CSE213 - OBJECT ORIENTED ANALYSIS AND DESIGN

AY-2020-2021

SEMESTER-VI

JANUARY-MAY

MODULE-1

COURSE OUTCOMES:

- Define the basic concepts of Object Oriented approaches

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

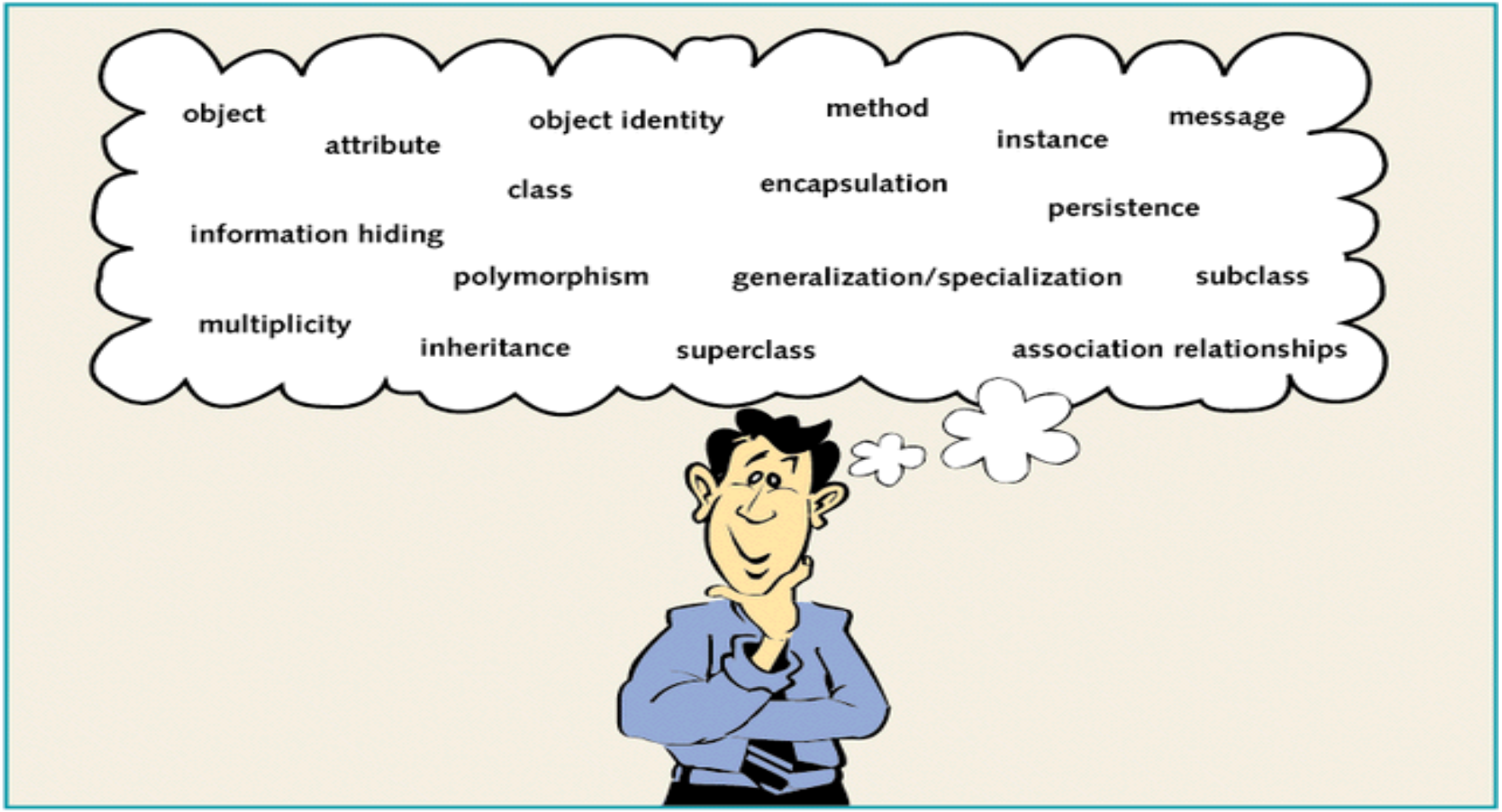


Figure 1-5 Key OO concepts

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

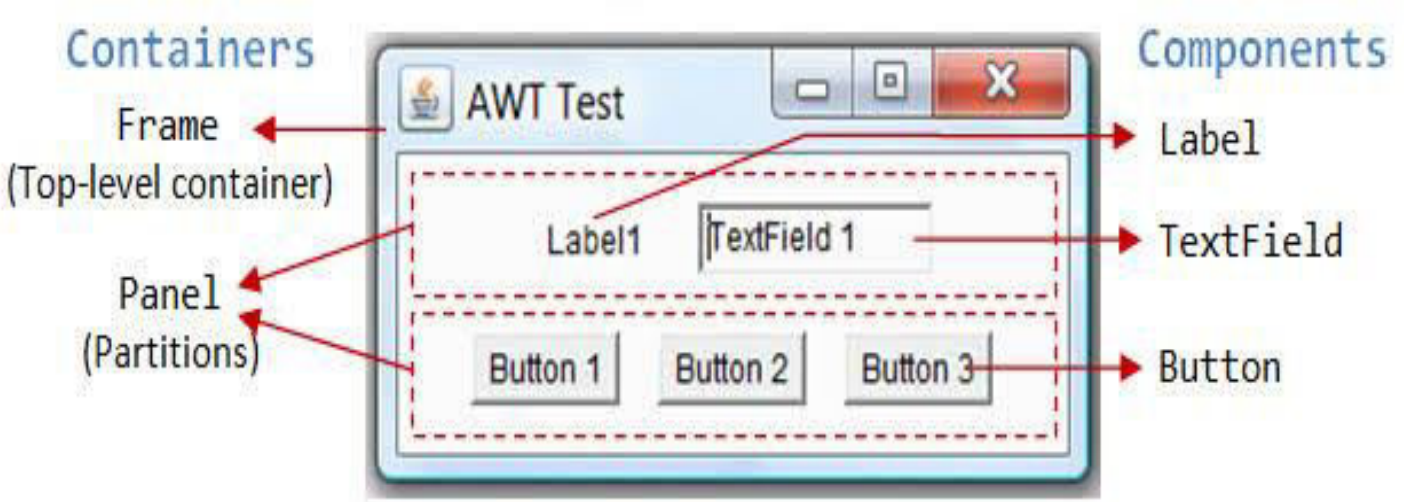
- Behavior that represents externally visible activities performed by an object in terms of changes in its state.
- Objects can be modelled according to the needs of the application.
- An object may have a physical existence, like a customer, a car, etc.; or an intangible conceptual existence, like a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



