Software Design and Implementation

- Object oriented programming and design
  - good design helps do away with late night Teer-fests, but some late nights are inevitable
  - your toolkit must include mastery of language/programming and design

What’s in the course?

- C++ and Java, team and mastery projects
  - team projects can be more and less than the sum of their parts
- high-level abstractions, low-level details
  - patterns, heuristics, and idioms
Program Design and Implementation

- Language independent principles of design and programming
  - design heuristics
    - coupling, cohesion, small functions, small interfaces ...
  - design patterns
    - factories, adapter, MVC aka observer/observable, ...

- Language specific:
  - Idioms
    - smart pointers, vectors/arrays, overloaded operators ...
  - idiosyncracies, idiocies
    - must define virtual destructor, stream zoo in Java, ...
 Administrivia

- check website
- Grading (see web pages)
  - group projects: small, medium, large
  - mastery programs (solo or semi-solo endeavors)
  - readings and summaries
  - quizzes
- Evaluating team projects, role of TA, UTA, consultants
  - face-to-face evaluation, early feedback
- Compiling, tools, environments, Linux, Windows
  - g++ 3.3, Java 2 aka 1.4, JRE, ...
Classes: Review/Overview

- A class encapsulates state and behavior
  - Behavior first when designing a class
  - Information hiding: who knows state/behavior?

- State is private/protected; some behavior is public
  - Private/protected helper functions
  - A class is called an object factory, creates lots of instances

- Classes communicate and collaborate
  - Parameters: use-a (send and receive)
  - Containment: has-a (aggregate of parts, responsible for)
  - Inheritance: is-a (extends and specializes)
C++ and Java class construction

- C++ uses .h and .cpp, Java uses .java
  - Documentation different (javadoc vs. doxygen)

- Default, overloaded, copy constructor
  - tvector, string, Date
  - Default constructor needed in C++, where?
  - Copy constructor needed to avoid shallow copy
  - In C++ destructors needed to free resources/self, Java?
  - Clone makes copy in Java (rare), share is default

- Private, protected, public, (package)
  - Private default in C++, package default in Java
  - Per method declaration in Java, class sections in C++
Design Criteria

*Good design comes from experience, experience comes from bad design*

Fred Brooks (or Henry Petroski)

- **Design with goals:**
  - ease of use
  - portability
  - ease of re-use
  - efficiency
  - first to market
  - ??????
How to code

- Coding/Implementation goals:
  - Make it run
  - Make it right
  - Make it fast
  - Make it small

- Spiral design (or RAD or waterfall or ...)
  - what’s the design methodology?
XP and Refactoring

(See books by Kent Beck (XP) and Martin Fowler (Refactoring))

- **eXtreme Programming (XP) is an agile design process**
  - Communication: unit tests, pair programming, estimation
  - Simplicity: what is the simplest approach that works?
  - Feedback: system and clients; programs and stories
  - Courage: throw code away, dare to be great/different

- **Refactoring**
  - Change internal structure without changing observable behavior
  - Don’t worry (too much) about upfront design
  - Simplicity over flexibility (see XP)
Modules, design, coding, refactor, XP

- Do the simplest thing that can possibly work (XP)
  - Design so that refactoring is possible
  - Don’t lose sight of where you’re going, keep change in mind, but not as the driving force [it will evolve]

- Refactor: functionality doesn’t change, code does
  - Should mean that new tests aren’t written, just re-run
  - Depends on modularity of code, testing in pieces

- What’s a module in C++
  - Could be a class, a file, a directory, a library, a namespace
  - We should, at least, use classes, files, directories
Design Heuristics: class/program/function

(see text by Arthur Riel)

- **Coupling**
  - classes/modules are independent of each other
  - goal: minimal, loose coupling
  - do classes collaborate and/or communicate?

- **Cohesion**
  - classes/modules capture one abstraction/model
  - keep things as simple as possible, but no simpler
  - goal: strong cohesion (avoid kitchen sink)

- **The open/closed principle**
  - classes/programs: open to extensibility, closed to modification