L3: Object-Oriented Programming (OOP)

Alex Steiger

CompSci 201: Spring 2024

1/22/24

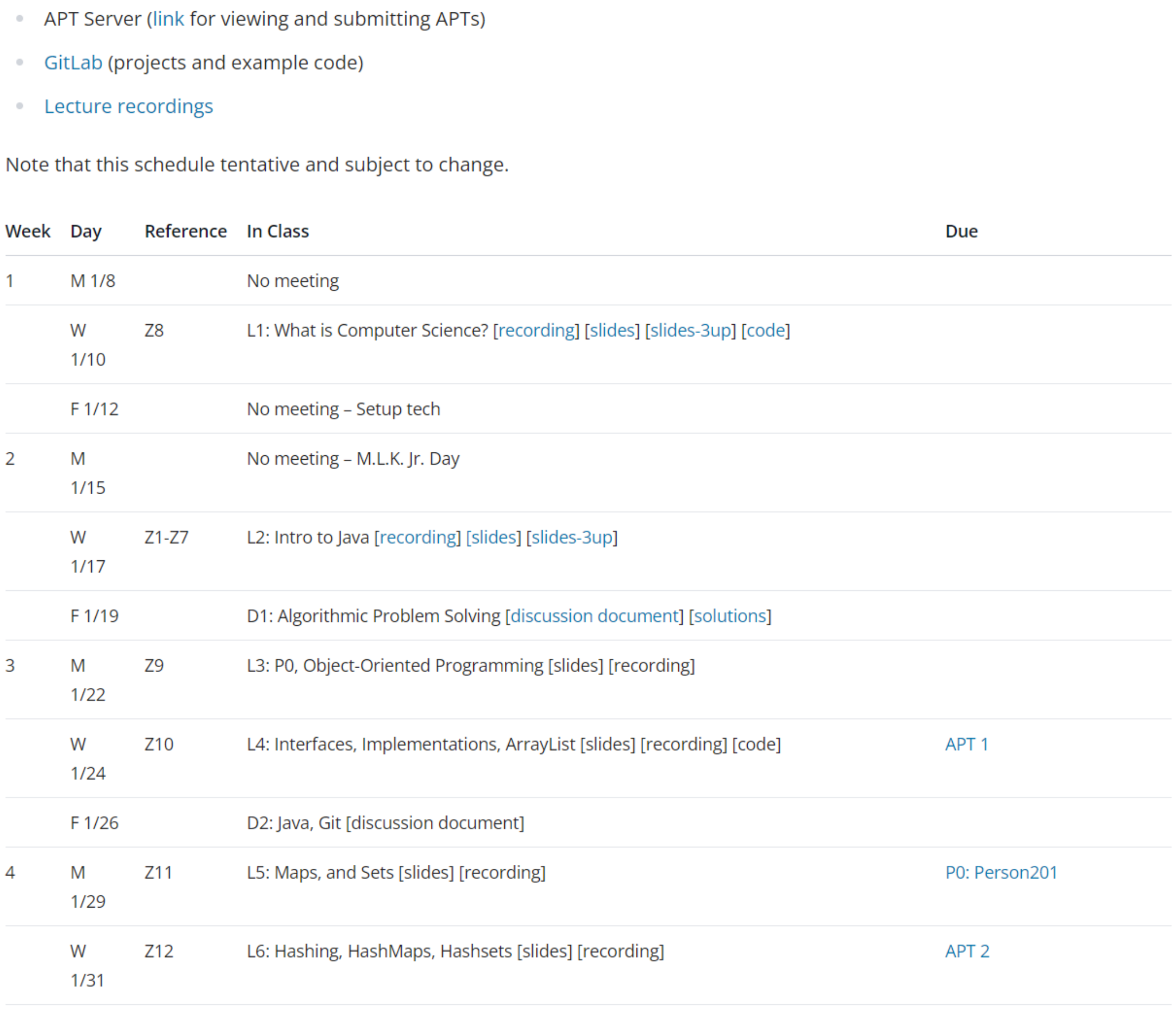
Logistics, Coming up

- This Wednesday, 1/24
  - Interfaces, Implementations, ArrayList data structure
  - First APT set (short programming exercises) due
    - Any time on 1/24, one day late with no penalty
    - 10%/day penalty thereafter, max 1 week

- This Friday, 1/26
  - Discussion 2: APTs, Sets, Strings, Git

- Next Monday 1/29
  - Project 0: Person201 due (warmup project)
    - See link on Schedule

Schedule



- APT Server (link for viewing and submitting APTs)
- GitLab (projects and example code)
- Lecture recordings

Note that this schedule tentative and subject to change.

Week	Day	Reference	In Class	Due
1	M 1/8		No meeting	
	W 1/10	Z8	L1: What is Computer Science? [recording] [slides] [slides-3up] [code]	
	F 1/12		No meeting – Setup tech	
2	M 1/15		No meeting – M.L.K. Jr. Day	
	W 1/17	Z1-Z7	L2: Intro to Java [recording] [slides] [slides-3up]	
	F 1/19		D1: Algorithmic Problem Solving [discussion document] [solutions]	
3	M 1/22	Z9	L3: P0, Object-Oriented Programming [slides] [recording]	
	W 1/24	Z10	L4: Interfaces, Implementations, ArrayList [slides] [recording] [code]	APT 1
	F 1/26		D2: Java, Git [discussion document]	
4	M 1/29	Z11	L5: Maps, and Sets [slides] [recording]	P0: Person201
	W 1/31	Z12	L6: Hashing, HashMaps, Hashsets [slides] [recording]	APT 2

