Chapter 1: Principles of Programming and Software Engineering

Data Abstraction & Problem Solving with C++

Fifth Edition

by Frank M. Carrano

Software Engineering and Object-Oriented Design

- Coding without a solution design increases debugging time
- A team of programmers for a large software development project requires
  - An overall plan
  - Organization
  - Communication
- Software engineering
  - Provides techniques to facilitate the development of computer programs

An Examination of Problem Solving

- Problem solving
  - The process of taking the statement of a problem and developing a computer program that solves that problem
- Object-oriented analysis and design (OOA / D)
  - A process for problem solving
  - A problem solution is a program consisting of a system of interacting classes of objects
    - Each object has characteristics and behaviors related to the solution
    - A class is a set of objects having the same type

Aspects of an Object-Oriented Solution

- A solution is a C++ program consisting of:
  - Modules
    - A single, stand-alone function
    - A method of a class
    - A class
    - Several functions or classes working closely together
    - Other blocks of code

Aspects of an Object-Oriented Solution

- Functions and methods implement algorithms
  - Algorithm: a step-by-step recipe for performing a task within a finite period of time
  - Algorithms often operate on a collection of data

Aspects of an Object-Oriented Solution

- Create a good set of modules
  - Modules must store, move, and alter data
  - Modules use algorithms to communicate with one another
- Organize your data collection to facilitate operations on the data

Abstraction and Information Hiding

- Abstraction
  - Separates the purpose of a module from its implementation
  - Specifications for each module are written before implementation
  - Functional abstraction
    - Separates the purpose of a module from its implementation

Abstraction and Information Hiding

  - Data abstraction
    - Focuses on the operations of data, not on the implementation of the operations
  - Abstract data type (ADT)
    - A collection of data and a set of operations on the data
    - You can use an ADT's operations without knowing their implementations or how data is stored, if you know the operations' specifications

Abstraction and Information Hiding

- Data structure
  - A construct that you can define within a programming language to store a collection of data
- Develop algorithms and ADTs in tandem

Abstraction and Information Hiding

- Information hiding
  - Hide details within a module
  - Ensure that no other module can tamper with these hidden details
  - Public view of a module
    - Described by its specifications
  - Private view of a module
    - Implementation details that the specifications should not describe

Principles of Object-Oriented Programming (OOP)

- Object-oriented languages enable us to build classes of objects
- A class combines
  - Attributes (characteristics) of objects of a single type
    - Typically data
    - Called data members
  - Behaviors (operations)
    - Typically operate on the data
    - Called methods or member functions

Principles of Object-Oriented Programming

- Three principles of object-oriented programming
  - Encapsulation
    - Objects combine data and operations
    - Hides inner details
  - Inheritance
    - Classes can inherit properties from other classes
    - Existing classes can be reused
  - Polymorphism
    - Objects can determine appropriate operations at execution time

Object-Oriented Analysis and Design

- Analysis
  - Process to develop
    - An understanding of the problem
    - The requirements of a solution
      - *What* a solution must be and do
      - Not *how* to design or implement it
  - Generates an accurate understanding of what end users will expect the solution to be and do
  - Think about the problem, not how to solve it

Object-Oriented Analysis and Design

- Object-oriented analysis (OOA)
  - Expresses an understanding of the problem and the requirements of a solution in terms of objects within the problem domain
  - Objects can represent
    - Real-world objects
    - Software systems
    - Ideas
  - OOA describes objects and their interactions among one another

Object-Oriented Analysis and Design

- Object-oriented design (OOD)
  - Expresses an understanding of a solution that fulfills the requirements discovered during OOA
  - Describes a solution in terms of
    - Software objects
    - The collaborations of these objects with one another
      - Objects collaborate when they send messages (call each other's operations)
      - Collaborations should be meaningful and minimal
  - Creates one or more models of a solution
    - Some emphasize interactions among objects
    - Others emphasize relationships among objects

Applying the UML to OOA/D

- Unified Modeling Language (UML)
  - A tool for exploration and communication during the design of a solution
  - Models a problem domain in terms of objects independently of a programming language
  - Visually represents object-oriented solutions as diagrams
  - Its visual nature is an advantage, since we are visual creatures
  - Enables members of a programming team to communicate visually with one another and gain a common understanding of the system being built

