CS 5004

LECTURE 1: OVERVIEW

SIMPLE CLASSES, OBJECTS & TESTING

KEITH BAGLEY

SPRING 2022

Northeastern University
Khoury College of Computer Sciences

440 Huntington Avenue ▪ 202 West Village H ▪ Boston, MA 02115 ▪ T 617.373.2462 ▪ khoury.northeastern.edu

AGENDA

- Welcome Back!
  - Intro
- Big Ideas
- Course Themes
- Course Logistics
  - Review of the Syllabus, CANVAS, Office Hours & Other Course Items
- Q & A

REMEMBER THIS?

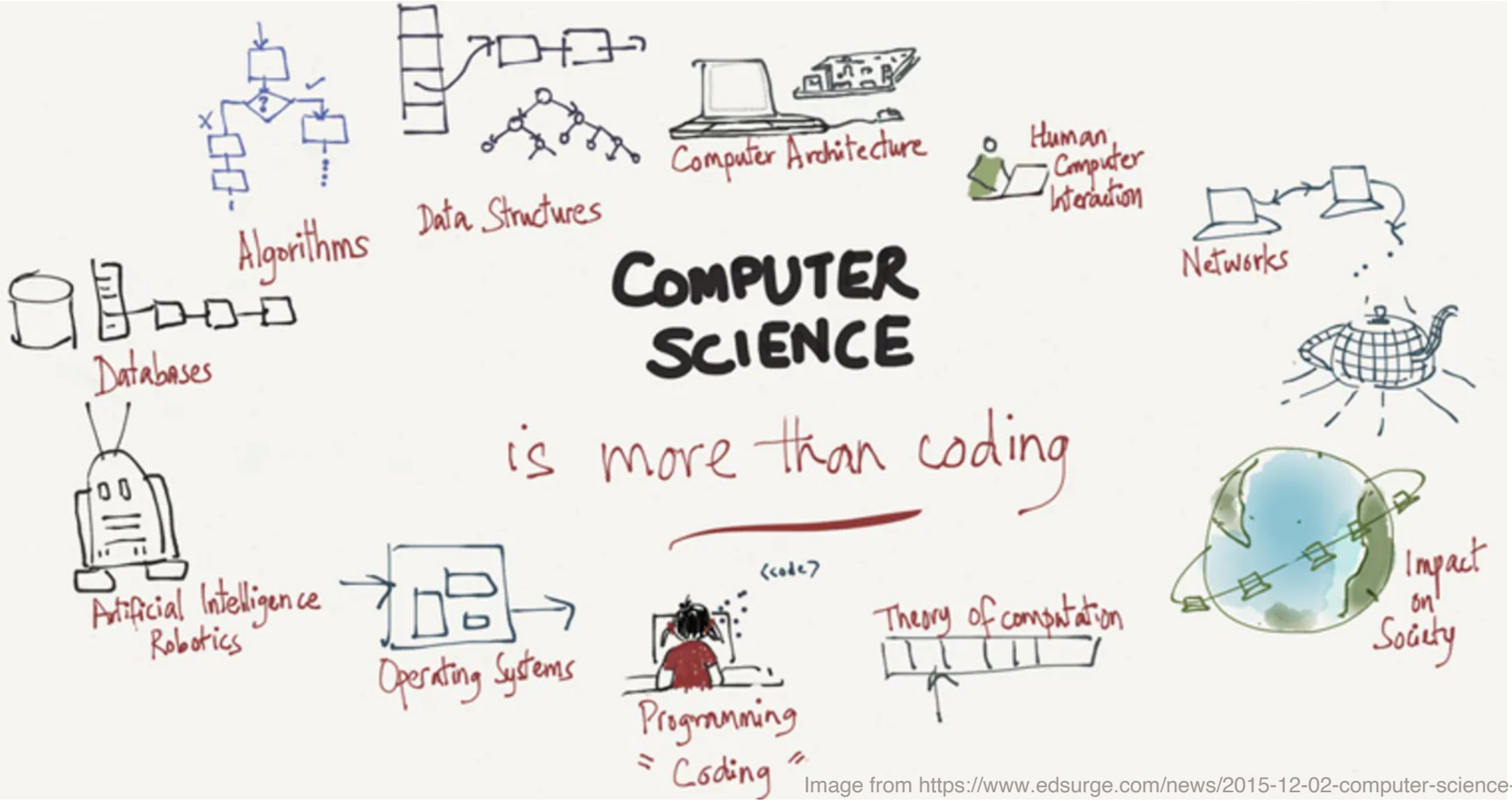


Image from https://www.edsurge.com/news/2015-12-02-computer-science-goes-beyond-coding
Courtesy of Sheena Vaidyanathan

A BIT OF ART & SCIENCE

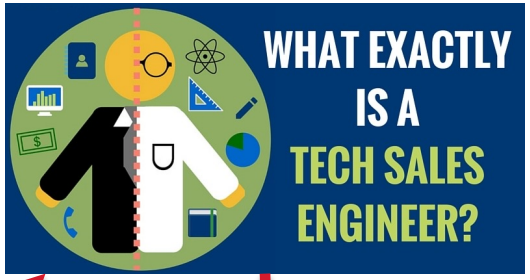
WHO AM I?

