Northeastern University - Seattle

Computer and Information Sciences

Program Design Principles – PDP CS5010

Week 1 – Introduction PDP

Lecture overview

- Aims of PDP
- Logistics of PDP
- Class exercise and discussion
- Design by Contract

Course Primary Aims

- At the end of this course you should be able to:
  - Design and build high quality software
  - Explain the major principles of the ‘art of programming’
  - Write understandable code
  - Be able to explain your design and code to your peers

Course Secondary Aims

- You will also:
  - Have advanced knowledge and skills in Java, including Java 8.0 features
  - Be able to write concurrent Java programs
  - Have experience with a number of widely used Java components

High Quality Software

- High quality software should be:
  - Correct
  - Comprehensible
  - Modifiable

Correct

- Meet functional requirements
  - Pass test cases
- Programming is not math
  - No one answer
    - But there are good ones and bad ones ☺
  - No single design method or approach
- Programming is a design exercise
  - Apply design principles
  - Apply best practices such as design patterns
  - Justify and explain your thinking

Comprehensible

- Your code has two equally important audiences:
  - CPU and systems
  - Other engineers
- Your code should be
  - Easy for others to understand
  - Well documented
- This will be tested in walkthroughs
  - You'll need to explain your design and code to TAs and Professors

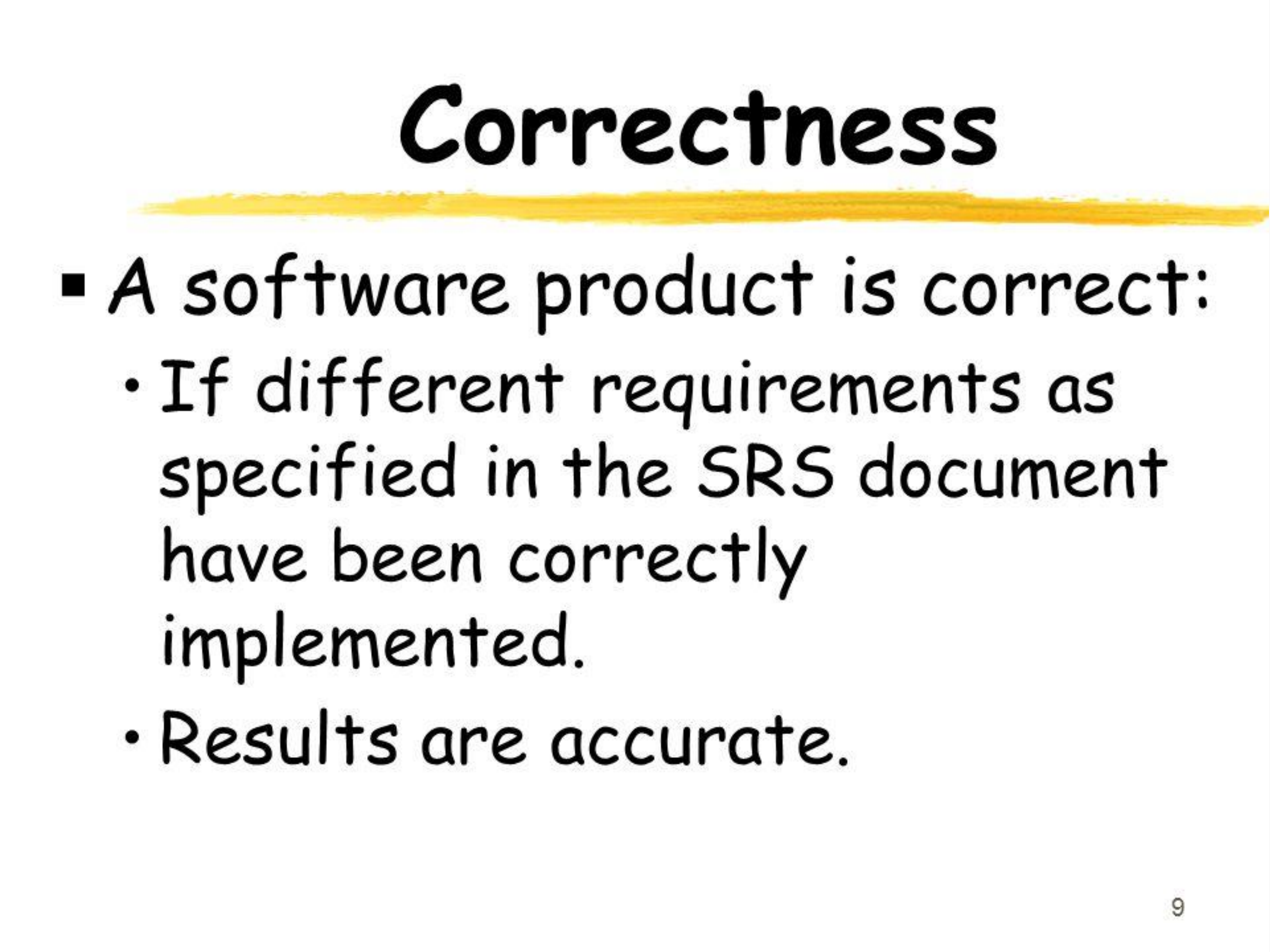
?
?
?