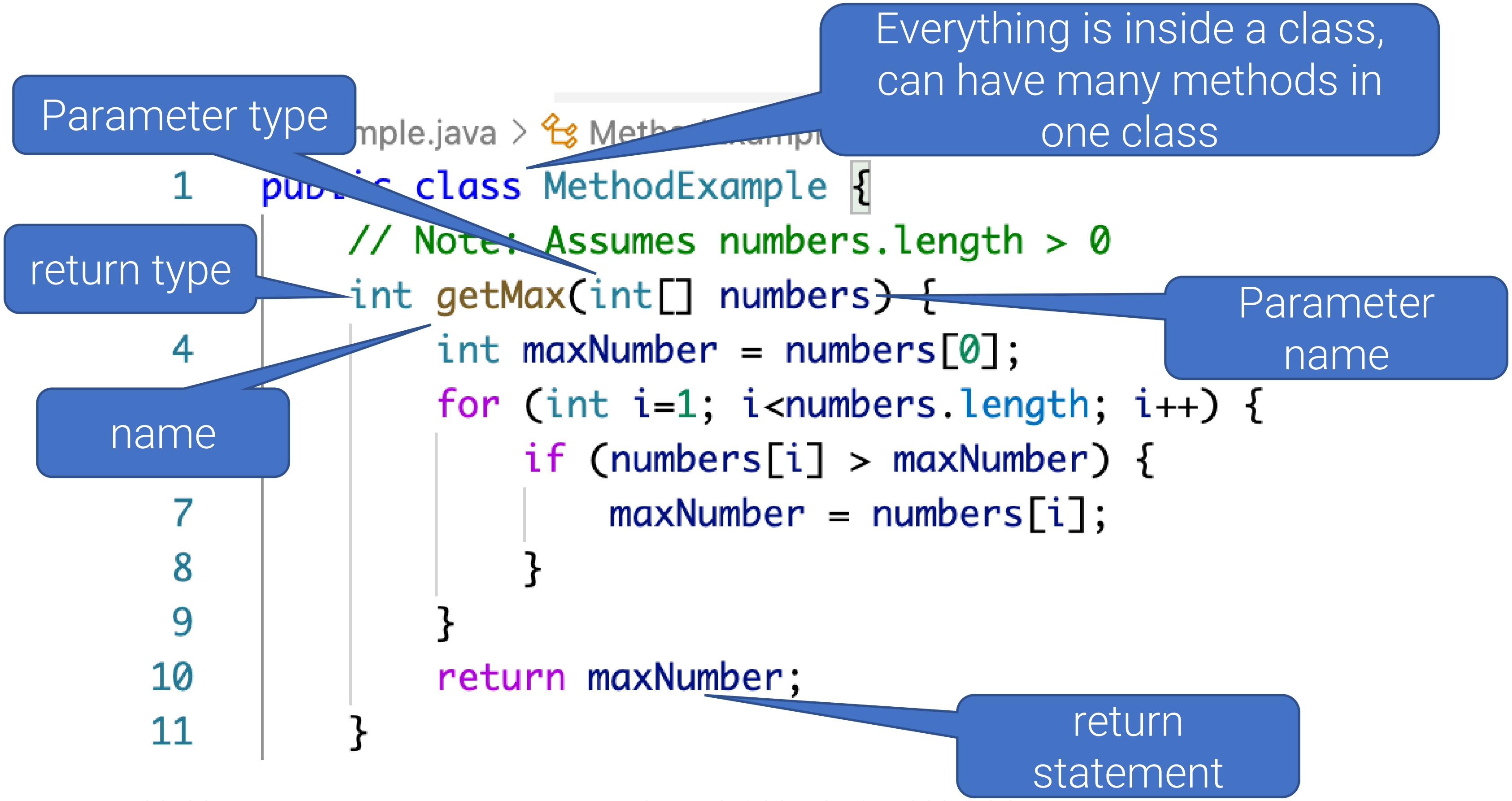
Course Policy Reminders

- **Collaboration reminder**: Can discuss projects and APTs conceptually, but ***code must be your own***.
  - If you can't write the code yourself, you're not going to be ready for whatever you want to do next.
- Getting Help reminder: We want to help!
  - Getting Help page
  - Su-Th *every evening,* Use OhHai to queue
  - Some daytime hours, plus Ed discussion
  - Expect help about your process and how to make progress – not "solutions" or for TAs to debug your code for you.

Java Intro Wrap-up

Anatomy of Java methods

A function defined in a class. No “regular” functions in Java, all methods.



mple.java > Meth

```java
public class MethodExample {
    // Note: Assumes numbers.length > 0
    int getMax(int[] numbers) {
        int maxNumber = numbers[0];
        for (int i=1; i<numbers.length; i++) {
            if (numbers[i] > maxNumber) {
                maxNumber = numbers[i];
            }
        }
        return maxNumber;
    }
```

Everything is inside a class, can have many methods in one class

Parameter type

return type

Parameter name

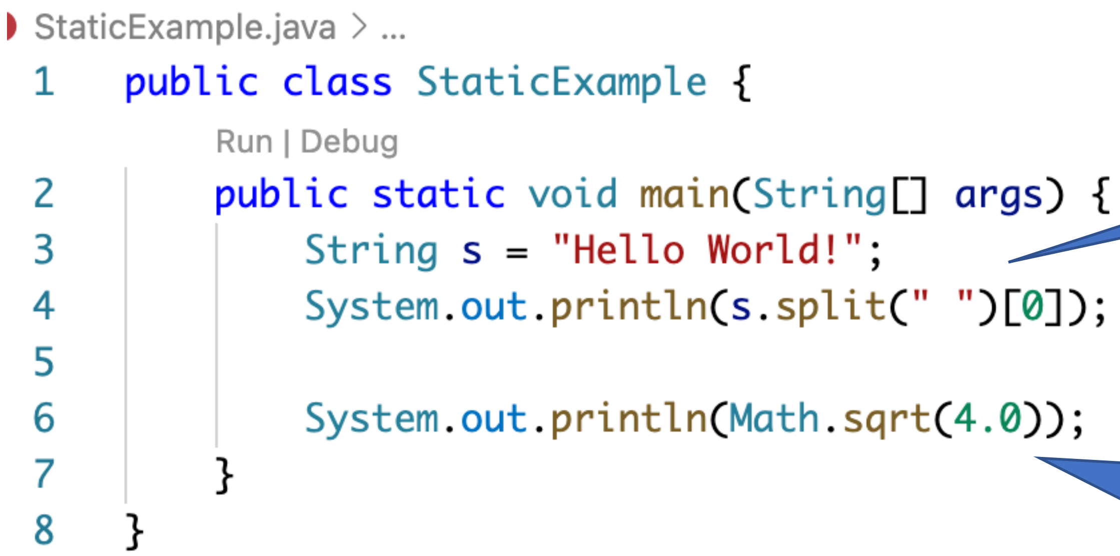
name

return statement

Static vs. Non-static Methods

- Non-static methods are called on a created **object.** Has access to arguments *and* object data.

- Static methods are called on the class, no access to object data. Often called utility “functions.”



```
StaticExample.java > ...
public class StaticExample {
        Run | Debug
        public static void main(String[] args) {
            String s = "Hello World!";
            System.out.println(s.split(" ")[0]);

            System.out.println(Math.sqrt(4.0));
        }
}
```

Note that `split` is called on a String object

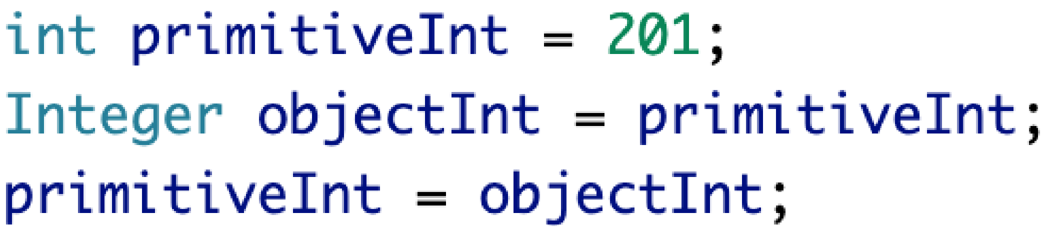
Whereas `sqrt` is called on the `Math` class

Java API Collections, Primitive vs. object types

Why **`ArrayList<Integer>`** … instead of **`ArrayList<int>`**…?

- Java API Collections (ArrayList, HashSet, …) only store *reference types*, not primitive types.

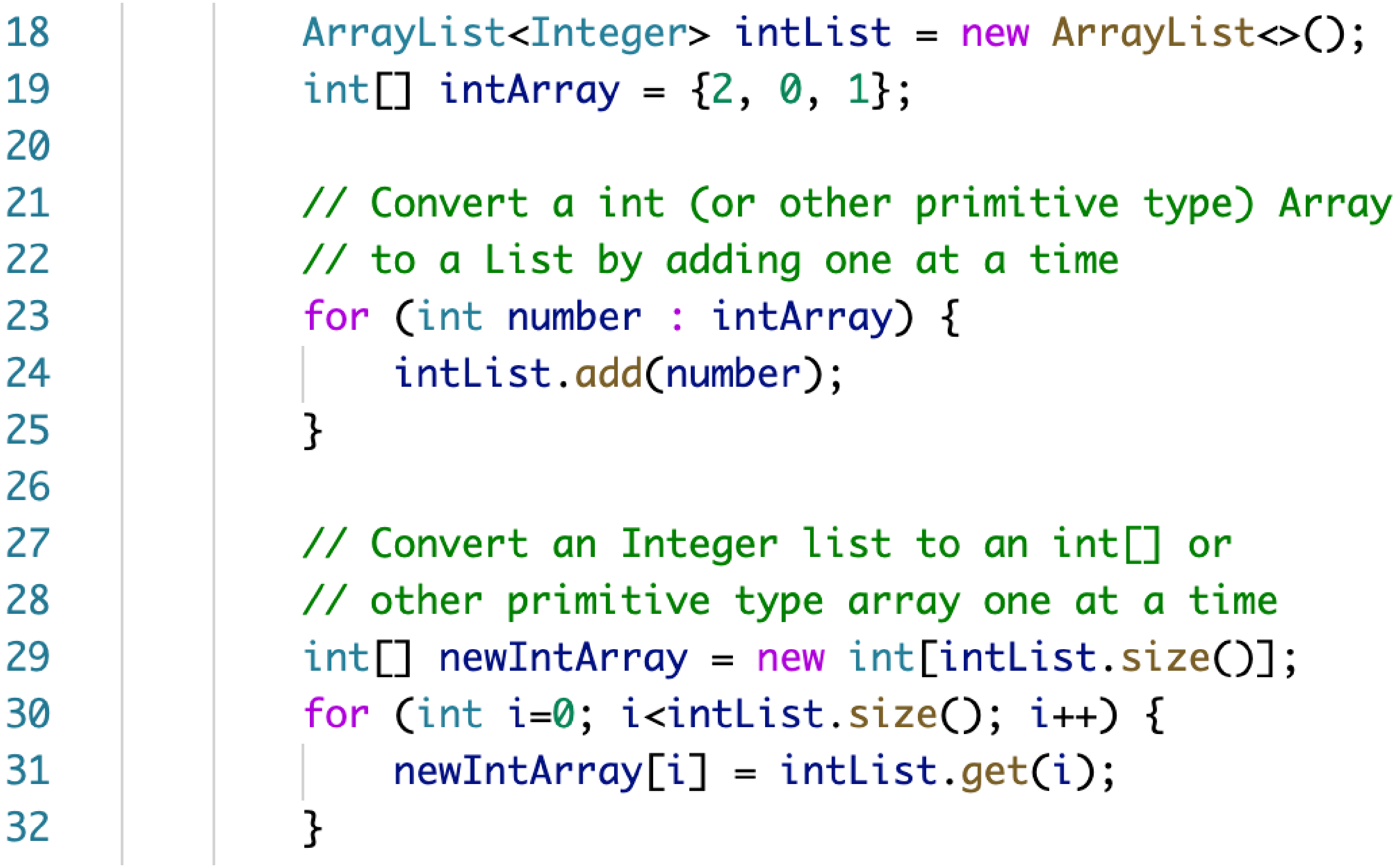
- **`Integer`** is a "wrapper class" for **`int`**, can convert back and forth "automatically."



```
int primitiveInt = 201;
Integer objectInt = primitiveInt;
primitiveInt = objectInt;
```

