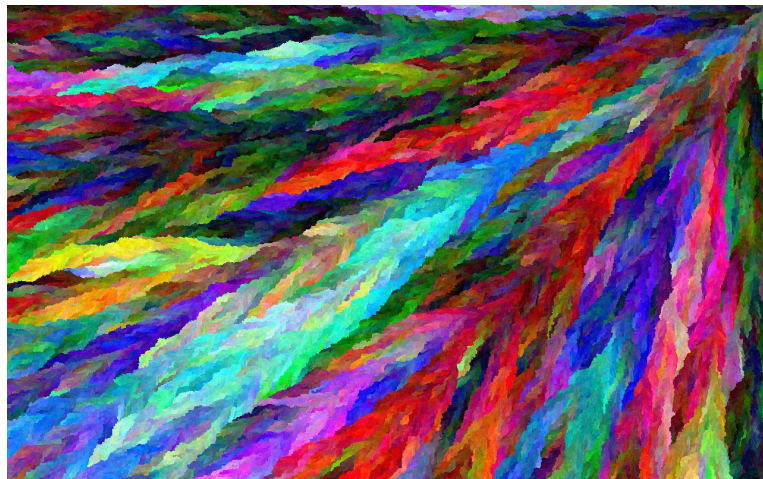


Course website: <https://sp25.datastructur.es>

Ask your questions in the Zoom chat! (link on website)



Lecture 1

Introduction to 61B, Java

CS61B, Spring 2025 @ UC Berkeley

Lecturers: Justin Yokota, Josh Hug

Slides Credit: Josh Hug

Welcome to 61B

Lecture 1, CS61B, Fall 2024

- **Welcome!**
 - **Welcome to 61B**
 - 61B Logistics
- Our First Java Programs
 - Hello World
 - Hello Numbers
 - Larger
 - Reflections on Java
- Workflow
 - Compilation
 - IntelliJ
- HW0: Due Friday!

What is 61B about?

- Java? Not the focus of the course!
- Writing code that runs efficiently.
 - Good algorithms.
 - Good data structures.
- Writing code efficiently.
 - Designing, building, testing, and debugging large programs.
 - Use of programming tools.
 - git, IntelliJ, JUnit, and various command line tools.

Assumes solid foundation in programming fundamentals, including:

- Object oriented programming, recursion, lists, and trees.

Other great features of 61B:

- The most popular topics for job interview questions in software engineering.
 - Examples: Hash tables, binary search trees, quick sort, graphs, Dijkstra's algorithm.
- Some really cool math. Examples:
 - Asymptotic analysis.
 - Resizing arrays.
 - The isometry between self-balancing 2-3 trees and self-balancing red black trees.
 - Graph theory.
 - P vs NP.
- Once you're done: the confident sense that you can build any software.

Josh Hug: Berkeley faculty since 2014.

- EE B.S. from UT Austin in 2003.
- EECS Ph.D. from UC Berkeley 2011.
- Taught at Princeton 2011-2014.
- Spent all of last year in the **Netherlands**.
- **11th time** teaching this class (but it's been 2 years since my last time), designed most of the homeworks and projects.

Plug: **Spring 2025** **DUTCH 1 001 - LEC 001** offered through **German** 


Elementary Dutch

Elementary Dutch

 Esmee van der Hoeven

Jan 21 2025 - May 09 2025

 M, W

 12:00 pm - 1:59 pm

 **Dwinelle B33B** 

Jan 21 2025 - May 09 2025

 F

 1:00 pm - 1:59 pm

 **Dwinelle B33B** 

Justin Yokota: Berkeley Lecturer since 2022.

- BA/BS in Math/CS from UC Berkeley 2021.
- EECS Masters from UC Berkeley 2022.
- Just came back from MIT Mystery Hunt
- 5th time teaching CS 61B
- Office Hours: Tuesday/Thursday 12:00-1:30 in Soda 329

Question for You

What do you hope / expect to learn from this class? Why are you taking it?

