OBJECT ORIENTED ANALYSIS AND DESIGN

UNIT-III: Object-Oriented Analysis process – Use-case Model-Object Analysis: Classification- Noun Phrase approach – Classes, responsibilities and collaborators - Associations-Super-Sub class relationships-Aggregation.

TEXT BOOK: Ali Bahrami, "Object Oriented Systems Development", TATA McGraw-Hill Edition, 2008.

PREPARED BY
Dr. D. Devakumari

CONTENT

- Object-Oriented Analysis process
- Use-case Model
- Object Analysis: Classification
- Noun Phrase approach
- Classes, responsibilities and collaborators
- Associations
- Super-Sub class relationships
- Aggregation.

Use-case Model

- *Use cases are scenarios for understanding system requirements.*
- *A use case an interaction between users and a system.*
- *It captures the goal of the users and the responsibility of the system to its users.*
- *The use-case model describes the uses of the system and shows the courses of events that can be performed.*
- *A use-case model can be developed by taking to typical users and discussing the various things they might want to do with the application being prepared.*

- **Use cases under the microscope**
- **Uses and Extends Associations**
- **Identifying the Actors**
- **Guidelines for Finding use cases**
- **How Detailed must a use case Be? when to stop Decomposing and when to continue**
- **Dividing use cases into packages:**
- **Naming a use cases: -**

Use cases under the microscope

- **Use case:** Use case is a special flow of events through the system.
- **Actors:** An actor is a user playing a role with respect to the system.
- **In a system:** this simply means that the actor communicate with the system’s use case.
- **A measurable value:** a use case must help the actor to perform a task that has some identifiable value; for example, the performance of a use case in terms of price or cost. For example, borrowing books is something of value for a member of the library.
- **Transaction:** a transaction is an atomic set of activities that are performed either full or not at all.
- **Use-case name: Borrow books**. A member takes books from the library to read at home, registering them at the checkout desk so the library can keep track of its books. Depending on the member’s record, different courses of events will follow.
- **Use-case name: Get an interlibrary loan**. A member requests a book that the library does not have. The book is located at another library and ordered through an interlibrary loan.
- **Use-case name: Return books.** A member bring borrowed books back to the library.
- **Use-case name: check library card.** A member submits his or her library card to the clerk, who checks the borrower’s record.
- **Use-case name: Do research.** A member comes to the library to do research. The member can search in a variety of ways (such as through books, journals, CD-ROM, WWW) to find information on the subjects of that research.
- **Use-case name Read books, newspaper.** A member comes to the library for a quiet place to study or read a newspaper, journal, or book.
- **Use-case name: purchase supplies.** The supplier provides the books, journals, and newspapers purchased by the library.

Uses and Extends Associations.

- The extends association is used when you have one use case that is similar to another use case but does a bit more or is more specialized: in essence, it is like a subclass.
- The uses association occurs when you are describing your use cases and notice that some of them have sub flows in common. To avoid describing a sub flow more than once in several use cases, you can extract the common sub flow and make it a use case of its own.

Identifying the Actors

- A user may play more than one role. For instance, a member of a public library also may play the role of volunteer at the help desk in the library. However, an actor should represent a single user.
- Who is using the system? Or, who is affected by the system? Or, which groups need help from the system to perform a task?
- Who affects the system? Or, which user groups are needed by the system to perform its functions? These functions can be both main functions and secondary functions, such as administration.

1. Which external hardware or other system (if any) use the system to perform tasks?
2. What problems does this application solve (that is, for whom)?
3. And, finally, how do users use the system (use case)? What are they doing with the system?

Guidelines for finding use cases:

- The step for finding use cases

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

How Detailed must a use case Be? when to stop Decomposing and when to continue

- Develop one use-case diagram as the System use case and draw packages on this use case to represent the various business domains of the system.
- For each package, you may create a child use- case diagram.
- On each child use-case diagram, you can draw all of the use cases of the domain, with actions and inter actions.
- You can further refine the way the use cases are categorized.
- The extends and uses relationships can be used to eliminate redundant modelling of scenarios.

Dividing use cases into packages:

- A design is broken down into packages. you must narrow the focus of the scenarios in your system.
- For example, in a library system, the various scenarios involve a supplier providing books or a member doing research or borrowing books.
- In this case, there should be three separate packages, one each for Borrow books
- Do research, and purchase books.

Naming a use cases: -

- The name should express what happens when an instance of the use case is performed.
- For example, the use case that describes what happens when a person deposits money onto an ATM machine could be Named either receive money or deposit money.

