YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

CPSC 427a: Object-Oriented Programming *Handout #9*
Professor M. J. Fischer *December 1, 2010*

Study Guide to Exam 2

For the exam, you are responsible for everything covered by exam 1 (see Study Guide to Exam 1) as well as the following materials covered since that exam: the contents of lectures 12–22 and corresponding class demos, the concepts used in problem set 4, and the entire textbook (Chapters 1–18) except for the following sections:

- Omit section 8.4 (event traces)
- Omit chapter 11 (modules and makefiles)
- Omit section 16.6 (virtual inheritance)

Below is an index to the lecture notes. It lists all of the sections, subsections, and slide titles from lectures 12–22.

33 Interacting Classes and UML (cont.) [lecture 12]

- Accessing `B` in `A`'s methods
- "Law" of Consistency/Encapsulation
- "Law" of Demeter
- "Law" of Demeter

34 Design Exercise: Family Datebook [lecture 12]

- Design Exercise: FamilyDatebook

35 Model-Viewer-Controller Paradigm [lecture 12]

- Model-Viewer-Controller design paradigm

36 Demo: Stopwatch [lecture 12]

- Realtime measurements

37 Demo: Stopwatch [lecture 13]

- Realtime measurements
- `HirezTime` class
- HirezTime class structure
- `StopWatch` class

38 Demo: Hangman Game [lecture 13]

38.1 Game Rules

- Hangman game

38.2 Code Design

- Overall design
- Use cases
- Code structure: Model
- Code structure: Viewer and controller
- Class `Game`

38.3 Storage Management

Storage management13 • String store

38.4 Refactored Game

- Refactored hangman game
- Flex arrays
- Flex array implementation issues
- String store limitation

39 Demo: Hangman Game (cont.) [lecture 14]

39.1 Refactored Game

- Refactored hangman game
- Flex arrays
- Flex array implementation issues
- String store limitation
- Refactoring `Board` class

40 Coding Practices Reminders [lecture 14]

- Get and set functions
- Coping with privacy problems
- Type safety

41 Casts and Conversions [lecture 14]

- Casts in `C`
- Different kinds of casts
- `C++` casts
- Explicit cast syntax
- Implicit casts
- Ambiguity

- `explicit` keyword

42 Operator Extensions [lecture 14]

- How to define operator extensions
- Other special cases

43 Virtue Demo [lecture 15]

- Virtual virtue
- Main virtue

44 Linear Data Structure Demo [lecture 15]

- Using polymorphism
- Interface file
- Class Linear
- Example: Stack
- Example: Queue
- Class structure
- `C++` features
- `#include` structure

45 Templates [lecture 15]

- Template overview
- Template functions
- Specialization
- Template classes
- Compilation issues
- Template parameters
- Using template classes

46 Multiple Inheritance [lecture 16]

- What is multiple inheritance
- Object structure
- Diamond pattern

47 Handling Circularly Dependent Classes [lecture 16]

- Tightly coupled classes
- Example: `List` and `Cell`
- Circularity with `#include`
- What happens?
- Resolving circular dependencies

48 Template Example [lecture 16]

- `16-Multiple-template`
- Container class hierarchy
- Item class hierarchy
- `Ordered` template class
- Alternative `Ordered` interfaces

49 The C++ Standard Library [lecture 16]

- A bit of history
- Containers
- Common container operations
- `vector<T>`

50 The C++ Standard Library (cont.) [lecture 17]

- Iterators
- Iterator example
- Using iterator inside a class
- Using subscripts and `size()`
- Algorithms
- STL `sort` algorithm
- Reverse sort example
- Reverse sort example (cont.)
- `pair<T1, T2>`
- `map<Key,Val>`
- Using a `map<Key,Val>`
- Copying from one container to another
- Copying from one container to another – example
- `string` class

51 STL and Polymorphism [lecture 18]

- Derivation from STL containers
- Replacing authority with understanding
- Two kinds of derivation
- How are they the same?
- What is simple derivation good for?
- What are the problems with simple derivation?
- What is polymorphic derivation good for?
- What are the problems of polymorphic derivation?
- Contrasts between simple and polymorphic derivation
- Containment as an alternative to simple derivation
- Argument for containment
- STL container as a base class
- Can I turn an STL container into a polymorphic base class?

- A polymorphic base class
- Dynamic cast

52 Exceptions [lecture 18]

- Exceptions
- Exception handling
- C-style solution using status returns
- C++ exception mechanism

53 Exceptions (cont.) [lecture 19]

- C++ exception mechanism (recall)
- Throwing an exception
- Catching an exception
- What kind of object should an exception throw?
- Standard exception class
- Catching standard exceptions
- Deriving your own exception classes from `std::exception`
- Multiple catch blocks
- Rethrow
- Throw restrictions
- Uncaught exceptions: Ariane 5
- Uncaught exceptions: Ariane 5 (cont.)
- Termination

54 Design Patterns [lecture 19]

- General OO principles
- What is a design pattern?
- Adaptor pattern
- Adaptor diagram
- Indirection
- Proxy pattern
- Polymorphism pattern
- Polymorphism diagram
- Controller
- Three kinds of controllers
- Bridge pattern
- Bridge diagram
- Subject-Observer or Publish-Subscribe: problem
- Subject-Observer or Publish-Subscribe: pattern
- Subject-Observer or Publish-Subscribe: diagram
- Singleton pattern
-

55 Design Patterns for Flexible and Reusable Design [lecture 20]

55.1 Software Engineering

- Reusability, Flexibility, and Maintainability
- The Waterfall Software Process
- Why a Pure Waterfall Process is Usually Not Practical
- The Spiral Process
- Advantage of OO Design
- Aspect of Reusability
- Making a Class Re-usable
- Reducing Dependency Among Classes
- Aspect of Flexibility
- Some Techniques to Achieve Flexibility
- Roadmap
- What is a Design Pattern
- UML/OMT Notation

55.2 Strategy Pattern

- Example: Duck Game
- Initial Design
- Design Change: add fly()
- Problem
- Anticipating Changes
- Handling Varying Behaviors
- Design
- Programming to implementation vs. interface/supertype
- Implementation
- Exercise
- The Strategy Pattern
- Exercise (UML diagram)
- Summary: Design Principles

56 Design Patterns for Flexible and Reusable Design (continued) [lecture 21]

56.1 Factory Pattern

- Example: KitchenViewer Interface
- KitchenViewer Example
- Selecting Antique Style
- KitchenViewer Using Standard Inheritance
- The Abstract Factory Idea
- Abstract Factory Design Pattern Applied to KitchenViewer
- Abstract Factory Design Pattern
- Concrete and Abstract Layers

- Abstract Factory Application Sequence Diagram
- Potential use of this Design Pattern?
- References

56.2 Decorator Pattern

- Example: Starbuzz Coffee
- Problem
- Problem (UML diagram)
- Attempt 1
- Potential Changes
- Design idea
- Design approach 1
- Decorator design
- Decoration Delegation Process
- Decorator Class Model
- Sequence Diagram for *Decorator*
- Decoration Features
- Exercise

56.3 Design Pattern Classification

- Some Common Design Patterns

56.4 Observer Pattern

- Example: Weather-O-Rama
- Weather-O-Rama
- Weather-O-Rama Interface
- First Implementation
- Observer Pattern
- Observer Design Pattern
- How does Observer apply these design principles?
- Discussion

57 Graphical User Interfaces [lecture 22]

- User Interfaces
- Interfaces for C++
- Overall Structure of a GUI
- Concurrency and Events
- Event Loop
- A GUI event structure
- Interface between user and system code
- Binding system calls to user functions
- Polymorphic binding
- Binding through callback registration
- Callback using function pointers: GUI side

- Callback using function pointer: User side
- Type safety
- Signals and slots

58 The `gtkmm` Framework [lecture 22]

- Structure of `gtkmm`
- Compiling a `gtkmm` program
- Linking a `gtkmm` program
- Using a GUI
- Example: `clock`
- Main program