- First Java class (excited to learn a new language)
- Learn Computer Science that's more applicable to the real world
-

Who are you?

- Freshman? Sophomore? Junior? Senior? Grad student? None of the above?
- CS Major? DS Major? Intending to be a CS/DS Major? Something else?
- CS 61A? Java experience? Experience with any other programming languages?

61B Logistics

Lecture 1, CS61B, Fall 2024

- **Welcome!**
 - Welcome to 61B
 - **61B Logistics**
- Our First Java Programs
 - Hello World
 - Hello Numbers
 - Larger
 - Reflections on Java
- Workflow
 - Compilation
 - IntelliJ
- HW0: Due Friday!

Joining the Course

- Course staff does not control getting into the course.
 - [Contact a major advisor](#) if you have questions.
- Being added to course platforms (Ed, Gradescope) is **manual**. We do this every day. Please **do not** email us asking to be added to the platforms unless it's been more than 2 days since you joined the course.
- If you are a pending concurrent enrollment student, you should be added to bCourses, Ed, and Gradescope **1–2 days after** you've submitted your application.
 - We cannot process/approve applications until the department tells us to. We will post on Ed if there are updates.
 - **Do not** email course staff about getting added to the course platforms unless it's been **more than 2 days** since you've submitted your application.

Course Components

Lectures provide you with an introduction and a foundation.

You'll learn most of what you learn in the class by:

- Programming (labs, hws, projects, discussion section).
- Solving interesting problems (study guides, HW3, HW4 old exam problems, discussion section).

This class is divided into three phases:

- Phase 1 (weeks 1 - 4): Intro to Java and Data Structures.
 - All coding work is solo.
 - Moves VERY fast.
 - **HW0 (intro to Java) due Friday (in two days!)**
- Phase 2 (weeks 5 - 10): Data Structures:
 - All coding work is solo.
 - Moves moderately fast.
- Phase 3 (weeks 12 - 14): Algorithms and Software Engineering.
 - Coding work is entirely dedicated to final project, done in pairs.
 - Slower pace.

Four types of points in this class:

- Low effort, everyone should get them: **Weekly Surveys, Course Evaluations**
 - Median score is 100%
- High effort, everyone should get them: **HW, Project, Lab**
 - Median score is 100%
- High effort, not everyone gets them: **Exams**
 - Mean score is 65%
 - Final exam score can replace midterms if you have a bad midterm (or two).
- Pacing points: **Attending Discussion, Lab, and keeping up with Lecture**
 - Small amount of extra credit for keeping up with class.
 - Will not increase your score beyond 75% (B).
 - Example: You have 740 points and earn 20 pacing points, you get 750 points.
- B to B+ threshold is 65% on exams, 95% on everything else.

Full details around point distributions, letter grade assignments, grade replacement, etc. are on the website.

Please don't cheat...

- Last year, we had >100 cheating cases, mostly from project copying.

Most cases can be categorized into one of three types:

- Accidental cheating (exceeding our limits of allowed collaboration)
 - Most common case
- Rational cheating (deliberately trying to "game" our system)
 - Thankfully very rare
- Irrational cheating (reaching a point of desperation and making a bad decision)
 - Also fairly common

How to avoid Accidental Cheating

Our goal in the class is to maximize the amount of learning that gets done.

General rule: If something can potentially allow someone to get a nonzero score on an assignment without them reaching the assignment's learning objectives, it likely constitutes cheating.

- For most projects, the learning objective is NOT "can you write code in Java", but rather "can you develop a solution to a problem from scratch".

Examples of things that are not allowed:

- Using a friend's solution as a "reference" when writing your own solution
- Using LLMs like ChatGPT or Github Copilot to come up with an algorithm for you
- Working with a non-project partner in a way that you both end up with the same underlying solution to a project.

