1: Introduction & Object-Oriented Programming
Lecture 1: Introduction & Object-Oriented Programming

- Introductions
- Motivation
- Logistics
- Object-Oriented Programming Paradigm
- Java Basics

Our team

- Gabe Alzate (head TA)
- Jack Bernstein
- Iren Coskun
- Alex Franklin
- Xander Koo
- Jason Rodolitz
- Cecilia Sanborn
- Huey Sun
INTRODUCTIONS

Who are you?

- College
- Year
- Major
- Enrolled in class
- Permed
- CS51P/CS51J/AP

MAKE SURE TO USE THE SIGN-IN SHEET FOR THE FIRST TWO LECTURES
INTRODUCTIONS

Take an index card and write down...

- Why are you taking this course?
- What do you hope to get out of the course?
Lecture 1: Introduction & Object-Oriented Programming

- Introductions
- Motivation
- Logistics
- Object-Oriented Programming Paradigm
- Java Basics
What is CS062?

- Beginner to intermediate-level course
- **Data structures:** Emphasis on how to organize data in a computer based on problem needs
- **Advanced Programming:** Emphasis on how to write efficient algorithms for modern applications following the Object-Oriented Programming (OOP) paradigm
Why study Data Structures and Algorithms?

- Their impact is broad and far-reaching
- They may unlock the secrets of life and of the universe
- Old roots, new opportunities
- To become a proficient programmer
- For intellectual stimulation
- For fun and profit
- To major/minor in Computer Science
Their impact is broad and far-reaching

What will it take for us to trust algorithms?

Algorithms Aid Search for Source of Spacetime Rumbles

Flawed Algorithms Are Grading Millions of Students’ Essays

Fooled by gibberish and highly susceptible to human bias, automated essay-scoring systems are being increasingly adopted, a Motherboard investigation has found

What People Hate About Being Managed by Algorithms, According to Study of Uber Drivers

by Mareike Möhlmann and Ola Henfridsson

How Healthcare Is Using Big Data And AI To Cure Disease

by Nicole Martin
They may unlock the secrets of life and of the universe
MOTIVATION

Old roots, new opportunities

BRIEF & INCOMPLETE HISTORY OF DATA STRUCTURES

- Stack operations (1946) by Alan M. Turing
- Hash table (IBM, 1955)
- Linked List (Rand Corporation, 1960)
- Binary Search Trees
- Heapsort (1964)
- “Bubble Sort” becomes mainstream (1976)
- Arrays implemented by Computers, Mergesort algorithm invented (John von Neumann, 1945)
- Computer aided proof of graph problem

http://macbookandheels.com/algorithm/2018/10/31/teachingds/
To be a proficient programmer

- “Bad programmers worry about the code. Good programmers worry about data structures and their relationships”
  Linus Torvalds (architect of Linux and git)

- “Algorithms + Data Structures = Programs”
  Niklaus Wirth
For intellectual stimulation

“For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing.”
Francis Sullivan, The Joy of Algorithms
MOTIVATION

For fun and profit

https://www.pinterest.com/pin/89931323793521544/
MOTIVATION

To major/minor in Computer Science

Year Entered Pomona

2015 & before

2016 & 2017

2018 & after

Old Requirements

1. Introductory Courses
   - CSCI 051x
   - CSCI 052
   - CSCI 055/MATH 103
   - CSCI 062
2. Mathematics
   - MATH 060
3. Core Courses
   - CSCI 081
   - CSCI 105
   - CSCI 131
   - CSCI 140
4. Electives
   - 3 additional CS electives
5. Senior Seminar
   - CSCI 190
6. Colloquium
   - All talks during senior year

Old Intro Sequence & New Upper Division

1. Introductory Courses
   - CSCI 051x
   - CSCI 052
   - CSCI 055/MATH 103
   - CSCI 062
2. Mathematics
   - MATH 058/MATH 060
3. Core Courses
   - CSCI 101
   - CSCI 105
   - CSCI 140
4. Electives
   - 3 additional CS electives – no external electives allowed
5. Senior Seminar
   - CSCI 190
6. Colloquium
   - 12 talks during junior year
   - 12 talks during senior year

New Intro Sequence & Old Upper Division

1. Introductory Courses
   - CSCI 051x
   - CSCI 054
   - CSCI 062
2. Mathematics
   - MATH 058/MATH 060
3. Core Courses
   - CSCI 101
   - CSCI 105
   - CSCI 140
4. Electives
   - 3 additional CS electives
5. Senior Seminar
   - CSCI 190
6. Colloquium
   - All talks during senior year

