Object Oriented Analysis and Design: An Overview
Balaji Rajagopalan

Credits: Material for the slides is drawn from a variety of sources including Object Oriented Analysis and Design using UML by Ali Bahrami.
What is Object Orientation (OO)?

- Traditional (structured) approaches viewed software along two orthogonal axes: Algorithms + Data Structures
- The OO centers on the object which combines the two orthogonal views
Why OO?

• Higher *level of abstraction*

• *Seamless transition* among different phases of software development

• Encouragement of good programming (Class carefully delineates *interface* - *what the class can do* and *implementation of that interface* - *How the class does what it does*)

• *Promotes reusability*

Note that OO analysis and design does not make OO programming language a requirement for implementation
The result of using the OO approach is…..

• Object-Oriented (OO) systems development is a way to develop software by building self-contained modules that can be more easily:
  • Replaced
  • Modified
  • and Reused.
1. Identify the users/actors (Chapter 6): Who is (or will be) using the system?

2. Develop a simple business process model

The advantage of developing a business process model is that it familiarizes you with the system and therefore the user requirements.

3. Develop the use case (Chapter 6): What are (or will be) the users doing with the system? Use cases provide comprehensive documentation of the system under study.

Use cases capture the goal of the users and the responsibility of the system to its users.

4. Interaction diagrams (Chapter 7)

4.1 Develop sequence diagrams

4.2 Develop collaboration diagrams.

4.3 Iterate and refine

5. Classification (Chapter 8)

5.1 Identify Classes

5.2 Identify Relationships

5.3 Identify Attributes

5.4 Identify Methods

5.5 Iterate and refine.

The process of creating sequence or collaboration diagrams is a systematic way to think about how a use case can take place, and doing so, it forces you to think about objects involves in your application.

6. Apply design axioms to design classes, their attributes, methods, associations, structures, and protocols (Chapter 9)

6.1. Refine and complete the static UML class diagram (object model) by adding details to the UML class diagram (Chapter 10)

6.1.1 Refine attributes

6.1.2 Design methods and protocols by utilizing UML activity diagram for representation of method's algorithm

6.1.3 Refine (if required) associations between classes

6.1.4 Refine (if required) class hierarchy and design with inheritance

6.2 Iterate and refine (reapply Design axioms).

7.0 Design the access layer (Chapter 11)

7.1. Create access layer classes by mirroring the business classes

7.2. Define relationships

7.3. Simplify classes and structures

7.3.1 Eliminate redundant classes

7.3.2 Eliminate method classes

7.4 Iterate and refine

8. Designing view layer classes (Chapter 12)

8.1 Macro-level UI design Process - Identifying View layer Objects

8.2 Micro-level UI design activities:

8.2.1 Designing the view layer objects by applying design axioms and corollaries

8.2.2 Prototyping the view layer interface.

8.3. Usability and user satisfaction testing (Chapter 14):

8.4 Iterate and refine

9. Iterate and refine the design/analysis: If needed repeat the preceding steps
Object Basics

• 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,
• Understand meta-classes.
What is an object?

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

• An object is an entity.
  – It knows things (has attributes)
  – It does things (provides services or has methods)
Attributes (Contd.)

I am a Car.
I know my color, manufacturer, cost, owner and model.
Attributes (Contd.)

I am a Fish.
I know my date of arrival and expiration.
It does things (methods)

I know how to compute my payroll.
Methods (Contd.)

I know how to stop.
What is an object? (Contd.)

• Attributes or properties describe object’s state (data) and methods define its behavior.

• Object is whatever an application wants to talk about.
  – For example, Parts and assemblies might be objects of bill of material application.
  
  – Stocks and bonds might be objects of financial investment applications.
Objects

- 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.
  - A chart object is responsible for things like maintaining its data and labels, and even for drawing itself.
  - A window object is responsible for things like opening, sizing, and closing itself.
Two Basic Questions

• When developing an O-O application, two basic questions always arise.
  – What objects does the application need?
  – What functionality should those objects have?
Object’s Attributes and Methods

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

• Methods
  – define objects behavior 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.
Objects are Grouped in Classes

• The role of a class is to define the attributes and methods (the state and behavior) of its instances.

• 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
A Class is an Object Template, or an Object Factory.
Class Hierarchy

• An object-oriented system organizes classes into subclass-super hierarchy.

