ICS 3105
OBJECT ORIENTED SOFTWARE ENGINEERING

CHAPTER 3

Introduction to Object Orientation

Learning Outcomes

- By the end of this chapter, the learner should be able to:
  - Describe object orientation.
  - Discuss object oriented concepts.
  - Identify objects, attributes and operations from a narrative.

Introduction

- We live in a world of objects.
- These objects exist in nature, in human made entities, in business, and in the products that we use.
- They can be categorized, described, organized, combined, manipulated, and created.

Introduction

- Therefore, it is no surprise that an object-oriented view would be proposed for the creation of computer software—an abstraction that enables us to model the world in ways that help us to better understand and navigate it.

Object Orientation

- An object-oriented approach to the development of software was first proposed in the late 1960s.
- However, it took almost 20 years for object technologies to become widely used.

Object Orientation

- Throughout the 1990s, object-oriented software engineering became the paradigm of choice for many software product builders and a growing number of information systems and engineering professionals.
- As time passes, object technologies are replacing classical software development approaches.

Object Orientation

- Object technologies do lead to a number of inherent benefits that provide advantage at both the management and technical levels.
- There are many ways to look at a problem to be solved using a software-based solution.

Object Orientation

- One widely used approach to problem solving takes an object-oriented viewpoint.
- The problem domain is characterized as a set of objects that have specific attributes and behaviors.

Object Orientation

- The objects are manipulated with a collection of functions (called methods, operations, or services) and communicate with one another through a messaging protocol.
- Objects are categorized into classes and subclasses.

Object Orientation

- Object technologies lead to reuse, and reuse (of program components) leads to faster software development and higher-quality programs.
- Object-oriented software is easier to maintain because its structure is inherently decoupled.

Object Orientation

- This leads to fewer side effects when changes have to be made and less frustration for the software engineer and the customer.
- In addition, object-oriented systems are easier to adapt and easier to scale (i.e., large systems can be created by assembling reusable subsystems).