New Requirements

1. Introductory Courses
   - CSCI 051x
   - CSCI 054
   - CSCI 062
2. Mathematics
   - MATH 058/MATH 060
3. Core Courses
   - CSCI 101
   - CSCI 105
   - CSCI 140
4. Electives
   - 3 additional CS electives – no external electives allowed
5. Senior Seminar
   - CSCI 190
6. Colloquium
   - 12 talks during junior year
   - 12 talks during senior year
# A quick overview of lecture topics

<table>
<thead>
<tr>
<th>Topic</th>
<th>Data Structures/Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fundamentals</td>
<td>Arrays</td>
</tr>
<tr>
<td>Basic Data Structures</td>
<td>ArrayLists, Linked Lists, Stacks, Queues, Union-Find</td>
</tr>
<tr>
<td>Sorting</td>
<td>Bubblesort, Selection sort, Insertion Sort, Shellsort, Quicksort, Mergesort, Heapsort, Heaps, Priority Queues</td>
</tr>
<tr>
<td>Searching</td>
<td>BSTs, red-black trees, B-trees, kd-trees, hash tables</td>
</tr>
<tr>
<td>Graphs</td>
<td>BFS, DFS, Prim, Kruskal, Dijsktra</td>
</tr>
<tr>
<td>Concurrency &amp; Parallelism</td>
<td></td>
</tr>
</tbody>
</table>
The advanced programming side of CS62

- In contrast to CS51, labs and assignments will typically be different.
- Labs are shorter and deliverables are due Wednesday midnight.
- Assignments are week-long, due on Tuesday midnight.
- Labs will mostly teach you tools:
  - CLI, Eclipse, Debugger, Unit testing, git, profiling, etc
- In some labs you will implement data structures that we see in lectures.
- Assignments will be deliberately vague and will be using data structures to solve interesting problems.
  - No one will hire you and give you the solution to a problem.
Lecture 1: Introduction & Object-Oriented Programming

- Introductions
- Motivation
- Logistics
- Object-Oriented Programming Paradigm
- Java Basics
Prerequisites

- Officially, CS054/CS052 at Pomona
- We assume you are comfortable with moderate-size programs
  - Loops
  - Conditionals
  - Procedures/methods/functions
  - Objects
  - Recursion
- Comfortable with proofs
- Familiar with concepts of time/memory efficiency

DON’T WORRY!
REVIEW DURING FIRST TWO WEEKS
How can I succeed in CS062?

- Have breakfast, come to class, be on time
- Take notes, participate, ask questions, don’t stay confused
- Review slides and do the assigned reading/problems after each lecture
- Start the assignments early
- Use the tools we learn in the lab (e.g., Debugger)
- Practice writing code on paper
- Learn how to read and write documentation
- Come to office hours/mentor sessions
  - But ask for help *after* you have tried solving a problem by yourself
- Did I say start early?
How can I be a good citizen in CS062?

- Don’t use laptops/tablets/phones/other fancy electronics
  - Unless you have an accommodation

- Be mindful when in office hours/mentor sessions of the number of other students waiting for help.
  - Come with specific questions

- TAs are students too. Respect their time outside mentor sessions.

- Follow the departmental academic honesty policy
  - That’s non-negotiable and we use software to detect plagiarism
What will my average week look like?

- MWF lectures.
- Friday quizzes.
- Wednesday labs (mandatory) due on Wednesday midnight.
- Weekly assignments due on Tuesday midnight.

**Budget at least 8 hours outside the classroom**
Grading summary

- Weekly Programming Assignments: 35%
  - $3^n \%$ penalty for each late day
  - Can take a 3-day extension once - *use wisely*
- Midterm I: 15% (in lab)
- Midterm II: 15% (in lab)
- Final Exam: 25%
- Quizzes: 5%
  - Can skip one quiz - *use wisely*
- Labs: 5%

Resources


  - Brief summary of content
  - Exercises
  - Code

- **Piazza discussion forum**: monitored by the entire staff.

- **Github**: to submit assignments - you cannot make your code publicly available.

