What is Programming?

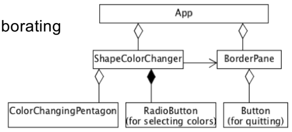
Aspects of Programming, Computer Languages, Objects and Object-Oriented Programming

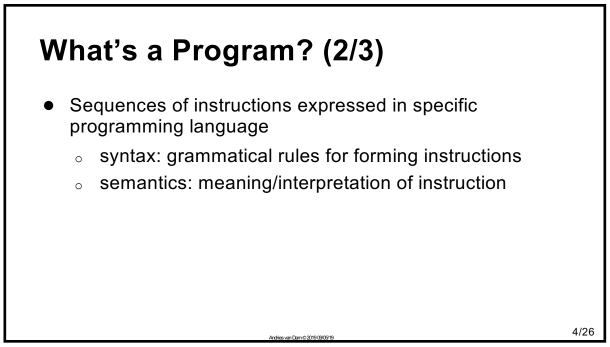
Many Aspects of Programming

- Programming is **controlling**
  - computer does exactly what you tell it to do – literal minded idiot savant
- Programming is **problem solving**
  - always trying to make the computer do something useful
  - e.g., finding an optimal travel route
  - methodology is applicable to other fields
- Programming is **creative**
  - must find the best solution out of many possibilities
- Programming is **modeling**
  - describe salient (relevant) properties and behaviors of a system of components (objects)
- Programming is **abstraction**
  - identify important features without getting lost in detail
- Programming is **concrete**
  - must provide detailed instructions to complete task
- Programming is a **craft**
  - a bit like architecture, engineering - disciplined and creative craft for building artifacts

What's a Program? (1/3)

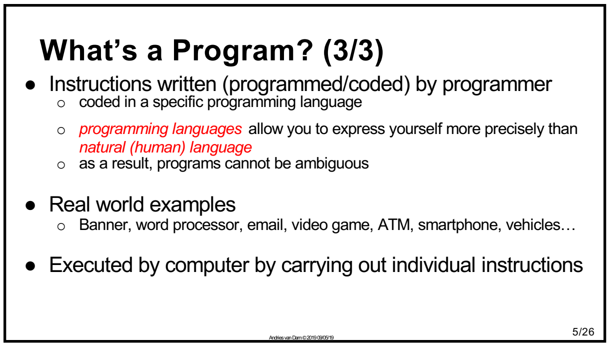
- Model of complex system
  - model**:** simplified representation of salient features of something, either tangible or abstract
  - system**:** collection of collaborating components





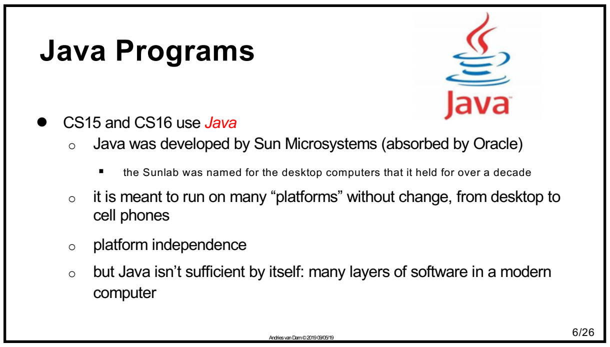
What's a Program? (2/3)

- Sequences of instructions expressed in specific programming language
  - syntax: grammatical rules for forming instructions
  - semantics: meaning/interpretation of instruction



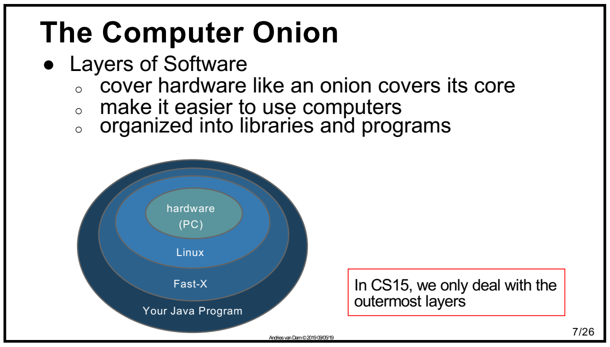
What's a Program? (3/3)

