19CAB12-OBJECT ORIENTED ANALYSIS AND DESIGN

Mrs.M.Menakapriya
ASP/MCA
Muthayammal Engineering College
Rasipuram

Objectives

•To provide a brief, hands-on overview of object-oriented concepts and its life cycle for software development.

•To learn for modeling the software and to design them using UML diagrams

•To understand the problem domain and to identify the objects from the problem specification.

•To understand, how to apply design axioms and corollaries for the classes and object relational systems.

•To gain knowledge about open source tools for Computer Aided Software Engineering

Outcomes

•Able to understand the object oriented concepts and to apply object oriented life cycle model for a project.

•Able to design static and dynamic models using UML diagrams.

•Able to perform object oriented analysis to identify the objects from the problem specification.

•Able to identify and refine the attributes and methods for designing the object oriented system.

•Able learn the open source CASE tools and to apply them in various domains.

UNIT-I INTRODUCTION

An overview – Object basics – Object state and properties – Behavior – Methods – Messages – Information hiding – Class hierarchy – Relationships – Associations – Aggregations- Identity – Dynamic binding – Persistence – Meta classes – Object oriented system development life cycle.

UNIT-II METHODOLOGY AND UML

Introduction – Survey – Rumbaugh, Booch, Jacobson methods – Unified modeling language – Static and Dynamic models – Rational Rose Suite - UML diagrams – Static diagram : Class diagram – Use case diagrams – Behavior Diagram : Interaction diagram – State chart diagram – Activity diagram - Implementation diagram: Component diagram – Deployment diagram – example - Design of online railway reservation system using UML diagrams - Dynamic modeling – Model organization – Extensibility.

UNIT-III OBJECT ORIENTED ANALYSIS

Identifying Use case – Business object analysis – Use case driven object oriented analysis – Use case model – Documentation – Classification – Identifying object, relationships, attributes, methods – Super-sub class – A part of relationships Identifying attributes and methods – Object responsibility – construction of class diagram for generalization, aggregation – example – vehicle class.

UNIT-IV OBJECT ORIENTED DESIGN

Design process and benchmarking – Axioms – Corollaries – Designing classes – Class visibility – Refining attributes – Methods and protocols – Object storage and object interoperability – Databases – Object relational systems – Designing interface objects – Macro and Micro level processes – The purpose of a view layer interface-OOUI - MVC Architectural Pattern and Design – Designing the system.

UNIT-V CASE TOOLS

Railway domain: Platform assignment system for the trains in a railway station – Academic domain: Students marks analyzing system – ATM system – Stock maintenance – Quiz System – E-mail Client system – Cryptanalysis – Health Care Systems, Use Open Source CASE Tools: StarUML / UML Graph for the above case studies.

REFERENCE BOOKS

•Ali Bahrami, “Object Oriented System Development”, McGraw Hill International Edition, 2008.

•Brahma Dathan, Sarnath Ramnath, “Object-Oriented Analysis, Design and Implementation”, Universities Press, 2010.

•Bernd Bruegge, Allen H. Dutoit, Object Oriented Software Engineering using UML, Patterns and Java, Pearson 2004.

•Craig Larman, Applying UML and Patterns – An Introduction to Object-Oriented Analysis and Design and Iterative Development”, 3rd Edition, Pearson Education, 2005.

•Grady Booch, James Rumbaugh, Ivar Jacobson, “The Unified Modeling Language User Guide”, Addison Wesley Long man, 1999.

•Martin Fowler, “UML Distilled A Brief Guide to Standard Object Modeling Language”, 3rd Edition, Addison Wesley, 2003.

•Russ Miles, Kim Hamilton, “Learning UML 2.0”, O‘Reilly, 2008.

INTRODUCTION

- **An Overview of Object Oriented Systems Development**
- **Object Basics**
- **Object Oriented Systems Development Life Cycle**

Object-oriented analysis and design – An Overview

- Object-oriented analysis and design (OOAD) is a popular technical approach for
  - analyzing,
  - designing an application, system, or business
  - by applying the object oriented paradigm and
  - visual modeling throughout the development life cycles for better communication and product quality.
- Object-oriented programming (OOP) is a method
  - based on the concept of “objects",
  - which are data structures that contain data,
  - in the form of fields,
  - often known as attributes;
  - and code, in the form of procedures,
  - often known as methods.

- **What is OOAD?-** Object-oriented analysis and design (OOAD) is a software engineering approach that models a system as a group of interacting objects .

- **Analysis** — understanding, finding and describing concepts in the problem domain.

- **Design** — understanding and defining software solution/objects that represent the analysis concepts and will eventually be implemented in code.

- **OOAD -** A software development approach that emphasizes a logical solution based on objects.

- Software development is dynamic and always undergoing major change.

- System development refers to all activities that go into producing information system solution.

- **System development activities consist of**
  - system analysis,
  - modelling,
  - design,
  - implementation,
  - testing and maintenance.

- A software development methodology → series of processes → can lead to the development of an application.

- Practices, procedures, and rules used to develop software, totally based on system requirements

ORTHOGONAL VIEWS OF THE SOFTWARE

- Two Approaches,
  - **Traditional Approach**
  - **Objected-Oriented Approach**

- **TRADITIONAL APPROACH**

  - Collection of **programs or functions.**
  - A system that is designed for **performing certain actions**.
  - **Algorithms + Data Structures = Programs**.
  - Software Development Models (**Waterfall, Spiral, Incremental, etc..)**

❖ OBJECT ORIENTED APPROACH

- OO development offers a different model from the traditional software → **based on functions and procedures.**
- software is a collection of **discrete object that encapsulate their data as well as the functionality.**
- Each object has **attributes (properties) and method (procedures).**
- software by building self contained modules or objects that can be easily **REPLACED, MODIFIED AND REUSED.**
- Objects **grouped in to classes and object are responsible for itself.**

Difference between Traditional and Object Oriented Approach

TRADITIONAL APPROACH	OBJECT ORIENTED SYSTEM DEVELOPMENT
Collection of procedures(functions)	Combination of data and functionality
Focuses on function and procedures, different styles and methodologies for each step of process	Focuses on object, classes, modules that can be easily replaced, modified and reused.
Moving from one phase to another phase is complex.	Moving from one phase to another phase is easier.
Increases duration of project	decreases duration of project
Increases complexity	Reduces complexity and redundancy

BENEFITS OF OBJECT ORIENTATION

- **Faster development,**
- **Reusability,**
- **Increased quality**
- modeling the **real world and provides us with the stronger equivalence of the real world‘s entities (objects)**.
- Raising the level of abstraction to the point where **application can be implemented in the same terms as they are described.**

WHY OBJECT ORIENTATION

- OO Methods enables to develop **set of objects that work** together → software → similar to traditional techniques.

- It adapts to

  - **Changing requirements**
  - **Easier to maintain**
  - **More robust**
  - **Promote greater design**
  - **Code reuse**

- Others

  - **Higher level of abstraction**
  - **Seamless transition among different phases of software development.**
  - **Encouragement of good programming technique.**
  - **Promotion of reusability.**

OVERVIEW OF UNIFIED APPROACH

- The **unified approach (UA)** is a methodology for **software development.**

- **Booch, Rumbaugh, Jacobson methodologies** gives the best practices, processes and guidelines for OO oriented software development.

- Combines with the **object management groups in unified modelling language.**

- UA utilizes the **unified modeling language (UML)** which is a set of **notations and conventions** used to describe and **model an application.**

Layered Architecture

- UA uses **layered architecture to develop applications**.
- **Creates object** that **represent elements to the user** through **interface or physically stored in database.**
- The layered approach consists of **user interface, business, access layers.**
- This approach **reduces the interdependence of the user interface, database access and business control.**
- **More robust and flexible system.**

OBJECT BASICS

Goals:

The developer should

- Define Objects and classes
- Describe objects, methods, attributes and how objects respond to messages,
- Define Polymorphism, Inheritance, data abstraction, encapsulation, and protocol,
- Describe objects relationships,
- Describe object persistence,

WHAT IS AN OBJECT?

- The term object was first formally utilized in the **Simula language** to **simulate some aspect of reality.**

- Attributes or properties describe object‘s state (data) and methods (properties or functions) define its behavior.

- An object is an entity.
  - It knows things (has attributes)
  - It does things (provides services or has methods
  - Examples in next Slide …….

OBJECT'S ATTRIBUTES

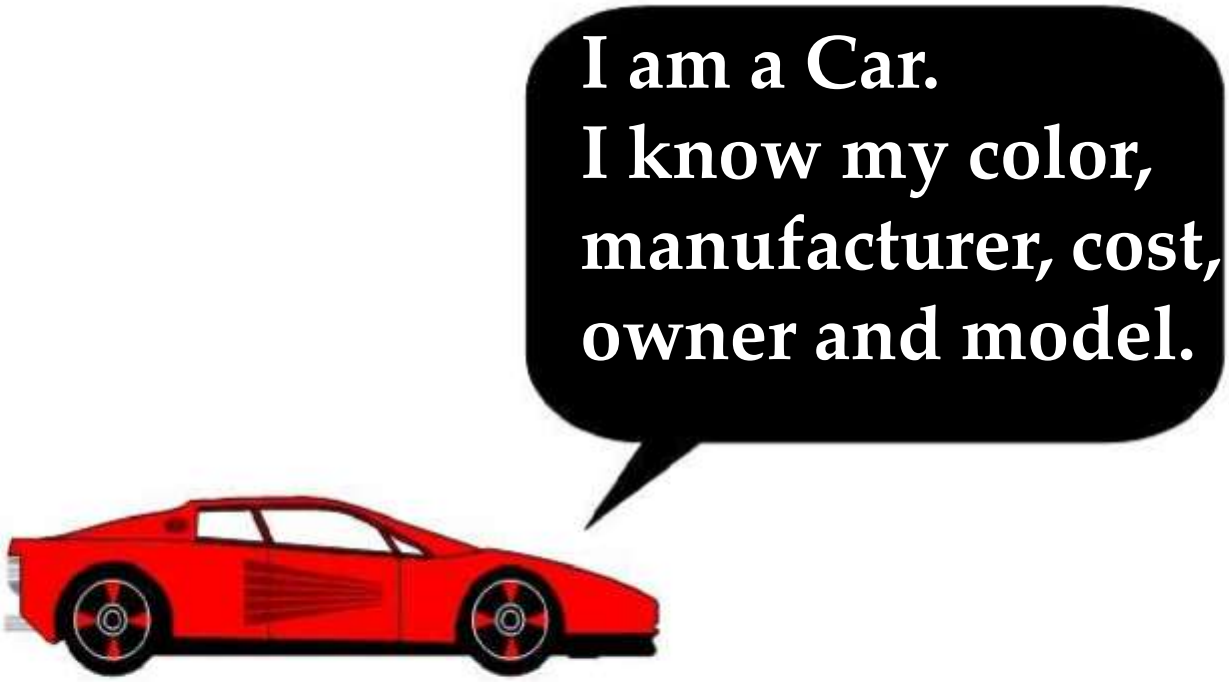
- **Attributes represented by data type.**
- **They describe objects states.**
- **In the Car example the car's attributes are:**
- **color, manufacturer, cost, owner, model, etc.**

OBJECT'S METHODS

- **Methods define objects behaviour and specify the way in which an Object's data are manipulated.**
- **In the Car example the car's methods are:**
- **drive it, lock it, tow it, carry passenger in it.**

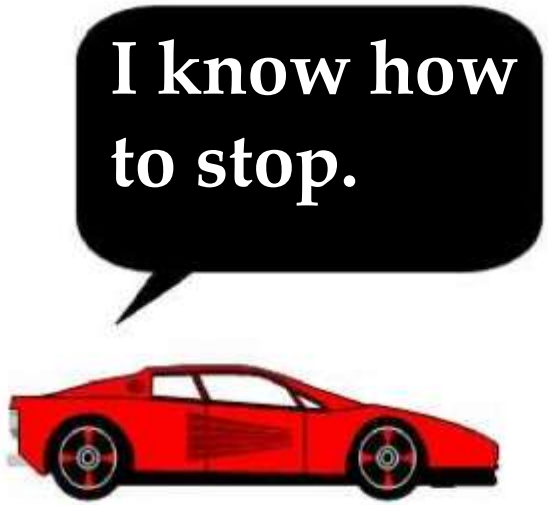
IT KNOWS THINGS (ATTRIBUTES)

ATTRIBUTES



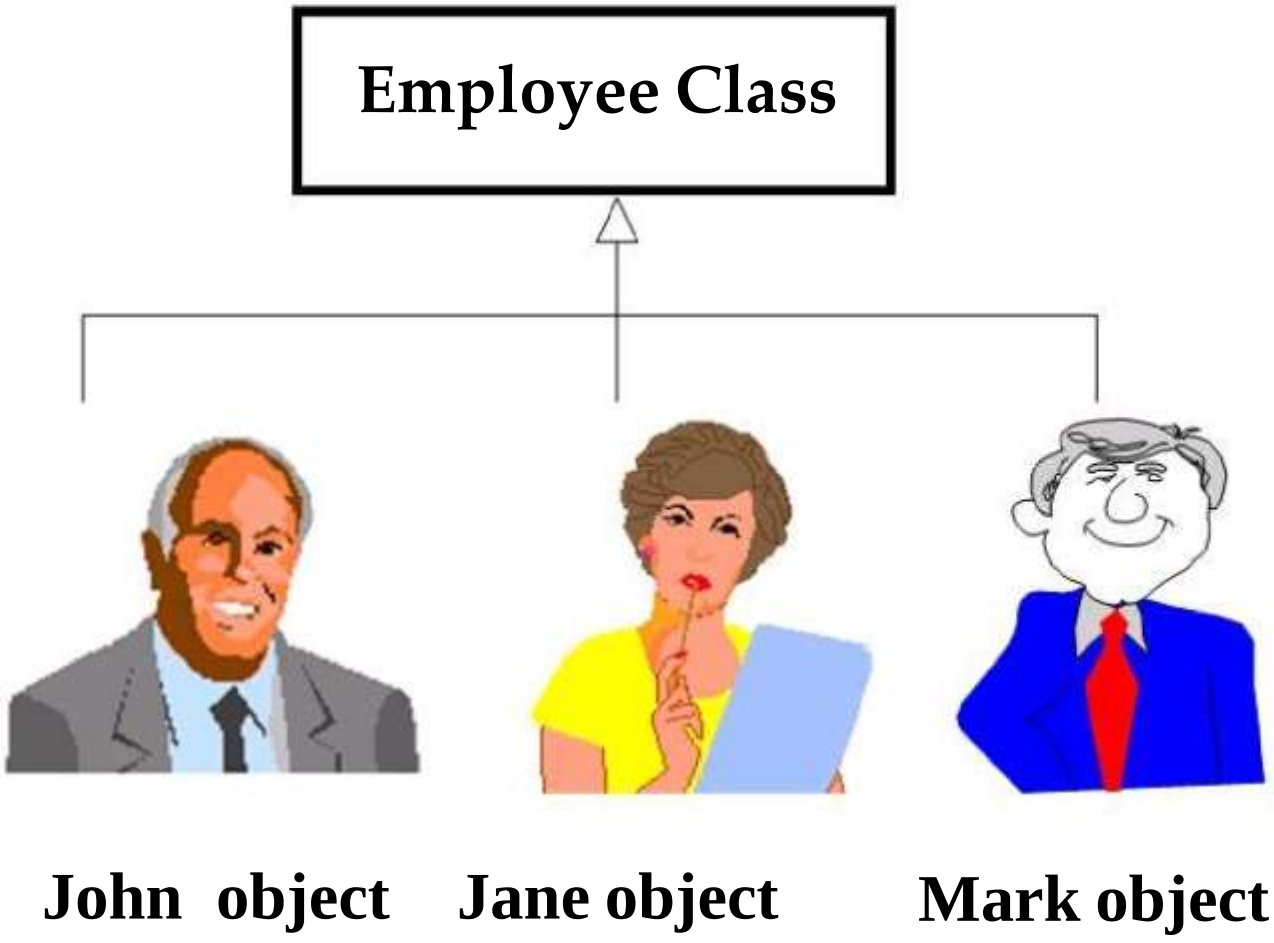
IT DOES THINGS (METHODS)

METHODS



OBJECTS ARE GROUPED IN CLASSES

- The role of a class is to **define the attributes and methods** (the state and behaviour) **of its instances.**
- Used to **distinguish one type of object from the other.**
- Set of objects**, that share common methods, structure, behaviour.**
- **Single object** is simply an **instance of class.**
- The **class car**, for example, **defines the property color.** Each individual car (object) will have a **value for this property, such as "maroon," "yellow" or "white."**


Employee Class
John object
Jane object
Mark object

Class

- A class represents a collection of objects having same characteristic properties that exhibit common behavior.
- Creation of an object as a member of a class is called instantiation.
- Thus, object is an instance of a class.
- Example: **Circle** – a class

  x, to the center

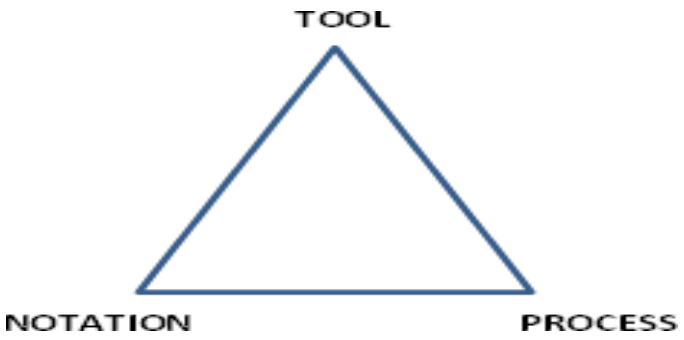
  a, to denote the radius of the circle
- Some of its operations can be defined as follows:

  findArea(), method to calculate area

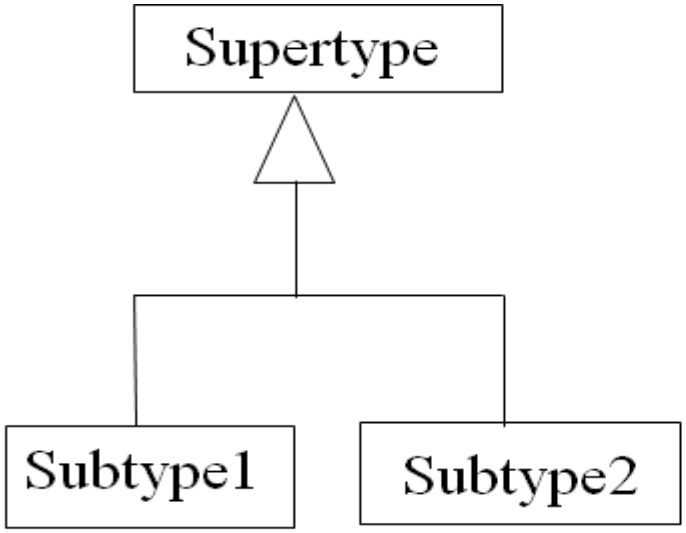
  findCircumference(), method to calculate circumference

Object oriented Methodologies

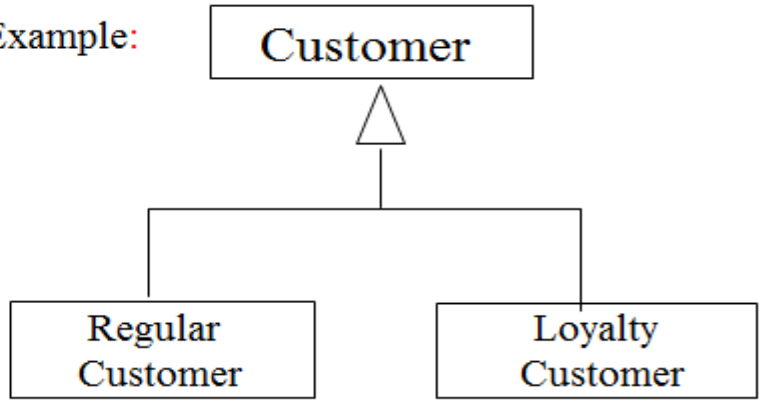
- Many methodologies have been developed for object oriented development.
- A methodology usually includes
  - **Notation** : Graphical representation of classes and their relationships with interactions.
  - **Process** : Suggested set of steps to carry out for transforming requirements into a working system.
  - **Tool :** Software for drawings and documentation



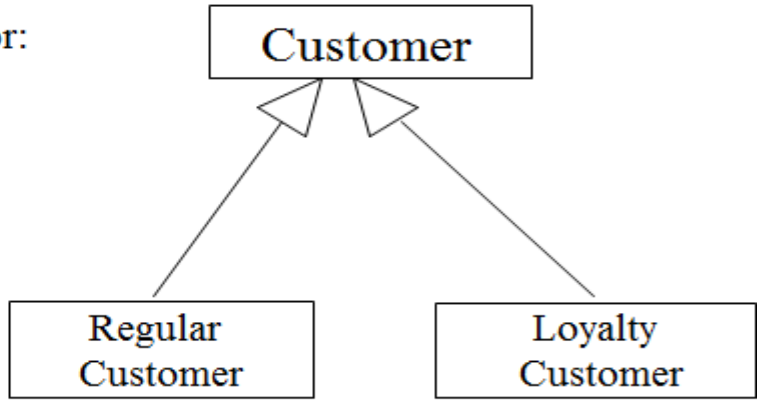
OO Relationships: **Generalization**



Example:



or:



