OBJECT ORIENTED ANALYSIS AND DESIGN– SBSA1403

COURSE OBJECTIVES:

- To Understand the fundamentals of Object-Oriented System Development
- To understand the object-oriented methodologies.
- To use UML in requirements elicitation and designing.
- To understand concepts of relationships and aggregations.
- To test the software against its requirements specification

UNIT I - INTRODUCTION

Overview of object-oriented language systems development – Object basics hierarchy – Object and identity –Static and dynamic binding – Object oriented SDLC.

UNIT II - OBJECT ORIENTED METHODOLOGIES

Rumbaugh et al.'s technique – Booch, Jacobson Methodologies – Patterns – Framework – Unified approach –UML – UML diagrams – UML dynamic modelling – UML extensibility – UML meta-model.

UNIT III - OBJECT ORIENTED ANALYSIS

Use case model – Object analysis classification – Approaches for identifying classes – Classes responsibilities and collaborators – Identifying object relationships, attributes and methods.

UNIT IV - OBJECT ORIENTED DESIGN

Design process and design axioms – Designing classes – Access Layer – Object storage and object Interoperability – View layer – Designing interface objects.

UNIT V - SOFTWARE QUALITY

Software quality assurance – Testing strategies – Test cases – Test plan – Myers debugging principle – System usability and measuring user satisfaction.

COURSE OUTCOMES

CO1 - Understand the basics object model for System development.

CO2 - Understand the object-Oriented Methodologies

CO3 - Express software design with UML diagrams

CO4 - Understand the concept of Relationships

CO5 - Design software applications using OO concepts.

CO6 -Understand the various testing methodologies for OO software

UNIT – I

INTRODUCTION

1. Overview of object-oriented language systems development

- System development refers to all activities that produce an information systems solution.
- System development activities are,
  - Analysis
  - Modelling
  - Design
  - Implementation
  - Testing
  - Maintenance
- A **software development methodology** is a series of processes that can lead to the development of an application.
- Two orthogonal views of the software,
  - Algorithms + Data structures = Programs
- **Program:** A software system is a mechanism for performing certain action on certain data.

1.1 The two orthogonal views of the software

- Traditional system development methodology
  - This approach focuses on the **"Functions"**
- Object oriented system development
  - This approach focuses on the **"Object"** which combines **"Data and Functionality"**

1.2 Object oriented system development methodology

- OOSD methodology is based on functions and procedures.
- OOSD is the way to develop software by building self- contained modules or objects that can be easily,
  - Replaced
  - Modified
  - Reused.
- In an object-oriented system, everything is an object: numbers, arrays, records, fields, files, forms, an invoice, etc.
- An Object is anything, real or abstract, about which we store data and those methods that manipulate the data.
- Conceptually, each object is responsible for itself.
  - E.g., 1. A chart object is responsible for things like maintaining its data andlabels, and even for drawing itself.
  - E.g., 2. A window object is responsible for things like opening, sizing, andclosing itself.

1.3 Why an object orientation?

- To create sets of objects that work together with to produce software the problem.
- The systems are easier to adapt to changing requirements, easier to maintain, more robust and promote greater design and code reuse.
- Object oriented development allows us to create modules of functionality.
- Once objects are defined, they will perform their desired functions.
- Here are some reasons,
  - Higher level of abstraction
  - Seamless transition among different phases of software development.
  - Encouragement of good programming techniques
  - Promotion of reusability
- Higher level of abstraction
  - The top-down approach supports abstraction at the function level
  - The object-oriented approach supports abstraction at the object level.Since **"objects encapsulate both data and functions they work at a higher level of abstraction."**
- Seamless transition among different phases of software development.
  - The object-oriented approach essentially uses the same language to talk about analysis, design, programming, database design.
  - This seamless approach **reduces the level of complexity and redundancy and makes for clear, more robust system development**.
- Encouragement of good programming techniques
  - Object oriented languages produce more modular and reusable code via the concepts of class and inheritance.
  - The object-oriented languages are C++, Smalltalk or Java.
- Promotion of reusability
  - Objects are reusable because they are modelled directly out of a real-world problem domain.
  - The object orientation adds inheritance which is a powerful technique that allows classes to be built from each other.

1.4 Overview of Unified approach

- Unified approach is a methodology for software development.
- The UA is based on methodologies by BOOCH, RUMBAUGH & JACOBSON tries to combine the best practices, processes and guidelines along with the object management group's unified modelling language.
- The unified modelling language (UML) is a set of notations and conventions used to describe and model an application.
- Applying past development efforts to future projects will improve the quality of the product and reduce the cost and development time.
- Steps to be followed in unified approach are

1. Identify the users/actors
2. Develop a simple business process model
3. Develop the use case
4. Interaction diagrams
5. Classification
6. Apply design axioms to design classes
7. Design the access layer
8. Design the view layer
9. Iterate and refine the design/analysis.

2. Object basics

- Object: It is a real-world entity that has properties and methods.
    - E.g., Car.
- Properties (or Attributes): It describes the state (or data) of an object, refer Fig.1
    - E.g., Color, manufacturer, cost, model and owner etc.

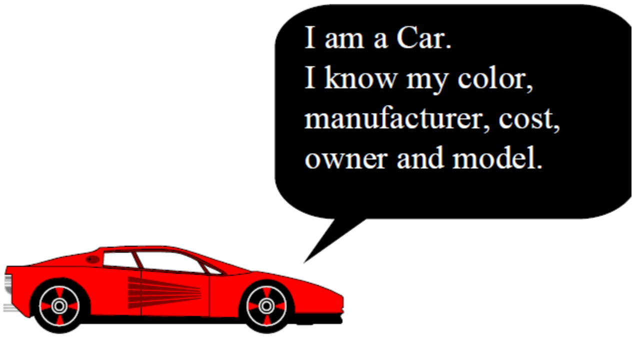


Fig. 1.1. Car Attributes

- Methods (or Procedures): It defines its behavior (or operations), refer Fig.2
    - E.g., Go, stop, turn left, turn right and etc.…

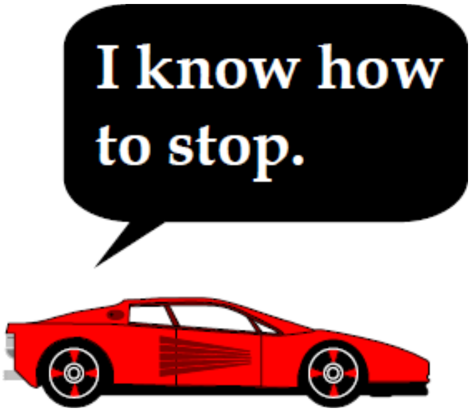


Fig. 1.2. Car Methods

- When developing an object-oriented application, two basic questions always arise,
  - What objects does the application needs?
  - What functionality should those objects have?

2.1 Object-Oriented Philosophy

- **Object oriented programming** is that it allows the base concepts of the language to be extended to include ideas and terms closer to those of its applications.
- The **traditional development methodologies** are either algorithm centric or data centric
  1. **Algorithm-centric methodology:** An algorithm that can accomplish the task, and then build data structures for that algorithm to use.
  2. **Data-centric methodology:** To structure the data, and then build the algorithm around that structure.
- **Object-oriented methodology** is the algorithm and the data structures are packaged together as an object, which has a set of attributes or properties.

2.2 Objects are grouped into classes

- Classes are used to distinguish one type of objects from another
- A class is a set of objects that share a common structure and a common behavior
- Each object is an instance of a class, refer Fig. 3 and 4
- A class is a specification of,
  1. Structure (instance variable)
  2. Behaviour (methods)
  3. Inheritance for objects

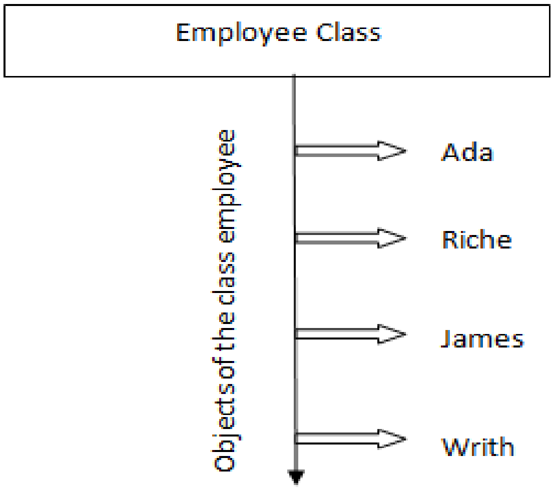


Fig. 1.3. Ada, Riche, James and Writh are objectsof the class Employee

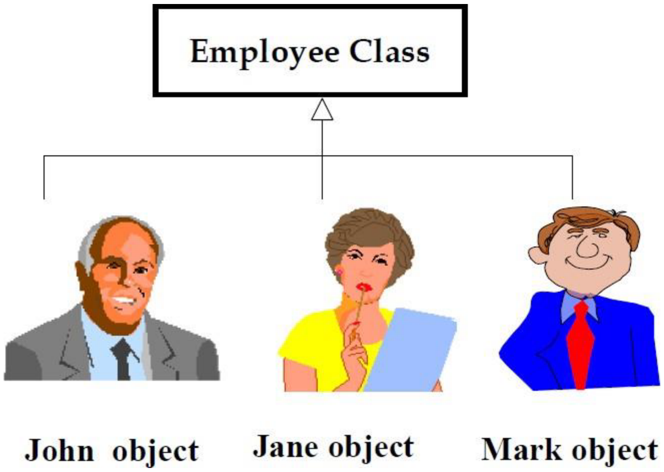


Fig. 1.4. Employee class

2.3 Attributes or properties

- It represented by data type.
- It describe the state of an object
- Eg. Car attributes, see Fig. 5.

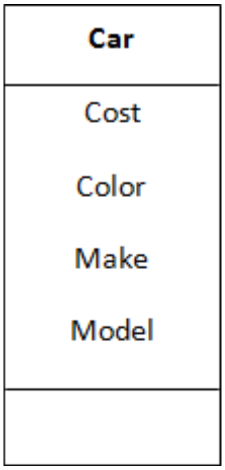


Fig. 1.5. Attributes

2.4 Behavior or methods

- An object behaviour is described in methods or procedures
- A method implements the behaviour of an object.
- Behaviour denotes the collection of methods that abstractly describes what an object is capable of doing.
- Eg. Drive a car – Methods, refer Fig.6.

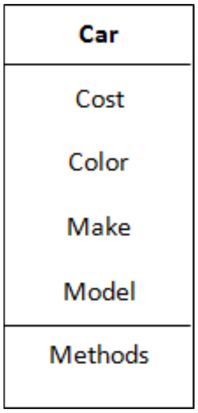


Fig. 1.6. Methods

2.5 Objects Respond to Messages

- A message is much more general than function call.
- E.g., Problem is to make French onion soup. Sol: Your instruction is the message, the way the French onion soup prepared is the method, and the French onion soup is the object.
- Objects perform operations in response to messages.
- Objects respond to messages according to methods defined in its class.
- E.g., See the Fig. 7
  1. Send a Brake message to a Car object
  2. Send a multiplication *7 message to 5 objects