- Keith Bagley, Ph.D., D.Min.
- Associate Clinical Professor, Director – MS ALIGN,

Plus…
- Industry
- Husband
- Father
- Multi-vocational
- Pet owner
- Community Active
- Son!

I'm more than a title @ NU
So are you!
Embrace your holistic self

FOR THOSE WHO DON'T KNOW ME



NEW NORMAL – ENDURING TIPS FOR SUCCESS

- COVID-19 introduced challenges & opportunities
  - "New Normal" – Online by necessity, not by design
  - Opportunity to re-evaluate some content delivery
  - Challenges "staying engaged" with studies & peers
- Remember who you are and why you're here.
  Do not let external circumstances alter your internal compass!
- Truism: The only thing "constant" is the fact that things will change
- Touchdowns are scored one play at a time. Move forward every day 🏈
  - Sorry Futbol fans…Goals are scored one kick at a time? ☺ ⚽
- Reach out for help early & often

WARM-UP

- What is your “nemesis food”?

(A) Candy

(B) Burgers

(C) Salad

Yes!

(D) Yes

WARM-UP

- Where would you like to go on vacation after the pandemic?

(A) Hawaii

(B) Carribbean

(C) Florida

Yes!

(D) Yes

WARM-UP

- Are you excited about CS 5004?

(A) Yes

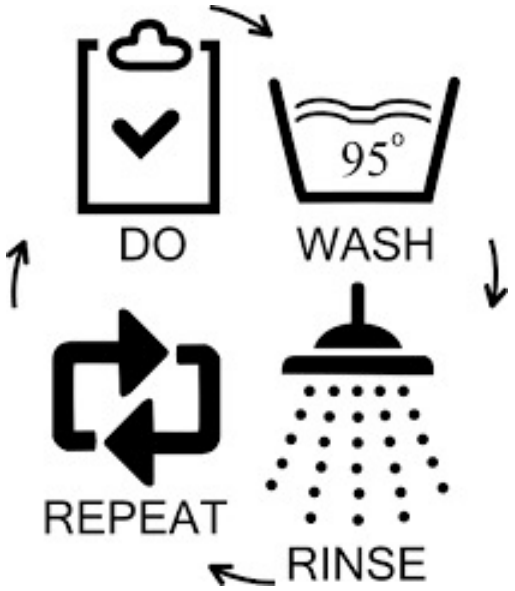
(B) Heck Yes

(C) Absolutely Yes

(D) Are you kidding me? YES!

WHAT WE DO - PLAN YOUR WEEK!

- Individual quizzes should be completed by the individual. No collaboration
- Labs are assigned on Mon/Tues – practice concepts from previous lecture
- HW released Wed, due a week from that Saturday
- Note: Every assignment (unless otherwise noted) is due on Saturday before 11:59pm!



HOMEWORK

- Individual for the first 2/3 of the course – no code sharing
- First few assignments are “prescriptive”; last few assignments are “open ended” with respect to what solution approaches you can use
- Grading
  - Automated Style checker
  - Solution correctness via Server-side tests
  - Self-evaluation
  - Manual grading

- Note: Homework & Lab assignments can be submitted as many times as you want before the deadline. **Coaching Self-evals can only be submitted ONCE!**

LATE POLICY

- THREE (3) No-questions asked late days for homework. Can only use 1 late day per homework

COURSE THEMES

- Abstraction
  - Ignoring certain details to focus on higher aspects (verb)
  - The product of the process of abstraction (noun)
- Object-Oriented vs. Procedural
  - We did mostly procedural in CS5001, with a touch of OO
- Testing & Debugging
- Functional Programming
- Java language
  - Similar to CS5001 & Python, we're using Java as the vehicle to learn concepts
- Design Patterns

LET'S GET CRACKIN'

WHERE ARE WE COMING FROM?

- If you know Python, or are coming to CS 5004 from CS 5001, you're familiar with:
  - a dynamically typed language
  - a language that doesn't require you to "look under the hood" often
  - a language that has nice features that handle mundane "housekeeping"
  - a hybrid language and system that supports multiple programming paradigms:
    - Procedural, Functional, Object-Oriented
  - a language that sounds dangerous, but is actually from a comedy troupe

  Python

WHERE ARE WE GOING?