HOMER

15 Min

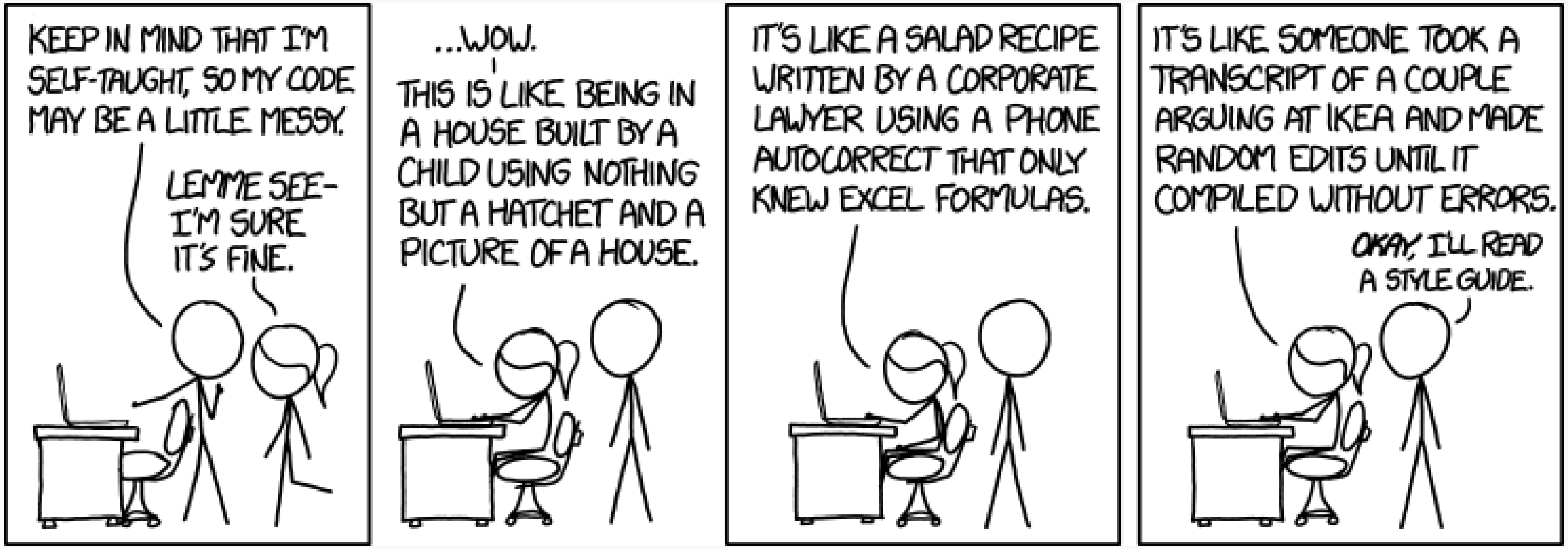
Modifiable

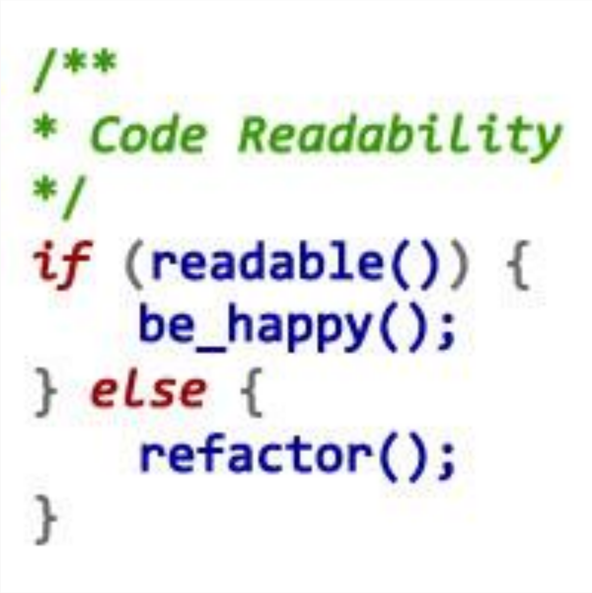
- Software systems always change and evolve
  - Your code should be comprehensible so other engineers can use and modify it
- Design principles make it possible to build modifiable software
  - But there are always trade-offs
  - Some changes are easier to make than others
    - And some will be hard/impossible
  - The art of design is to anticipate likely/most common changes and accommodate those



