CS62 Class 1: Intro & Java basics

Java Fundamentals

Slide deck: https://cs.pomona.edu/classes/cs62/#schedule

Slides adopted from Prof. Papoutsaki. Thanks!

Class 1 agenda

- Course overview. What will you learn?
- Course intros
- Getting started in Java
  - Variables
  - Print statements
- Classes and Objects in Java

Course overview

CS62

Search CS62

Canvas Gradescope

- Overview
- Schedule
- Course Staff
- Grading
- Course Policies
- Calendar

CS62: Data Structures and Advanced Programming

Spring 2025 • Pomona College

Lecture: Tues/Thurs 2:45-4:00pm • Edmunds 101

Lab: Weds 7:00-9:50pm • Edmunds 105

Overview

This course couples work on program design, analysis, and verification with an introduction to the study of data structures that are important in the construction of sophisticated computer programs. Students are expected to program in the object-oriented programming language Java. Students will learn to design complex and reliable software engineering systems through writing code modules that are easy to read, debug, test, verify, analyze, and modify.

Students will learn to analyze programs using big-"O" notation to understand their runtimes, as well as apply affordance analysis, to understand the technical and the ethical trade-offs of using different algorithms and data structures to solve computational problems. This course focuses on the efficiency of writing programs (design) and running programs (runtime).

This course is a prerequisite for *most* upper level Computer Science courses.

Prerequisites: The formal prerequisite for this course is CS54 at Pomona. We also assume that all students enrolled are comfortable writing small to medium-sized programs (around 500 lines of code with several interacting classes) in either Java or Python. The knowledge assumed is generally equivalent to the different versions of CS51 as offered at Pomona or the Computer Science advanced placement exam. Be aware that neither CS5 nor CS60 at HMC satisfy the prerequisites for this course.

Please use the course website!!

https://cs.pomona.edu/classes/cs62/

What is a data structure?

What is a data structure?

- Data structures are *abstractions* that help us efficiently organize and store data while writing code.
- Data structures you've seen in CS51P
  - Lists
  - Dictionaries
- In this class, we'll actually implement the data structures ourselves, as well as apply built-in ones to solve software engineering problems
- We'll understand what "efficiently" means (and talk about other metrics that are just as important!)

Course thirds

1 Java & Basic DS

- Reviewing CS51P concepts in Java
- New OOP concepts
  - Inheritance
  - Interfaces, generics
- Basic data structures (DS):
  - ArrayLists
  - Run time & affordance analysis
  - Linked lists (single & doubly)
  - Stacks & queues

2 Sorting & Searching

- Algorithms:
  - Selection & insertion sort
  - Mergesort
  - Quicksort
  - Heapsort
- Data structures:
  - Trees (binary search, 2-3, red-black...)
  - Priority queues, heaps
  - Dictionaries
  - Hashtables
  - Abstract data types

3 Graphs & SWE

- Data structures:
  - Graphs
- Algorithms:
  - Shortest path in a graph
  - Minimum spanning trees
- Final SWE project
  - Human-centered design methods for software engineering
  - Careers panel

...and one checkpoint for each third :)

Learning goals

- Understand how data structures work and how to implement them yourself
  - Understand the time-complexity analysis of algorithms, as well as affordance analysis to understand their ethical trade offs and their history
- Be able to write long, complex, modular, understandable programs (> 1k lines of code) in the OOP (object-oriented programming) paradigm
- Be able to choose the best data structure for an open-ended, real world problem and implement a working solution (*final project*)
  - Feel confident in interviewing for SWE internships (e.g., solve Leetcode problems)

Course assignments + weekly flow

- In CS62, unlike in CS51P, assignments and labs are different.
  - You have 10 labs (to be completed during the Wednesday night lab), and 10 programming assignments
  - Labs are meant to teach SWE skills (command line, Git, debugging, etc.)
  - Assignments are released in lab and **due Tues 11:59pm**
- 3 checkpoints + 1 final project
- Most labs will also start with a 5-10 minute **quiz** on the lecture material in the last week
- Note: the last week of this course will be on Zoom (4/29+) since I have conference travel (but we will finishing covering all course material before then)

What about “advanced programming?”

