Chapter 1: Data Abstraction: The Walls

Vocabulary

Write out a description of each vocabulary term for reference later.

Term	**Description**
Abstract base class	
Abstract class	
Abstract data type (ADT):	
Abstraction, data abstraction	
Algorithm	
Attribute	
Bag ADT	
Behavior	
Cohesion	
Complete interface	
Coupling	
Data flow	
Data member	
Data structure	
Encapsulation	
End user	
Function, member function (method)	
Function prototype/declaration/header	
Generic type	
Implementation	
Information hiding	
Inheritance	
Minimal interface	
Models	
Modules	

Object-oriented analysis	
Object-oriented design	
Operation contract	
Polymorphism	
Postcondition	
Precondition	
Robust	
Unified Modeling Language (UML)	

Concepts

Encapsulation, Page 3

What is encapsulation and why is it good to use encapsulation in your design?

Cohesion, Page 4

Cohesion refers to what a module handles. If a module handles lots of different types of functionality, it would have low cohesion.

A module that is highly cohesive is:

Is it more desirable to design high or low cohesion modules?

Coupling, Page 5

Coupling refers to the relationship among different modules in a codebase.

Loosely coupled:

Highly coupled:

Which is more desirable, and why?

Example: For the following modules, label whether each one is highly coupled or loosely coupled.

`WordWriter`	`SpellChecker`
Functions: `SaveDocument, LoadDocument,`	Functions: `SetLanguage, GetLanguage,`

`NewDocument, SetHeader, AddText, UndoText, SetAlignment, AddFormula`	`GetSpellingSuggestions, HighlightProblemWords`

Operation contracts, Page 6

Function conditions

When designing functions that other programmers may use, it is important to assert your expectations: What is the state of the program *before* the function is run (precondition), and *after* the function is run (postcondition).

Example: Write out an example Precondition and Postcondition for sorting an array.

```
// Precondition:
// Postcondition:
void Sort( int arr[], int size );
```

Unusual conditions, Page 8

When writing reusable code that could be used by many programmers across many codebases, it is important to deal with errors and *unusual conditions* properly. Some ways that you can deal with unusual conditions are:

1.

2.

3.

4.

5.

Abstraction, Page 9

What is abstraction and how does it factor into designing code?

Information hiding, Page 10

When writing modules, you may want to hide the inner-workings of *how* the code operates. These might be the member variables of a class, for instance.

Why is information hiding considered good design?

What can information hiding protect against?

Interfaces, Page 12

A minimal interface is:

A complete interface is:

Design, Page 12

UML

When creating documentation or discussing design with other developers, it can be useful to use UML diagrams to get your point across. In a UML diagram, how do you represent each of the following? (Look online for specifics)

Public variable/function:

Private variable/function:

Protected variable/function: