Chapter 6

Problem Solving and Algorithm Design
Layers of a Computing System

Communication
Application
Operating System
Programming
Hardware
Information
Chapter Goals

- Determine whether a problem is suitable for a computer solution
- Describe the computer problem-solving process and relate it to Polya’s How to Solve It list
- Distinguish between following an algorithm and developing one
- Apply top-down design methodology to develop an algorithm to solve a problem
Chapter Goals

- Define the key terms in object-oriented design
- Apply object-oriented design methodology to develop a collection of interacting objects to solve a problem
- Discuss the following threads as they relate to problem solving: information hiding, abstraction, naming things, and testing
Problem Solving

• **Problem solving**  The act of finding a solution to a perplexing, distressing, vexing, or unsettled question
Ask Questions...

...to understand the problem

- What do I know about the problem?
- What is the information that I have to process in order to find the solution?
- What does the solution look like?
- What sort of special cases exist?
- How will I recognize that I have found the solution?
Look for Familiar Things

- You should **never reinvent the wheel**

- In computing, you see certain problems **again and again** in different guises

- A good programmer sees a task, or perhaps **part of a task (a subtask)**, that has been solved before and plugs in the solution
Divide and Conquer

• Break up a large problem into smaller units that we can handle
  • Applies the concept of abstraction
  • The divide-and-conquer approach can be applied over and over again until each subtask is manageable
Algorithms

- **Algorithm** A set of instructions for solving a problem or subproblem in a finite amount of time using a finite amount of data
- The instructions must be unambiguous
The Interactions Between Problem-Solving Phases

Problem-Solving Phase

1. Analyze
2. General Solution (Algorithm)
3. Test

Implementation Phase

1. Specific Solution (Program)
2. Test
3. Maintenance
Problem Solving
Problem Solving

$$r = \frac{V^2}{g \tan(b)}$$

- How wide is your turn?
- Slow down to the lowest possible speed
- What about bank?
Problem Solving

Radius of Turn vs. Velocity

Stall Speed

Va

Radius of turn (1000 ft)

Velocity (KTAS)
Top-Down Design

- This process continues for as many levels as it takes to expand every task to the smallest details
- A step that needs to be expanded is an abstract step
A General Example

- Planning a large party

Diagram:

- Invite the people
  - Plan the menu
    - Get cook books
    - Look for suggestions
    - Decide on food
  - Shop for food
    - Cook the food
- Prepare the food

Subtasks:

- Invite people
- Make a list
  - Write down names
    - Wait a day
    - Check list
    - Add to list
  - Call the people
    - Get phone numbers
      - While more to call
      - Call
      - Mark list
Object-Oriented Design

• A problem-solving methodology that produces a solution to a problem in terms of self-contained entities called objects

• **Object**  A thing or entity that makes sense within the context of the problem
  For example, a student
Object-Oriented Design

- A group of similar objects is described by an **object class**, or **class**
- A class contains fields that represent the properties and behaviors of the class
  - A **field** can contain data value(s) and/or methods (subprograms)
  - A **method** is a named algorithm that manipulates the data values in the object
Relationships Between Classes

- **Containment**
  - “part-of”
  - An address class may be part of the definition of a student class

- **Inheritance**
  - Classes can inherit data and behavior from other classes
  - “is-a”
Object-Oriented Design Methodology

- **Four stages** to the decomposition process
  - Brainstorming
  - Filtering
  - Scenarios
  - Responsibility algorithms
## CRC Cards

<table>
<thead>
<tr>
<th>Class Name:</th>
<th>Superclass:</th>
<th>Subclasses:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
<td>Collaborations</td>
<td></td>
</tr>
</tbody>
</table>

Page 165
Brainstorming

A group problem-solving technique that involves the spontaneous contribution of ideas from all members of the group

- All ideas are potential good ideas
- Think fast and furiously first, and ponder later
- A little humor can be a powerful force

Brainstorming is designed to produce a list of candidate classes
Filtering

- Determine which are the core classes in the problem solution
- There may be two classes in the list that have many common attributes and behaviors
- There may be classes that really don’t belong in the problem solution
Scenarios

• Assign **responsibilities** to each class
• There are **two types** of responsibilities
  • What a class must know about itself (**knowledge responsibilities**)
  • What a class must be able to do (**behavior responsibilities**)
Scenarios

• Each class **encapsulates** its data but shares their values through knowledge responsibilities.

• **Encapsulation** is the bundling of data and actions in such a way that the logical properties of the data and actions are *separated from the implementation details*
Responsibility Algorithms

• The algorithms must be written for the responsibilities
  • Knowledge responsibilities usually just return the contents of one of an object’s variables
  • Action responsibilities are a little more complicated, often involving calculations
Let's repeat the problem-solving process for creating an address list

Brainstorming and filtering
- Circling the nouns and underlining the verbs

Create an address list that includes each person's name, address, telephone number, and e-mail address. This list should then be printed in alphabetical order. The names to be included in the list are on scraps of paper and business cards.
Computer Example

- First pass at a list of classes

```
address list
name
address (telephone)
address (E-mail)
list
order
names
list
scrap
paper
cards
```
Computer Example

- Filtered list

address list
Name
address
Telephone
E-Mail
# CRC Cards

## Person

<table>
<thead>
<tr>
<th>Class Name: Person</th>
<th>Superclass:</th>
<th>Subclasses:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
<td>Collaborations</td>
<td></td>
</tr>
<tr>
<td>Initialize itself (name, address, telephone, e-mail)</td>
<td>Name, Address, Telephone, E-mail</td>
<td></td>
</tr>
<tr>
<td>Print</td>
<td>Name, Address, Telephone, E-mail</td>
<td></td>
</tr>
</tbody>
</table>

## Name

<table>
<thead>
<tr>
<th>Class Name: Name</th>
<th>Superclass:</th>
<th>Subclasses:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
<td>Collaborations</td>
<td></td>
</tr>
<tr>
<td>Initialize itself (name)</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Print itself</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>
Responsibility Algorithms

Initialize

name.Initialize()
address.Initialize()
telephone.Initialize()
email.Initialize()

Print

name.Print()
address.Print()
telephone.Print()
email.Print()
Information Hiding/Abstraction

- **Information Hiding** and **Abstraction** are two sides of the same coin.
  - **Information Hiding** The practice of hiding the details of a module with the goal of controlling access to the details of the module.
  - **Abstraction** A model of a complex system that includes only the details essential to the viewer.
Abstraction is the result with the details hidden

- **Data abstraction** Separation of the logical view of data from their implementation.
- **Procedural abstraction** Separation of the logical view of actions from their implementation.
- **Control abstraction** Separation of the logical view of a control structure from its implementation.
Programming Languages

- Instructions written in a **programming language** can be *translated* into the instructions that a computer can execute directly

- **Program** A meaningful sequence of instructions for a computer
  - **Syntax** The part that says how the instructions of the language can be put together
  - **Semantics** The part that says what the instructions mean
Homework

• Read Chapter Six, Sections 6.3 & 6.4
Mid-Term

- **Take Home** Exam – Non-Trivial (think!)
- Cover Chapters 1-5 & 16 & Anything Covered In Class
- Given Out: Oct 16
- Due Back: Oct 23
- No Lateness!!!
Good Night

...Unbiased Opinion!