3. Send a Compute payroll message to Employee object

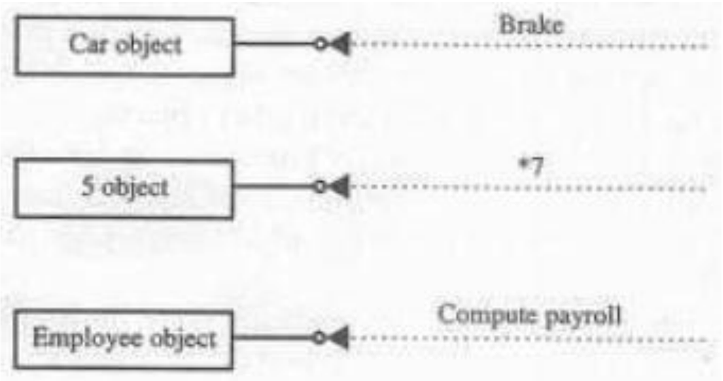


Fig. 1.7. Object responds to message

2.6 Encapsulation and information hiding

- Information hiding is the principle of hiding internal data and procedures of an object, refer Fig.8

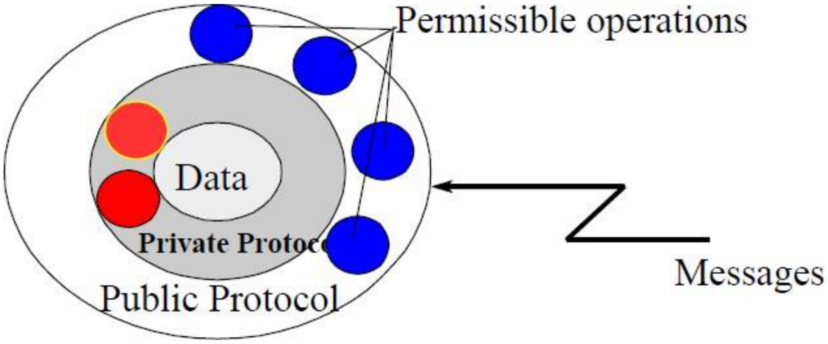


Fig. 1.8. Access specifiers

- Encapsulation protects the data from corruption
- Providing an interface to each object in such a way as to reveal as little as possible about inner workings.
- Encapsulation means in general, protection mechanism by using access specifiers such as public, protected and private.
- Data are abstracted when they are shielded or hided by a full set of methods and only those methods can access the data portion of an object.
- E.g., A car engine.

3. Class Hierarchy

- An object-oriented system organized classes into a "Subclass-Super class hierarchy".
- At the top of the hierarchy are the most general (base) classes and at the bottom are the most specific classes (sub-classes).
- The parent class also is known as the bases class or super class.
- A subclass inherits all of the properties and methods (procedures) defined in its super class, refer Fig. 9 and 10.

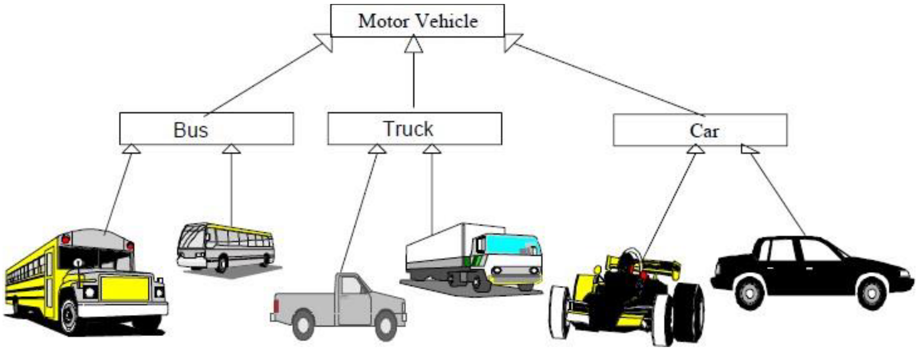


Fig. 1.9. Super class / Sub class Hierarchy

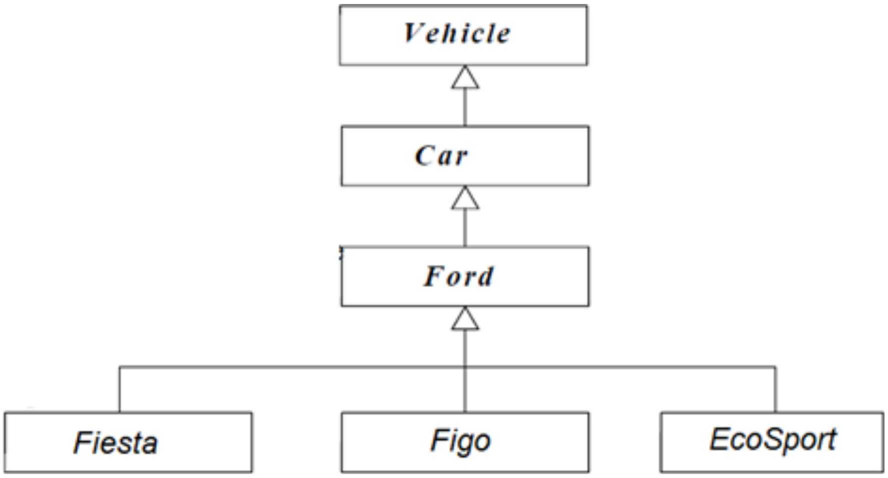


Fig. 1.10. Class hierarchy for Ford class

3.1 Inheritance

- Inheritance is a relationship between classes where one class is the parent class of another (derived) class, refer Fig. 11.
- Inheritance allows classes to share and reuse behaviours and attributes.
- The real advantage of inheritance is that we can build and reuse what we already have.