- CS 5004 will immerse you in
  - a (more) statically-typed language
  - a language that requires a bit more attention to details
  - a language that has powerful features and that handles *some* mundane "housekeeping"
  - a fully object-oriented language & system
  - a language that sounds like a drink, but originally named after a tree (Oak)

  Java

PROCEDURAL APPROACH WITH FUNCTIONS

- Using our old friend Python to help us:

```
def calc_area(choice):
    ''' function do_area
        Input: choice - the type of shape selected by the user
        Returns: area of a shape, depending on the type chosen
        Does: Asks the user for shape's dimensions to cacluate area
    '''
    if choice == 'S':
        length = float(input('Enter the square length: '))
        return round((length * length),2)
    elif choice == 'C':
        radius = float(input('Enter the circle radius: '))
        return round((PI * radius**2),2)
    elif choice == 'T':
        base = float(input('Enter the triangle base: '))
        height = float(input('Enter the triangle height: '))
        return round((0.5 * base * height),2)
    return 0
```

```
def main():
    answer = ''
    while answer != 'Q':
        question = ('Select the Shape you want:\n' +
        ' S: Square\n C: Circle\n T: Triangle\n' +
        ' Q: Quit?\n Your answer: ')
        answer = menu(question)
        if answer == 'Invalid Choice':
            print(answer, '...please choose again')
        elif answer != 'Q':
            print('Area = ', calc_area(answer))
        else:
            print('Thanks for using my Shape program!')

main()
```

PROCEDURAL APPROACH: A PYTHON VIEW

- Using our old friend Python to help us:

```
def calc_area(choice):
    ''' function do_area
        Input: choice - the type of shape selected by the user
        Returns: area of a shape, depending on the type chosen
        Does: Asks the user for shape's dimensions to cacluate area
    '''
    if choice == 'S':
        length = float(input('Enter the square length: '))
        return round((length * length),2)
    elif choice == 'C':
        radius = float(input('Enter the circle radius: '))
        return round((PI * radius**2),2)
    elif choice == 'T':
        base = float(input('Enter the triangle base: '))
        height = float(input('Enter the triangle height: '))
        return round((0.5 * base * height),2)
    return 0
```

Functions and data are separate

I pass choice into calc_area() as outside data

Conditional code to determine data

”if” statement to determine what to do with the data

Handing back the data to a client

The function doesn’t “own” the data so control passes to someone else who might modify it, depending on the type of data it is

PROCEDURAL APPROACH

- Nothing is inherently wrong with a procedural approach; it's been used in computing for decades
- Some trade-off's with other programming paradigms
  - e.g. not "owning" the data, possible 'ripple effect' as conditionals are used to trigger behavior, etc.
- If you are familiar with Python, it is a hybrid language that allows multiple paradigms (including procedural) to be used

OBJECT ORIENTED APPROACH

- What if we took a different approach where:
  - The data and the functions that operate on that data were grouped together?
  - We could build our own abstractions for our programs and extend the system?
  - We could send conceptually similar “messages” to instances of these abstractions, and have them respond appropriately without needing a bunch of “if” statements?
- We would call that an **object-oriented** approach

OBJECT ORIENTED APPROACH

- Using our old friend Python again, since it supports OO too!

```
PI = 3.1415

'''
    Class: Square.
    It knows how to draw itself, calculate its area, compare to other
    Shapes (for equality) and calculate its perimeter. Some code
    duplication in these classes since we're not using inheritance, but
    since Python is a dynamically typed language we don't need inheritance
    to implement protocol/interfaces
'''
class Square:
    def __init__(self, length):
        self.length = length
        self.name = 'Square'
    def perimeter(self):
        return self.length * 4
    def area(self):
        return round(self.length**2,2)
    def __str__(self):
        return self.name + " with length of " + str(self.length)
    def __eq__(self, other):
        if self.area() == other.area():
            return True
        return False


def main():
    # construct random shapes and load them into a list.
    print("Hi there! Let's make some Shapes!")
    shapes = []
    for i in range(10):
        shapes.append(shape_factory())
    for each in shapes:
        print("{} : has an area of {}".format(each, each.area()))
        print()
```

Functions & data are grouped
Methods and attributes are owned by the class

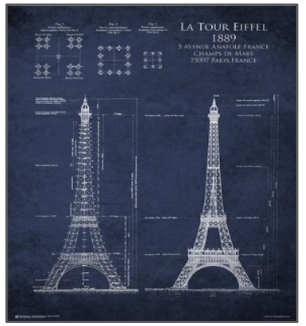
Better control of data access**
Class designer determines what data to share and expose**

No conditional code to determine data type
Objects know what they are, and what data they have

SOME TERMINOLOGY

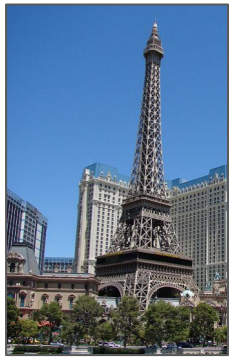
- **Class**: a blueprint for how to make an object. Classes describe the "thing" (abstraction) we're talking about – its:
  - **attributes** (data it owns/knows),
  - **operations** (behavior, what it knows how to do), and
  - **state and relationships** to other things
- **Object**: an "instance" (or one of the "things") from the class
  - Objects have attributes, operations and unique identity

blueprint



Paris

Las Vegas



Other (almost) synonyms you might hear:
Instance variables -> attributes
Methods or "member functions -> operations

SOME TERMINOLOGY

- **Encapsulation**: an object's data is packaged together with its corresponding operations
- **Information Hiding**: uses encapsulation to emphasizes the separation of external "visible" properties of an object from internal hidden properties, and restrict access by enforcing well-defined interfaces

Encapsulation

Information Hiding

OO CONCEPTS ARE GENERAL ACROSS LANGUAGES

- Depending on the language, some practitioners prefer one of the synonyms
  - E.g. “member function” rather than method
- Not all languages provide facilities for every concept
  - E.g. Python provides encapsulation, but only rudimentary (if any) information hiding

```
def main():
    # construct shapes and load them into a list. No 'if' statements
    # required here

    shapes = []
    for i in range(10):
        shapes.append(shape_factory())

    for each_shape in shapes:
        each_shape.introduce_yourself()
        print('--')
        print(each_shape.name) # encapsulation, not info hiding!
```

Not information hiding!