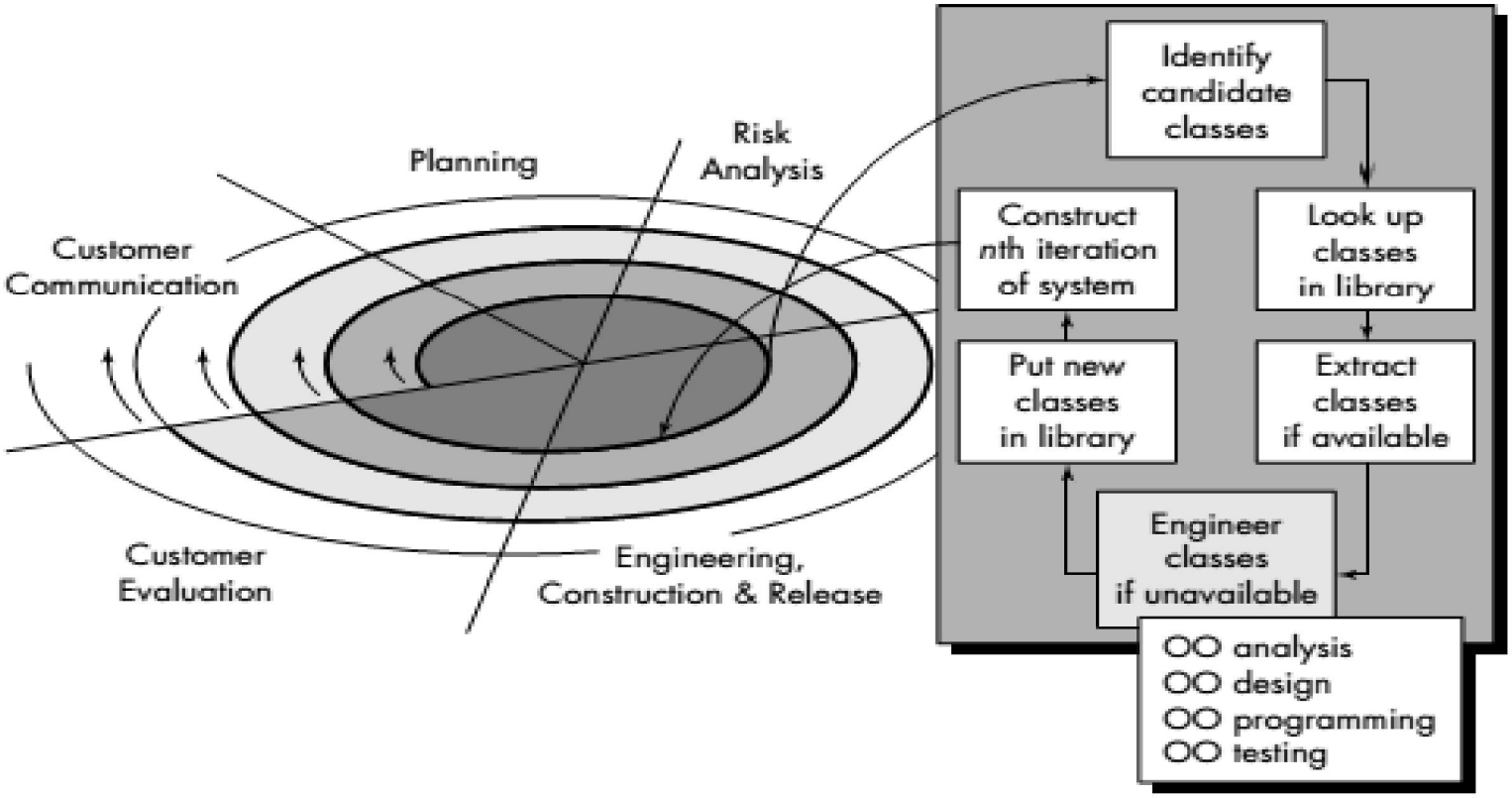
Object Oriented Paradigm

- For many years, the term object oriented (OO) was used to denote a software development approach that used one of a number of object-oriented programming languages.

Object Oriented Paradigm

- Today, the OO paradigm encompasses a complete view of software engineering.

The OO process model



The OO process model

- The OO process moves through an evolutionary spiral that starts with customer communication.
- It is here that the problem domain is defined and that basic problem classes are identified.

The OO process model

- Planning and risk analysis establish a foundation for the OO project plan.
- The technical work associated with OO software engineering follows the iterative path shown in the shaded box.

The OO process model

- OO software engineering emphasizes reuse.
- Therefore, classes are “looked up” in a library (of existing OO classes) before they are built.

The OO process model

- When a class cannot be found in the library, the software engineer applies object-oriented analysis (OOA), object-oriented design (OOD), object-oriented programming (OOP), and object-oriented testing (OOT) to create the class and the objects derived from the class.

The OO process model

- The new class is then put into the library so that it may be reused in the future.
- The object-oriented view demands an evolutionary approach to software engineering.

The OO process model

- As we will see throughout this and the following chapters, it would be exceedingly difficult to define all necessary classes for a major system or product in a single iteration.

The OO process model

- As the OO analysis and design models evolve, the need for additional classes becomes apparent.
- It is for this reason that the paradigm just described works best for OO.

Object Oriented Concepts

- Any discussion of object-oriented software engineering must begin by addressing the term object-oriented.

Objects and Classes

- The fundamental concepts that lead to high-quality design apply equally to systems developed using conventional and object-oriented methods.
- For this reason, an OO model of computer software must exhibit data and procedural abstractions that lead to effective modularity.

Objects and Classes

- A class is an OO concept that encapsulates the data and procedural abstractions required to describe the content and behavior of some real world entity.

Objects and Classes

- The data abstractions (attributes) that describe the class are enclosed by a “wall” of procedural abstractions (called operations, methods, or services) that are capable of manipulating the data in some way.

Encapsulation and information Hiding

- The only way to reach the attributes (and operate on them) is to go through one of the methods that form the wall.
- Therefore, the class encapsulates data (inside the wall) and the processing that manipulates the data (the methods that make up the wall).

Encapsulation and information Hiding

- This achieves information hiding and reduces the impact of side effects associated with change.
- Since the methods tend to manipulate a limited number of attributes, they are cohesive; and because communication occurs only through the methods that make up the “wall,” the class tends to be decoupled from other elements of a system.

