Welcome to CS / ENGRD 2110!

While we're waiting to begin, get a head start by signing into PollEv!

Use your Cornell email address
(but don't "Sign in with Google")

Lecture 1: Introduction

CS 2110

January 20, 2026

Matt Eichhorn

- PhD (Applied Math) at Cornell
- BS (Computer Science, Math)

Teaching:
- CS 2110, CS 2800, ENGRI 1101

Research:
- Algorithm Design: optimization with unknown / random inputs
- Statistics on networks: how do interventions cascade through populations

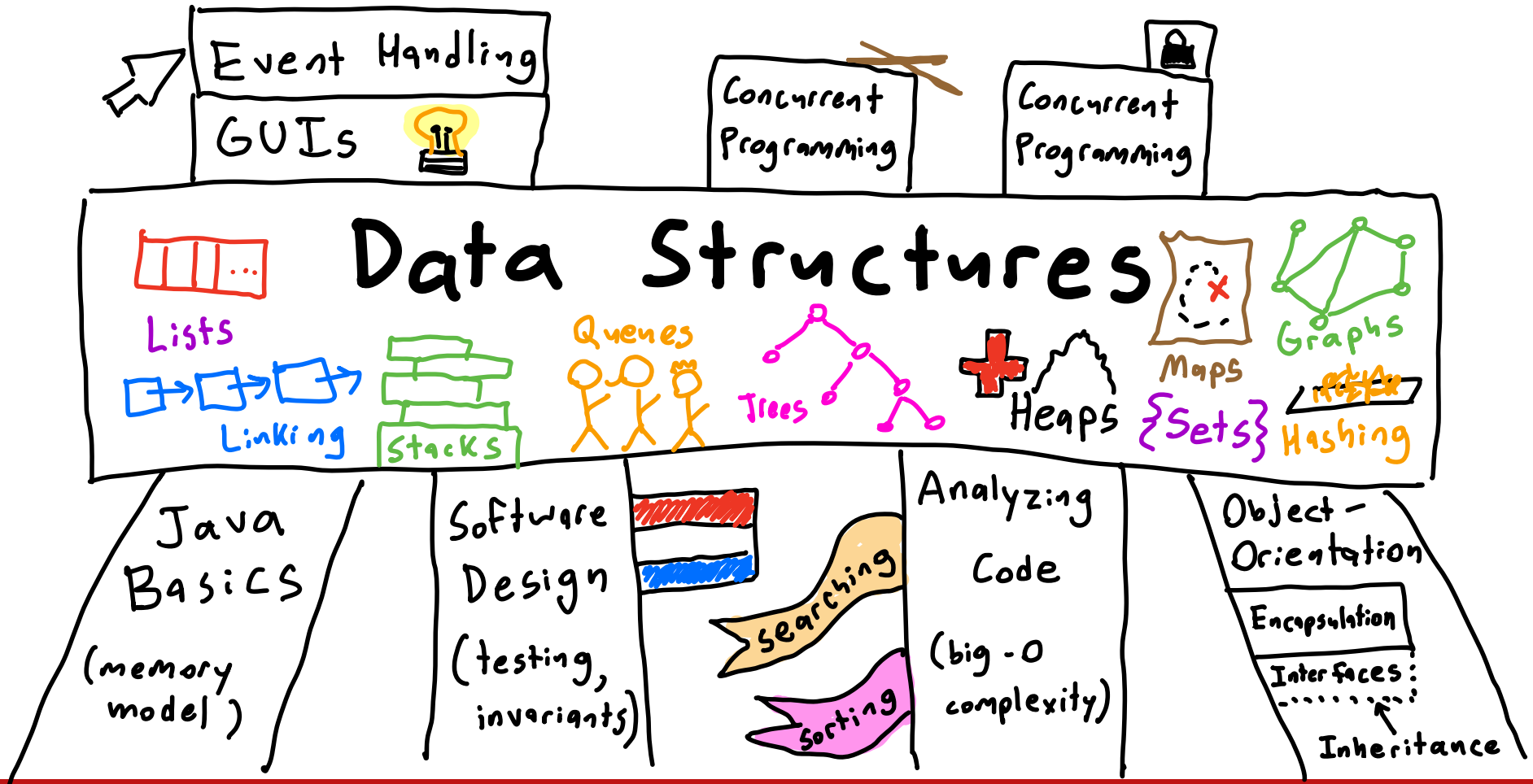
Interests:
- Origami, crosswords, board games, Buffalo Bills

Goal of CS 2110: Write *Better* Code

What makes code *better*?

