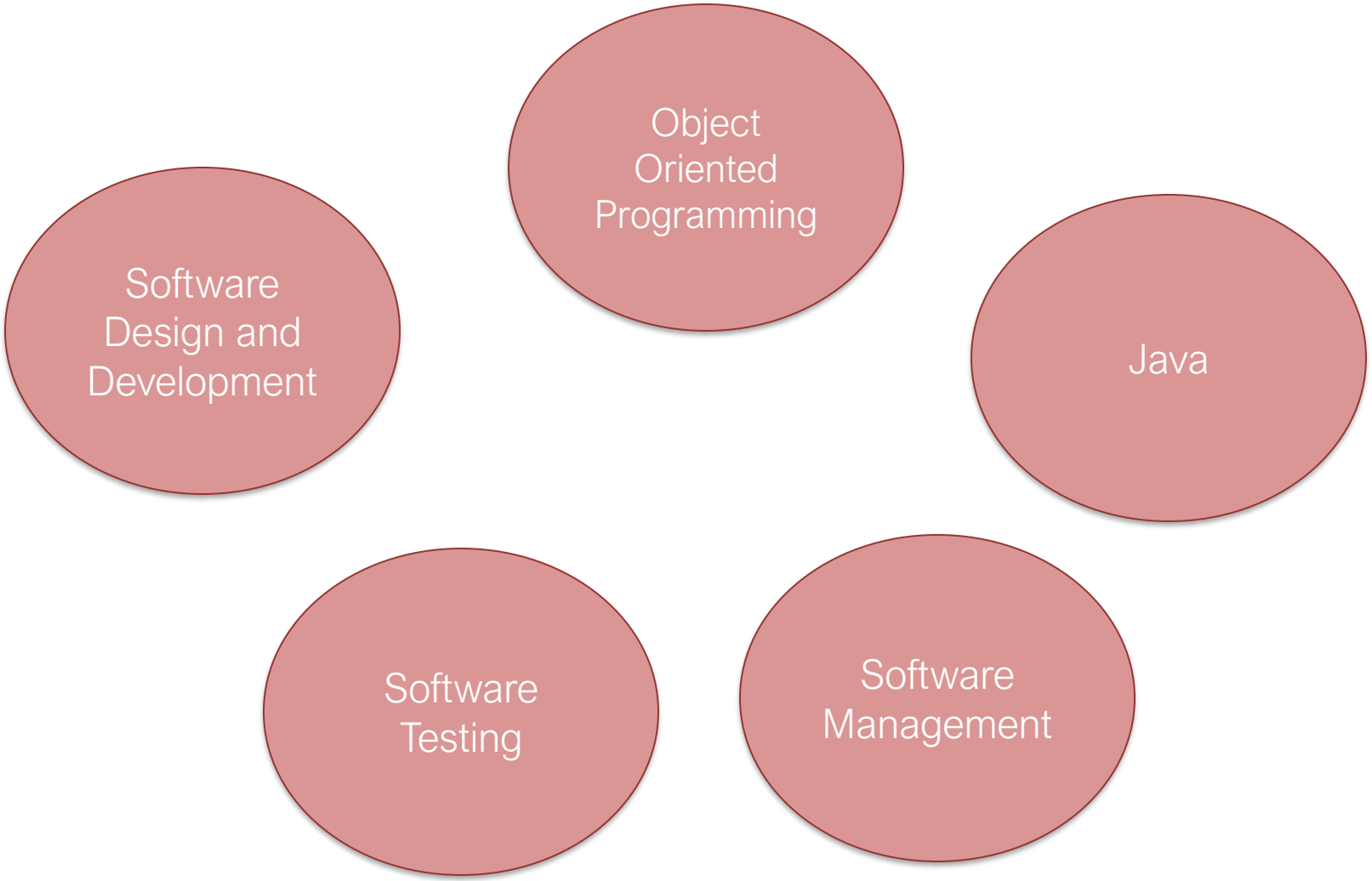
CS 5010:
Program Design Paradigm
Fall 2022

Brian Cross

b.cross@northeastern.edu

One More Time: What is CS 5010, Fall 2022?



