CS 342: Object-Oriented Software Development Lab
Introduction to Patterns and Frameworks

David L. Levine
Christopher D. Gill
Department of Computer Science
Washington University, St. Louis
{levine,cdgill}@cs.wustl.edu

http://classes.cec.wustl.edu/~cs342/

Philosophy

Good design and programming is not learned by generalities, but by seeing how significant programs can be made

- clean,
- easy to read,
- easy to maintain and modify,
- human-engineered,
- efficient, and
- reliable,

by the application of good design and programming practices.

Course Goals

- Enhance object-oriented design and programming skills
- Appreciate the value of patterns in designing large-scale software
- Attain proficiency with C++
- Learn some useful programming tools and techniques

OO Features

- Inheritance
  - *interface*: to define external object behavior
  - *implementation*: to reuse base class code
  - extension vs. specialization
- Polymorphism
  - to support identical treatment of objects of different types
- Dynamic binding
  - to defer object behavior definition until runtime
OO Design Principles

- The Open/Closed Principle
  - Classes should be open for extension, closed for modification.
- Decomposition
- Modules
- Interfaces

Why Use C++?

- To maximize execution speed
- To support reuse, with separation of interface and implementation
- To support data abstraction and dynamic binding
- For portability
- For backward source compatibility with C
- For link compatibility with C, Basic, Fortran, Ada, etc.
- To maximize execution speed

C++ Features

- Function declarations (prototypes) and definitions
  - Overloading
- Class declarations and member definitions
  - member functions, data members
    * manager, helper, accessor, and implementor functions
  - static member functions, static data members
  - template classes
- virtual functions
- Constructors and destructors
  - base/member initialization section
  - virtual destructors

C++ Features, (cont’d)

- Initialization, assignment, and comparison operators
- Memory management
  - pointers and references
  - heap management (new and delete)
  - pass-by-value and pass-by-reference
- const
- Inline functions
- I/O
- Exception handling
**C++ Objects**

- Code and data
- Memory allocation and object lifetime
- What is an object? How do I create an object?
- Objects, references, pointers, arrays
- Header files
- Things to watch out for
  - Be careful with aliases (references) and pointers!
  - Never use an object after it has been deleted from the heap!
  - Never delete an object that you didn’t allocate with new!
  - Never pass the address of a local object out of a function!
  - Avoid dynamic allocation of local variables, unless lifetime must extend past function end.
  - Minimize contents of header files.

---

**Inheritance: General**

- Benefits of inheritance
  - reuse
  - support for polymorphism via abstract interfaces
  - extension of existing implementation
  - modelling of hierarchically-related objects in problem space
  - reduced testing effort
  - reuse
- Drawbacks of inheritance
  - difficulty of understanding complex hierarchies
  - interface stability of base classes sometimes requires even more complexity
  - small performance penalties of multiple inheritance and dynamic binding
  - multiple inheritance can lead to wasted space, or use of complicated alternatives such as virtual inheritance

---

**Inheritance: C++**

- Single and multiple inheritance
- Protection
  - public, protected, and private members
- Derived class objects can be treated as base class objects
  - However, the other direction requires downcasting (or run-time type identification)
- Base class default constructors will be called implicitly, if not constructed explicitly
- Base class destructors always called implicitly
- Virtual inheritance

---

**C++ I/O**

- Why are C++ iostreams so cool?
  - Object types are inferred by compiler, not listed explicitly in code
  - Overloading provides uniformity
  - Extensible (and composable)
  - Independent of input/output devices
- What are some disadvantages of iostreams?
  - Non-traditional interface
  - Not supported on all platforms
  - Higher overhead than direct I/O
  - Evan doesn’t like them :-)

---
**Key Design Concepts and Principles**

- Important design concepts and design principles include:
  - Decomposition
  - Abstraction
  - Subset Identification
  - Information Hiding
  - Virtual Machine Structuring
  - Modularity
  - Separating Policy and Mechanism
  - Hierarchy

- Main purpose of these concepts and principles is to manage software system complexity and improve software quality.

**Modularity Dimensions**

- Modularity has several dimensions and encompasses specification, design, and implementation levels:
  - Criteria for evaluating design methods with respect to modularity
    * Modular Decomposability
    * Modular Composability
    * Modular Understandability
    * Modular Continuity
    * Modular Protection
  - Principles for ensuring modular designs:
    * Language Support for Modular Units
    * Few Interfaces
    * Small Interfaces (Weak Coupling)
    * Explicit Interfaces
    * Information Hiding

**The C++ STL**

- A collection of composable class and function templates
  - Helper class and function templates: operators, pair
  - Container and iterator class templates
  - Generic algorithms that operate over *iterators*
  - Function objects
  - Adaptors

- Enables generic programming in C++
  - Each generic algorithm can operate over *any iterator for which the necessary operations are provided*
  - Extensible: can support new algorithms, containers, iterators

**Generic Programming: why use the STL?**

- Reuse: “write less, do more”
  - The STL hides complex, tedious and error prone details
  - The programmer can then focus on the problem at hand
  - *Type-safe* plug compatibility between STL components

- Flexibility
  - Iterators decouple algorithms from containers
  - Unanticipated combinations easily supported

- Efficiency
  - Templates avoid virtual function overhead
  - Strict attention to time complexity of algorithms
Patterns and Frameworks

- Patterns support reuse of software architecture and design
  - Patterns capture the static and dynamic structures and collaborations of successful solutions to problems that arise when building applications in a particular domain
- Frameworks support reuse of detailed design and code
  - A framework is an integrated set of components that collaborate to provide a reusable architecture for a family of related applications.
- Together, design patterns and frameworks help to improve software quality and reduce development time
  - e.g., reuse, extensibility, modularity, performance

Design Pattern Space

- Creational patterns
  - Deal with initializing and configuring classes and objects
- Structural patterns
  - Deal with decoupling interface and implementation of classes and objects
- Behavioral patterns
  - Deal with dynamic interactions among societies of classes and objects

Creational Patterns

- Factory Method
  - Method in a derived class creates associates
- Factory
  - Assembles resources necessary to create an object
- Singleton
  - Factory for a singular (sole) instance

Structural Patterns

- Adapter
  - Translator adapts a server interface for a client
- Bridge
  - Abstraction for binding one of many implementations
- Composite
  - Structure for building recursive aggregations
- Facade
  - Facade simplifies the interface for a subsystem
Behavioral Patterns

- **Iterator**
  - Aggregate elements are accessed sequentially
- **Memento**
  - Snapshot captures and restores object states privately
- **Observer**
  - Dependents update automatically when a subject changes
- **Strategy**
  - Abstraction for selecting one of many algorithms

---

Behavioral Patterns (cont’d)

- **Command**
  - Encapsulates an operation as a first-class object
- **Template Method**
  - Algorithm with some steps supplied by a derived class

---

Tools and Techniques

- **Scientific Method**
- **Problem reporting**
- **Application of tools to programming problems**
  - Debuggers
  - Source control
  - Build control
  - Static and dynamic analysis tools
  - Scripts

---

Reuse, Reuse, Reuse

"Lesser artists borrow, great artists steal."

– Igor Stravinsky