- more efficient (speed, energy use, memory)
- easier to read, adapt, maintain, extend
- user-friendly with good features
- more reliable, fewer bugs
- robust to security vulnerabilities
- ⋮

Course Map


Event Handling
GUIs
Concurrent Programming
Concurrent Programming
Data Structures
Lists
Linking
Stacks
Queues
Trees
Heaps
Maps
{Sets}
Graphs
Hashing
Java Basics
(memory model)
Software Design
(testing, invariants)
searching
sorting
Analyzing Code
(big-O complexity)
Object-Orientation
Encapsulation
Interfaces:
Inheritance

Course Logistics

Everything you need is linked from our course website

courses.cis.cornell.edu/courses/cs2110/2026sp/

(find a link on our Canvas page)

- Syllabus
- Lecture notes
- Slides and demo code (after lecture)
- Discussion materials
- Assignments
- Grades

Class Norms

Prioritizing Effort and Integrity:
Assignments are designed for their learning benefit, not their end product
Your submissions must reflect your own efforts/abilities (not generative AI)
Adhere to the course collaboration policy (syllabus)

Mutual Respect:
We all are here with the same goals
Communicate kindly/supportively to peers/course staff
Let us know if anyone behaves disrespectfully

Inclusivity:
We come with different backgrounds and needs
Everyone in the course is meant to be here
Let the course staff know how we can best support you

Let the learning begin!

Today's Learning Outcomes

1. Explain the difference between *statically* and *dynamically* typed programming languages.

2. Determine the static type of an expression involving one of more operators, method calls, and/or implicit/explicit type coercions.

3. Draw a memory diagram that visualize the state of execution of a program involving multiple method calls.

Poll Everywhere

PollEv.com/javabear text **javabear** to 22333

What is your "go to" programming language?

Java	17% **(A)**	Python	74% **(B)**
C / C++	3% **(C)**	Matlab	1% **(D)**
Javascript / Typescript	1% **(E)**	R	2% **(F)**
Something Else	2% **(G)**		

A Basic Python Program

```
# Returns the number of minutes in `days` days
def days_to_minutes(days):
  hours = 24 * days
  minutes = 60 * hours
  return minutes

if __name__ == "__main__":
    print("Enter a number of days: ", end="")
    days = input() # get user input from the console
    mins = days_to_minutes(days)
    print("There are " + mins + " minutes in " + days + " days.")
```

start here

Poll Everywhere

PollEv.com/javabear text **javabear** to **22333**

What will be the output of this program?

- There are 1440 minutes in 3 days. **(A)**
- There are 4320 minutes in 3 days. **(B)**
- (The program will crash because of an error) **(C)**
- Something else... **(D)**

A Basic Python Program

```
# Returns the number of minutes in `days` days
def days_to_minutes(days):
  hours = 24 * days
  minutes = 60 * hours
  return minutes

if __name__ == "__main__":
    print("Enter a number of days: ", end="")
    days = input() # get user input from the console
    mins = days_to_minutes(days)
    print("There are " + mins + " minutes in " + days + " days.")
```

String concatenation

days is a String, not a number

Dynamic vs. Static Typing

The type of a variable (or expression) determines its possible values and how it can be used.

Dynamically Typed languages infer types at runtime (Python)
- shorter code, but less "safe"