- **Office hours**:
  - Papoutsaki - MF 2-5 pm and by appointment. Edmunds 222
  - Kampe - TW 10am-12pm Edmunds 128. TW 4-5:30pm Edmunds 229

- **Mentor sessions**: 6-10 pm, T 6-9 pm, W 8-10 pm, Sa 1-3 pm, Su 8-10 pm, in Edmunds 227

- **Course website**: [http://www.cs.pomona.edu/classes/cs062/](http://www.cs.pomona.edu/classes/cs062/)
Lecture 1: Introduction & Object-Oriented Programming

- Introductions
- Motivation
- Logistics
- Object-Oriented Programming Paradigm
- Java Basics
What is Object-Oriented Programming (OOP)?

▸ “a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships”. Grady Booch

▸ Popular OOP languages: Java, C++, C#, Python (kinda).
What is an object?

- A software bundle of related state and behavior.
- Can have a physical existence e.g., a customer, a ticket, a car.
- Can have an intangible conceptual existence e.g., a meeting, a process.
- **State:** characteristic properties and their values modeled as fields.
  - e.g., a bicycle has variables for current speed (18mph) and gear (5th)
- **Behavior:** methods operate on internal state of objects and serve as the primary mechanism for object-to-object communication.
  - e.g., change gear, apply brakes, speed up or down, etc.
What is a class?

- A blueprint or prototype from which objects are created.
- An object is an instance of a class and the process of creating it is called instantiation.
Practice Time

- Models of real-world objects contain ___ and ___.
- A software object’s state is stored in ___.
- A software object’s behavior is exposed through ___.
- A blueprint for a software object is called a ____.
Answers

- Models of real-world objects contain **fields** and **methods**.
- A software object’s state is stored in **fields**.
- A software object’s behavior is exposed through **methods**.
- A blueprint for a software object is called a **class**.

[https://docs.oracle.com/javase/tutorial/java/concepts/QandE/answers.html](https://docs.oracle.com/javase/tutorial/java/concepts/QandE/answers.html)
Lecture 1: Introduction & Object-Oriented Programming

- Introductions
- Motivation
- Logistics
- Object-Oriented Programming Paradigm
- Java Basics
Declaring classes

```java
public class MyClass {
    // field, constructor, and method declarations
}
```

- Class name is a noun and capitalized by convention.
- The class body is surrounded by curly braces.
A possible implementation of a bicycle class in Java

```java
/**
 * Represents a bicycle
 * @author https://docs.oracle.com/javase/tutorial/java/concepts/class.html
 */
public class Bicycle {
    // instance variables
    private int cadence = 0;
    private int speed = 0;
    private int gear = 1;

    public void changeCadence(int newValue) {
        cadence = newValue;
    }

    public void changeGear(int newValue) {
        gear = newValue;
    }

    public void changeSpeed(int change) {
        speed = speed + change;
    }

    public int getCadence() {
        return cadence;
    }

    public void printGear() {
        System.out.println("Gear:" + gear);
    }

    public String toString() {
        return "cadence:" + cadence + " speed:" + speed + " gear:" + gear;
    }
}
```

- All code in a Java program must belong to a class.
- // comment within a line.
- /* multi-line comment.*/
- /**documentation comment (JavaDoc).*/
- The source code is saved in .java files.
- The name of the class should match the name of the source file e.g., Bicycle.java.
- Curly braces ({ and }) are used to surround bodies of classes, methods, and loops.
- Statements end with a semicolon (;
- Fields cadence, speed, gear represent the state of a bicycle object.
- Methods changeCadence, changeGear, etc. define how the object will interact with the world.
- System.out.println is Java's way of printing a string to the console.
- Override toString if you want to change how objects are printed.
- To run your code you will need a special method called main - there is no main in the Bicycle class.
- You can have a main method per class. Typically one of them will control the program and the rest will be used to test each class.
Bicycle Demo program

```java
/**
 * Basic demonstration of how to work with bicycle objects
 * @author https://docs.oracle.com/javase/tutorial/java/concepts/class.html
 */

public class BicycleDemo {
    public static void main(String[] args) {
        // Create two different Bicycle objects
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();

        System.out.println(bike1);

        // Invoke methods on those objects
        bike1.changeCadence(50);
        bike1.changeSpeed(10);
        bike1.changeGear(2);
        bike1.printGear();
        System.out.println(bike1);

        bike2.changeCadence(50);
        bike2.changeSpeed(10);
        bike2.changeGear(2);
        bike2.changeCadence(40);
        bike2.changeSpeed(-10);
        bike2.changeGear(3);
        bike2.printGear();
        System.out.println(bike1);
    }
}
```

- In the main method, we instantiate two objects of type Bicycle with the `new` keyword, that is two new bicycles are being brought into this world.
- Object name + dot operator + method/variable to create a reference to an object's method/field
  - e.g., `bike1.changeCadence(50);`
- Void methods do not return anything.
  - `printGear` is void
- `System.out.println(someObject)` calls the `toString` method of the class `someObject` belongs to.

