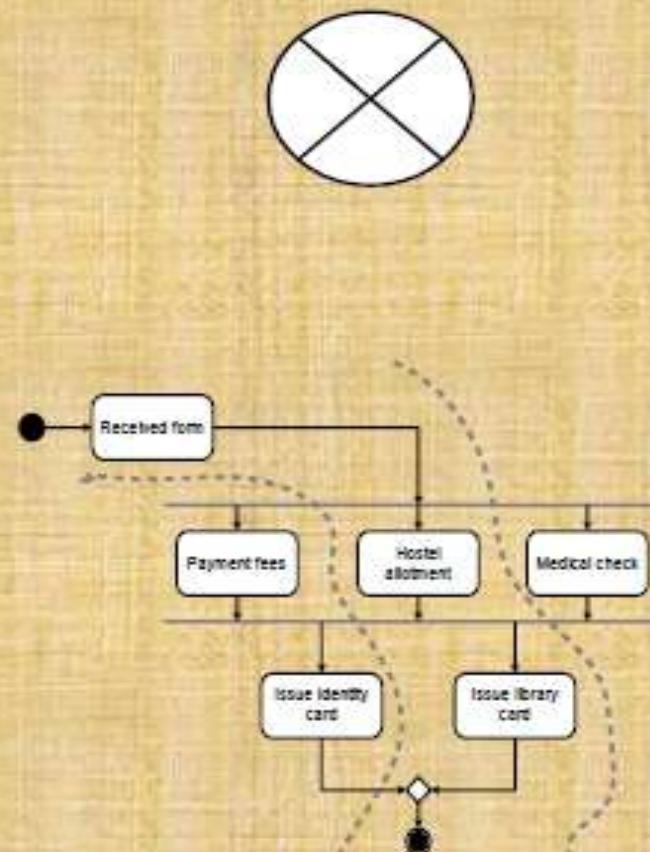
✓ **Flow final**

- The circle with X through it. This indicates that Process stop at this point



✓ **Swim lane**

- A partition in activity diagram by means of dashed line, called swim lane. This swim lane may be horizontal or vertical
- Each zone represents the responsibilities of a particular class or department



Swimlane Guidelines

- Order Swimlanes in a Logical Manner
- Apply SwimLanes To Linear (sequential)
- Have Less Than Five Swimlanes
- Consider Swimareas For Complex Diagrams
- SwimLane Suggest The Need to Reorganize Into Smaller Activity Diagrams
- Consider Horizontal Swimlanes for Business Processes Left to right

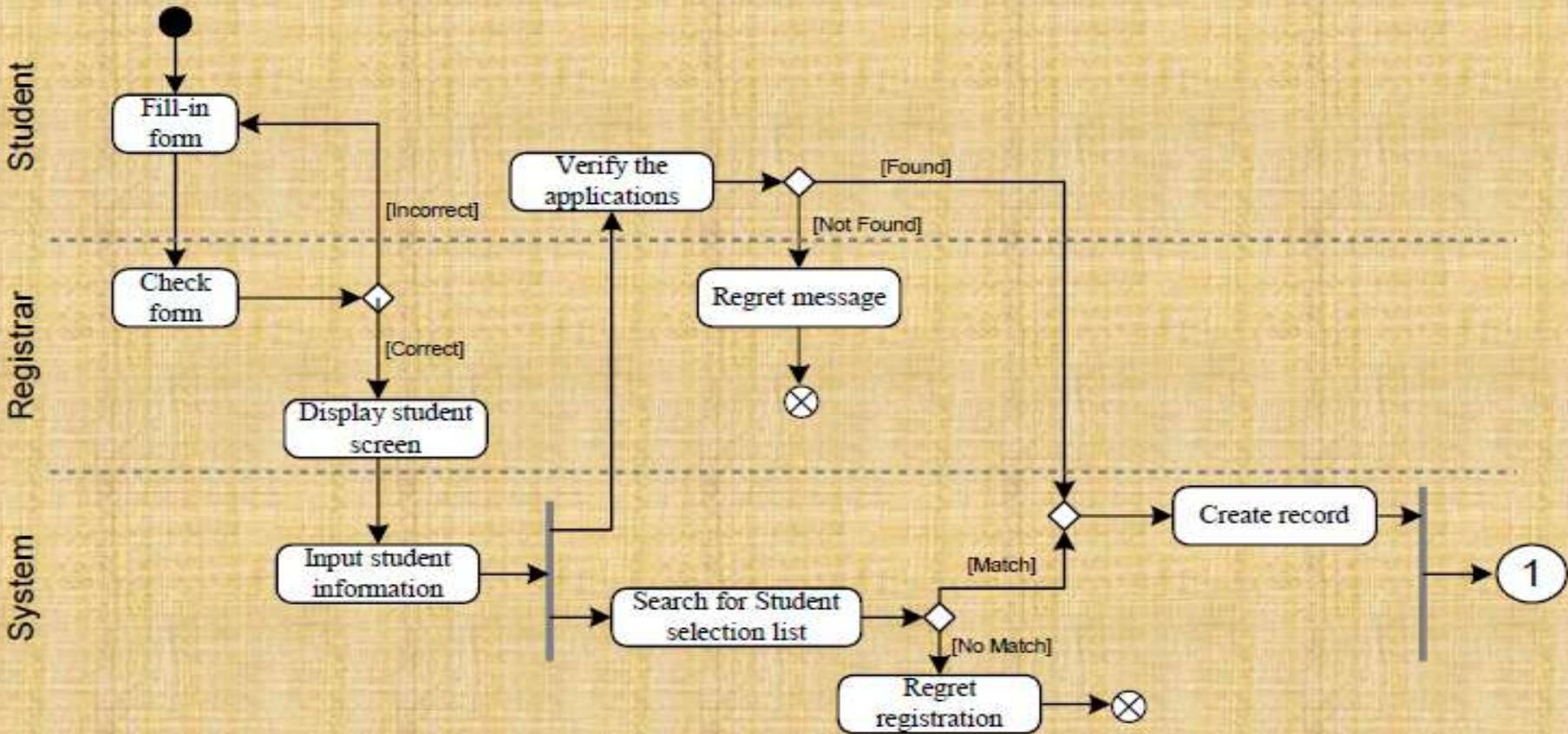


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Example of Activity Diagram...