OBJECT ORIENTED APPROACH: JAVA

- Let's visit our new friend (or frenemy)

OBJECT ORIENTED APPROACH: JAVA

```java
/**
 * This class represents a person The person has a first name, last name and an year of birth.
 */
class Person {
  private String firstName;
  private String lastName;
  private int yearOfBirth;

  /**
   * Constructs a Person object and initializes it to the given first name, last name and year of
   * birth.
   *
   * @param firstName   the first name of this person
   * @param lastName    the last name of this person
   * @param yearOfBirth the year of birth of this person
   */

  public Person(String firstName, String lastName, int yearOfBirth) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.yearOfBirth = yearOfBirth;
  }

  /**
   * Get the first name of this person.
   *
   * @return the first name of this person
   */
  public String getFirstName() {
    return this.firstName;
  }
```

Encapsulation + Information Hiding

Notice the **public** & **private** access specifiers

Javadoc comments similar to PyDoc

Also: annotations & Javadoc tags w/ @

Same idea of objects “owning” their stuff

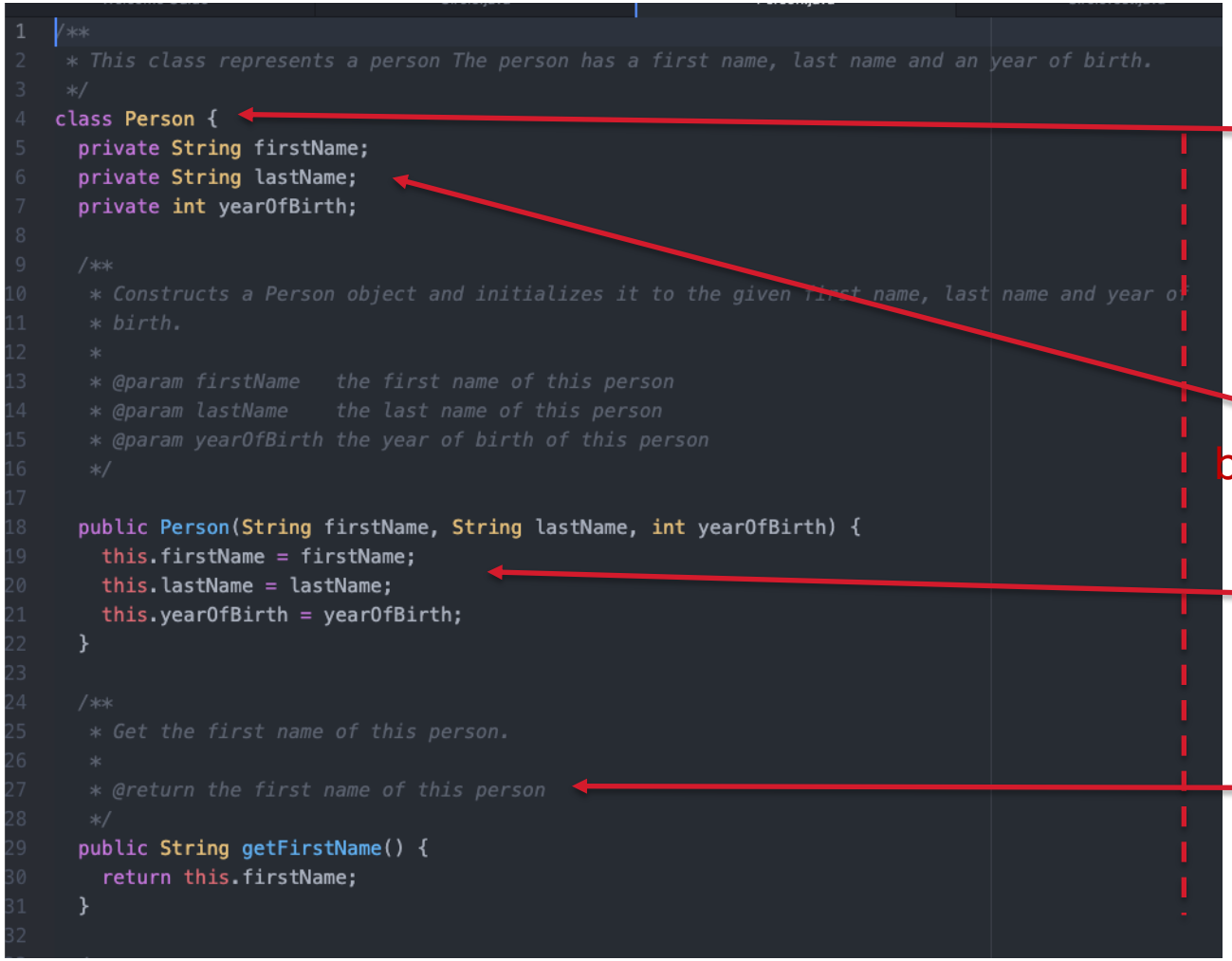
Python uses **self**; Java uses **this**