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**

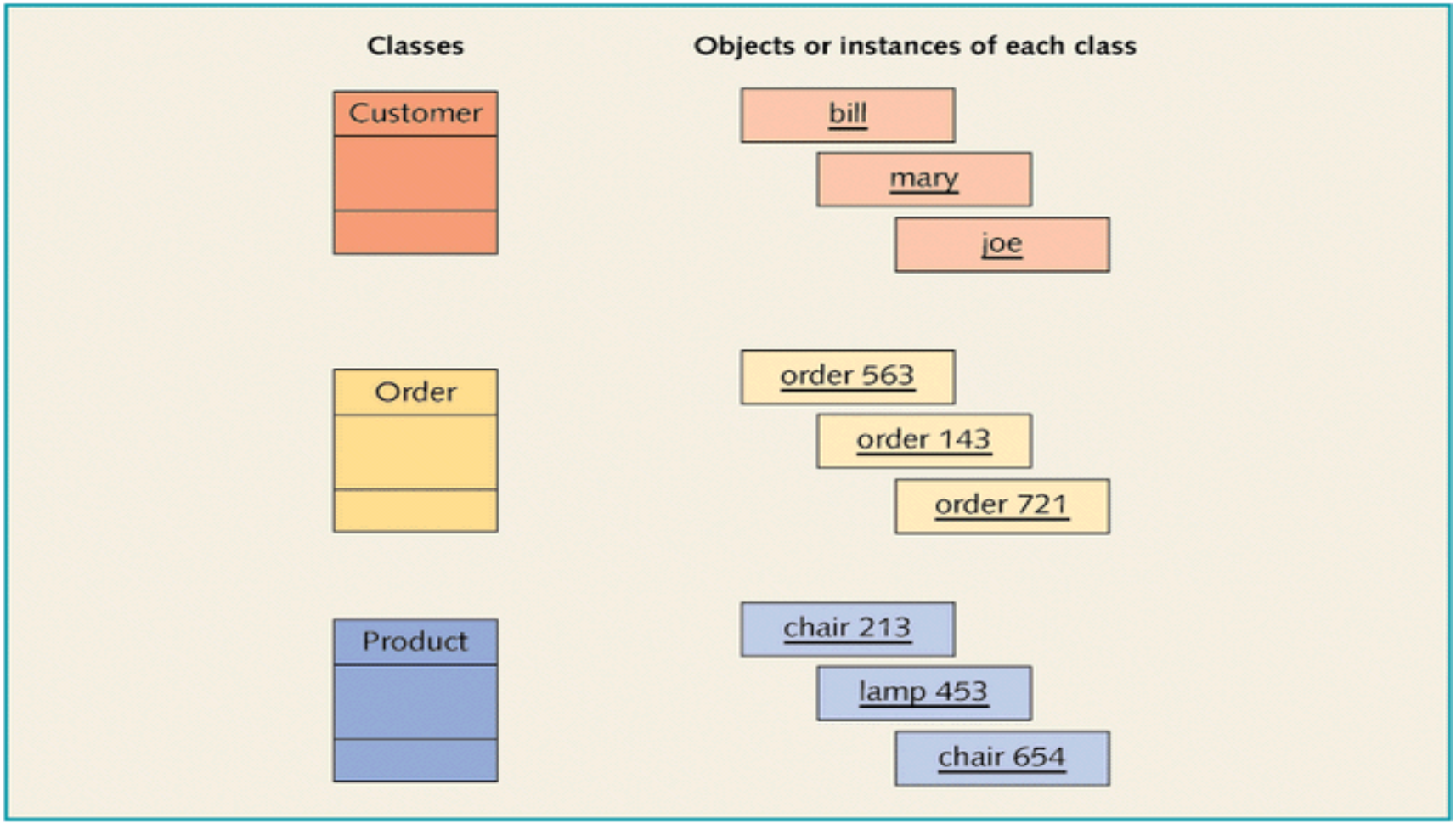


Figure 1-9 Class versus objects or instances of the class

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. Behaviuor

```
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

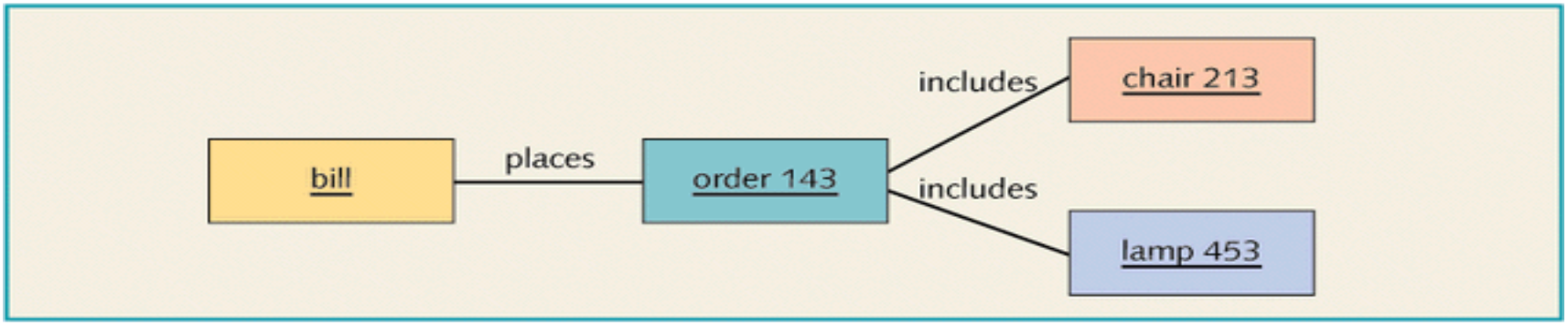


Figure 1-10 Associating objects with other objects

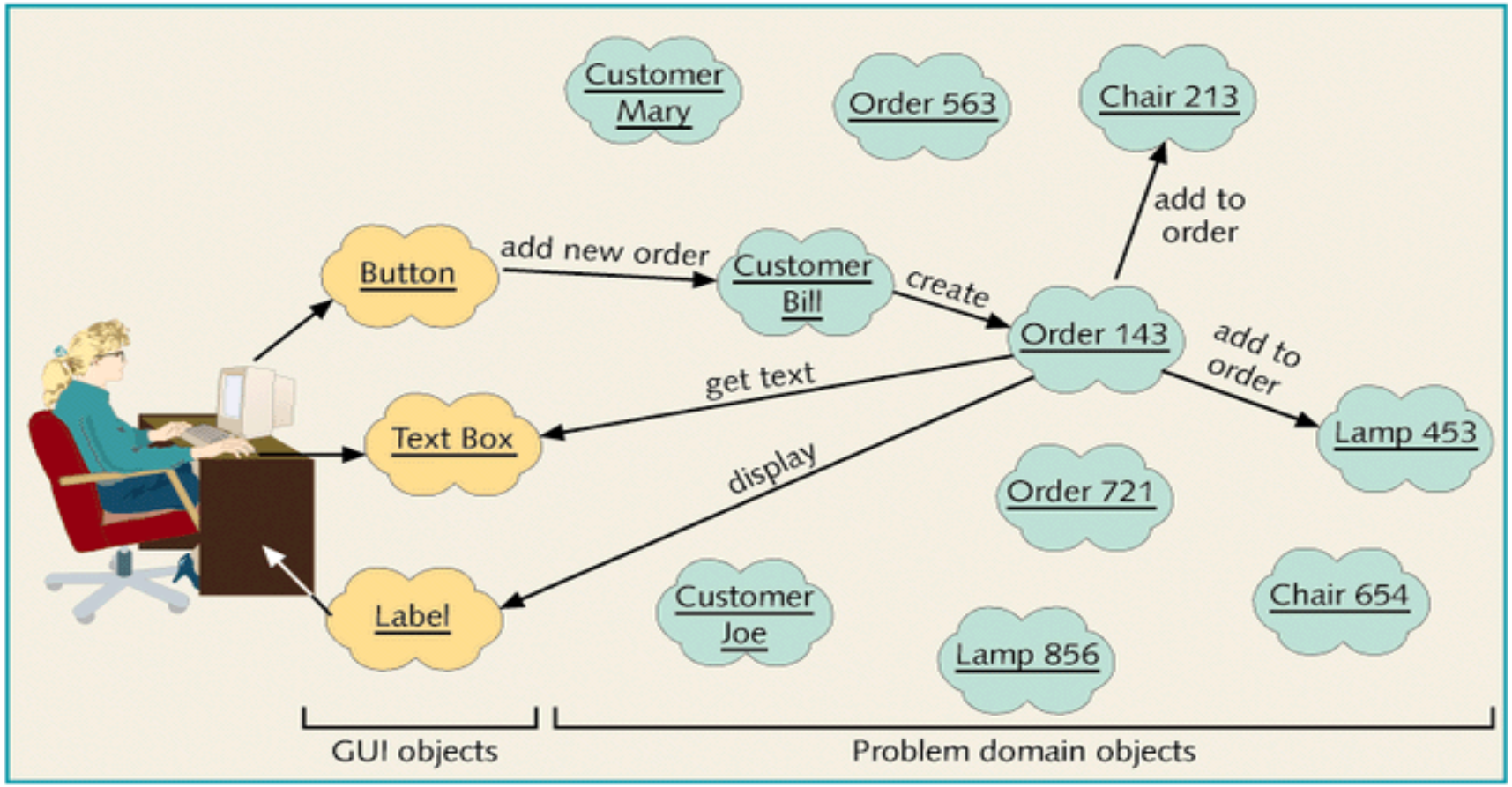


Figure 1-8 Order-processing system where objects interact by sending messages

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

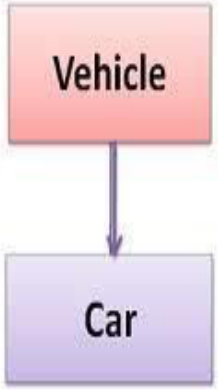
- Besides, the subclass may add its own attributes and methods and may modify any of the super-class methods. Inheritance defines an "is – a" relationship.

Example

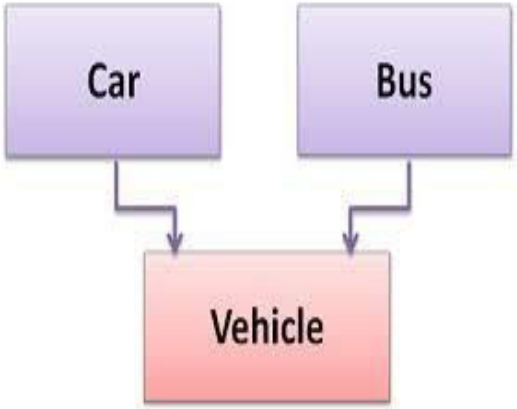
- From a class Mammal, a number of classes can be derived such as Human, Cat, Dog, Cow, etc. Humans, cats, dogs, and cows all have the distinct characteristics of mammals. In addition, each has its own particular characteristics.
- It can be said that a cow "is – a" mammal.

Types of Inheritance

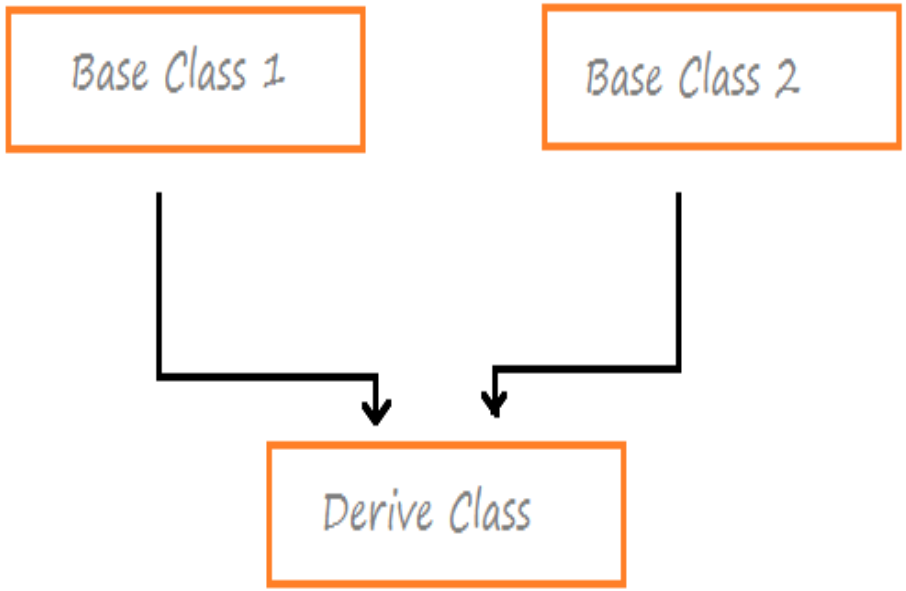
- **Single Inheritance** – A subclass derives from a single super-class.
- **Multiple Inheritance** – A subclass derives from more than one super-classes.
- **Multilevel Inheritance** – A subclass derives from a super-class which in turn is derived from another class and so on.
- **Hierarchical Inheritance** – 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.



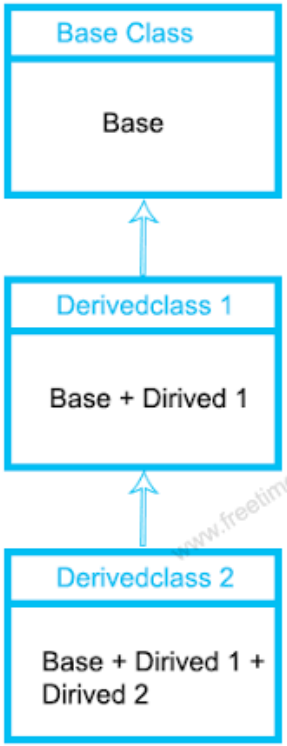
Single Inheritance



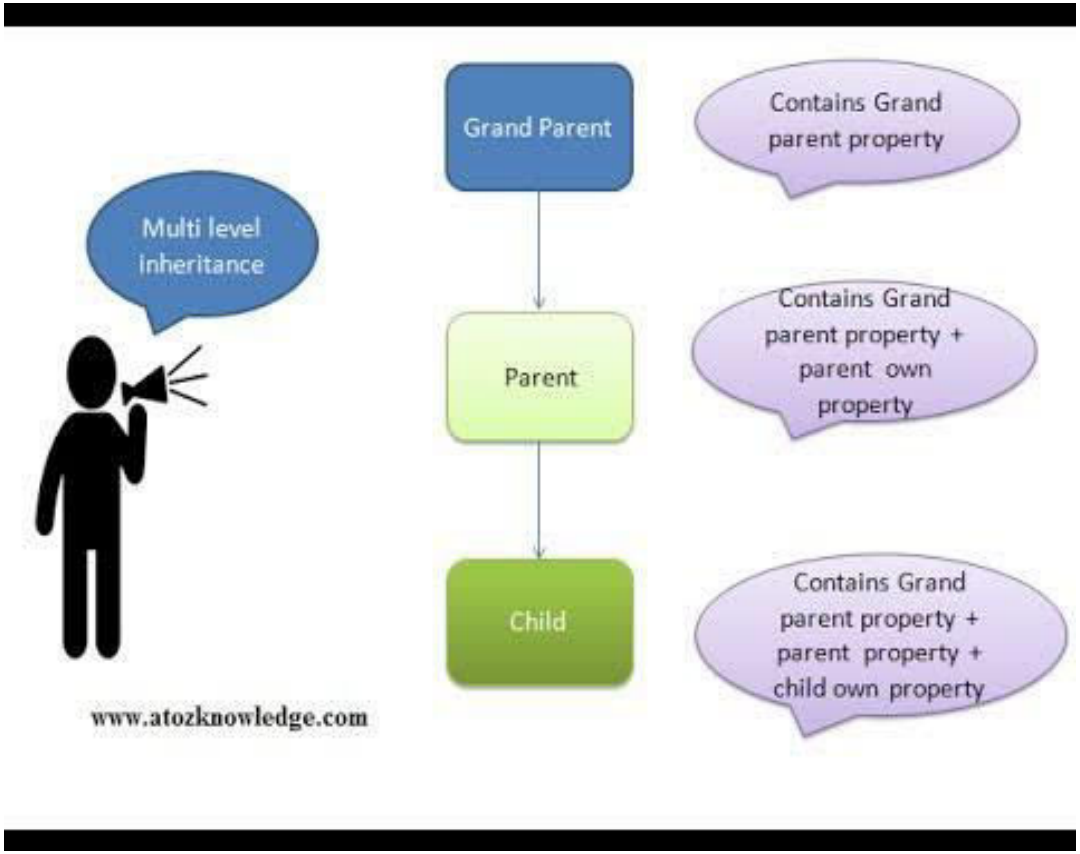
Multiple Inheritance

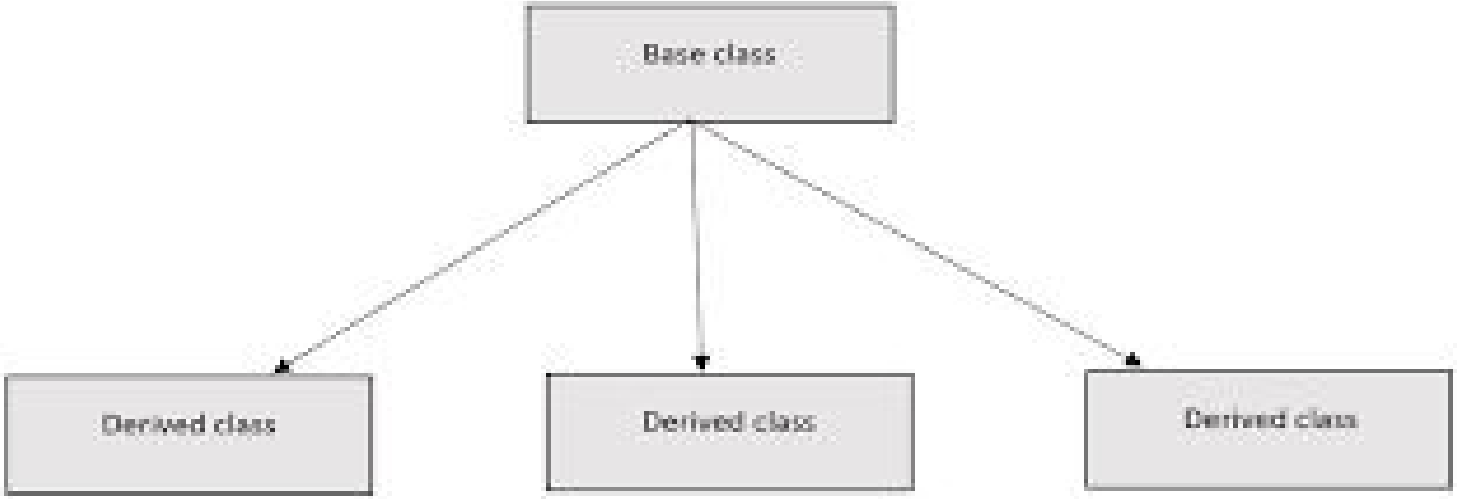


Multiple Inheritance



Multilevel Inheritance





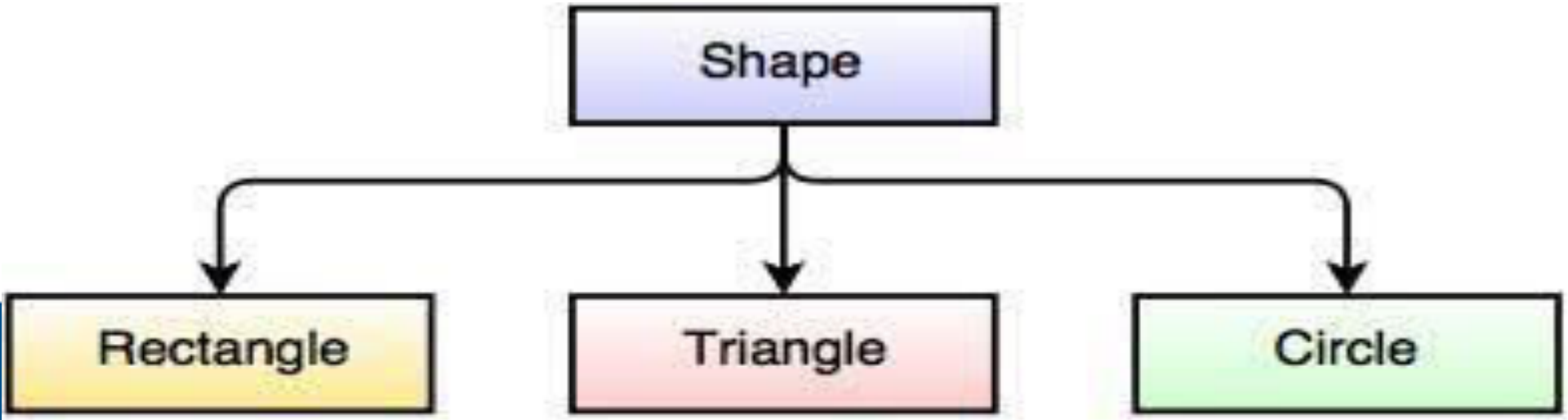
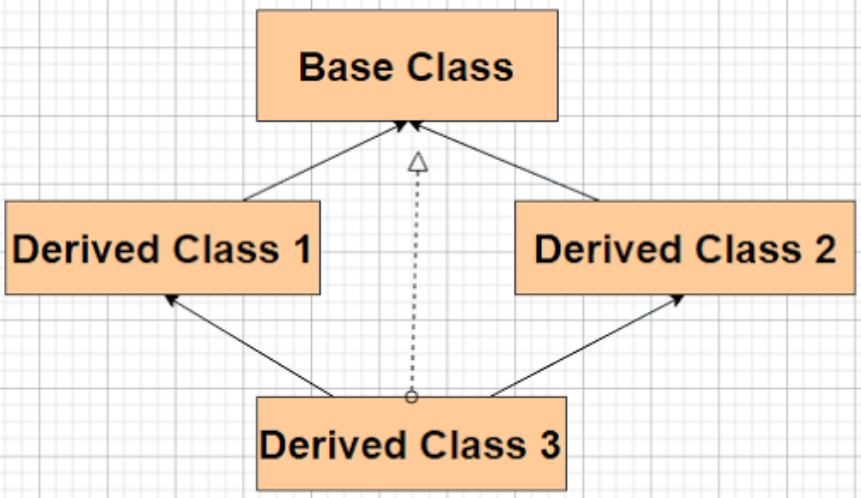


Fig. Hierarchical Inheritance with Class Shape



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!!**

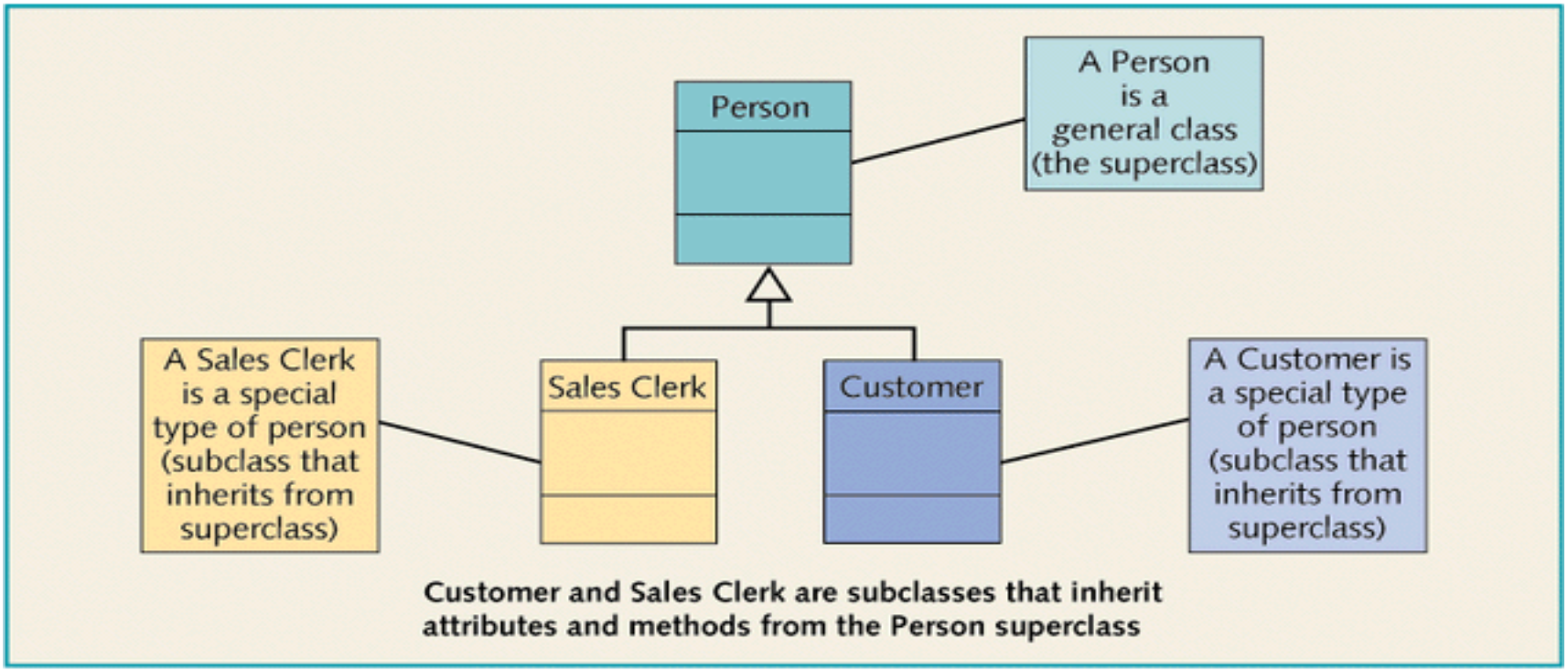


Figure 1-11 Superclass and subclass

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.

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()

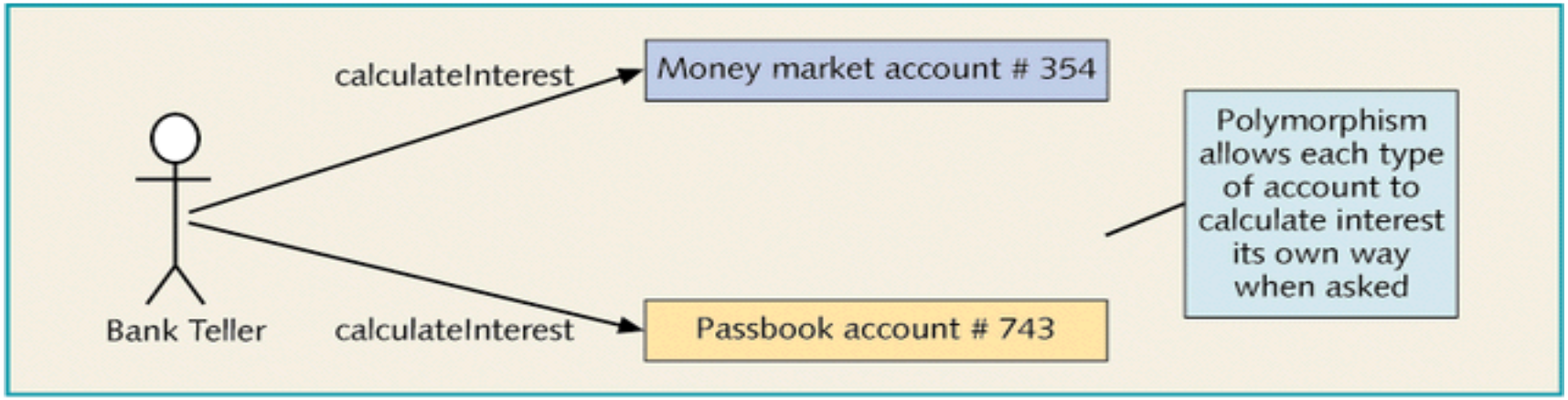


Figure 1-12 Polymorphism for two different types of bank accounts

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

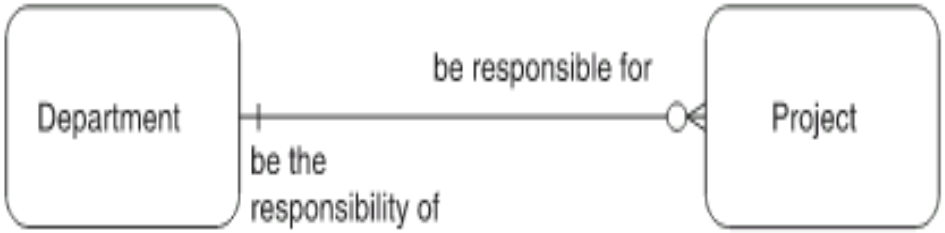
- Degree of an association denotes the number of classes involved in a connection. Degree may be unary, binary, or ternary.
- A **unary relationship** connects objects of the same class.
- A **binary relationship** connects objects of two classes.
- A **ternary relationship** connects objects of three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

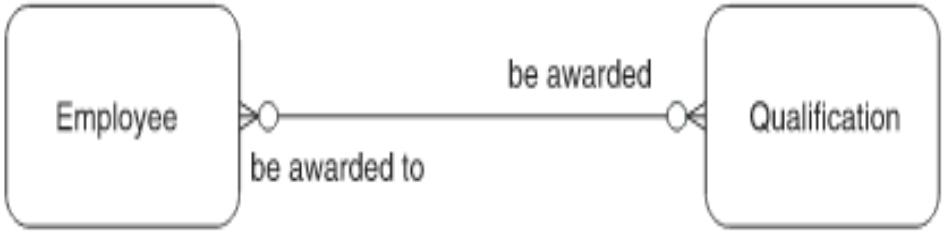
one-to-one

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



one-to-many

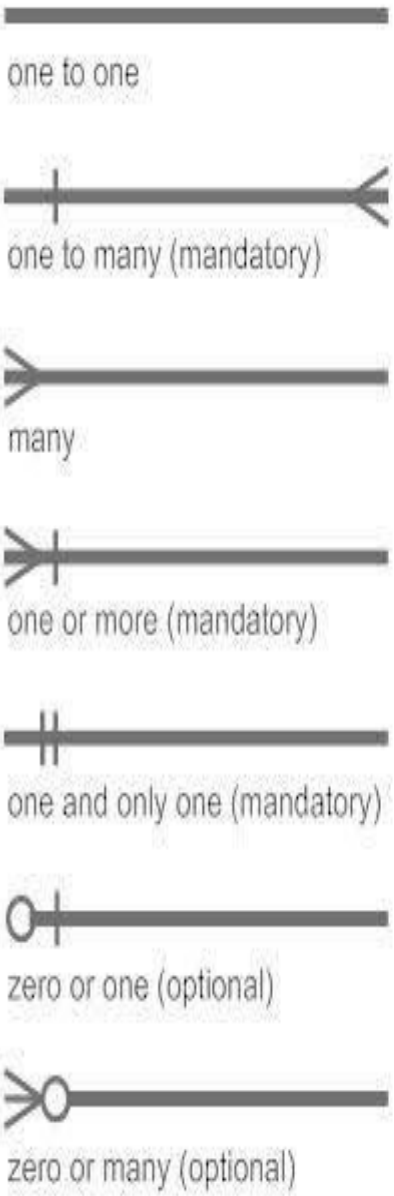
Each Department may be responsible for one or more Projects.
Each Project must be the responsibility of one Department.



many-to-many

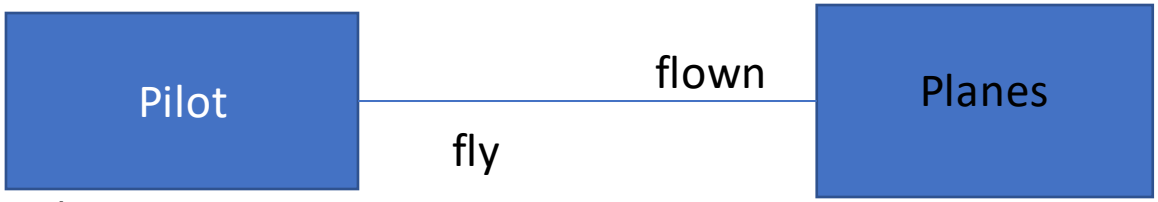
Each Employee may be awarded one or more Qualifications.
Each Qualification may be awarded to one or more Employees.

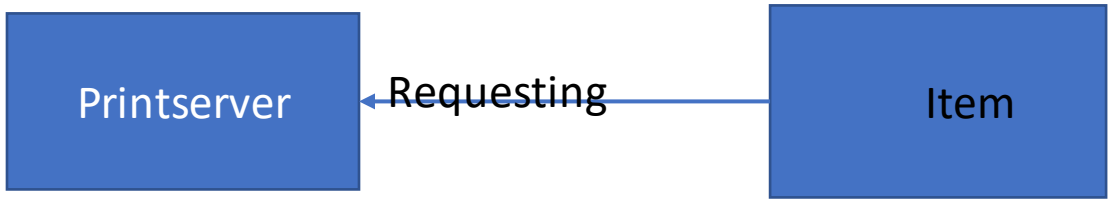
Information Engineering Style



Objects relationships: Association

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





Aggregation or Composition

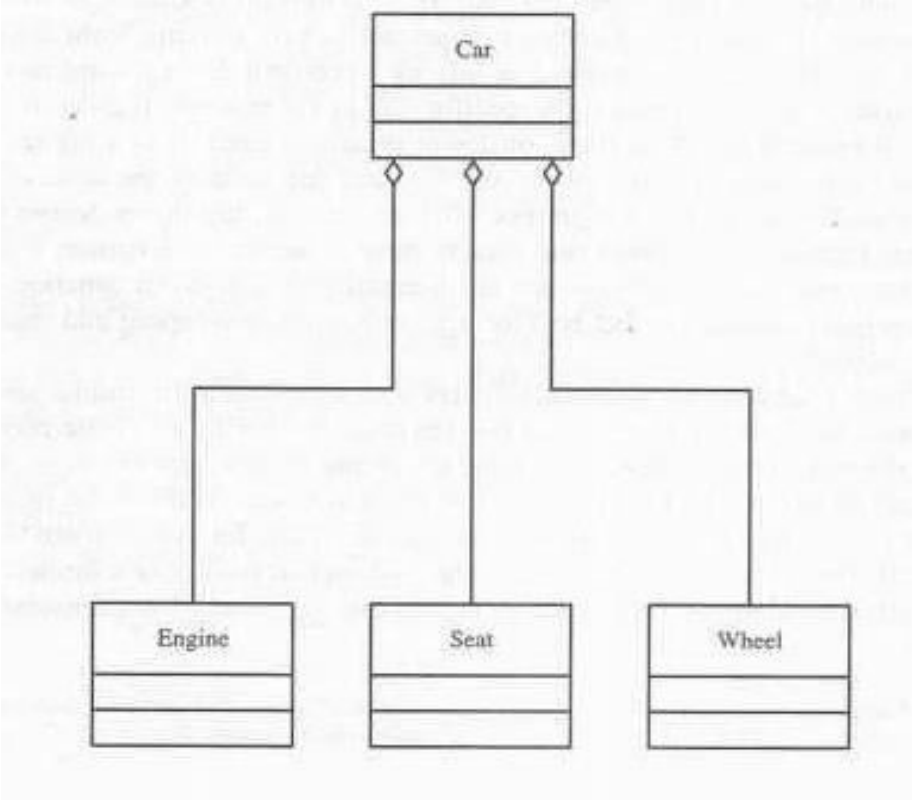
- Aggregation or composition is a relationship among classes by which a class can be made up of any combination of objects of other classes.
- It allows objects to be placed directly within the body of other classes.
- Aggregation is referred as a “part–of” or “has–a” relationship, with the ability to 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



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()

}

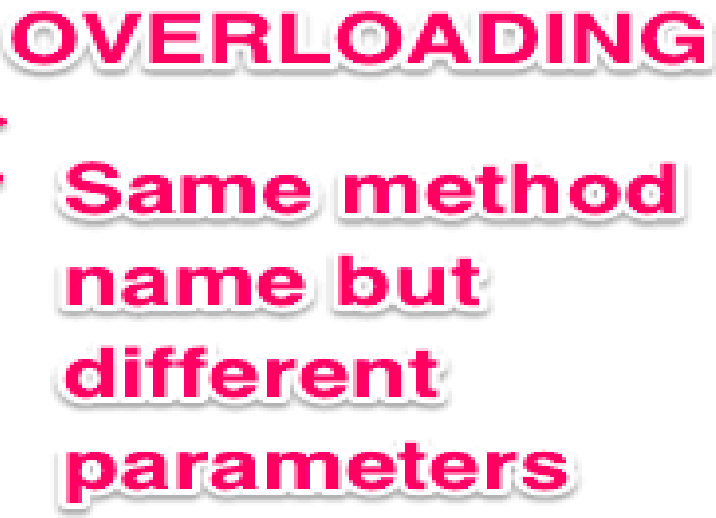
Add()

{

}

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");
     }
   }
}
```