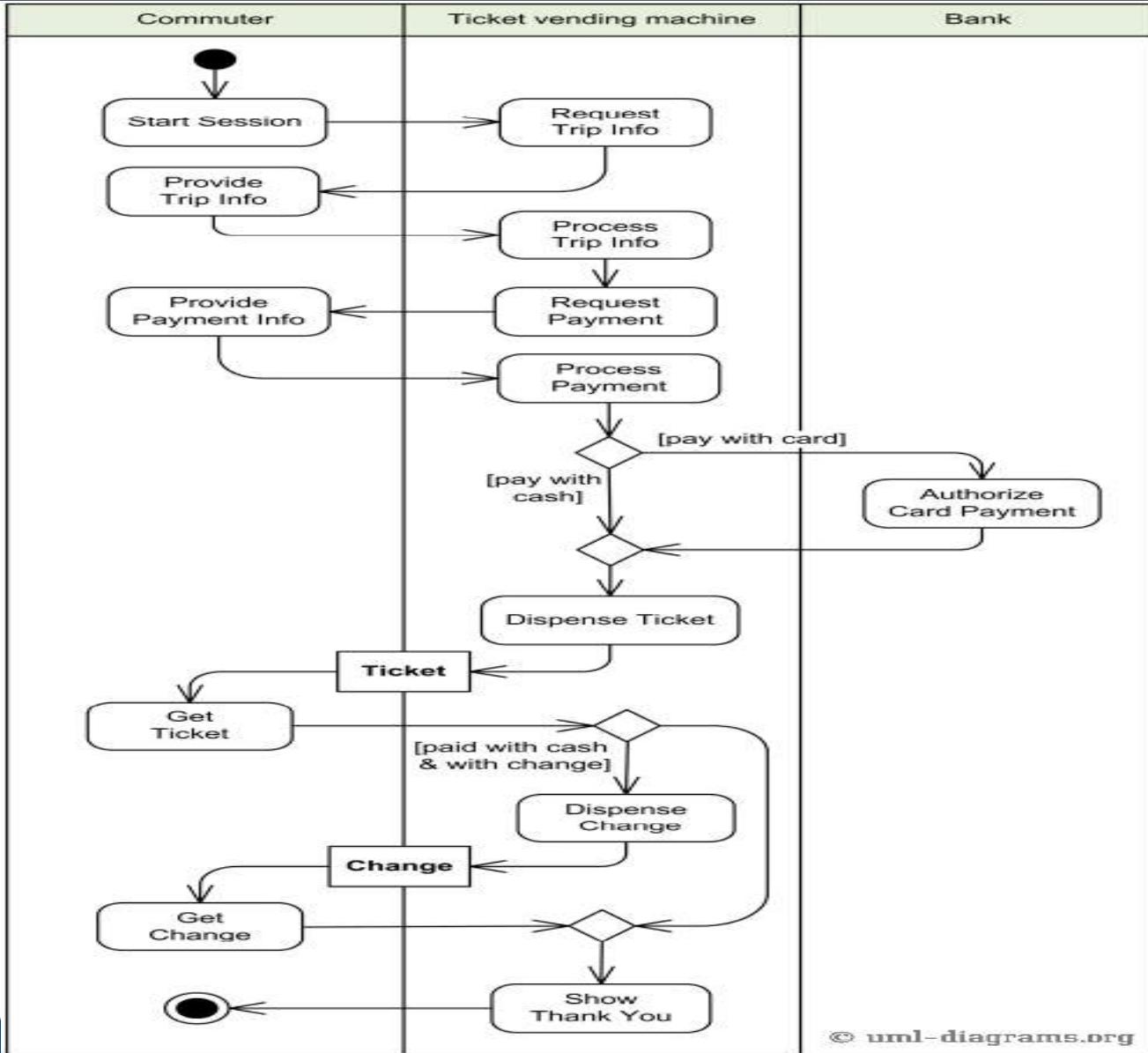


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Draw an Activity Diagram for Ticket Vending Machine



© uml-diagrams.org



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



UML : ACTIVITY DIAGRAM

- Example : creating document
- Open the word processing package
- Create a file
- Save the file under a unique name within its directory
- Type the document
- If graphics are necessary, open the graphics package, create the graphics, and paste the graphics into the document
- If a spreadsheet is necessary, open the spreadsheet package, create the spreadsheet, and paste the spreadsheet into the document
- Save the file
- Print a hard copy of the document
- Exit the word processing package



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013





**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Drawback of Activity Diagram...

- ❖ Activity diagrams tell you what is happening, but not who does what.
- ❖ In domain modelling, this diagram type does not convey which people or departments are responsible for each activity.
- ❖ In programming, it does not convey which class is responsible for each activity