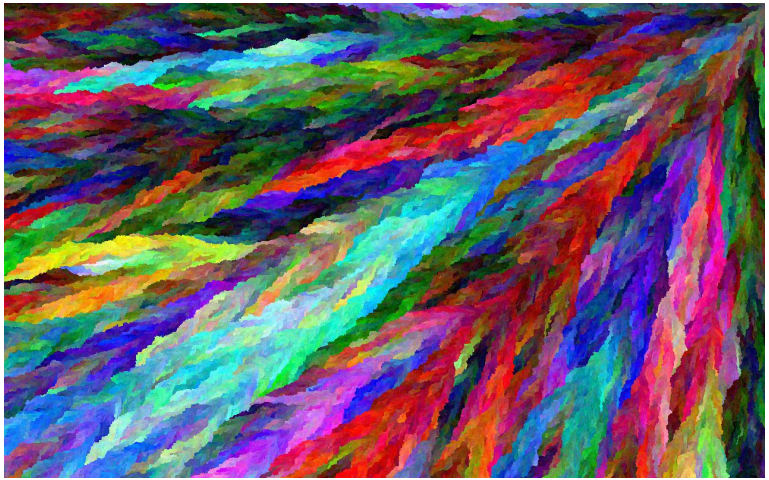
Course website: https://sp24.datastructur.es

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



Lecture 1

Introduction to 61B, Java

CS61B, Spring 2024 @ UC Berkeley

Lecturers: Peyrin Kao, Justin Yokota

Slides Credit: Josh Hug

Welcome to 61B

Lecture 1, CS61B, Spring 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!
- Bonus Slides: Workflow
  - Compilation
  - IntelliJ

61B Overview

What is 61B about?

- 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.
  - Java (not the focus of the course!)

Assumes solid foundation in programming fundamentals, including:

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

Why 61B?

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=NP.
- Once you're done: the confident sense that you can build any software.

Question for You

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

- Job. $$$$$$$
- I want to be able to run my code efficiently (finally)
- I want an A.
- Coding from scratch.
- Greater grasp of data structures and algorithms

Who are you?

- Freshman? Sophomore? Junior? Senior? Grad student? None of the above?
- CS Major? Intending to be a CS Major? Something else?
- CS 61A? Java experience?

61B Logistics

Lecture 1, CS61B, Spring 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!
- Bonus Slides: Workflow
  - Compilation
  - IntelliJ

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).

Evaluation

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.

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://sp24.beacon.datastructur.es/ will allow you to request extensions. These can be retroactive, but we recommend requesting in advance.

- **Short extension** (**24 hours or less**): Immediate approval by automated system.
- **Long extension** (**24 - 72 hours**): GSI will review within one business day.

If you have an emergency and need more than 72 hours, see syllabus.

Class Phase

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.
  - Coding work is entirely dedicated to final project, done in pairs.
  - Slower pace.

Hello World

Lecture 1, CS61B, Spring 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!
- Bonus Slides: Workflow
  - Compilation
  - IntelliJ

Intro to Java

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 only ever written code in MATLAB, 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-fa23

Coding Demo: Hello World

hello.py

HelloWorld.java

```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");
    }
}
```

Java and Object Orientation

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, Spring 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!
- Bonus Slides: Workflow
  - Compilation
  - IntelliJ

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
    }
}
```

Java and Static Typing

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, Spring 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!
- Bonus Slides: Workflow
  - Compilation
  - IntelliJ

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));
    }
}
```

Larger: Reflections

- 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, Spring 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!
- Bonus Slides: 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:

- 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.

Object-Oriented Programming

Lecture 1, CS61B, Spring 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!
- Bonus Slides: Workflow
  - Compilation
  - IntelliJ

CS61A Review: Object-Oriented Programming

- A model for organizing programs
  - Modularity: Define each piece without worrying about other pieces, and they all work together
  - Allows for data abstraction: You can interact with an object without knowing how it's implemented
- Objects
  - An object bundles together information and related behavior
  - Each object has its own local state
  - Several objects may all be instances of a common type
- Classes
  - A class serves as a template for all of its instances
  - Each object is an *instance* of some class