HIGH QUALITY SOFTWARE

High Quality Software

- High quality software should be:
  1. Correct
  2. Comprehensible
  3. Modifiable

Correctness

Requirements as specified in the Software Requirements Spec (SRS) document have been correctly implemented.

Results are accurate

High Quality Software: Correct

- Meet functional requirements
  - Pass test cases
- But 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

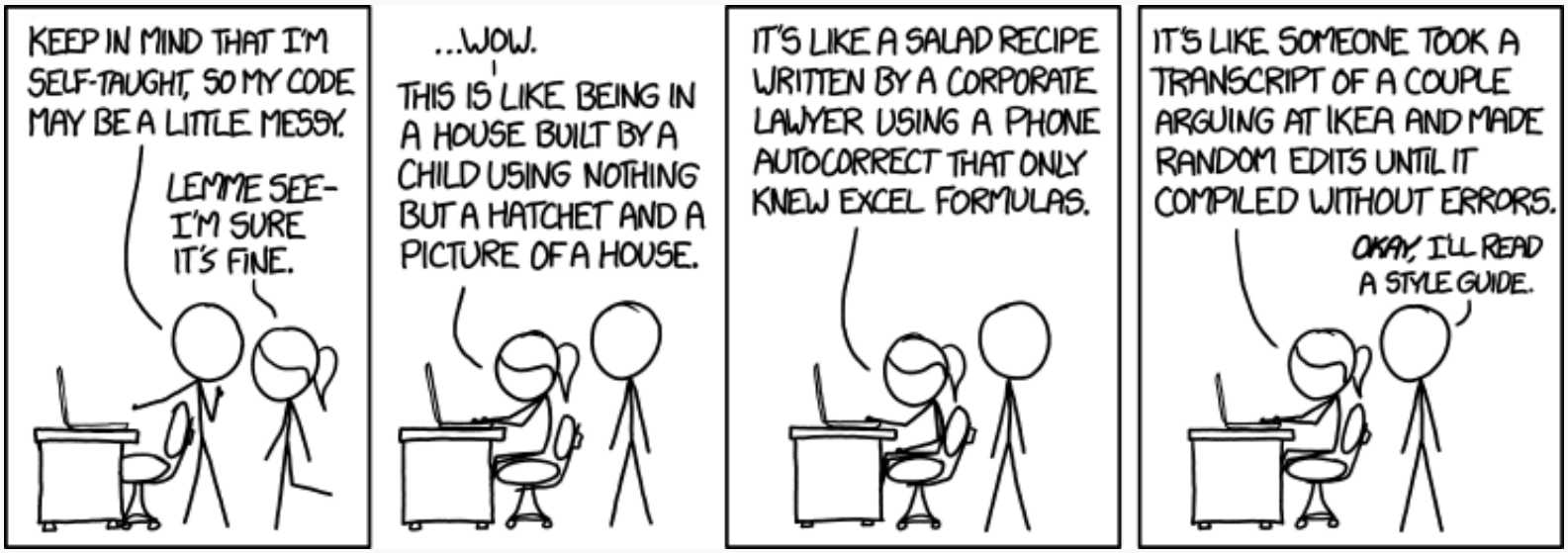
High Quality Software: Comprehensible

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

High Quality Software: Comprehensible

High Quality Software: Comprehensible

High Quality Software: Comprehensible



High Quality Software: Comprehensible

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

High Quality Software: 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

The End Goal – Software Engineer

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 design, tests
  - External: documentation, style
  - Higher-level: e.g., system architecture
- Software quality is important in this class and in the profession (but it doesn't happen over night)

Software Engineering and Practice

PROGRAMMING PARADIGMS

Programming Paradigms

- Programming paradigm - general approach used to implement a program
  - A programming style that indicates the approach on how a solution is implemented in a programming language
- Programming languages are typically designed with at least one paradigm in mind
  - But a language can typically accommodate more than one programming style
  - It is also possible to use one programming style to encode another

Popular Programming Paradigms

- Imperative (procedural) paradigm
- Applicative (functional) paradigm
- Object oriented paradigm
- Logical programming paradigm
- Event-driven paradigm
- Meta-programming