Same principle for other primitive types, e.g., **`double`** vs. **`Double`**

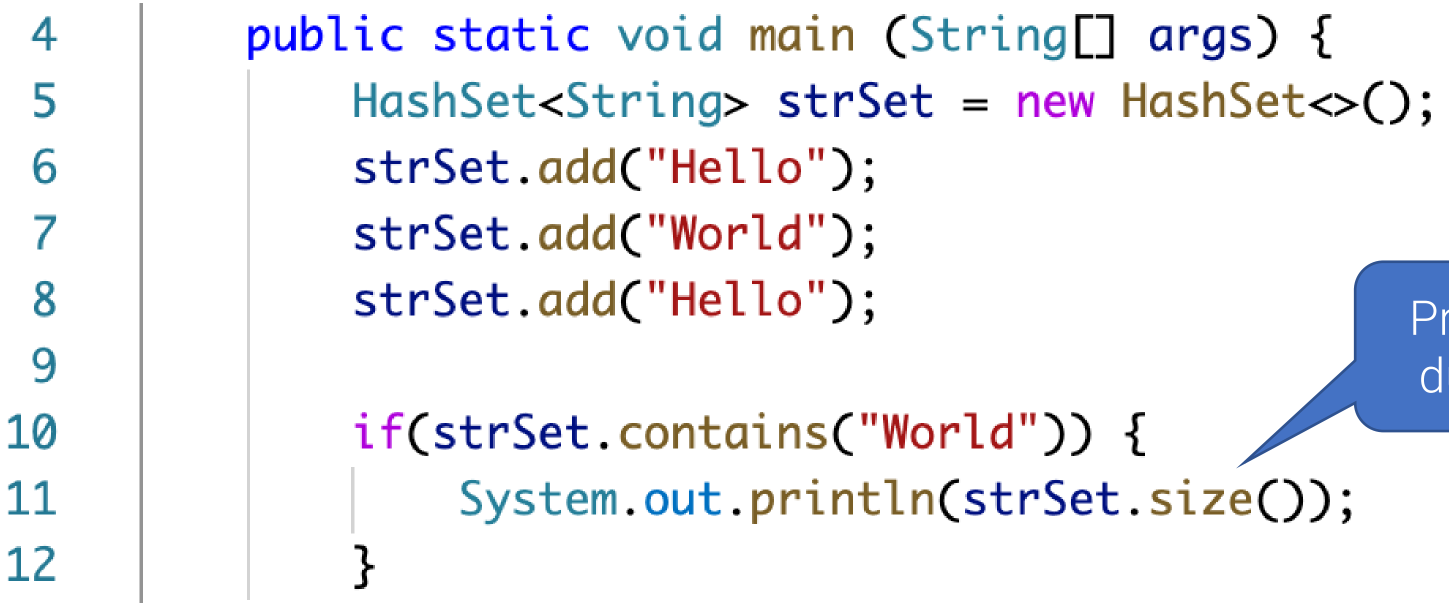
ArrayList <-> Array Conversion, Primitive Types



```java
        ArrayList<Integer> intList = new ArrayList<>();
        int[] intArray = {2, 0, 1};

        // Convert a int (or other primitive type) Array
        // to a List by adding one at a time
        for (int number : intArray) {
            intList.add(number);
        }

        // Convert an Integer list to an int[] or
        // other primitive type array one at a time
        int[] newIntArray = new int[intList.size()];
        for (int i=0; i<intList.size(); i++) {
            newIntArray[i] = intList.get(i);
        }
```

Java API HashSet

- More on HashSet later, but the basics:
  - At top of file, import with: `import java.util.HashSet;`
  - Part of Java API Collections like ArrayList
  - Uses **`add()`**, **`size()`**, **`contains()`** like ArrayList
  - Does *not* store duplicates nor order items(no **`get()`**)

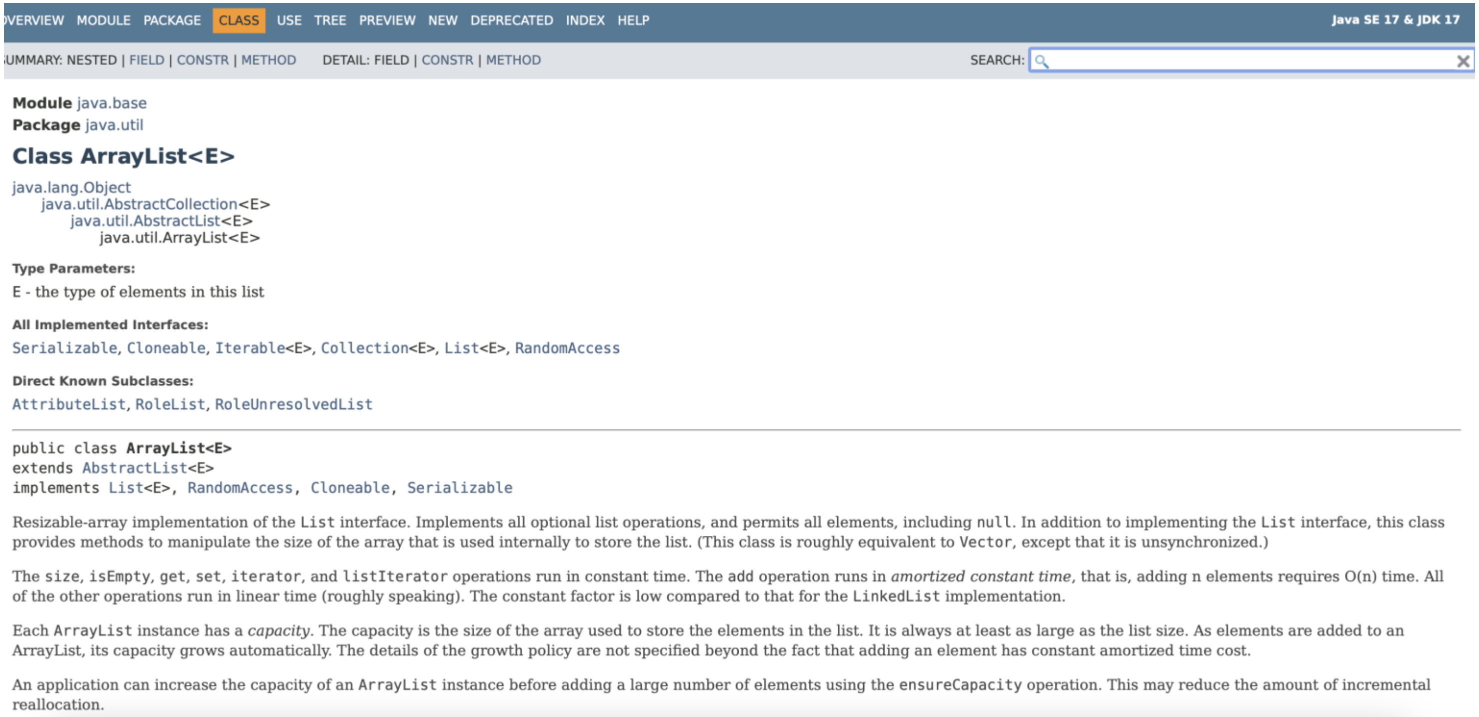


```
    public static void main (String[] args) {
        HashSet<String> strSet = new HashSet<>();
        strSet.add("Hello");
        strSet.add("World");
        strSet.add("Hello");

        if(strSet.contains("World")) {
            System.out.println(strSet.size());
        }
```