Object Analysis: Classification

Approaches for Identifying classes:-

- Four alternative approaches for identifying classes: The noun phrase approach; the
- common class patterns approach; the use- case derive, sequence/ collaboration modelling
- approach; and the classes, Responsibilities, and collaborators (CRC) approach.

Noun Phrase approach

- *The noun phrase approach was proposed by Rebecca Wirfs-Brock, Brian Wilker -son, and Lauren Wiener.*
- *In this method, you read through the requirements or use cases looking for noun phrases.*
- *Nouns in the textual description are considered to be classes and verbs to be methods of the classes.*
- *The nouns are listed, and the list divided into three categories: relevant classes, fuzzy classes (The 'fuzzy area," classes we are not sure about), and irrelevant classes.*

1. **Identifying Tentative classes.**
2. **Selecting classes from the Relevant and fuzzy categories.**
3. **The vialNet Bank ATM System: Identifying classes by using Noun phrase Approach.**
4. **Initial List of Noun phrases: candidate classes.**
5. **Reviewing the Redundant classes and Building a common vocabulary.**
6. **Reviewing the classes containing Adjectives.**
7. **Reviewing the possible Attributes.**
8. **Reviewing the class purpose.**

Identifying Tentative classes: -

- The following are guidelines for selecting classes in an application.
- Look for nouns and noun phrase in the use cases.
- Some classes are implicit or taken from general knowledge.
- All classes must make sense in the application domain; avoid computer Implementation classes- defer them to the design stage. carefully choose and define class names.
- Finding classes is an incremental and iterative process.

Selecting classes from the Relevant and fuzzy categories.

- **Redundant classes**:- DO not keep two classes that express the same information. Choose your vocabulary carefully use the word that is being used by the user of the system.
- **Adjective classes**:- An adjective can suggest a different kind of object, different use of the Same object, or it could be utterly irrelevant
- For example, Adult Members behave differently then youth Members, so the two should be classified as different classes.
- **Attribute classes**:- Tentative objects that are used only as values should be defined or restated as attributes and not as a class.
- **Irrelevant classes**:- Each class must have a purpose and every class should be clearly defined and necessary.