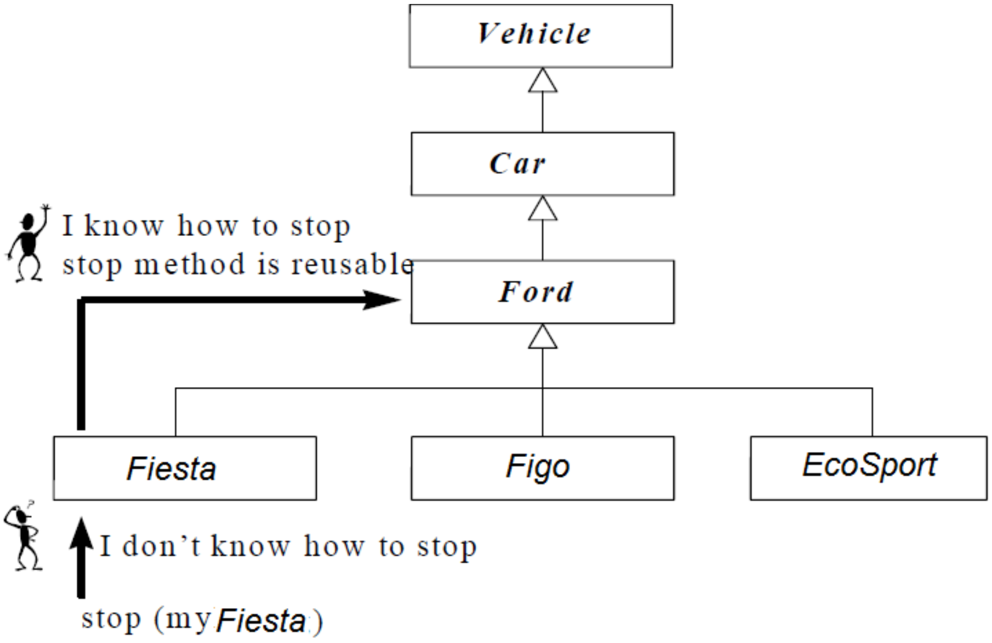


Fig. 1.11. Inheritance allows reusability

- **Dynamic Inheritance**
  1. It allows objects to change and evolve over time.
  2. Dynamic inheritance refers to the ability to add, delete or change parents from objects at run time.

3.2 Multiple Inheritance

- Object oriented systems permit a class to inherit its state (attributes) and behaviors from more than one super class. This kind of inheritance is referred to as "Multiple Inheritance". Refer Fig. 12.

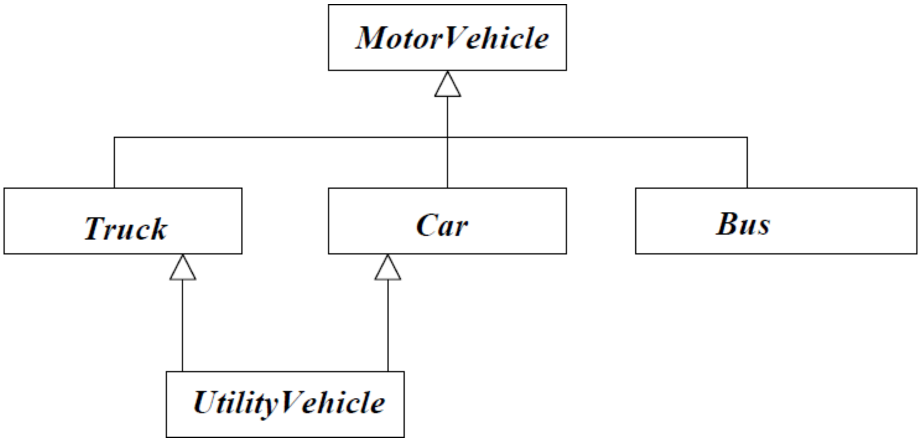


Fig. 1.12. Utility vehicle inherits from both the car and truck class

3.3 Polymorphism

- Poly means “Many” and morph means “Form”
- Polymorphism means that the same operation may behave differently on different classes. Refer Fig. 13 and 14