Correctness

- A software product is correct:
  - If different requirements as specified in the SRS document have been correctly implemented.
  - Results are accurate.


KEEP IN MIND THAT I'M SELF-TAUGHT, SO MY CODE MAY BE A LITTLE MESSY.
LEMME SEE– I'M SURE IT'S FINE.
...WOW.
THIS IS LIKE BEING IN A HOUSE BUILT BY A CHILD USING NOTHING BUT A HATCHET AND A PICTURE OF A HOUSE.
IT'S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCORRECT THAT ONLY KNEW EXCEL FORMULAS.
IT'S LIKE SOMEONE TOOK A TRANSCRIPT OF A COUPLE ARGUING AT IKEA AND MADE RANDOM EDITS UNTIL IT COMPILED WITHOUT ERRORS.
OKAY, I'LL READ A STYLE GUIDE.



```
/**
* Code Readability
*/
if (readable()) {
    be_happy();
} else {
    refactor();
}
```

The end goal – Software Engineer

The Difference Between

A SOFTWARE ENGINEER

&

A SOFTWARE DEVELOPER

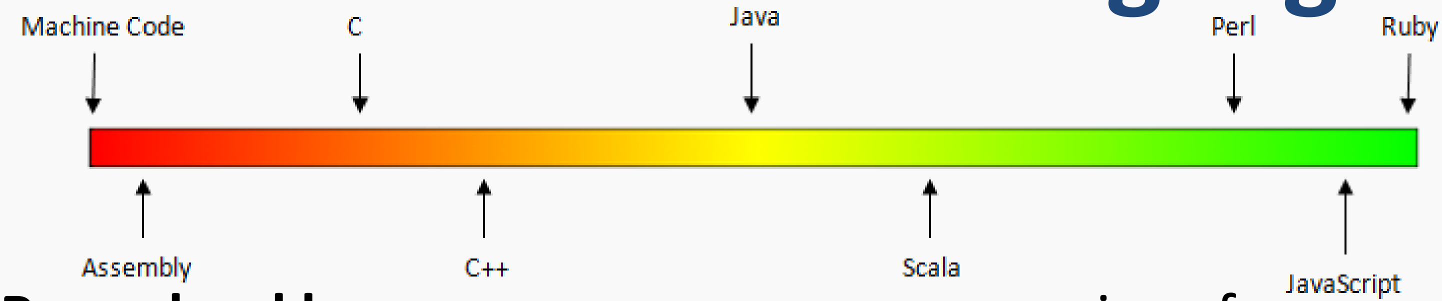
Software Engineering and Practice

- Good software is not just the right output.
- Many other goals exist.
- "Software engineering" promotes the creation of good software, in all its aspects
  - Directly code-related: class and method design
  - Externally: documentation, style
  - Some of it is higher-level: system architecture
- Software quality is important in this class AND in the profession

HIRING
SOFTWARE
ENGINEERS

NAVIGATING YOUR
FIRST WEEK
AS A
SOFTWARE ENGINEER
NEW HIRE

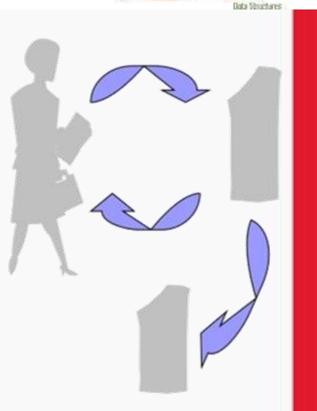
Some modern languages



Procedural languages: programs are a series of command

Pascal (1970): designed for education

C (1972):low-level operating systems and device drivers



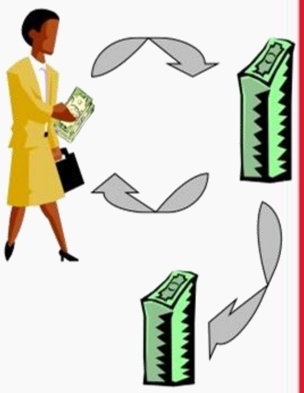
Withdraw, deposit, transfer

Object-oriented languages: programs use interacting "objects"

