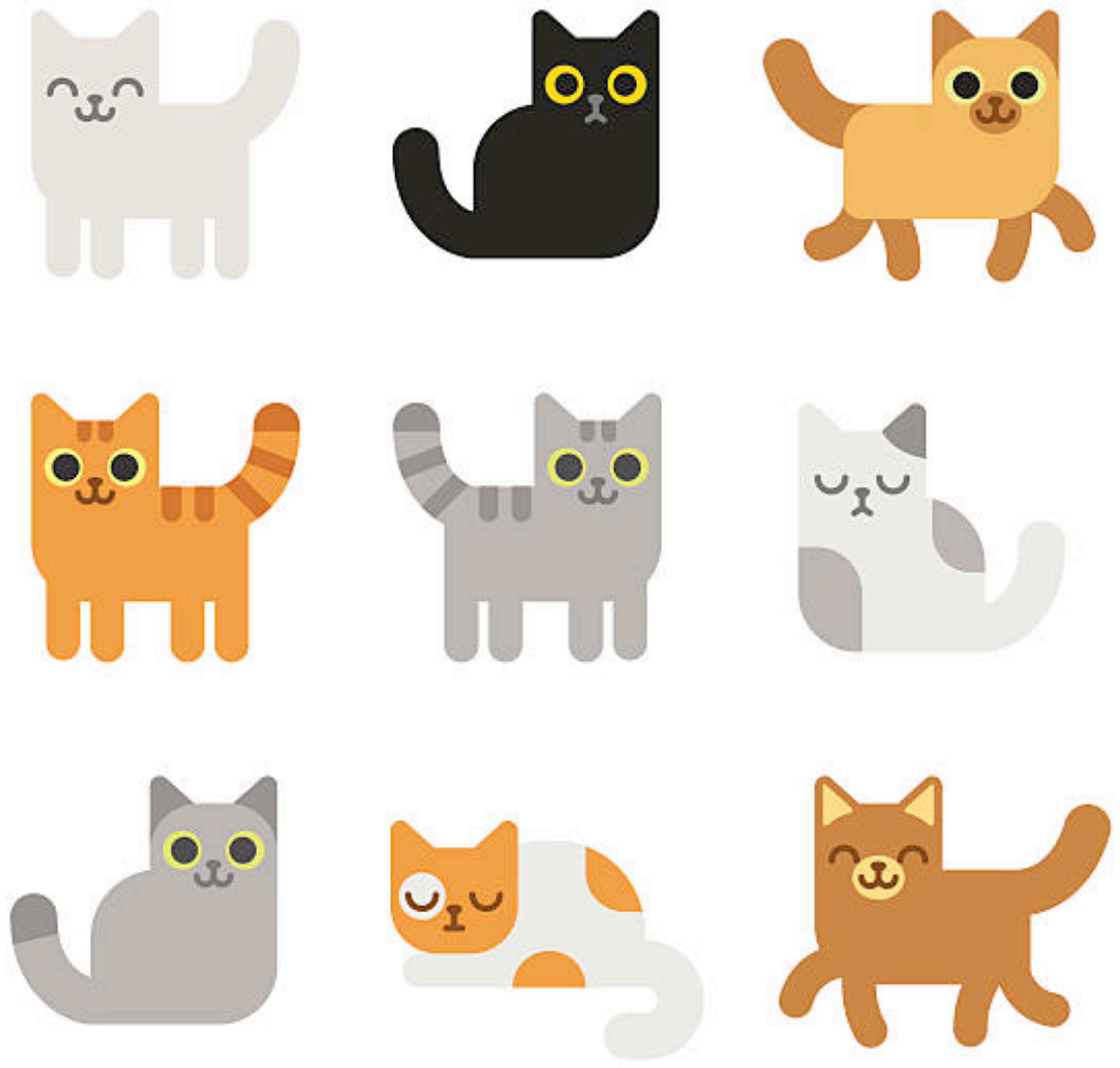
CS62 Class 2: More Classes

Java Fundamentals

```
public class Cat {
    String name;
    public Cat(String name) {
        this.name = name;
    }
}
```

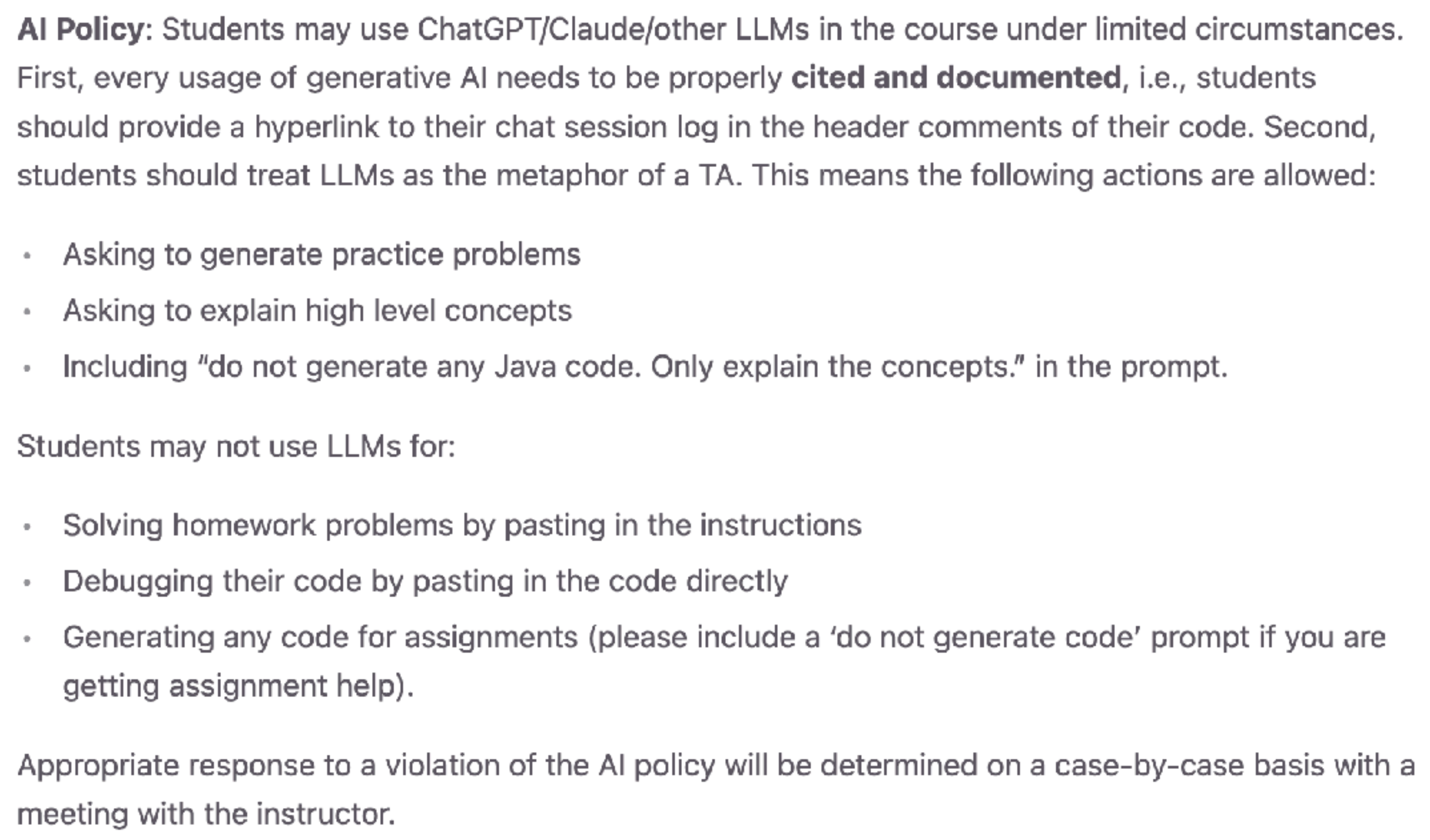
Review: if you're here early, make sure you can explain this code to the person next to you.



```
Cat fluffy = new Cat("fluffy");        Cat bear = new Cat("bear");
```