• At the top of the hierarchy are the most general classes and at the bottom are the most specific
A subclass inherits all of the properties and methods (procedures) defined in its superclass.
Inheritance

(programming by extension)

• Inheritance is a relationship between classes where one class is the parent class of another (derived) class.

• Inheritance allows classes to share and reuse behaviors and attributes.

• The real advantage of inheritance is that we can build upon what we already have and,

• Reuse what we already have.
Inheritance (Contd.)

I know how to stop
stop method is reusable

I don’t know how to stop
stop (myMustang)
Multiple Inheritance

• OO systems permit a class to inherit from more than one superclass.

• This kind of inheritance is referred to as multiple inheritance.
Multiple Inheritance (Contd.)

- For example utility vehicle inherits from Car and Truck classes.
Encapsulation and Information Hiding

- Information hiding is a principle of hiding internal data and procedures of an object.
Encapsulation and Information Hiding (Contd.)

• By providing an interface to each object in such a way as to reveal as little as possible about its inner workings.

• Encapsulation protects the data from corruption.
Protocol

- Protocol is an interface to the object.
- TV contains many complex components, but you do not need to know about them to use it.
Message

• Objects perform operations in response to messages.

• For example, you may communicate with your computer by sending it a message from hand-held controller.
A Case Study - A Payroll Program

• Consider a payroll program that processes employee records at a small manufacturing firm. This company has three types of employees:

  • 2. *Office Workers*: Receive an hourly wage and are eligible for overtime after 40 hours.
  • 3. *Production Workers*: Are paid according to a piece rate.
Structured Approach

FOR EVERY EMPLOYEE DO

BEGIN

IF employee = manager THEN
    CALL computeManagerSalary

IF employee = office worker THEN
    CALL computeOfficeWorkerSalary

IF employee = production worker THEN
    CALL computeProductionWorkerSalary

END
What if we add two new types of employees?

• Temporary office workers ineligible for overtime,
• Junior production workers who receive an hourly wage plus a lower piece rate.
FOR EVERY EMPLOYEE DO
BEGIN
   IF employee = manager THEN
      CALL computeManagerSalary
   END
   IF employee = office worker THEN
      CALL computeOfficeWorker_salary
   END
   IF employee = production worker THEN
      CALL computeProductionWorker_salary
   END
   IF employee = temporary office worker THEN
      CALL computeTemporaryOfficeWorkerSalary
   END
   IF employee = junior production worker THEN
      CALL computeJuniorProductionWorkerSalary
   END
END
An Object-Oriented Approach

• What objects does the application need?
  – The goal of OO analysis is to identify objects and classes that support the problem domain and system's requirements.
  – Some general candidate classes are:
    – Persons
    – Places
    – Things
What are some of the application’s classes?

- *Employee*
- *Manager*
- *Office Workers*
- *Production Workers*
Class Hierarchy

• Identify class hierarchy
• Identify commonality among the classes
• Draw the general-specific class hierarchy.
Class Hierarchy (Contd.)

Employee
- name
- address
- salary
- SS#

OfficeWorker
- dataEntry
- ComputePayroll
- printReport

Manager
- dataEntry
- ComputePayroll
- printReport

ProductionWorker
- dataEntry
- ComputePayroll
- printReport
OO Approach

FOR EVERY EMPLOYEE DO
BEGIN
  employee computePayroll
END
If new types of employees were added…

Employee

- name
- address
- salary
- SS#

Office Worker

- dataEntry
- ComputePayroll
- printReport

Manager

- dataEntry
- ComputePayroll
- printReport

Production Worker

- dataEntry
- ComputePayroll
- printReport

Temporary Office Worker

- ComputePayroll

Junior Production Worker

- ComputePayroll
Polymorphism

• Polymorphism means that the same operation may behave differently on different classes.

• Example: *computePayroll*
Associations

• The concept of association represents relationships between objects and classes.
• For example a pilot can fly planes.
Associations (Contd.)

Pilot can fly flown by Planes
Clients and Servers

• 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).
Clients and Servers (Contd.)

PrintServer \[\text{Request for printing}\] Item
Objects and Persistence

- Objects have a lifetime.
- 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.
Summary

• Rather than treat data and procedures separately, object-oriented programming packages them into "objects."
• O-O system provides you with the set of objects that closely reflects the underlying application
• Advantages of object-oriented programming are:
  – The ability to reuse code,
  – develop more maintainable systems in a shorter amount of time.
  – more resilient to change, and
  – more reliable, since they are built from completely tested and debugged classes.