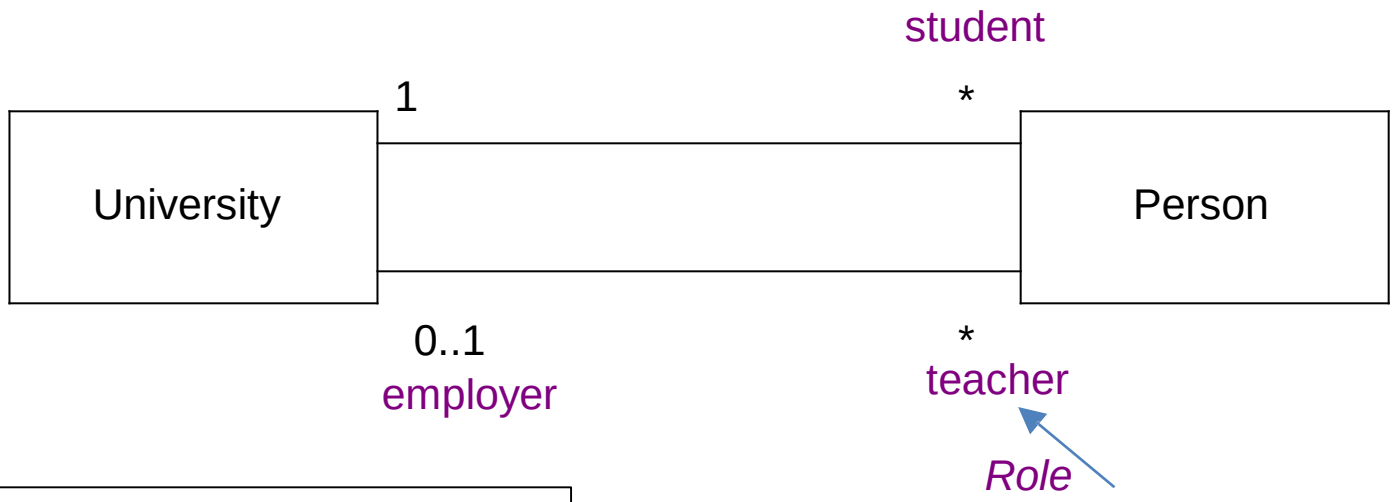
- Generalization expresses a parent/child relationship among related classes.

- Used for abstracting details in several layers

OO Relationships: **Association**

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

Association: Multiplicity and Roles



Multiplicity

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

Role

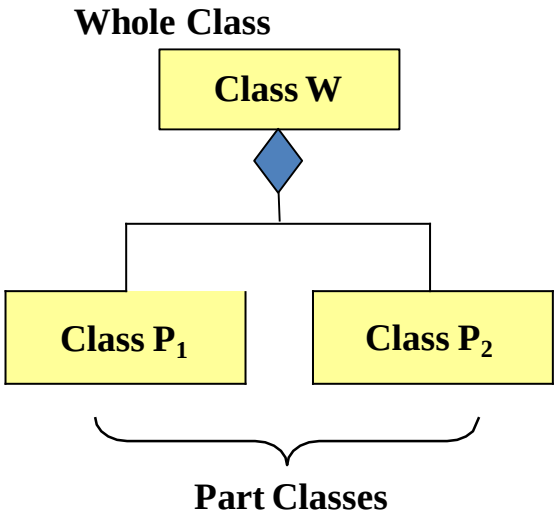
"A given university groups many people; some act as students, others as teachers. A given student belongs to a single university; a given teacher may or may not be working for the university at a particular time."

Association: Model to Implementation


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

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

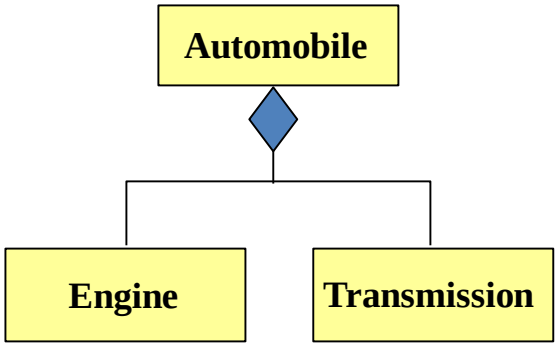
OO Relationships: **Composition**



Composition: **expresses a relationship among instances of related classes. It is a specific kind of Whole-Part relationship.**

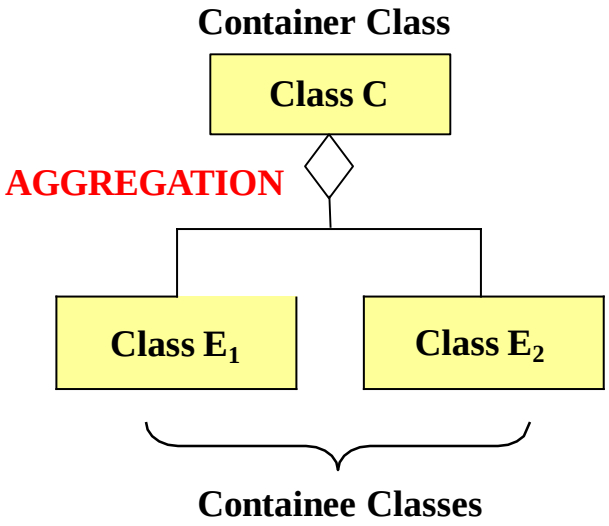
It expresses a relationship where an instance of the Whole-class has the responsibility to create and initialize instances of each Part-class.

Example



It may also be used to express a relationship where instances of the Part-classes have privileged access or visibility to certain attributes and/or behaviors defined by the Whole-class.

OO Relationships: **Aggregation**



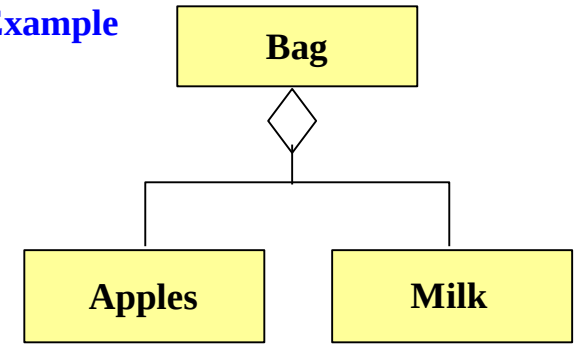
Aggregation: expresses a relationship among instances of related classes. It is a specific **kind of Container-Containee** relationship.

It expresses a relationship where an instance of the Container-class has the responsibility to **hold and maintain instances** of each Containee-class that have been created outside the Container-class.

Aggregation should be used to express a more informal relationship than composition expresses. That is, it is an appropriate relationship where the **Container and its Containees**

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

Example



Aggregation vs. Composition

•**Composition** is really a strong form of **aggregation**

- •components have only one owner
- •components cannot exist independent of their owner
- •components live or die with their owner

e.g. Each car has an engine that can not be shared with other cars.

•**Aggregations** may form "part of" the aggregate, but may not be essential to it. They may also exist independent of the aggregate.

e.g. Apples may exist independent of the bag.

Objects

Object	Identity	Behaviors	State
An employee	“Mr. John”	Join(), Retire()	Joined, Retired.
A book	“Book with title Object Oriented Analysis Design”	AddExemplar,	Rent, available, reserved
A sale	“Sale no 0015, 15/12/98”	SendInvoiced(), Cancel().	Invoiced, cancelled.

Object Class

Class is a description of a set of objects that share the same attributes, operations, methods, relationship and semantics.

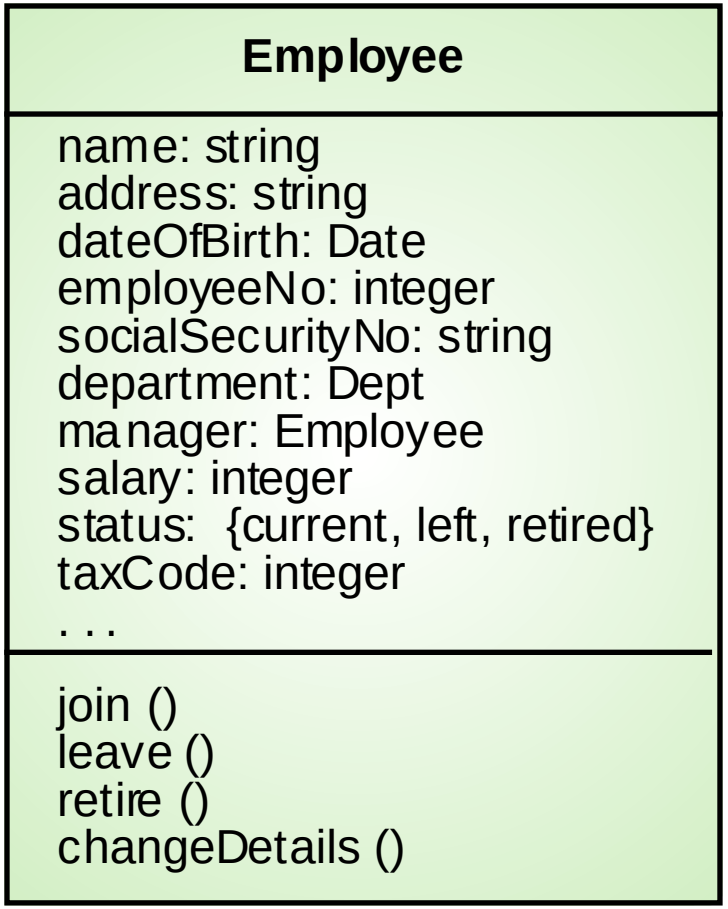
Object classes are templates for objects. They may be used to create objects.

An object represents a particular instance of a class.

Term of objects

Attribute: data items that define object.
Operation: function in a class that combine to form behavior of class.
Methods: the actual implementation of procedure (the body of code that is executed in response to a request from other objects in the system).

Employee object & class



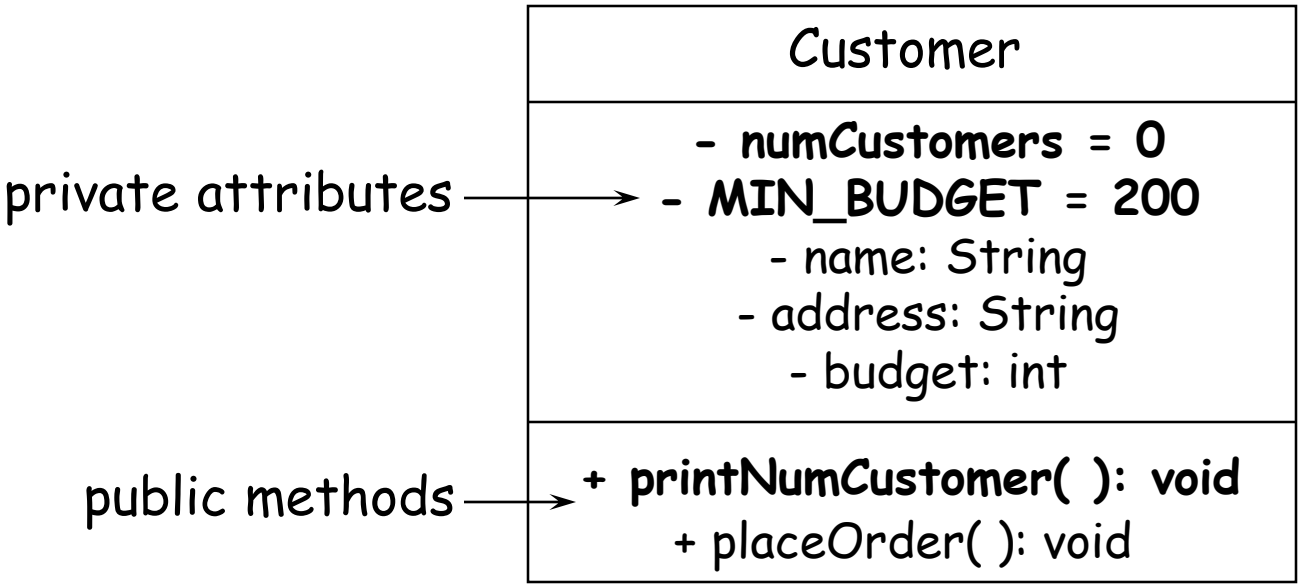
Encapsulation and Data Hiding

Packaging related data and operations together is called encapsulation.

