Object-Oriented Concepts

Problem Solving

Object-Oriented 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 of the solution**

Object-Oriented Analysis

- **Think about the problem,**
  - ***not how to solve it***
- **Objects can represent**
  - Real-world objects
  - Software systems
  - Ideas
- **OOA describes objects and their interactions with one another**

Expresses an understanding of the problem and the requirements of a solution in terms of objects within the problem domain

Pearson

Object-Oriented Design

Expresses an understanding of a solution that fulfills the requirements discovered during OOA

- **Describes a solution in terms of**
  - Software objects
  - Collaborations of these objects with one another

Pearson

Object Collaborations

- **Design Collaborations Among Objects**
  - Collaborations should have a specific purpose related to requirements
- **Design creates models of a solution**
  - Some emphasize ***interactions*** among objects
    - Objects interact when they send messages
      - Call each other's methods (member functions)
  - Others emphasize ***relationships*** among objects
    - **"is a"** and **"has a"** relationships

Pearson

Object Cohesion

- **Degree to which class member functions operate on data within the class.**
- ***High cohesion (good)***
  - Member functions only operate on class data
  - Easy to reuse in other software projects
  - Easy to revise or correct
  - Robust:
    - less likely to be affected by change;
    - performs well under unusual conditions
- **A person with low cohesion has “too many irons in the fire”**

Object Coupling

- **Degree to which a class depends on other classes**
- **Objects with low coupling are independent of one another**
- ***Low coupling (good)***
  - Easier to change:
    - Change to one module won't affect another
  - Easier to understand
  - Easier to reuse
  - Has increased cohesion
- **Coupling cannot be (and should not) be eliminated entirely**
- **UML Class diagrams show dependencies among classes, and so coupling**

Pearson

Object-Oriented Design

Object-Oriented Solutions

- **Create a good set of modules**
  - Modules must store, move, and alter data
  - Modules communicate with one another
  - Organize our data collection to facilitate operations on the data
- **Algorithm:**
  - A step-by-step recipe for performing a task within a finite period of time
  - Algorithms operate on a collection of data
  - Functions implement algorithms

Abstraction

- **Separates the purpose of a module from its implementation**
  - Specifications for each module are written before implementation
    - Specifications are in the C++ header file
- **Functional abstraction**
  - Separates the purpose of a function from its implementation
  - C++ function prototypes (headers)
- **Data abstraction**
  - Focuses on the operations of data, not on the implementation of the operations
  - C++ structures

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

Classes

- **Object-oriented languages**
  - Enable us to build classes of objects
- **A class combines**
  - **Attributes (characteristics) of objects**
    - Data stored by the object
    - Called data members or data fields
  - **Behaviors (operations)**
    - Typically operate on the data members
    - Called member functions or methods

Interfaces

- **Declares publicly accessible member functions (behaviors or operations)**
- **Describes only way programmers can interact with the class**
- **Classes should be easy to understand, and so have few member functions**
  - Desire to provide power to class clients is at odds with this goal

Complete and Minimal Interfaces

- **Complete interface**
  - Provides methods for any reasonable task consistent with the responsibilities of the class
  - Important that an interface is complete
- **Minimal interface**
  - Provides only essential methods
  - Classes with minimal interfaces are easier to understand, use, and maintain
  - Less important than completeness

Operation Signature

- **Programmer's interface for a member function**
  - ***Name***
    - How to refer to the abstracted code
  - ***Parameters (number, order, type)***
    - What is sent to the method
    - Function should not modify parameters (ideal world)
    - Qualifiers (`const`)
  - ***Return Type***
    - How code communicates results back to caller
    - Should return single value (ideal world)

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
    - *In C++ this requires pointers*

Abstract Data Type (ADT)

- **A collection of data and operations on that data**
  - ADT operations can be used without knowing their implementations or how data is stored
  - Classes
- **Data Structure**
  - A construct that within a programming language to store a collection of data
  - An implementation of an ADT