- Instructions written (programmed/coded) by programmer
  - coded in a specific programming language
  - *programming languages* allow you to express yourself more precisely than *natural (human) language*
  - as a result, programs cannot be ambiguous
- Real world examples
  - Banner, word processor, email, video game, ATM, smartphone, vehicles…
- Executed by computer by carrying out individual instructions



Java Programs

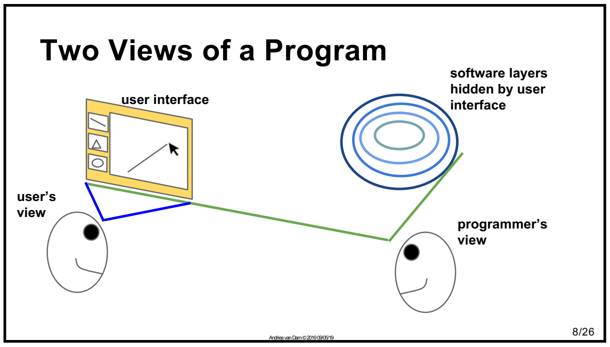
- CS15 and CS16 use *Java*
  - Java was developed by Sun Microsystems (absorbed by Oracle)
    - the Sunlab was named for the desktop computers that it held for over a decade
  - it is meant to run on many "platforms" without change, from desktop to cell phones
  - platform independence
  - but Java isn't sufficient by itself: many layers of software in a modern computer



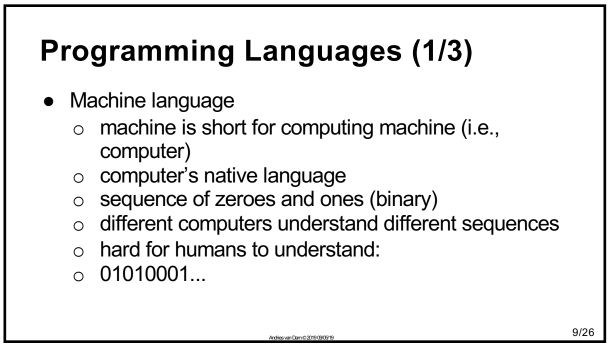
The Computer Onion

- Layers of Software
  - cover hardware like an onion covers its core
  - make it easier to use computers
  - organized into libraries and programs

In CS15, we only deal with the outermost layers

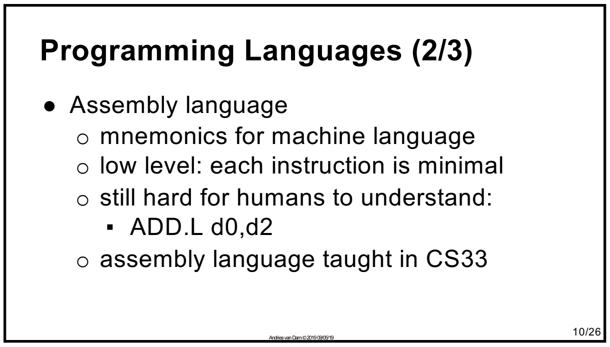


Two Views of a Program



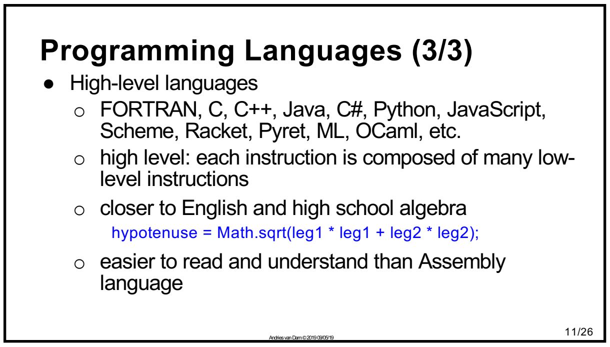
Programming Languages (1/3)

- Machine language
  - machine is short for computing machine (i.e., computer)
  - computer's native language
  - sequence of zeroes and ones (binary)
  - different computers understand different sequences
  - hard for humans to understand:
  - 01010001...