No conditional code to determine data type

Objects know what they are, and what data they have

OTHER JAVA ELEMENTS

WE'LL COVER SOME OF THIS IN MORE DETAIL LATER



```java
/**
 * This class represents a person The person has a first name, last name and an year of birth.
 */
class Person {
  private String firstName;
  private String lastName;
  private int yearOfBirth;

  /**
   * Constructs a Person object and initializes it to the given first name, last name and year of
   * birth.
   *
   * @param firstName   the first name of this person
   * @param lastName    the last name of this person
   * @param yearOfBirth the year of birth of this person
   */

  public Person(String firstName, String lastName, int yearOfBirth) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.yearOfBirth = yearOfBirth;
  }

  /**
   * Get the first name of this person.
   *
   * @return the first name of this person
   */
  public String getFirstName() {
    return this.firstName;
  }
```

{ Curly Braces } instead of tab formatting to designate "blocks"
Lines end with semi-colon ;

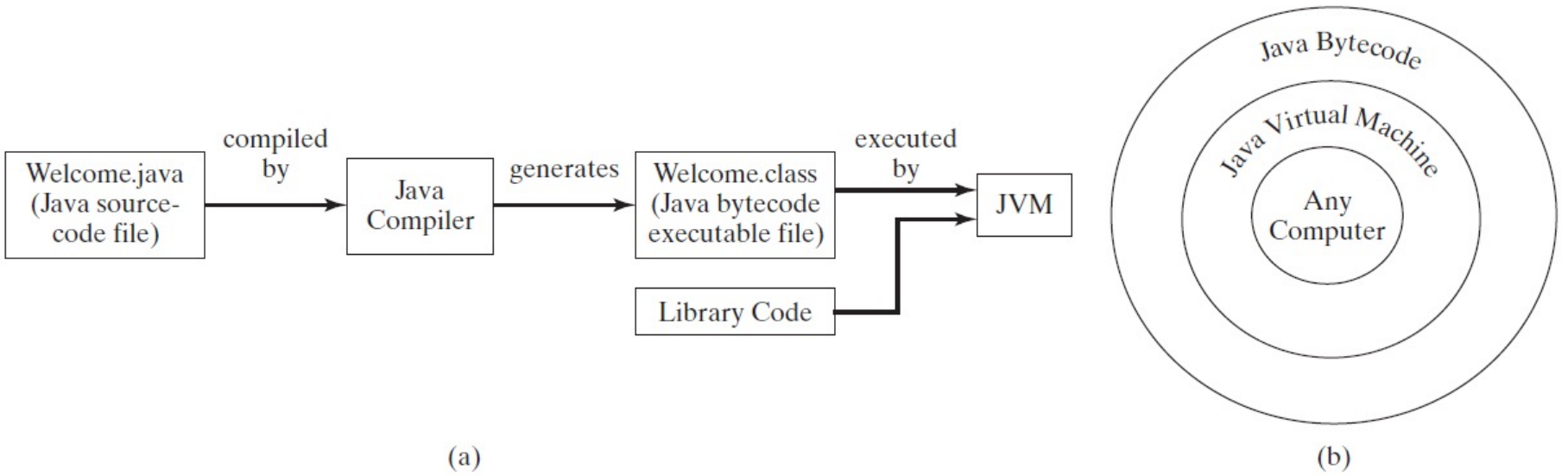
Variables **must** be **defined** with a type before being used. Variable types are unchanging

this is like **self**, but <u>not</u> explicitly passed in to methods

Comments
// is similar to # for single-line comments
/** */ is for Javadoc (sort of like "" "")
/* is for regular multi-line comments

SIMILAR TO PYTHON – VIRTUAL MACHINE

- Similar to the intermediate bytecode you became familiar with last semester (if you took 5001), Java also uses a virtual machine to allow for portability. Your Java compiler compiles to bytecode, not machine code.
  Those of you in 5008 are using C: that language compiles to machine code



JAVA BASIC DATA TYPES

- Basic "Primitive" types Java provides (Numeric)

Name	Range	Storage Size
`byte`	-2^7 to $2^7 - 1$ (-128 to 127)	8-bit signed
`short`	-2^{15} to $2^{15} - 1$ (-32768 to 32767)	16-bit signed
`int`	-2^{31} to $2^{31} - 1$ (-2147483648 to 2147483647)	32-bit signed
`long`	-2^{63} to $2^{63} - 1$ (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed
`float`	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
`double`	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754

JAVA BASIC DATA TYPES

- Java also provides these primitives
  - *char – single Unicode character data*
  - *boolean – Boolean for true and false*

And a very useful Object type for handling strings: the *String* class.

