

Concepts and Principles

Chapter 1

Objectives

You will learn:

- the difference between analysis and design.
- the difference between functional and OO approach to design.
- OO terminology.
- how to define an object and its characteristics.
- the tools of OOD, OOA and OO programming.

Objectives

You will learn:

- how objects interact.
- to appreciate relationships between objects.
- the concepts of actors, servers and agents.
- inheritance, aggregation, association and instantiation.

Object-Oriented Design

- Structured or functional design is derived from the functioning of a system.
- O-O design is different; it considers design from the point of view of:
 - interacting objects of various types.
 - information hiding.

Objects

- An object is defined as an entity which has:
 - a private state.
 - a set of operations to manipulate that state.
- An object is an instance of a type; the type specifies the valid set of operations available for each object of that type.
- The objects provide abstract behavior; the detailed implementation is hidden from the user.
- The state of the object may be accessed only by the defined operations.

Deriving an O-O Design

- Consists of conceptualizing/planning:
 - The types of object that may be needed in a system.
 - The ways that instances of those types interact.
- Start with a natural language overview of the system; then identify:
 - nouns = objects or their attributes.
 - verbs = operations.
 - adjectives = subclasses.

Object Components

- The objects consist of:
 - data attributes
 - (e.g. an array or variable).
 - operations
 - (e.g. functions) which can be applied to the data.
- These 2 components are sometimes known as:
 - data members.
 - member functions.
- The only disadvantage to O-O design is that it is difficult to identify those objects most suitable to providing the best design.

SYS-ED/Computer Education Techniques, Inc.

1: 7

Example

A natural language overview which is part of the e-mail system:

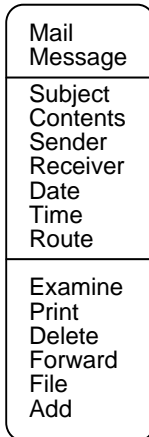
Messages in a mail box can be examined, printed, deleted, forwarded or filed, and new messages can be added.

A mail message consists of a subject line and the contents of the message, plus the sender's name, the receiver's name, the date and time sent, and the (circuitous) route it took.

SYS-ED/Computer Education Techniques, Inc.

1: 8

Graphical Representation of Objects



- Objects can be represented as round-cornered rectangles.
- Attributes or data members.
- Operations or member functions.

Inheritance

- Is an important feature in O-O systems, providing significant power.
- An object is an instance or concrete example of a type or class.
- Similar objects belong to the same class.
- Classes can be formed into a hierarchy; with subclasses inheriting all the attributes and operations of their superclass(es); as well as having additional ones of their own.

Advantages of O-O Design

- Security
 - Information is hidden rather than shared by the system.
 - Global variables and shared data structures are avoided; reducing coupling and the possibility of changes to data by different parts of the system.
 - This helps to make the system more understandable.
 - Access to data is only possible through the object's interface.
- Abstraction
 - Implementation details are hidden, making the objects:
 - independent, with high cohesion.
 - understandable, in bite-size chunks.
 - easy to maintain, changes are confined to a small area.
 - reusable.

Coupling

- Coupling or dependency is the degree to which each program module relies on each one of the other modules.
- Coupling is usually contrasted with cohesion. Low coupling often correlates with high cohesion, and vice versa.
- Low coupling refers to a relationship in which one module interacts with another module through a stable interface and does not need to be concerned with the other module's internal implementation.
- With low coupling, a change in one module will not require a change in the implementation of another module. Low coupling is often a sign of a well-structured computer system, and when combined with high cohesion, supports the general goals of high readability and maintainability.
- Systems that do not exhibit low coupling might experience the following difficulties when performing application development:
 - Change in one module force a ripple of changes in other module.
 - Modules are difficult to understand in isolation.
 - Modules are difficult to reuse or test because dependent modules must be included.

Cohesion

- Cohesion is a measure of how well the lines of source code within a module work together to provide a specific piece of functionality.
- Cohesion is an ordinal type of measurement and is usually expressed as "high cohesion" or "low cohesion" when being discussed.
- Modules with high cohesion tend to be preferable because they are associated with robustness, reliability, reusability, and ready comprehension.
- Low cohesion is associated with traits such as being difficult to maintain, difficult to test, difficult to reuse, and difficult to understand.