Encapsulation and information Hiding

- All of these design characteristics lead to high quality software.
- Stated another way, a class is a generalized description (e.g., a template, pattern, or blueprint) that describes a collection of similar objects.

Inheritance

- By definition, all objects that exist within a class inherit its attributes and the operations that are available to manipulate the attributes.
- A superclass is a collection of classes, and a subclass is a specialized instance of a class.

Inheritance

- These definitions imply the existence of a class hierarchy in which the attributes and operations of the superclass are inherited by subclasses that may each add additional “private” attributes and methods.

Attributes

- Attributes are attached to classes and objects, and they describe the class or object in some way.
- Real life entities are often described with words that indicate stable features.
- Most physical objects have features such as shape, weight, color, and type of material.

Attributes

- People have features including date of birth, parents, name, and eye color.
- A feature may be seen as a binary relation between a class and a certain domain.
- The “binary relation” implies that an attribute can take on a value defined by an enumerated domain.

Attributes

- In most cases, a domain is simply a set of specific values.
- For example, assume that a class automobile has an attribute color.
- The domain of values for color is {white, black, silver, gray, blue, red, yellow, green}.

Attributes

- In more complex situations, the domain can be a set of classes.
- The features (values of the domain) can be augmented by assigning a default value (feature) to an attribute.

Operations, Methods, and Services

- An object encapsulates data (represented as a collection of attributes) and the algorithms that process the data.
- These algorithms are called operations, methods, or services and can be viewed as modules in a conventional sense.