- Assignments will be deliberately vague and will be using appropriate data structures to solve interesting problems.
  - “Write a program to implement this game”, rather than “Please fill out these methods in this pre-built class…”
- Realistically, no one will hire you and give you the steps to solve a problem.
- In this class, you should develop the intuition to understand how to approach problems - and we’re here to help!

Our wonderful TAs!

Kellie Au (she/her)

Junior

Adrian Clement (he/him)

Junior

Asya Lyubavina (she/her)

Junior

Dylan O'Connor (he/him)

Junior

Francisco Morales Puente (he/him)

Junior

Grading

- 30% checkpoints (10% each)
- 30% weekly homework assignments
- 30% final project
- 5% quizzes
- 5% labs
- You can submit checkpoint corrections within a week of grades being released for up to 50% back (answer key will be released after regrades are due)
- You can retake quizzes within a week in my OH. Your lowest score is dropped

Course policies

- Late days: like in CS51P, all assignments have an automatic extension until 6:59pm on Wednesday (right before lab) if you don't finish them by Tuesday midnight
  - If you need more time, please email/Slack me *before* the extension period is active
- Students with accommodations should reach out to the SDRC to schedule alternate checkpoint proctoring times ASAP
- All policies are flexible - just talk to me! We're here to help you succeed!
- As a final note: this class feels more like CS51P compared to CS54, but it is way harder and way more work. We're really ramping up the programming. If you find yourself overwhelmed, please go to OH/mentor hours!

You have agency!

- To discuss in lab tomorrow:
  - How should we incorporate more Leetcode style problems in this course? (The assignments/labs are largely the same as last semester.) Should they be in the quizzes? Should we have an extra credit oral interview opportunity?
  - What should this class's AI policy be? (Ban all usage? OK to use in instructional formats?) If you violate this policy, what should the consequences be?
  - Other course and participation norms

Course intros

Prof. Li

they/them • jingyi.li@pomona.edu • Edmunds 111 • jingyi.me

- Just teaching CS62
  OH: Tues 11a-12p, Weds 4-5pm, Thurs 1-2p

- Research: human-computer interaction, specifically in art creation tools. I run the Doodle Lab.

- Things that make me happy:
  - drawing/painting/cosplaying/sewing
  - going to concerts, interior design, reading, Pokémon
  - birding, biking, the sun

Your turn!

- Name
- Pronouns (if you'd like)
- Did you take 51P with me or did you pass out?
- 1 thing that you're looking forward to this semester
- 1 thing you're worried about this semester

Getting started in Java

Java basics

- One of the most popular general-purpose programming languages.
- Java follows the object-oriented programming paradigm which means that our code is organized as cooperative collections of **objects**, each of which represents an **instance** of some **class**.
- Java code is written in .java files. Each Java file has one Java class which matches the name of the file.
  - e.g., Lecture1.java will have a `Lecture1` class where we'll write all of our code.
- In order to run a Java program, we will need a special main method.
- We will use VS Code as an IDE (Integrated Development Environment).
- In contrast to Python, we will use curly braces ({}) instead of tabs to create logical blocks of code.
- Single-line comments follow // and multi-line are enclosed within /**/.

Example Java file `Lecture1.java`

These need to match

```java
public class Lecture1 {
    public static void main(String[] args) {
        //This is a comment

        /*
         * This is a multi-line comment.
         * Hi!
         */
    }
}
```

A hypothetical scenario

- We want to write a program for the Office of Registrar to organize information about Pomona students.
- Let's think of what information we would need about a Pomona student. E.g.,:
  - Name
  - Email
  - Pomona ID
  - The year they entered Pomona
  - Academic standing
  - Have they graduated
  - Etc.

```python
class Basket:
    """
    This class simulates a fruit basket
    attr: capacity: (int) the capacity of the basket
    attr: fruits: (dict) a dictionary of fruit and its count, e.g., {"fruit": num}
    """

    def __init__(self, capacity=10):
        """
        creates an initially empty basket of fruit with the given capacity
        :param capacity (int): maximum number of fruit that can be held.  10 if not specified
        """
        self.capacity = capacity
        self.fruits = dict()

    def get_num_fruit(self):
        """
        :return: returns the number of fruit in the basket as an int
        """
        num_of_fruits = 0
        for num in self.fruits.values():
            num_of_fruits += num

        return num_of_fruits

    def get_max_fruit(self):
        """
        :return: returns the capacity of the basket as an int
        """
        return self.capacity
```