C++ (1985):"object-oriented" improvements to C

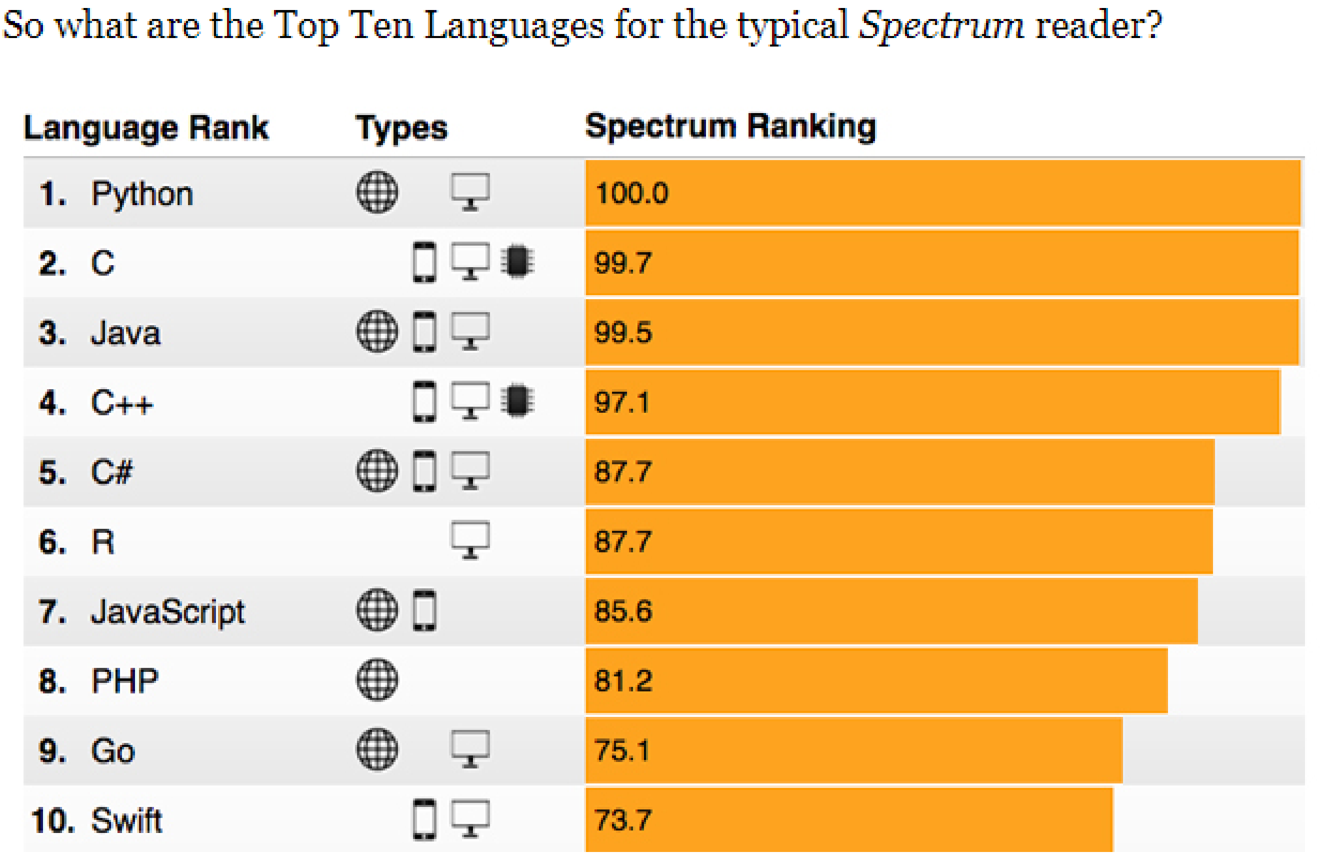
Java (1995):

- Designed for embedded systems, web apps/servers
- Runs on many platforms (Windows, Mac, Linux, cell phones...)