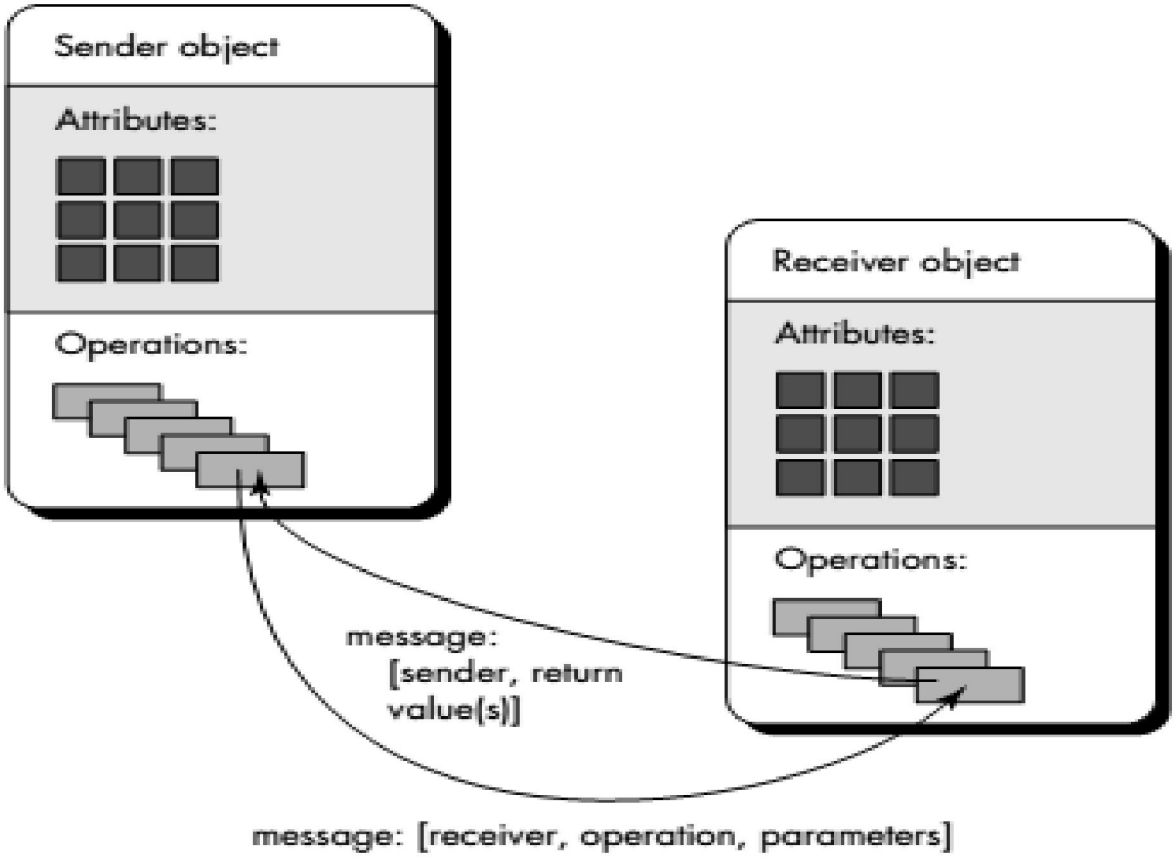
Operations, Methods, and Services

- Each of the operations that is encapsulated by an object provides a representation of one of the behaviors of the object.
- For example, the operation GetColor for the object automobile will extract the color stored in the color attribute.

Messages

- Messages are the means by which objects interact.
- Using the terminology introduced in the preceding section, a message stimulates some behavior to occur in the receiving object.
- The behavior is accomplished when an operation is executed.

Messages


Sender object
Attributes:
Operations:
Receiver object
Attributes:
Operations:
message:
[sender, return
value(s)]
message: [receiver, operation, parameters]