THINGS TO REMEMBER AS YOU CODE IN JAVA

- Java is statically and strongly typed. Variables need to be declared before used
- Curly braces are mandatory for creating a scope (method, code, loop, etc.)
- Semicolons are required to terminate a line of source code.
- *For floating point numbers, prefer double over float*
- *The keyword final is used to create constants in Java*

LET'S DISCUSS

- Object Orientation starts with understanding the objects in your system and asking
  - What do they know?
  - What do they do?

What does an object know?
(data & relationships)

What does an object do?
(behaviors)

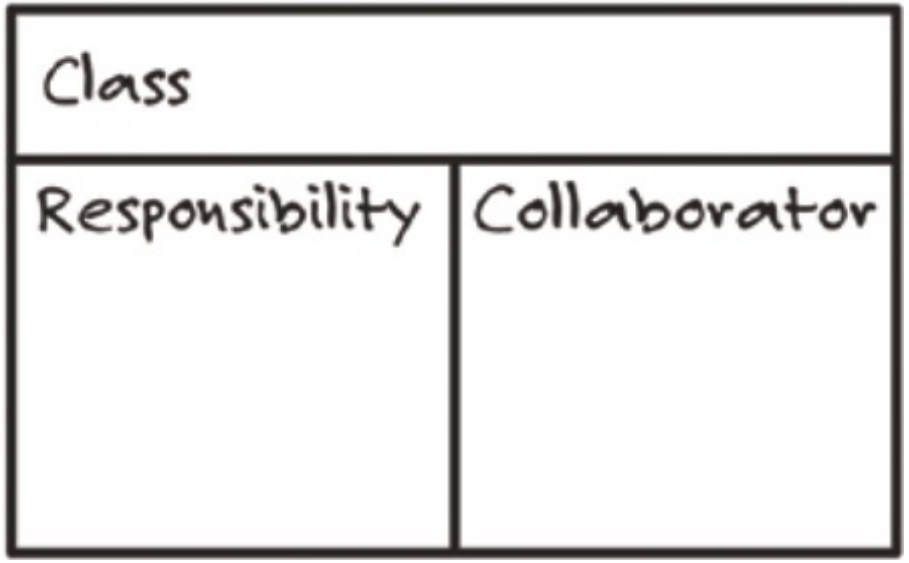
LET'S DISCUSS

- Objects should be cohesive & coherent
  - They should do one theme well
  - They should NOT know everything about your system

- Example Circle
- Before we code anything, how can we capture the essence of "knows/does"?

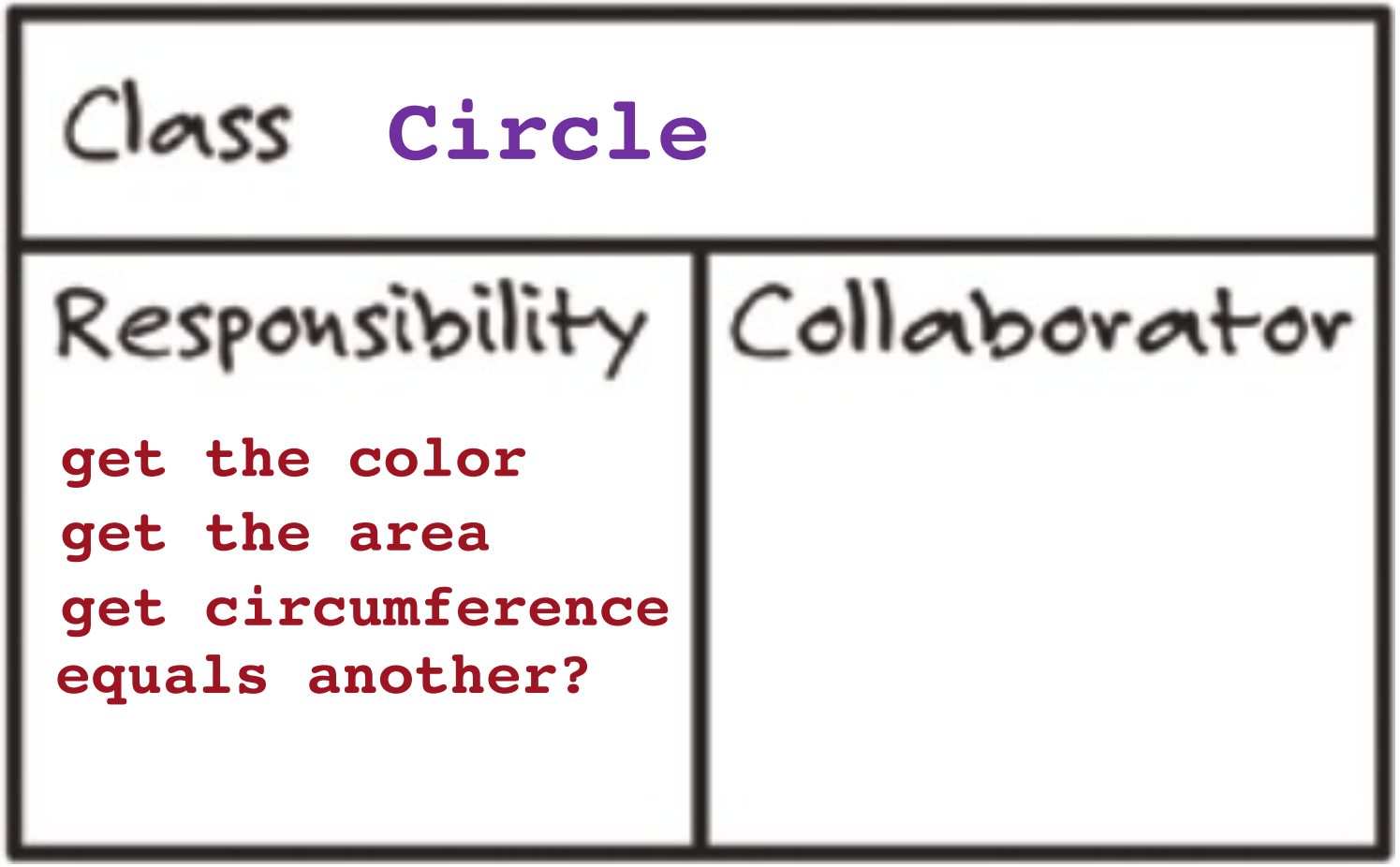
CLASS, RESPONSIBILITIES, COLLABORATORS (CRC)