Recall from 51P…

Variables

Variable review

- Variables have types and values
- References to data stored in memory
  - E.g., in Python: `age = 18`
- Used to reference and manipulate stored information
  - E.g., in Python: `age = age + 2`
- What variable **types** do you know?
  - Integers, floats, strings, booleans, lists…

Declaring and initializing variables

- Unlike Python, Java is **statically-typed**: all variable types must first be declared before use:
  - `dataType variableName = value;`
- For example:
  - `int numberOfCS62Students = 17;`
    - int means it can hold integers, that is positive and negative whole numbers.
    - The name of the variable is numberOfCS62Students.
    - = assigns the value on the right to the variable on the left.
    - The variable is initialized to 17.
    - You always need to finish a statement in Java using a semi-colon ;.

Assigning new values

- Once a variable is declared, I can reference it elsewhere in the program and assign to it a new value.
  - `variableName = newValue;`
- For example:
  - I could change the number of students to 18, if a new student were to join:
  - `numberOfCS62Students = 18;`
- Note that once a variable has been declared, we do **not** declare again its type. But don't forget the semi-colon.

Assigning new values with static typing

- Recall since Python wasn't an explicitly typed language, you could do stuff like this and it would work:
  - ```
    x = 4
    x = "hello"
    ```
  - ```
    def area(x, y):
        return x * y
    ```
  - ```
    >>> print(area(4, 5))
    20
    ```
  - ```
    >>> print(area("happy", 3)
    "happyhappyhappy"
    ```

your mental model expects the type of x and y to be int, but Python will not reject other types (like string, since string multiplication works)