**WHAT WILL THIS PROGRAM PRINT?**

```
cadence:0 speed:0 gear:1
Gear:2
cadence:50 speed:10 gear:2
Gear:3
cadence:50 speed:10 gear:2
```
Access Modifiers

- **public** modifier - the field/method is accessible from all classes.

- **private** modifier - the field/method is accessible only within its own class.

- More that we will learn later...
Variables

- Containers for storing data values.
- Java is statically-typed: all variables must be declared along with their data type before they can be used.
  - e.g., `int cadence = 0;`
  - e.g., `String name;`
- Data types: primitives, classes, interfaces, and arrays.
Instance variables (non-static or member fields)

- Declared in a class but outside of any method.
- Each object has its own unique copy of the variable. E.g.,

```java
public class Bicycle {
    private int cadence = 0;
    private int speed = 0;
    private int gear = 1;
}
```

- Invoked as `myObject.variableName`
- It’s always a good idea to keep them `private`. 
Static variables (class fields)

- Declared with the `static` modifier.
- All objects share the same copy. E.g.,

```java
private class Bicycle {
    public static int numberOfBicycles;
}
```
- Invoked as `ClassName.variableName`

**USE SPARINGLY!**
Local variables

- Declared within a method.
- Destroyed after the execution of the method.
- Can only be accessed within the method.
- No access modifier.

```java
public int countToTen() {
    int counter = 0;
    //...
}
```
Naming Variables

- Variable names are case-sensitive.
- No white space.
- Start with small letter.
- Subsequent characters can be letters, digits, $, or _.
- Use full words that make sense.
- If name contains more than two words, capitalize the first letter of each subsequent word. e.g., numberOfBicycles.
- If your variable is a constant, capitalize everything. e.g., PI.
Identifier

- The name of a class, interface, method, or variable.
- Each category has its own naming conventions.

<table>
<thead>
<tr>
<th>Reserved Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract</td>
</tr>
<tr>
<td>assert</td>
</tr>
<tr>
<td>boolean</td>
</tr>
<tr>
<td>break</td>
</tr>
<tr>
<td>byte</td>
</tr>
<tr>
<td>case</td>
</tr>
<tr>
<td>catch</td>
</tr>
<tr>
<td>char</td>
</tr>
<tr>
<td>class</td>
</tr>
<tr>
<td>const</td>
</tr>
<tr>
<td>continue</td>
</tr>
<tr>
<td>default</td>
</tr>
<tr>
<td>do</td>
</tr>
<tr>
<td>double</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>enum</td>
</tr>
<tr>
<td>extends</td>
</tr>
<tr>
<td>false</td>
</tr>
<tr>
<td>final</td>
</tr>
<tr>
<td>finally</td>
</tr>
<tr>
<td>float</td>
</tr>
<tr>
<td>for</td>
</tr>
<tr>
<td>goto</td>
</tr>
<tr>
<td>if</td>
</tr>
<tr>
<td>implements</td>
</tr>
<tr>
<td>import</td>
</tr>
<tr>
<td>instanceof</td>
</tr>
<tr>
<td>int</td>
</tr>
<tr>
<td>interface</td>
</tr>
<tr>
<td>long</td>
</tr>
<tr>
<td>native</td>
</tr>
<tr>
<td>new</td>
</tr>
<tr>
<td>null</td>
</tr>
<tr>
<td>package</td>
</tr>
<tr>
<td>private</td>
</tr>
<tr>
<td>protected</td>
</tr>
<tr>
<td>public</td>
</tr>
<tr>
<td>return</td>
</tr>
<tr>
<td>short</td>
</tr>
<tr>
<td>static</td>
</tr>
<tr>
<td>strictfp</td>
</tr>
<tr>
<td>super</td>
</tr>
<tr>
<td>switch</td>
</tr>
<tr>
<td>synchronized</td>
</tr>
<tr>
<td>this</td>
</tr>
<tr>
<td>throw</td>
</tr>
<tr>
<td>throws</td>
</tr>
<tr>
<td>transient</td>
</tr>
<tr>
<td>true</td>
</tr>
<tr>
<td>try</td>
</tr>
<tr>
<td>void</td>
</tr>
<tr>
<td>volatile</td>
</tr>
<tr>
<td>while</td>
</tr>
</tbody>
</table>
Primitive Data Types