- Some of you may have jumped directly to UML, but we'll cover CRC cards first! ☺
- Can be analog old-school index cards or electronic version
- Used to document the responsibilities and collaborations of a class
  - Responsibilities: What does it do, and the public-facing view of what it knows
  - Collaborators: Objects it works with to get its job done



CRC CARDS

- What about a Circle?



CRC CARDS

- Conceptual Modeling, so we're using more "natural language" than programming language
- We want to (eventually) create the code to manage Circle objects for a drawing program. We'd like to be able to talk about the color, perimeter and area of circles. And maybe even ask if the 2 or more circles are the "same"
- Need to figure out how to translate the above to OO language (like Java)
  - Transformation from conceptual to logical design
    - How do we represent what circles "know"? What data types?
    - What does "same" mean in this context?

CRC FOOTNOTE – ROLE PLAYING

- Not LARP or Cosplay!
- CRC cards can be “low barrier” roleplay tool to discover additional objects, relationsips & operations. Can also be used to explore testing scenarios
- Team members perform roles associated with the objects previously identified. Exploratory activity:
  - Can freely make new cards on-demand if needed
  - Tear up and dispose of existing cards if Team finds some object/classes need to be refactored or combined

LET'S PUT IT TOGETHER IN JAVA, TOGETHER

- Help me code the Circle in Java

LET'S TEST IT TOGETHER IN JAVA, TOGETHER

- Help me code the Tests for Circle in JUnit

STRETCH BREAK

YOUR TURN!

- Follow a similar recipe for a Square
- Create a CRC card (on paper if you want) to represent the Square concept
  - For now, it's redundant, but squares know their color, and width. They can answer their perimeter, area, and is-same-as for other squares. We'll get a better design of this in future weeks; work with these basics for now.
- Write the Java code & Junit4 tests
- Once you're done, we'll do it together, taking a slightly different approach

TEST-DRIVEN APPROACH

- Slightly different from what we did with our Circle
- Write code to pass tests
- Code is complete when all tests pass
- Think about tests first

TEST-DRIVEN APPROACH

- Think about the tests
- For each unit under test*
  1) Write an empty implementation (or almost empty, if you need to return something)
  2) Write the unit tests
  3) Implement the methods required to pass the tests

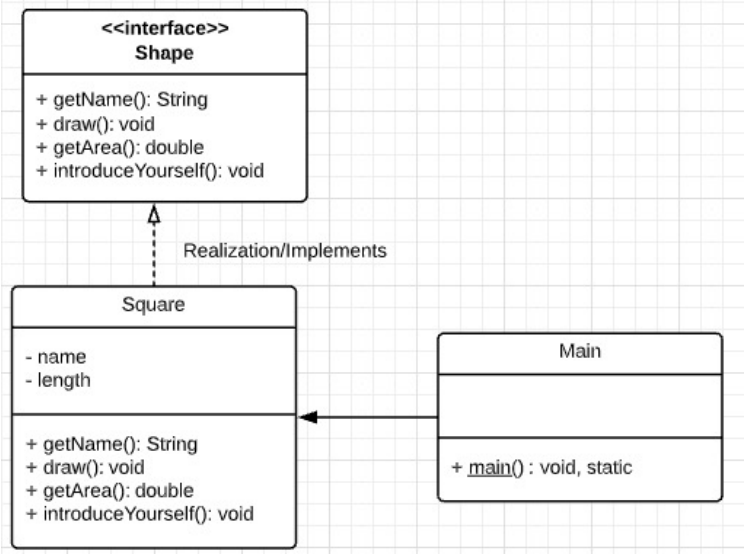
* As a variation, some folks do this on a per-method basis – e.g.: (1) write empty method, (2) write test method, (3) write method code to pass the test. Same flow, but in smaller chunks.