Imperative Programming Paradigm

- Imperative programs are made up of procedures and data
  - Procedures are made up of a sequence of statements
  - Each statement may alter the existing data in place (typically this means there is no return value from our procedure)

- Conceptually, we can then think of an imperative program's execution as a machine that executes each statement in the appropriate sequence, and a store that holds all of the data that each statement will alter (or mutate)

- Some popular procedural languages: C, Pascal, and Ada

Functional Programming Paradigm

- Functional programs are made up of functions and data as values
- Functions are made up of one expression
- A function takes input values and returns output values

- Conceptually, a functional program's execution is an evaluation, much like the evaluation process in simple mathematical expressions

- Some popular functional languages: Lisp, Scheme, ML, Haskell

Object Oriented Programming Paradigm

- OO programs are made up of objects that have purpose and may communicate with each other by messages

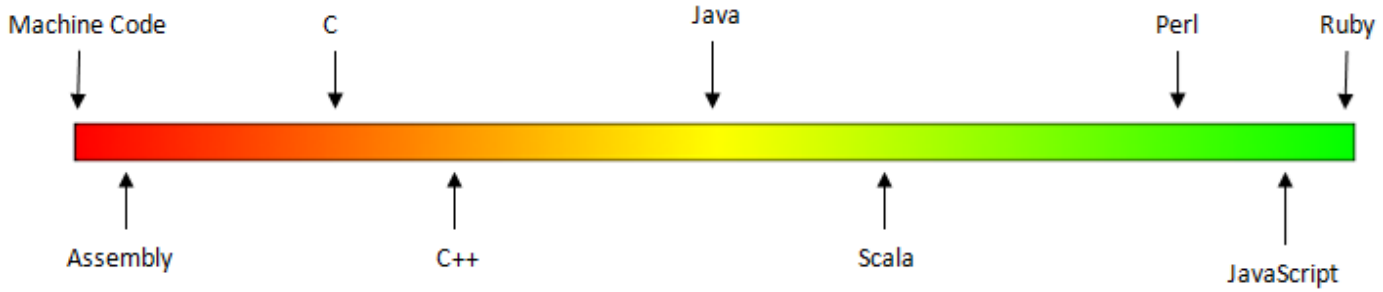
- Class-based OO program - notion of a class that is made up of fields (data) and methods (messages that this class understands)
  - From a class, we can then create an object that has its own set of fields but shares the same methods
  - A method is made up of a sequence of statements and/or expressions that may manipulate the object's fields and may return an object as a result of executing that method

Some popular OOD languages: Java, C++ , C#, Eiffel, Smalltalk

Other Programming Paradigms

- Logical Paradigm
  - Uses formal mathematic logic (i.e. set theory)
  - Languages: Prolog, Datalog
- Event-driven Paradigm
  - Flow of program determined by events
  - Graphical user interfaces
  - Languages: Java, C#, JavaScript, Visual Basic, etc
- Meta-programming
  - Self-modifying code
  - Languages: LISP, Prolog