Programming Languages (2/3)

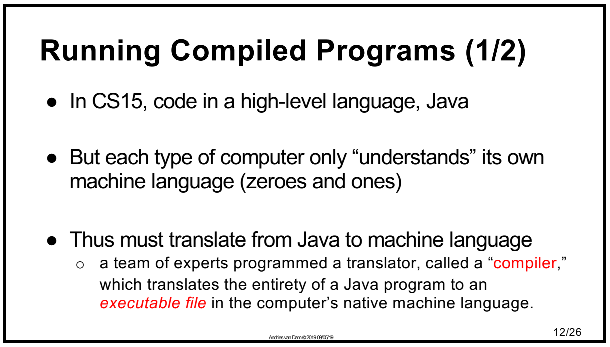
- Assembly language
  - mnemonics for machine language
  - low level: each instruction is minimal
  - still hard for humans to understand:
    - ADD.L d0,d2
  - assembly language taught in CS33



Programming Languages (3/3)

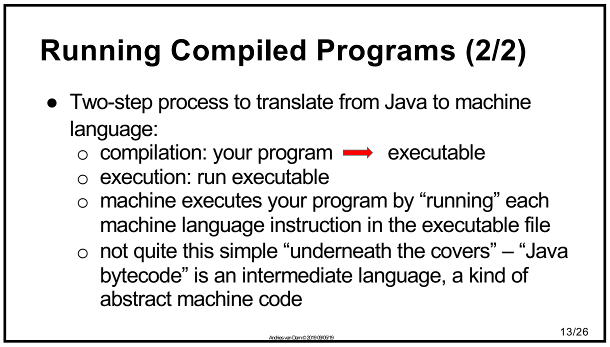
- High-level languages
  - FORTRAN, C, C++, Java, C#, Python, JavaScript, Scheme, Racket, Pyret, ML, OCaml, etc.
  - high level: each instruction is composed of many low-level instructions
  - closer to English and high school algebra

    ```
    hypotenuse = Math.sqrt(leg1 * leg1 + leg2 * leg2);
    ```
  - easier to read and understand than Assembly language



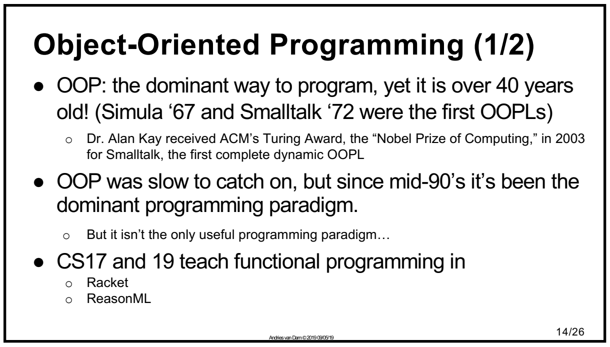
Running Compiled Programs (1/2)

- In CS15, code in a high-level language, Java
- But each type of computer only “understands” its own machine language (zeroes and ones)
- Thus must translate from Java to machine language
  - a team of experts programmed a translator, called a “compiler,” which translates the entirety of a Java program to an *executable file* in the computer’s native machine language.



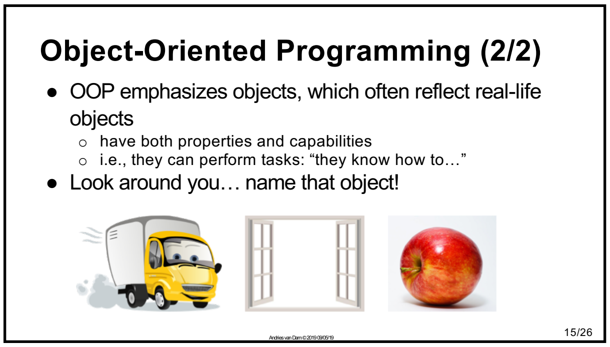
Running Compiled Programs (2/2)

- Two-step process to translate from Java to machine language:
  - compilation: your program ➡ executable
  - execution: run executable
  - machine executes your program by “running” each machine language instruction in the executable file
  - not quite this simple “underneath the covers” – “Java bytecode” is an intermediate language, a kind of abstract machine code