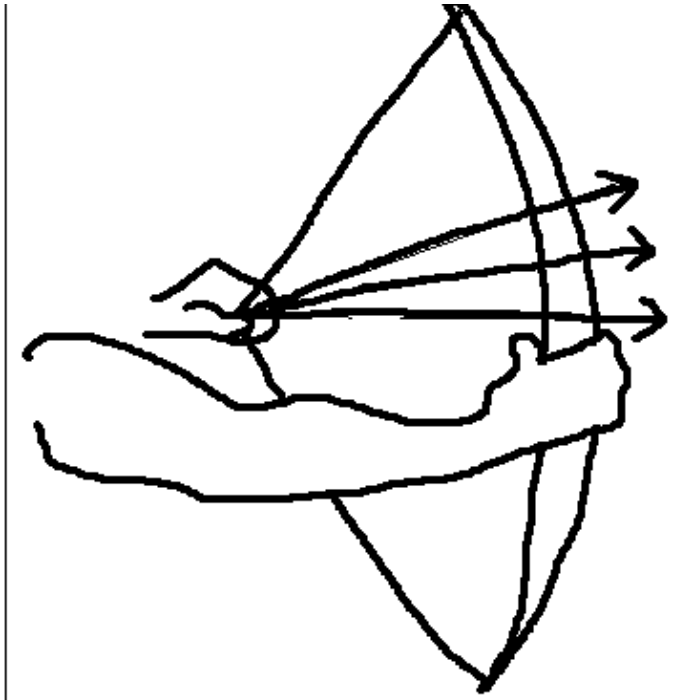


Overriding means having two methods with the same method name and parameters (i.e., method signature). One of the methods is in the parent class and the other is in the child class.