Customer, money, account

https://spectrum.ieee.org/ns/IEEE_TPL_2017/index/2017/1/0/0/1/1/50/1/50/1/50/1/30/1/30/1/30/1/20/1/20/1/5/1/5/1/20/1/100/

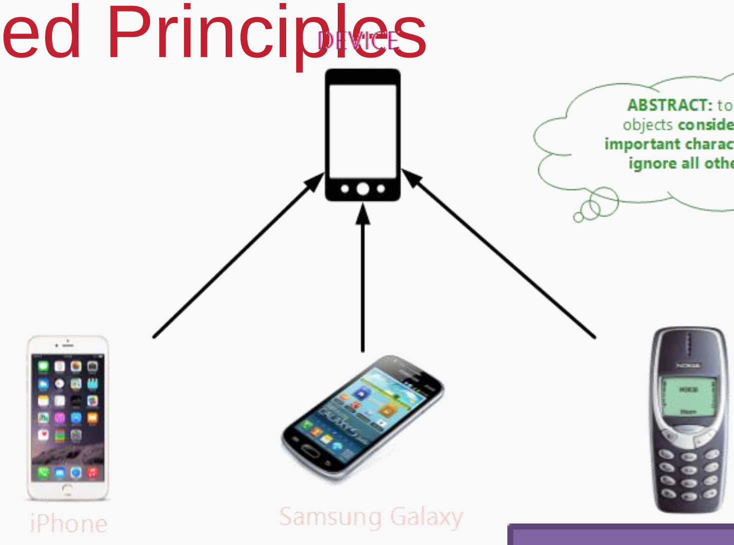


So what are the Top Ten Languages for the typical *Spectrum* reader?

Language Rank	Types	Spectrum Ranking
1. Python	🌐 🖥	100.0
2. C	📱 🖥 ▣	99.7
3. Java	🌐 📱 🖥	99.5
4. C++	📱 🖥 ▣	97.1
5. C#	🌐 📱 🖥	87.7
6. R	🖥	87.7
7. JavaScript	🌐 📱	85.6
8. PHP	🌐	81.2
9. Go	🌐 🖥	75.1
10. Swift	📱 🖥	73.7

Object Oriented Principles

- Abstraction



ABSTRACT: to deal with objects **considering their important characteristics** and **ignore all other** details.

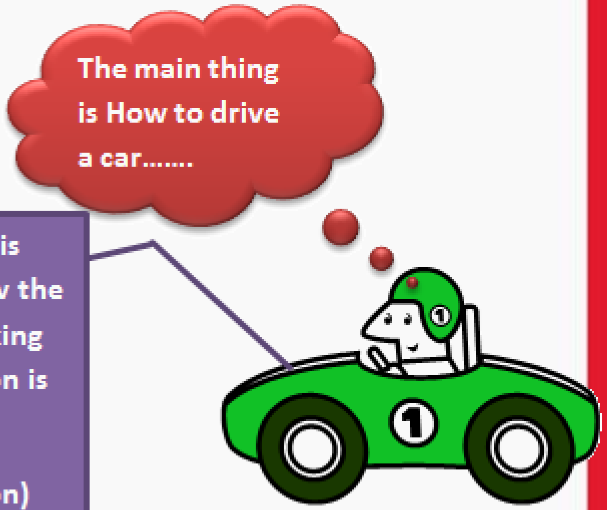
- Encapsulation

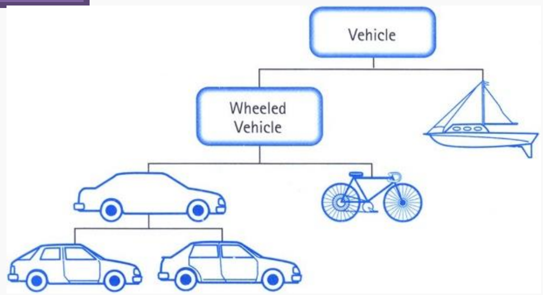
Object's data cannot be accessed directly from outside the object

- Inheritance - "Is-a" relationship
- Polymorphism – objects with the same specification have different implementation

How the car is moving and how the engine is working this information is hidden.

