Object-Oriented Design & Programming

Fawzi Emad
Chau-Wen Tseng

Department of Computer Science
University of Maryland, College Park

Overview

- Object-oriented design
  - Goals
  - Techniques
  - Object-oriented view
  - Examples
Goals

- Improve software design
  - Reduce implementation effort
  - Scalable to large software projects
- Try to take advantage of two techniques
  - Abstraction
  - Encapsulation

Techniques – Abstraction

- Abstraction
  - Provide simple high-level model of
    - Physical entity
    - Activity
- Helpful for managing complexity
- Enables information hiding
  - Can change implementation & representation
  - Will not affect other software components
Types of Abstraction

- Procedural abstraction
  - Specify what actions should be performed
  - Hide algorithms

- Data abstraction
  - Specify data objects for problem
  - Hide representation

Abstraction Example

Abstraction of a Student Roster

- Data
  - List of student names

- Actions
  - Create roster
  - Add student
  - Remove student
  - Print roster

<table>
<thead>
<tr>
<th>STUDENT ROSTER</th>
<th>List of names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create()</td>
<td>AddStudent()</td>
</tr>
<tr>
<td>RemoveStudent()</td>
<td>Print()</td>
</tr>
</tbody>
</table>
Techniques – Encapsulation

Encapsulation
- Confine information so it is only visible / accessible through an associated external interface

Approach
- For some entity X in program
  - Abstract data in X
  - Abstract actions on data in X
  - Collect data & actions on X in same location
  - Protects and hides X

Encapsulation

Extension of abstraction
- Always abstract data & function together
- Encapsulated entity ⇒ Abstract Data Type (ADT)

Examples
- List ADT
  - May be implemented as array, linked list, etc...
- Java collections library
Benefits of Encapsulation

- Easier to make code modifications
  - Due to information hiding
- Promotes code reuse
  - Interface to data structure clearly defined
  - Easier to reuse code
- Code reuse increases productivity

Object-Oriented Design

- View software as
  - A collection of entities (objects)
  - Functions associated with each object
  - Communication between objects
- Exploits abstraction & encapsulation
- Can rely on programming language support
Object-Oriented View

Example problem description
- Thermostat uses dial setting to control a heater to maintain constant temperature in room

![Diagram of Thermostat (dial) with getTemperature() and heaterOn() methods, Room, and Heater]

History of Object-Oriented Design

Preceded by procedure-oriented view
- Earliest approach to programming
- Uses procedure abstraction
- Similar to actual machine instructions
- Focus on control flow, program scope
- Examples: Fortran, Cobol, Pascal, Basic

Example
- Thermostat()
  1. Get room temperature
  2. If (temperature < setting) turn heater on
  3. Else turn heater off
  4. Goto step 1
OO Programming Languages

Development history
- Simula (Dahl & Nygaard, 1962)
  - Modeling discrete event simulation
- Smalltalk (Kay, 1972)
  - General programming
- C++ (Stroustrup, 1979)
  - Manage complexity in huge software projects
- Java (Gosling, 1991)
  - Designed for embedded processors

Factors in Success of OO Design

Growing demand
- More experience with large software projects

Improvements in language design
- Made OO programming easier

Improvements compiler technology
- Support more language features efficiently

Improvements in hardware
- Handled inefficiencies in OO programming
- Made performance less critical
Elements of Object-Oriented Design

- **Objects**
  - Entities in program

- **Methods**
  - Functions associated with objects

- **Classes**
  - Groups of objects with similar properties

- **Inheritance**
  - Relationship between classes

---

**Objects**

- **Definition**
  - Entity that has state, behavior, and identity
  - **State (data)**
    - Properties possessed by object
    - Current values of those properties
  - **Behavior (methods)**
    - How objects react to changes in state
    - How objects interact with each other
  - **Identity (references)**
    - Mechanism to distinguish between objects
Object Example

Thermostat
- State
  - DesiredTemp
  - CurrentTemp
  - HeaterState
- Behavior
  - SetDesiredTemp()
  - TurnHeaterOn()
  - TurnHeaterOff()
- Identity
  - this

Object Example

Thermostat
- State
  - DesiredTemp: integer 78°
  - CurrentTemp: integer 72°
  - HeaterState: boolean ON
Object State

- **Properties**
  - Static, unchanging
  - May view as types

- **Values**
  - Dynamic, changes
  - Within bounds set by properties

Methods

- **Definition**
  - Procedures associated with object
  - Specify behavior of objects

- **Invocation** ⇒ sending message to object

- **Example**
  - `myThermostat.setDesiredTemp(78)`
  - `myThermostat.turnHeaterOn()`
  - `myThermostat.turnHeaterOff()`
Method Types

- **Accessor**
  - Return state information

- **Mutator**
  - Modify state information

- **Constructor**
  - Create & initialize new object

- **Destructor**
  - Remove object & free up resources

Identity

- How to distinguish between objects
- Reference variables
  - Used in object-oriented programming languages
  - Points to objects
  - Multiple variables may point to same object
Reference Variables

Example

Identity

- Equivalence
  - Whether two objects are equal

- Name equivalence
  - Reference variables point to same object

- Content equivalence
  - Objects from same class
  - State in each object are identical
Equivalence

Example

Name Equivalent

Content Equivalent

Thermostat
State:
DesiredTemp: 65
currentTemp: 66
HeaterState: on
Behavior:
SetDesiredTemp()
TurnHeaterOn()
TurnHeaterOff()

Thermostat
State:
DesiredTemp: 75
currentTemp: 83
HeaterState: off
Behavior:
SetDesiredTemp()
TurnHeaterOn()

Thermostat
State:
DesiredTemp: 65
currentTemp: 66
HeaterState: on
Behavior:
SetDesiredTemp()
TurnHeaterOn()
TurnHeaterOff()