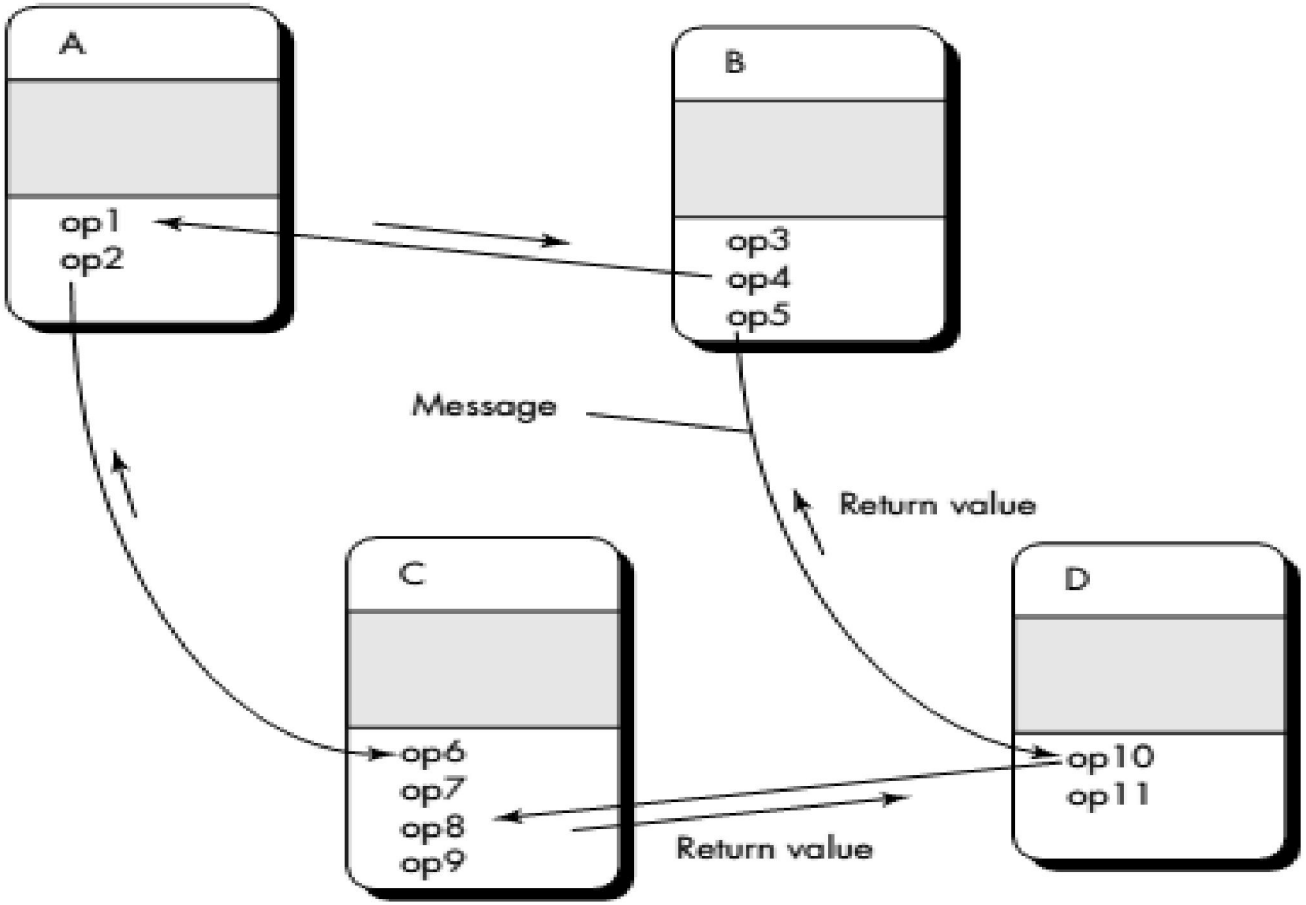
Messages

- An operation within a sender object generates a message of the form Message: [destination, operation, parameters] where
  - Destination defines the receiver object that is stimulated by the message,
  - Operation refers to the operation that is to receive the message, and
  - Parameters provides information that is required for the operation to be successful.

Messages

- An object is requested to perform one of its operations by sending it a message telling the object what to do.
- The receiver [object] responds to the message by first choosing the operation that implements the message name, executing this operation, and then returning control to the caller.

Messages


A
op1
op2
B
op3
op4
op5
Message
Return value
C
op6
op7
op8
op9
D
op10
op11
Return value

Messages

- Messaging ties an object-oriented system together.
- Messages provide insight into the behavior of individual objects and the OO system as a whole.

Encapsulation, Inheritance, and Polymorphism

- Three characteristics of object-oriented systems make them unique.
- The OO class and the objects spawned from the class encapsulate data and the operations that work on the data in a single package.

Encapsulation, Inheritance, and Polymorphism

- This provides a number of important benefits:

1. The internal implementation details of data and procedures are hidden from the outside world (information hiding). This reduces the propagation of side effects when changes occur.

Encapsulation, Inheritance, and Polymorphism

2. Data structures and the operations that manipulate them are merged in a single named entity—the class. This facilitates component reuse.
3. Interfaces among encapsulated objects are simplified. An object that sends a message need not be concerned with the details of internal data structures. Hence, interfacing is simplified and the system coupling tends to be reduced.

Encapsulation, Inheritance, and Polymorphism

- Inheritance is one of the key differentiators between conventional and OO systems.
- A subclass Y inherits all of the attributes and operations associated with its superclass, X.

Encapsulation, Inheritance, and Polymorphism

- This means that all data structures and algorithms originally designed and implemented for X are immediately available for Y—no further work need be done.
- Reuse has been accomplished directly.

Encapsulation, Inheritance, and Polymorphism

- Any change to the data or operations contained within a super class is immediately inherited by all subclasses that have inherited from the super class.
- Therefore, the class hierarchy becomes a mechanism through which changes (at high levels) can be immediately propagated through a system.

Encapsulation, Inheritance, and Polymorphism

- At each level of the class hierarchy, new attributes and operations may be added to those that have been inherited from higher levels in the hierarchy.

Encapsulation, Inheritance, and Polymorphism

- Whenever a new class is to be created, the software engineer has a number of options:
  - The class can be designed and built from scratch. That is, inheritance is not used.
  - The class hierarchy can be searched to determine if a class higher in the hierarchy contains most of the required attributes and operations. The new class inherits from the higher class and additions may then be added, as required.

Encapsulation, Inheritance, and Polymorphism

- The class hierarchy can be restructured so that the required attributes and operations can be inherited by the new class.
- Characteristics of an existing class can be overridden and private versions of attributes or operations are implemented for the new class.

Encapsulation, Inheritance, and Polymorphism

- It is important to note that restructuring the hierarchy can be difficult, and for this reason, overriding is sometimes used.
- In essence, overriding occurs when attributes and operations are inherited in the normal manner but are then modified to the specific needs of the new class.