Data hiding: hides the internal data from external by methods (interface).

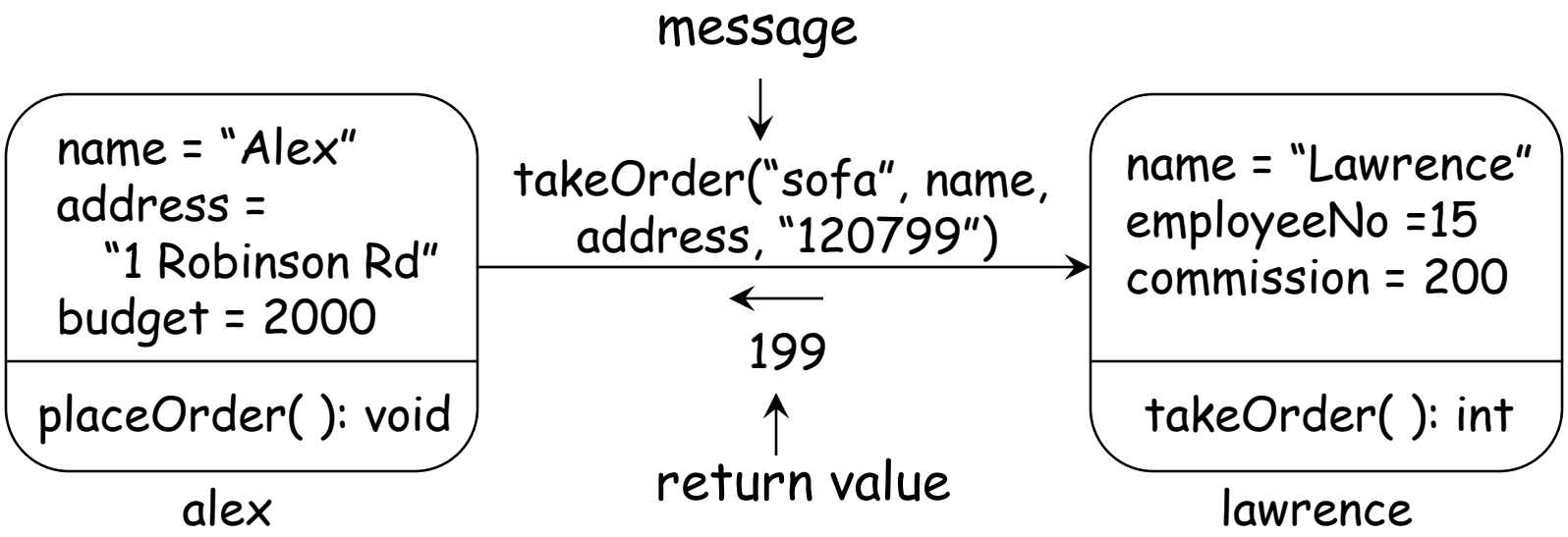
Encapsulation

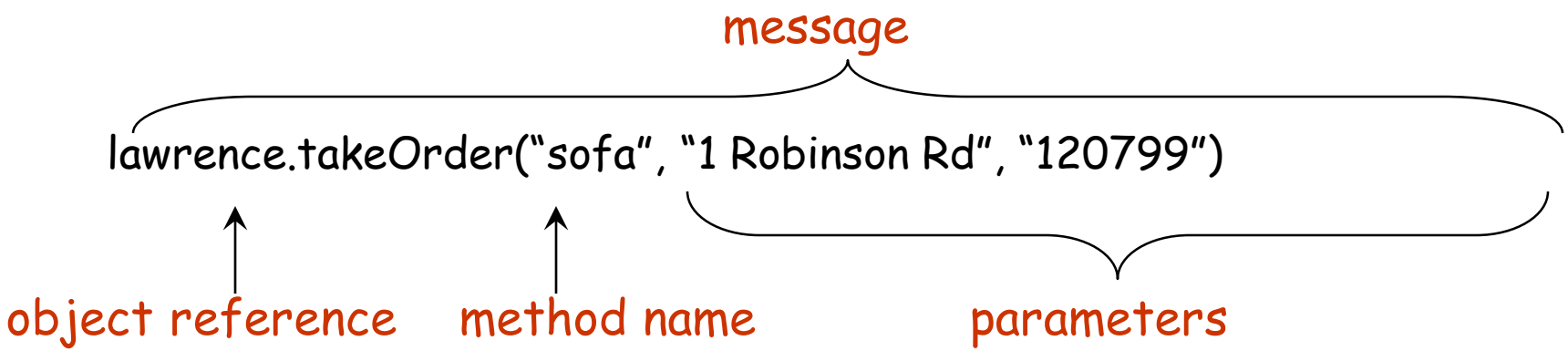
- *private* attributes and methods are encapsulated within the class, they cannot be seen by clients of the class
- *public* methods define the interface that the class provides to its clients



