Practice What You Preach

Full Separation of Concerns in CS_1/CS_2

Hamzeh Roumani
Department of Comp Science & Engineering
York University, Toronto, Canada

york'06

CONTEXT

SIGCSE, its conferences and activities, and attitudes of its members toward OOP in CS1

york'06

Contents

york'06

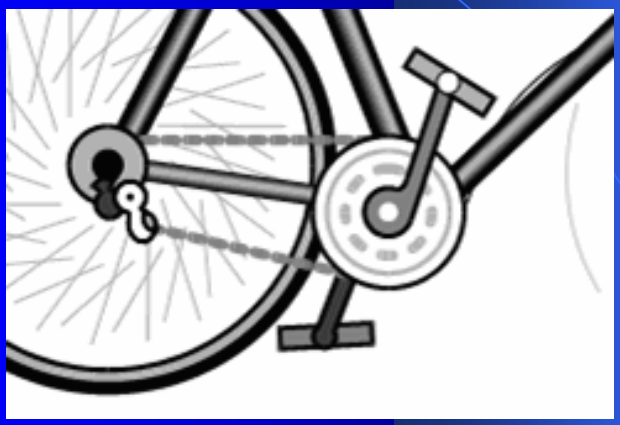
The King

The Minister

york'06

Session 1:

Pedaling and the Chain



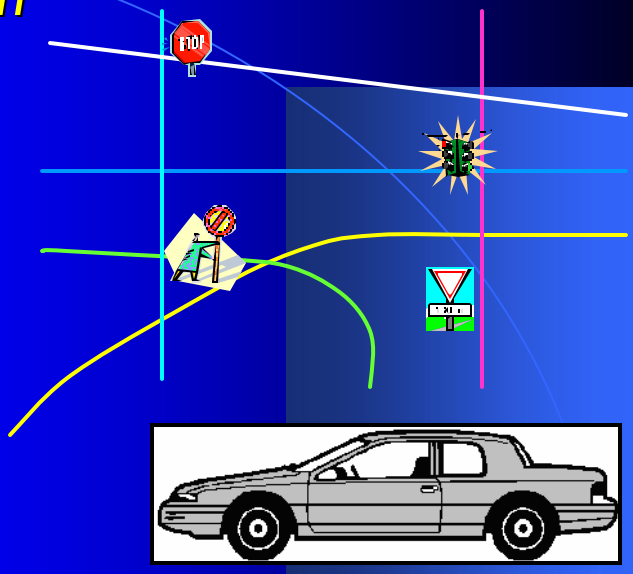
york'06

The Bicycle Course

1. Pedaling & the Chain
2. Braking & the Pads
3. Steering and the Axis
4. Etiquette of the Road

york'06

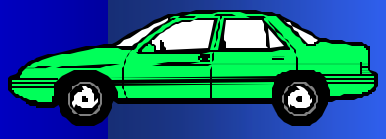
The Queen



york'06

The Car Course

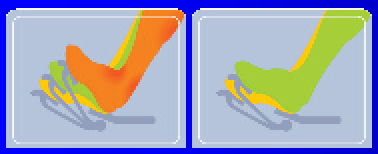
1. The Gas pedal, Spark Plugs, and the Green Light
2. The Brake Pedal, Disk Breaks, and the Red Light
3. The Steering Wheel, Pumps, and Turn Signals

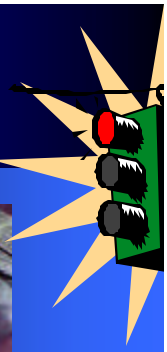


york'06

Session 2:

Stopping the Car





york'06

york'06

Q: What makes a car stop?

A: When the traffic light turns red, the brake fluid gets compressed and this pulls on the pedal so the driver must depress it. This stops the car.

york'06

york'06

Contents

What went wrong?

- The car course should have been longer; perhaps split into two or three courses?
- The Minister had little or no experience teaching cars?
- The younger generation is weaker and shallower; they cannot handle cars?
- Cars considered harmful; let's go back to good'ol bicycles.

The bike course taught you how to ride and how the bike works.