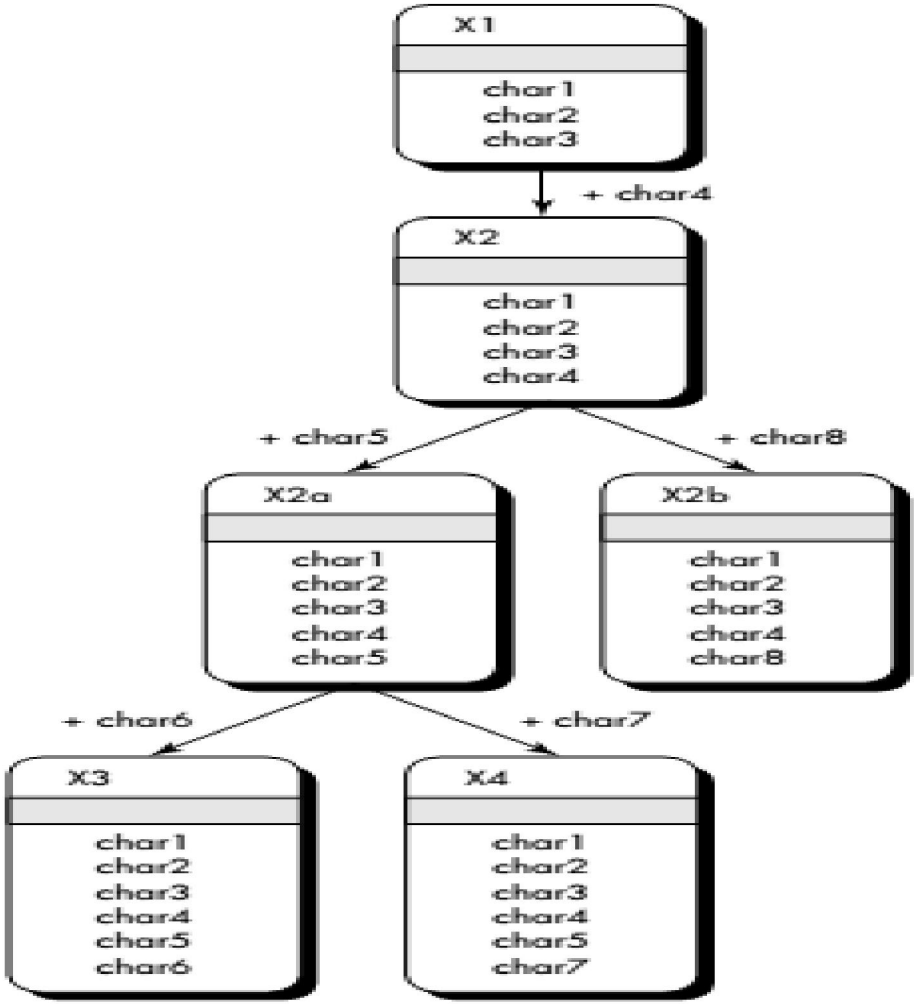
Encapsulation, Inheritance, and Polymorphism

- In some cases, it is tempting to inherit some attributes and operations from one class and others from another class.
- This is called multiple inheritance, and it is controversial.

Encapsulation, Inheritance, and Polymorphism

- In general, multiple inheritance complicates the class hierarchy and creates potential problems in configuration control
- Because multiple inheritance sequences are more difficult to trace, changes to the definition of a class that resides high in the hierarchy may have an unintended impact on classes defined lower in the architecture.

Encapsulation, Inheritance, and Polymorphism



Polymorphism

- Polymorphism is a characteristic that greatly reduces the effort required to extend an existing OO system.
- To understand polymorphism, consider a conventional application that must draw four different types of graphs: line graphs, pie charts, histograms

Polymorphism

- Ideally, once data are collected for a particular type of graph, the graph should draw itself.
- To accomplish this in a conventional application (and maintain module cohesion), it would be necessary to develop drawing modules for each type of graph.

Identifying Classes and Objects

- We can begin to identify objects by examining the problem statement or by performing a "grammatical parse" on the processing narrative for the system to be built.
- Objects are determined by underlining each noun or noun clause and entering it in a simple table.

Identifying Classes and Objects

- Synonyms should be noted.
- If the object is required to implement a solution, then it is part of the solution space; otherwise, if an object is necessary only to describe a solution, it is part of the problem space.