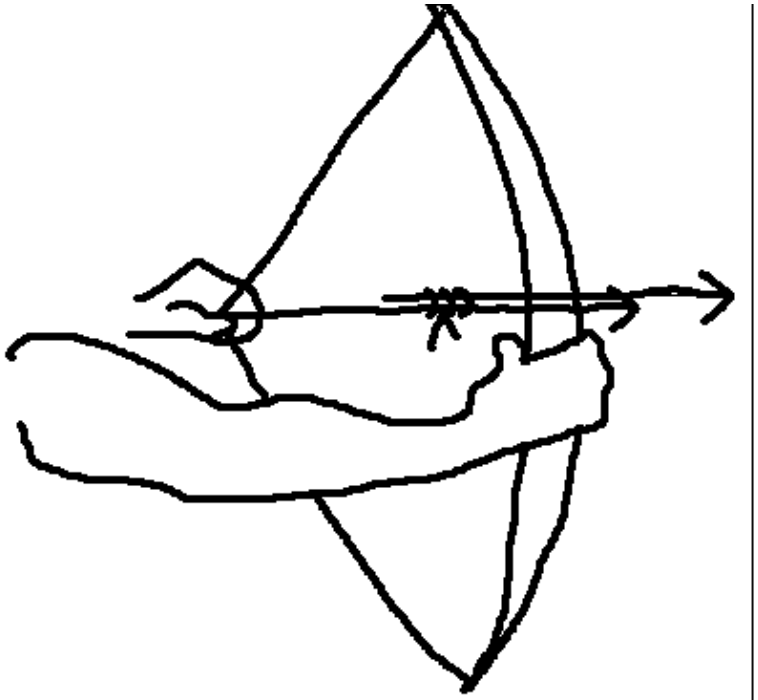
```
class Cat{

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



Overloading



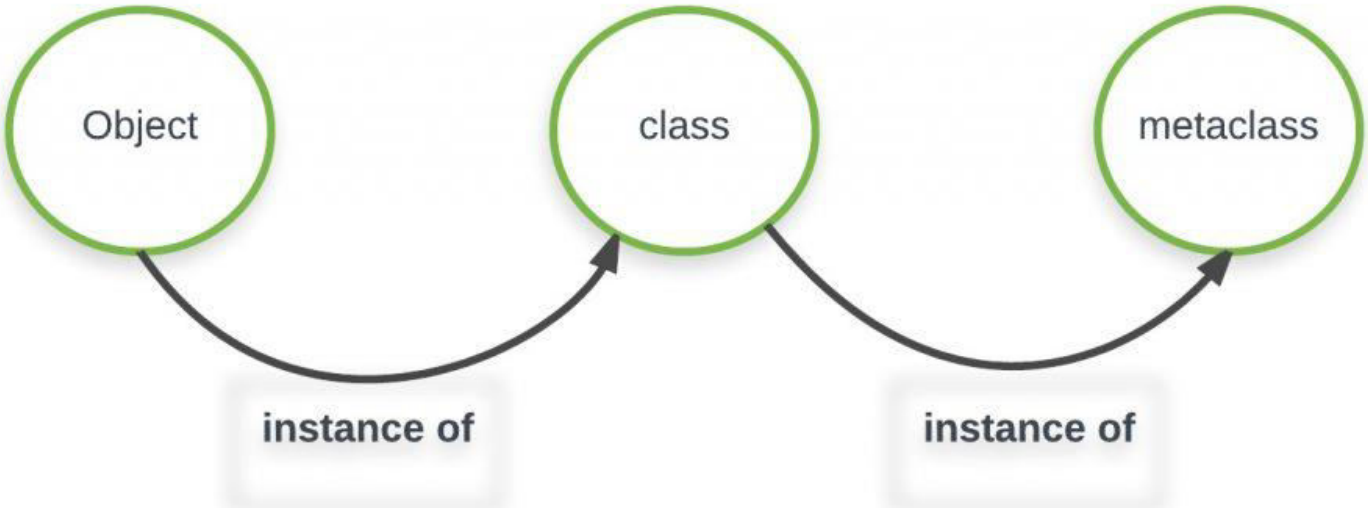
Overriding

Differences Between Static Binding And Dynamic Binding In Ja

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 h run time.
Method overloading is the best example of static binding.	Method overriding is the best example of dyn 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, the overridden.

Meta-Classes



- **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.**

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

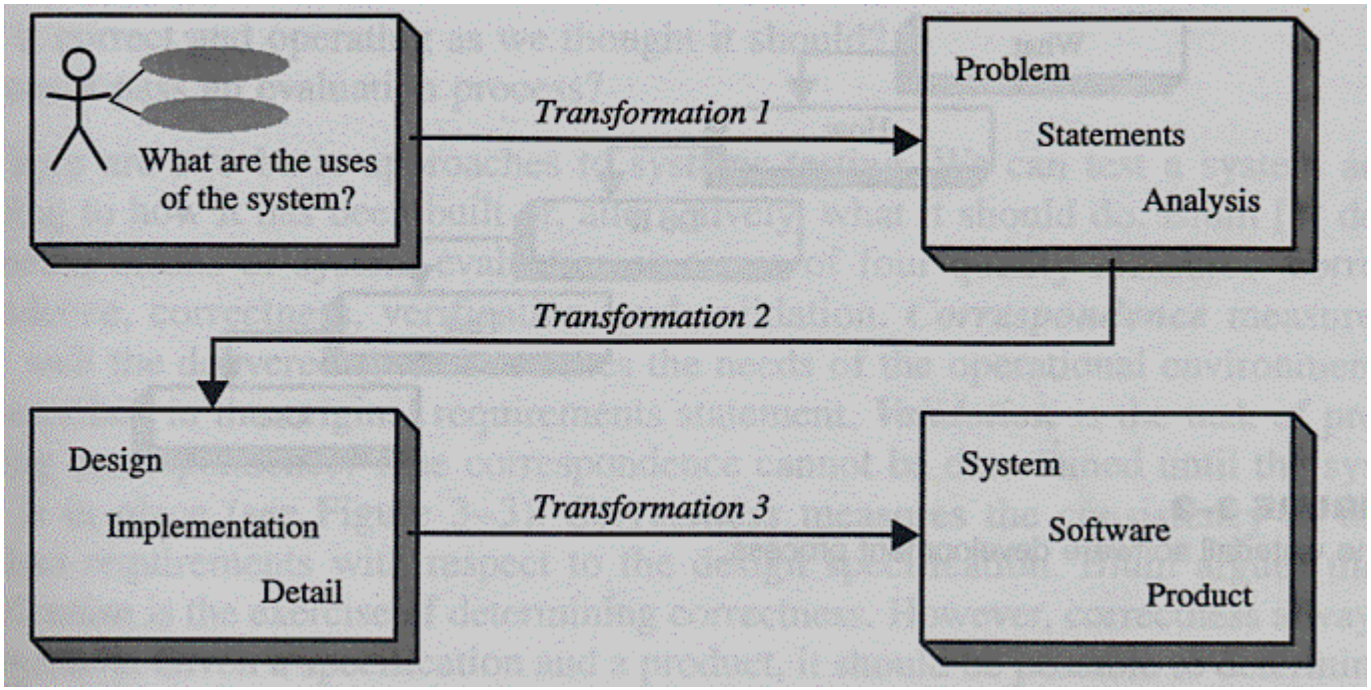
  ✓more time spent on gathering requirements, developing requirements model & analysis model, then turn into design model

  ✓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.

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



1. transformation 1(analysis) - translates user's need into system's requirements & responsibilities

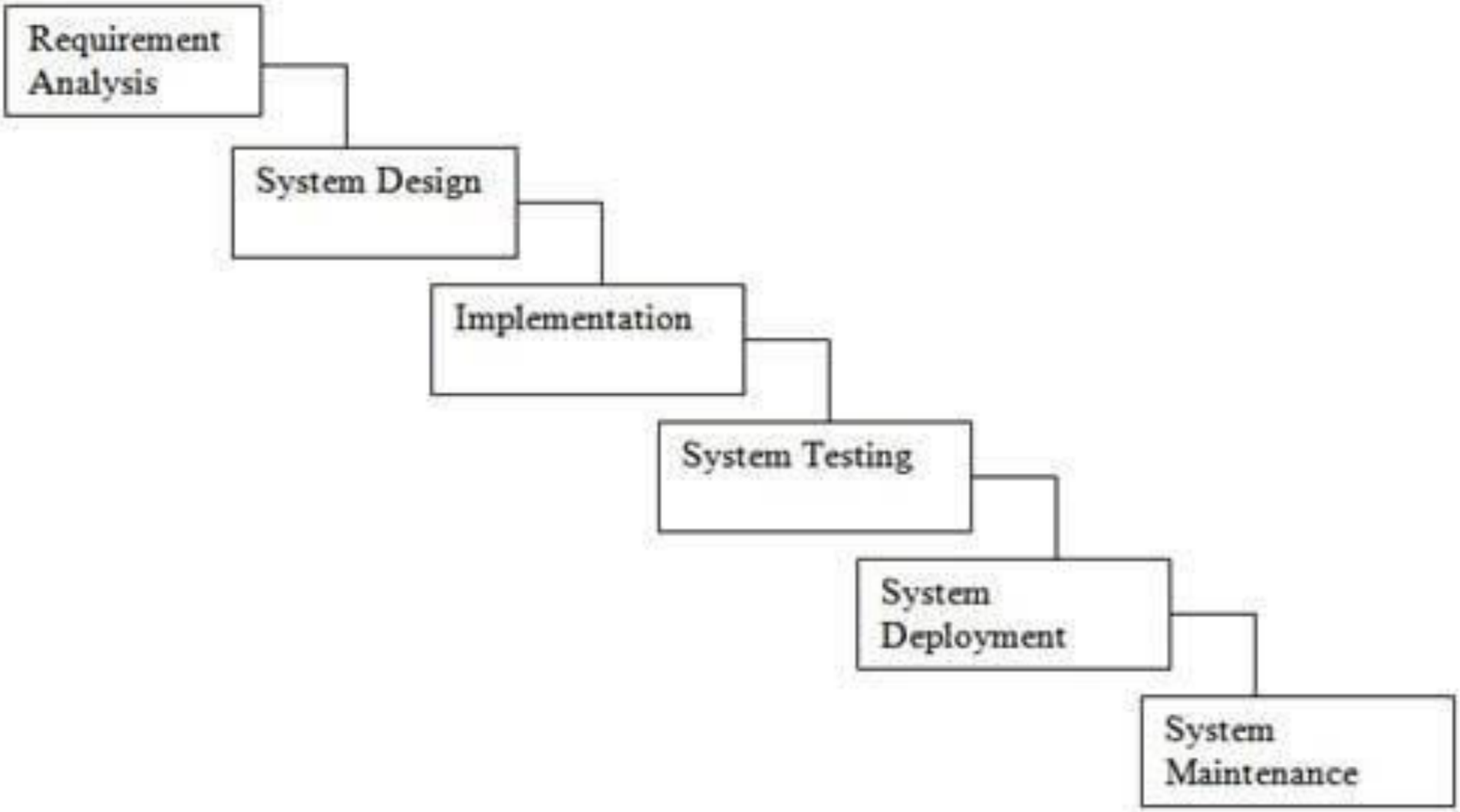
2. transformation 2 (design) - begins with problem statement, ends with detailed design that can be transformed into operational system

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

Waterfall software development process



Waterfall Model - © www.SoftwareTestingHelp.com

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**

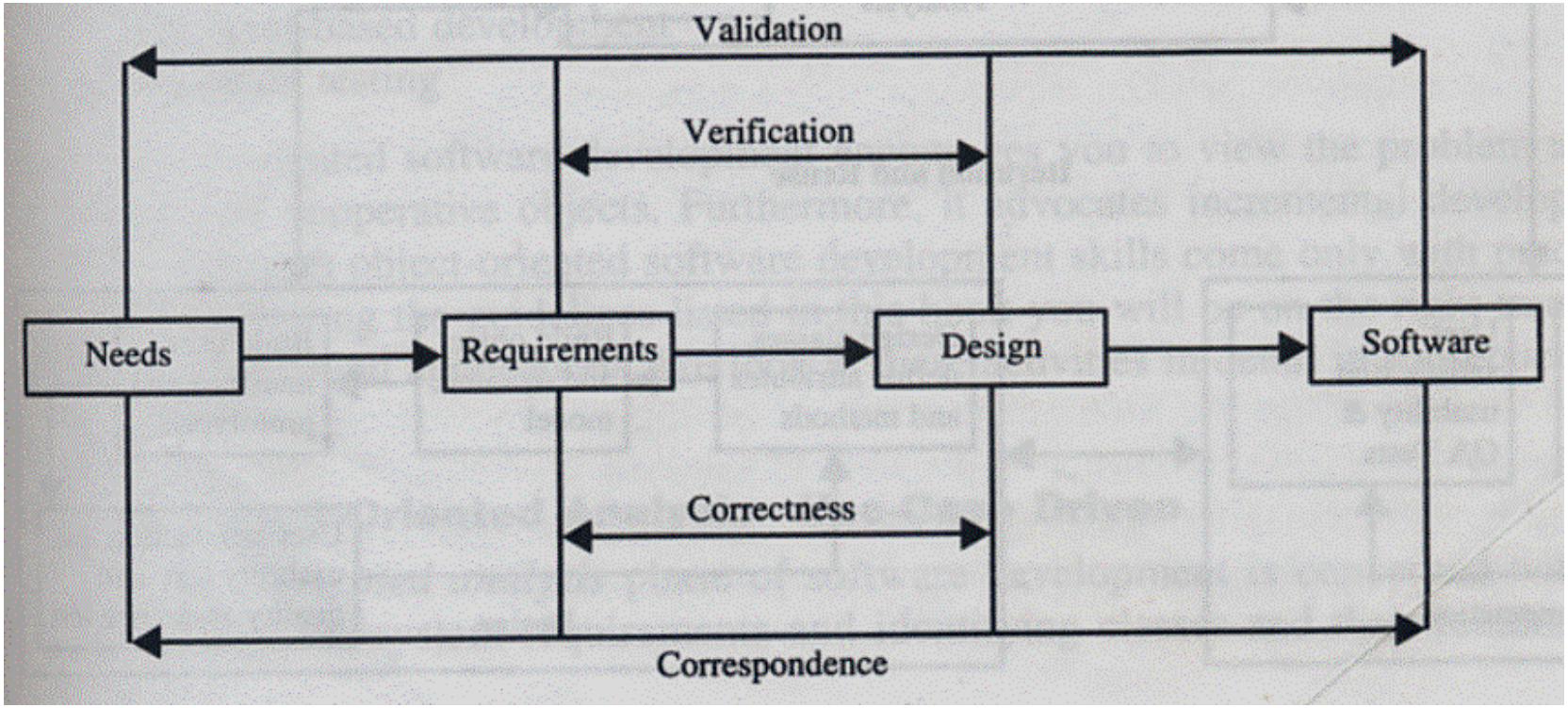
task of predicting correspondence (true correspondence only determined after system is in place)

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)



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 pproach: 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.

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.

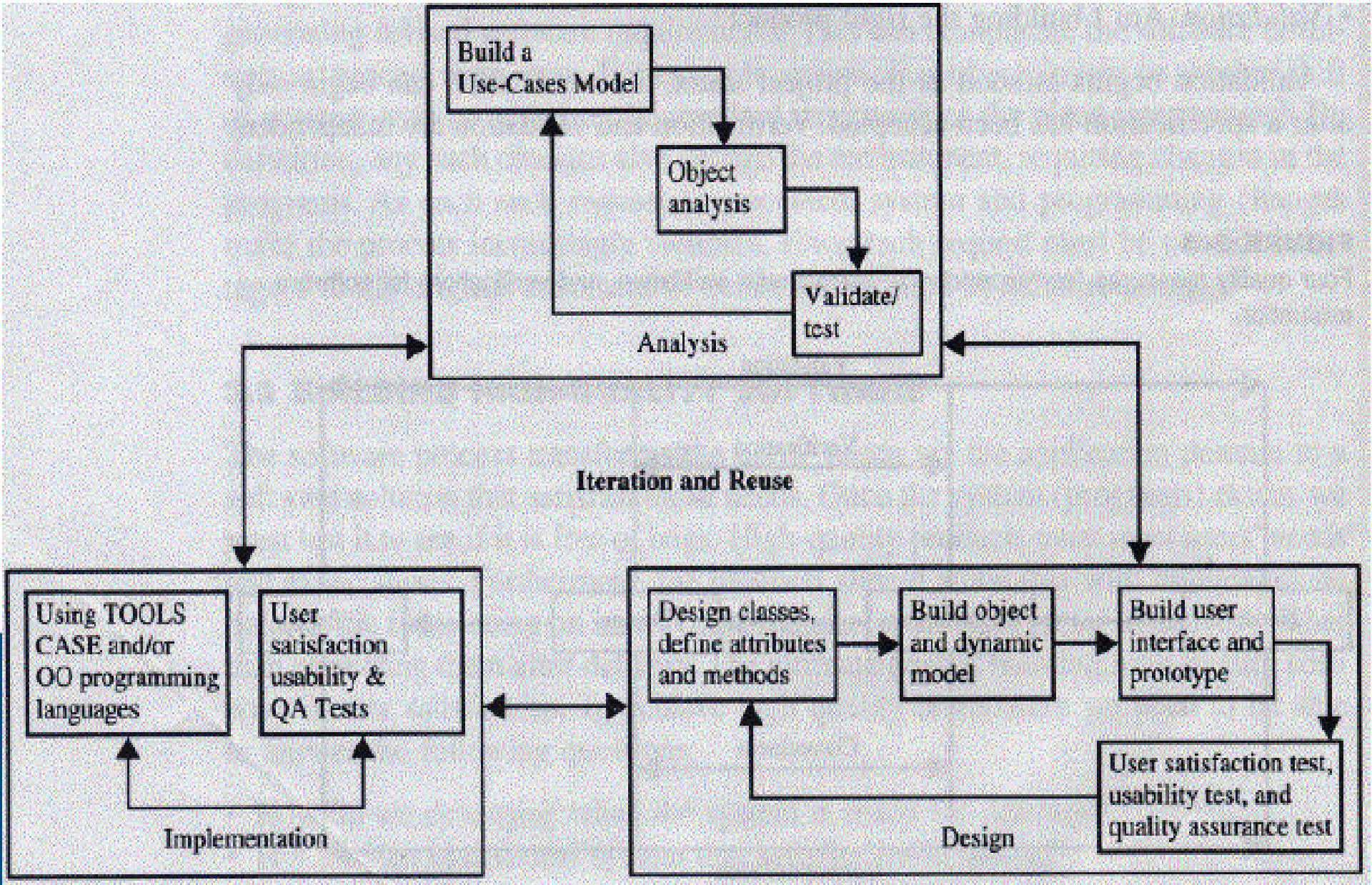
The designer decides whether –

- 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

Summary

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

References

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

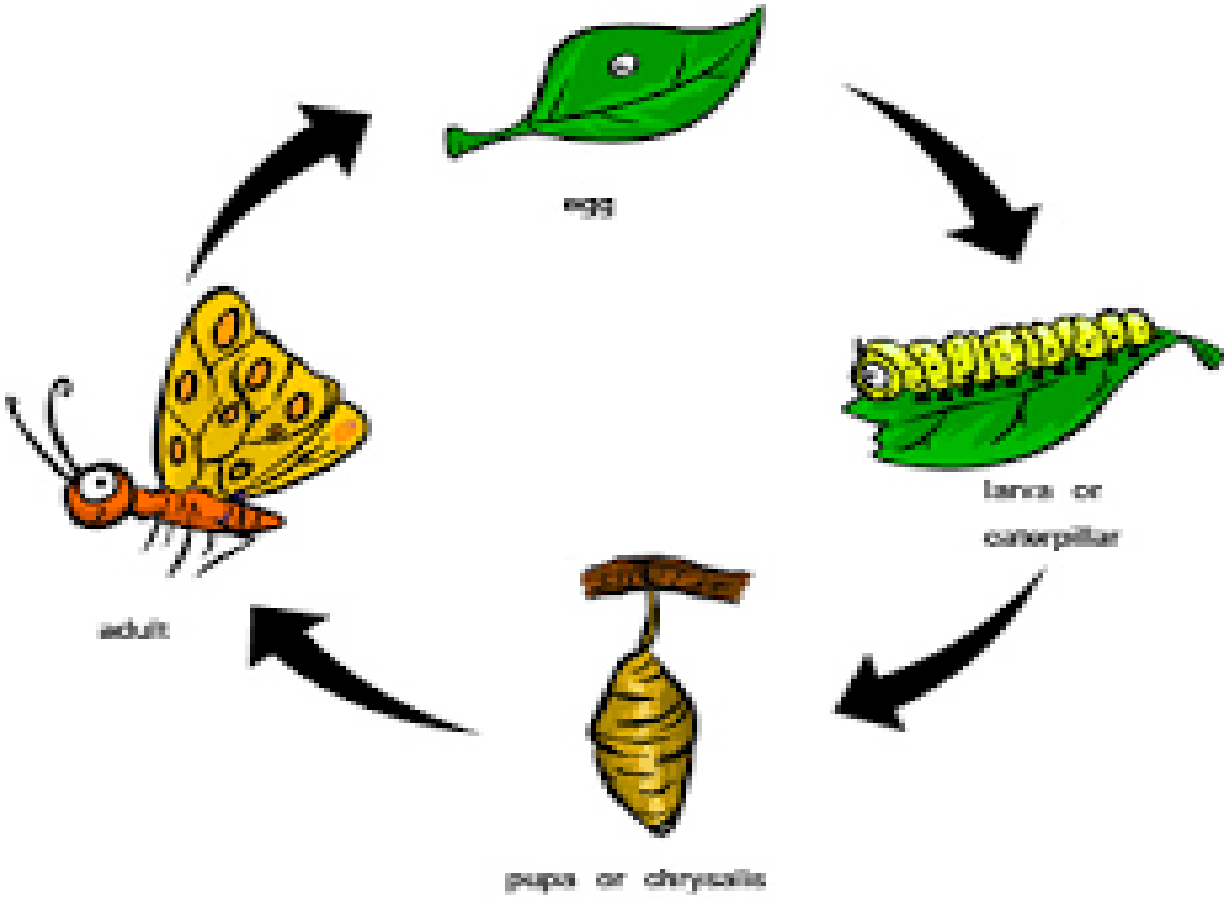
Stimulating Questions

1. How is software verification different form validation?