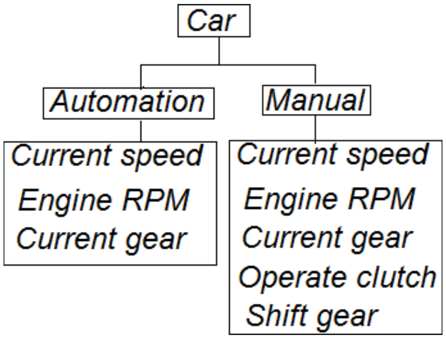


Fig. 1.13. Car object represents polymorphism

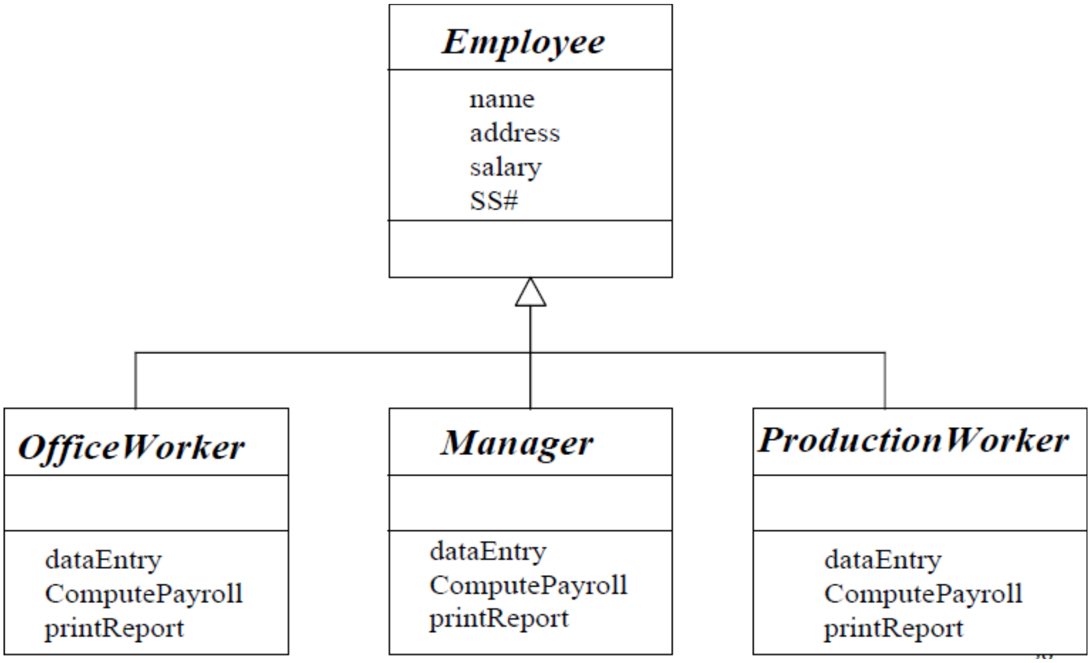


Fig. 1.14. Compute Payroll represents polymorphism

3.4 Object relationship and Association

- The concept of association represents relationships between objects and class.
- Associations are bidirectional by which they can be traversed in both directions
- E.g., A pilot can fly planes, refer Fig.15.

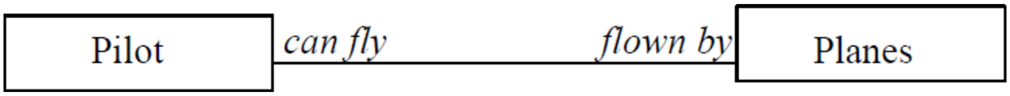
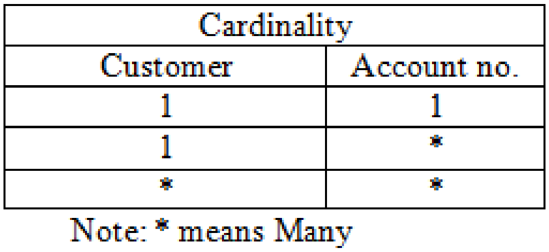


Fig. 1.15. Association represents the relationship among objects which is bidirectional

- **Cardinality:** it is an important issue in associations which specifies how manyinstances of one class may relate to a single instance of a associated class.
- E.g., Client account relationship, refer Fig. 16.



Cardinality	
Customer	Account no.
1	1
1	*
*	*

Note: * means Many

Fig. 1.16. Cardinality in association

- Consumer - producer association, also known client-server association or use relationship.
- A special form of association is a client-server relationship.
- This relationship can be viewed as one-way interaction: one object (client) requests the service of another object (server). Refer Fig. 17.

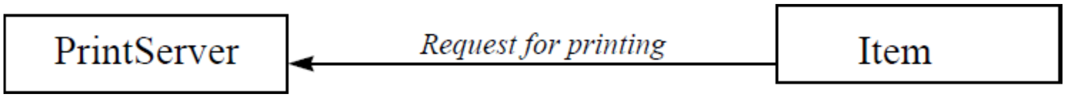


Fig. 1.17. One-way direction association

4. Object and Identity

- A special future of object-oriented system is that every object has its own unique and immutable identity.
- The identity name never changes even it all the properties of the object change, i.e., it isindependent of the object's state.

- The identity does not depend on the objects name, or its key or its location.
- In an object system, object identity often is implemented through some kind of "OBJECT IDENTIFIER" (OID) or "UNIQUE IDENTIFIER" (UID). An OID is dispersed by a part of object-oriented programming system that is responsible for guaranteeing the uniqueness of every identifier.
- OID's are never reused.
- E.g., The car may have an **owner** property, which refer the class person. A person has an **owns** property that contains a reference to the car instance. The relationship between the car and person can be implemented and maintained by a reference, refer Fig. 18.

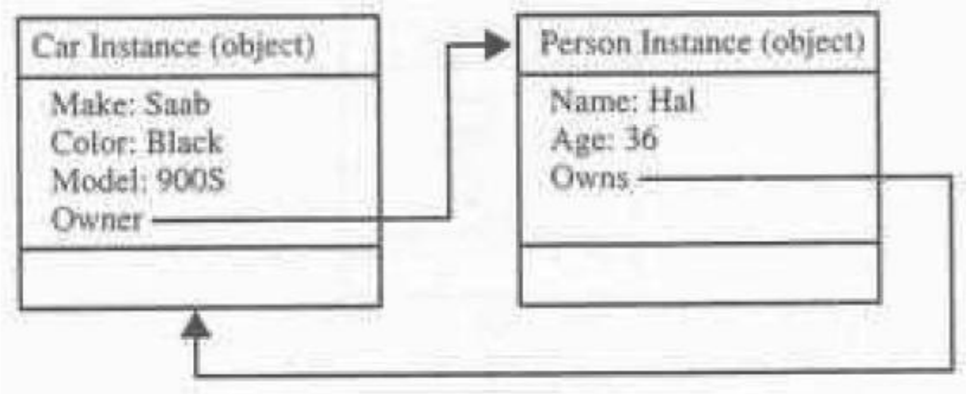


Fig. 1.18. The owner property of the car contains the reference to the personinstance named Hal

4.1 Static and Dynamic Binding

- The process of determining, which function to invoke at run time is called as "Dynamic binding".
- E.g., Polymorphism function calls
- Making this determination earlier, at compile time is called as "Static binding".
- E.g., Normal function calls

4.2 Object Persistence

- Objects have a life time. They are explicitly created and can exist for a period of time.
- An object can persist beyond application session boundaries, during which the object is stored in a database or a file.