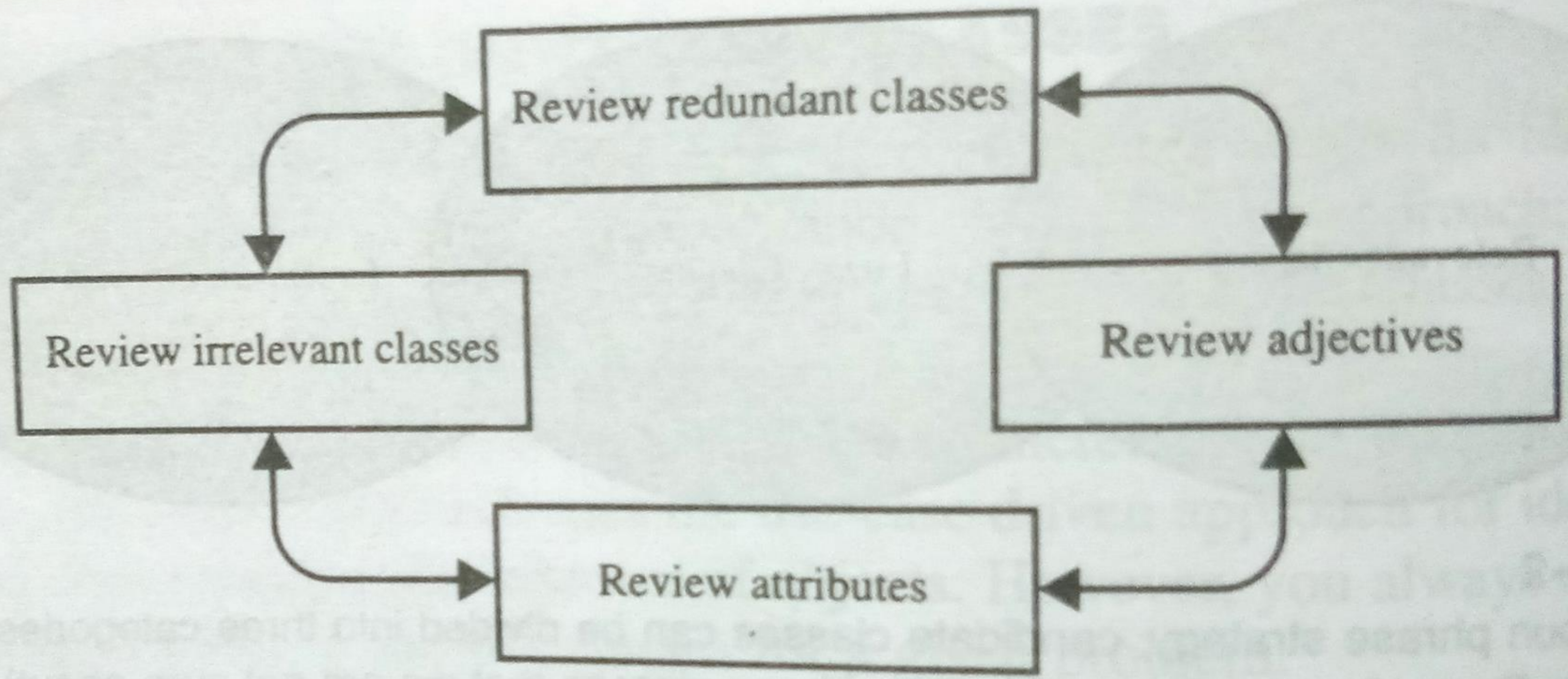


FIGURE 7–3
The process of eliminating the redundant classes and refining the remaining classes is not sequential. You can move back and forth among these steps as often as you like.

- **The vialNet Bank ATM System: Identifying classes by using Noun phrase Approach.**
- **Initial List of Noun phrases: candidate classes:-**
- The following irrelevant classes can be eliminated because they do not belong to the problem statement: Envelope, four Digits, and step.

- Account Balance
- Amount
- Approval process
- ATM Card
- ATM Machine
- Bank
- Bank client
- card
- cash
- check
- checking
- checking Account
- client
- client's Account
- currency
- Dollar
- Envelope
- Four Digits
- Fund
- Invalid PIN
- message
- Money
- password
- PIN
- PIN code
- Record
- savings
- savings Account
- step
- system
- Transaction
- Transaction History.

Reviewing the Redundant classes and Building a common vocabulary.

client, Bank client	=	Bank client (the term chosen)
Account, client's Account	=	Account
PIN, PIN Code	=	PIN
checking, checking Account	=	checking Account
savings, savings Account	=	Saving Account
Fund, Money	=	Fund
ATM card, card	=	ATM card.

Reviewing the classes containing Adjectives

- In this example, we have no classes containing adjectives that we can eliminate.

Reviewing the possible Attributes.

- ***Amount***: A value, not a class.
- ***Account Balance***: An attribute of the Account class.
- ***Invalid PIN:*** It is only a value, not a class.
- ***Password:*** An attribute, possibly of the Bank client class.
- ***Transaction History:*** An attribute, possibly of the Transaction class.
- ***PIN:*** An attribute, possibly of the Bank Client class.

Reviewing the class purpose: -

- The classes that add no purpose to the system have been deleted from the list. The candidate classes are these:
  - ***ATM Machine classes****: provides an interface to the ViaNet bank.*
  - ***ATM card class****: Provides a client with a key to an account.*
  - ***Bank client class****: A client is an individual that has a checking account and, possibly, a savings account.*
  - ***Bank class****: Bank clients belongs to the Bank. It is a repository of accounts and processes the account's transactions.*
  - ***Account class****: An Account class is a formal (or abstract) class, it defines the common behaviours that can be inherited by more Specific classes such as checking account and savings Account.*
  - ***Checking Account classes****: It models a client's checking account and provides more specialized withdrawal Service.*
  - ***Savings Account class****: it models a client's savings Account.*
  - ***Transaction class****: keeps track of transaction, time, date, type, amount, and balance.*

Classes, responsibilities and collaborators

- classes, Responsibilities, and collaborators is a technique used for identifying classes‘ responsibilities and therefore the attributes and methods.
- classes, Responsibilities, and collaborators is based on the idea that an object either can accomplish a certain responsibility itself or it may require the assistance of other objects.
- By identifying an object's responsibility and collaborators (co-operative objects with which it works) you can identify its attributes and methods.
- classes, Responsibilities, and collaborators cards are 4" X 6" index cards. All the information for an object is written on a card, which is cheap, portable, readily available, and familiar.