Thank You !!!

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

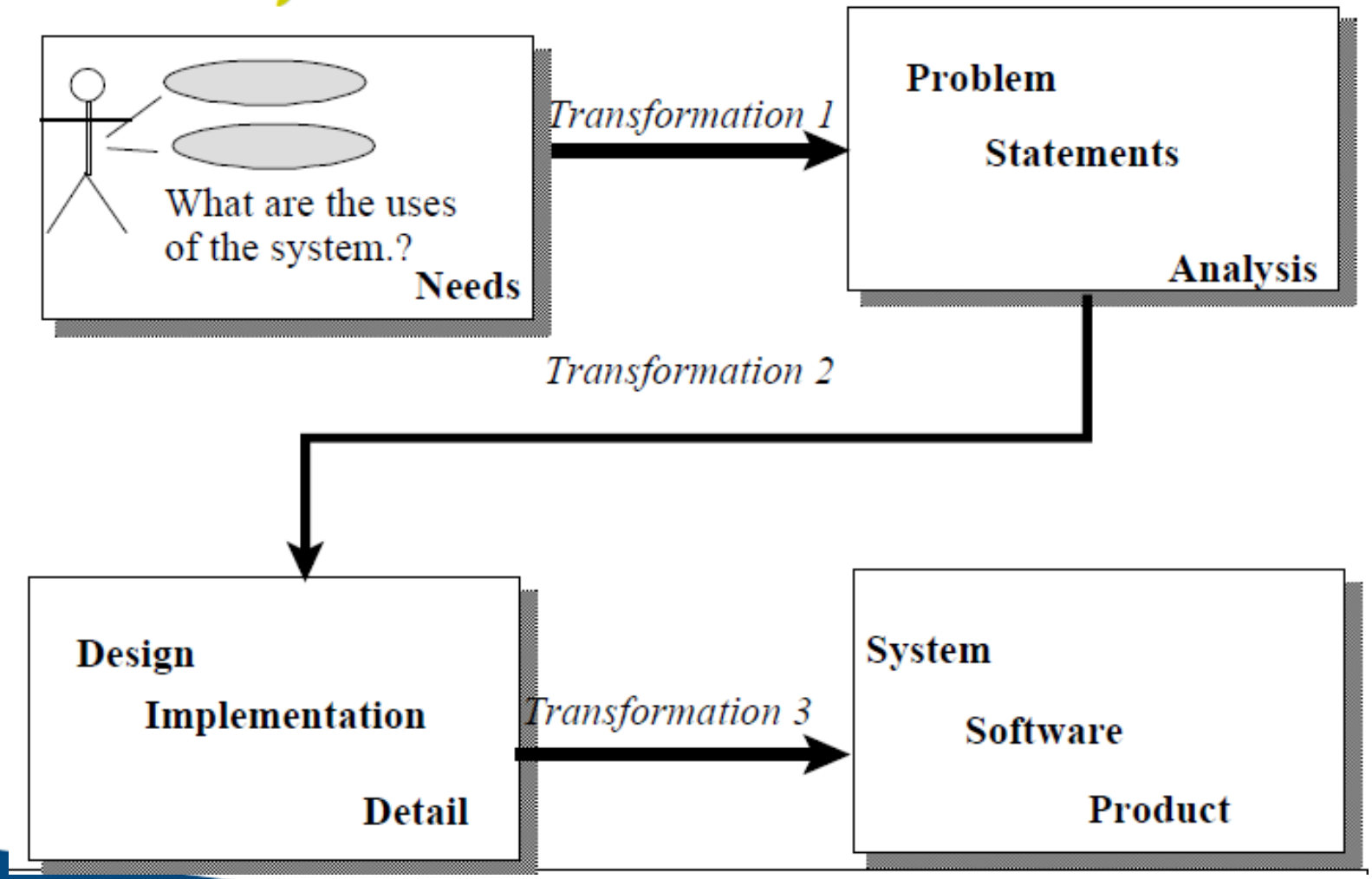
Evocation



Goals (Con't)

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

Software Process (Con't)

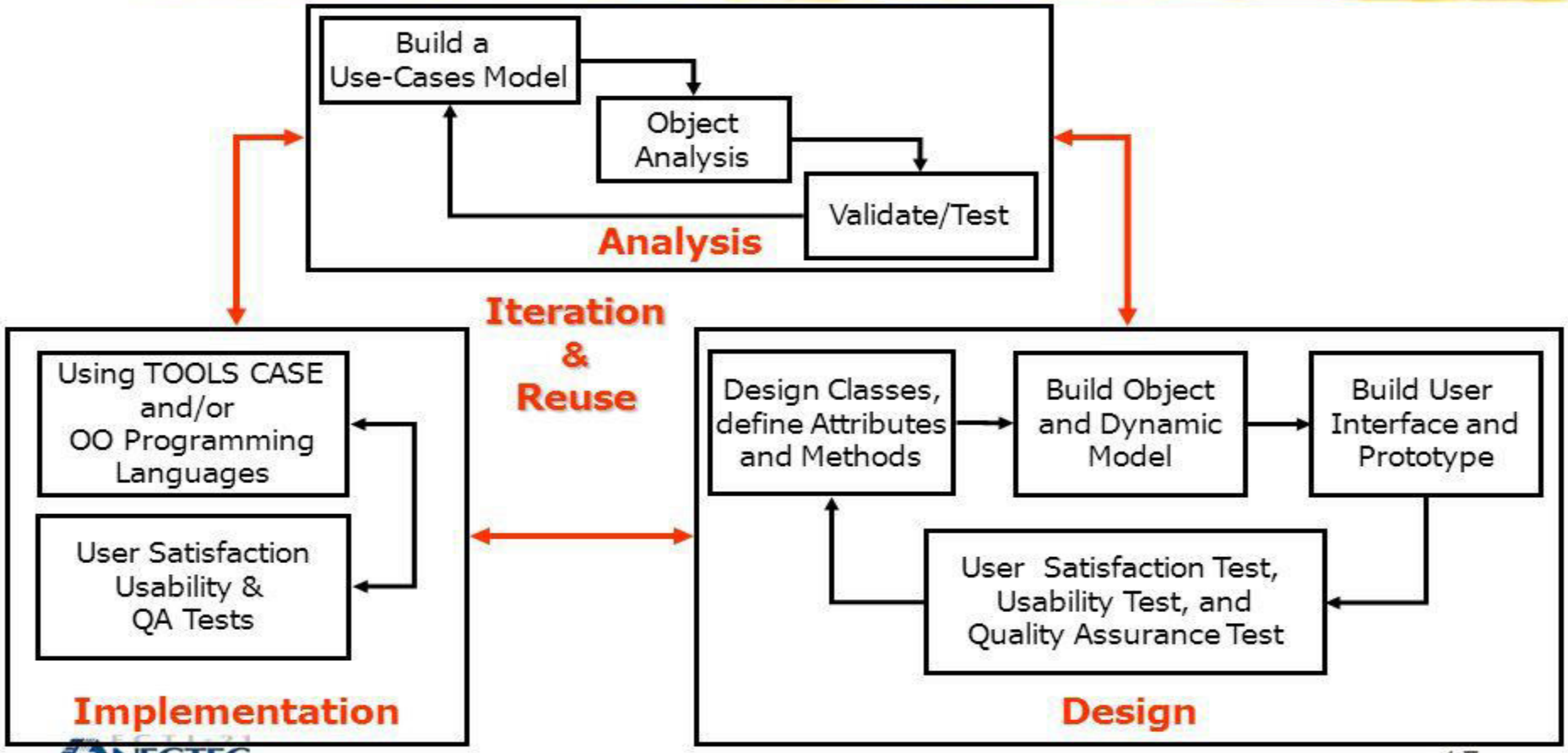


1. transformation 1(analysis) - translates user's need into system's requirements & responsibilities

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



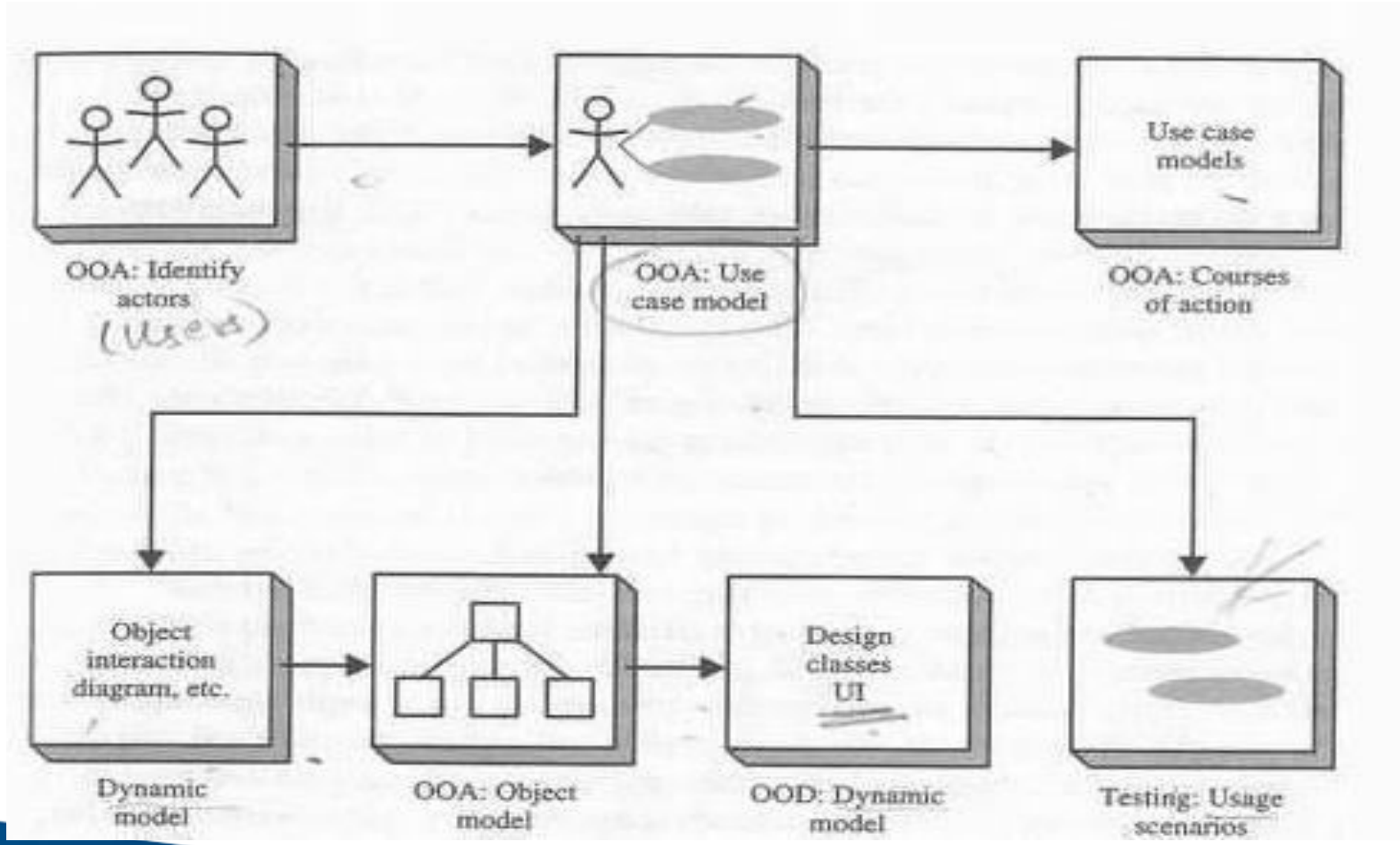
NECTEC

Object-Oriented Systems Development activities

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

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]



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

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

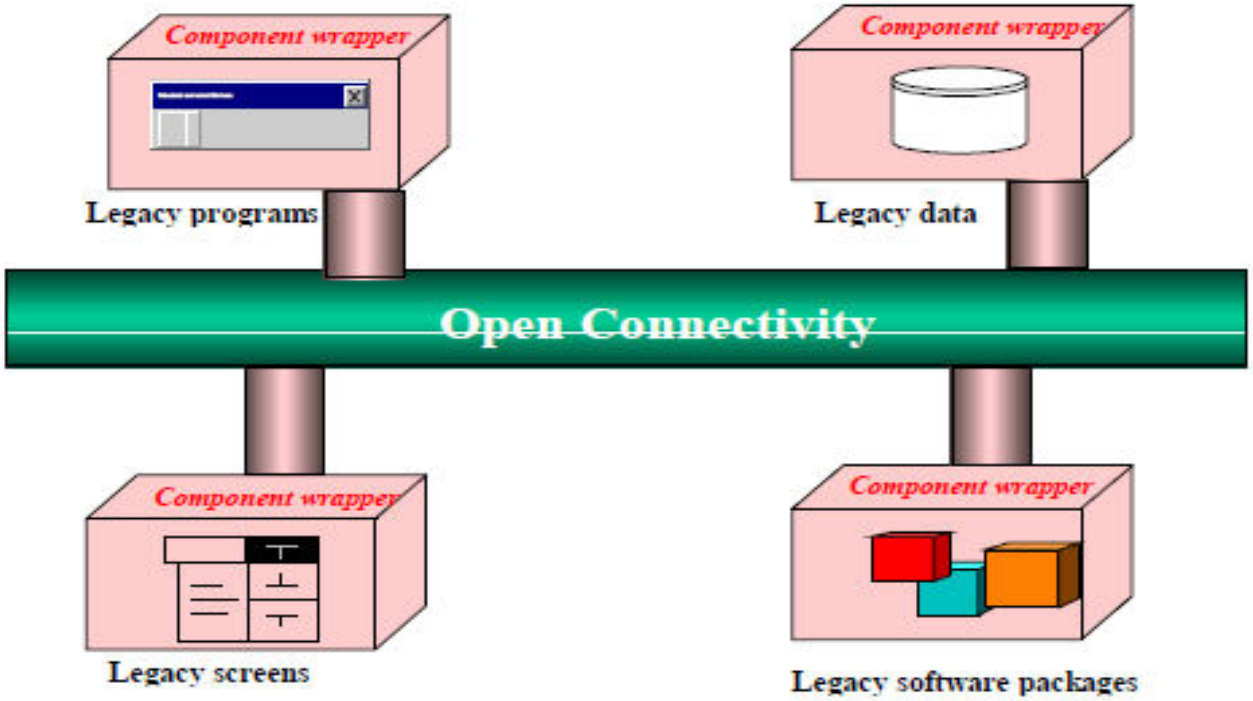
Implementation

- 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.

Component-based development (CBD) Con't)



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.**

Rapid Application Development (RAD) (Con't)

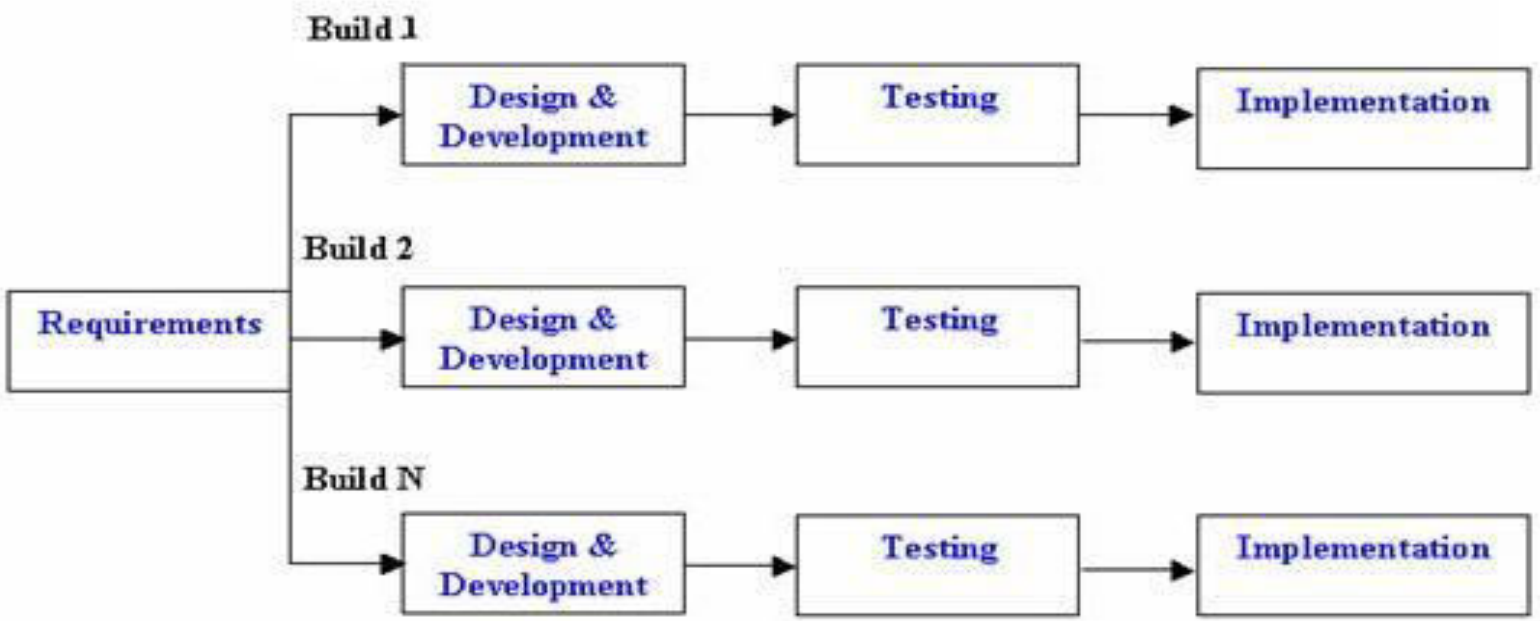
- RAD does not replace SDLC but complements it, since it focuses more on process description and can be combined perfectly with the object-oriented approach.

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.



Incremental Life Cycle Model

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 ?

References

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

Object-Oriented Methodologies

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.

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?

Types

- **There are three types of Object Oriented Methodologies**

1. Object Modeling Techniques (OMT)
2. Object Process Methodology (OPM)
3. Rational Unified Process (RUP)

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

- **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.

- 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

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.

III. Functional Model

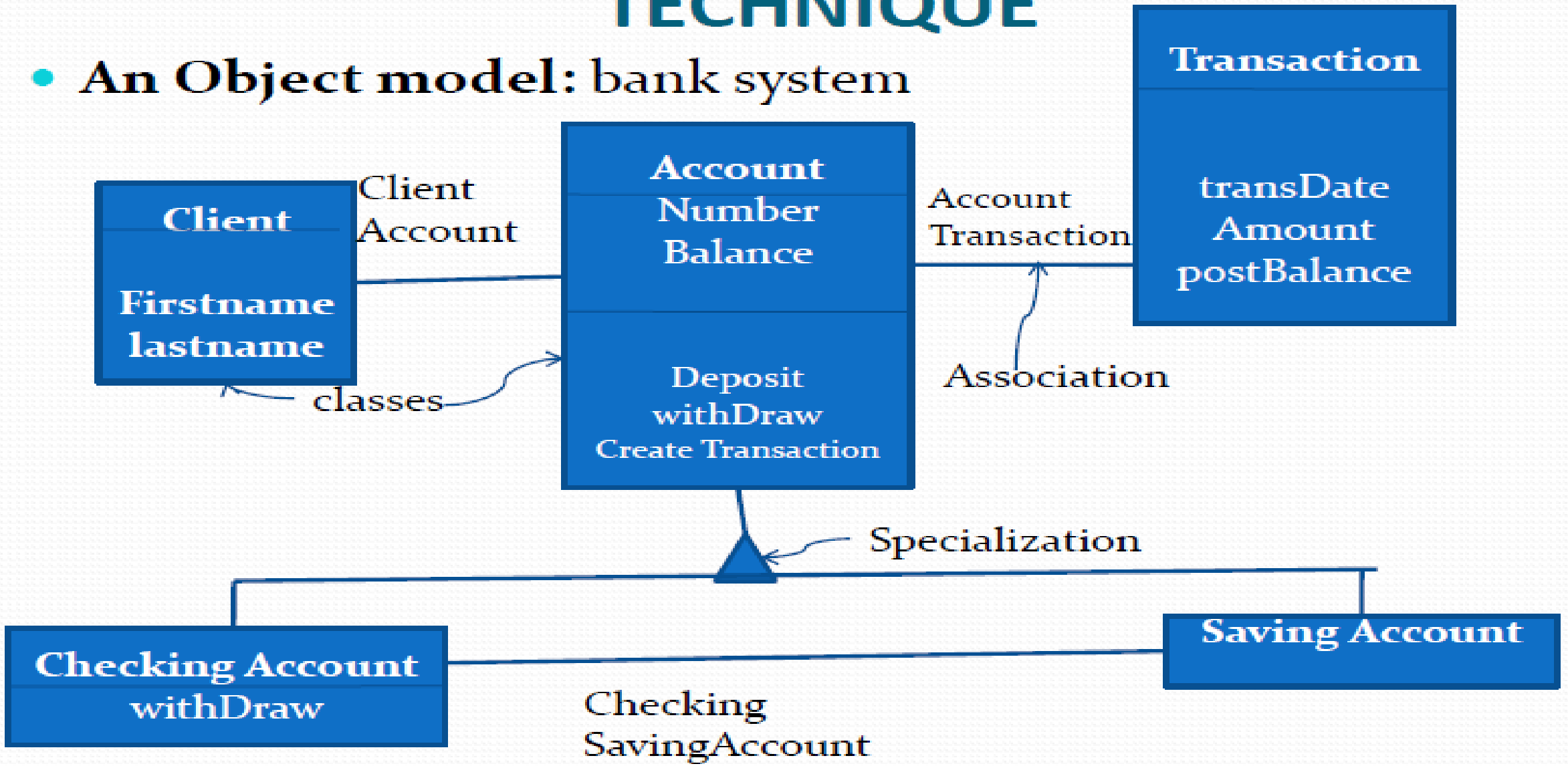
- It describes the data transformations of the system.
- It describes the flow of data and the changes that occur to the data throughout the system.