4.3 Meta Classes

- Class is an object. If it is an object it must belong to a class, such a class belongs to a class called a "Meta Class" or a class of classes.
- Meta-classes are used by a compiler.
    1. E.g. The meta-classes handle messages to classes, such as constructors, new and class variables, which are variables shared between all instance of a class.

5. Object Oriented – SDLC

- The essence of the software development process that consists of
  - Analysis
  - Design
  - Implementation
  - Testing
  - Refinement
- To transform users' needs into a software solution that satisfies those needs.

5.1 Software Development Process

- "System or software development can be viewed as a process"
- The process can be divided into small, interacting phases called sub-processes.
- Each sub-processes 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.
- The "Software development process" can be viewed as a series of transformations, where the output of one transformation becomes the input of the subsequent transformation. Refer Fig.19.

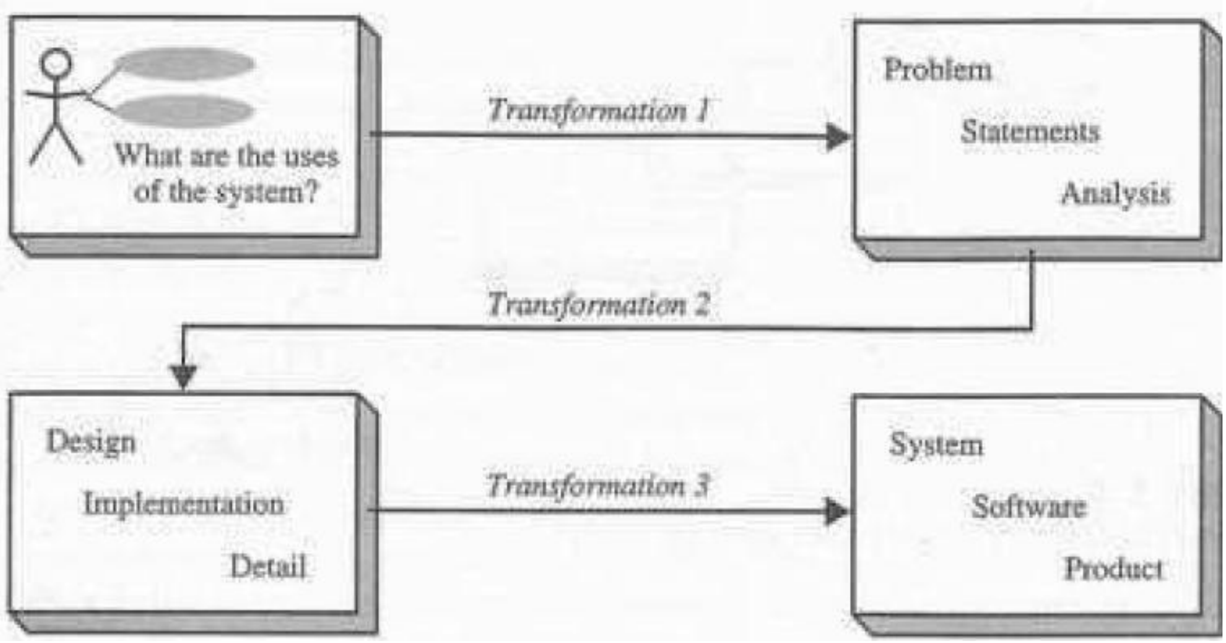


Fig. 1.19. Software process reflecting transformation from needs to a software product that satisfiesthose needs.

- **Transformation 1 (Analysis):** Translates the users' needs into system requirementsand responsibilities.
- **Transformation 2 (Design):** Begins with a problem statement and ends with adetailed design that can be transformed into an operational system.
- **Transformation 3 (Implementation):** Refines the detailed design into the systemdeployment that will satisfy the user's needs.

"An Example (see Fig. 20) of the software development process is Waterfall approach"

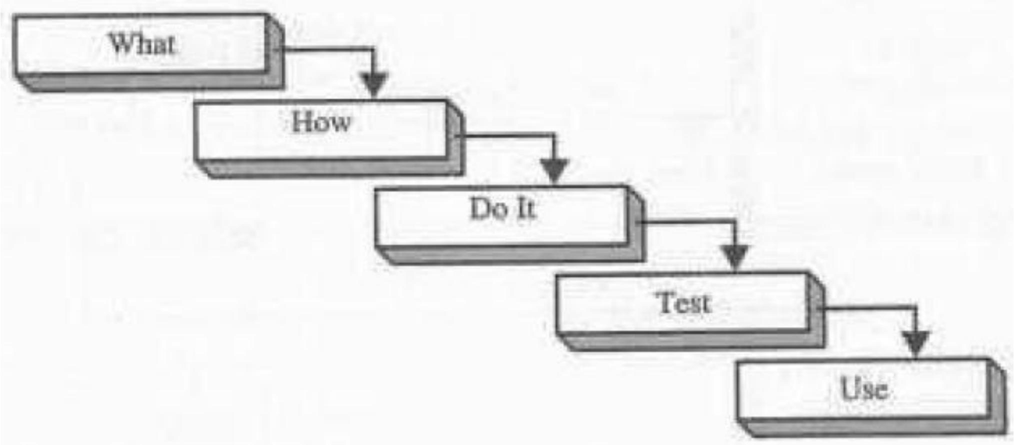


Fig. 1.20. The waterfall software development process

"Waterfall Approach" starts with deciding **what** is to be done. Once the requirements have been determined, we must decide **how** to accomplish them. This is followed by a step in which we **do it**, whatever it has required us to do. We then must**test** the result to see if we have satisfied the user's requirements, finally we **use** what we have done.