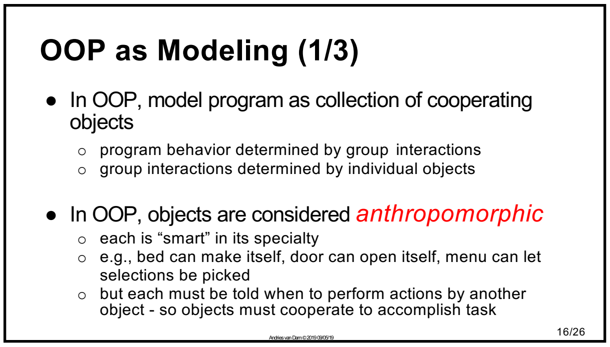
Object-Oriented Programming (1/2)

- OOP: the dominant way to program, yet it is over 40 years old! (Simula ‘67 and Smalltalk ‘72 were the first OOPLs)
  - Dr. Alan Kay received ACM’s Turing Award, the “Nobel Prize of Computing,” in 2003 for Smalltalk, the first complete dynamic OOPL
- OOP was slow to catch on, but since mid-90’s it’s been the dominant programming paradigm.
  - But it isn’t the only useful programming paradigm…
- CS17 and 19 teach functional programming in
  - Racket
  - ReasonML



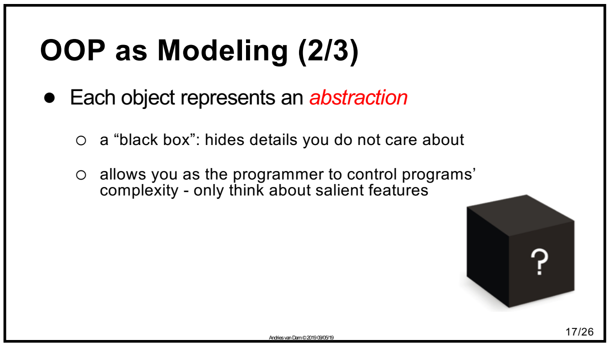
Object-Oriented Programming (2/2)

- OOP emphasizes objects, which often reflect real-life objects
  - have both properties and capabilities
  - i.e., they can perform tasks: “they know how to…”
- Look around you… name that object!



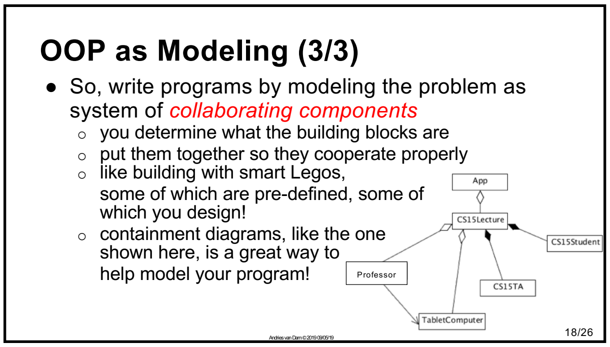
OOP as Modeling (1/3)

- In OOP, model program as collection of cooperating objects
  - program behavior determined by group interactions
  - group interactions determined by individual objects
- In OOP, objects are considered *anthropomorphic*
  - each is "smart" in its specialty
  - e.g., bed can make itself, door can open itself, menu can let selections be picked
  - but each must be told when to perform actions by another object - so objects must cooperate to accomplish task



OOP as Modeling (2/3)

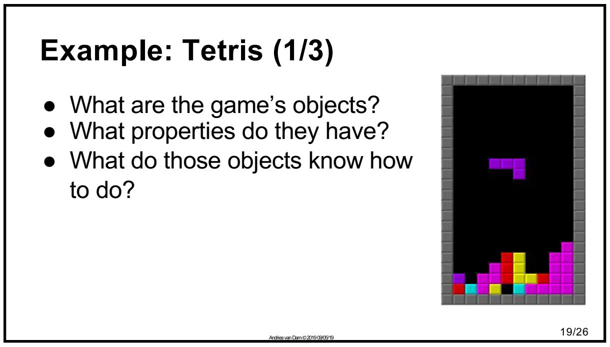
- Each object represents an *abstraction*
  - a "black box": hides details you do not care about
  - allows you as the programmer to control programs' complexity - only think about salient features