Identifying Classes and Objects

- Objects manifest themselves in one of the ways:
  - External entities (e.g., other systems, devices, people) that produce or consume information to be used by a computer-based system.
  - Things (e.g, reports, displays, letters, signals) that are part of the information domain for the problem.

Identifying Classes and Objects

- Occurrences or events (e.g., a property transfer or the completion of a series of robot movements) that occur within the context of system operation.
- Roles(e.g., manager, engineer, salesperson) played by people who interact with the system.
- Organizational units(e.g., division, group, team) that are relevant to an application.

Identifying Classes and Objects

- Places(e.g., manufacturing floor or loading dock) that establish the context of the problem and the overall function of the system.
- Structures(e.g., sensors, four-wheeled vehicles, or computers) that define a class of objects or in the extreme, related classes of objects.

Identifying Classes and Objects

- Coad and Yourdon suggest six selection characteristics that should be used as an analyst considers each potential object for inclusion in the analysis model:
  - 1. Retained information.
  - 2. Needed services.
  - 3. Multiple attributes.
  - 4. Common attributes.
  - 5. Common operations.
  - 6. Essential requirements.

Retained information

- The potential object will be useful during analysis only if information about it must be remembered so that the system can function.

Needed Services

- The potential object must have a set of identifiable operations that can change the value of its attributes in some way.

Multiple attributes

- During requirement analysis, the focus should be on "major" information; an object with a single attribute may, in fact, be useful during design, but is probably better represented as an attribute of another object during the analysis activity.

Common attributes

- A set of attributes can be defined for the potential object and these attributes apply to all occurrences of the object.

Common operations

- A set of operations can be defined for the potential object and these operations apply to all occurrences of the object.

Essential requirements

- External entities that appear in the problem space and produce or consume information essential to the operation of any solution for the system will almost always be defined as objects in the requirements model.

Identifying Classes and Objects

- To be considered a legitimate object for inclusion in the requirements model, a potential object should satisfy all (or almost all) of these characteristics.

Identifying Classes and Objects

- The decision for inclusion of potential objects in the analysis model is somewhat subjective, and later evaluation may cause an object to be discarded or reinstated.
- However, the first step of OOA must be a definition of objects, and decisions (even subjective ones) must be made.

Specifying Attributes

- Attributes describe an object that has been selected for inclusion in the analysis model.
- In essence, it is the attributes that define the object—that clarify what is meant by the object in the context of the problem space.

Specifying Attributes

- For example, if we were to build a system that tracks football statistics for professional football players, the attributes of the object player would be quite different than the attributes of the same object when it is used in the context of the professional football pension system.

Specifying Attributes

- In the former, attributes such as name, position, goals average, fielding percentage, years played, and games played might be relevant.

Specifying Attributes

- For the latter, some of these attributes would be meaningful, but others would be replaced (or augmented) by attributes like average salary, credit toward full vesting, pension plan options chosen, mailing address, and the like.

Specifying Attributes

- To develop a meaningful set of attributes for an object, the analyst can again study the processing narrative (or statement of scope) for the problem and select those things that reasonably "belong" to the object.

Specifying Attributes

- In addition, the following question should be answered for each object:
  - What data items (composite and/or elementary) fully define this object in the context of the problem at hand?

Defining Operations

- Operations define the behavior of an object and change the object's attributes in some way.
- More specifically, an operation changes one or more attribute values that are contained within the object.

Defining Operations

- Therefore, an operation must have "knowledge" of the nature of the object's attributes and must be implemented in a manner that enables it to manipulate the data structures that have been derived from the attributes.

Defining Operations

- Although many different types of operations exist, they can generally be divided into three broad categories:
  - Operations that manipulate data in some way (e.g., adding, deleting, reformatting, selecting),
  - Operations that perform a computation, and
  - Operations that monitor an object for the occurrence of a controlling event.

Defining Operations

- As a first iteration at deriving a set of operations for the objects of the analysis model, the analyst can again study the processing narrative (or statement of scope) for the problem and select those operations that reasonably belong to the object.

Defining Operations

- To accomplish this, the grammatical parse is again studied and verbs are isolated.
- Some of these verbs will be legitimate operations and can be easily connected to a specific object.

End of Chapter 3

CAT 1

Date: 8th November 2015

Scope: Chapter 1, 2 & 3

Time: 8 am