- Question: What happens when you type this in Java?
  - ```
    int x = 4;
    x = "hello";
    ```
- Answer: a compiler error `incompatible types: String cannot be converted to int`
    ```

Aside: Compiler versus runtime errors

- One benefit of having a *statically typed* language like Java is now most errors are *compiler errors:* Your code will not even compile (i.e., run) without fixing them.
  - This is because Java has an under the hood “type checker” that verifies your code has correct typing
- When your code runs, but it errors during the execution, that is called a *runtime error* (e.g., trying to access an out of bound index in an array)

Naming conventions: camelCase

- Naming variables is very hard. They should be accurately descriptive and understandable to another reader (and to you, days later).
- It should start with a lowercase letter such as `id`, `name`.
- It should not start with the special characters like `&`, `$`, `_`.
- They should be one word. If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as `firstName`, `lastName`.
  - This is known as **camelCase**.

in contrast, Python's variable naming conventions is called snake_case: use underscores.
eg: first_name instead of firstName

https://www.javatpoint.com/java-naming-conventions

Worksheet time!

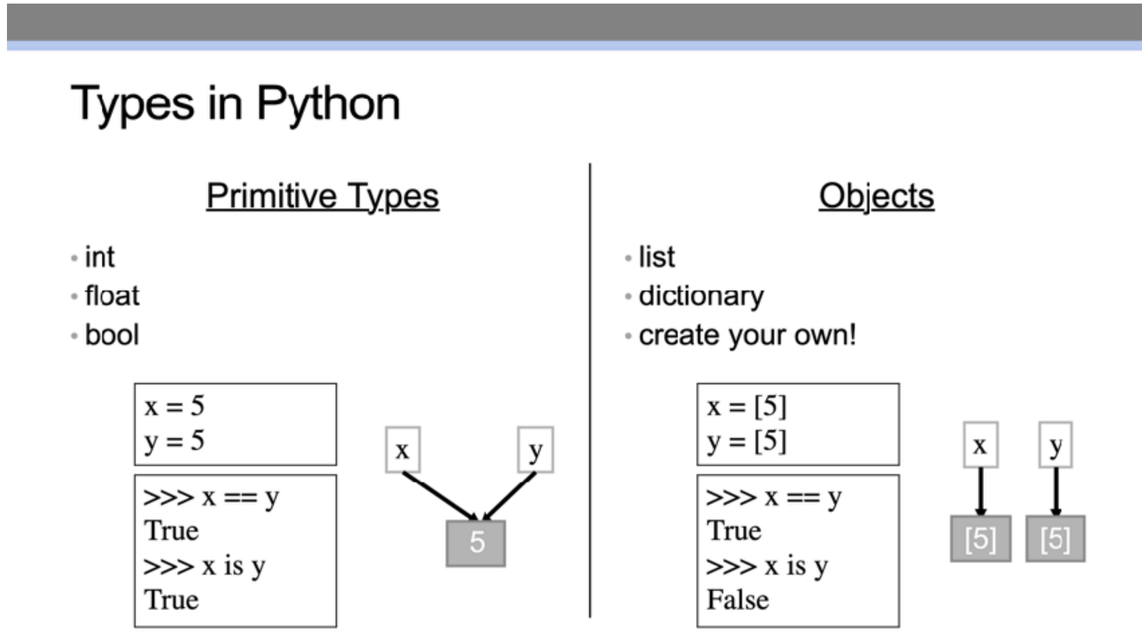
- Do problems 1a & 1b on your worksheet.
  - Declare a variable that stores the number of CS classes you have taken before CS62 at Pomona and initialize it to the appropriate number.
  - Now assume you access this variable at the end of this semester. Assign to it the new value that corresponds to the total number of CS classes you will have taken, including CS62 (and potentially CS101).

Worksheet answers

- You should end up with something like:
  - `int numberOfCSClasses = 2;`
  - `numberOfCSClasses = 3;`

Primitive data types

- Recall: In Python, there are primitive data types (int, float, bool) and types that are *objects* (lists, dictionaries, user defined classes)
- In addition to int, Java supports in total eight **primitive** data types. A primitive type is predefined by Java and is named by a reserved keyword (that means they have a special meaning. e.g., I can't have a variable named int).
- The 8 primitive data types in Java:
  - `byte, short, int, long, float, double, boolean, char`



The most important primitive data types to know

- `int` - for integers.
  - e.g., `int numberOfCS62Students = 40;`
- `double` - for decimal-point numbers.
  - e.g., `double temperatureCelsius = 27.5;`
- `boolean` - for the set of values `{true, false}`.
  - `boolean lovingCS62 = true;`
    - *Note that in contrast to Python,* `true` *and* `false` *are not capitalized.*
- `char` - for single alphanumeric characters and symbols.
  - `char firstLetter = 'a';`

Note: `float` is also for decimals, but we'll prefer using `double` in this class. Floating point math has rounding errors - you'll learn why in CS105

Reserved words

- You can't have these as variable names:

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

Strings

- Character strings are not primitive data types but are supported through the `java.lang.String` class. Note that `String` is capitalized.
- We enclose strings in double quotes. For example:
  - `String name = "Jingyi";`
- Note that **single quotes** are reserved for the `char` data type.
  - `char firstLetter = ‘J';`

Print statements

Print statements

- The method `System.out.println()` is used to print an argument that is passed to. For example:
  - `System.out.println("Hello World");`
  - `System.out.println(name);`
    `//will print Jingyi`
  - `System.out.println(numberOfCS62Students);`
    `//will print 18`

String concatenation

- Like in Python, Strings are concatenated with the + operator, as in `"Hello," + " world" + "!"` which results in `"Hello, world!”`
  - Note the spaces in the strings: `" world”`
- The + operator is widely used in print statements, e.g.,
  - `System.out.println("My name is " + name + " and I will be teaching " + numberOfCS62Students + " students this semester");`
  - Note that in contrast to Python, you do *not* need to convert non-string arguments to string, this is done automatically.

Worksheet time!

- Do problem 1c on your worksheet.
  - Declare and initialize a variable whose type is a primitive and pass it into a print statement, using string concatenation at least once.

Worksheet answers

- There are many different ways to do this, e.g.,:
  - `int years = 4;`
  - `years += 4;`
  - `System.out.println("Trump will be in office for " + years + " years. :(");`

Returning to our hypothetical scenario

- We want to write a program for the Office of Registrar to organize information about Pomona students.
- Let's think of what information we would need about a Pomona student. E.g.,:
  - Name
  - Email
  - Pomona ID
  - The year they entered Pomona
  - Academic standing
  - Have they graduated
  - Etc.

Let's save this information as variables!

What are the types of each variable?

- Name -> String
- Email -> String
- Pomona ID -> int or String
- The year they entered Pomona -> int
- Academic standing -> String
- How many credits they have taken so far -> int
- Have they graduated -> boolean

But this was for ONE student

- Would we need to make a variable for every single student at Pomona?
- And how can we logically organize them together so that it is clear which variables correspond to which student?
- What if we need to change information about a student?
- What if we want to distinguish between unique information (e.g., name) and shared information across all students (e.g., current semester)?
- **Our code just doesn't scale up.**

Classes & Objects in Java

Object-oriented programming to the rescue

- **Objects**: logical bundles of software of related **state** (data) and **behavior** (procedures working on that data).
- **State:** the individual characteristics stored in **variables** (or fields).
  - e.g., name, ID, year entered Pomona, etc.
- **Behavior: methods** (functions) operate on internal state of objects and serve as the primary mechanism for object-to-object communication.
  - Determine academic standing based on student's credits and GPA, award them Latin Honors based on GPA, etc.

Object-oriented programming to the rescue

- Going back to our Python example…

…and we wrap it all up in a custom **class**

state variables:

self.capacity

self.fruits

behavior methods:

get_num_fruit()

get_max_fruit()