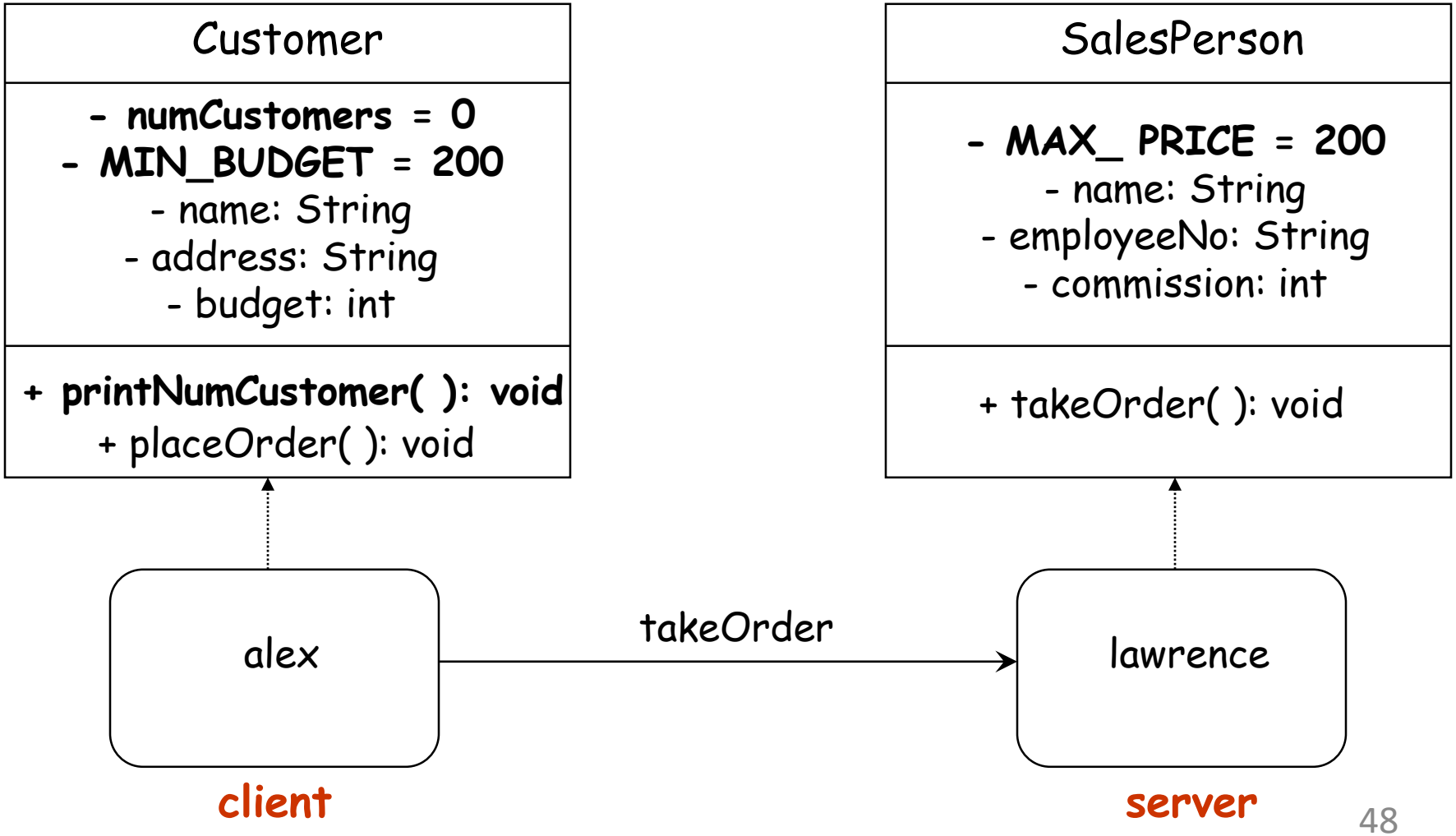
Customer class

Message Passing





Message Passing

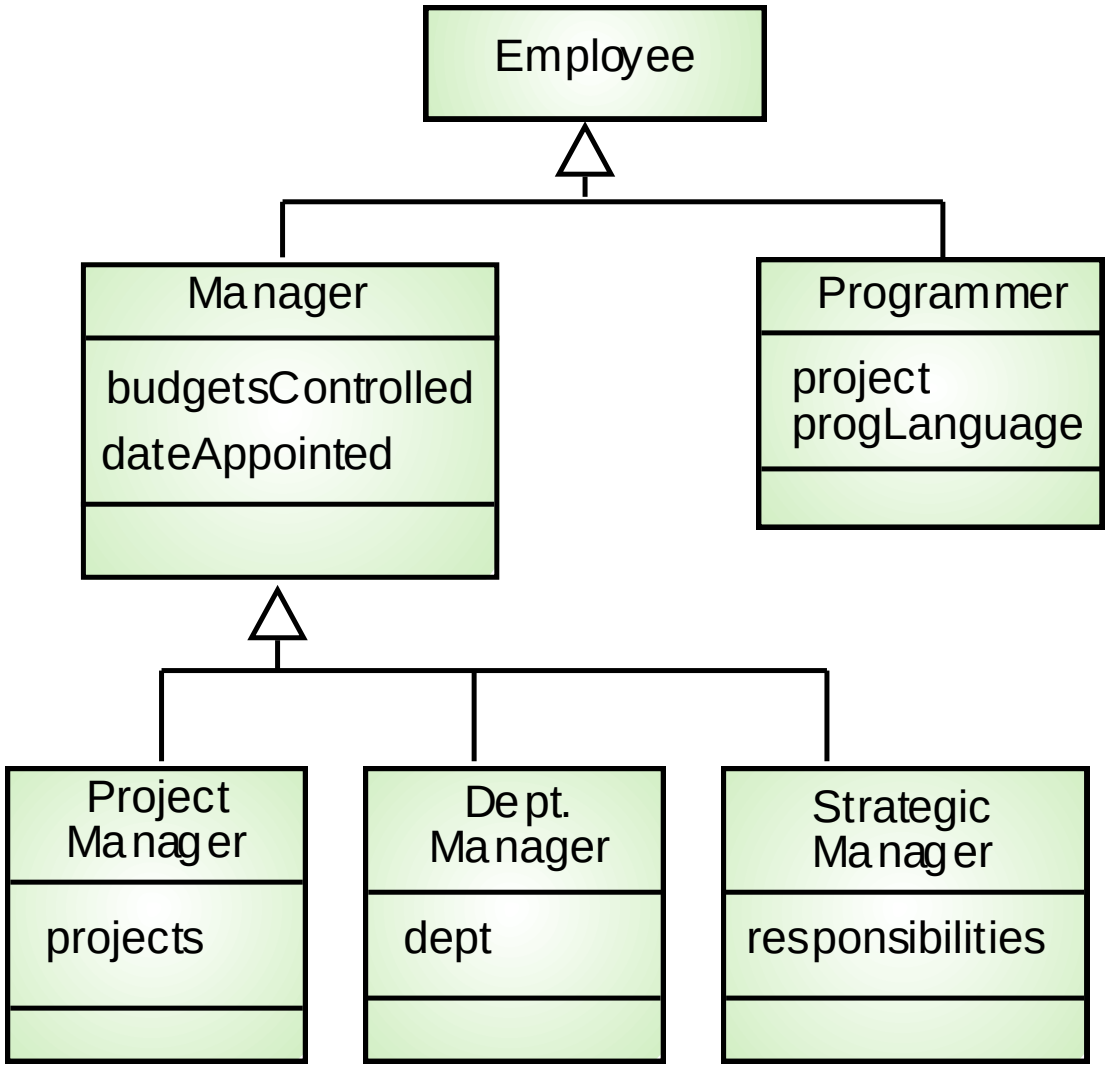


Inheritance

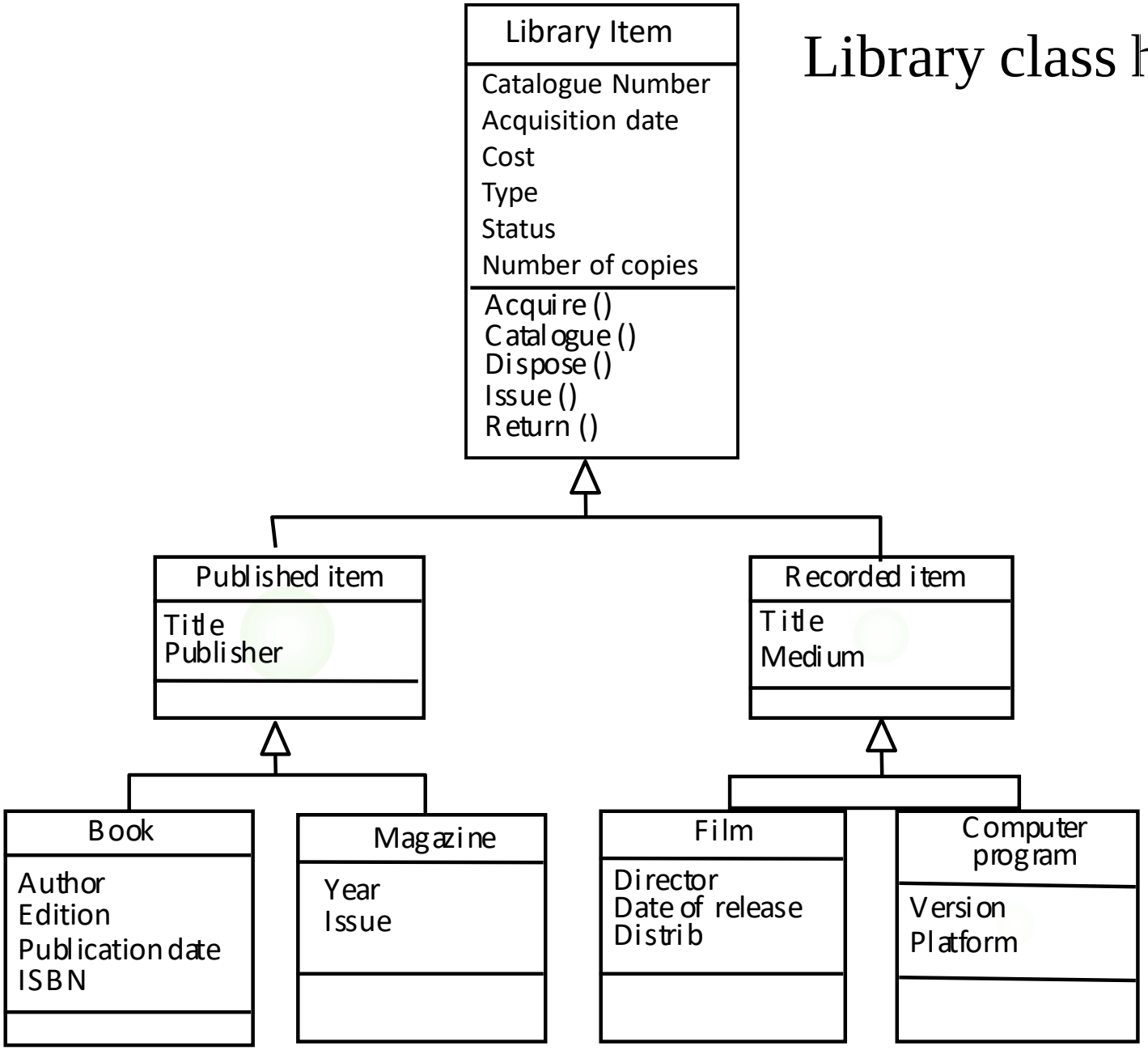
- Object classes may inherit attributes and services from other object classes.

- Inheritance represents the generalization of a class.