From lab: course norms & AI policy

- Course norms
  - It's OK to be wrong! No bad or stupid questions. It's good to ask questions!
  - Acknowledge each other's efforts
  - If you participate a lot in one class, try to wait a few seconds before answering to give other people a chance to answer
- What you all added:
  - Be nice! Be respectful! Be collaborative
  - Check in if lecture is going too fast
  - We all come from different backgrounds and levels of experience
  - Be open to helping & asking for help



AI Policy: Students may use ChatGPT/Claude/other LLMs in the course under limited circumstances. First, every usage of generative AI needs to be properly **cited and documented**, i.e., students should provide a hyperlink to their chat session log in the header comments of their code. Second, students should treat LLMs as the metaphor of a TA. This means the following actions are allowed:

- Asking to generate practice problems
- Asking to explain high level concepts
- Including "do not generate any Java code. Only explain the concepts." in the prompt.

Students may not use LLMs for:

- Solving homework problems by pasting in the instructions
- Debugging their code by pasting in the code directly
- Generating any code for assignments (please include a 'do not generate code' prompt if you are getting assignment help).

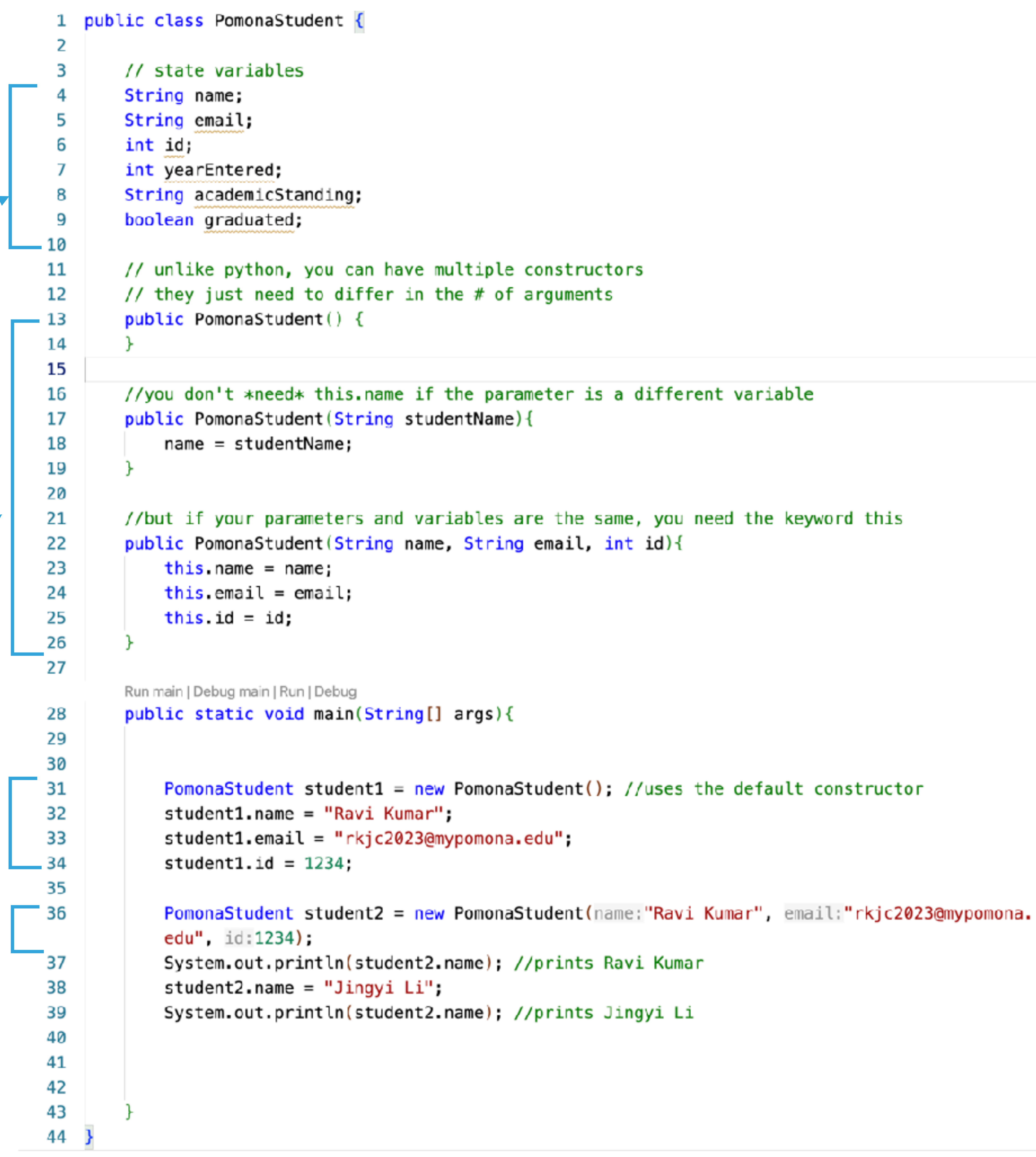
Appropriate response to a violation of the AI policy will be determined on a case-by-case basis with a meeting with the instructor.

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 2: More Java Basics

- Behaviors & Methods
  - `Static` keyword
  - String representation of objects
- Access modifiers & data hiding (`public` vs `private` keywords)
- Java Syntax (conditionals, loops)
- Intro to Arrays

Methods

Java technically doesn't have functions

- You can call it a function, but Java only has *methods*
  - Methods are what we think of as functions, but located inside a class. Java is an OOP language, which means everything needs to be in a class. Thus, every function is a method.
  - (Compare to Python, where most of your code was function definitions, or Haskell, where everything has to be a function.)
- Method syntax:

You have to declare the return type before the name of the function

```
public int add(int x, int y){
    return x + y;
}
```

You also have to declare the type of the parameters

Compare with Python:

```
def add(x, y):
    return x + y
```

PomonaStudent class with a getter and setter method

```
public class PomonaStudent {

    String name;
    String email;
    int id;
    int yearEntered;
    String academicStanding;
    boolean graduated;

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

    public void setYearEntered(int yearEntered){
        this.yearEntered = yearEntered;
    }

}
```

getter and setter naming convention:

`getVarName(), setVarName(varName)`

Review question: Why do we need this.yearEntered in the setter, but not the getter?