Examples of things that are allowed:

- High-level discussion (e.g. breaking the main project problem down to ~2-3 smaller subproblems, then individually solving those subproblems)
- "Rubber duck" programming
- Copying/generating snippets of code *if* all you need is the Java syntax (e.g. computing a^b where a and b are numbers)

Why Cheating is not Rational

Many of our misconduct penalties are designed to deter rational cheating

- If you get caught cheating on an assignment, that's an automatic 0 on that assignment, and a referral to the Center for Student Conduct for university sanctions. Further cheating or a single case of very egregious cheating can lead to harsher penalties.

Our cheat detection methods are very robust

- We have software to detect similarities between all current submissions, submissions from previous semesters, and solutions you can find online
 - LLMs in general have a tendency to respond to project-related questions with code directly from online solutions...
 - Our software checks your entire history, and is not fooled by things like variable renaming and switching around a few lines of code.
- The work required to bypass all our cheat-detection methods is often more than the work to just do the project honestly, so don't cheat.

Even if you bypass all those cheat-detection methods and get points for something that's not your work, you're losing out on the practice we want you to get from the assignment, and will likely lose any points you gained when you do worse on the next exam as a result.

How to avoid Irrational Cheating

Desperation can cause you to do things you would not normally do, even if you know rationally that it'll likely put you in a worse position later.

The best way to avoid irrational cheating is to make sure you never reach that point of desperation in the first place:

- Start projects early! Part of the project is getting stuck and finding a way to get unstuck. Plan enough buffer time in your schedule to make sure you can still finish even if you're stuck for longer than expected.

If you're falling behind in the class, let us know! We'll work with you to maximize the amount of learning you can get from the class.

Lateness Policies

The deadlines in this class are the day by which assignments should be completed.

- They've been calibrated carefully against lecture, labs, discussions, and exams.
- In weeks 1 - 5, the timing is especially important!

There is no partial credit for work submitted late. Gradescope gives zero points by default to late work.

To provide some flexibility, <https://beacon.datastructur.es/> will allow you to request extensions. These can be retroactive, but we recommend requesting in advance.

If you have extenuating circumstances, see syllabus.

Hello World

Lecture 1, CS61B, Fall 2024

- Welcome!
 - Welcome to 61B
 - 61B Logistics
- **Our First Java Programs**
 - **Hello World**
 - Hello Numbers
 - Larger
 - Reflections on Java
- Workflow
 - Compilation
 - IntelliJ
- HW0: Due Friday!

Let's try writing some simple Java programs.

- First I'll write them in Python (~99% of you have seen Python).
- Then I'll write the equivalent Java program.

If you've never written in code in Python or Java, this will be a little harder for you, but still comprehensible.

This section might be a bit boring if you have Java experience.

(See video or code linked on course website)

Lecture code repository: <https://github.com/Berkeley-CS61B/lectureCode-sp25>

Coding Demo: Hello World

hello.py

HelloWorld.java

```
public class HelloWorld {  
    public static void main(String[] args) {  
  
    }  
}
```

Coding Demo: Hello World

hello.py

```
print("hello world")
```

HelloWorld.java

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("hello world");  
    }  
}
```

Reflections on Hello World:

- In Java, all code must be part of a class.
- Classes are defined with `public class CLASSNAME`
- We use `{ }` to delineate the beginning and ending of things.
- We must end lines with a semicolon.
- The code we want to run must be inside `public static void main(String[] args)`
 - We'll learn what this means later.

Java is an object oriented language with strict requirements:

- Every Java file must contain a class declaration*.
- **All code** lives inside a class*, even helper functions, global constants, etc.
- To run a Java program, you typically define a main method using `public static void main(String[] args)`

*: This is not completely true, e.g. we can also declare “interfaces” in .java files that may contain code. We'll cover these soon.

Hello Numbers

Lecture 1, CS61B, Fall 2024

- Welcome!
 - Welcome to 61B
 - 61B Logistics
- **Our First Java Programs**
 - Hello World
 - **Hello Numbers**
 - Larger
 - Reflections on Java
- Workflow
 - Compilation
 - IntelliJ
- HW0: Due Friday!

Coding Demo: Hello Numbers

hellonumbers.py