Prints 2, no duplicates

API Documentation

Reading documentation is an important skill:
docs.oracle.com/en/java/javase/17/docs/api


OVERVIEW MODULE PACKAGE CLASS USE TREE PREVIEW NEW DEPRECATED INDEX HELP
Java SE 17 & JDK 17
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD
SEARCH:

Module java.base
Package java.util

Class ArrayList<E>

```
java.lang.Object
    java.util.AbstractCollection<E>
        java.util.AbstractList<E>
            java.util.ArrayList<E>
```

Type Parameters:

E - the type of elements in this list

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

Direct Known Subclasses:

AttributeList, RoleList, RoleUnresolvedList

```
public class ArrayList<E>
extends AbstractList<E>
implements List<E>, RandomAccess, Cloneable, Serializable
```

Resizable-array implementation of the List interface. Implements all optional list operations, and permits all elements, including null. In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to Vector, except that it is unsynchronized.)

The size, isEmpty, get, set, iterator, and listIterator operations run in constant time. The add operation runs in *amortized constant time*, that is, adding n elements requires O(n) time. All of the other operations run in linear time (roughly speaking). The constant factor is low compared to that for the LinkedList implementation.

Each ArrayList instance has a *capacity*. The capacity is the size of the array used to store the elements in the list. It is always at least as large as the list size. As elements are added to an ArrayList, its capacity grows automatically. The details of the growth policy are not specified beyond the fact that adding an element has constant amortized time cost.

An application can increase the capacity of an ArrayList instance before adding a large number of elements using the ensureCapacity operation. This may reduce the amount of incremental reallocation.

WOTO

Go to

Not graded for correctness, just participation.

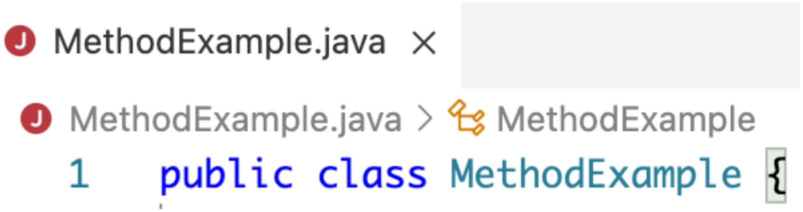
Try to answer *without* looking back at slides and notes.

But do talk to your neighbors!

Java style and comments

Class names:

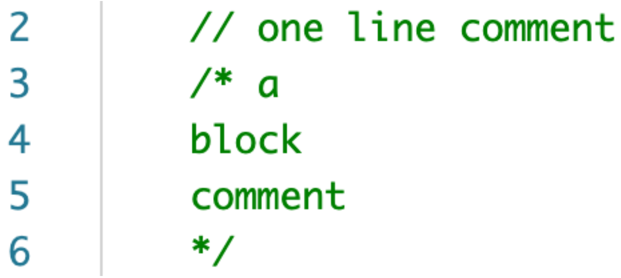
- Capitalized & CamelCase
- MUST match name of .java file!



```java
public class MethodExample {
```

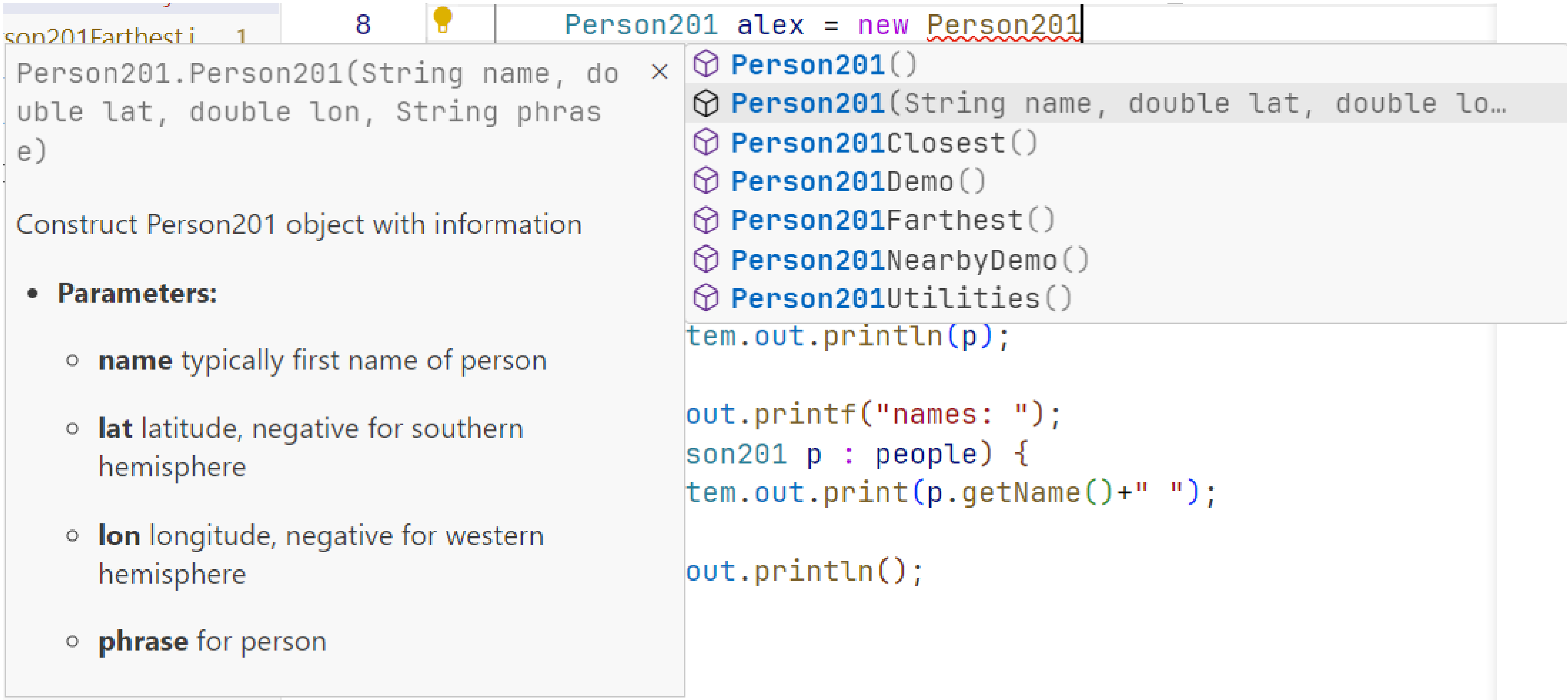
Comments:

- // for one line
- /* … */ for multiple lines



```java
        // one line comment
        /* a
        block
        comment
        */
```

Javadoc: Advanced comments



son201Farthest.j 1

```
    Person201 alex = new Person201
```

```
Person201.Person201(String name, double lat, double lon, String phrase)
```

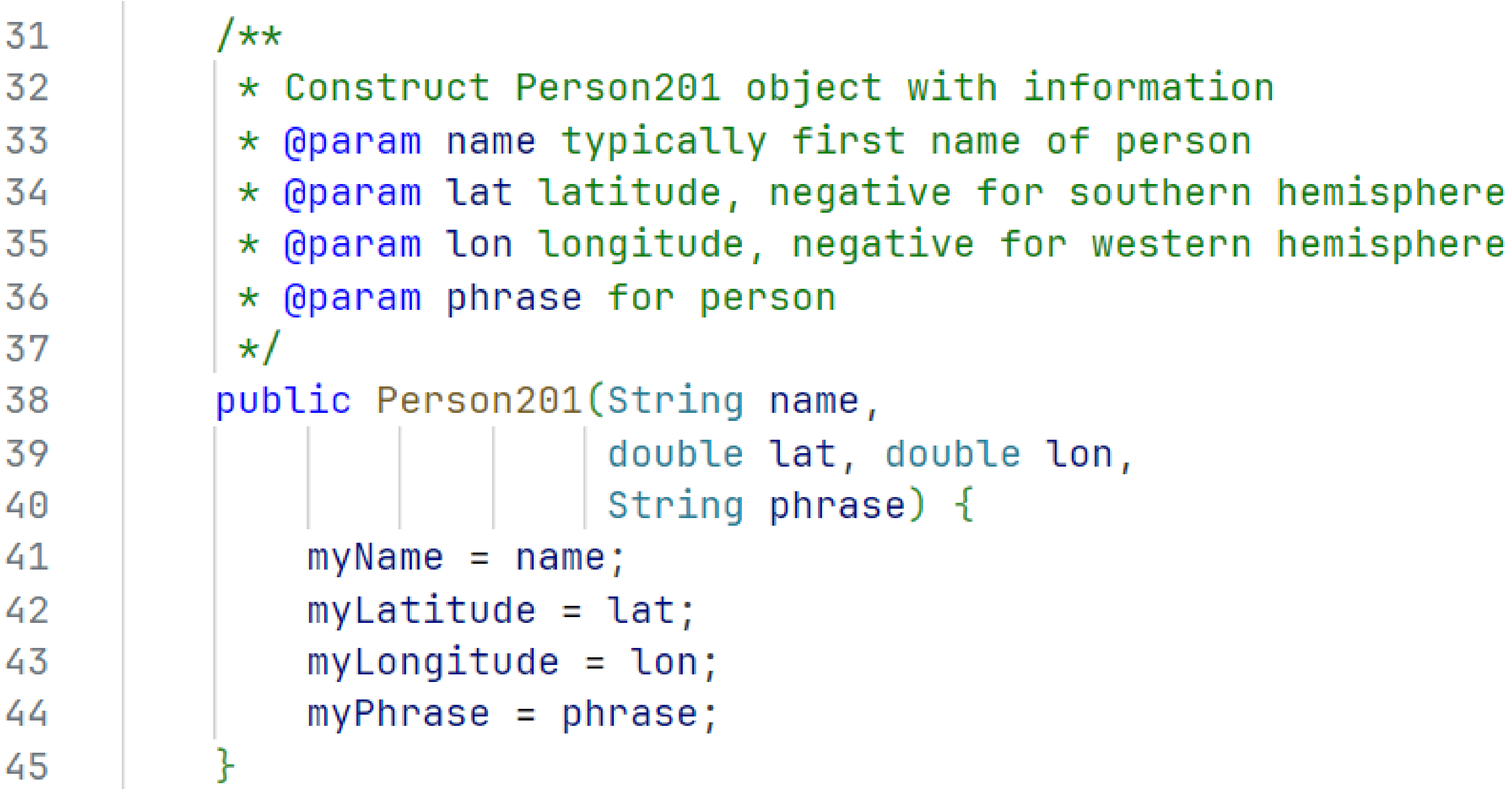
Construct Person201 object with information

- **Parameters:**
  - **name** typically first name of person
  - **lat** latitude, negative for southern hemisphere
  - **lon** longitude, negative for western hemisphere
  - **phrase** for person

```
Person201()
Person201(String name, double lat, double lo…
Person201Closest()
Person201Demo()
Person201Farthest()
Person201NearbyDemo()
Person201Utilities()
```

```
tem.out.println(p);

out.printf("names: ");
son201 p : people) {
tem.out.print(p.getName()+" ");

out.println();
```

Writing Javadoc



```java
/**
 * Construct Person201 object with information
 * @param name typically first name of person
 * @param lat latitude, negative for southern hemisphere
 * @param lon longitude, negative for western hemisphere
 * @param phrase for person
 */
public Person201(String name,
                 double lat, double lon,
                 String phrase) {
    myName = name;
    myLatitude = lat;
    myLongitude = lon;
    myPhrase = phrase;
}
```