```python
class Basket:
    """
    This class simulates a fruit basket
    attr: capacity: (int) the capacity of the basket
    attr: fruits: (dict) a dictionary of fruit and its count, e.g., {"fruit": num}
    """

    def __init__(self, capacity=10):
        """
        creates an initially empty basket of fruit with the given capacity
        :param capacity (int): maximum number of fruit that can be held.  10 if not specified
        """
        self.capacity = capacity
        self.fruits = dict()

    def get_num_fruit(self):
        """
        :return: returns the number of fruit in the basket as an int
        """
        num_of_fruits = 0
        for num in self.fruits.values():
            num_of_fruits += num

        return num_of_fruits

    def get_max_fruit(self):
        """
        :return: returns the capacity of the basket as an int
        """
        return self.capacity
```

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.
- In our example, a class would be a general blueprint for what defines a Pomona student in general terms. An object would be an actual instance of a student whose information we specified based on that general blueprint.

Declaring a class

```
public class ClassName {

    // variables (state)

    // methods (behavior)

}
```

- The class body is surrounded by curly braces.
- Class name is a noun and capitalized by convention.

Writing our first class

- To solve our problem, let's make a PomonaStudent.java file and within it write a PomonaStudent class:

```
public class PomonaStudent {


}
```

Writing our first class

- Now, we need to define the *state* through creating *variables* that store our data. We just list these below the class header (unlike in Python, where they have to go in __init__()).

```
1  public class PomonaStudent {
2
3      String name;
4      String email;
5      int id;
6      int yearEntered;
7      String academicStanding;
8      boolean graduated;
9
10 }
```

Worksheet time!

- Do problem 2a on your worksheet.
  - Assume you are volunteering at Claremont Priceless Pets and are writing a big application so they can digitally manage their pet adoptions (right now, it's all printed paper).
  - You have determined you want to make one class for each type of pet.
  - Define a class `Cat` and declare variables that correspond to a cat's name, sex, age, days spent in rescue, and whether it has been adopted.

Worksheet answers

```
public class Cat {
    String name;
    String sex;
    int age;
    int daysInRescue;
    boolean adopted;
}
```

sex could also be coded as an int (0 = male, 1 = female, 2= intersex) or other categorical variables, but strings give us the most flexibility and human-readableness

Instantiating objects

- To instantiate a new object use the `new` keyword. E.g.,
  - `PomonaStudent student1 = new PomonaStudent();`
- Once you have instantiated an object, you can change its state through the dot operator. E.g.,
  - `student1.name = "Ravi Kumar";`
  - `student1.email = "rkjc2023@mypomona.edu";`

This is syntax heavy, but important!
Compare to Python's syntax:
student1 = PomonaStudent()
student1.name = "Ravi Kumar"

Instantiating objects: do it in main()

- We typically (but not always) instantiate objects in the `main` method of a class. E.g.,