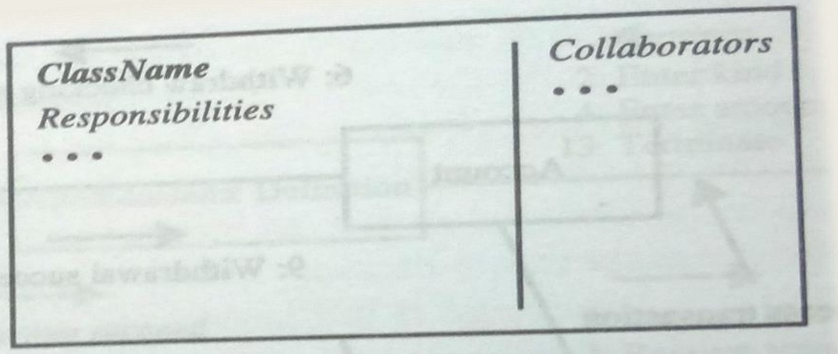


FIGURE 7–9
A Classes, Responsibilities, and Collaborators (CRC) index card.

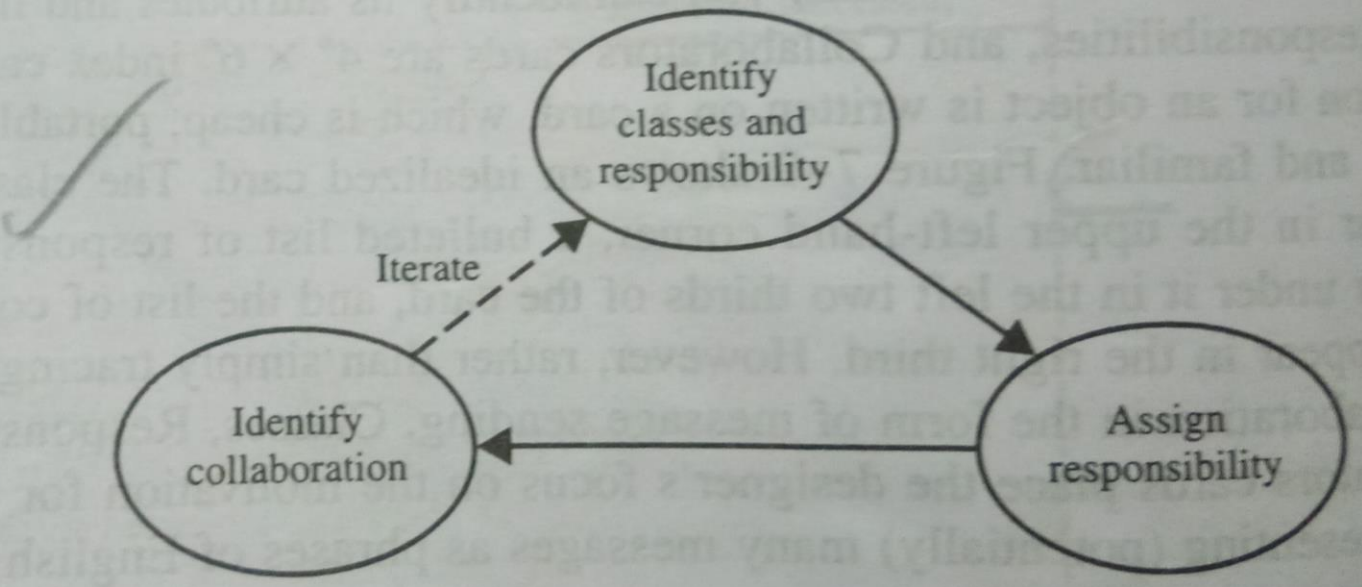
Classes, Responsibilities, and collaborators process.

- Three Steps:

1. Identify classes' responsibilities (and identify classes).
2. Assign responsibilities.
3. Identify collaborators.

FIGURE 7–10

The Classes, Resposibilities, and Collaborators process.



■ **The Vio Net Bank ATM System: Identifying classes by using classes, Responsibilities, and collaborators.**

- To identify objects' responsibilities such as attributes and methods in that system.
- The class Account is responsible mostly to the Bank client class and it collaborates with several objects to fulfil its responsibilities.
- Among the responsibilities of the All want class to the Bank client class is to keep track of the Bank client balance, Account number, and other data that need to be remembered.
- These are the attributes of the Account class. Furthermore, the Account class provides certain services or methods, such as means for Bank client to deposit or withdraw an amount and display the account's Balance.