(Encapsulation)





PDP LOGISTICS

Content Overview

- We will be using Java
- Next week – Whirlwind Tour of Java
  - After that we assume Java competence
- Advanced OO Design Principles and Patterns
- Data Structures and Algorithms
- Concurrency
- Functional programming
- Networking and distribution

Web Site

https://cs5010pdp2017fall.github.io/

Lectures

- Each lecture will be a mix of presentation and class exercises
- We’ll expect you to have done the recommended reading associated with each week

Assignments

- 9 programming assignments
  - 6x1 week
  - 3x2 weeks (these are obviously harder!)
- First 4 assignments are solo
- Last 5 are in pairs
  - We choose the partners ☺

Assessment

- Code submission due Mondays at 6pm on weeks of deadlines
- Tuesday – walkthroughs held where you explain your code to TAs/Professors
- Logistics for walkthroughs coming soon

Assessment Grade

- 30% - correctness
  - Pass tests
  - Produce correct output
- 20% - presentation of solution
- 50% - design
- See web site for specifics.

Professors – You have 4 ☺

Ian

Tamara

Adrienne

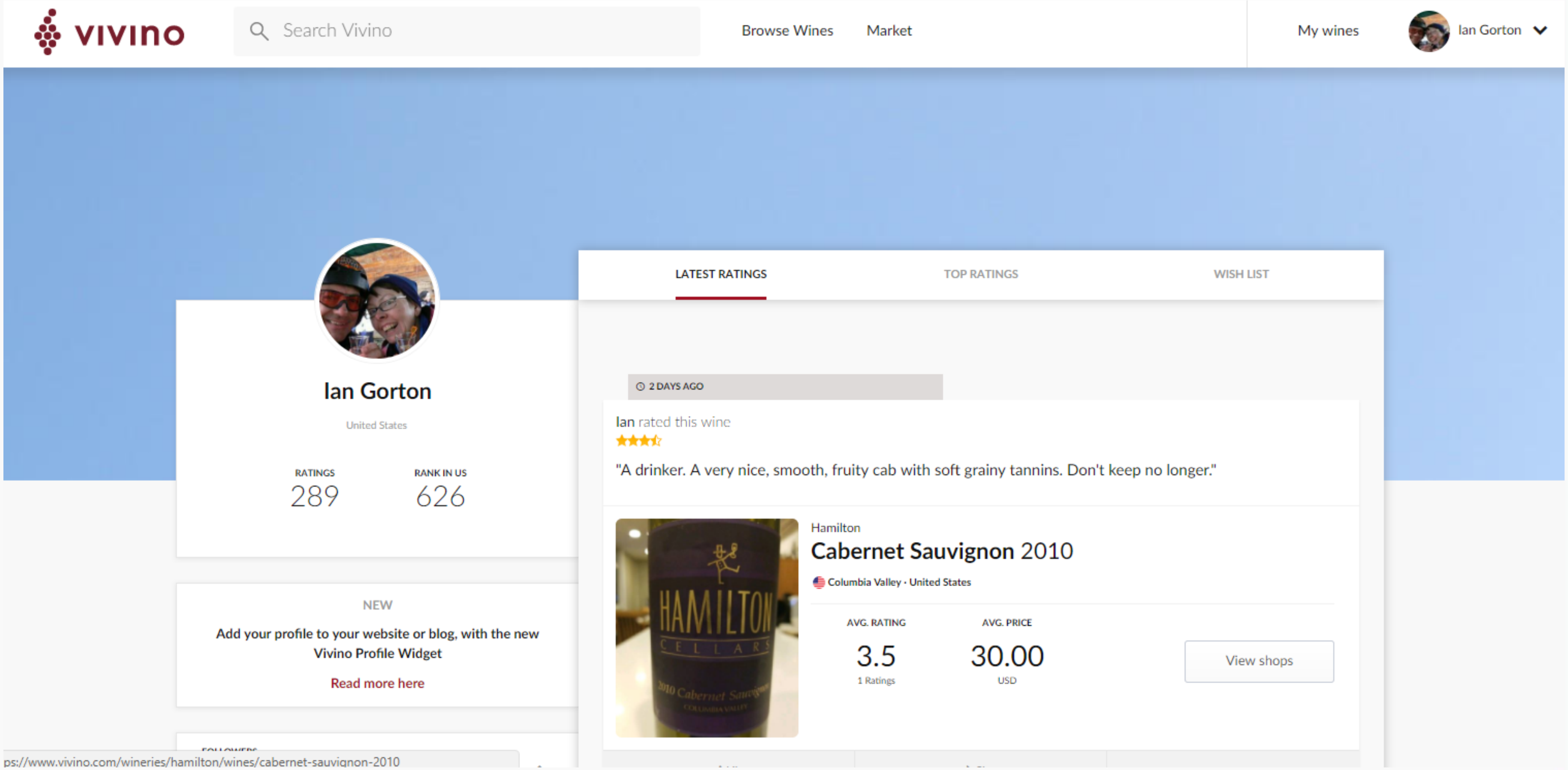
Maria

And many TAs …..