Applying the UML to OOA/D

- UML use case for OOA
  - A set of textual scenarios (stories) of the solution
    - Each scenario describes the system's behavior under certain circumstances from the perspective of the user
      - Focus on the responsibilities of the system to meeting a user's goals
    - Main success scenario (happy path): interaction between user and system when all goes well
    - Alternate scenarios: interaction between user and system under exceptional circumstances
  - Find noteworthy objects, attributes, and associations within the scenarios

Applying the UML to OOA/D

- An example of a main success scenario
  - Customer asks to withdraw money from a bank account
  - Bank identifies and authenticates customer
  - Bank gets account type, account number, and withdrawal amount from customer
  - Bank verifies that account balance is greater than withdrawal amount
  - Bank generates receipt for the transaction
  - Bank counts out the correct amount of money for customer
  - Customer leaves bank

Applying the UML to OOA/D

- An example of an alternate scenario
  - Customer asks to withdraw money from a bank account
  - Bank identifies, but fails to authenticate customer
  - Bank refuses to process the customer's request
  - Customer leaves bank

Applying the UML to OOA/D

- UML sequence (interaction) diagram for OOD
  - Models the scenarios in a use case
  - Shows the interactions among objects over time
  - Lets you visualize the messages sent among objects in a scenario and their order of occurrence
  - Helps to define the responsibilities of the objects
    - What must an object remember?
    - What must an object do for other objects?

Applying the UML to OOA/D

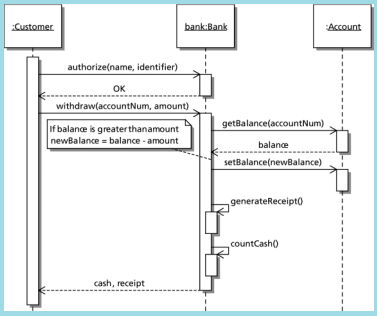

Figure 1-2 Sequence diagram for the main success scenario

Applying the UML to OOA/D

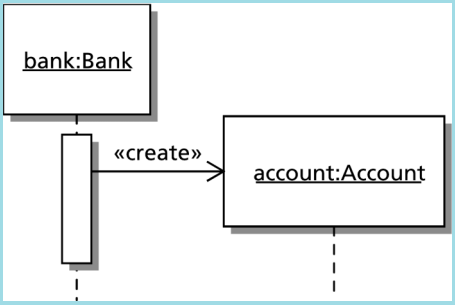

Figure 1-3 Sequence diagram showing the creation of a new object

Applying the UML to OOA/D

- UML class (static) diagram
  - Represents a conceptual model of a class of objects in a language-independent way
  - Shows the name, attributes, and operations of a class
  - Shows how multiple classes are related to one another

Applying the UML to OOA/D

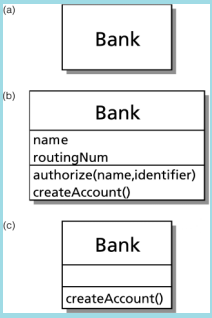

Figure 1-4 Three possible class diagrams for a class of banks

Applying the UML to OOA/D

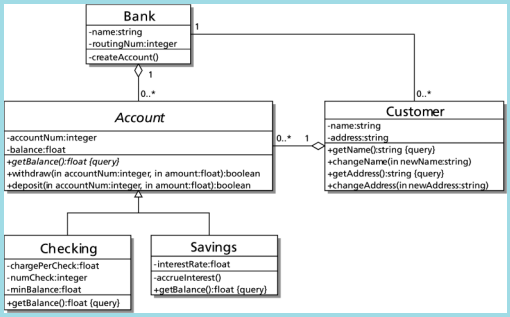

Figure 1-5 A UML class diagram of a banking system

Applying the UML to OOA/D

- Class relationships
  - Association
    - The classes know about each other
    - Example: The Bank and Customer classes
  - Aggregation (Containment)
    - One class contains an instance of another class
    - Example: The Bank and Account classes
    - The lifetime of the containing object and the object contained are not necessarily the same
      - Banks "live" longer than the accounts they contain