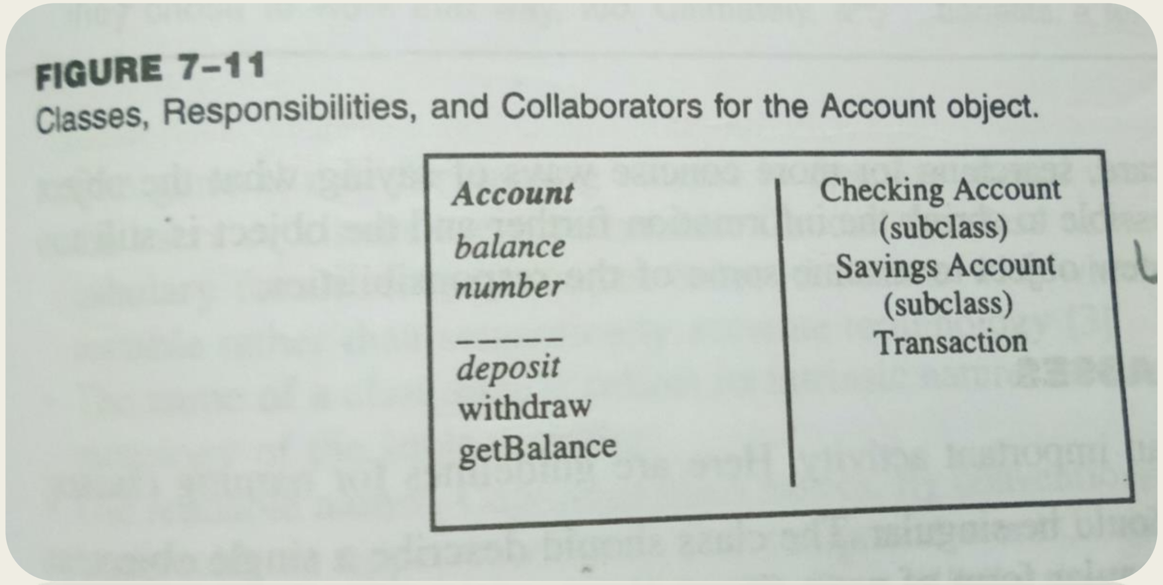


FIGURE 7–11

Classes, Responsibilities, and Collaborators for the Account object.

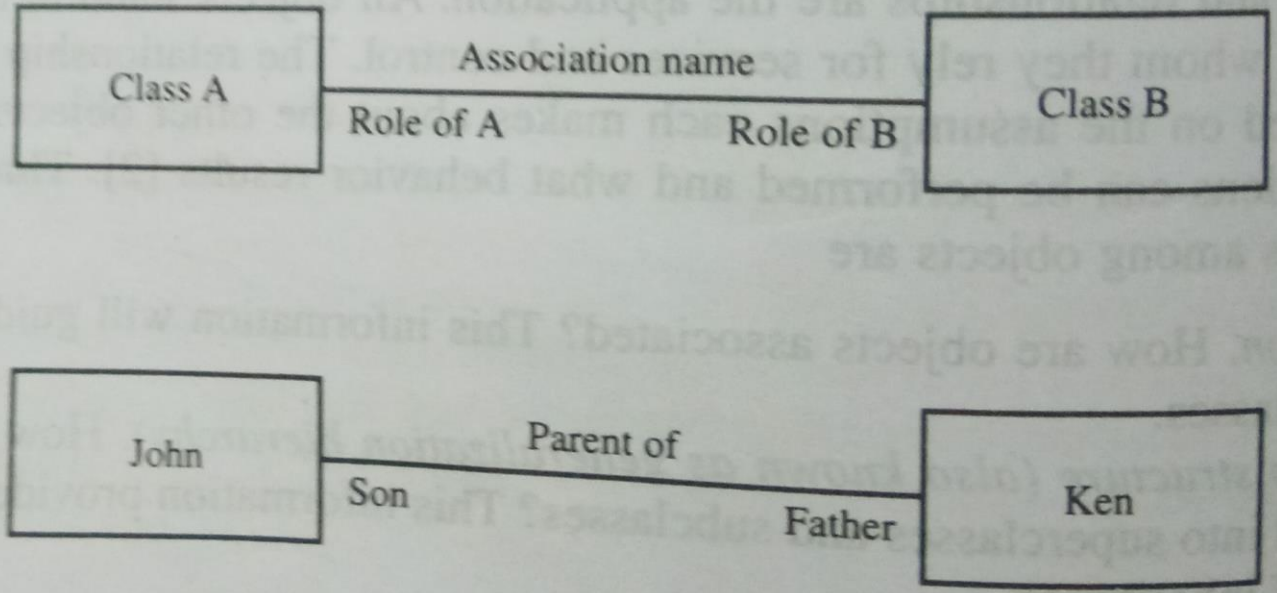
Account	Checking Account (subclass)
balance	Savings Account (subclass)
number	Transaction
-----	
deposit	
withdraw	
getBalance	

Associations

- Association represents a physical or conceptual connection between two or more objects.
- For example, if an object has the responsibility for telling another object that a credit card number is valid or invalid the two classes have an association.