Works because linkage between cause and effect is direct and visible: you pedal, the chain rotates, the bike moves.

The car course teaches you how to drive and how the car works.

Doesn't work because the journey from the gas pedal, thru spark plugs, to the wheel is long, invisible, and complex.

york'06

Why is it, then, that most people today can easily learn how to drive?

york'06

The Hood

- Today, most people can learn how to drive easily thanks to the hood.
- It hides the complexity by encapsulating it under it.
- It allows drivers to think of abstractions (steering wheel, brake, etc.) rather than how engines work.
- It separate the concerns.

Rather than remove it, celebrate it!

york'06

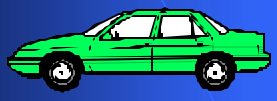
Contents

york'06

Pascal, Turing…

Simplicity allows us to teach usage (calling a procedure or function) and implementation (making one) together.

OOP



To confront the complexity, must separate class usage (e.g. method invocation) from class implementation.

york'06

Separation of Concerns: The Prime Directive

- **Dijkstra, EWD447**
  But nothing is gained --on the contrary!-- by tackling these various aspects simultaneously. It is what I sometimes have called "the separation of concerns", which, even if not perfectly possible, is yet the only available technique for effective ordering of one's thoughts, that I know of. This is what I mean by "focusing one's attention upon some aspect": it does not mean ignoring the other aspects, it is just doing justice to the fact that from this aspect's point of view, the other is irrelevant.
- **The main theme in the evolution of Computer Science**
  Algorithms & data structures; MVC, Computer architecture, Network Protocol Stack, …
- **Numerous SIGCSE papers**
  All calling for "An Exodus from Implementation-Biased Teaching" (see paper)

york'06

Separation of Concerns

- So important, so well-recognized, so obvious!
- We adopt it in all our 2nd, 3rd, 4th year course.

It is therefore mind boggling that our very first course should ignore this directive and that all the popular textbooks cover usage and implementation together* in the same chapter and sometimes in the same sentence!

*E.g. formal parameters – arguments, new – this, super – polymorphism, validation - precondition …

york'06

Separation of Concerns

The New CS_1

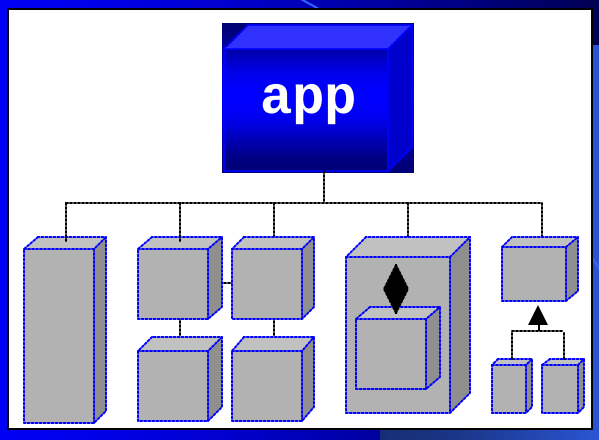
Write apps (`main` method) → What

The New CS_2

Implement classes → How

york'06

Component-Based Architecture



The components (drawn from J2SE) can be standalone or members of aggregation and/or inheritance hierarchies.

york'06

WordStat

But what if the energy to develop and support these contracts is overwhelming? A system that consists of just five standard components, each available in four commercial implementations, produces 20 configurations to test, certify, and support. How does this scale out? According to a distinguished engineer from IBM, a recent internal study revealed that only about one in three dollars that IBM spends on software product development goes toward new features. The other two-thirds are spent on non-value-added integration costs. IBM is no less efficient in software development than the rest of the industry. Mathematics is hard to argue with...