CS61A Review: Constructors

- Constructors: A special method that creates a new object (in other words, a new instance of the class)
  - In Python: `__init__`
  - In Java: Same name as the class
- Can take in additional arguments (in the example, `m`)
- Can be used to initialize *instance variables* (local state) for the new object
- In Java: We also have to *declare* instance variables before using them

Car.java

```java
public class Car {
    public String model;
    public int gas;

    public Car(String m) {
        model = m;
        gas = 5;
    }
}
```

car.py

```python
class Car:



    def __init__(self, m):
        self.model = m
        self.gas = 5
```

CS61A Review: Methods

- Calling a method on an object might change its state
- The object knows how to manage its own state, based on method calls
- In Java: The return value of the method must have a type

Car.java

```java
public class Car {
    public void drive() {
        if (gas < 5) {
            System.out.println("Cannot drive!");
            return;
        }
        gas -= 5;
        System.out.println(model + " goes vroom!");
    }

    public int gasLeft() {
        return gas;
    }

    public void addGas(int amount) {
        gas = gas + amount;
    }
}
```

car.py

```python
class Car:
    def drive(self):
        if (gas < 5):
            print("Cannot drive!")
            return

        gas -= 5
        print(self.model + " goes vroom!")


    def gasLeft(self):
        return self.gas


    def addGas(self, amount):
        gas = gas + amount
```

Java Syntax: this Keyword

- The `this` keyword can be used to access the current object's instance variables or methods
- Unlike Python, where `self` is mandatory, using `this` is not mandatory (as long as variable names are unique)
- More details here

These two methods work exactly the same.

Car.java

```
public int gasLeft() {
   return gas;
}
```

Car.java

```
public int gasLeft() {
   return this.gas;
}
```

Object-Oriented Programming Demo: Car

Car.java

```java
public class Car {
    public String model;
    public int gas;

    public Car(String m) {
        model = m;
        gas = 5;
    }

    public void drive() {
        if (gas < 5) {
            System.out.println("Cannot drive!");
            return;
        }
        gas -= 5;
        System.out.println(model + " goes vroom!");
    }

    public int gasLeft() {
        return gas;
    }

    public void addGas(int amount) {
        gas = gas + amount;
    }
}
```

car.py

```python
class Car:



    def __init__(self, m):
        self.model = m
        self.gas = 10

    def drive(self):
        if (gas < 5):
            print("Cannot drive!")
            return

        gas -= 5
        print(self.model + " goes vroom!")



    def gasLeft(self):
        return self.gas


    def addGas(self, amount):
        gas = gas + amount
```

CS61A Review: Creating Objects

- In Java:
  - We have to declare the variables of type Car before using them
  - The new keyword instantiates a new object
  - We call the constructor to create the object

Car.java

```
public static void main(String[] args) {
   Car c1;
   Car c2;

   c1 = new Car("Honda Civic");
   c2 = new Car("Model T");
}
```

car.py

```
c1 = Car("Honda Civic")
c2 = Car("Model T")
```

CS61A Review: Accessing Objects

- Use dot notation to access methods of an object

Car.java

```
public static void main(String[] args) {
    ...
    System.out.println(c1.gasLeft()); // 5

    c1.drive(); // Honda Civic goes vroom
    System.out.println(c1.gasLeft()); // 0

    c1.addGas(1);
    System.out.println(c1.gasLeft()); // 1

    c1.drive(); // Cannot drive

    System.out.println(c2.gasLeft()); // 5
}
```

car.py

```
...
print(c1.gasLeft()) # 5

c1.drive() # Honda Civic goes vroom
print(c1.gasLeft()) # 0

c1.addGas(1)
print(c1.gasLeft()) # 1

c1.drive() # Cannot drive

print(c2.gasLeft()) # 5
```

Object-Oriented Programming Demo: Car

Car.java

```java
public static void main(String[] args) {
    Car c1;
    Car c2;

    c1 = new Car("Honda Civic");
    c2 = new Car("Model T");

    System.out.println(c1.gasLeft()); // 5

    c1.drive(); // Honda Civic goes vroom
    System.out.println(c1.gasLeft()); // 0

    c1.addGas(1);
    System.out.println(c1.gasLeft()); // 1

    c1.drive(); // Cannot drive

    System.out.println(c2.gasLeft()); // 5
}
```

car.py

```python
c1 = Car("Honda Civic")
c2 = Car("Model T")

print(c1.gasLeft()) # 5

c1.drive() # Honda Civic goes vroom
print(c1.gasLeft()) # 0

c1.addGas(1)
print(c1.gasLeft()) # 1

c1.drive() # Cannot drive

print(c2.gasLeft()) # 5
```

HW0: Due Friday!