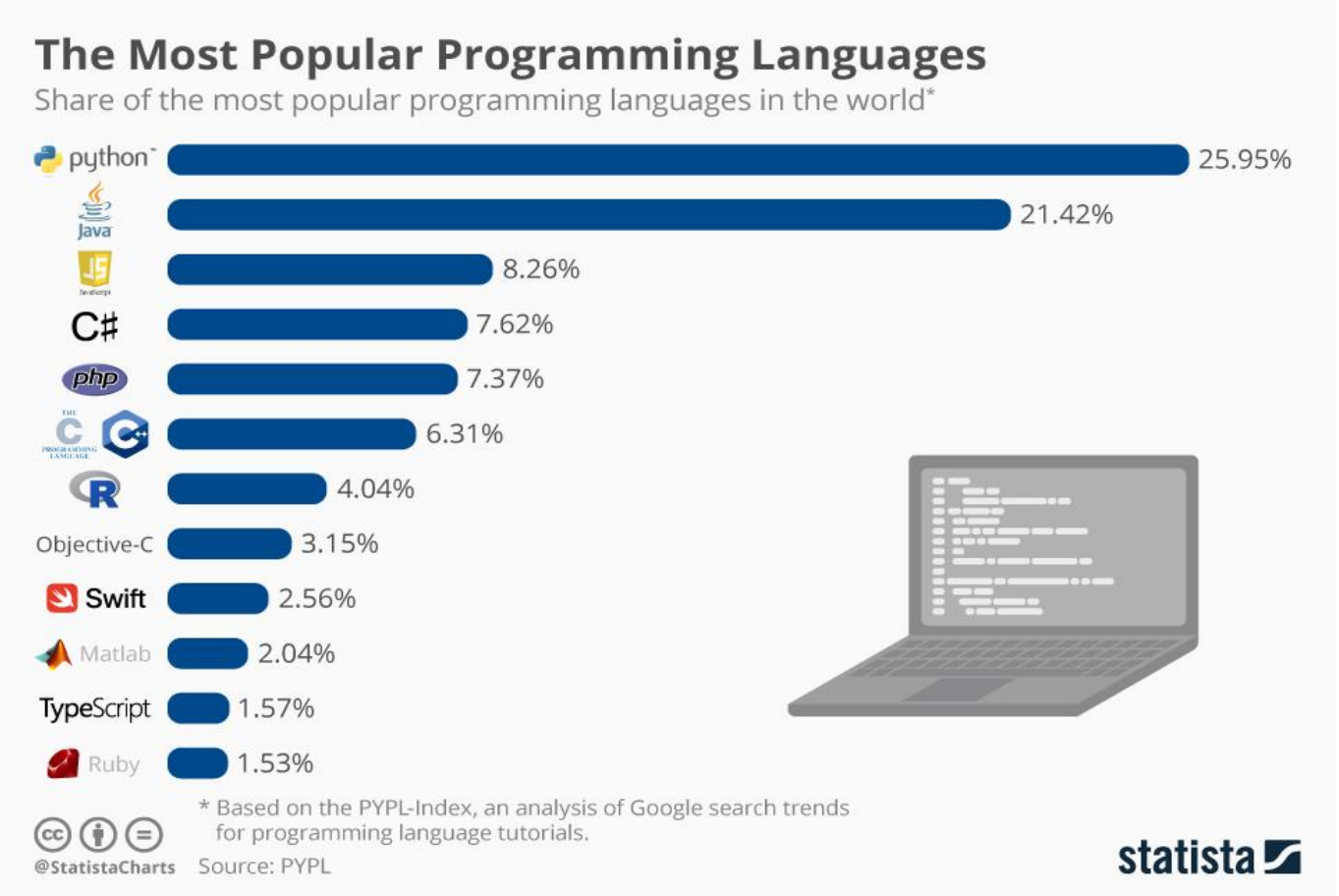
Some Modern Languages



- Procedural languages: programs are a series of command
  - 1970 - Pascal - designed for education
  - 1972 - C - low-level, operating systems and device drivers

- Object-oriented languages: programs interact using "objects"
  - 1985 - C++
  - 1995 - Java
    - Designed for embedded systems, web apps/servers
    - Runs on many platforms (Windows, Mac, Linux, cell phones...)

Most Popular Programming Languages



[Pictures credit: https://https://www.statista.com/chart/16567/popular-programming-languages/]

OBJECT ORIENTED DESIGN AND ANALYSIS

Objects and Classes

- Object – an entity consisting of states and behavior
  - States stored in variables/fields
  - Behavior represented through methods

- Class – template/blueprint describing the states and the behavior that an object of that type supports

Object Oriented Design Principles

- Inheritance
- Abstraction
- Encapsulation
- Information hiding
- Polymorphism

OOD Principles: Inheritance

[Pictures credit: https://medium.com/java-for-absolute-dummies/inheritance-in-java-programming-39176e0016f3]

Inheritance – the ability for a class to extend or override functionality (states and behavior) of other classes

OOD Principles: Abstraction

[Pictures credit:https://www.pinterest.com/pin/423408802450668522/]