Write an app that takes a text file and produces a list of its word frequencies (case insensitive, space delimiters)

york'06

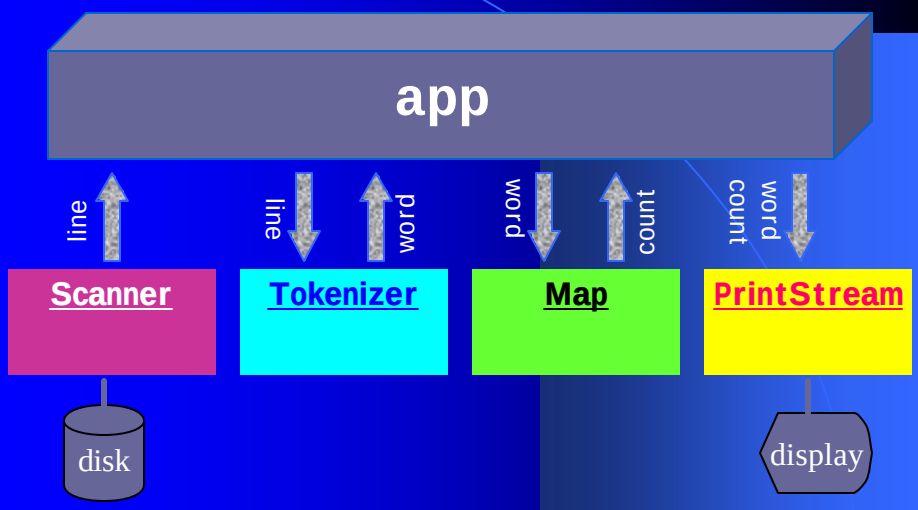
WordStat

But what if the energy to develop and support these contracts is overwhelming? **A** system **that** consists of just five standard components, each available in four commercial implementations, produces 20 configurations to test, certify, and support. How does this scale out? According to **a** distinguished engineer from IBM, **a** recent internal study revealed **that** only about one in three dollars **that** IBM spends on **software** product development goes toward new features. The other two-thirds are spent on non-value-added integration costs. IBM is no less efficient in **software** development than the rest of the industry. Mathematics is hard to argue with…