OOP as Modeling (3/3)

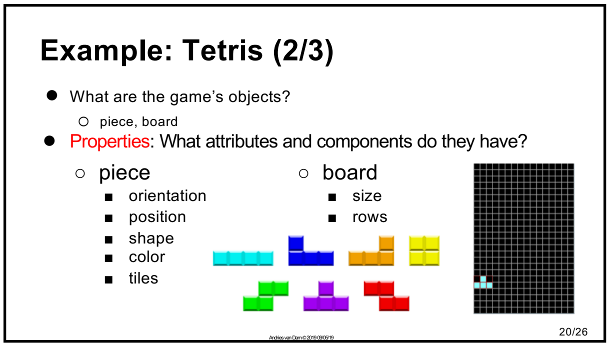
- So, write programs by modeling the problem as system of *collaborating components*
  - you determine what the building blocks are
  - put them together so they cooperate properly
  - like building with smart Legos, some of which are pre-defined, some of which you design!
  - containment diagrams, like the one shown here, is a great way to help model your program!

App
CS15Lecture
CS15Student
Professor
CS15TA
TabletComputer



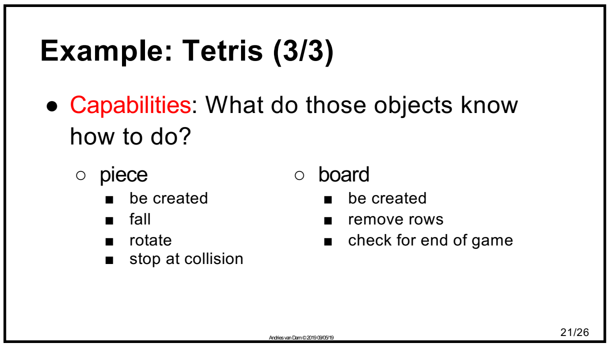
Example: Tetris (1/3)

- What are the game's objects?
- What properties do they have?
- What do those objects know how to do?



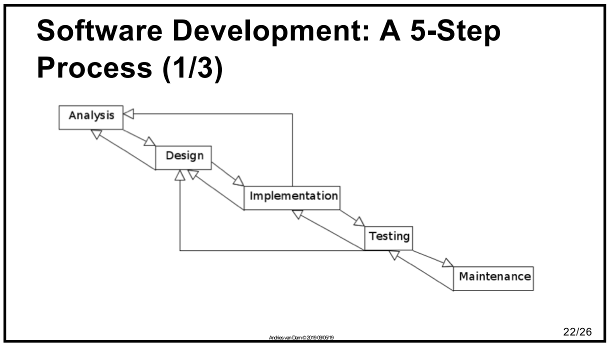
Example: Tetris (2/3)

- What are the game's objects?
  - piece, board
- Properties: What attributes and components do they have?
  - piece
    - orientation
    - position
    - shape
    - color
    - tiles
  - board
    - size
    - rows



Example: Tetris (3/3)

- Capabilities: What do those objects know how to do?
  - piece
    - be created
    - fall
    - rotate
    - stop at collision
  - board
    - be created
    - remove rows
    - check for end of game



Software Development: A 5-Step Process (1/3)

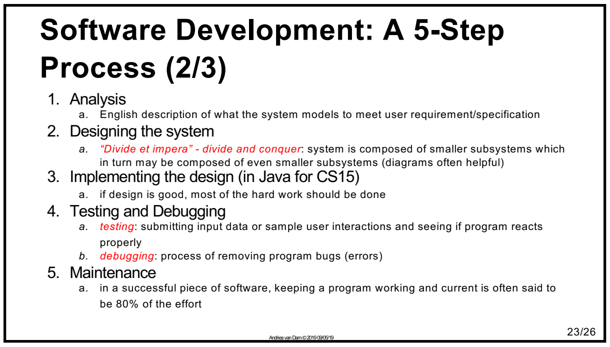
Analysis

Design

Implementation

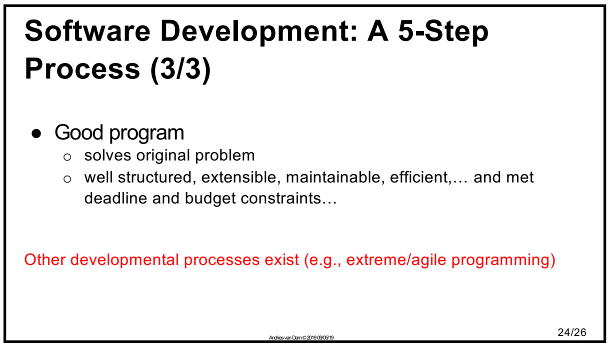
Testing

Maintenance



Software Development: A 5-Step Process (2/3)