Lecture 1, CS61B, Spring 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!**
- Bonus Slides: 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.

Post-Lecture Q&A

If you have questions, come find us in Dwinelle 104!

We have to clear out of the lecture hall for the next class.

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.

Bonus Slides: Compilation

Lecture 1, CS61B, Spring 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!
- **Bonus Slides: 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.

Bonus Slides: IntelliJ

Lecture 1, CS61B, Spring 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!
- **Bonus Slides: Workflow**
  - Compilation
  - **IntelliJ**

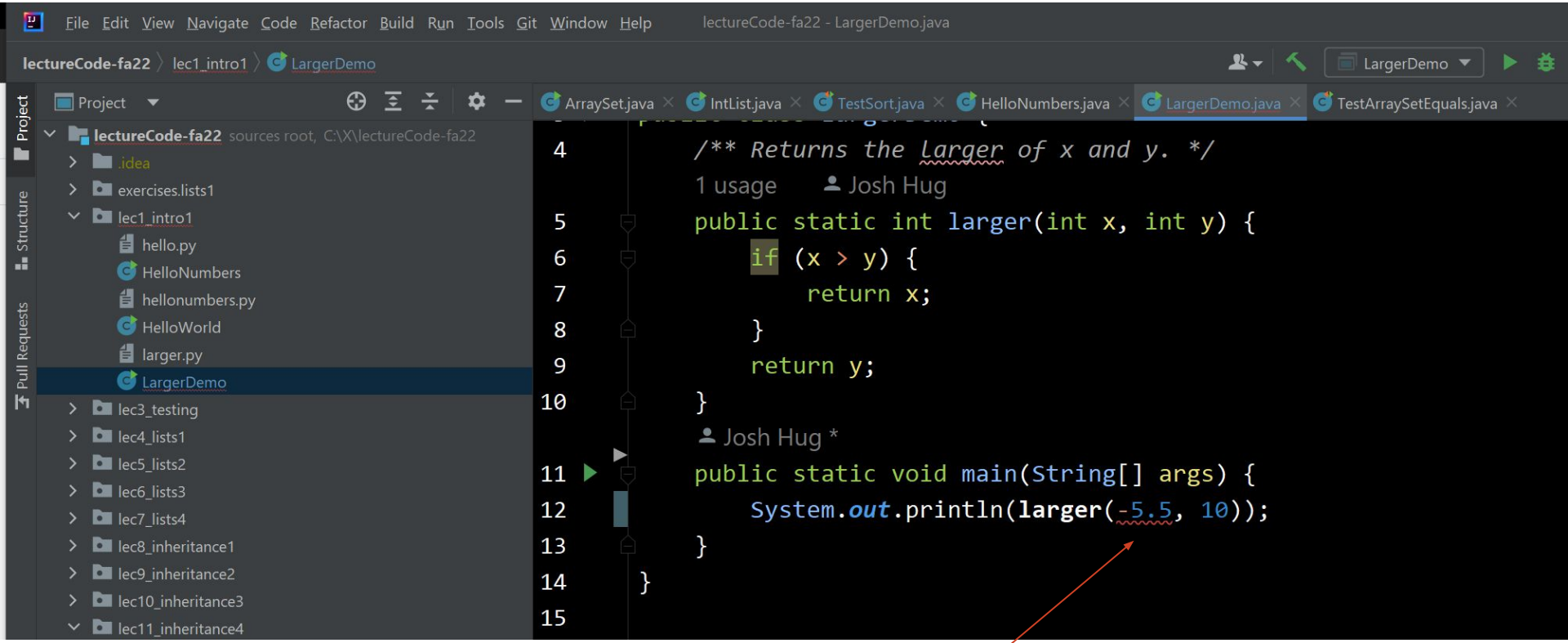
Example Workflows

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.