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.

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.

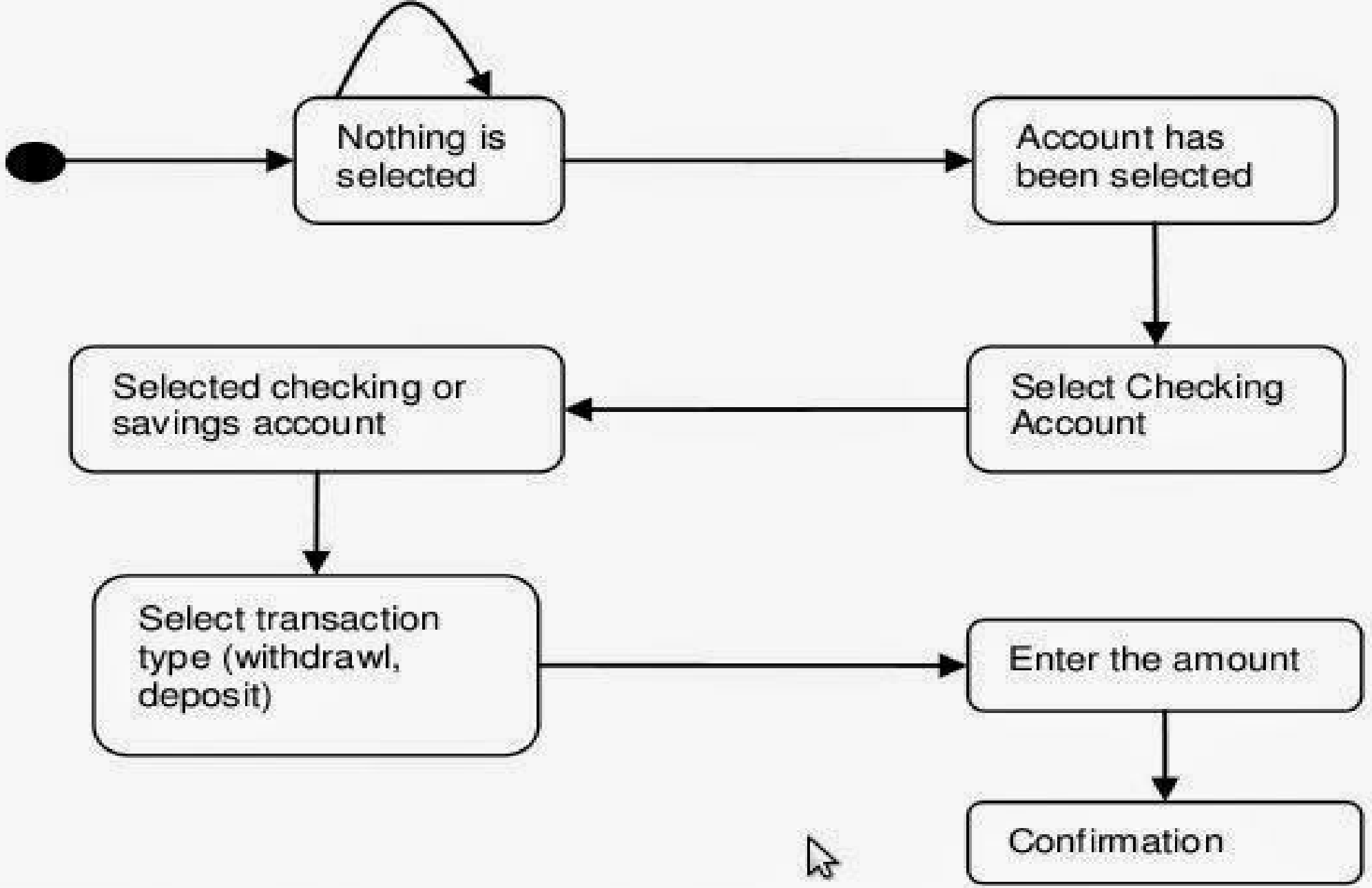


Fig. 4.2: State transition diagram for the bank application user interface. The round boxes represent states and the arrows represent transitions.

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

Data Flow Diagram

- Four primary symbols

Process- any function being performed

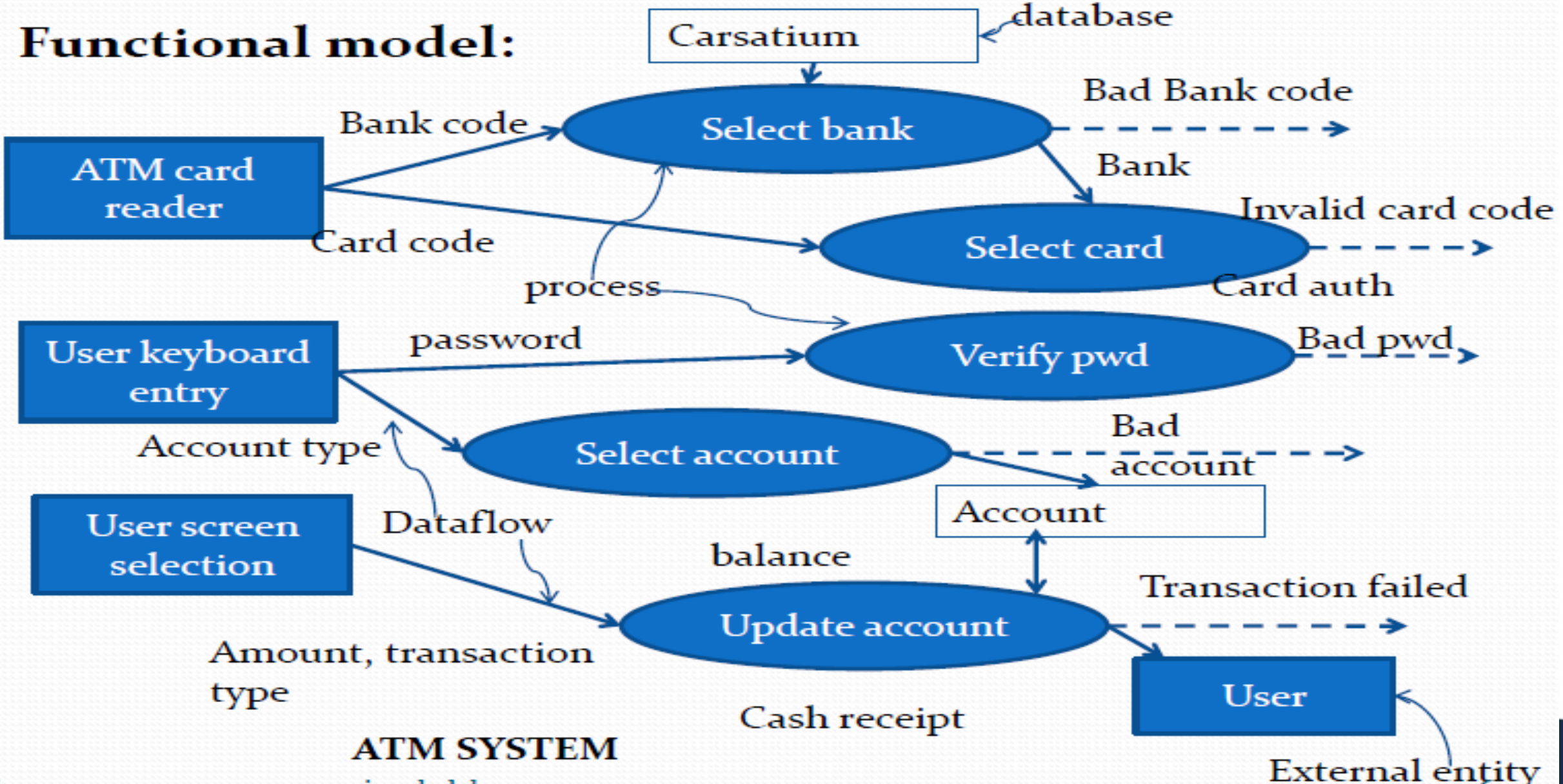
Data Flow- Direction of data element movement

Data Store – Location where data is stored

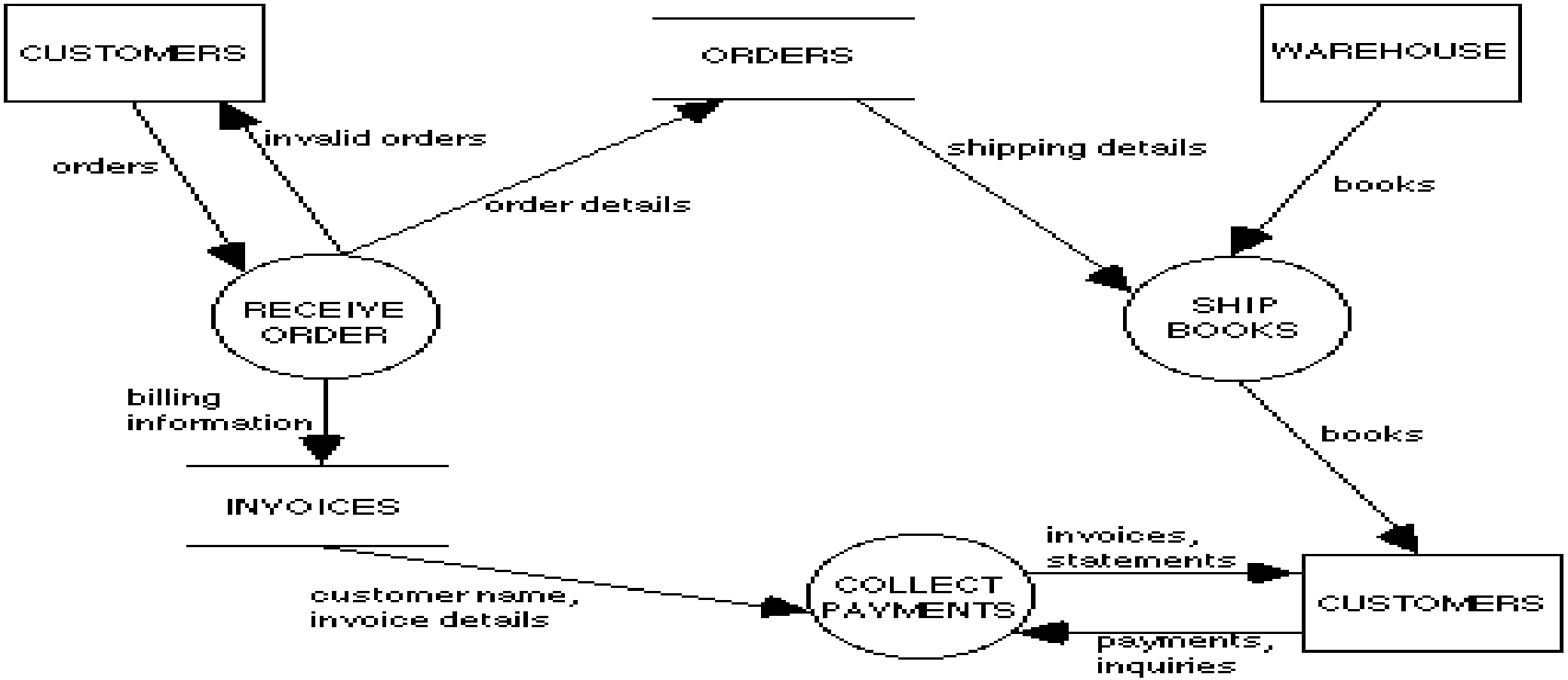
External Entity-Source or Destination of a data element

RUMBAUGH ETAL'S OBJECT MODELING TECHNIQUE

- **Functional model:**



Example for Data Flow Diagram



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

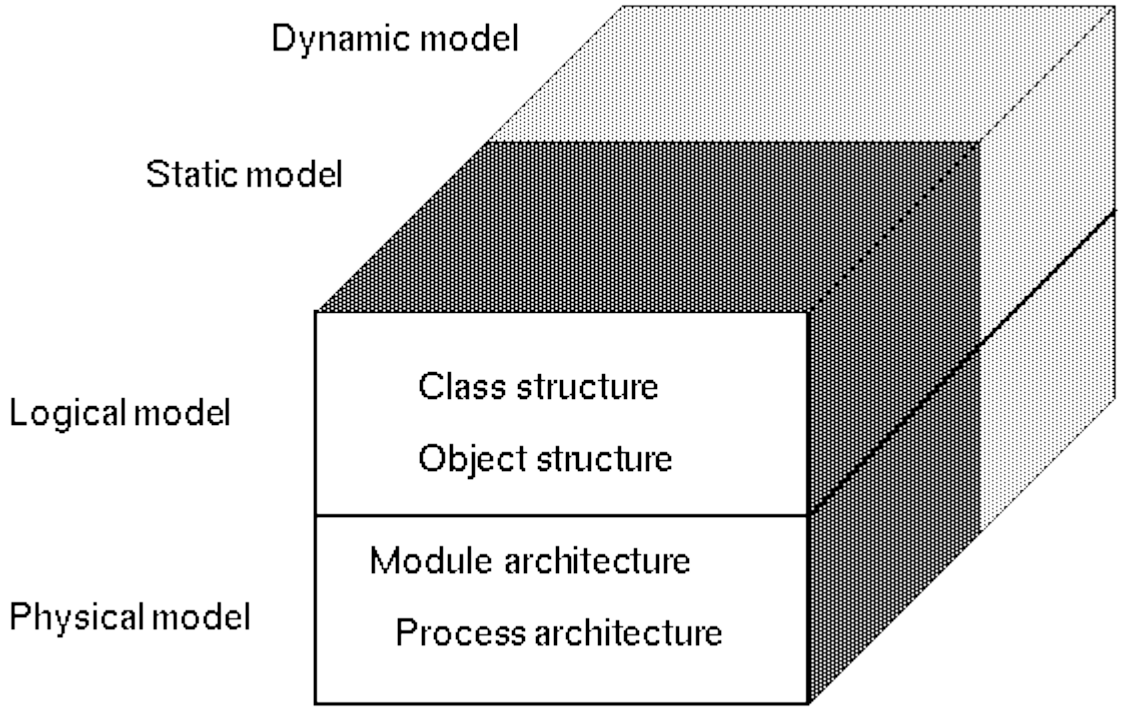
Booch Methodology

- 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

- 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

- 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

Overview


Dynamic model
Static model
Logical model
Class structure
Object structure
Physical model
Module architecture
Process architecture

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

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

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

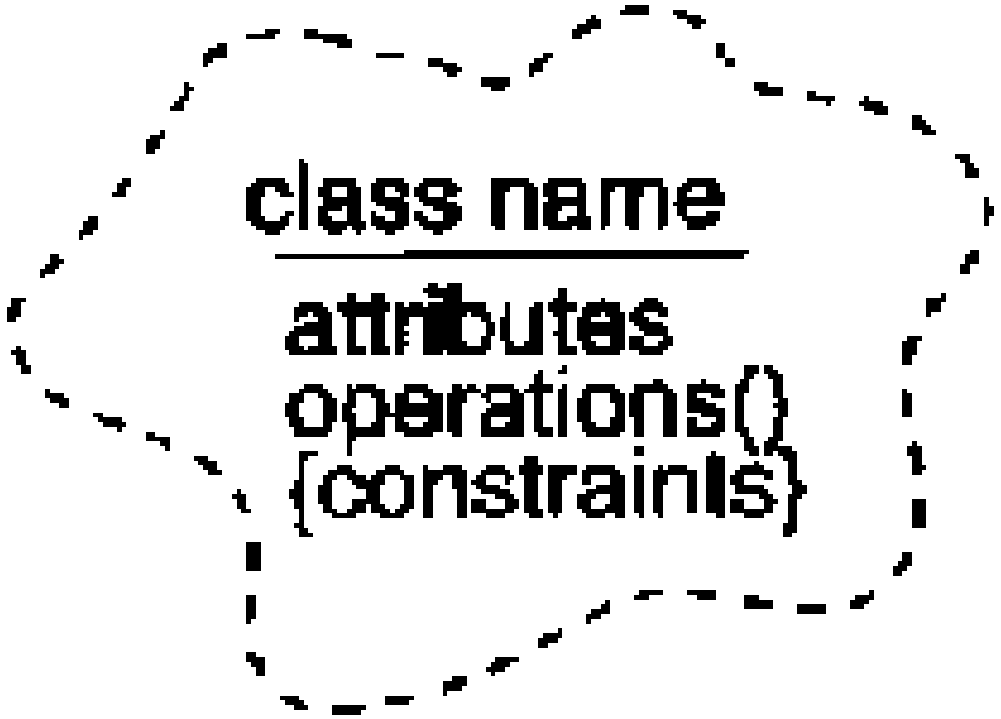
It consists of the following Six diagrams:

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

Class Diagrams

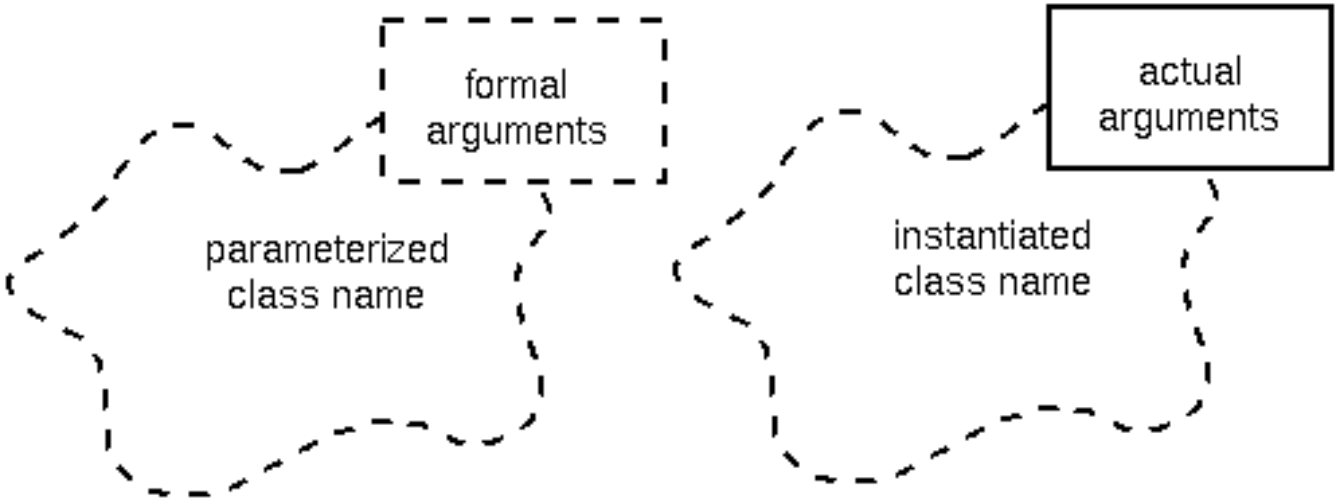
- Class:
  - ➢The modular unit of OO software decomposition
  - ➢ As a type, a class represents a set of similar run-time data elements, or *objects*
  - ➢Share a common structure and a common behavior
  - ➢As a decomposition unit
    - a class is a group of related **services** packed together under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

Notations for Class

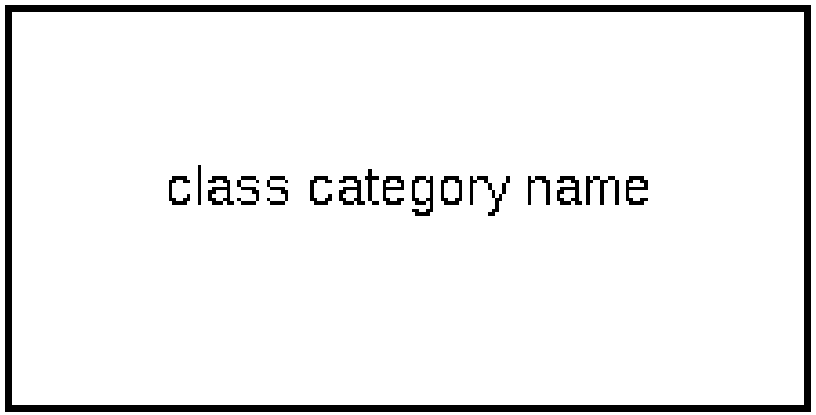


- class Class Name {
  public:
  // constructors
  // destructors
  // operators
  // modifiers
  // selectors
  protected:
  // member objects
