CPS 108, Fall 2007

- Object oriented programming and design, we’ll use Java and ... (javascript, GWT, XML, TLA)
  - Language independent concepts including design patterns, e.g., Model-View-Controller, iterator, factory, strategy, ...
  - Design independent concepts, e.g., coupling, cohesion, testing, refactoring, profiling, ...

- Non OO programming and design, we’ll use whatever is appropriate
  - Could be php, javascript, python, J2EE
  - From classes to functions, from hashes to maps

Goals for students in Compsci 108

- Adept at solving problems requiring programming
  - Design, test, implement, release, revise, maintain

- Reasonably facile with basic Java idioms/libraries
  - How to read the API, knowing what to ignore
  - Basic language features, basic libraries

- How to find, use, extend APIs
  - When DIY is appropriate
  - Is wheel-reinvention good, bad, ugly, other?

More goals for 108 students

- Know patterns catalog, vocabulary and use
  - Instead of GOF, use Shalloway Trott

- Experience working in teams
  - How to accommodate team needs, balance against individual needs (and goals)

- Comfort in working alone, how to get and use help
  - Peers, UTAs, TA, prof, Internet, ...

Administrivia

- check website and bulletin board regularly
  - See links to bulletin board and other stuff

- Grading (see web pages)
  - group projects: small and large
  - mastery programs (solo or semi-solo endeavors)
  - readings and summaries
  - Tests (one in-class, one take-home)
**Administrivia (continued)**

- Evaluating team projects, role of TA, UTA, consultants
  - face-to-face evaluation, early feedback

- Compiling, tools, environments, Linux, Windows, Mac
  - Java 6, we’ll use features of this release
  - GWT, which is based on Java 4
  - Eclipse in all environments
  - CVS or SVN (tbd)

**Lofty Goals and Real Work**

- How do we make a tag cloud?

**Starting: Blank Screen Phenomenon**

- What will we do first?
  - How will we know it’s right?
  - What’s the next step?

- How do we get help?
  - Where do we look?
  - What do we look for?

- What’s the next step?

**Classes: Review/Overview**

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

- State is private; some behavior is public
  - Private/protected helper functions
  - A class is called an object factory, creates lots of instances
How do classes and objects work?

- Classes communicate and collaborate
  - Parameters: use-a, send and receive
  - Containment: has-a, aggregate, responsible for
  - Inheritance: is-a, extends, specializes

- Understanding inheritance and interfaces
  - What is polymorphism?
  - When is polymorphism not appropriate?
  - What is an interface in Java (C++, Ruby)?

Design Criteria

Good design comes from experience, experience comes from bad design

Fred Brooks

- 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 Java?
  - Could be a class, a file, a directory, a package, a jar file
  - We should, at least, use classes and packages

**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

**Eric Raymond**

- Open source evangelist
- The Cathedral and the Bazaar
- 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

**David Parnas (ACM fellow)**

I would advise students to pay more attention to the fundamental ideas rather than the latest technology. The technology will be out-of-date before they graduate. Fundamental ideas never get out of date. However, what worries me about what I just said is that some people would think of Turing machines and Goedel’s theorem as fundamentals. I think those things are fundamental but they are also nearly irrelevant. I think there are fundamental design principles, for example structured programming principles, the good ideas in “Object Oriented” programming, etc.