TEACHING ASSISTANTS
WERE CREATED BECAUSE
TEACHERS
NEED HEROES TOO

CLASS EXERCISE

Vivino.com



Ian Gorton

United States

RATINGS	RANK IN US
289	626

NEW

Add your profile to your website or blog, with the new Vivino Profile Widget

Read more here

2 DAYS AGO

Ian rated this wine ★★★★½

"A drinker. A very nice, smooth, fruity cab with soft grainy tannins. Don't keep no longer."

Hamilton

Cabernet Sauvignon 2010

Columbia Valley · United States

AVG. RATING	AVG. PRICE
3.5 1 Ratings	30.00 USD

ps://www.vivino.com/wineries/hamilton/wines/cabernet-sauvignon-2010

Vivino

- Database of knowledge about wine worldwide
  - Wine producers
  - The wines they produce
  - Retailers that sell each wine
  - Classification of all wines into ~250 categories
- Users rate wines they drink
  - Rating and comments
  - Other users can ‘Like’ ratings
  - Users can follow others (followed by/followers)
  - Users get rankings based on number of reviews

Exercise

- In groups of 2 or 3, discuss:
  - What are the major abstractions in this problems domain
    - E.g. Classes
  - How are they are related?
    - Associations/compositions
    - Dependencies (one way/two way?)
- Remember – this is a client server app
  - Server lives ‘in the cloud’, shared by ….
  - (Typically) mobile client apps

DESIGN BY CONTRACT

Programming ‘in the Small’ versus ‘in the large’

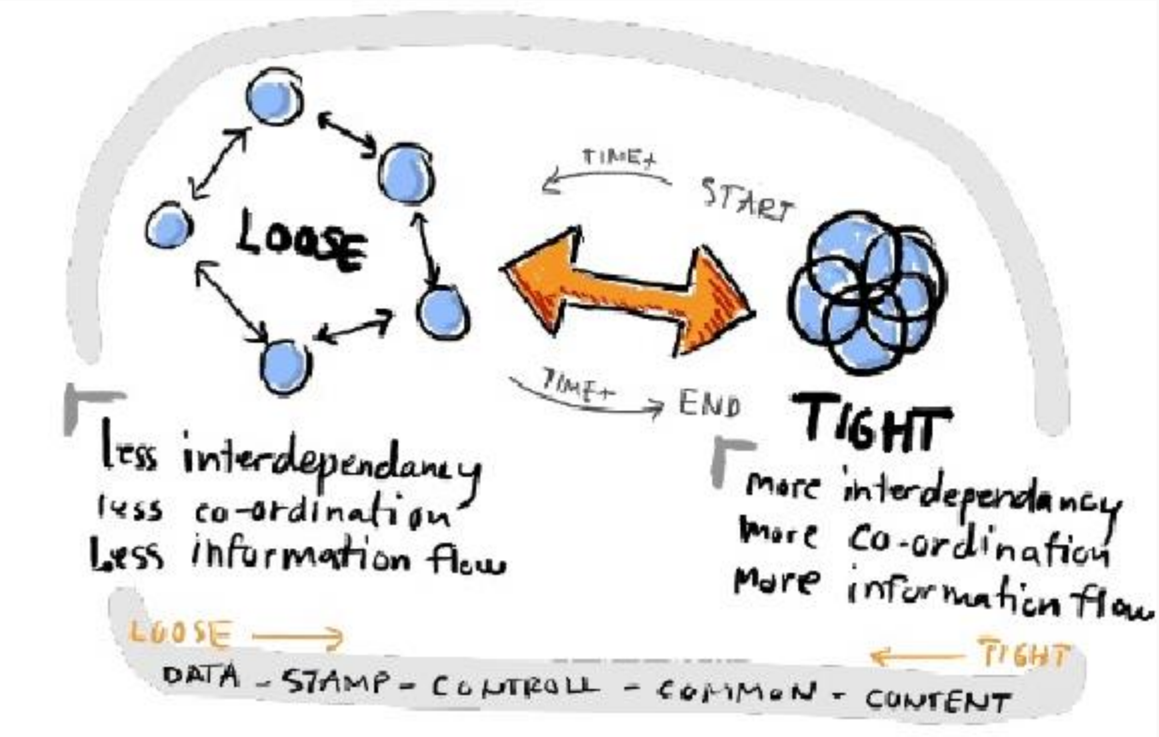
- Small programs (e.g a few hundred LoCs)
  - Easy to write
  - Easy to fully understand
  - Easy to change

- Big programs (e.g. 1 million LoCs)
  - Hard to write
  - Impossible to fully understand
  - Hard to change

why?
why?
why?
why?
why?