```
x = 0;
while x < 10:
    print(x)
    x = x + 1
```

HelloNumbers.java

```
public class HelloNumbers {
    public static void main(String[] args) {

        x = 0;
        while (x < 10) {
            System.out.println(x);
            x = x + 1;
        }

    }
}
```


Coding Demo: Hello Numbers

hellonumbers.py

```
x = 0;
while x < 10:
    print(x)
    x = x + 1
```

HelloNumbers.java

```
public class HelloNumbers {
    public static void main(String[] args) {
        int x;
        x = 0;
        while (x < 10) {
            System.out.println(x);
            x = x + 1;
        }
    }
}
```

Coding Demo: Hello Numbers

hellonumbers.py

```
x = 0;
while x < 10:
    print(x)
    x = x + 1
```

HelloNumbers.java

```
public class HelloNumbers {
    public static void main(String[] args) {

        int x = 0;
        while (x < 10) {
            System.out.println(x);
            x = x + 1;
        }

    }
}
```

Coding Demo: Hello Numbers

hellonumbers.py

```
x = 0;
while x < 10:
    print(x)
    x = x + 1

x = "horse" # works
print(x)
```

HelloNumbers.java

```
public class HelloNumbers {
    public static void main(String[] args) {

        int x = 0;
        while (x < 10) {
            System.out.println(x);
            x = x + 1;
        }
        x = "horse";           // doesn't work
        String x = "horse";    // doesn't work
    }
}
```

Coding Demo: Hello Numbers

hellonumbers.py

```
x = 0;
while x < 10:
    print(x)
    x = x + 1

# crashes here
print(5 + "horse")
```

HelloNumbers.java

```
public class HelloNumbers {
    public static void main(String[] args) {

        int x = 0;
        while (x < 10) {
            System.out.println(x);
            x = x + 1;
        }

        x = "horse"; // program doesn't run
    }
}
```

Reflections on Hello Numbers:

- Before Java variables can be used, they must be declared.
- Java variables must have a specific type.
- Java variable types can never change.
- Types are verified before the code even runs!

Java is statically typed!

- All variables, parameters, and methods must have a declared type.
- That type can never change.
- Expressions also have a type, e.g. “larger(5, 10) + 3” has type int.
- The compiler checks that all the types in your program are compatible **before the program ever runs!**
 - e.g. `String x = larger(5, 10) + 3` will fail to compile.
 - This is unlike a language like Python, where type checks are performed DURING execution.

Larger

Lecture 1, CS61B, Fall 2024

- Welcome!
 - Welcome to 61B
 - 61B Logistics
- **Our First Java Programs**
 - Hello World
 - Hello Numbers
 - **Larger**
 - Reflections on Java
- Workflow
 - Compilation
 - IntelliJ
- HW0: Due Friday!

Coding Demo: Larger

larger.py

```
def larger(x, y):  
    if (x > y):  
        return x  
    return y
```

LargerDemo.java

```
public class LargerDemo {  
    public static larger(x, y) {  
        if (x > y) {  
            return x;  
        }  
        return y;  
    }  
  
}
```

Coding Demo: Larger

larger.py

```
def larger(x, y):  
    if (x > y):  
        return x  
    return y
```

LargerDemo.java

```
public class LargerDemo {  
    public static int larger(int x, int y) {  
        if (x > y) {  
            return x;  
        }  
        return y;  
    }  
  
}
```


Coding Demo: Larger

larger.py

```
def larger(x, y):  
    if (x > y):  
        return x  
    return y  
  
print(larger(-5, 10))
```

LargerDemo.java

```
public class LargerDemo {  
    public static int larger(int x, int y) {  
        if (x > y) {  
            return x;  
        }  
        return y;  
    }  
  
    public static void main(String[] args) {  
        System.out.println(larger(-5, 10));  
    }  
}
```

- Functions must be declared as part of a class in Java.
A function that is part of a class is called a "method."
So in Java, all functions are methods.
- To define a function in Java, we use "public static".
We will see alternate ways of defining functions later.
- All parameters of a function must have a declared type,
and the return value of the function must have a declared type.
Functions in Java return only one value!

Coding Demo: Larger

LargerDemo.java

```
/** Demonstrates creation of a method in Java. */
public class LargerDemo {
    /** Returns the larger of x and y. */
    public static int larger(int x, int y) {
        if (x > y) {
            return x;
        }
        return y;
    }

    public static void main(String[] args) {
        System.out.println(larger(-5, 10));
    }
}
```

Reflections on Java

Lecture 1, CS61B, Fall 2024

- Welcome!
 - Welcome to 61B
 - 61B Logistics
- **Our First Java Programs**
 - Hello World
 - Hello Numbers
 - Larger
 - **Reflections on Java**
- HW0: Due Friday!
- Workflow
 - Compilation
 - IntelliJ

Compilation vs. Interpretation

In Java, compilation and interpretation are two separate steps.



Why make a class file at all?

- `.class` file has been type checked. Distributed code is safer.
- `.class` files are 'simpler' for machine to execute. Distributed code is faster.
- Minor benefit: Protects your intellectual property. No need to give out source.

You can learn more about all this in 61C and particularly 164.

Note: `.class` files are easily reversible into similar looking Java files.

Reflections on Static Typing

The Good:



The Bad:



The Good:

- Catches certain types of errors, making it easier on the programmer to debug their code.
- Type errors can (almost) never occur on end user's computer.
- Makes it easier to read and reason about code.
- Code can run more efficiently, e.g. no need to do expensive runtime type checks.

The Bad:

- Code is more verbose.
- Code is less general, e.g. would need a second larger function to compare non-integers like 5.5.

Compilation

Lecture 1, CS61B, Spring 2025

- Welcome!
 - Welcome to 61B
 - 61B Logistics
- Our First Java Programs
 - Hello World
 - Hello Numbers
 - Larger
 - Reflections on Java
- HW0: Due Friday!
- **Workflow**
 - **Compilation**
 - IntelliJ

Demo: Compilation in Terminal