O-O Languages and Tools

- O-O languages make the implementation of O-O designs easy: e.g. using C++ classes, Ada packages, Modular-2 modules, Java classes.
- VB has many of the features of an object oriented language.
- Non-O-O languages such as C or Pascal can be used to implement an O-O design.
- Tools are now available for O-O.

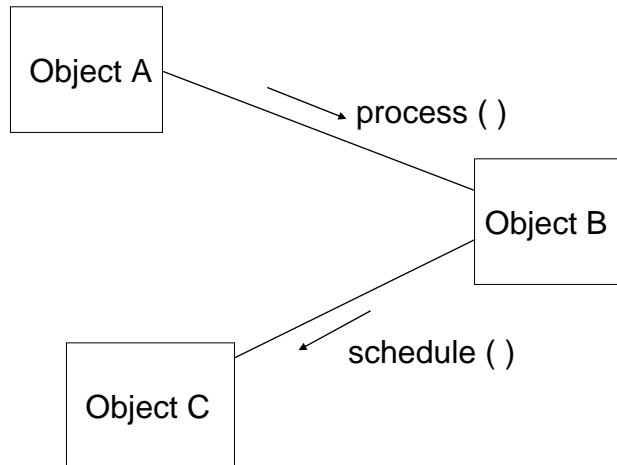
Object Interaction

- The objects in a system are not independent; they interact with each other.
- Deciding on the classes, with their attributes and operations, is insufficient for design purposes.
- It will also be necessary to identify how different classes interact with, or are related to, each other.
- This can be done in two ways:
 - by communicating with each other.
 - by having some sort of relationship.

Communicating Objects

- Objects communicate with each other via message passing.
- To send a message from object A to object B, it will be necessary to invoke one of object B's operations from object A.
- The message may include information being sent to object A, and/or it may bring back a reply or return value.
- Design documentation should incorporate message passing between objects.
- Object diagrams should be drawn showing communicational links between the different objects of the system.

Object Diagram



SYS-ED/Computer Education Techniques, Inc.

1: 17

Actors, Servers, and Agents

- Actors, servers and agents are simple concepts, which are useful when drawing up object diagrams.
- Actors send messages to, but never receive messages from other objects.
- Servers never send messages to, only receive messages from other objects.
- Agents send and receive messages.

SYS-ED/Computer Education Techniques, Inc.

1: 18

Class Relationships

- There are 4 main relationships:
 - aggregation
 - association
 - inheritance
 - instantiation
- The cardinality of the relationships aggregation and association may be one-to-one, one-to many, or many-to-many.

Inheritance

- A generalization / specialization relationship exists between classes.
- Useful phrase - "is a type of" , "extends" or "is a".
 - For example: Part-time student is a student.
- A part-time student is a specialized subclass of the more general class student.
- A part-time student inherits all the attributes and operations of the superclass student.
- Classes may be formed into a hierarchy.

Inheritance: Advanced Concepts

Multiple inheritance

- A class may inherit from more than one superclass.
 - e.g. hovercraft, inheriting from boat and plane.

Polymorphism (many forms)

- A subclass may alter inherited attributes or operations.
 - e.g. line graph and bar graph both inherit from graph which has an operation draw, but both have their own variations of the operation.

Abstract classes

- Have no instances e.g. vehicle.
- Are usually towards the top of an inheritance hierarchy.

Aggregation

- A whole/part relationship between classes.
- Useful phrase “part of”, and sometimes “has”
e.g. a page is a part of a book.
- Also known as container/component relationship.
- Cardinality:
 - one-to-one or one-to-many.

Association

- Less clear-cut than inheritance and aggregation.
- Denotes some semantic dependency between the classes (other than “is a type of” or “is a part of” or “has”).
- Often when one class is associated with another then one has a role to play with respect to the other.
- Association is also known as utilization or use, with one class acting as a server and the other a client.
- Often an association is changed into another more clear-cut relationship as a design is developed.

Instantiation

- Instantiation is unlike the previous relationships in that it is not a relationship between classes.
- Instantiation is a relationship between a class and an object.
- An object is an instance of a class.
- At any one time a class may have zero or more object instances.
- Classes can be thought of as static entities, with fixed properties: attributes, operations, a name, an interface.
- Conversely, objects may be thought of as dynamic entities with changing properties - the values of their attributes, the status of execution of their operations.