The Ripple Effect

- A seemingly simple change leads to many unexpected changes
- The parts of the programs are dependent upon each
  - Change one, must change many
  - Tightly coupled
- The number of interactions/dependencies makes code unmanageable


LOOSE
TIME+
START
TIME+
END
TIGHT
less interdependancy
less co-ordination
Less information flow
more interdependancy
more co-ordination
More information flow
LOOSE
TIGHT
DATA - STAMP - CONTROLL - COMMON - CONTENT

Modularity

- Decompose the problem into parts
  - Modules, packages, classes, components, etc
- Create minimal dependencies between the parts
  - Loosely coupled, limit ripple effect
- Dependencies based on specifications
  - Hide implementation details from other parts
  - Details can change as long as specification not violated

Specification

- Defines a contract between a ‘using’ class and a ‘used’ class
  - E.g client, server
- Describes expectations of each other
  - What data the client must pass to the server
  - What effects passing the expected data will have on the server
  - What the server will return to the client
  - What conditions can be guaranteed to hold after the request is complete

Why not just read the code?

- Code is complicated!!
  - And changes
- Specification concisely tells the client what the code does, not how it does it
- Specification abstracts away unnecessary details
  - Easy to understand, clear and unambiguous
  - Specifies what the client can always depend on when using the module

Elements of a contract

- Preconditions of the module
  - What conditions the module requests from its clients
  - Check upon entry to module
- Postconditions of the module
  - What guarantees the module gives to clients
  - What conditions must hold for all objects of this module if implemented correctly

Violations

- Precondition violation
  - Blame the client
- Postcondition violation
  - Blame the server
  - In reality we have a bug

Example – A fixed stack

- Push(T t)
  - Precondition: stack is not full
  - Postcondition: numElem = numElem’+1
  - Stack[numElem] = t
  - numElem >= 0 and < max
- T Pop()
  - Precondition: stack is not empty
  - Postcondition: numElem =numElem’-1
  - Postcondition: Returns Stack[numElem’]
  - numElem >= 0 and <=max

Module Invariant

When to check?

- Preconditions
  - Upon module entry
    - Or as early as feasible
  - Throw an exception if violated
- Postconditions
  - Just before returning
  - Violations indicate errors in the module
    - Useful for debugging
    - In production?

Using Javadoc

- Javadoc can be used for writing specification
  - Method signature
  - Text description of method
  - @param: description of what gets passed in
  - @return; description of what gets returned
  - @throws: exceptions that may occur

http://www.oracle.com/technetwork/articles/java/index-137868.html

Example

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists.
 *
 * @param  url  an absolute URL giving the base location of the image
 * @param  name the location of the image, relative to the url argument
 * @return      the image at the specified URL
*/
 public Image getImage(URL url, String name) {
        try {
            return getImage(new URL(url, name));
        } catch (MalformedURLException e) {
            return null;
        }
 }
```

To specify a contract, we'll add …

- @precondition: specify all obligations on the client. These must hold before method call
- @postcondition: specify conditions that must hold at end of method for correct execution

Example (not correct Javadoc for brevity)

```
static void listAdd(List lst1, List lst2)
@precondition: lst1 and lst2 are non-null.
@precondition: lst1 and lst2 are the same size.
@postcondtion: lst1[i] = lst1[i] + lst2[i]
@return none
```

One for you ….

```
Public class Vivino {
    public Credentials login(String user, String pwd) {}
    public WineList getMyWines (Credentials user) {}
    public Receipt buyWines(WineList selectedWines) {}
    public bool payForWine(CreditCard cc) {}
}
```

What Next

- Get your Java IDE environment configured
- Become a Java expert
  - You have a week ☺
  - Your bedtime reading
    - Joshua Bloch, Effective Java 2nd Edition

What Next (2)

- First assignment released on Friday
- Lecture next week – Whirlwind Java tour
- First assignment deadline:
  - Monday 6pm Sept 18th
- First Walkthroughs
  - Tuesday 19th Sept
  - Time slots all day, sign up ‘sheet’ coming soon

time for questions