FIGURE 8–1

Basic association. See Chapter 5 for a detailed discussion of association.



Identifying Associations

- Identifying associations begins by analysing the interactions between classes.
- you must examine the responsibilities to determine dependencies.
- In other words, if an object is responsible for a specific task (behaviour) and lacks all the necessary knowledge needed to perform the task, then the object must delegate the task to another object that possesses such knowledge.
- Is the class capable of fulfilling the required task by itself?
- If not, what does it need?
- From what other class can it acquire what it needs?

Guidelines for Identifying Association

- A dependency between two or more classes may be an association.
- A reference from one class to another is an association.

Common Association patterns.

- Location association - next to, part of, contained in. For example, consider a soup object, cheddar cheese is a-part-of soup.
- Communication association -talk to, order to, For example, a customer places an order (communication association) with an operator person.

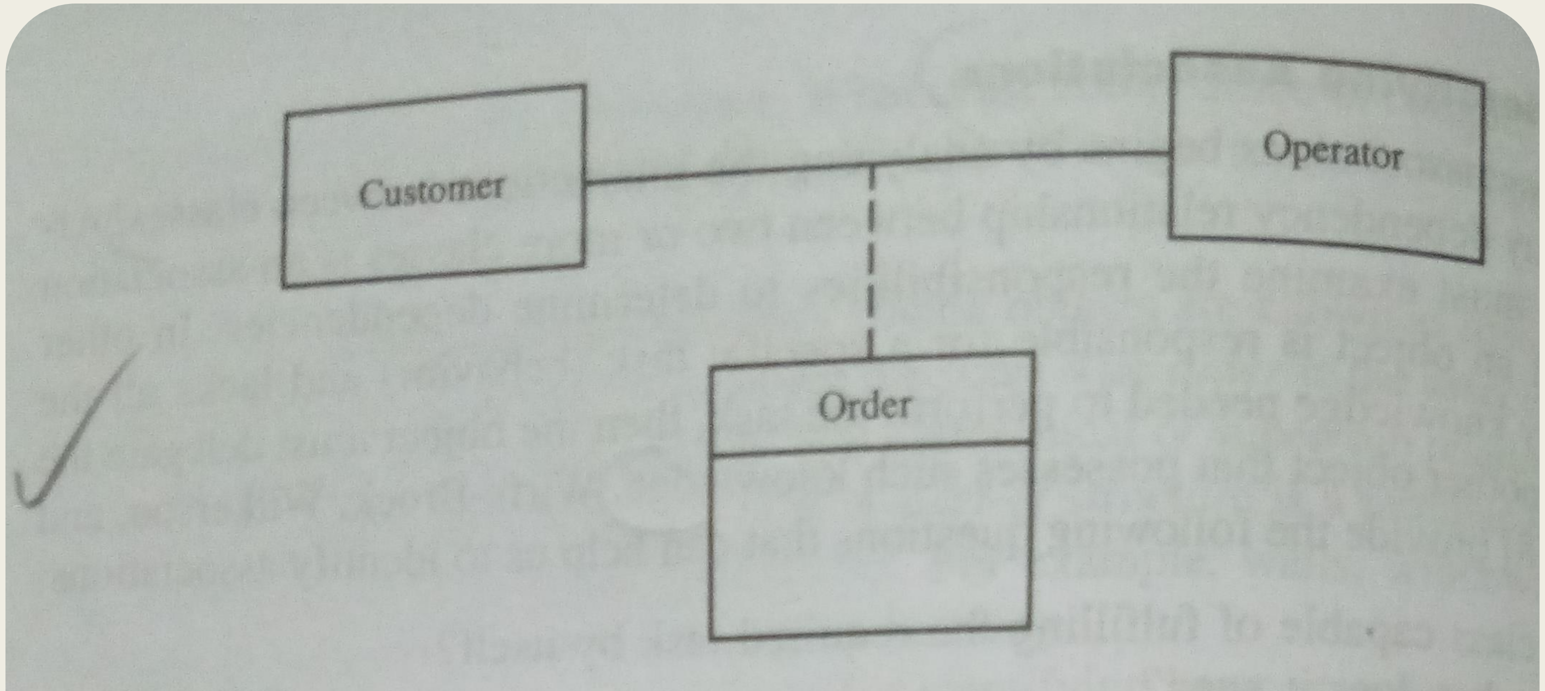


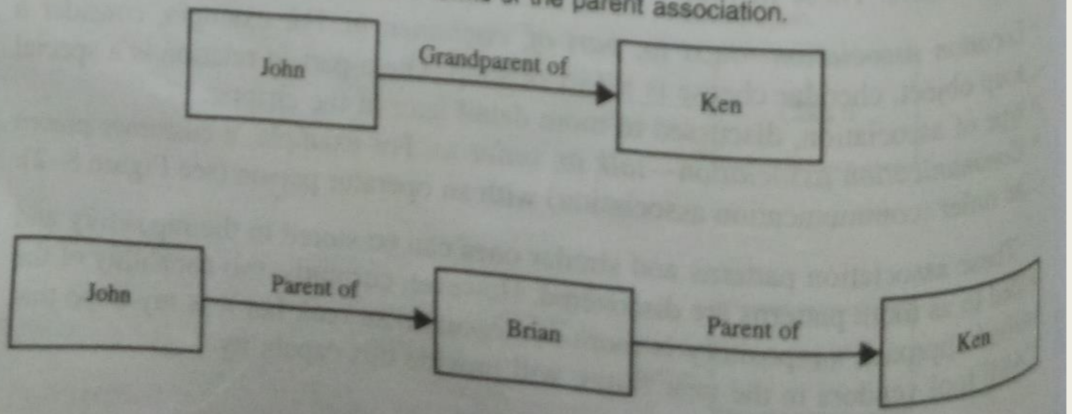
FIGURE 8–2
A customer places an order (communication association) with an operator person.

Eliminate Unnecessary Associations.

- Implementation association. Defer implementation Specific associations to the design phase.
- Ternary associations. Ternary or n-ary association among more than two classes.
- Ternary associations complicate the represent rations. when possible, restate ternary associations as binary associations.
- Directed actions for derived) association. Directed actions (derived) associations can be defined in terms of other associations.