a = 3, software = 2, that = 3, …

york'06

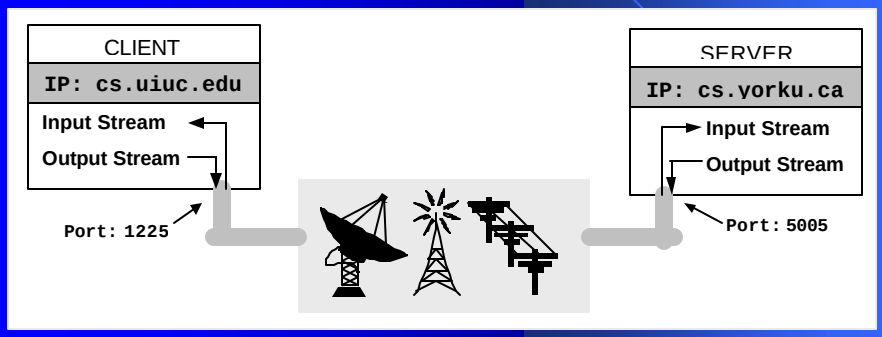
WordStat Architecture



york'06

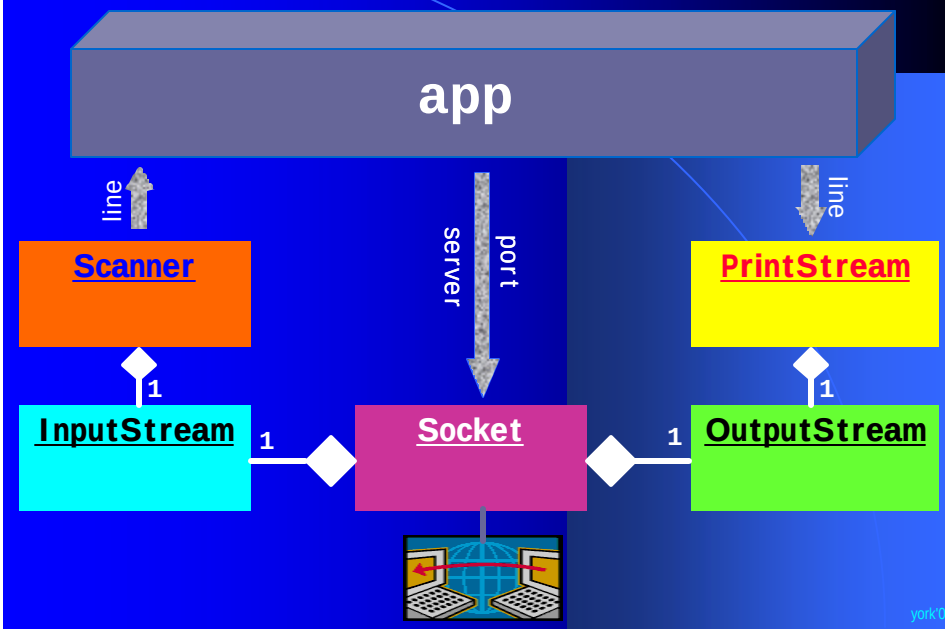
Connect

Write an app that connects to a server having a given IP and port and then communicates with it based on its protocol.



york'06

Connect Architecture



york'06

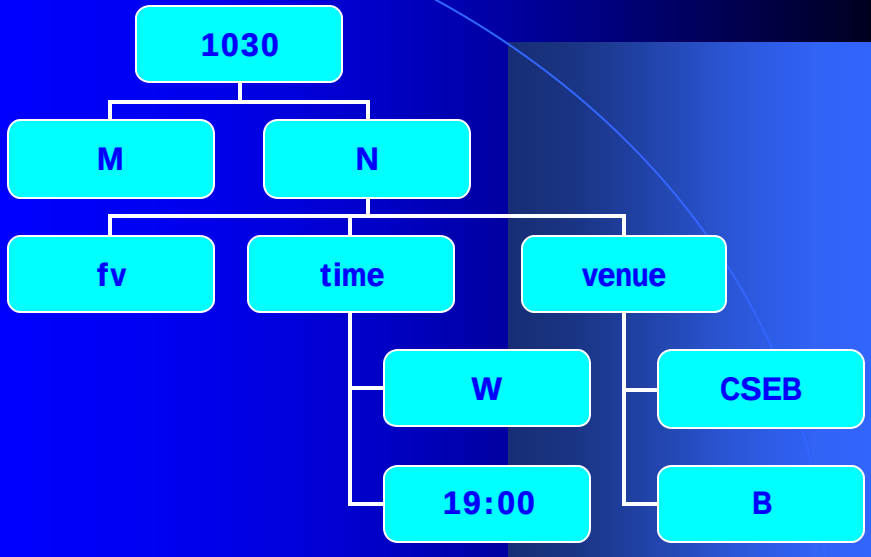
ParseXML

Given an XML timetable, determine how many courses use a given building.