5.2 Building high quality Software

- The ultimate goal of building high quality software is user satisfaction.
- To achieve high quality in software we need to be able to answer the following questions.
  1. How do we determine when the 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?
- Four Quality measures for software evaluation (see Fig. 21)
  1. **Correspondence:** It measures how well the delivery system matches the needsof the operational environment
  2. **Validation:** It is the task of predicting correspondence

3. **Correctness:** It measures the consistency of the product requirements withrespect to the design specification
4. **Verification:** It is the exercise of determining correctness

- **Verification:** Am, I building the product, right?
- **Validation:** Am, I building the right product?

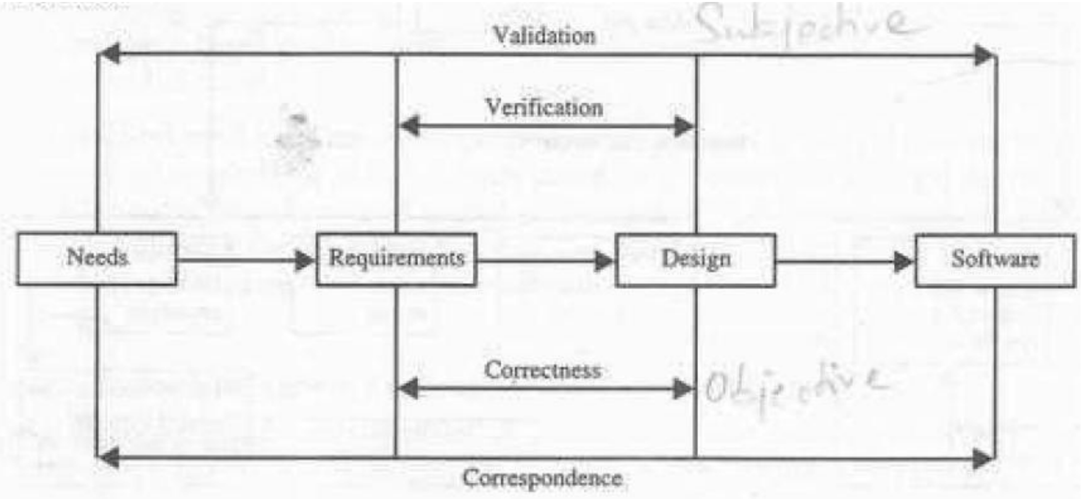


Fig. 1.21. Four quality measures for software evaluation

Validation begins as soon as the project starts, but verification can begin only after a specification has been accepted. Validation & verification are independent of each other.

6. Object Oriented System Development Life Cycle (SDLC): A Use Case Driven Approach

- The object oriented SDLC (Software Development Life Cycle) consists of 3 macro processes, refer Fig. 22.
  i) Object Oriented Analysis (OOA)
  ii) Object Oriented Design (OOD)
  iii) Object Oriented Implementation (OOI)
- It includes the following activities
  1. Object Oriented Analysis – Use case driven
  2. Object Oriented Design
  3. Prototyping
  4. Component based development
  5. Incremental testing

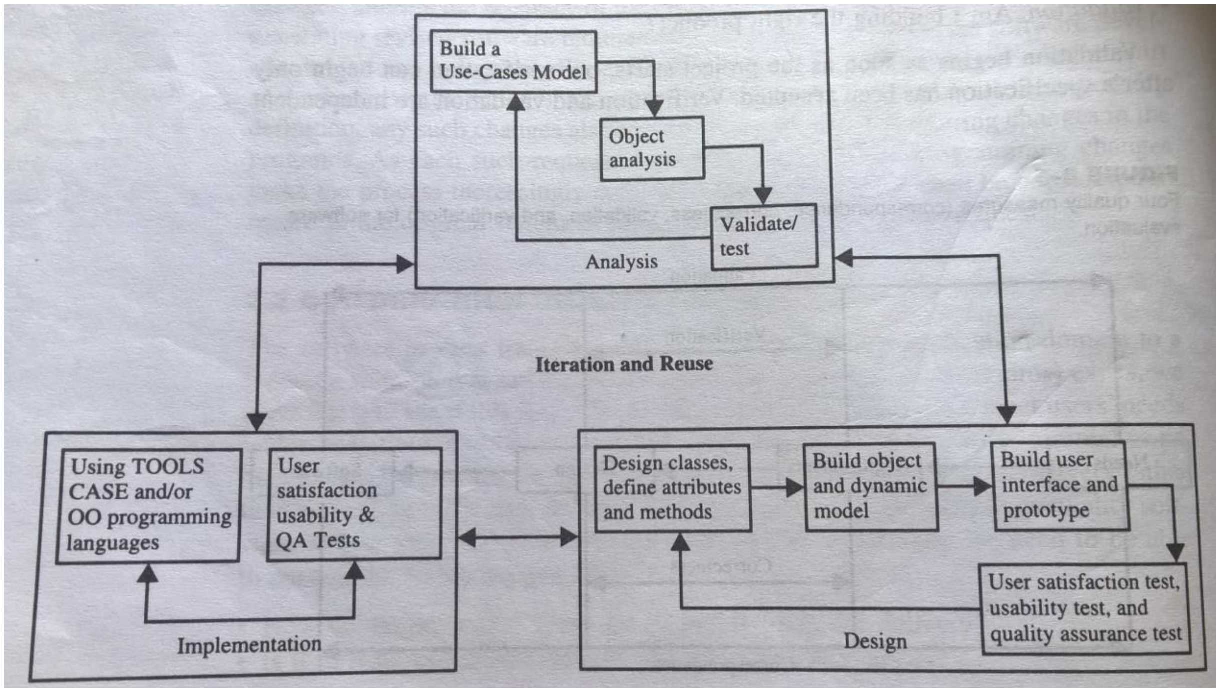


Fig. 1.22. The object-oriented system development approach

1) Object Oriented Analysis (OOA) – Use case driven

o OOA phase of software development is concerned with

a) Determining the system requirements

b) Identifying classes & their relationship

c) To understand the system requirements, we need to identify the users or actors

d) **"Use Case"** – It is a typical interaction between user and a system that captures users' goals and needs

e) **"Use Case Model"** – The use case model represents the users view of the system or the user needs

f) The process of developing the use cases, like other object-oriented activities are iterative

g) The objects need to have meant only within the content of the application domain.
e.g., The application domain might be a payroll system and the objects are paycheck, employee, worker, supervisor, office administrator.