- Since they are redundant avoid these types of association. For example, Grandparent of can be defined in terms of the parent of association.

FIGURE 8–3

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



Super-Sub class Relationships

- The super-sub class relationship represents the inheritance relationships between related classes, and the class hierarchy determines the lines of inheritance between classes.
- superclass - subclass relationships, also known as generalization hierarchy, allow objects to be built from other objects.

<u>Guidelines for Identifying super sub Relationship, a Generalization.</u>

- TOP_down: Look for noun phrases composed of various adjectives in a class name.
- Bottom -up: Look for classes with similar attributes or methods.
- Reusability: Move attributes and behaviours (Methods) as high as possible in the hierarchy.
- Multiple inheritance: Avoid excessive use of multiple inheritance.

<u>A-part-of Relationships_ Aggregation:</u>

- A-part-of relationship, also called aggregation, represents the situation where a class consists of several component classes.
- Two major properties of a-part-of-relationship are transitivity and antisymmetry.
- **Transitivity**: The property where, if A is part of B and B is part of C, then A is part of C.
- **Antisymmetry**: The property of a-part. of relation where, if A is part of B, then B is not part of A.

- For example, an engine is part of a car, but a car is not part of an engine.
- A clear distinctions between the part and the whole can help us determine where responsibilities for certain behaviour must reside.
- A Filled diamond signifies the strong form of aggregation, which is composition.
- For example, one might represent aggregation such as container and collection as hollow diamonds and use a solid diamond to represent composition, which is a strong from for aggregation.