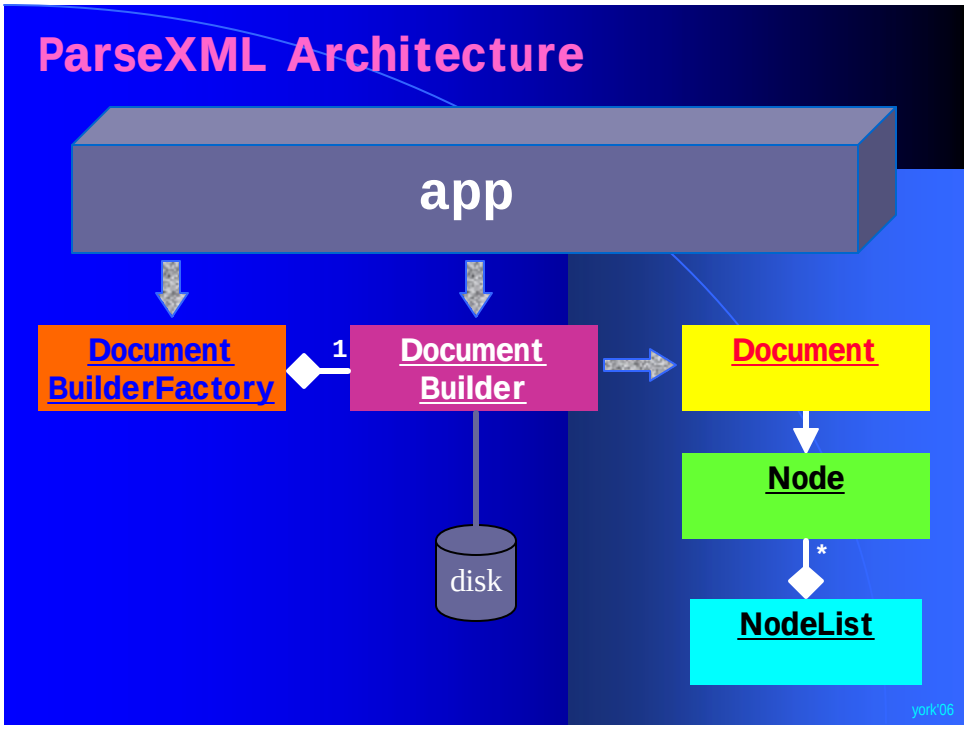
```
<?xml version="1.0" encoding="UTF-8"?>
<timetable term="W06" xmlns="www.cs.yorku.ca/2005-06/W">

  <course credit = "3" id = "1030">
    <section id = "M">
      <instructor>hr</instructor>
      <time><day>MWF</day><hour>10:30</hour></time>
      <venue><building>RSS</building><room>137s</room></venue>
    </section>

    <section id = "N">
      <instructor>fv</instructor>
      <time><day>W</day><hour>19:00</hour></time>
      <venue><building>CSEB</building><room>B</room></venue>
    </section>
  </course>

</timetable>
```