```
public static void main(String[] args){
    PomonaStudent student1 = new PomonaStudent();
    student1.name = "Ravi Kumar";
    student1.email = "rkjc2023@mypomona.edu";
    student1.id = 1234;
}
```

Making a constructor

- We can also initialize fields during instantiation.
- To do, we will need a special type of method, **a constructor.**
- Constructors are methods that have the same name with the class and can take 0 or more parameters that typically correspond to all or a subset of the variables. E.g.,

Compare to Python's constructor syntax:
def __init__(self, name, email, id):

```
public PomonaStudent(String studentName, String studentEmail, int studentId){
    this.name = studentName;
    this.email = studentEmail;
    this.id = studentId;
}
```

note the "this" keyword - same to Python's "self"

parameters

- We can now instead write, to instantiate:

```
PomonaStudent student2 = new PomonaStudent("Ravi Kumar",
"rkjc2023@mypomona.edu", 1234);
```

Compare to Python's instantiation:
student2 = PomonaStudent("Ravi Kumar", "rkjc2023@mypomona.edu", 1234)

arguments

The default, no argument constructor

- If we don't specify a constructor, Java makes implicitly one for us, the zero-argument constructor.
  - All variables are initialized to their default value, i.e.,
    - `int->0`
    - `double ->0.0`
    - `boolean -> false`
    - and any object reference (e.g., String) is set to `null.`
- The no-argument constructor is what we invoked before:
  - `PomonaStudent student1 = new PomonaStudent();`
- Note: Once we specify a constructor, we HAVE to explicitly create a no-argument constructor; our code above would stop working otherwise.

you can think about "null" as Python's "None"

Multiple constructors: overloading

- Unlike in Python, in Java, you can have more than one constructors that specify different ways that an object of our class can be instantiated.
  - E.g., a different constructor could only initialize a student's name upon instantiation. i.e.:

```
public PomonaStudent(String studentName) {

    name = studentName;

}
```

- This is known as overloading. Java knows which constructor you mean to use by matching the number, type, and order of arguments you are passing to the equivalent parameters.

Our code, with constructors

```java
public class PomonaStudent {

    // state variables
    String name;
    String email;
    int id;
    int yearEntered;
    String academicStanding;
    boolean graduated;

    // unlike python, you can have multiple constructors
    // they just need to differ in the # of arguments
    public PomonaStudent() {
    }

    public PomonaStudent(String name){
        this.name = name;
    }

    public PomonaStudent(String name, String email, int id){
        this.name = name;
        this.email = email;
        this.id = id;
    }

}
```

Accessing instance variables using .

- Once we have instantiated an object, we can access its instance (or member) variables using the **dot operator**. E.g.,

```
public static void main(String[] args){
    PomonaStudent student2 = new PomonaStudent("Ravi Kumar”, "rkjc2023@mypomona.edu",
    1234);
    System.out.println(student2.name); //prints Ravi Kumar
    student2.name = "Jingyi Li";
    System.out.println(student2.name); //prints Jingyi Li - we changed the name
}
```

- We cannot access instance variables without specifying the object. For example:

```
public static void main(String[] args){
    System.out.println(name); //won’t compile - WHOSE name?
    System.out.println(PomonaStudent.name); //incorrect - remember, we need to use
                                            //object names, not class names!
}
```

`this` keyword

- The keyword `this` refers to the current object. We can use it to differentiate between instance variables and parameters when they have the same name.

```
public PomonaStudent(String name, String email, int id) {
    this.name = name;
    this.email = email;
    this.id = id;
}
```

Instance variables

Parameters

```
public class Main {
  int x;

  // Constructor with a parameter
  public Main(int x) {
    this.x = x;
  }

  // Call the constructor
  public static void main(String[] args) {
    Main myObj = new Main(5);
    System.out.println("Value of x = " + myObj.x);
  }
}
```

This example prints `Value of x = 5`

Without the `this.x = x` in the constructor (if the line was just `x = x`) this example would print `Value of x = 0` (x's default value)

Our code, with object instantiation in main()