LET'S CODE AND TEST THE SQUARE TDD STYLE

- Remember, we're using JUnit4

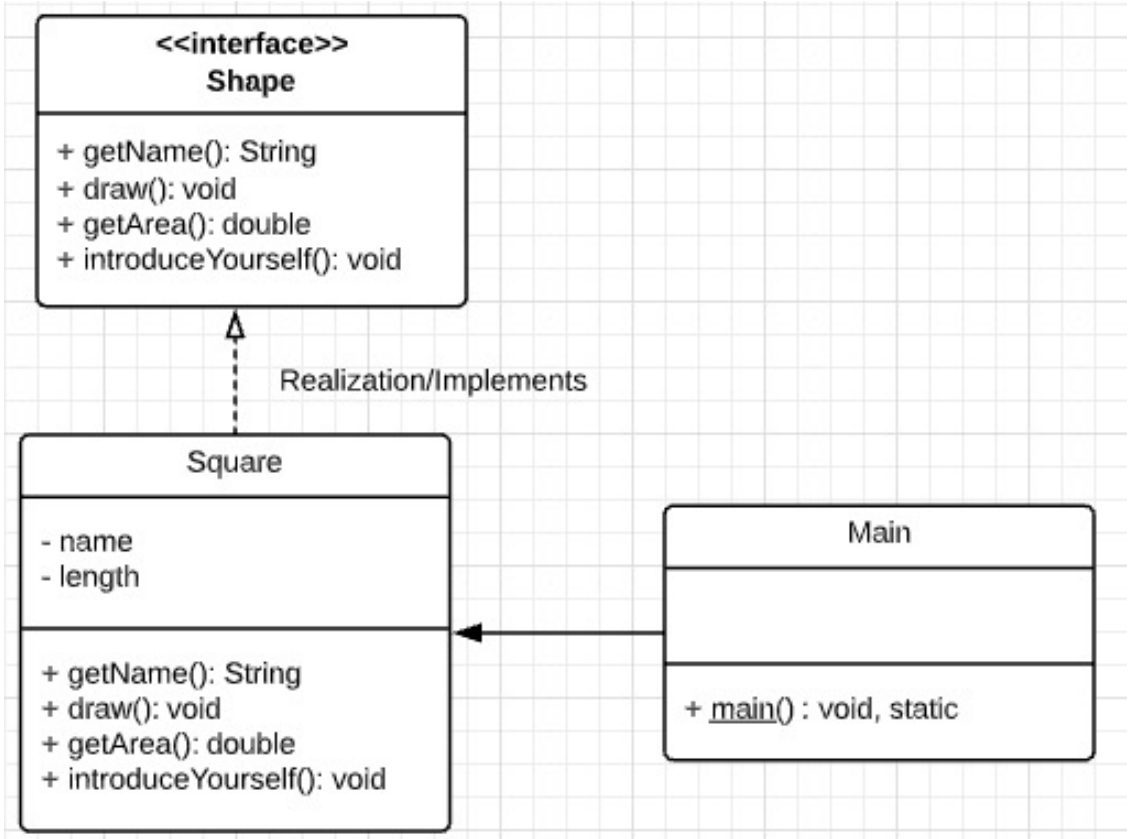
A TOUCH OF UML

- The Unified Modeling Language is a graphical notation with syntax and semantics that raises the abstraction level for specifying and designing systems
  - It can also be used for Model-Driven Development to produce applications via model compilers – a bit on that near the end of this class
  - Acts as a "lingua franca" or common language that cuts across implementations
    - Doesn't matter if you use Java, Python, C#, etc.
  - Captures both structural and behavioral elements
    - Class diagrams, Interaction Diagrams, State Diagrams, etc



A TOUCH OF UML

- What we have represented is simply **one** view. It's a structural view that shows the "form" of part of our system, and how elements are related
- Other views (behavioral) show the sequence of interactions and message sends between objects, or the state transitions that a single object goes through as events trigger those changes
- Can you see how the CRC cards map to this?
- We'll cover more UML in future lectures



SUMMARY

- Both Java & Python are OO languages
  - OO concepts are higher-level and transferable across implementation systems. Some languages may not support all features discussed
  - Java is a pure OO language; Python is hybrid/multi-paradigm
  - “better” is in the eye of the beholder
    - Computer scientists & software engineers should pick the best tool for the job

      Or, they’re told what to use by their employer! ☺

COURSE LOGISTICS – CANVAS WALKTHROUGH

- We'll walk through the Canvas syllabus
- Remember to do Module 0 and Module 1 this week. Do Lab 0 on your own to be ready for Lab 1 next week
- Quiz 1 is due on Saturday at 11:59pm. 5 questions.

Q & A

THANKS!

- Stay safe, be encouraged, & see you next week!