Abstraction - the ability to segregate implementation from an interface

OOD Principles: Encapsulation

[Pictures credit:http://small-pets.lovetoknow.com/reptiles-amphibians/names-pet-turtles]

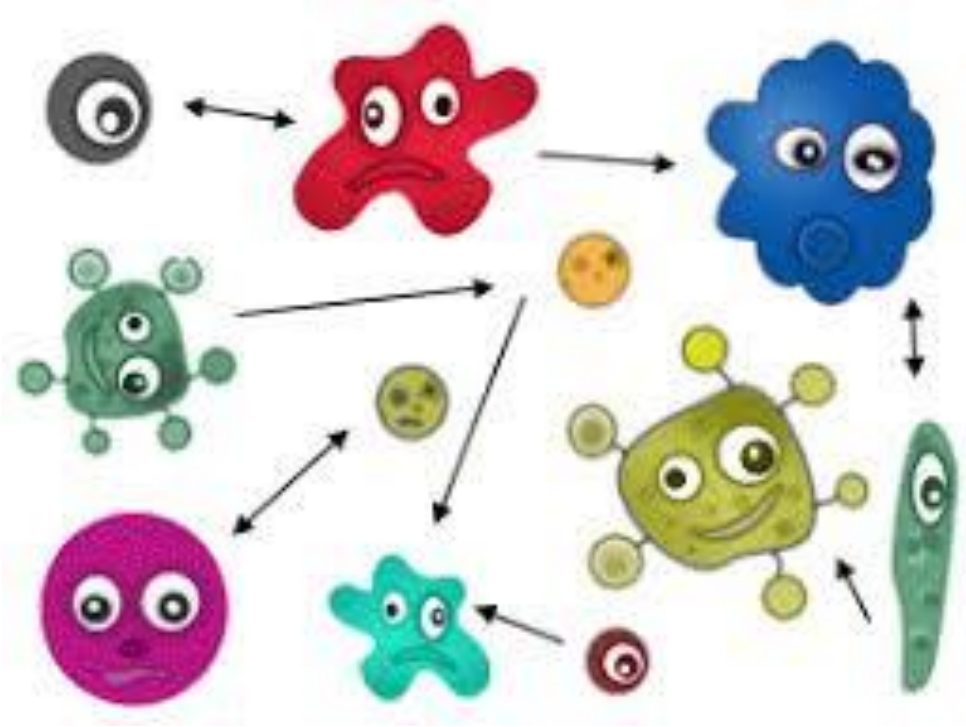
Encapsulation idea – data types and methods operating on that data coupled within an object/class

OOD Principles: Information Hiding

[Pictures credit:http://www.sickchirpse.com/kebab-shop-owner-three-days-hiding-london-terror-attacks/]

Information hiding idea – expose only the necessary functionality (through interface), and hide everything else

OOD Principles: Polymorphism



[Pictures credit: http://www.thewindowsclub.com/polymorphic-virus]

Polymorphism – the ability to define different classes and methods as having the same name but taking different data types

CLASS EXERCISE

Class Exercise

- Example: food delivery mobile app

[Pictures credit: http://www.charlottemagazine.com/Charlotte-Magazine/October-2016/Rating-Charlottes-Food-Delivery-Services/]

Class Exercise

- In groups of 2 or 3, discuss:
  - What are the major abstractions in this problem's domain
    - E.g. Classes
  - How are they related?
    - Associations/compositions
    - Dependencies (one way/two way?)

[Pictures credit: http://www.charlottemagazine.com/Charlotte-Magazine/October-2016/Rating-Charlottes-Food-Delivery-Services/]

Class Exercise

- Example: food delivery mobile app

- Objects:
  - Customers
  - Drivers
  - Restaurants
  - Menus
  - Order

[Pictures credit: http://www.charlottemagazine.com/Charlotte-Magazine/October-2016/Rating-Charlottes-Food-Delivery-Services/]

Class Exercise

- Example: food delivery mobile app

- Customers - properties:
  - Name
  - Address
  - Phone number
  - E-mail address
  - Order

[Pictures credit: http://www.charlottemagazine.com/Charlotte-Magazine/October-2016/Rating-Charlottes-Food-Delivery-Services/]