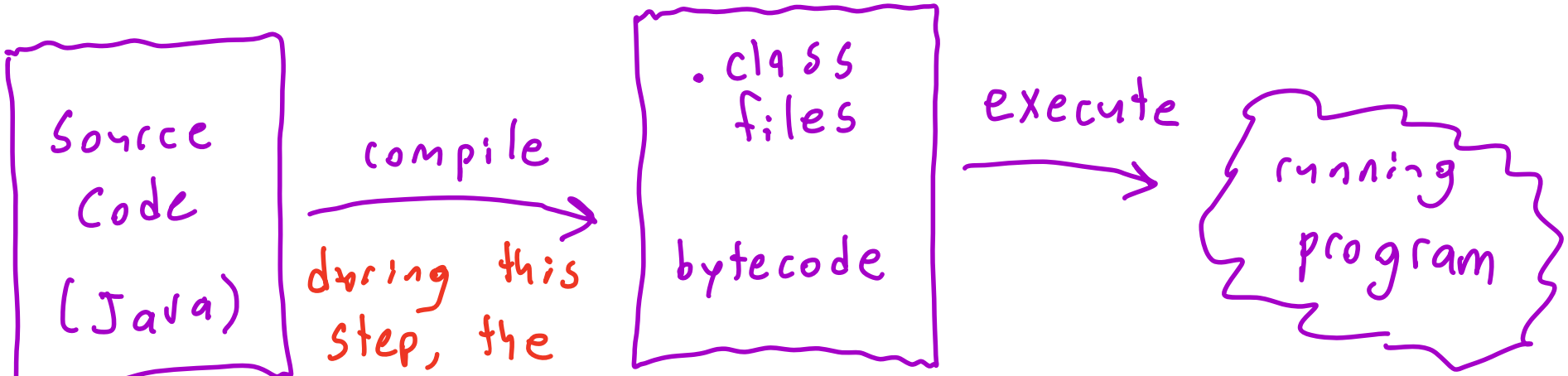
Statically Typed languages check types before code is run (Java)
- often require explicit type declarations by programmer

Coding Demo: Translation to Java

(We include slides like this one in case you want a place to take notes during/about the demo.)

Compilation

↳ intermediary "code translation" step that happens before code runs



Primitive Types

8 in Java, most important are:

- int (4 byte) (whole numbers)
- double (8 byte) (decimal numbers)
- char (2 byte) (characters)
- boolean (1 bit) (true/false)

- Fixed size, values stored directly in variables

```
int age = 27;
```

(27: int literal)

age: int 27

Expressions

An expression is a unit of code with type/value.

- literals: 27 (int) 30.5 (double) true (boolean) 'A' (char)
- variables: int age=27; age (int, value=27)
- using operators: 27*12 (int, value=324)
- method call:
  static int daysToMins(int days)
  daysToMins(1) (int, value=1440)

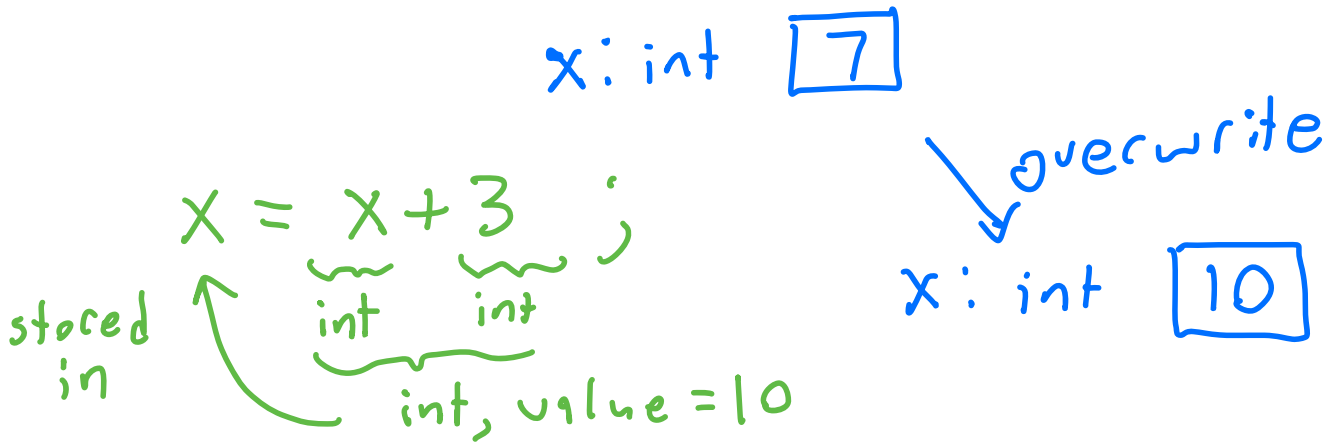
Compiler determines types of all expressions and checks for agreement.