h) Documentation

a. It is another important activity which does not end with object-oriented analysis but should be carried out throughout the system development.

b. 80-20 rule generally applies for documentation, i.e., 80% of the work canbe done with 20% of the documentation.

2) Object Oriented Design (OOD)

a) The goal of OOD is to design the classes identified during the analysis phase
b) OOA and OOD are distinct disciplines, but they can be interconnected.
c) In OOD, design classes, define attributes and methods then build object and dynamic model.
d) First build the object model based on objects and their relationship then iterate and refine the model
    a. Design and refine classes
    b. Design and refine attributes
    c. Design and refine the methods
    d. Design and refine the structures
    e. Design and refine the associations
e) Guidelines to use in your OOD
    a. Reuse rather than build a new class know the existing classes
    b. Design a large number of simple classes, rather than a small number of complex classes
    c. Design methods
    d. A detailed analysis, what you have proposed. If possible, go-back and refine the classes.

3) Prototyping

a) A prototype is a version of a software product developed in the early stages of the products life cycle for specific, experimental purposes.
b) Prototyping can further define the use-cases, and actually makes use-case modelling much easier.
c) Prototyping provides the developer a means to test and refine the user interface and increase the usability of the system.
d) Prototypes have been categorized in various ways
    a. Horizontal prototype:
        a. It's a simulation of the interface but contains no functionality.
        b. An advantage is quick to implement.
    b. Vertical prototype:
        a. It is a subset of the system features with complete functionality.
        b. An advantage is that the few implemented functions can be tested in great depth.
    c. Analysis prototype:
        a. It is an aid for exploring the problem domain.
        b. The final product will use the concepts exposed by the prototype, not its code.
    d. Domain prototype:

a. It is an aid for the incremental development of the ultimate software solution.
b. It demonstrates the feasibility of the implementation.

4) Implementation: Component Based Development

- Component Based Development (CBD) is an industrial approach to software development.
- Software components are functional units, or building blocks offering a collectionof reusable services.
- A CBD developer can assemble components to construct a complete software system.
- A component-based development is concerned with the implementation and system integration aspects of software development.
- **CASE:** Computer Aided Software Engineering Tools allow their users to rapidly develop information systems.
- **Goal:** The main goal of CASE technology is the automation of the entire systems development life cycle process using a set of integrated software tools, such as modelling, methodology and automatic code generation.
- **RAD:** Rapid Application Development
- RAD is to build a version of an application rapidly to see whether we actually understood the problem.
- It builds the application quickly through tools such as Delphi, Visual Age, Visual Basic.

5) Incremental Testing

- To test an application for bugs and performance.
- After development of applications are finally given to Quality Assurance group for testing the process.

TEXT/ REFERENCE BOOKS:

1. Ali Bahrami, “Object oriented systems development using the unified modelling language”, 1st Edition, McGraw-Hill, 1998.
2. Grady Booch, James Rumbaugh, and Ivar Jacobson, “The Unified Modelling Language User Guide”, 3rd Edition Addison Wesley,2007.
3. John Deacon, "Object Oriented Analysis and Design", 1st Edition, Addison Wesley, 2005.
4. Grady Booch, James Rumbaugh, and Ivar Jacobson, “The Unified Modelling Language User Guide”, 3rd Edition Addison Wesley.
5. John Deacon, “Object Oriented Analysis and Design”, 1st Edition, Addison Wesley.
6. Bernd Oestereich, “Developing Software with UML, Object - Oriented Analysis and Design in Practice”, Addison-Wesley

Questions

Part-A			
Q.No	**Questions**	**Competence**	**BT Level**
1.	Describe software development methodology?	Knowledge	BTL 1
2.	Write are the orthogonal views of the software?	Create	BTL 6
3.	Define object-oriented system development methodology?	Knowledge	BTL 1
4.	Classify traditional approach and object-oriented approach	Understand	BTL 2
5.	Compare the advantages of object-oriented development?	Understand	BTL 2
6.	Describe the components of the unified approach.	Knowledge	BTL 1
7.	Illustrate the uses of UML?	Analyze	BTL 4
8.	Contrast the methods and the messages	Analyze	BTL 4
9.	Illustrate How are the objects identified in object-oriented system?	Analyze	BTL 4
10.	Develop the software development process?	Create	BTL 6
Part-B			
Q.No	**Questions**	**Competence**	**BT Level**
1.	Summarize the overview of object-oriented language systems development in detail.	Understand	BTL 2
2.	Illustrate the various object or class hierarchy in detail.	Analyze	BTL 4
3.	Reframe the object basics in object-oriented development methodology in detail.	Evaluate	BTL 5
4.	Sketch the following: a) Object identity b) Static and dynamic binding	Apply	BTL 3
5.	Illustrate the object-oriented system development life cycle (SDLC): A Use Case Driven Approach in detail.	Analyze	BTL 4
6.	Defend prototyping? How it is useful?	Evaluate	BTL 5
7.	Contrast software verification is can differ from software validation?	Analyze	BTL 4