Class Exercise

- Example: food delivery mobile app

- Driver - properties:
  - Hours of operation
  - Current location
  - Next delivery

[Pictures credit: http://www.charlottemagazine.com/Charlotte-Magazine/October-2016/Rating-Charlottes-Food-Delivery-Services/]

Class Exercise

- Example: food delivery mobile app

- Restaurant - properties:
  - Cuisine
  - Address
  - Hours of operation (open/close)
  - “Priciness” ($, $$, $$$)
  - Customer rating
  - Menu

[Pictures credit: http://www.charlottemagazine.com/Charlotte-Magazine/October-2016/Rating-Charlottes-Food-Delivery-Services/]

Class Exercise

- Example: food delivery mobile app

- Menu - properties:
  - Offered meals
  - Meal prices

[Pictures credit: http://www.charlottemagazine.com/Charlotte-Magazine/October-2016/Rating-Charlottes-Food-Delivery-Services/]

Class Exercise

- Example: food delivery mobile app

- Order - properties:
  - Total quantity
  - Total price
  - Paid/Not paid

[Pictures credit: http://www.charlottemagazine.com/Charlotte-Magazine/October-2016/Rating-Charlottes-Food-Delivery-Services/]

Class Exercise

- Example: food delivery mobile app

- Customer - responsibilities:
  - Search for restaurants
  - Choose a restaurant
  - Check the menu
  - Select meals from the menu
  - Make an order
  - Pay for an order
  - Track an order
  - Cancel an order

[Pictures credit: http://www.charlottemagazine.com/Charlotte-Magazine/October-2016/Rating-Charlottes-Food-Delivery-Services/]

Class Exercise

- Example: food delivery mobile app

- Driver - responsibilities:
  - Receive an order
  - Deliver an order
  - Change paid/not paid status of an order

[Pictures credit: http://www.charlottemagazine.com/Charlotte-Magazine/October-2016/Rating-Charlottes-Food-Delivery-Services/]

Class Exercise

- Example: food delivery mobile app

- Restaurant - responsibilities:
  - Receive an order
  - Dispatch an order to a driver
  - Track the order

[Pictures credit: http://www.charlottemagazine.com/Charlotte-Magazine/October-2016/Rating-Charlottes-Food-Delivery-Services/]

Class Exercise

- Example: food delivery mobile app

- Menu - responsibilities:
  - Update meals
  - Update meal prices

[Pictures credit: http://www.charlottemagazine.com/Charlotte-Magazine/October-2016/Rating-Charlottes-Food-Delivery-Services/]

Class Exercise

- Example: food delivery mobile app

- Order - responsibilities:
  - Update quantity
  - Cancel order
  - Track order status

[Pictures credit: http://www.charlottemagazine.com/Charlotte-Magazine/October-2016/Rating-Charlottes-Food-Delivery-Services/]

DESIGN BY CONTRACT

Programming “In the Small” vs. “In the Large”