Answer: in Java, the “this” keyword is only necessary when the parameter variable name is the same as the instance variable name. The getter has no arguments, so yearEntered is always referring to the instance variable. The setter needs the this keyword to differentiate.

Instance methods

- Instance methods are a collection of grouped statements that perform a logical operation and control the behavior of objects (e.g., getters and setters): basically, the methods you use in your class objects
- By convention, method names should be a verb (+ noun) in lowercase.
- Method signature: method name and the number, type, and order of its parameters.
- Once a method is invoked/called, the control goes back to the calling program as soon as a `return` statement is reached. If it does not return anything the return value is `void`. E.g.,
  - `public void printName(){System.out.println(name);}`
- Can be overloaded (same name, different number, type, or order of parameters). This is common for constructors. Note that **constructors do not have a return type**.
- Invoked using the **dot operator**, e.g.,
  - student1.printName();

Scope & local variables

- Local variables are temporary variables created within a method or constructor.
- Once the last line of the method/constructor is reached, the local variable ceases to exist.
- Access modifiers (e.g., `public`) CANNOT be used for local variables.
- There is *no* default value for local variables, so local variables should be declared with a type and assigned with an initial value before the first use.

```
public int add(int x, int y){
  int sum = x + y;
  return sum;
}
```

`static` **keyword**

Static variables and methods

- Static (or class) variables are variables **shared across all objects**. E.g.,
  - `static int studentCounter;`
  - This means each unique Pomona student instance will all share the same studentCounter.
- **Static methods:** When a method only accesses static (and local) variables, then it can be defined as static. E.g.,

```
static void graduateAllStudents(){
  studentCounter = 0;
}
```

- Can be accessed in instance methods **through the class name**, without needing to instantiate an object. E.g.,
  - `System.out.println(PomonaStudent.studentCounter);`
- In HW1, where we aren't really using the OOP paradigm and just writing 3 methods, all the methods are declared *static*.

PomonaStudent class with a static variable & method

```
public class PomonaStudent {

    String name;
    String email;
    int id;
    int yearEntered;
    String academicStanding;
    boolean graduated;

    static int studentCounter;

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

    public int getYearEntered(){
        return yearEntered;
    }

    public void setYearEntered(int yearEntered){
        this.yearEntered = yearEntered;
    }

    public static void graduateAllStudents(){
       studentCounter = 0;
    }
}
```

By convention, we declare in order:

1. instance variables
2. static variables
3. constructors
4. getters/setters
5. other methods

PomonaStudent class with a static variable & method

```
public class PomonaStudent {

    String name;
    String email;
    int id;
    int yearEntered;
    String academicStanding;
    boolean graduated;

    static int studentCounter;

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

    public int getYearEntered(){
        return yearEntered;
    }

    public void setYearEntered(int yearEntered){
        this.yearEntered = yearEntered;
    }

    public static void graduateAllStudents(){
       studentCounter = 0;
    }
}
```

Question: Why can we just write studentCounter in main? Why not student1.studentCounter?

A: because main is a static class, it can directly access static variables. (student1.studentCounter technically works, but you'll get a warning - it's better to do PomonaStudent.studentCounter)

Question: Why do we need to write student1.getYearEntered() in main, but not student1.graduateAllStudents()? Why is just graduateAllStudents() OK?

A: because main is a static class, it can directly access static methods. (student1.graduateAllStudents() technically works, but you'll get a warning - it's better to do PomonaStudent.graduateAllStudents())

```
public static void main(String[] args) {
   PomonaStudent student1 = new
   PomonaStudent("name", "email@pomona.edu", 123);
   student1.setYearEntered(2022);
   System.out.println(student1.getYearEntered());
   System.out.println(studentCounter);
   graduateAllStudents();
   System.out.println(studentCounter);
}
```

Summary of instance vs static variables & methods

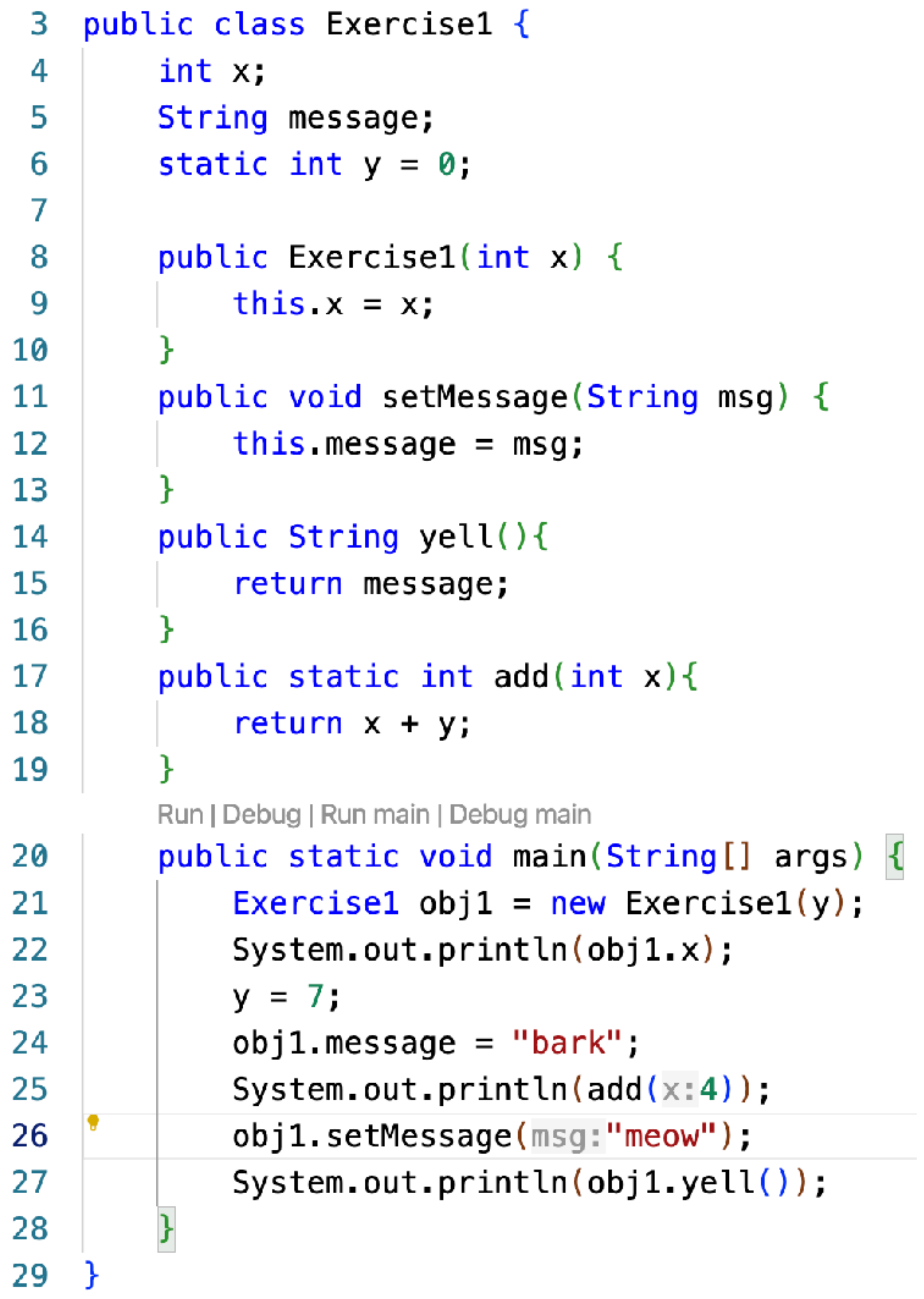
- Instance methods can access instance variables and instance methods directly.
- Instance methods can access static variables and static methods directly (use the class name, e.g., `PomonaStudent.studentCounter`).
- Static methods can access static variables and static methods directly.
- Static methods **cannot** access instance variables or instance methods directly—they must use an object reference. (e.g., `getName()` does not work in main, but `student1.getName()` does)
    - This is what “`Error: Cannot make a static reference to the non-static field`” means
- Static methods cannot use the `this` keyword as there is no instance of an object for this to refer to.