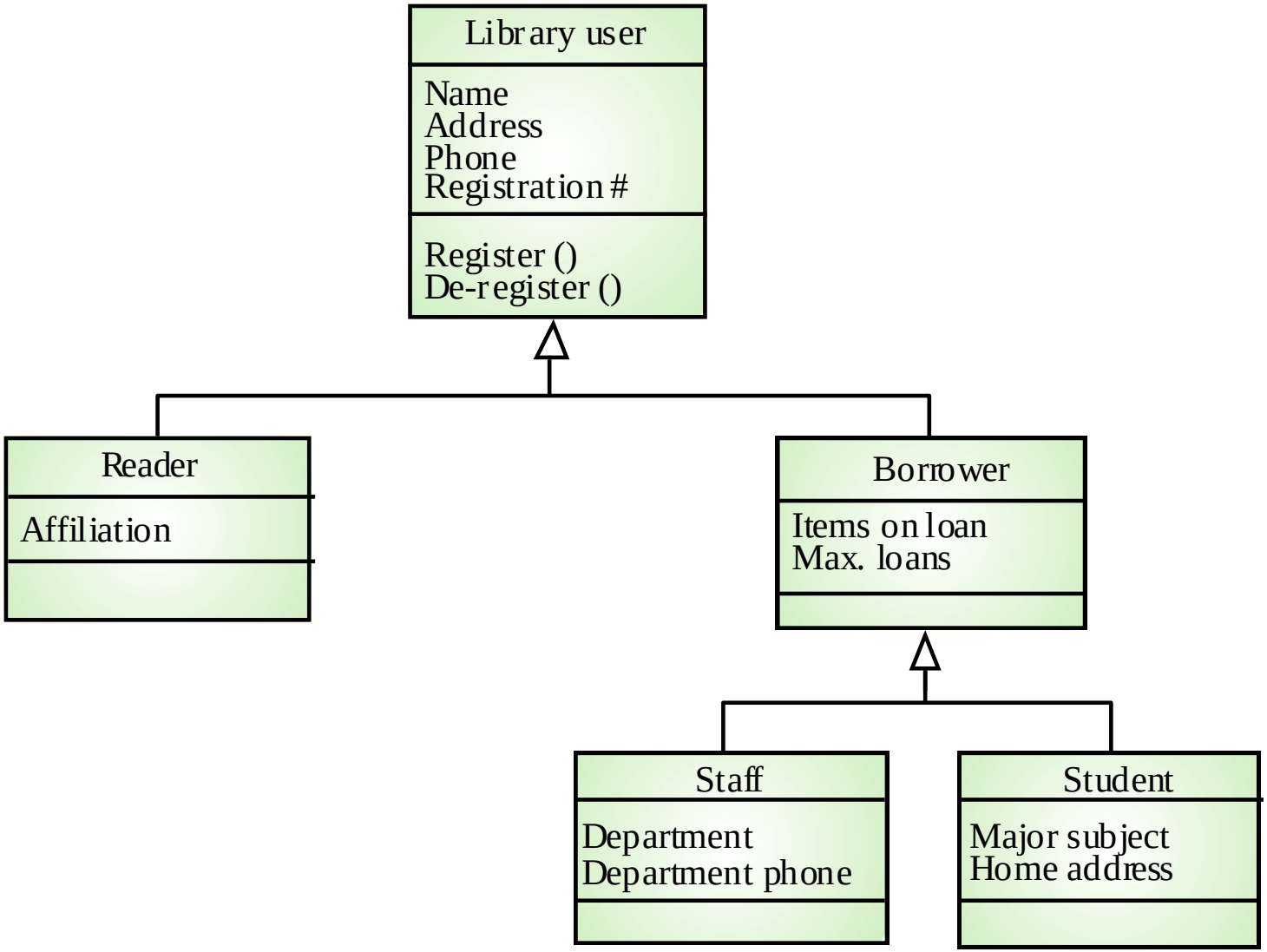
A generalisation hierarchy



Library class hierarchy



User class hierarchy



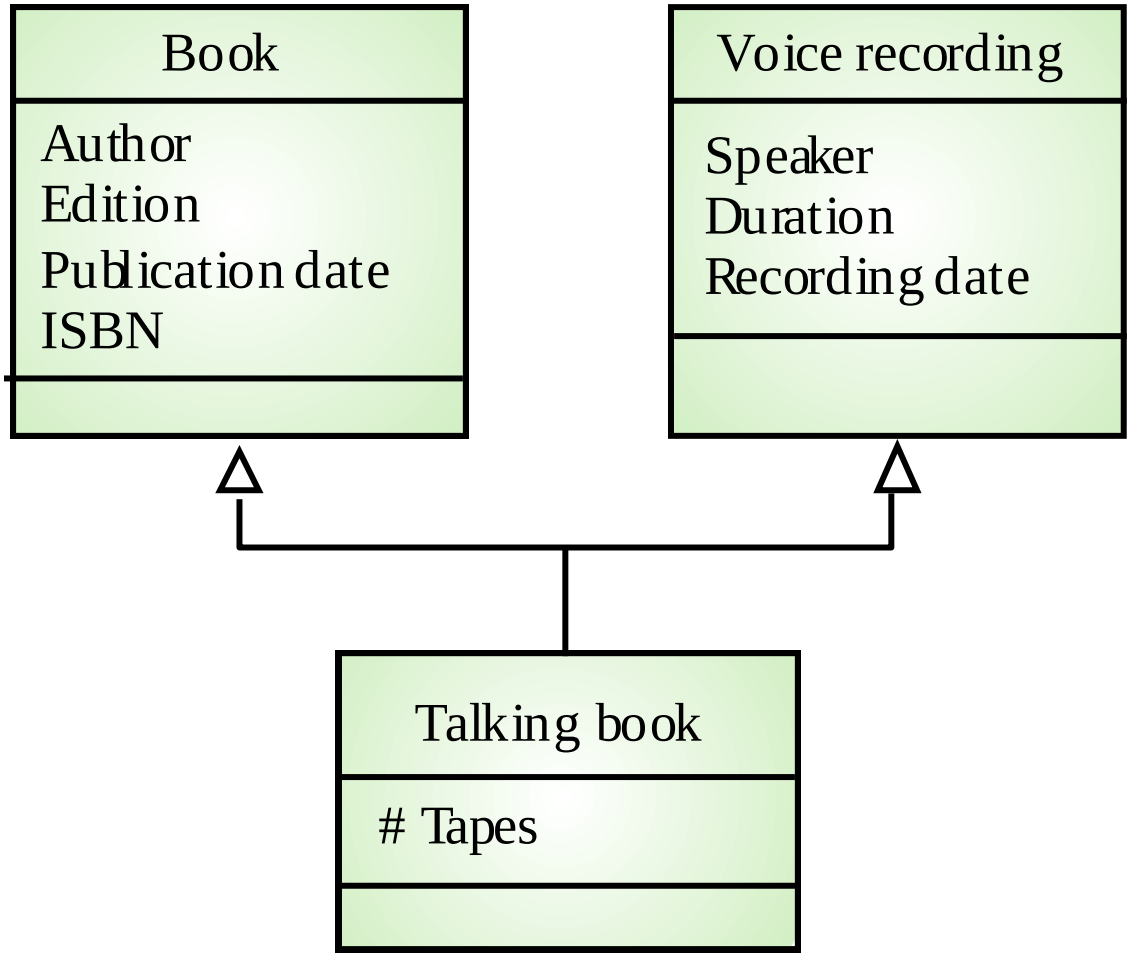
Multiple inheritance

•Rather than inheriting the attributes and services from a single parent class, a system which supports multiple inheritance allows object classes to inherit from several super-classes

•Can lead to semantic conflicts where attributes/services with the same name in different super-classes have different semantics

•Makes class hierarchy reorganisation more complex

Multiple Inheritance



Advantages of Inheritance

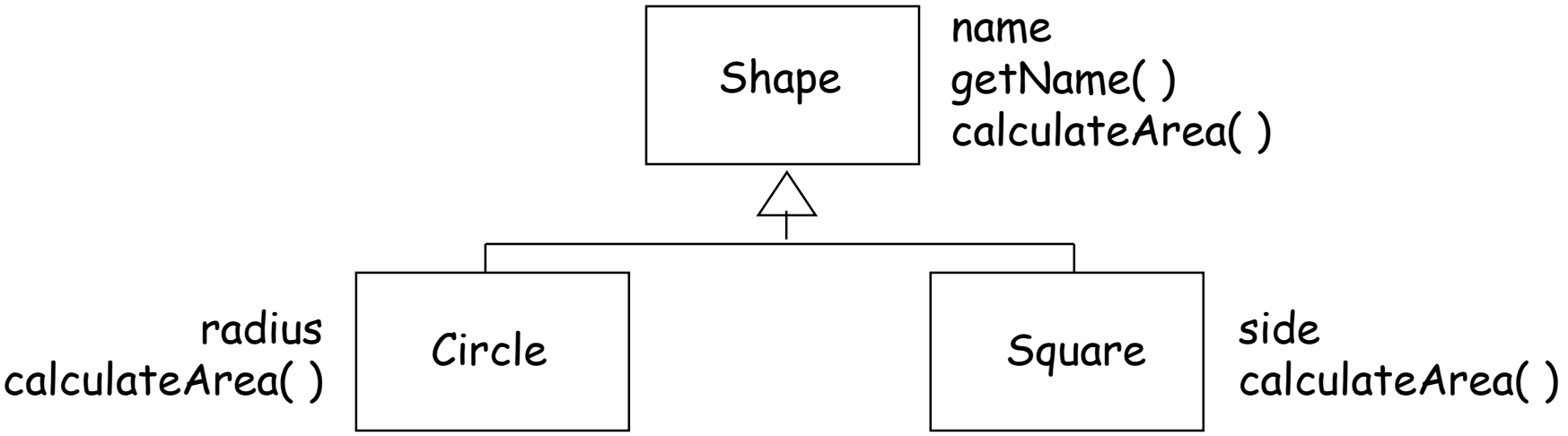
- It is an abstraction mechanism which may be used to classify entities

- It is a reuse mechanism at both the design and the programming level

- The inheritance graph is a source of organisational knowledge about domains and systems

Polymorphism

- The ability of different objects to perform the appropriate method in response to the same message is known as polymorphism.
- The selection of the appropriate method depends on the class used to create the object



Identity (Object and identity):-

A special feature of object-oriented system is that every object has its own unique and immutable identity. An object's identity comes into being when the object is created and continues to represent that object from then on. This identity never is confused with another object, even if the original object has been deleted.

In an object system, object identity often is implemented through some kind of **object identifier** (OID) or unique identifier (UID).

Dynamic Binding (Static and Dynamic Binding):-

The process of determining (dynamically) at run time which functions to invoke is termed **dynamic binding.**

Making this determination earlier, at compile time, is called **static binding.**

Static binding optimizes the calls; dynamic binding occurs when polymorphic calls are issued.

Dynamic binding allows some methods invocation decisions to be deferred until the information is known.

Persistence (Object Persistence):-

Objects have a lifetime. They are explicitly created and can exist for a period of time that, traditionally, has been the duration of the process in which they were created. A filer or a database can provide support for objects having a longer lifeline longer than the duration of the process for which they were created.From a language perspective, this characteristic is called **object persistence.**

An object can persist beyond application session boundaries, during which the object is stored in a file or a database, in some file or database form. The object can be retrieved in another application session and will have the same state and relationship to other objects as at the time it was saved.The lifetime of an object can be explicitly terminated.

Meta-Classes

Is a class an object? A class is an object, a class belongs to a class called a meta-class, or a class of classes.

For example, classes must be implemented in some way; perhaps with dictionaries for methods, instances, and parents and methods to perform all the work of being a class. This can be declared in a class named **meta-class.**

The meta-class also can provide services to application programs, such as returning a set of all methods, instances, or parents for review (or even modification)

Object – Oriented Systems Development Life Cycle:-

Introduction:

The essence of the software Development Process that consists of analysis, design, implementation, testing, and refinement is to transform users' needs into a software solution that satisfy those needs.

The Software Development Process:

System development can be viewed as a process. Within the process, it is possible to replace one sub process has the same interface as the old one to allow it to fit into the process as a whole.

For example, Object oriented approach

The process can be divided into small, interacting phases- sub processes. Each subprocess must have the following :

- A description in terms of how it works.
- Specification of the input required for the process.
- Specification of the output to be produced.

.

Software process – transforming needs to software product

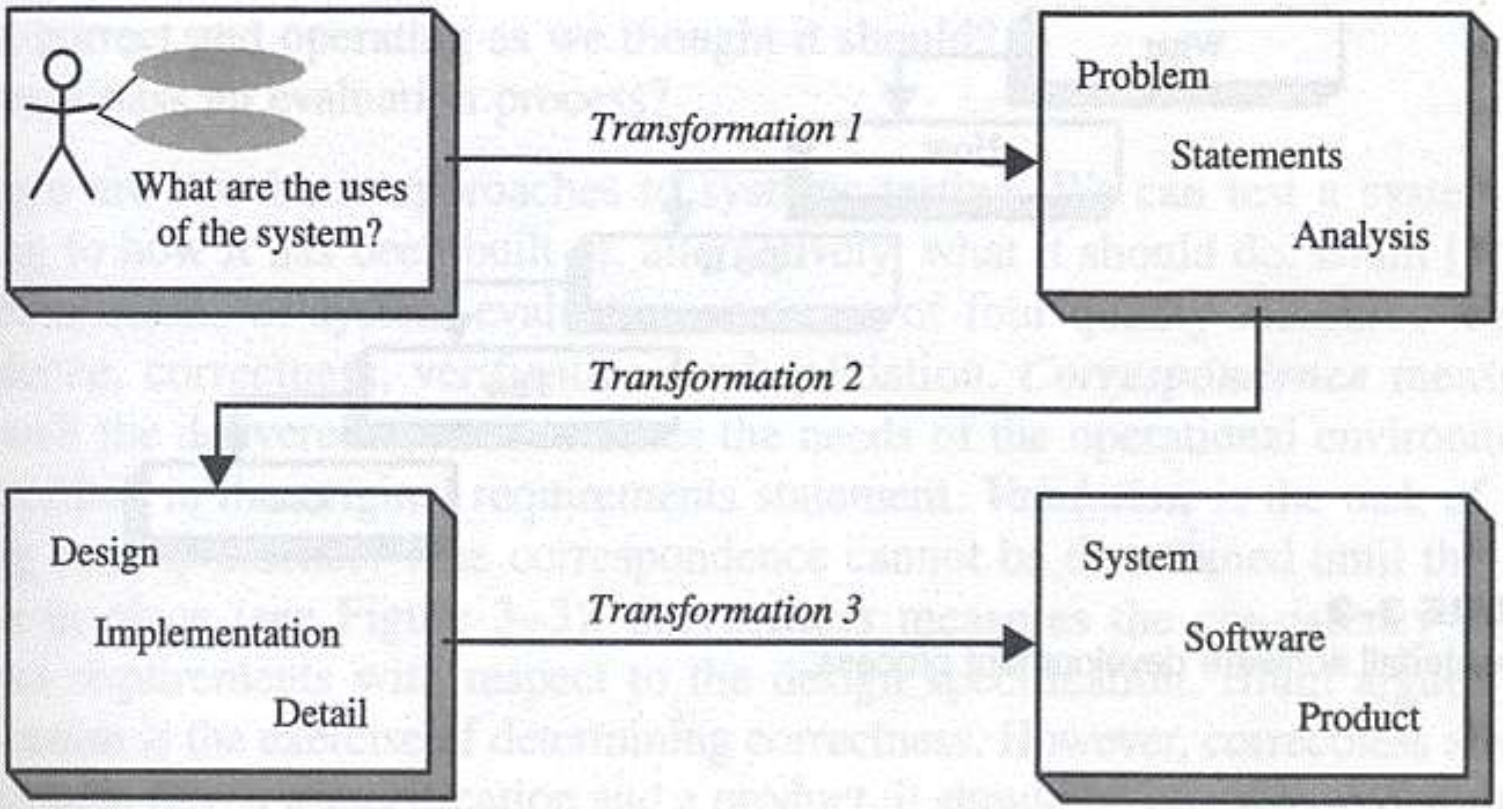


FIGURE 3–1
Software process reflecting transformation from needs to a software product that satisfies those needs.