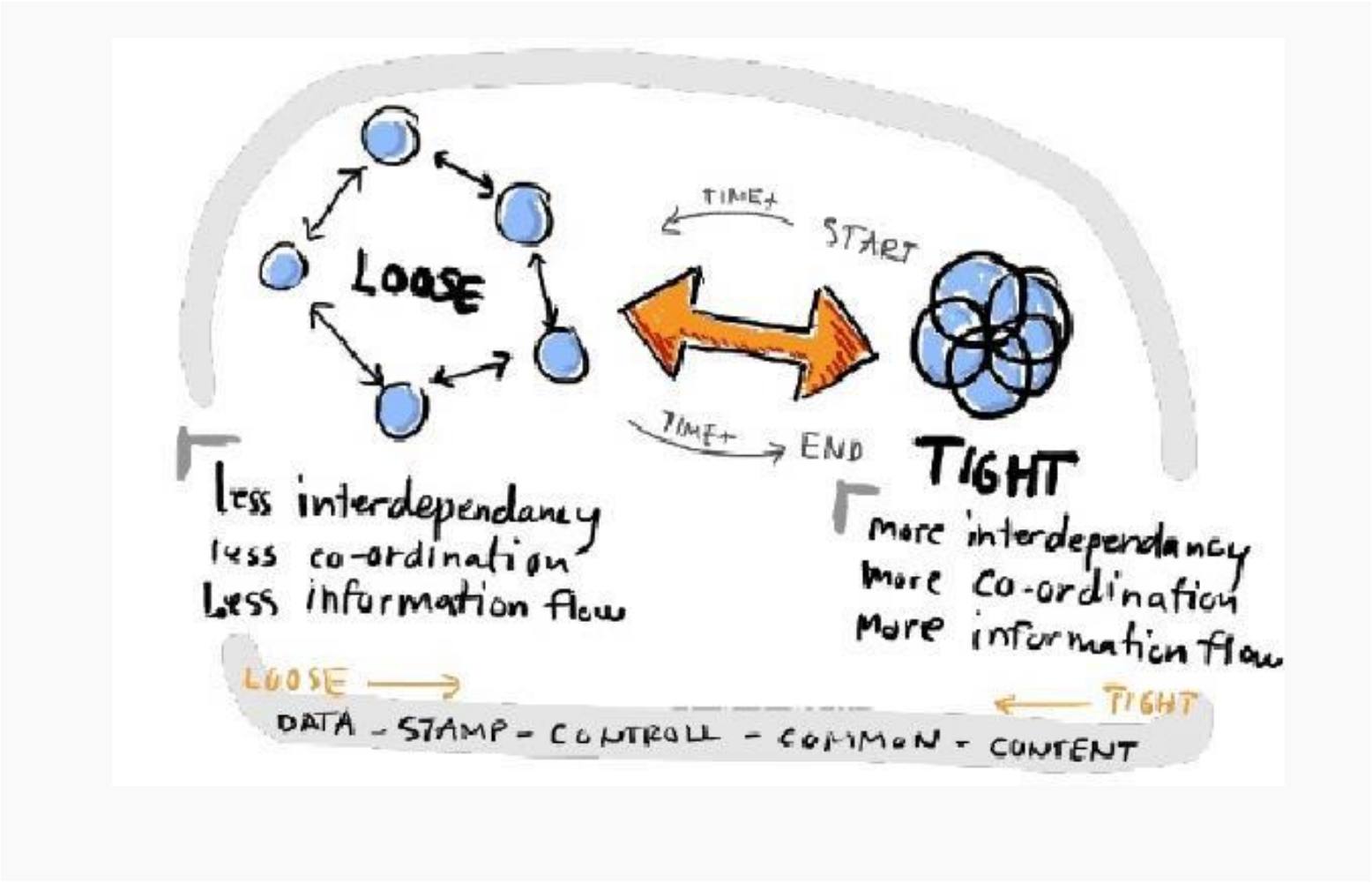
- Small programs - few hundred LoCs
- Easy to write – Easy to fully understand
- Easy to change

- Big programs -1 million LoCs
- Hard to write
- Often difficult to fully understand
- Hard to change

The Ripple Effect

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

The Ripple Effect



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 Code?

- Code is complicated!!
- And it 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 Size 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

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 and valid values
  - `@return`; description of what gets returned and guaranteed values
  - `@throws`: exceptions that may occur