Constant variables (`final` keyword)

- If you want a variable to be constant, that is its value to remain unchanged once it is initialized, you can use the keyword `final`. E.g.,
    - `final int LEVELS = 5;`
- It is conventional to capitalize the variable name to convey it is a constant.
- It is common for a final variable to also be static. E.g.,
    - `static final double PI = 3.141592653589793;`

Worksheet time!

- Do problem 1 on your worksheet. Work in a group of 2-4.



```java
public class Exercise1 {
    int x;
    String message;
    static int y = 0;

    public Exercise1(int x) {
        this.x = x;
    }
    public void setMessage(String msg) {
        this.message = msg;
    }
    public String yell(){
        return message;
    }
    public static int add(int x){
        return x + y;
    }
    public static void main(String[] args) {
        Exercise1 obj1 = new Exercise1(y);
        System.out.println(obj1.x);
        y = 7;
        obj1.message = "bark";
        System.out.println(add(x:4));
        obj1.setMessage(msg:"meow");
        System.out.println(obj1.yell());
    }
}
```

Worksheet answers

- It prints
  0
  11
  meow

.toString()

String representation of an object

- If we want to print an object, we must override the method toString. e.g.,

```
public String toString(){
     return "Name: " + name + "\nemail: " + email + "\nid: " + id;
}
public static void main(String[] args){
     PomonaStudent student1 = new PomonaStudent("Ravi Kumar",
     “rkjc2023@mypomona.edu", 1234);
     System.out.println(student1);
}
```

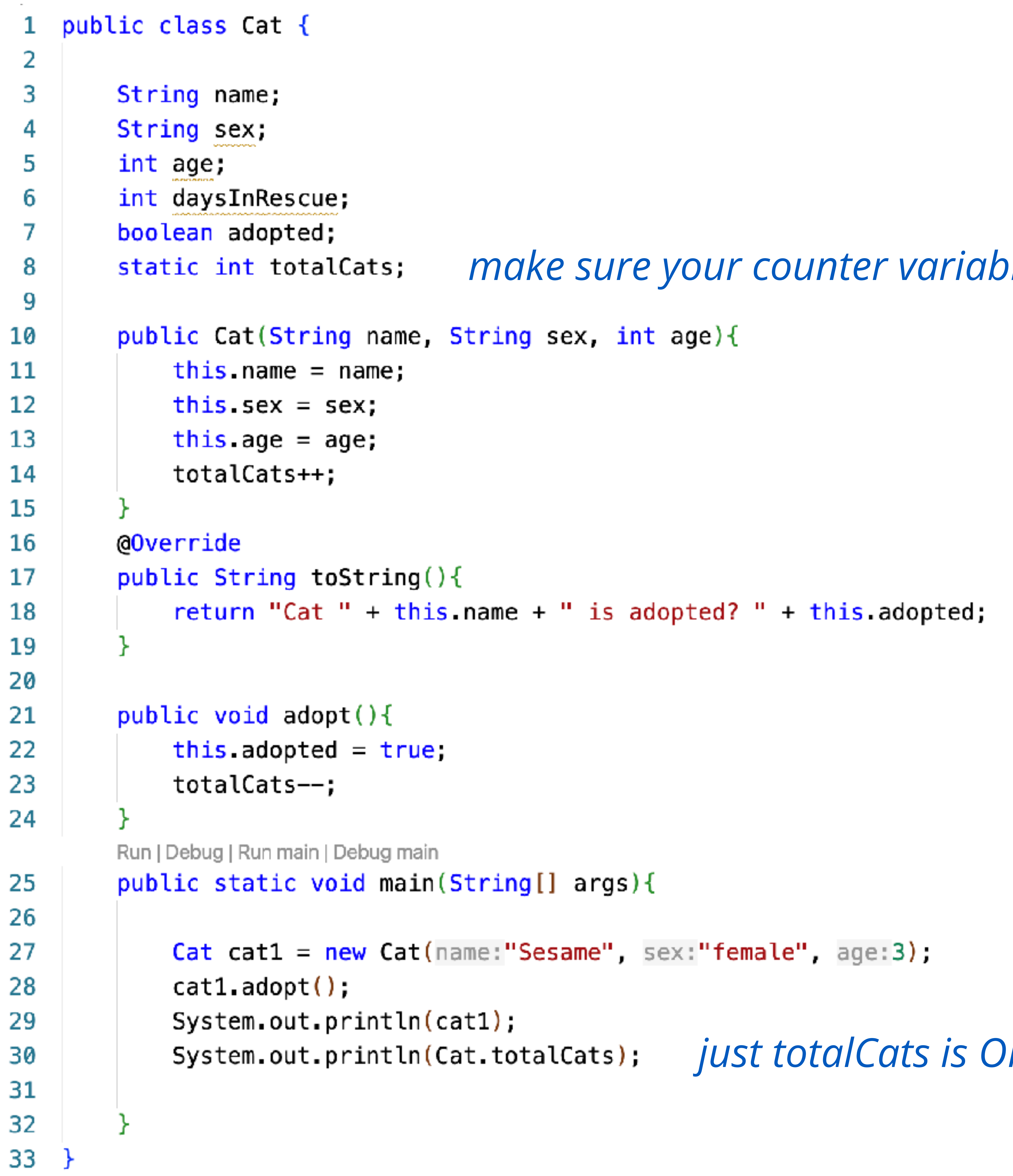
- Will print:
  - Name: Ravi Kumar
  - email: rkjc2023@mypomona.edu
  - id: 1234

compare to Python’s

```
def __str__():
    return “Name: “ + self.name
```

Worksheet time!