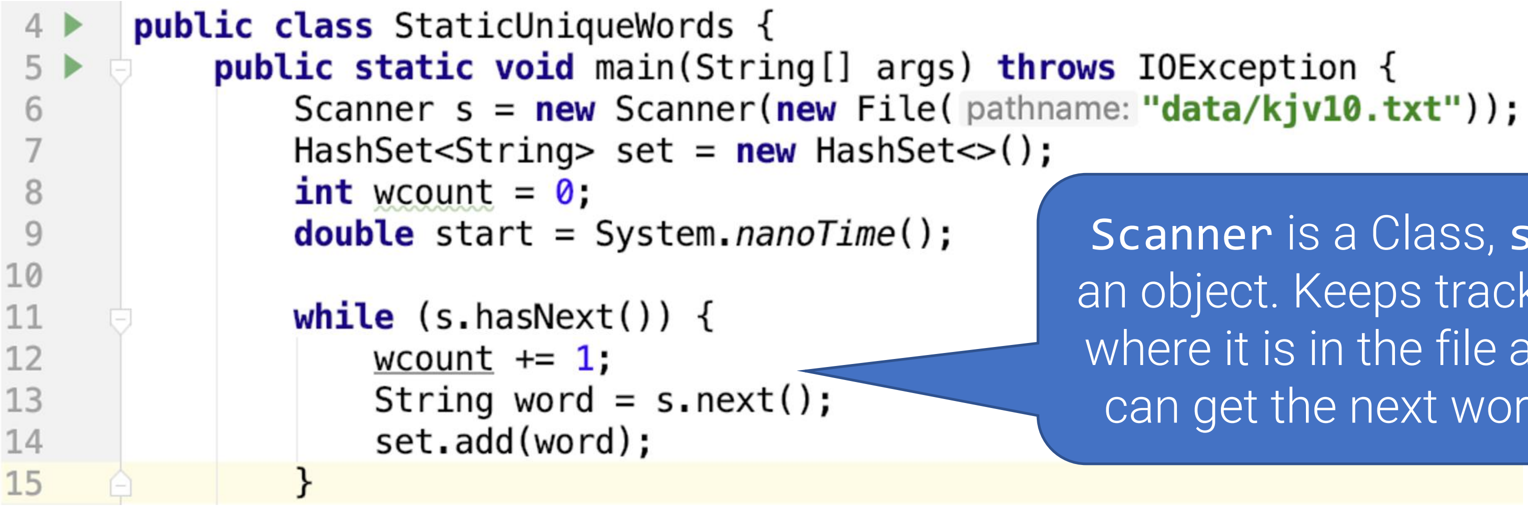
Common annotations for methods include:
`@param, @returns, @throws`

Object-Oriented Programming (OOP)

Java is object-oriented

- A language is **object-oriented** if programs in that language are organized by the specification and use of objects.
- “An **object** consists of some internal data items plus operations that can be performed on that data.”−zyBook

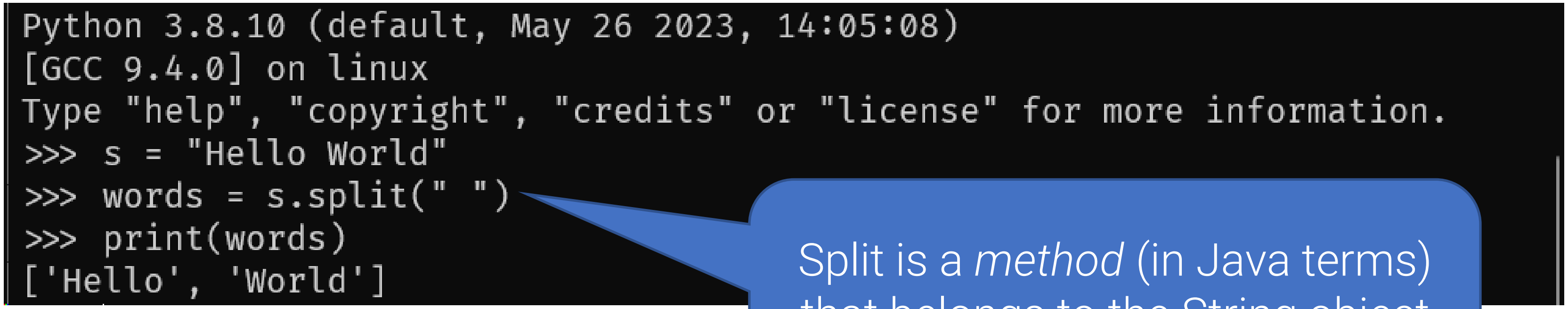
We call these *methods*



```java
public class StaticUniqueWords {
    public static void main(String[] args) throws IOException {
        Scanner s = new Scanner(new File( pathname: "data/kjv10.txt"));
        HashSet<String> set = new HashSet<>();
        int wcount = 0;
        double start = System.nanoTime();

        while (s.hasNext()) {
            wcount += 1;
            String word = s.next();
            set.add(word);
        }
```

`Scanner` is a Class, `s` is an object. Keeps track of where it is in the file and can get the next word.

Aside: Python uses objects too



```
Python 3.8.10 (default, May 26 2023, 14:05:08)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> s = "Hello World"
>>> words = s.split(" ")
>>> print(words)
['Hello', 'World']
```

Split is a *method* (in Java terms) that belongs to the String object **s** on which it is called

Same syntax in Python and Java for method calls:
`<object>.<method>(<method_arguments>)`

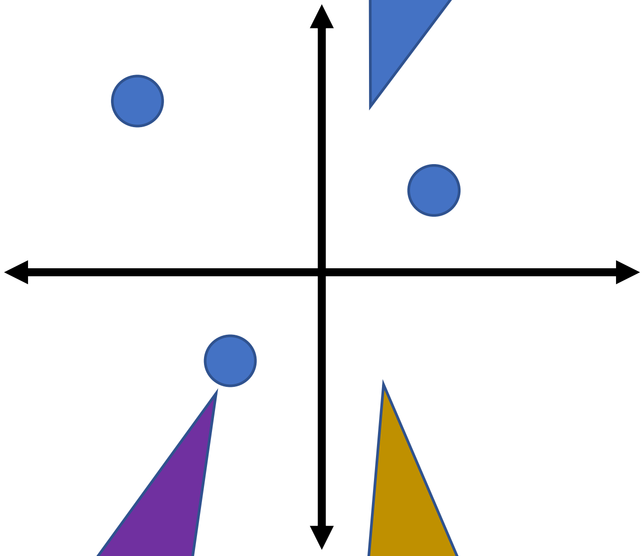
Object Concept

Consider points in two-dimensions.