- Java supports 8 primitive data types.
- Primitives use a small amount of memory to represent a single item of data and support certain operations on its value.
- All data of same primitive data type use the same amount of memory.
- Cannot be used to instantiate type variables, that is no `new` keyword.
- Have corresponding object “wrapper” types:
  - Integer, Double, Float, Boolean, etc.
### Primitive Data Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Bits</th>
<th>Default</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8</td>
<td>0</td>
<td>byte b = 10;</td>
</tr>
<tr>
<td>short</td>
<td>16</td>
<td>0</td>
<td>short s = 2;</td>
</tr>
<tr>
<td>int</td>
<td>32</td>
<td>0</td>
<td>int i = 47;</td>
</tr>
<tr>
<td>long</td>
<td>64</td>
<td>0L</td>
<td>long l = 4747L;</td>
</tr>
<tr>
<td>float</td>
<td>32</td>
<td>0.0f</td>
<td>float f = 47.0f;</td>
</tr>
<tr>
<td>double</td>
<td>64</td>
<td>0.0</td>
<td>double d = 47.0;</td>
</tr>
<tr>
<td>char</td>
<td>16</td>
<td>'\u0000'</td>
<td>char c = 'a';</td>
</tr>
<tr>
<td>boolean</td>
<td>1</td>
<td>false</td>
<td>boolean fun = true;</td>
</tr>
</tbody>
</table>

The compiler will assign default values to uninitialized instance and static fields. If you do not initialize local variables you will run into a compile-time error!
The most important primitive data types to know

- **int** - for integers.
- **double** - for real numbers.
- **boolean** - for the set of values \{**true**, **false**\}.
- **char** - for alphanumeric characters and symbols.

**STRINGS ARE NOT PRIMITIVES**
- instead use class **String**.
Classes

- Main data types in Java.
  - e.g., String.
- Thousands more coming with Java by default.
- You can instantiate your own with the `new` keyword.
  - `Bicycle myBike = new Bicycle();`
- Contain fields (can be a primitive or class type) and methods.
- Respond to messages to communicate with the outside world by invoking methods.
- Reference default value is `null`.
A vocabulary refresher for variables

- **Declaration**: state the type of variable and its identifier. A variable can only be declared once. E.g., `int x;`
- **Initialization**: the first time a variable takes a value. E.g., `x = 3;`
  - Can be combined with declaration, e.g., `int y = 3;`
- **Assignment**: discarding the old value and replacing it with a new. E.g., `x = 2;`
- Static or instance variables are automatically initialized with default values, i.e. `null` for references to objects, `0` for `int`, `false` for `boolean`, etc.
- Local variables are not automatically initialized and your code won’t compile if you have not initialized them and you are trying to use them. E.g.,

  ```java
  public void foo() {
    int x;
    System.out.println(x);
    // The local variable x might not have been initialized
  }
  ```
Consider the following class:

```java
public class IdentifyMyParts {
    public static int x = 7;
    public int y = 3;
}
```

a. What are the class variables?

b. What are the instance variables?

c. What is the output from the following code:

```java
IdentifyMyParts a = new IdentifyMyParts();
IdentifyMyParts b = new IdentifyMyParts();
a.y = 5;
b.y = 6;
a.x = 1;
b.x = 2;
System.out.println("a.y = " + a.y);
System.out.println("b.y = " + b.y);
System.out.println("a.x = " + a.x);
System.out.println("b.x = " + b.x);
System.out.println("IdentifyMyParts.x = " + IdentifyMyParts.x);
```
Answers

a. x

b. y

c. a.y = 5
   b.y = 6
   a.x = 2
   b.x = 2
   IdentifyMyParts.x = 2
Lecture 1: Introduction & Object-Oriented Programming

- Introductions
- Motivation
- Logistics
- Object-Oriented Programming Paradigm
- Java Basics
Readings:

- Oracle’s guide: What Is an Object? What Is a Class?
  [https://docs.oracle.com/javase/tutorial/java/concepts/index.html](https://docs.oracle.com/javase/tutorial/java/concepts/index.html)

- Classes and Objects: [https://docs.oracle.com/javase/tutorial/java/javaOO/index.html](https://docs.oracle.com/javase/tutorial/java/javaOO/index.html)

- Variables: [https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html)

Practice Problems:

- How would you model the ticketing system for a local movie theater in OOP?