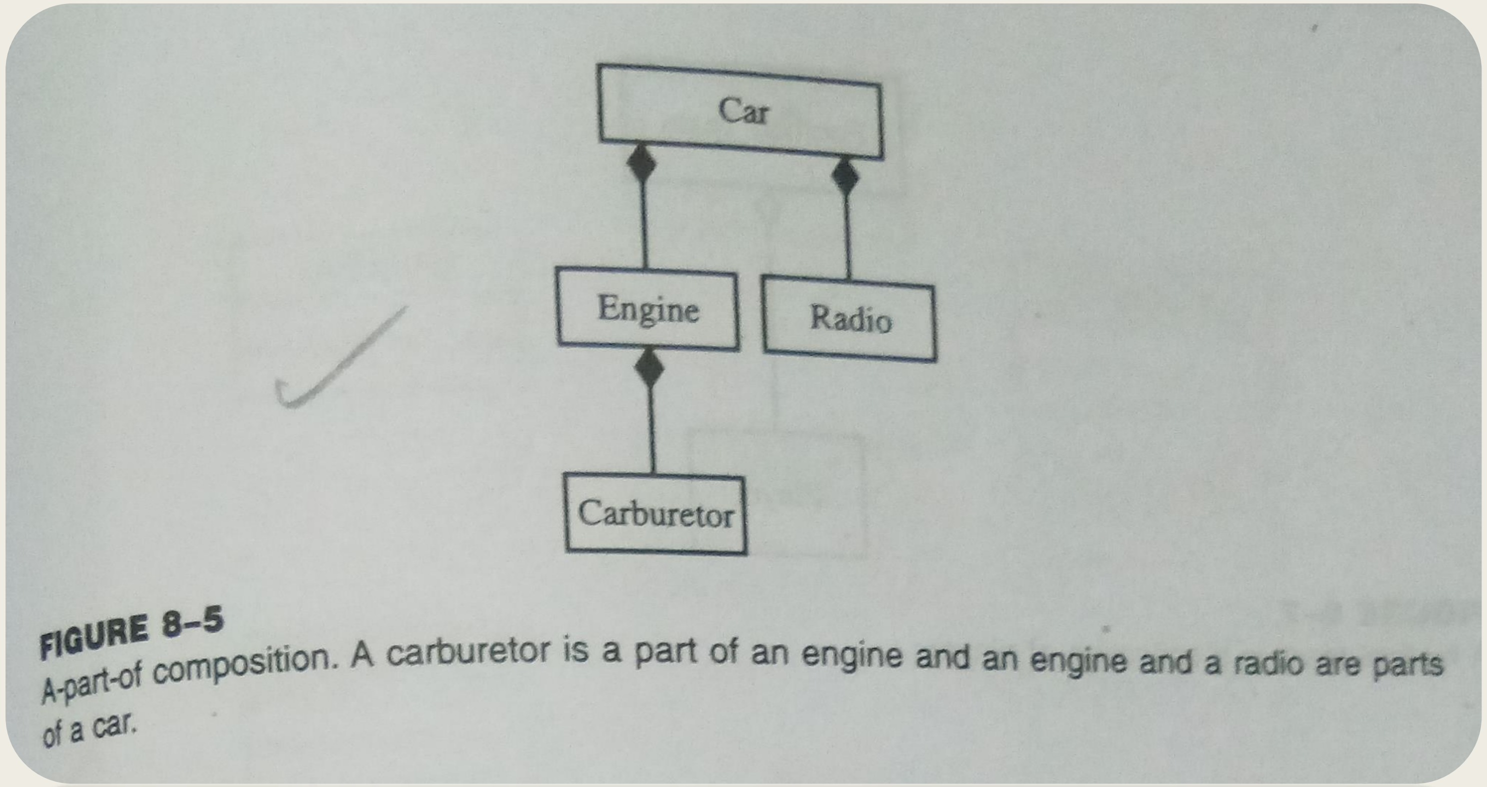
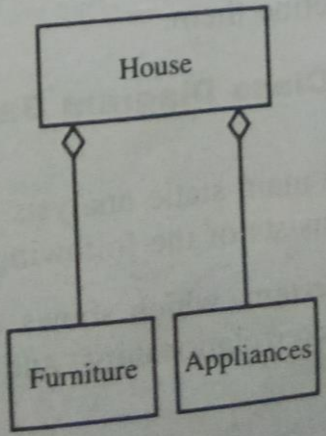


FIGURE 8–5
A-part-of composition. A carburetor is a part of an engine and an engine and a radio are parts of a car.

A-part of Relationship patterns.

- Assembly:- An assembly is constructed from its parts and an assembly- part situation physically exists;
- For example, a French onion soup is an assembly of onion, butter, flour, wine, French bread, cheddar cheese, and so on.
- Container: A Physical whole encompasses but is not constructed from physical parts;
- For example, a house can be considered as a container for furniture and appliances [Fig 8-6]
- Collection- member: A conceptual whole encompasses parts that may be physical or conceptual
- For example, a football team is a collection of players [Fig 8-7].

FIGURE 8–6
A house is a container.



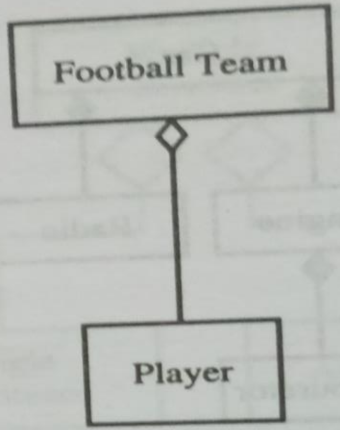


FIGURE 8–7
A football team is a collection of players.