- Do problem 2 a-e on your worksheet. Work in a group of 2-4.
    a. Add a counter in the Cat class that represents the total number of cats in the rescue.
    b. Update the constructor to increase the counter by one every time a Cat object is created.
    c. Write an adopt() method that updates a cat's adoption status and decreases the counter.
    d. Update the toString() method to say the cat's name and if was adopted or not.
    e. Create a new cat and adopt it in main. Print out the cat object.



```java
public class Cat {

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

    public Cat(String name, String sex, int age){
        this.name = name;
        this.sex = sex;
        this.age = age;
        totalCats++;
    }
    @Override
    public String toString(){
        return "Cat " + this.name + " is adopted? " + this.adopted;
    }

    public void adopt(){
        this.adopted = true;
        totalCats--;
    }
    Run | Debug | Run main | Debug main
    public static void main(String[] args){

        Cat cat1 = new Cat(name:"Sesame", sex:"female", age:3);
        cat1.adopt();
        System.out.println(cat1);
        System.out.println(Cat.totalCats);

    }
}
```

make sure your counter variable is static!

this keyword not necessary

just totalCats is OK too

Access keywords & data hiding

Data hiding (aka encapsulation)

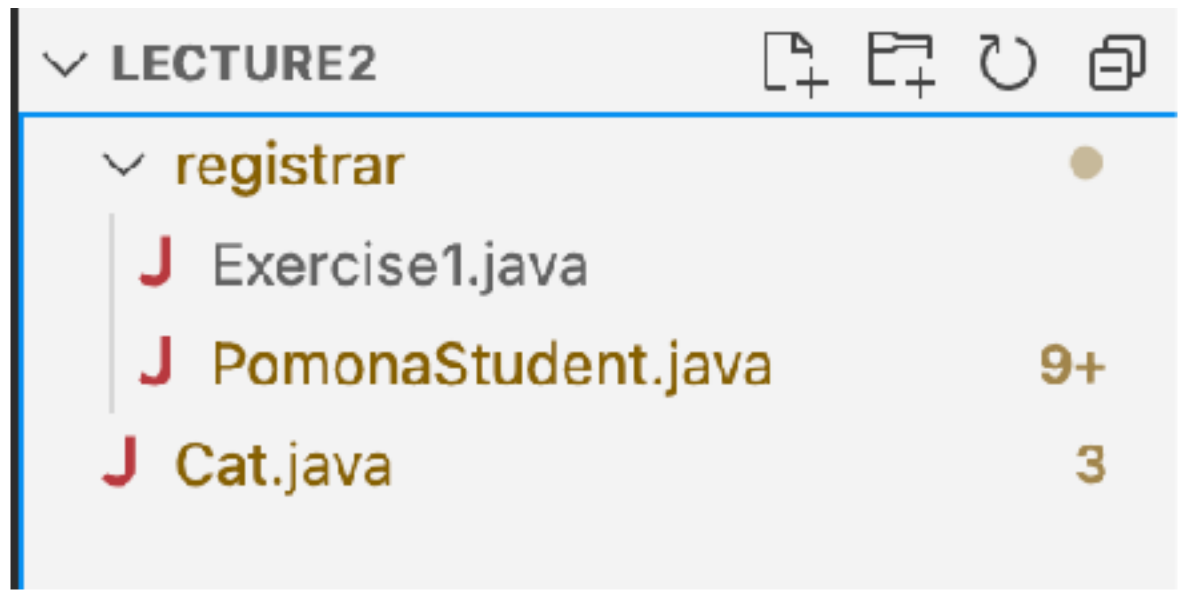
- Data hiding is a core concept in Object-Oriented Programming.
- We encapsulate data and related methods in one class and we restrict who can see and modify data.
    - For example, FERPA protects the privacy of students so the Registrar cannot share their academic record freely, even if it's their parents who request it.
- Java uses access modifiers to set the access level for classes, variables, methods and constructors.

Access Modifiers: public, private, default, protected

- You are already familiar with the public keyword. E.g., `public class PomonaStudent`.
- For classes, you can either use public or *default*:
  - public: The class is accessible by any other class. E.g.,
    - `public class PomonaStudent`
  - *default*: The class is only accessible by classes in the same package (think of it as in the same folder. More soon). This is used when you don't specify a modifier. E.g.,
    - `class PomonaStudent`
- For variables, methods, and constructors, you can use any of the following:
  - `public:` the code is accessible by any other class
  - `private`: The code is only accessible within the declared class
  - *default*: The code is only accessible in the same package. This is used when you don't specify a modifier
  - `protected:` The code is accessible in the same package and subclasses (more later).

Package

- A grouping of related classes that provides access protection and name space management. E.g.,
  - `java.lang` and `java.util` for fundamental classes or `java.io` for classes related to reading input and writing output.
- Packages correspond to folders/directories.
- Lower-case names. E.g.,
  - `package registrar;`
  - at top of file and file has to be within registrar folder
- `import java.util.*;`
  - for including all classes.
- or `import java.util.Arrays;`
  - for more specific access.



demo: using PomonaStudent in Exercise1, since they're both in the registrar package

https://docs.oracle.com/javase/tutorial/java/package/packages.html

Data Hiding

- To follow the concept of data hiding, we prefer to define instance variables as `private`.
- We provide more lax (i.e. `default, protected,` or `public`) getter and setter methods to access and update the value of a `private` variable.

PomonaStudent class with data hiding

```
package registrar;
public class PomonaStudent {

    private String name;
    private String email;
    private int id;
    private int yearEntered;
    private String academicStanding;
    private boolean graduated;
    private static int studentCounter;

    String getName() {
        return name;
    }

    void setName(String name) {
        this.name = name;
    }

    String getEmail() {
        return email;
    }


    void setEmail(String email) {
        this.email = email;
    }
…
```