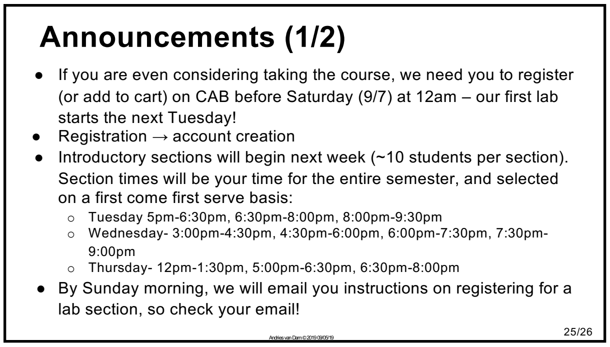
1. Analysis
   a. English description of what the system models to meet user requirement/specification
2. Designing the system
   a. *"Divide et impera" - divide and conquer*: system is composed of smaller subsystems which in turn may be composed of even smaller subsystems (diagrams often helpful)
3. Implementing the design (in Java for CS15)
   a. if design is good, most of the hard work should be done
4. Testing and Debugging
   a. *testing*: submitting input data or sample user interactions and seeing if program reacts properly
   b. *debugging*: process of removing program bugs (errors)
5. Maintenance
   a. in a successful piece of software, keeping a program working and current is often said to be 80% of the effort



Software Development: A 5-Step Process (3/3)

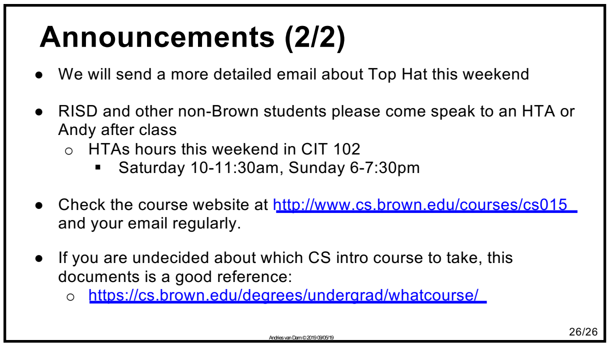
- Good program
  - solves original problem
  - well structured, extensible, maintainable, efficient,… and met deadline and budget constraints…

Other developmental processes exist (e.g., extreme/agile programming)



Announcements (1/2)

- If you are even considering taking the course, we need you to register (or add to cart) on CAB before Saturday (9/7) at 12am – our first lab starts the next Tuesday!
- Registration → account creation
- Introductory sections will begin next week (~10 students per section). Section times will be your time for the entire semester, and selected on a first come first serve basis:
  - Tuesday 5pm-6:30pm, 6:30pm-8:00pm, 8:00pm-9:30pm
  - Wednesday- 3:00pm-4:30pm, 4:30pm-6:00pm, 6:00pm-7:30pm, 7:30pm-9:00pm
  - Thursday- 12pm-1:30pm, 5:00pm-6:30pm, 6:30pm-8:00pm
- By Sunday morning, we will email you instructions on registering for a lab section, so check your email!



Announcements (2/2)

- We will send a more detailed email about Top Hat this weekend
- RISD and other non-Brown students please come speak to an HTA or Andy after class
  - HTAs hours this weekend in CIT 102
    - Saturday 10-11:30am, Sunday 6-7:30pm
- Check the course website at http://www.cs.brown.edu/courses/cs015 and your email regularly.
- If you are undecided about which CS intro course to take, this documents is a good reference:
  - https://cs.brown.edu/degrees/undergrad/whatcourse/