```java
public class PomonaStudent {

    // state variables
    String name;
    String email;
    int id;
    int yearEntered;
    String academicStanding;
    boolean graduated;

    // unlike python, you can have multiple constructors
    // they just need to differ in the # of arguments
    public PomonaStudent() {
    }

    //you don't *need* this.name if the parameter is a different variable
    public PomonaStudent(String studentName){
        name = studentName;
    }

    //but if your parameters and variables are the same, you need the keyword this
    public PomonaStudent(String name, String email, int id){
        this.name = name;
        this.email = email;
        this.id = id;
    }

    public static void main(String[] args){


        PomonaStudent student1 = new PomonaStudent(); //uses the default constructor
        student1.name = "Ravi Kumar";
        student1.email = "rkjc2023@mypomona.edu";
        student1.id = 1234;

        PomonaStudent student2 = new PomonaStudent(name:"Ravi Kumar", email:"rkjc2023@mypomona.edu", id:1234);
        System.out.println(student2.name); //prints Ravi Kumar
        student2.name = "Jingyi Li";
        System.out.println(student2.name); //prints Jingyi Li



    }
}
```

Worksheet time!

- Do problem 2b on your worksheet.
  - Fill in the Cat constructor to take 3 arguments: name, age, and sex.
  - In the main method, instantiate two objects of type Cat. Initialize their name, age, and sex to whatever you choose.
  - Once you instantiate the two Cat objects, initialize their days in rescue to whatever number you want.
  - Finally, adopt a cat!

Worksheet answers

```java
public class Cat {

    String name;
    String sex;
    int age;
    int daysInRescue;
    boolean adopted;

    public Cat(String name, String sex, int age){
        this.name = name;
        this.sex = sex;
        this.age = age;
    }

    Run main | Debug main | Run | Debug
    public static void main(String[] args){

        Cat cat1 = new Cat(name:"Sesame", sex:"female", age:3);
        Cat cat2 = new Cat(name:"Mochi", sex:"unknown", age:1);
        cat1.daysInRescue = 3;
        cat2.daysInRescue = 47;
        cat2.adopted = true;
    }

}
```

Summary

- The object-oriented programming paradigm captures *state* (through variables) and *behaviors* (through methods). Each class defines the kinds of state and behaviors each *instance* of the class should have. (Class = PomonaStudent, instance = student1, student2)
- We need to define *constructors* in our class to define how we make instances, or *instantiate* new objects
- We then actually instantiate objects usually in the main() function

```java
public class PomonaStudent {

    // state variables
    String name;
    String email;
    int id;
    int yearEntered;
    String academicStanding;
    boolean graduated;

    // unlike python, you can have multiple constructors
    // they just need to differ in the # of arguments
    public PomonaStudent() {
    }

    //you don't *need* this.name if the parameter is a different variable
    public PomonaStudent(String studentName){
        name = studentName;
    }

    //but if your parameters and variables are the same, you need the keyword this
    public PomonaStudent(String name, String email, int id){
        this.name = name;
        this.email = email;
        this.id = id;
    }

    Run main | Debug main | Run | Debug
    public static void main(String[] args){


        PomonaStudent student1 = new PomonaStudent(); //uses the default constructor
        student1.name = "Ravi Kumar";
        student1.email = "rkjc2023@mypomona.edu";
        student1.id = 1234;

        PomonaStudent student2 = new PomonaStudent(name:"Ravi Kumar", email:"rkjc2023@mypomona.
        edu", id:1234);
        System.out.println(student2.name); //prints Ravi Kumar
        student2.name = "Jingyi Li";
        System.out.println(student2.name); //prints Jingyi Li



    }
}
```

Lecture 1 wrap-up

- Your TODO: Fill in course survey by EOD
- See you in lab tomorrow (7pm Edmunds 105)!
  - In general, you should read the labs before coming, but no need to for this first one.
- On Thursday, we'll announce mentor hour times - if any of this review was fast/confusing, please review these slides, mark questions, and go ask for help.

Resources

- Variables: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html
- Oracle's guide: What Is an Object? What Is a Class? https://docs.oracle.com/javase/tutorial/java/concepts/index.html
- Classes and Objects: https://docs.oracle.com/javase/tutorial/java/javaOO/index.html