  private:
  // friends
  };

Parameterized class:



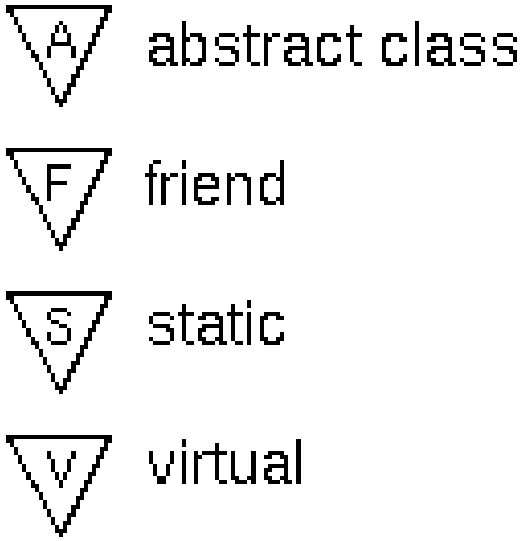
Class category:



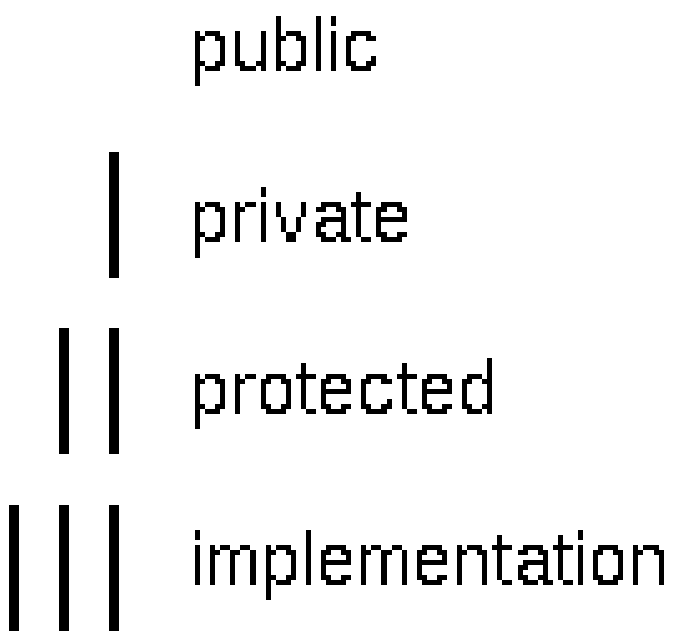
- A logical collection of classes
- Some of which are visible to other class categories, and others are hidden
- The classes in a class category collaborate to provide a set of services.

- 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)

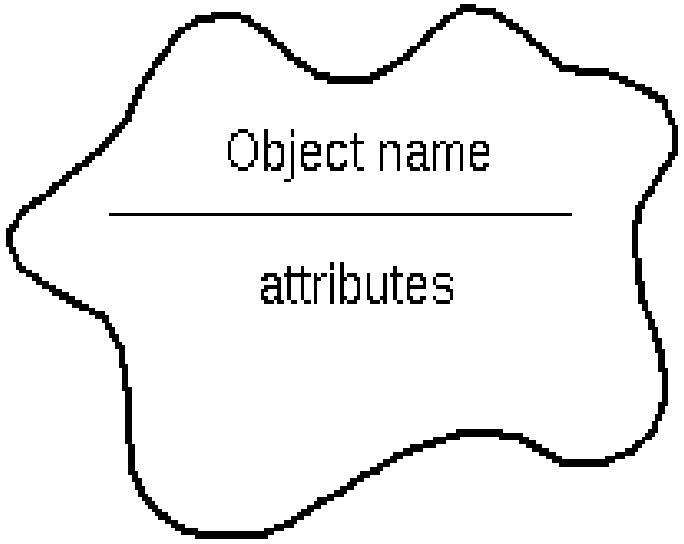


Export control (for attributes/operations and relationships):



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

Two processes:

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

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

- **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

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

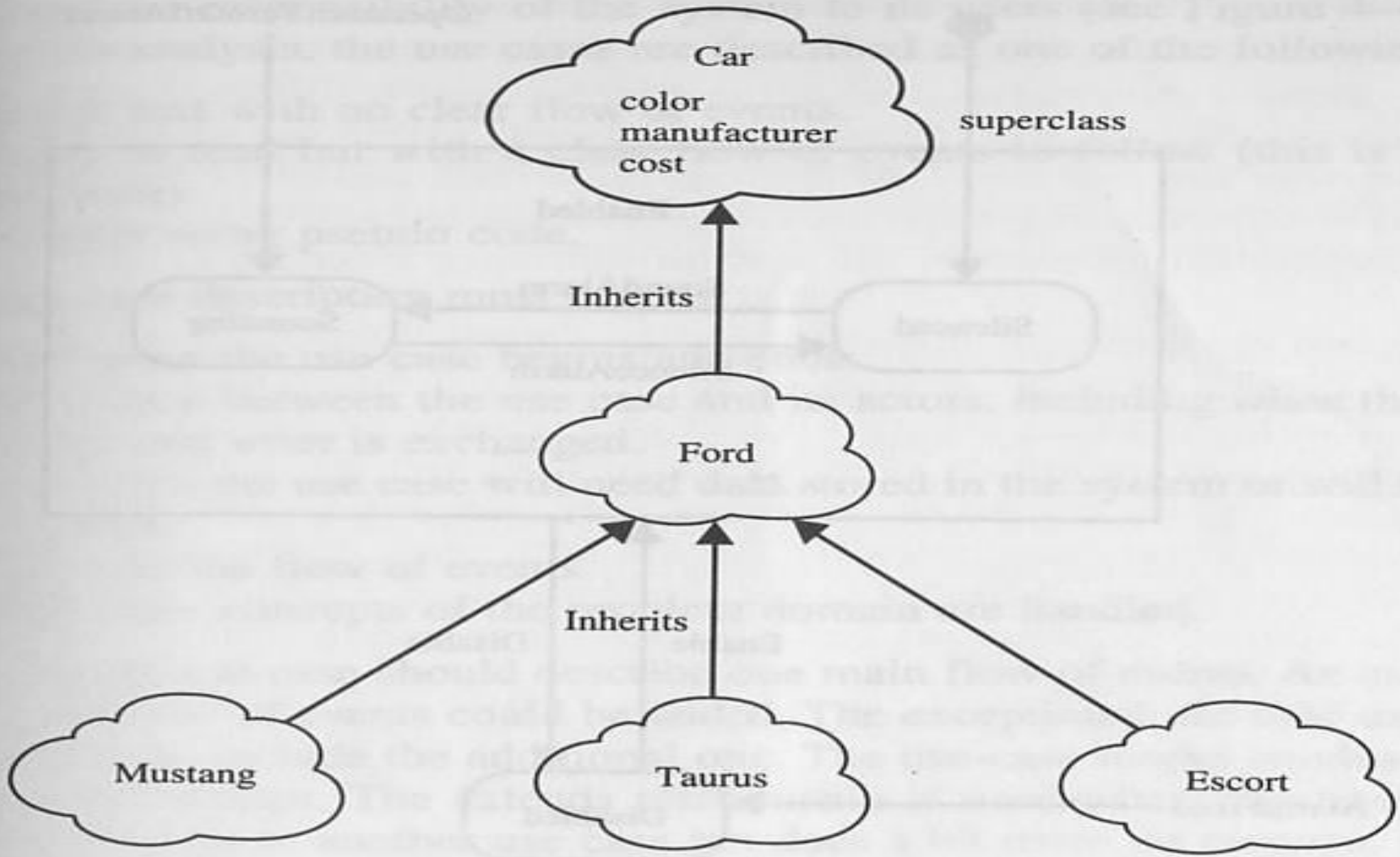
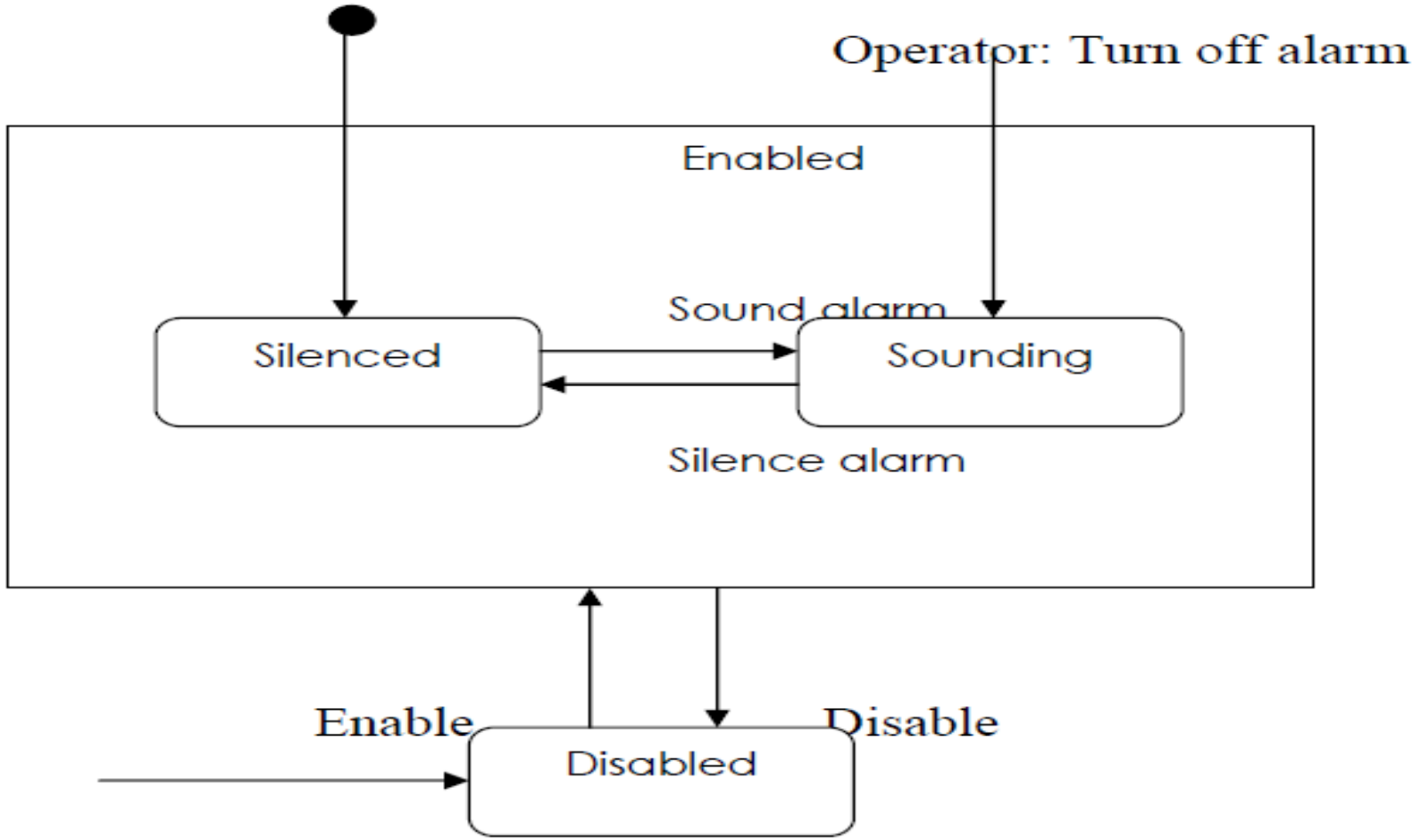


FIGURE 4–4
Object modeling using Booch notation. The arrows represent specialization; for example, the class Taurus is subclass of the class Ford.

An alarm state transition diagram with Booch



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-Wesly.
- https://www.slac.stanford.edu/BFROOT/www/doc/workbook_kiwi/coding/booch/method.html

Jacobson Methodology

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

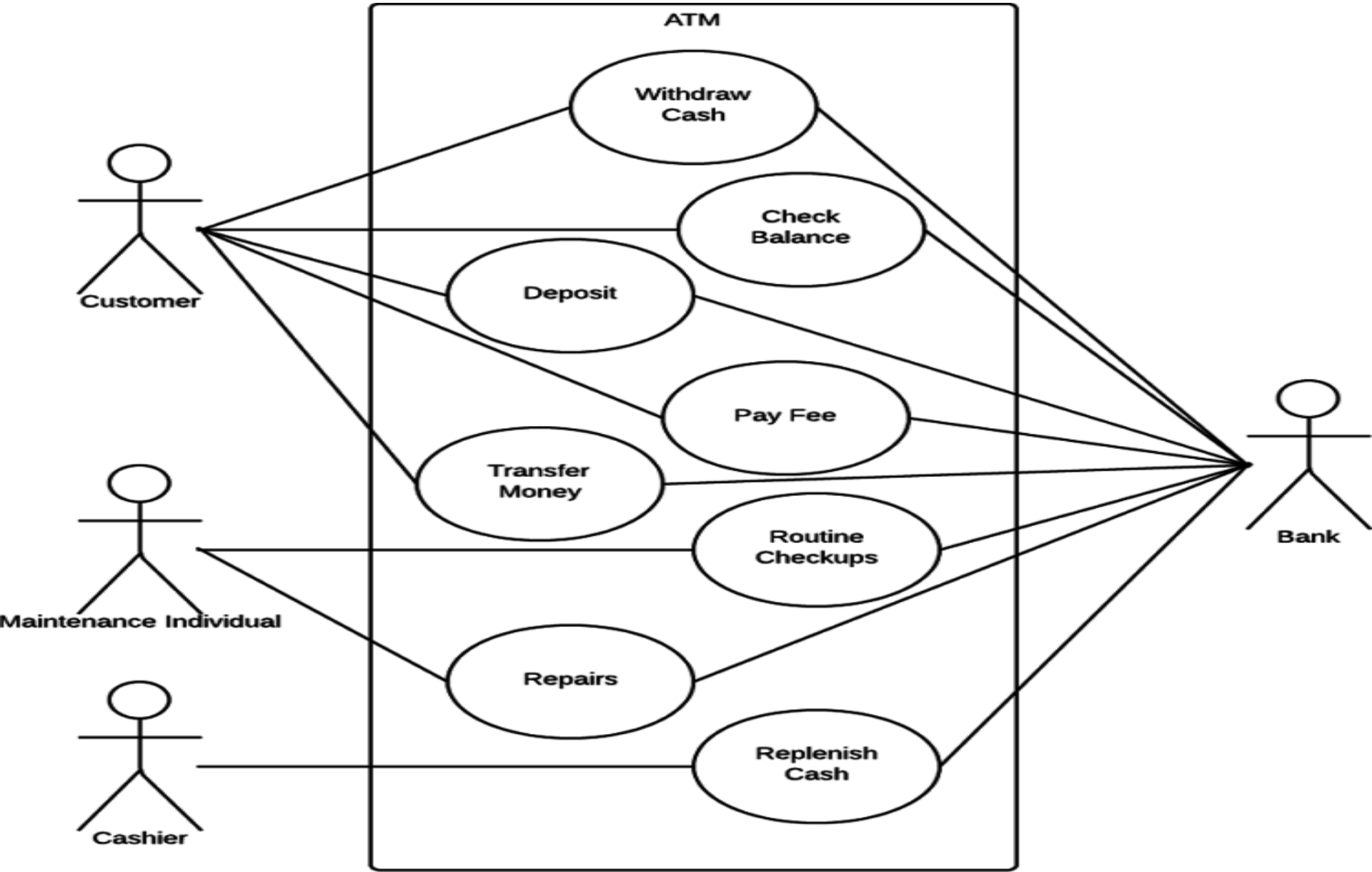
Use-Case

- 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.

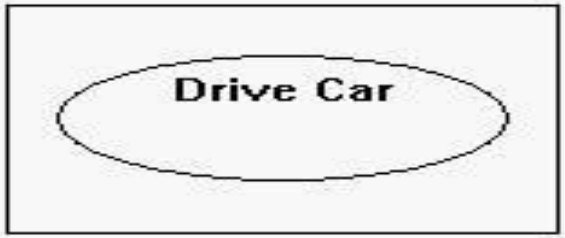
Sample Use-Case Diagram



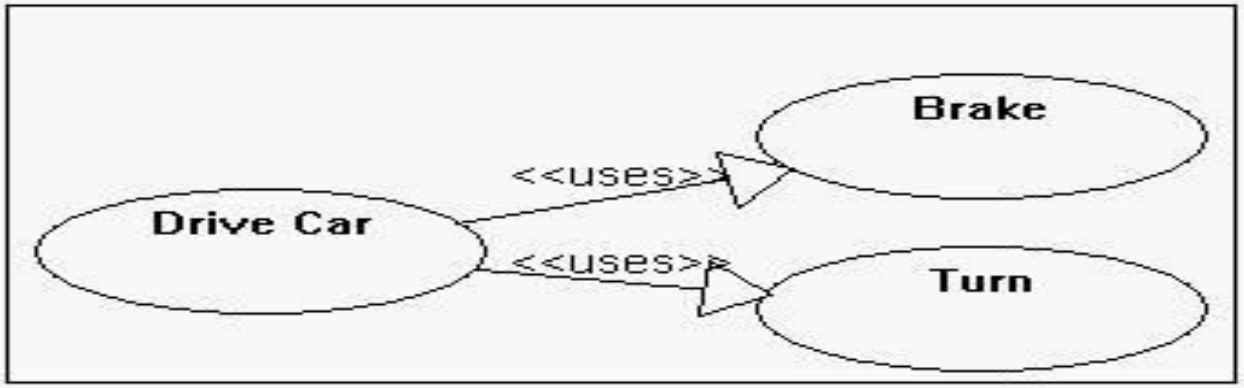
Extends and Uses Relationship

- **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.

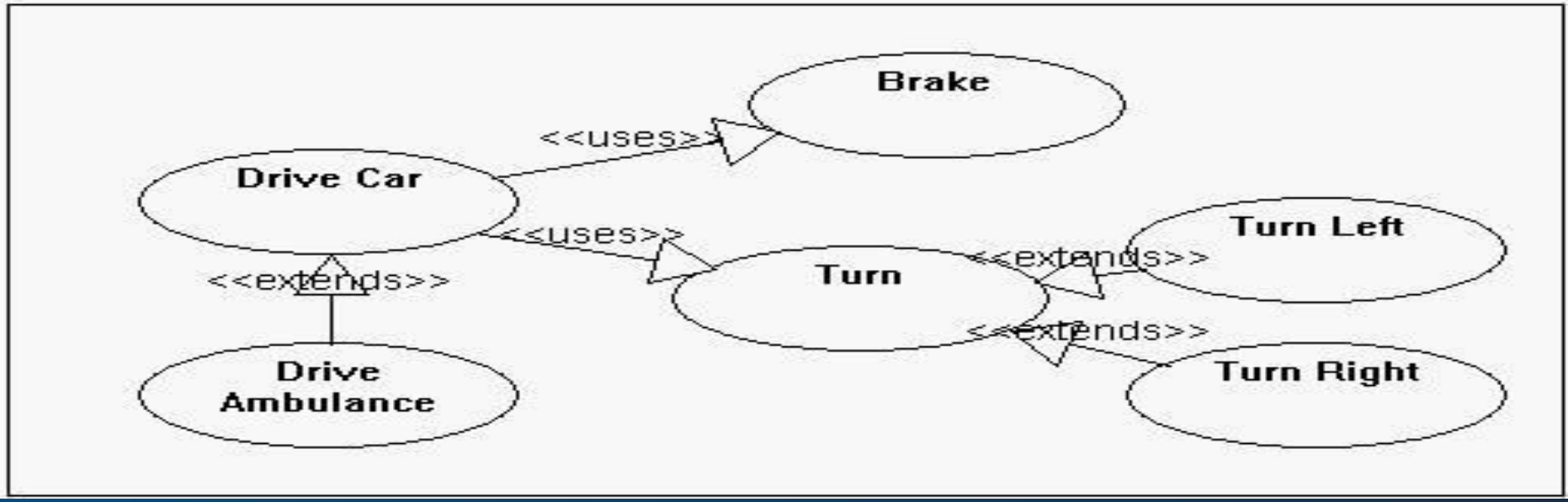
Evolution of a UML Use Case Diagram



... Becomes ...

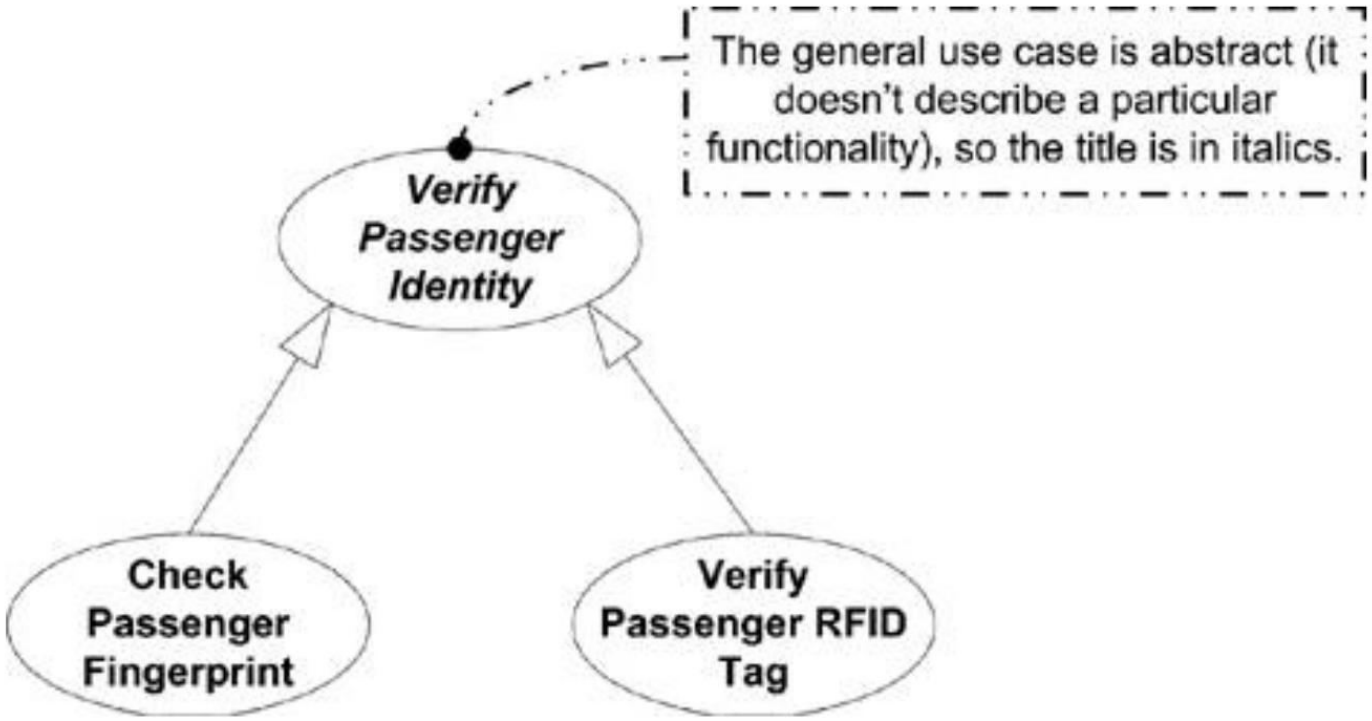


... Which Becomes ...



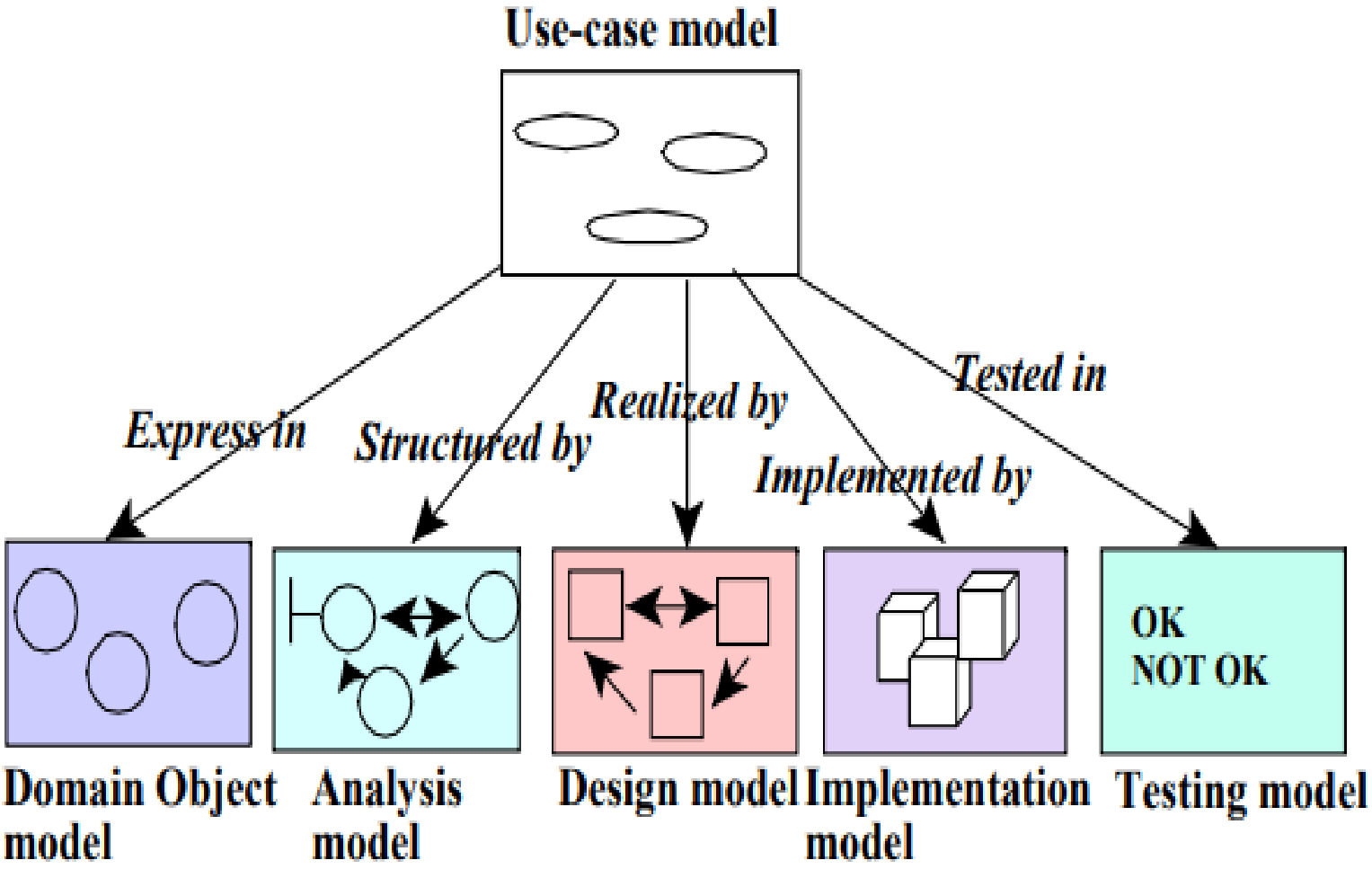
Use-Case Viewed as,

- 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.
Verify Passenger Identity
Check Passenger Fingerprint
Verify Passenger RFID Tag

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


Use-case model
Express in
Structured by
Realized by
Implemented by
Tested in
OK
NOT OK
Domain Object model