Waterfall Model – from ‘what’ to ‘use’

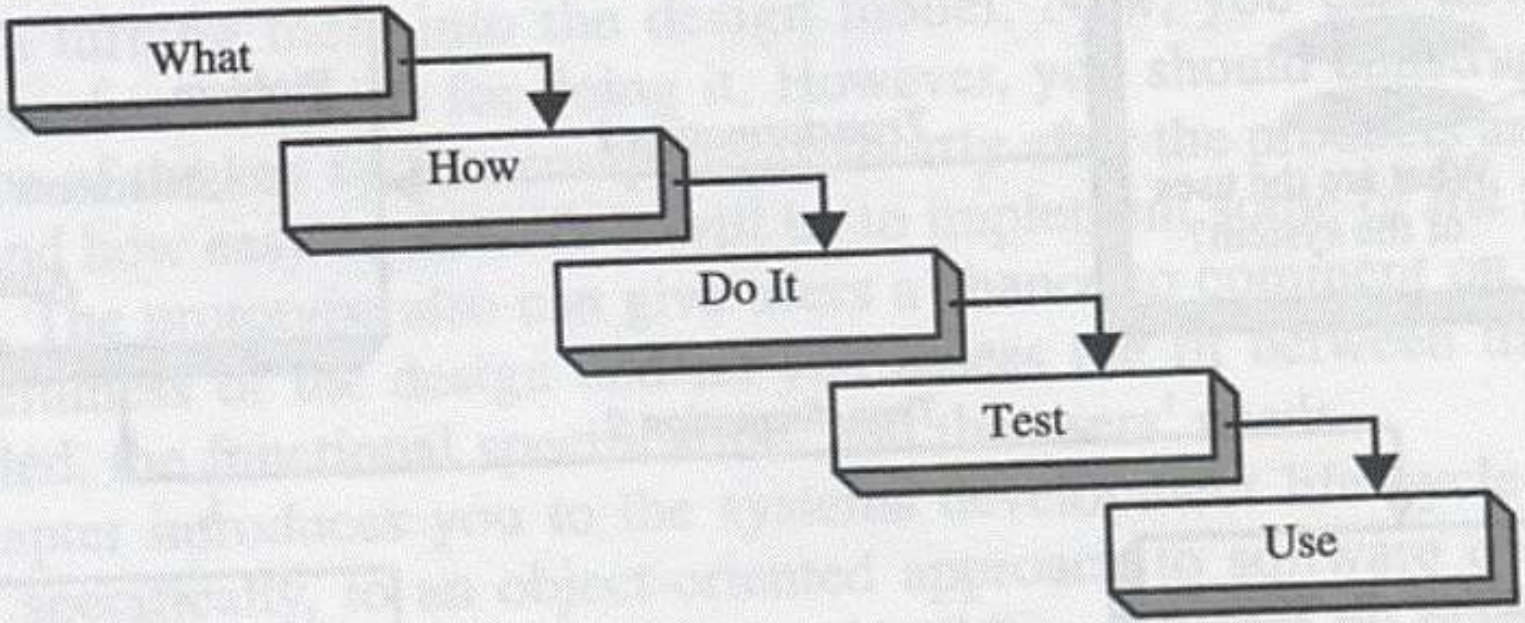
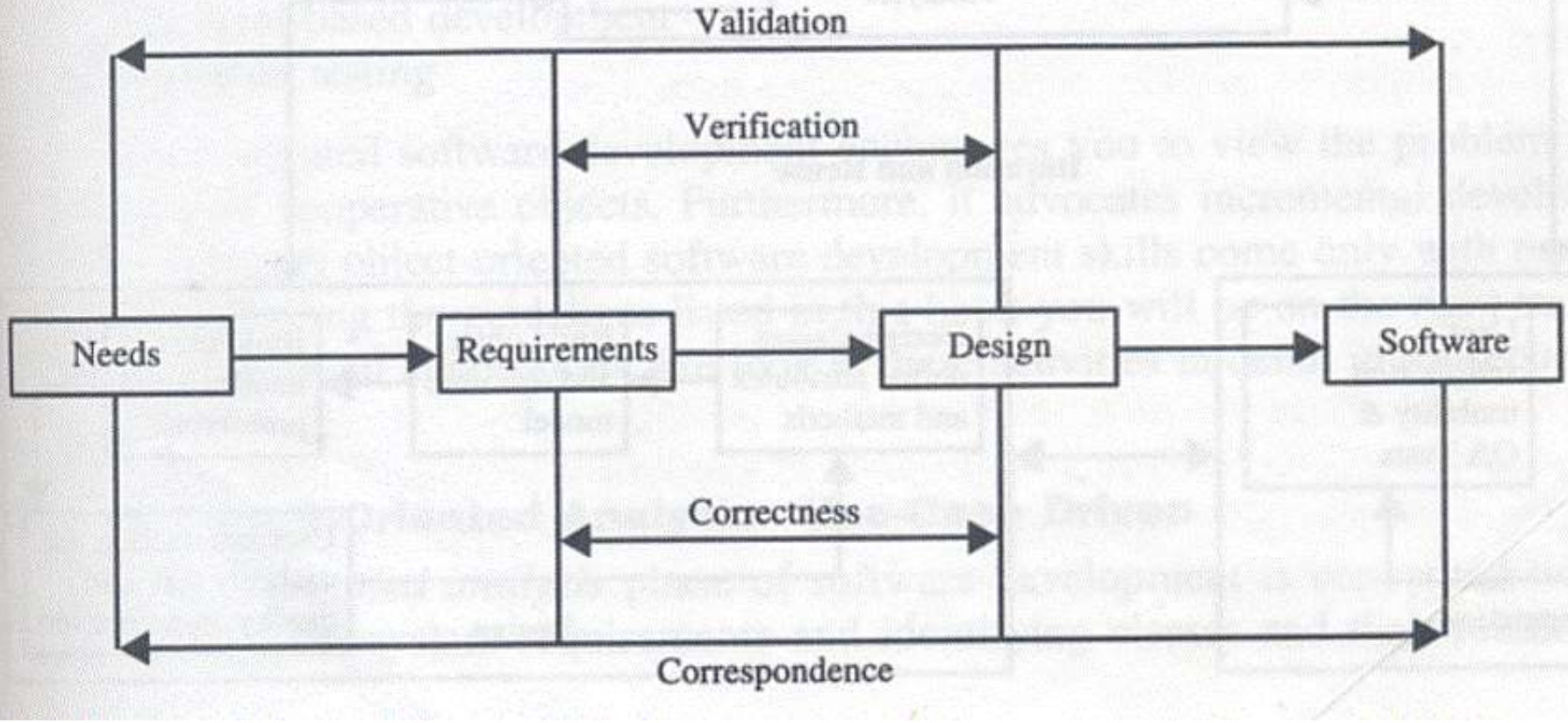


FIGURE 3–2
The waterfall software development process.

4 Quality Measures

FIGURE 3–3
Four quality measures (correspondence, correctness, validation, and verification) for software evaluation.



Verification vs Validation

Verification

am I building the product right ?

Begin after specification accepted

Validation

am I building the right product ?

Subjective - is specification appropriate ? Uncover true users' needs , therefore establish proper design ?

Begins as soon as project starts

Verification & validation independent of each other

even if product follows spec, it may be a wrong product if specification is wrong

eg: report missing, initial design no longer reflect current needs

If specification informal, difficult to separate verification and validation

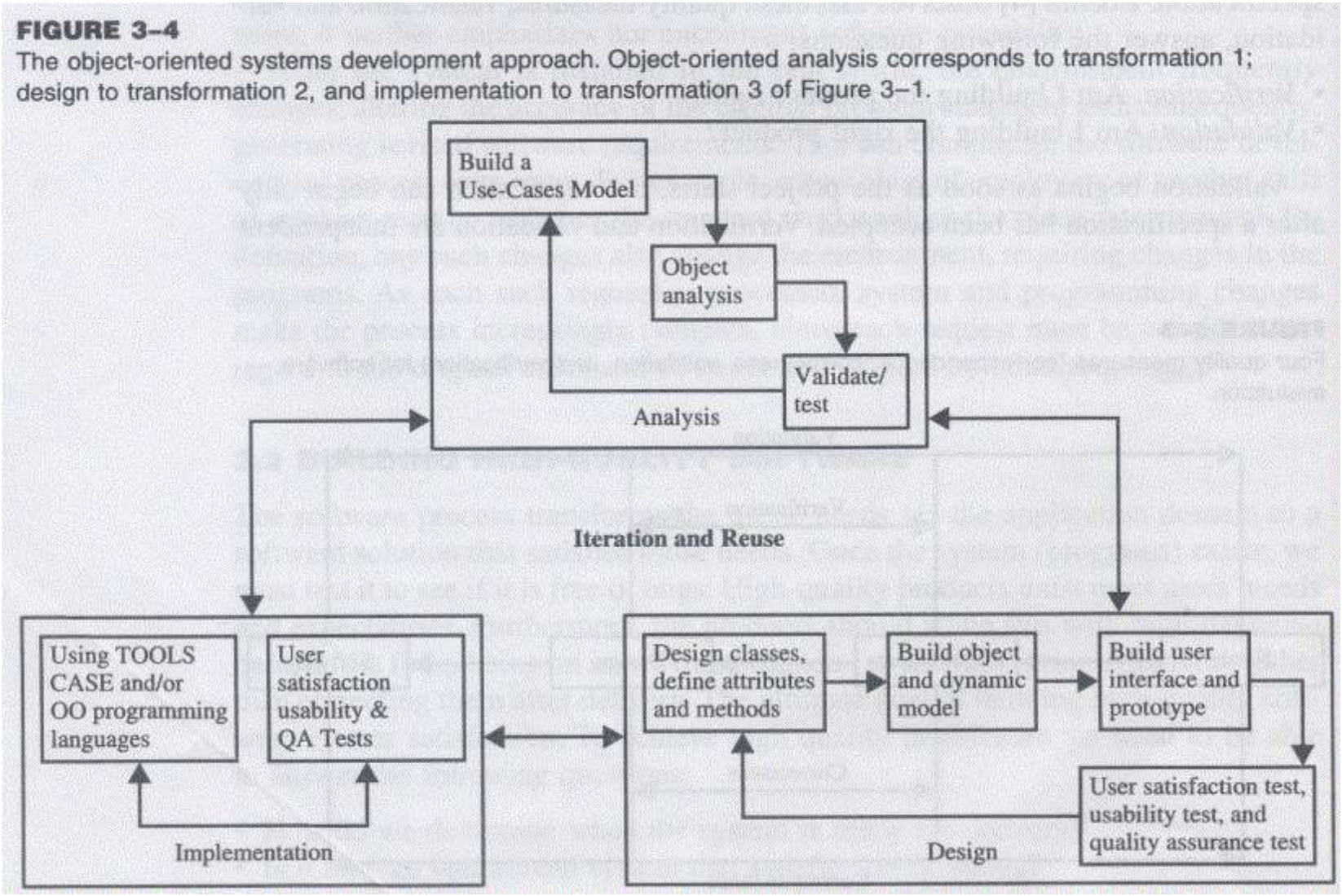
Object-oriented approach: A use-case driven approach

Object-oriented software development life cycle consists of
- Object-oriented analysis
- Object-oriented design
- Object-oriented implementation

Use-case model can be employed throughout most activities of software development
- designs traceable across requirements, analysis, design, implementation & testing can be produced
- all design decisions can be traced back directly to user requirements
- usage scenarios can be test scenarios

Object-oriented Systems Development Approach

FIGURE 3–4
The object-oriented systems development approach. Object-oriented analysis corresponds to transformation 1; design to transformation 2, and implementation to transformation 3 of Figure 3–1.



