CPS 108, Spring 2004

- **Software Design and Implementation**
  - **Object oriented programming and design**
    - Language independent concepts including design patterns, e.g., Model-View-Controller, iterator, factory, bridge, …
    - Design independent concepts, e.g., coupling, cohesion, testing, refactoring, profiling, …

- **What's in the course?**
  - **C++ and Java, team projects, mastery exams**
    - team projects can be more and less than the sum of their parts
  - **high-level abstractions, low-level details**
    - patterns, heuristics, and idioms

Application Programming Interface, API

- **Standard C++ headers, STL headers**
  - **Curses**
  - **wxWindows**

- **Java API**
  - **java.util, javax.swing, java.net, javax.crypto, …**
  - **Eclipse SWT**

- **Google API**

- **Gracenote API**

Program Design and Implementation

- **Language independent principles of design and programming**
  - **design heuristics**
    - coupling, cohesion, small functions, small interfaces ...
  - **design patterns**
    - factories, adapter, MVC, decorator, iterator, …

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

Administrivia

- **check website and bulletin board regularly**
  - **http://www.cs.duke.edu/courses/cps108/current/**
  - **See links to bulletin board and other stuff**
- **Grading (see web pages)**
  - **group projects: small, medium, large**
  - **mastery programs (solo or semi-solo endeavors)**
  - **readings and summaries**
  - **tests**
- **Evaluating team projects, role of TA, UTA, consultants**
  - **face-to-face evaluation, early feedback**
- **Compiling, tools, environments, Linux, Windows, Mac**
  - **g++ 3.3, Java 2 aka 1.4, JRE, …**

wordlines.cpp, understanding code

```
typedef map<string, set<int> > linemap;

while (getline(input,line)) {

    linecount++;
    istringstream iline(line);
    while (iline >> w) {
        linemap::iterator it = info.find(w);
        if (it == info.end()) {
            set<int> si;
            si.insert(linecount);
            info.insert(make_pair(w,si));
        }
        else {
            it->second.insert(linecount);
        }
    }
}
```

Questions about wordlines.cpp

- Conceptually, what's a map and what's a set?

- In terms of implementation how do they work?

- What's an iterator (abstractly and concretely)

- What are streams? ifstream, cin/cout, istring/ostring, ...

- What's a templated class?

- Other questions?

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: send and receive
  - Containment: has a reference to
  - Inheritance: is-a

C++ (and Java) class construction

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

- Default ctor, copy constructor, destructor, assignment operator
  - tvector, string, Date
  - 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?**
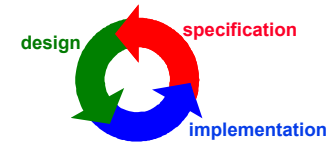
XP and Refactoring

(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

- **Make it run, make it right, make it fast, make it small**
- **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

---

C++ idioms/general concepts

- **Genericity**
  - ➢ Templates, STL, containers, algorithms
- **Copy/Assignment/Memory**
  - ➢ Deep copy model, memory management "required"
- **Low-level structures**
  - ➢ C-style arrays and strings compared to STL, Tapestry
- **const**
  - ➢ Good for clients, bad for designers/coders?

- **From C to C++ to Java**
  - ➢ function pointers, function objects, inheritance

Standard Libraries

- **In C++ there is the *Standard Library*, formerly known as the *Standard Template Library* or *STL***
  - ➢ Emphasizes generic programming (using templates)
  - ➢ Write a sorting routine, the implementation depends on
    - • Elements being comparable
    - • Elements being assignable

  We should be able to write a routine not specific to int, string or any other type, but to a generic type that supports being comparable/assignable
- **In C++ a templated function/class is a code-factory, generates code specific to a type at compile time**
  - ➢ Arguably hard to use and unsafe

Eric Raymond

- **Open source evangelist**
  - ➢ The Cathedral and the Bazaar
  http://ot.op.org/cathedral-bazaar.html
  - ➢ How to construct software
- **"Good programmers know what to write. Great ones know what to rewrite (and reuse)."**
- **How to convince someone that guns are a good idea? Put this sign up:**

- **THIS HOME IS A GUN-FREE ZONE**