Analysis model
Design model
Implementation model
Testing 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 know 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 know 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	2010s
C pointers	Java garbage collection
Emacs	Eclipse
Math library	Frameworks
Ad hoc programming	Agile methodology
Waterfall	Evolution/continuous integration
Flowcharts	UML
Write your own sort	Copy from Stack Overflow
Computer room	Computer in your pocket
Hard disk	Cloud
Text terminals	Touch screens
Email	Internet of Things
No regulation	No regulation

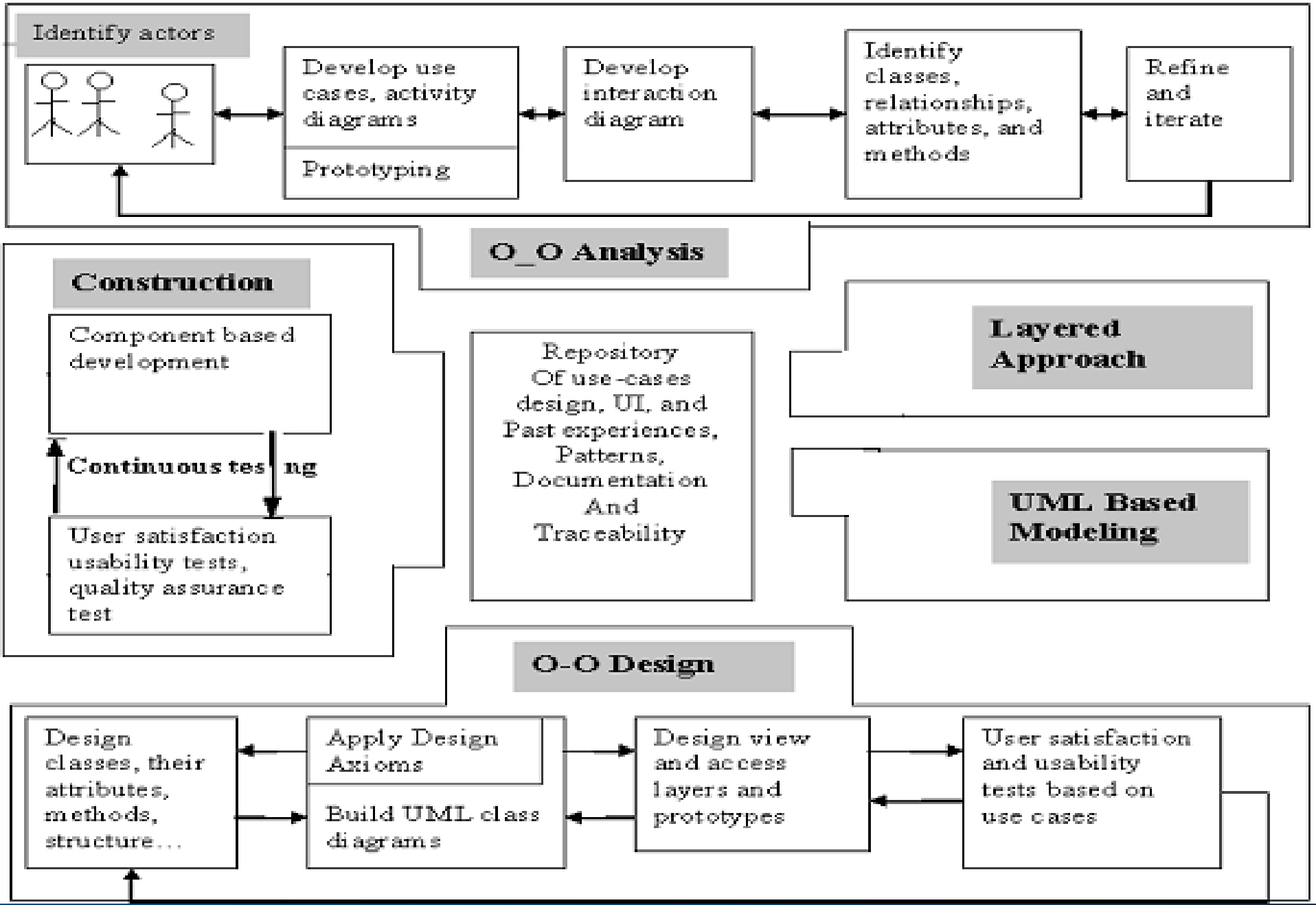
THE UNIFIED APPROACH

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.

- **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.

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

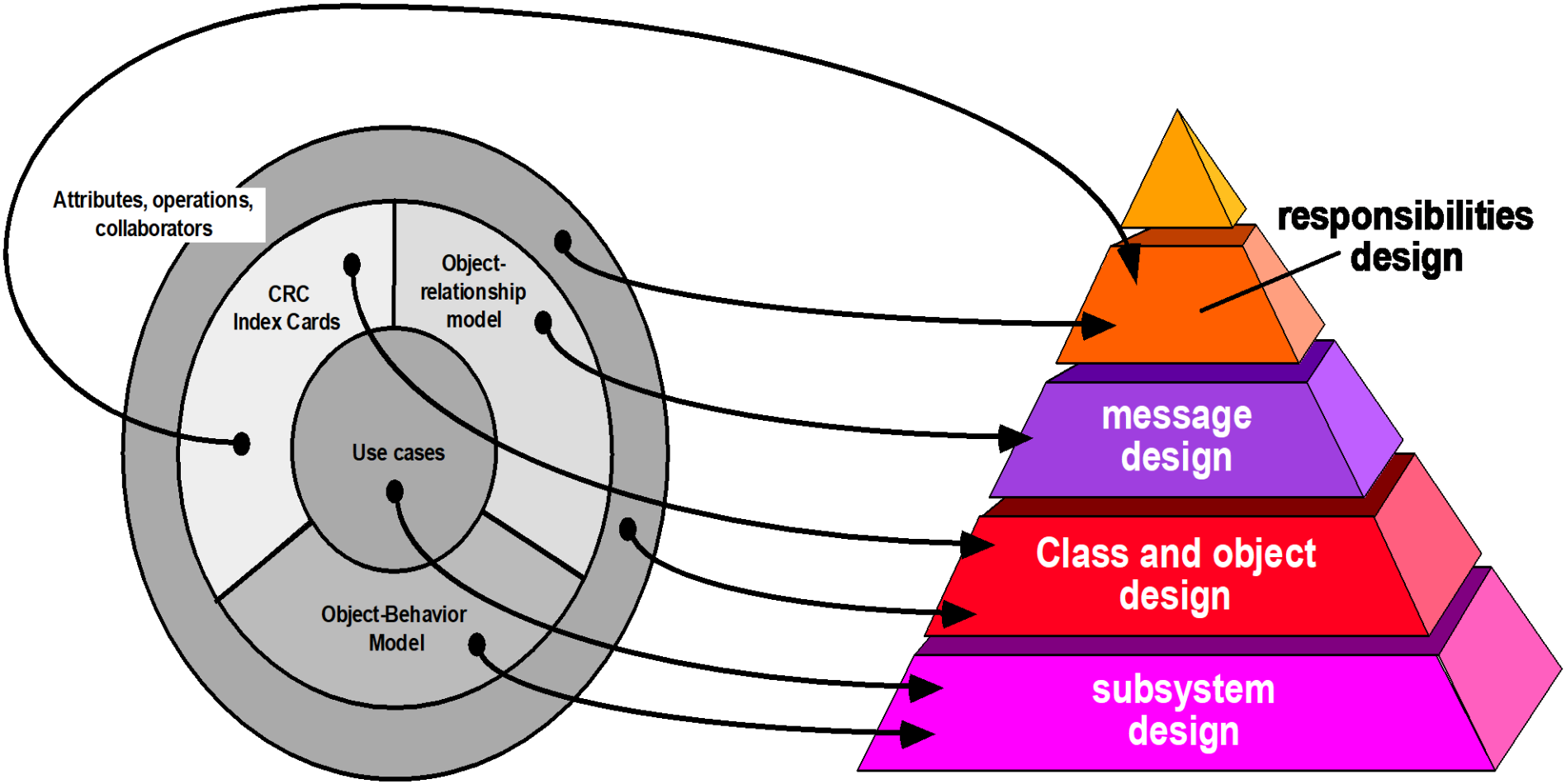
- Booch, provides the **most comprehensive** object-oriented design method.
- Rumbaugh et al.'s and Jacobson et al.'s high-level models provide **good avenues for getting started**.
- UA combines these by utilizing **Jacobson et al.'s *analysis and interaction diagrams*, Booch's *object diagrams*, and 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,…

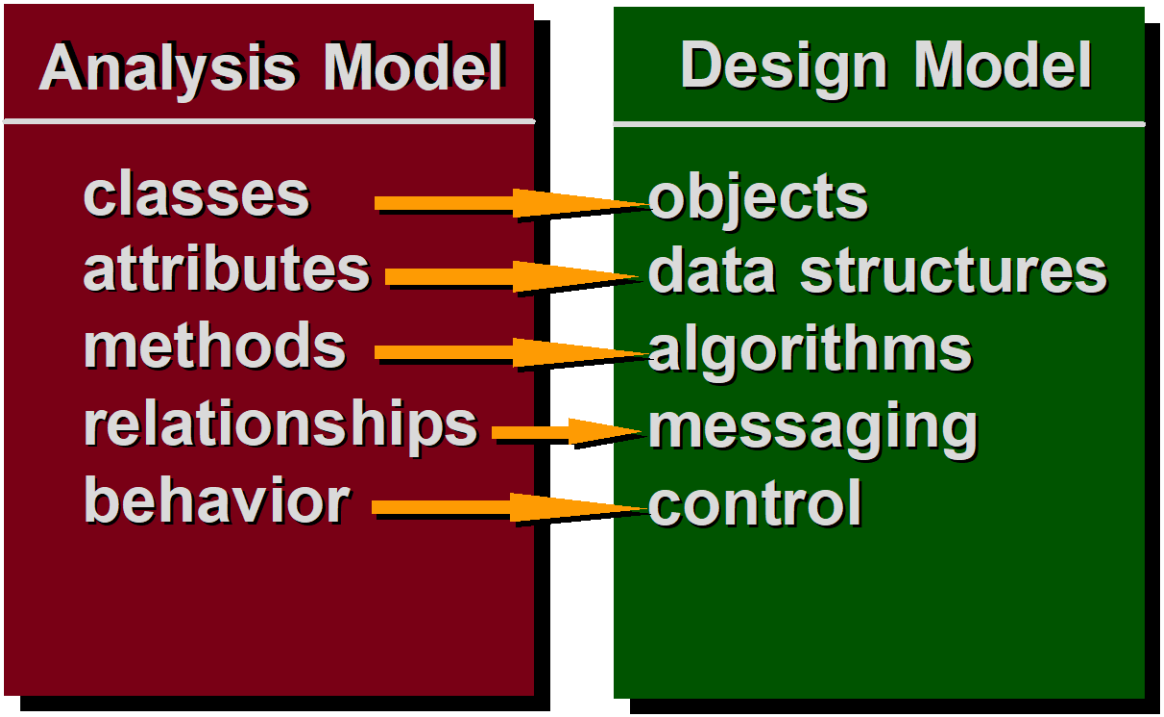
OOD Process consists of:

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


Attributes, operations, collaborators
CRC Index Cards
Object-relationship model
Use cases
Object-Behavior Model
responsibilities design
message design
Class and object design
subsystem design
THE ANALYSIS MODEL
THE DESIGN MODEL

OOA to OOD


Analysis Model
classes
attributes
methods
relationships
behavior
Design Model
objects
data structures
algorithms
messaging
control

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.

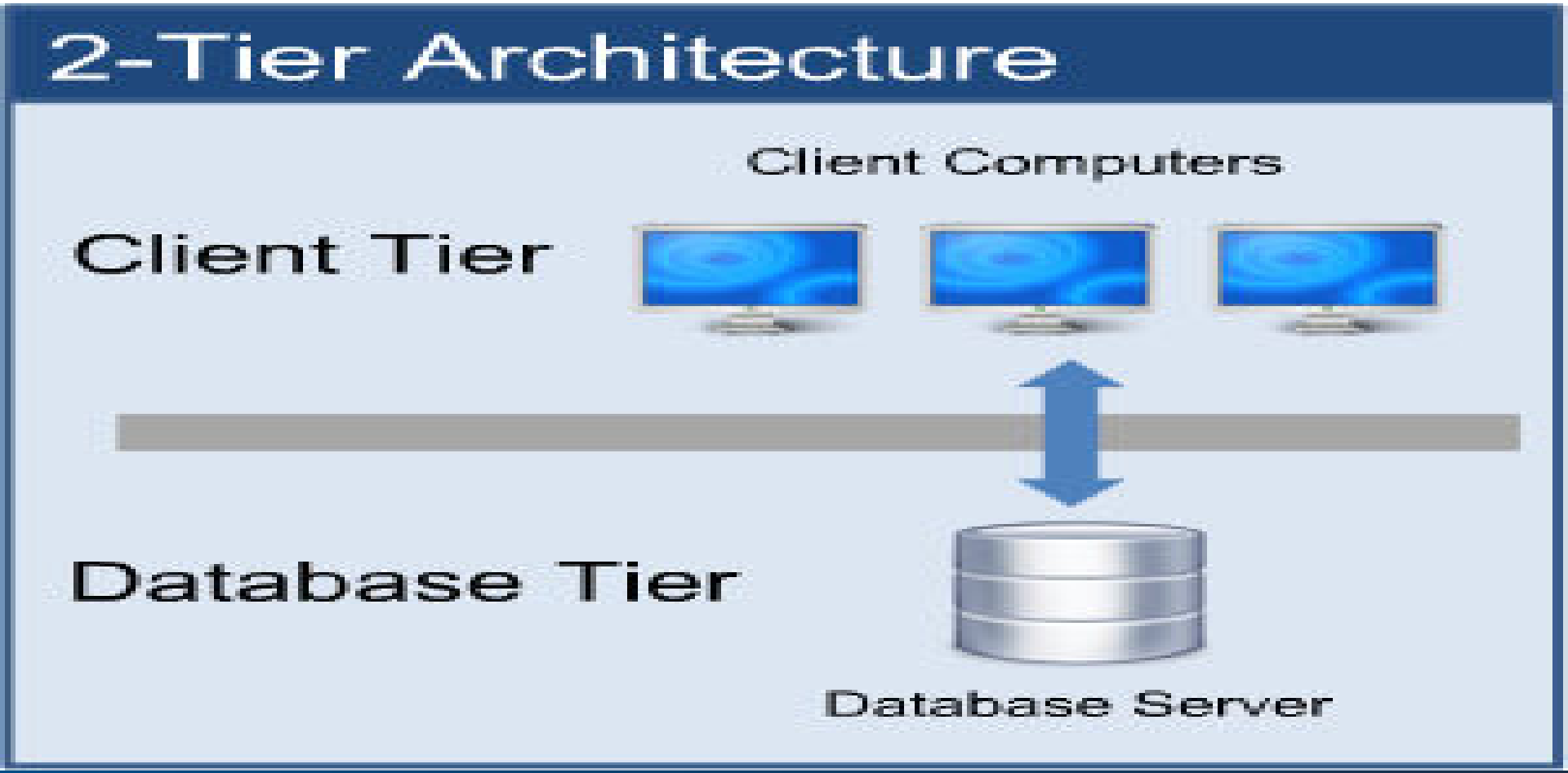
The UA Proposed Repository

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

The Layered Approach to Software Development

Two-Layered Architecture

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



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

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 !!!