Using Jacobson et al. life cycle model – traceable design across development

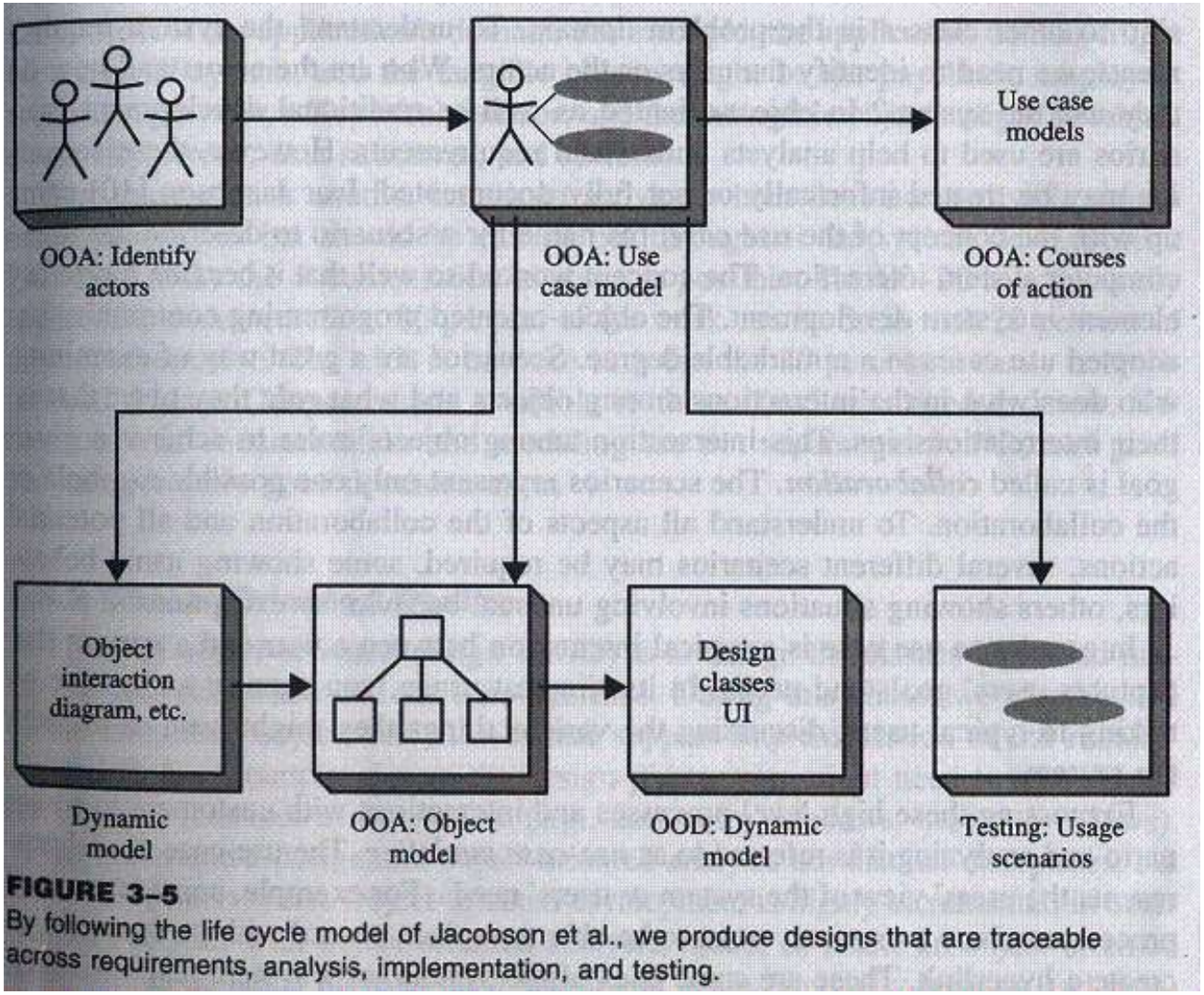


FIGURE 3–5
By following the life cycle model of Jacobson et al., we produce designs that are traceable across requirements, analysis, implementation, and testing.

Object-oriented software development

Activities

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

Encourages

- viewing of system as a system of cooperative objects
- incremental development

Object-oriented analysis - use-case driven

- *Use Case*, is a name for a scenario to describe the user–computer system interaction.

- Determine system requirements, identify classes & their relationship to other classes in domain

To understand system requirements

- need to identify the users or actors

- who are the actors ? How do they use system ?

- Scenarios can help (in traditional development, it is treated informally, not fully documented)

- Jacobson introduces concept of use case - scenario to describe user-computer system interaction

Use case

Typical interaction between user & system that captures users' goal & needs In simple usage, capture use case by talking to typical users, discussing various things they might want to do with system can be used to examine who does what in interactions among objects, what role they play, intersection among objects' role to achieve given goal is called collaboration several scenarios (usual & unusual behaviour, exceptions) needed understand all aspects of collaboration & all potential actions

Use cased modeling expressing high level processes & interactions with customers in a scenario & analyzing it gives system uses, system responsibilities
developing use case is iterative when use case model better understood & developed, start identifying classes & create their relationship

Identifying objects

What are physical objects in system ?

- Individuals,organizations, machines, units of information, pictures, whatever makes up application/ make sense in context of real world

objects help establish workable system

- work iteratively between use-case & object models
- incentive payroll - employee, supervisor, office administrator, paycheck, product made, process used to make product

Intangible objects ?

- Data entry screens, data structures

Documentation

- 80-20 rule
- 80% work can be done with 20% documentation
- 20% easily accessible, 80% available to few who needs to know
- modeling & documentation inseparatable
  - good modeling implies good documentation

Object-oriented Design

Goal : to design classes identified during analysis phase & user interface

Identify additional objects & classes that support implementation of requirements

Eg. add objects for user interface to system (data entry windows, browse windows)

Can be intertwined with analysis phase

Highly incremental, eg. can start with object-oriented analysis, model it, create object-oriented design, then do some more of each again & again, gradually refining & completing models of system

Activities & focus of oo analysis & oo design are intertwined, grown not built

Object-oriented Design

First, build object model based on objects & relationship

Then iterate & refine model

- Design & refine classes
- Design & refine attributes
- Design & refine methods
- Design & refine structures
- Design & refine associations

Prototyping

Prototype – version of software product developed in early stages of product's life cycle for specific, experimental purposes

- •Enables us to fully understand how easy/difficult it will be to implement some features of system
- •Gives users chance to comment on usability & usefulness of user interface design
- •Can assess fit between software tools selected, functional specification & user needs
- •Can further define use cases, makes use case modeling easier
  - •prototype that satisfies user + documentation -> define basic courses of action for use cases covered by prototype

Important to construct prototype of key system components shortly after products are selected

- •Pictures worth a thousand words
- •Build prototype with use-case modeling to design systems that users like & need

Categories of Prototypes

Horizontal prototype

- Simulation of interface (entire interface in full-featured system)
- Contain no functionality
- Quick to implement, provide good overall feel of system

Vertical prototype

- Subset of system features with complete functionality
- Few implemented functions can be tested in great depth

Hybrid prototypes

- Major portions of interface established, features having high degree of risk are prototyped with more functionality

Analysis prototype

- Aid in exploring problem domain, used to inform user & demonstrate proof of concept
- Not used as basis of development, discarded when it has serve purpose
- Final product use prototype concepts, not code

Domain prototype

- Aid for incremental development of the ultimate software solution
- Often used as tool for staged delivery of subsystems to users/other members of development team
- Demonstrate the feasibility of implementation
- Eventually evolve into deliverable product

Implementation: Component-based development

- No more custom development, now assemble from prefabricated components
- No more cars, computers, etc custom designed & built for each customer
- Can produce large markets, low cost, high quality products
- Cost & time reduced by building from pre-built, ready-tested components
- Value & differentiation gained by rapid customization to targeted customers

Component-based development

•Industrialised approach to system development, move form custom development to assembly of pre-built, pre-tested, reusable software components that operate with each other

•Application development improved significantly if applications assembled quickly from prefabricated software components

•Increasingly large collection of interpretable software components could be made available to developers in both general & specialist catalogs

•Components themselves can be constructed from other components, down to prebuilt components/old-fashioned code written in prg languages like C

•Visual tools/actual code can be used to glue together components Visual glue – Digitalk's Smalltalk PARTS, IBM VisualAge

•Less development effort, faster, increase flexibility

Rapid Application Development

•Set of tools & techniques to build application faster than typically possible with traditional methods

•Often used with software prototyping

•Iterational development

•Implement design & user requirements incrementally with tools like Delphi, VisualAge, Visual Basic, or PowerBuilder

Begins when design completed

•Do we actually understood problem (analysis) ?
•Does the system do what it is supposed to do (design) ?
•Make improvement in each iteration

Incremental testing

- Software development and all of its activities including testing are an iterative process.
- Waiting until after development waste money & time
- Turning over applications to quality assurance group not helping since they are not included in initial plan

Reusability:-

A major benefit of object-oriented system development is reusability, and this is the most difficult promise to deliver on. For an object to be really reusable, much more effect must be spent designing it.

Software components for reuse by asking the following questions:

- Has my problem already solved?
- Has my problem been partially solved?
- What has been done before to solve a problem similar to this one?

The reuse strategy can be based on the following:

- Information hiding (encapsulation).
- Conformance to naming standards.
- Creation and administration of an object repository.
- Encouragement by strategic management of reuse as opposed to constant development.
- Establishing targets for a percentage of the objects in the project to be reused.