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
Problem Solving

• G. Polya wrote *How to Solve It: A New Aspect of Mathematical Method*

• His How to Solve It list is quite general
  – Written in the context of solving mathematical problems
  – The list becomes applicable to all types of problems
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
Computer Problem-Solving

<table>
<thead>
<tr>
<th>Algorithm Development Phase</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Analyze</strong></td>
<td>Understand (define) the problem.</td>
</tr>
<tr>
<td><strong>Propose algorithm</strong></td>
<td>Develop a logical sequence of steps to be used to solve the problem.</td>
</tr>
<tr>
<td><strong>Test algorithm</strong></td>
<td>Follow the steps as outlined to see if the solution truly solves the problem.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Implementation Phase</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Code</strong></td>
<td>Translate the algorithm (the general solution) into a programming language.</td>
</tr>
<tr>
<td><strong>Test</strong></td>
<td>Have the computer follow the instructions. Check the results and make</td>
</tr>
<tr>
<td></td>
<td>corrections until the answers are correct.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maintenance Phase</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Use</strong></td>
<td>Use the program.</td>
</tr>
<tr>
<td><strong>Maintain</strong></td>
<td>Modify the program to meet chaining requirements or to correct any errors.</td>
</tr>
</tbody>
</table>

**Figure 6.2** The computer problem-solving process
Figure 6.3: The Interactions Between Problem-Solving Phases

- Problem-Solving Phase
  - Analyze
  - General Solution (Algorithm)
  - Test

- Implementation Phase
  - Specific Solution (Program)
  - Test
  - Maintenance
Pseudocode

• Uses a mixture of English and formatting to make the steps in the solution explicit

While (the quotient is not zero)
  Divide the decimal number by the new base
  Make the remainder the next digit to the left in the answer
  Replace the original decimal number with the quotient
Following an Algorithm

- Preparing a Hollandaise sauce

**Never-Fail Blender Hollandaise**

| 1 cup butter | 1/4 teaspoon Tabasco |
| 4 egg yolks   | 1/4 teaspoon dry mustard |
| 1/4 teaspoon salt | 2 tablespoons lemon juice |
| 1/4 teaspoon sugar |

Heat butter until bubbling. Combine all other ingredients in blender. With blender turned on, pour butter into yolk mixture in slow stream until all is added. Turn blender off. Keeps well in refrigerator for several days. When reheating, heat over hot, not boiling, water in double boiler. Makes about 1-1/4 cups sauce.
Following an Algorithm

- Preparing a Hollandaise sauce

Put butter in a pot
Turn on burner
Put pot on the burner
While (NOT bubbling)
  Leave pot on the burner
Put other ingredients in the blender
Turn on blender
While (more butter)
  Pour butter into blender in slow stream
Turn off blender
Developing an Algorithm

- The plan must be suitable in a suitable form
- Two methodologies that currently used
  - Top-down design
  - Object-oriented design
Top-Down Design

• Breaking the problem into a set of subproblems called **modules**

• Creating a hierarchical structure of problems and subproblems (modules)
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

Figure 6.6 Subdividing the party planning
A Computer Example

• Problem
  – 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
A Computer Example

Main
- Enter names into list
- Fill in missing data
- Put list into alphabetical order
- Print the list

Enter names into list
- Prompt for and enter names
- Insert names into list
  *includes other data as well*
Prompt for and enter names

Write “To any of the prompts below, if the information is not known, just press return.”
While (more names)
  Write “Enter the last name, a comma, a blank, and the first name; press return.”
  Read lastFirst
  Write “Enter street number and name; press return.”
  Read street
  Write “Enter city, a comma, a blank, and state; press return.”
  Read cityState
  Write “Enter area code and 7-digit number; press return.”
  Read telephone
  Write “Enter e-mail; press return.”
  Read eMail
Fill in missing data

Write "To any of the prompts below, if the information is still not known, just press return."

Get a name from the list

While there are more names

   Get a lastFirst
   Write lastFirst
   If (street is missing)
      Write "Enter street number and name; press return."
      Read street
   If (telephone is missing)
      Write "Enter area code and 7-digit number; press return."
      Read telephone
   If (eMail is missing)
      Write "Enter e-mail; press return."

Get a name from the list
Put list in alphabetical order
Sort list on lastFirst field

Print the list
Write “The list of names, addresses, telephone numbers, and e-mail addresses follows:”
Get a name from the list
While (there are more names)
  Write lastFirst
  Write street
  Write cityState
  Write e-Mail
  Write a blank line
  Get a name from the list
Testing the Algorithm

- The process itself must be tested
- Testing at the algorithm development phase involves looking at each level of the top-down design
Testing the Algorithm

- **Desk checking** Working through a design at a desk with a pencil and paper
- **Walk-through** Manual simulation of the design by the team members, taking sample data values and simulating the design using the sample data
- **Inspection** One person (not the designer) reads the design (handed out in advance) line by line while the others point out errors
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>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></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
Computer Example

• 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.
• First pass at a list of classes

address list
name
address
number (telephone)
address (E-Mail)
list
order
names
list
scrap
paper
cards
Computer Example

- Filtered list

address list
name
address
telephone
E-Mail
## CRC Cards

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

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