Statements and Assignment

A statement is a unit of code that describes an action (i.e., it produces a side effect)

Ex. Assignment statements

```
int x = 7; // store RHS value in LHS container
```



Primitive Type Coercion

Most operations preserve primitive types:

3 + 4 → 7
int int int

3.0 + 4.0 → 7.0
double double double

Coercion lets us alter expression types

Implicit (widening): 3 + 4.0 → 3.0 + 4.0 → 7.0
int double double double

Explicit (casting): 7 / 2 → 3 (truncated division)
int int int

(double) 7 / 2.0 → 3.5
7.0

* coercion affects expression evaluation but doesn't change underlying variables.

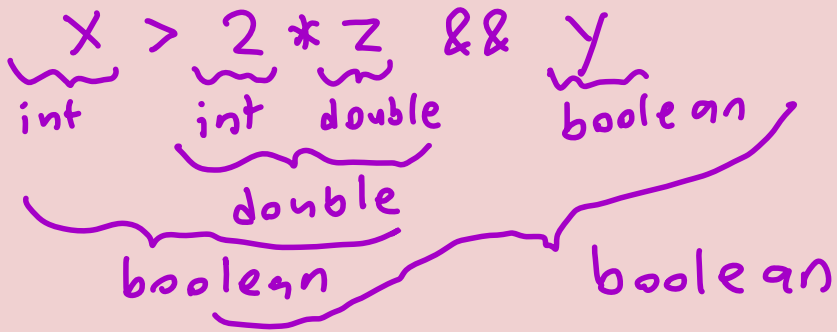
Poll Everywhere

PollEv.com/javabear text **javabear** to **22333**

What is the static type of the highlighted expression?

```
int x = 6;
boolean y = true;
double z = 4.3;

System.out.print(x > 2 * z && y)
```

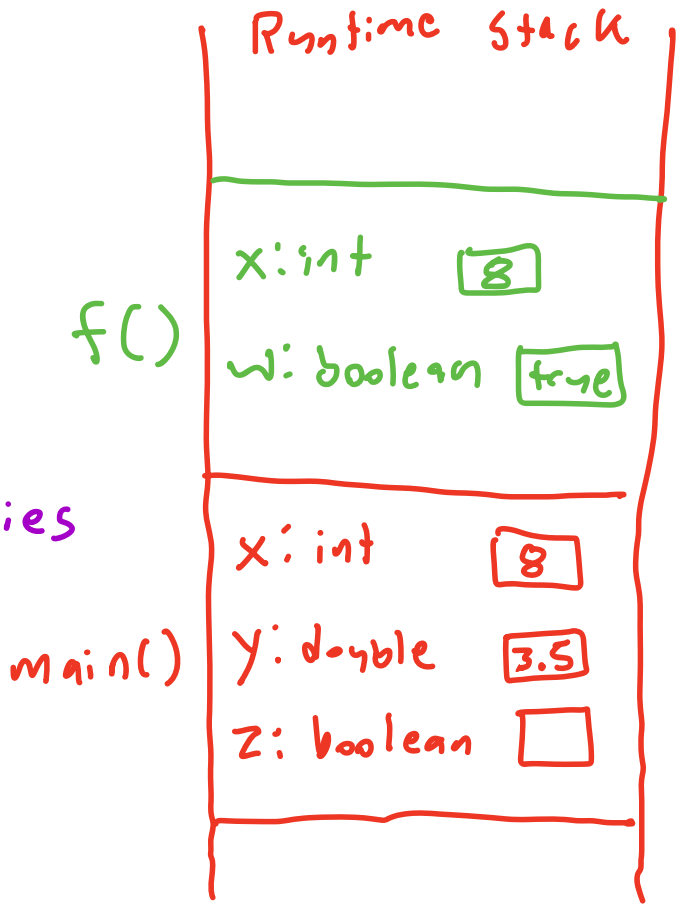


int	**(A)**
boolean	**(B)**
double	**(C)**
String	**(D)**

Program Execution

Starts by invoking the main() method

- whenever we invoke a method, it creates a call frame on the runtime stack
  - call frame has entries (variable boxes) for all parameters and local variables of the method
  - assignment statements update these entries
  - inner method calls "stack up" call frames
  - call frames destroyed when methods return



Diagramming our Code

* We didn't get to this slide in lecture, but you can step through an animated code trace in the Lecture 1 notes on the website

```
public class Convert {
  static int daysToMinutes(int days) {
    int hours = 24 * days;
    int minutes = 60 * hours;
    return minutes;
  }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in));
    int days = sc.nextInt();
    int mins = daysToMinutes(days);
    System.out.print(...);
  }
}
```