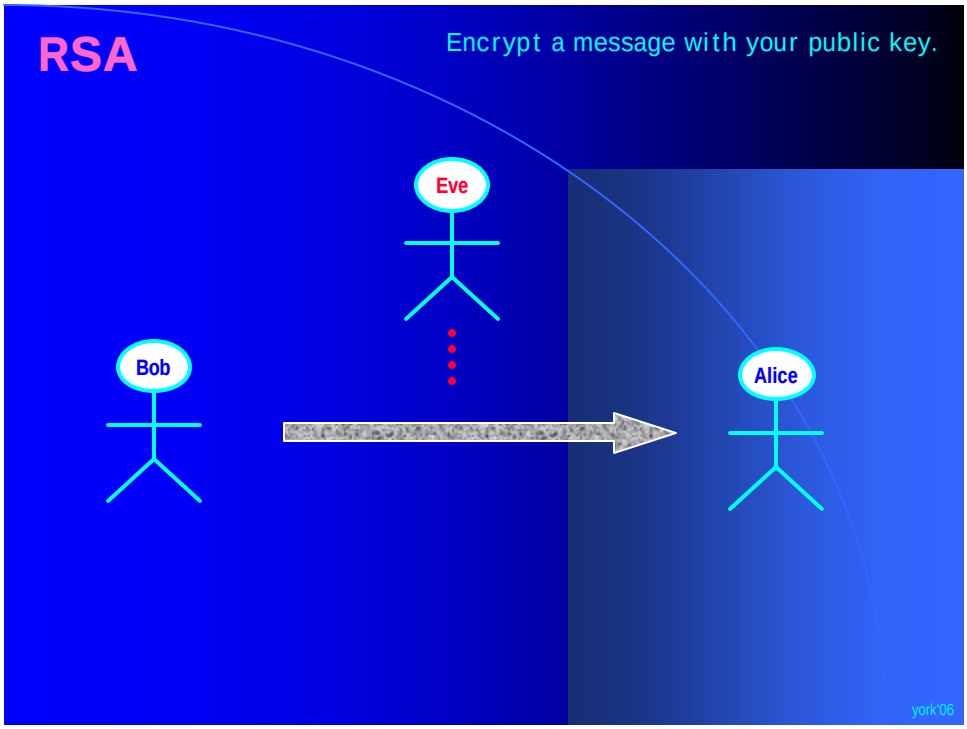
york'06

ParseXML

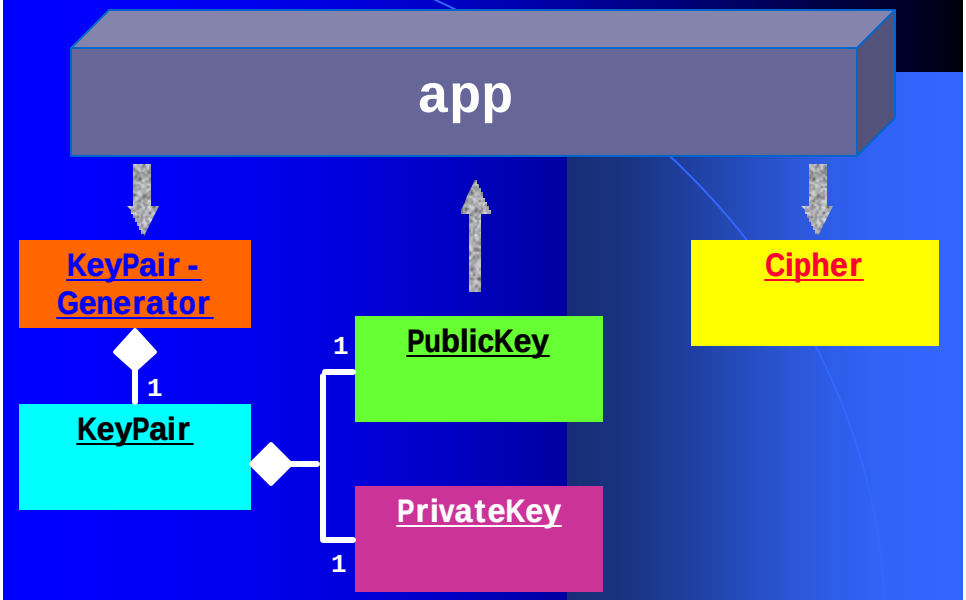


york'06


ParseXML Architecture
app
Document BuilderFactory
1
Document Builder
Document
Node
*
NodeList
disk
york'06


RSA
Encrypt a message with your public key.
Eve
Bob
Alice
york'06

RSA Architecture



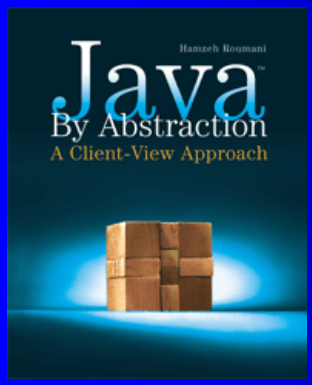
Contents

1. Once upon a time...
2. Reflections
3. Separation of Concerns
4. The York experience

york'06

CS_1 @York

- 1999: Moved to conventional (mixed-concerns) Java
- 2003: Moved to the client view using Java By Abstraction



1. What is Programming
2. Delegation
3. API
4. Objects
5. Control Structures
6. Strings, Tokenizers, Regex
7. Dev Methodologies, UML, Testing
8. Aggregation and Composition
9. Inheritance
10. The Collection Framework
11. Exception Handling
12. Applications

york'06

CS_1

Significant improvement in quality

- Short programs, confidence, debugging
- Focus on problem solving, not language
- Emphasize S/E and loop invariants
- Contracts and responsibilities are key
- Can develop "cool" applications quickly
- Promote system (integrative) thinking

Side Benefit: not everyone who takes CS_1 wants to be a car mechanic!

york'06

CS_2 @York

- 1999: Moved to conventional (mixed-concerns) Java
- 2004: Moved to the implementer's view

1. Implementing a utility class
2. Implementing a non-utility class
3. Implementing aggregation
4. Implementing generic collections
5. Implementing inheritance
6. Implementing abstract classes and interfaces
7. Review of the implementer's view
8. GUI Applications
9. Recursion
10. Searching and sorting
11. Introduction to data structures
12. Linked Lists

york'06

CS_2

- Marked improvement in quality
- The 2-course package seen as much easier
- Lots of "Aha!"
- Focus on the implementer's concern
- The implementer is a client too

york'06


Questions?
york'06