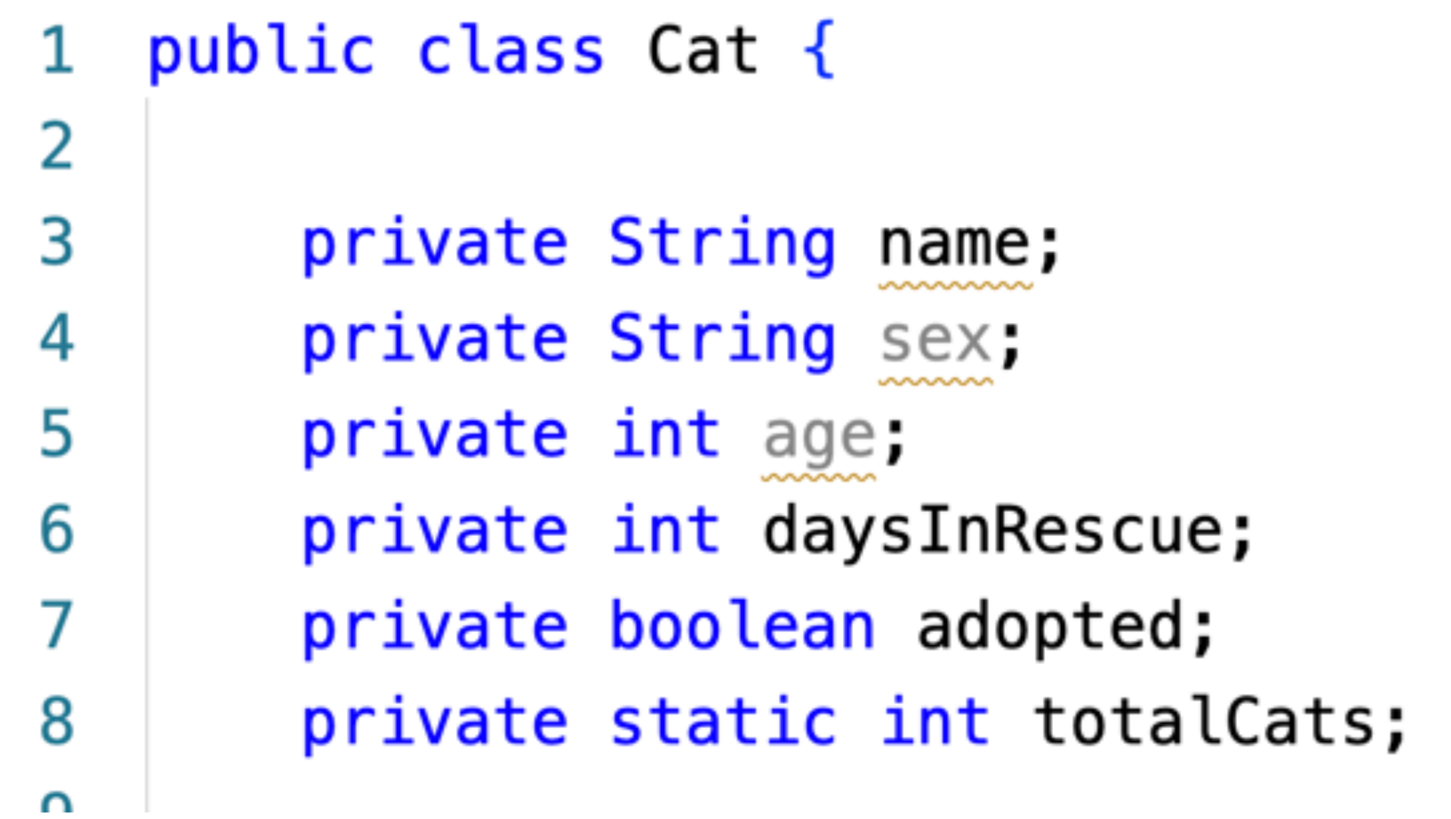
we have moved the file to the registrar/ folder and declared `package registrar`

all the instance variables have been declared `private`

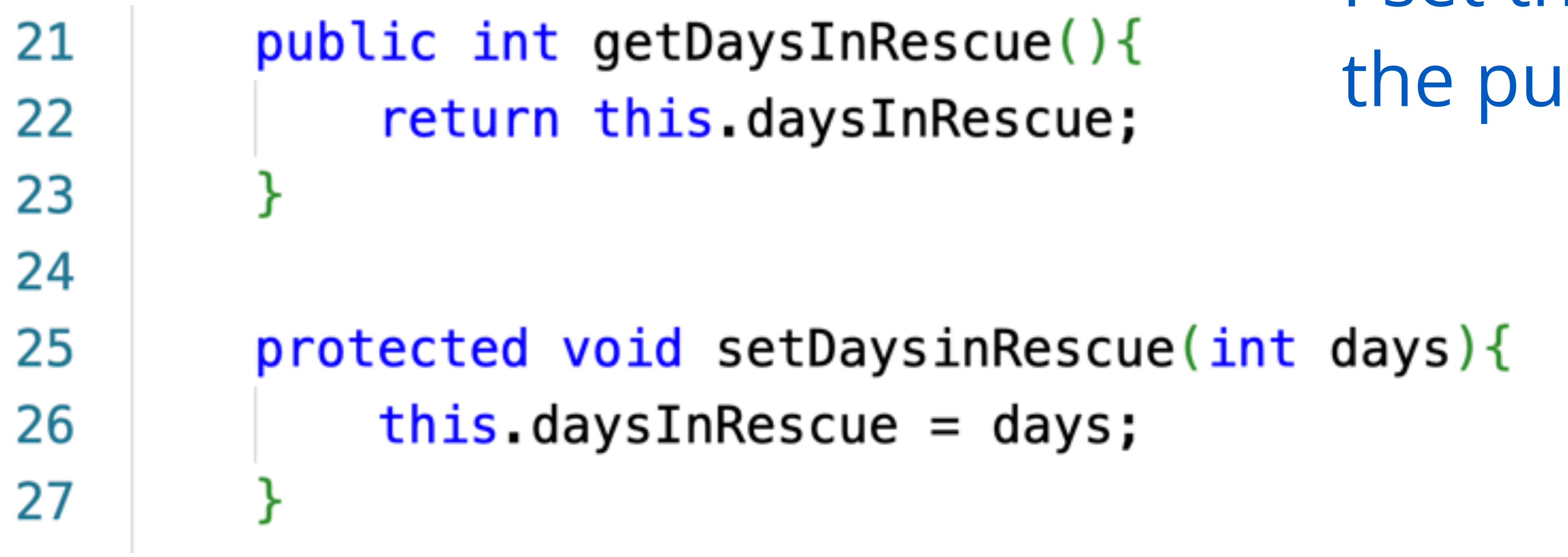
getters and setters are `default` access (any code in the package can use them)

Worksheet time!

- Do problem 2 f-g on your worksheet. Work in a group of 2-4.
    a. Update all the instance variables to be private.
    b. Define a getter method that returns the days spent in rescue, and a setter method that updates the days spent. Make sure they have the correct access modifiers.
    c. Create a new cat, set it to have spent 20 days in rescue, and print out the number of days.