```
jug ~/.../intro1
$ ls
HelloWorld.java

$ javac HelloWorld.java

$ ls
HelloWorld.class  HelloWorld.java

$ java HelloWorld
Hello World!
```

We won't cover these slides live in class, and they won't be tested on exams. Check out [the videos in the playlist](#) if you're interested.

IntelliJ

Lecture 1, CS61B, Spring 2025

- Welcome!
 - Welcome to 61B
 - 61B Logistics
- Our First Java Programs
 - Hello World
 - Hello Numbers
 - Larger
 - Reflections on Java
- HW0: Due Friday!
- **Workflow**
 - Compilation
 - **IntelliJ**

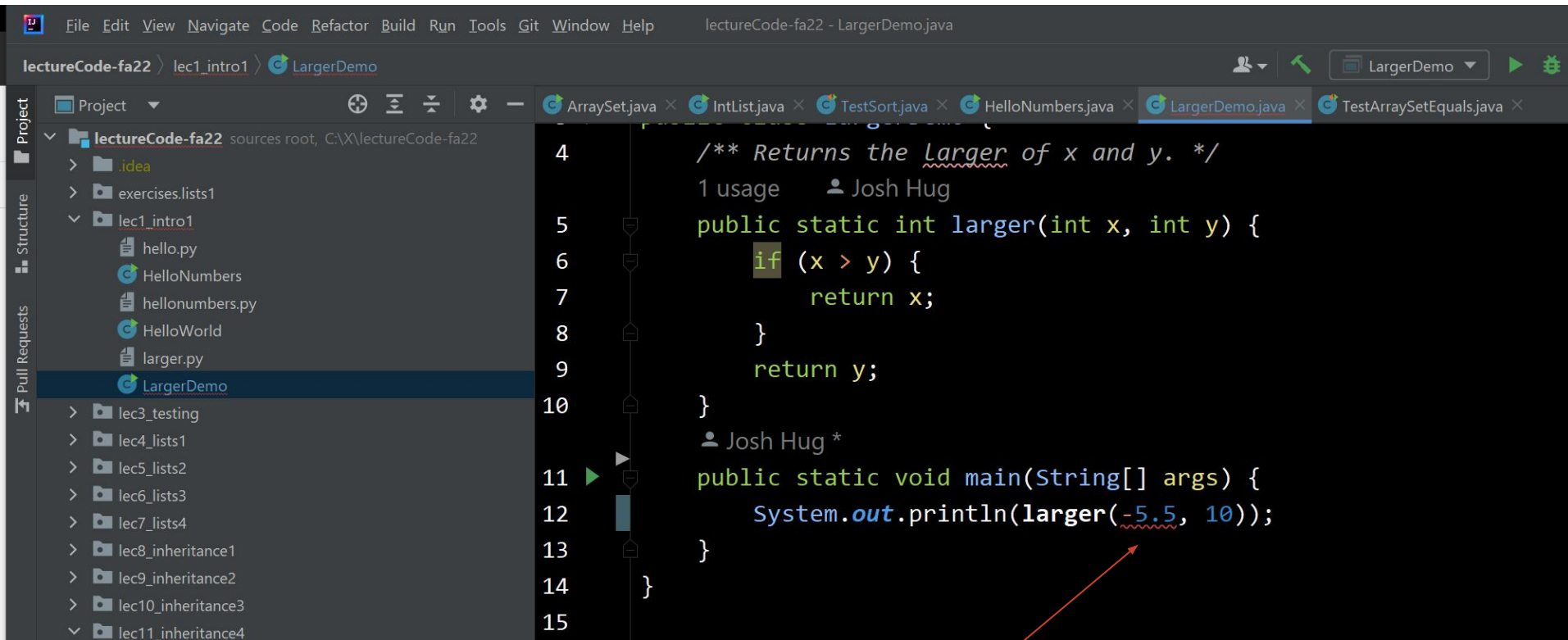
Example Workflows

There are many different workflows for writing programs.

- **Text editor + command line**: (CS61A, CS88). We just did this.
 - **Text editor**: Writing your code.
 - **Command line**: Running your code.
- **Jupyter Notebooks**: (Data 8)
 - Write and run code in the same environment.
- **Integrated Development Environment (IDE)**: (E7, 61B)
 - Write code and run code in the same environment.
 - Tons of additional features like a debugger, code autocomplete, continuous syntax checking, decompilation (from .class to .java), etc.

Let's see what our programs look like in the IDE for our course.

IntelliJ Screenshot



Example feature: IntelliJ automatically and continuously detects syntax errors.

Admonition

Our expectation is that everyone in this class is using IntelliJ.

- It is not strictly required, but staff will provide **no support** for other tools or workflows.

HW0: Due Friday!

Lecture 1, CS61B, Fall 2024

- Welcome!
 - Welcome to 61B
 - 61B Logistics
- Our First Java Programs
 - Hello World
 - Hello Numbers
 - Larger
 - Reflections on Java
 - Object-Oriented Programming
- **HW0: Due Friday!**
- Workflow
 - Compilation
 - IntelliJ

Time to Go Learn Java Basics!

I am not going to spend time in this class covering for loops, while loops, etc. in Java!

- You've seen this all before in some other language.

HW0 is out, and is due this Friday!

- We show you how to translate various Python constructs into Java, you write some short programs.
 - If you haven't seen Python before, you'll be fine.
- Not required to use IntelliJ for HW0 since IntelliJ setup isn't until lab 1.

If you can, start lab 1 early! Most of it is just downloading and installing software.