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

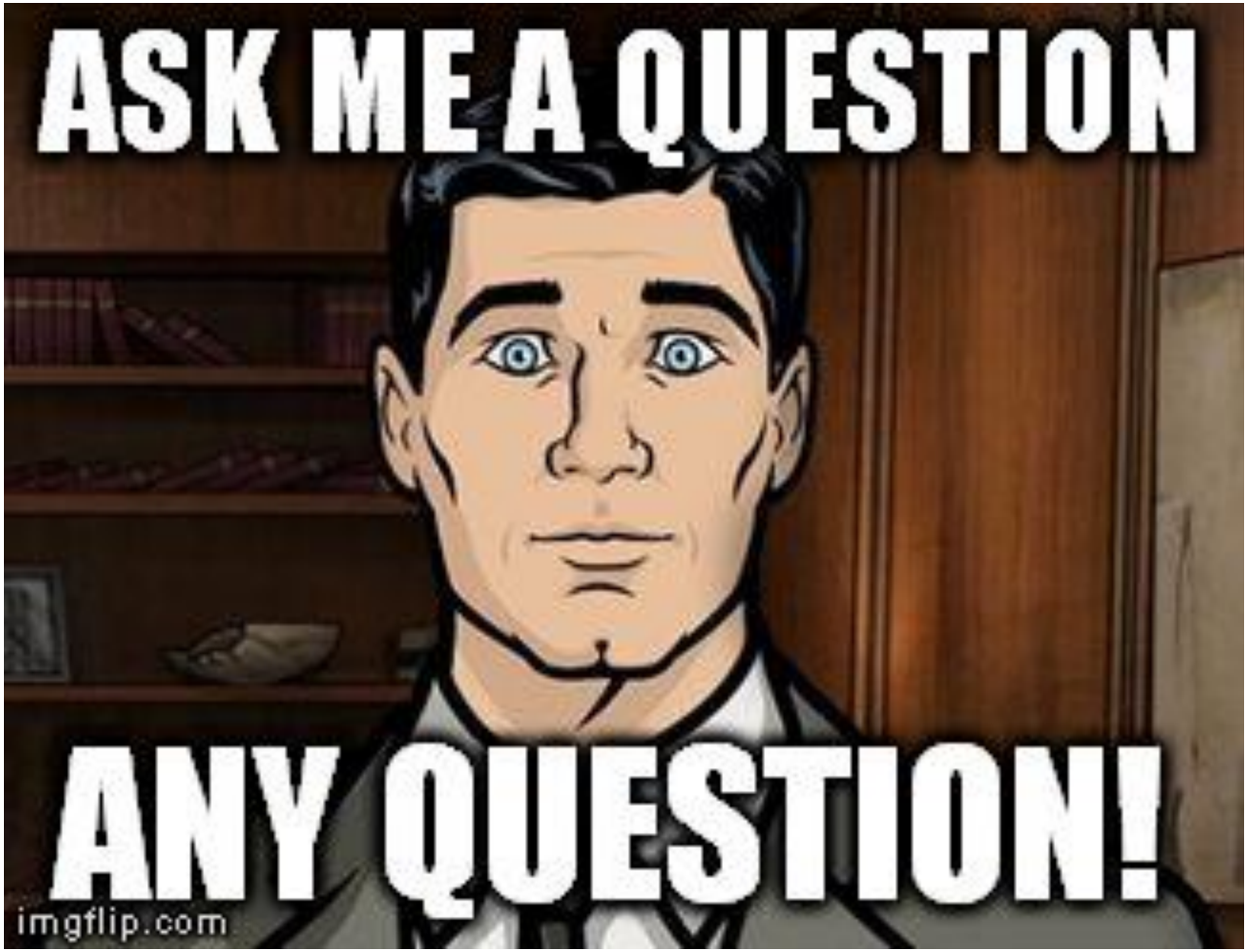
Example

```
/**
*  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.
*  @param url an absolute URL giving the base location of the image:
not null
*  @param name the location of the image, relative to the url argument:
not null
*  @return the image at the specified URL - valid Image object
*/

public Image getImage(URL url, String name) {
    if (url == null)
        throw new IllegalArgumentException("URL cannot be null);

    // same as above for name - not shown for brevity
    try {
        return getImage(new URL(url, name));
    } catch (MalformedURLException e) {
        return null; }
}
```

Your Questions



[Meme credit: imgflip.com]

What's Next?

- Take class survey (will post on Piazza)
- Get your Java IDE environment configured
- Become a Java expert
  - You have a week☺
- Your bedtime reading
  - Joshua Bloch, Effective Java 2nd/3rd Edition

References and Reading Material

- Java Getting Started (https://docs.oracle.com/javase/tutorial/getStarted/index.html)
- Object-Oriented Programming Concepts (https://docs.oracle.com/javase/tutorial/java/concepts/index.html)
- Language Basics (https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html)
- How to Design Classes (HtDC), Chapters 1-3