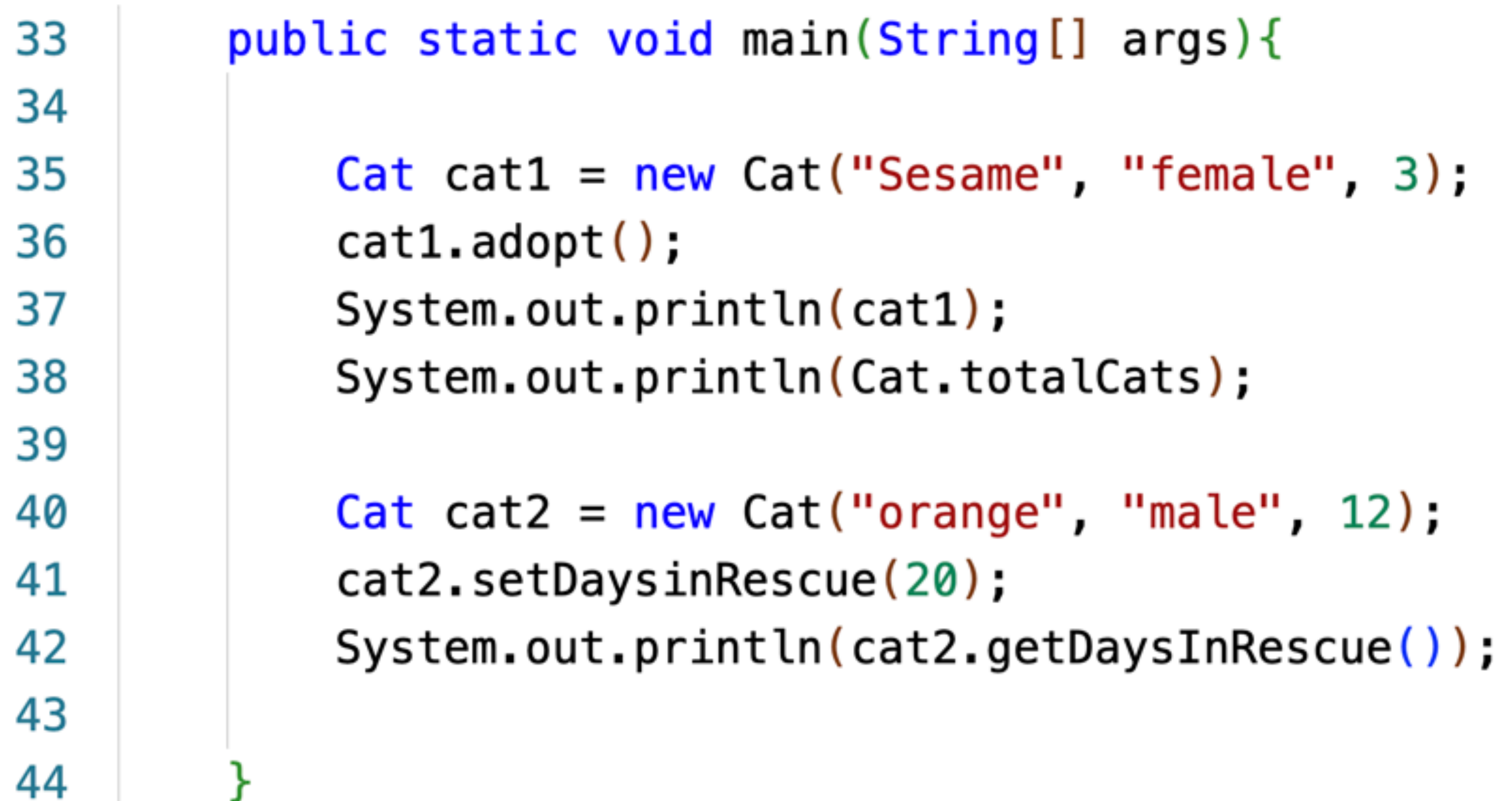
```java
public class Cat {

    private String name;
    private String sex;
    private int age;
    private int daysInRescue;
    private boolean adopted;
    private static int totalCats;
```



```java
    public int getDaysInRescue(){
        return this.daysInRescue;
    }

    protected void setDaysinRescue(int days){
        this.daysInRescue = days;
    }
```

I set the getter to public because it's OK if the public can see this variable

It's dangerous to set setters to public - what if there's a data attack - so I chose protected, but default/private are also good



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

        Cat cat1 = new Cat("Sesame", "female", 3);
        cat1.adopt();
        System.out.println(cat1);
        System.out.println(Cat.totalCats);

        Cat cat2 = new Cat("orange", "male", 12);
        cat2.setDaysinRescue(20);
        System.out.println(cat2.getDaysInRescue());

    }
```

remember, we have to call getters and setters now - *technically* you could directly access cat2.daysInRescue since it's all the Cat class, but it's good programming practice to always use the getter/setter methods for private instance variables

More Java syntax (conditionals, loops)

`if-else if-else` statement

- The most basic of control flow statements.
- Execute a certain section of code only if a particular test evaluates to true. Optionally, if not, execute another.
- Basic syntax:

```
if (expression) {
     statement
}
else if (expression) { //optional, can have many of these
     statement
}
else {  //also optional
     statement
}
```

`if–else if–else` example

```
int testscore = 76;
char grade;

if (testscore >= 90) {
    grade = 'A';
} else if (testscore >= 80) {
    grade = 'B';
} else if (testscore >= 70) {
   grade = 'C';
} else if (testscore >= 60) {
    grade = 'D';
} else {
    grade = 'F';
}
System.out.println("Grade = " + grade);
```

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/if.html

`while` loop

- Repeatedly execute a block of code as long as a specific condition is true.
- Basic syntax:

```
while (condition) {
```

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/while.html

```
  // code block to be executed

}
```

- Make sure your condition terminates otherwise you will enter an infinite loop.

while loop example

```
int i = 0;

while (i < 3) {

    System.out.println("CS62 will become my favorite class");

    i++;

}
```

- Will print:

```
CS62 will become my favorite class

CS62 will become my favorite class

CS62 will become my favorite class
```

Compare to Python:

```
i = 0
while i < 3:
    print(“i love cs62”)
    i += 1
```

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/while.html

for loop

- Repeatedly execute a block of code for a specific number of times:
- Basic syntax:

```
for (initialization; termination; increment) {
  // code block to be executed
}
```

- The `initialization` expression initializes the loop; it's executed once, as the loop begins.
- When the `termination` expression evaluates to false, the loop terminates.
- The `increment` expression is invoked after each iteration through the loop; it is perfectly acceptable for this expression to increment or decrement a value.

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html

for loop example

You can initialize your index variable in the loop heading (int k = 1)

```
for(int k=1; k<=5; k++){

    System.out.println("Count is: " + k);

}
```

Compare to Python:

```
for k in range(1, 6):
     print(“count is”, k)
```

- Will print

```
Count is 1

Count is 2

Count is 3

Count is 4

Count is 5
```

Arrays

Array

Kind of like Python lists but of a **fixed length.**

- Simple data structure that can hold a **fixed number of values** of the same data type.
- The length or storing capacity of an array is established when the array is created and after creation it is fixed. (Different than a Python list! There's no .append()!)
- Each item in an array is called an element, and each element is accessed by its numerical index.
- Numbering begins at 0. The 9th element, for example, would therefore be accessed at index 8.

Declaring and initializing arrays

- Declaring an array requires the use of square brackets next to the type of the values it will hold. For example:
  - `String[] cars;`
  - `int[] numbers;`
- When we declare it, we can also initialize it with certain values separated by comma. Note that we use {} curly brackets to contain the values. For example,
  - `String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};`
  - `int[] numbers = {10, 20, 30, 40};`

https://www.w3schools.com/java/java_arrays.asp

Accessing the elements of an array

- Accessing an array element is done using the square brackets. E.g.,
  - `String[] cars = {"Volvo", "BMW", "Ford", “Mazda"};`
  - `System.out.println(cars[0]);`
  - Will print Volvo

Changing the value of an element

- We will use again square brackets to index the element we want to change. E.g.,
    - `String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};`
    - `cars[0] = "Toyota";`
    - `System.out.println(cars[0]);`
    - Will now print Toyota instead of Volvo.

Array length

- We can determine the storing capacity of an array using the length property. E.g.,
  - `String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};`
  - `System.out.println(cars.length);`
  - Will print 4
- If you request an index that is either negative or larger than length-1, then you will get an ArrayIndexOutOfBoundsException.

Printing values in an array

- We’ve only printed individual values so far.

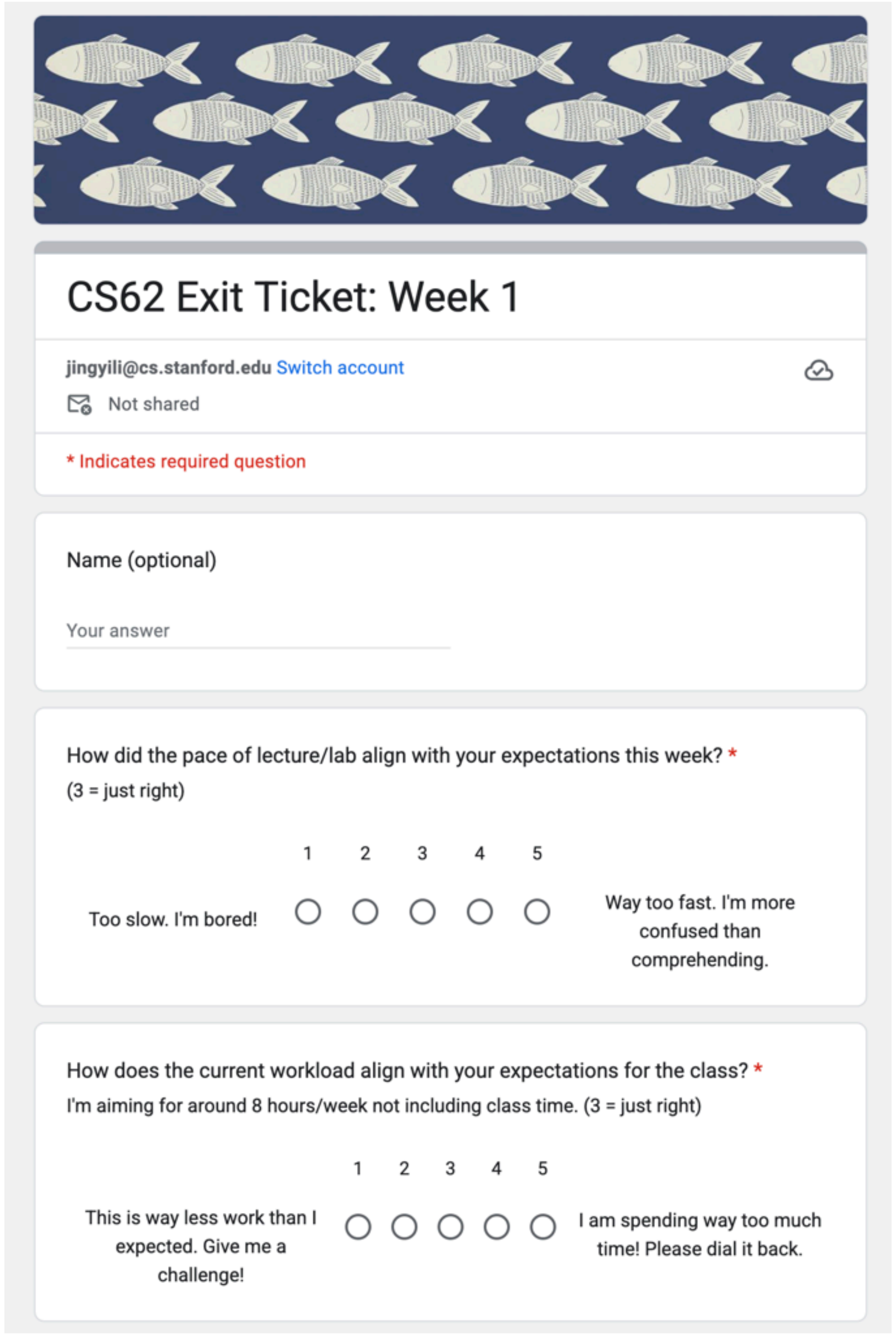
- To print the whole array, we need to use the Arrays.toString(x) method to print the contents of array x in a human readable format.

```
import java.util.Arrays;

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

System.out.println(cars); //prints [I@6d06d69c, array’s location in memory

System.out.println(Arrays.toString(cars)); // prints ["Volvo", "BMW",
"Ford", “Mazda”]
```

https://www.w3schools.com/java/java_arrays.asp

Exit tickets

- Exit tickets help me gauge how you're learning and my pacing
- Names are optional
- Let me know your feelings on an optional review session for the past week's material 10 minutes before lecture on Tuesdays (so 2:35pm)
- We'll do them at the end of every Thurs lecture
- This weeks: https://forms.gle/7EtuhypES1UAFvQB8



CS62 Exit Ticket: Week 1

jingyili@cs.stanford.edu Switch account

Not shared

* Indicates required question

Name (optional)

Your answer

How did the pace of lecture/lab align with your expectations this week? *
(3 = just right)

	1	2	3	4	5	
Too slow. I'm bored!	○	○	○	○	○	Way too fast. I'm more confused than comprehending.

How does the current workload align with your expectations for the class? *
I'm aiming for around 8 hours/week not including class time. (3 = just right)

	1	2	3	4	5	
This is way less work than I expected. Give me a challenge!	○	○	○	○	○	I am spending way too much time! Please dial it back.

Lecture 2 wrap-up

- Exit ticket
- TODO: HW1 due Tuesday night
  - Mentor hours are happening this week! First one tomorrow 4-5pm!
  - I will send out an email by Friday morning with suggested HW groups (3-4)

Resources

- 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
- Optional further practice: Now make a Dog class for your rescue. Make sure to hide sensitive data and write getter/setter methods, as well as fill in the toString(). Or, go back and redo the Employee lab.
- PS: If you want to see how Prof. Papoutsaki taught it, the URL for last semester is https://cs.pomona.edu/classes/cs62/archive/fa2024/schedule.html