Class is a blueprint for these objects

Data (instance variables)
- x-coordinate (x)
- y-coordinate (y)

Operations (methods)
- Create a point
- Print a point
- Change coordinates
- Get distance to another point

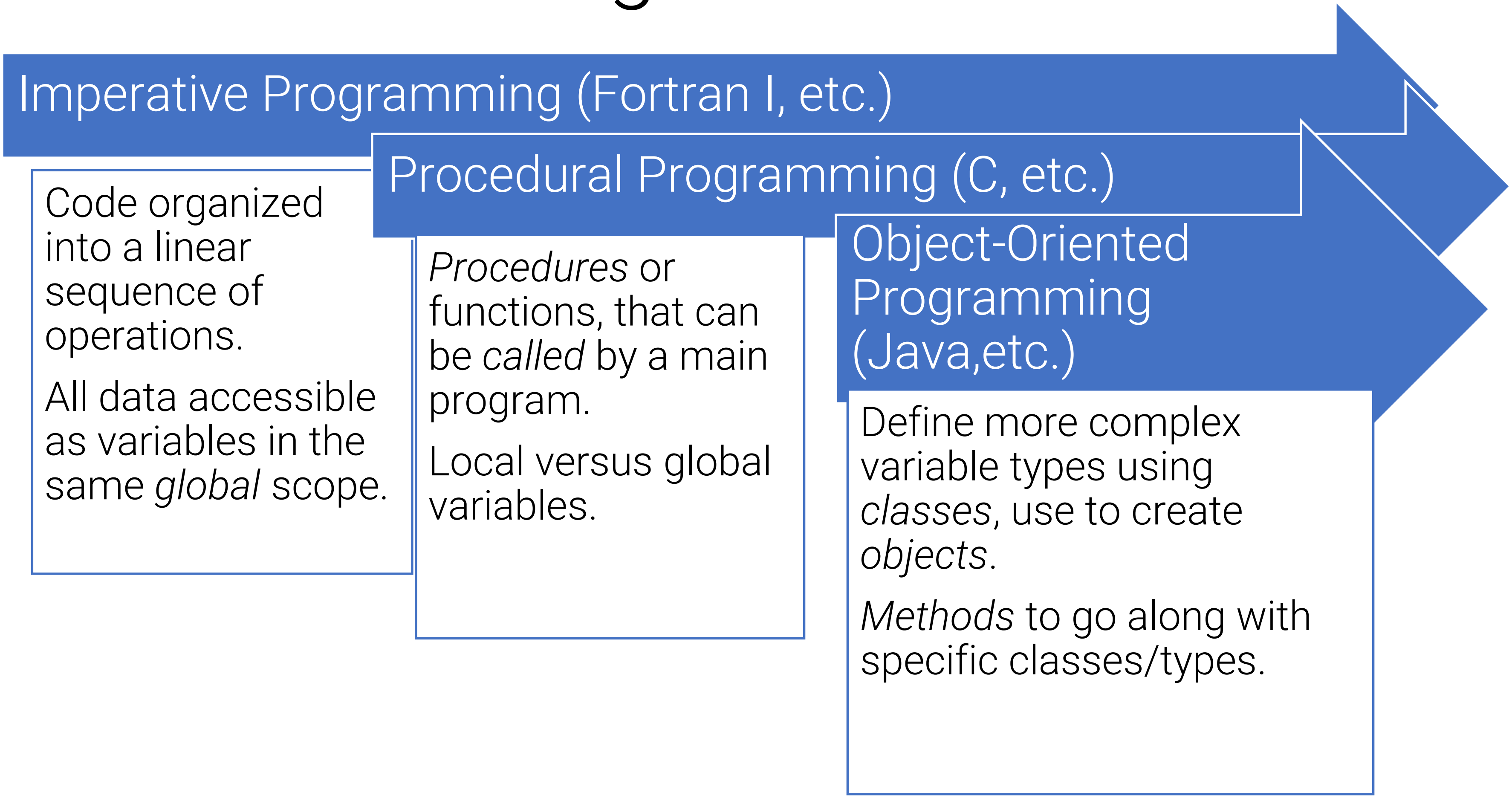


All different objects, but each of the same class

Each point object has its own x and y value.

Methods should be able to operate on a particular point

Language History: A story of increasing abstraction and organization



Imperative Programming (Fortran I, etc.)

Code organized into a linear sequence of operations.

All data accessible as variables in the same *global* scope.

Procedural Programming (C, etc.)

Procedures or functions, that can be *called* by a main program.

Local versus global variables.

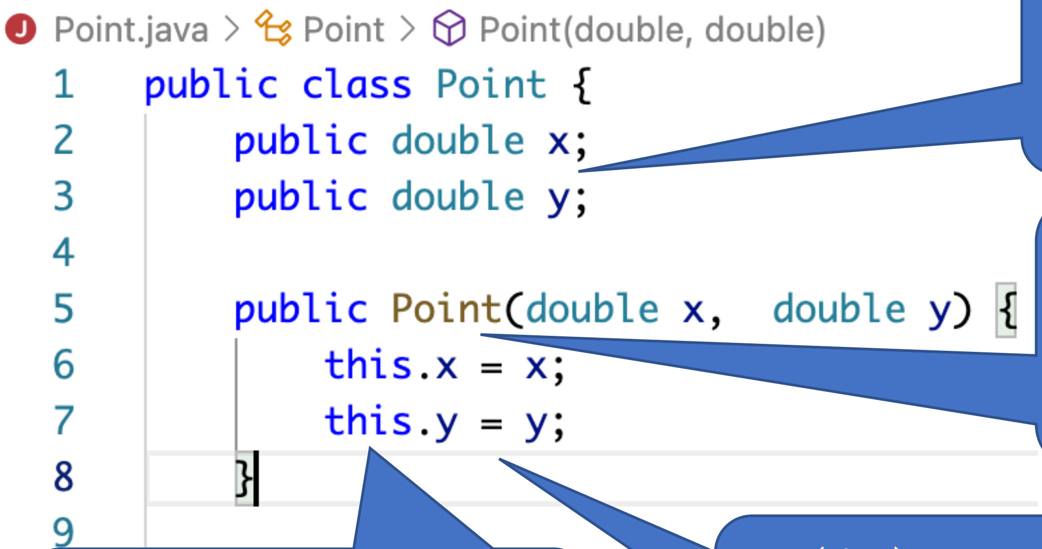
Object-Oriented Programming (Java,etc.)

Define more complex variable types using *classes*, use to create *objects*.

Methods to go along with specific classes/types.

Classes and objects

Class specifies the data and operations for a type of object. They are a template or a blueprint for objects. Alternately, objects are *instances* of a class.



Point.java > Point > Point(double, double)

```
public class Point {
    public double x;
    public double y;

    public Point(double x,  double y) {
        this.x = x;
        this.y = y;
    }
```

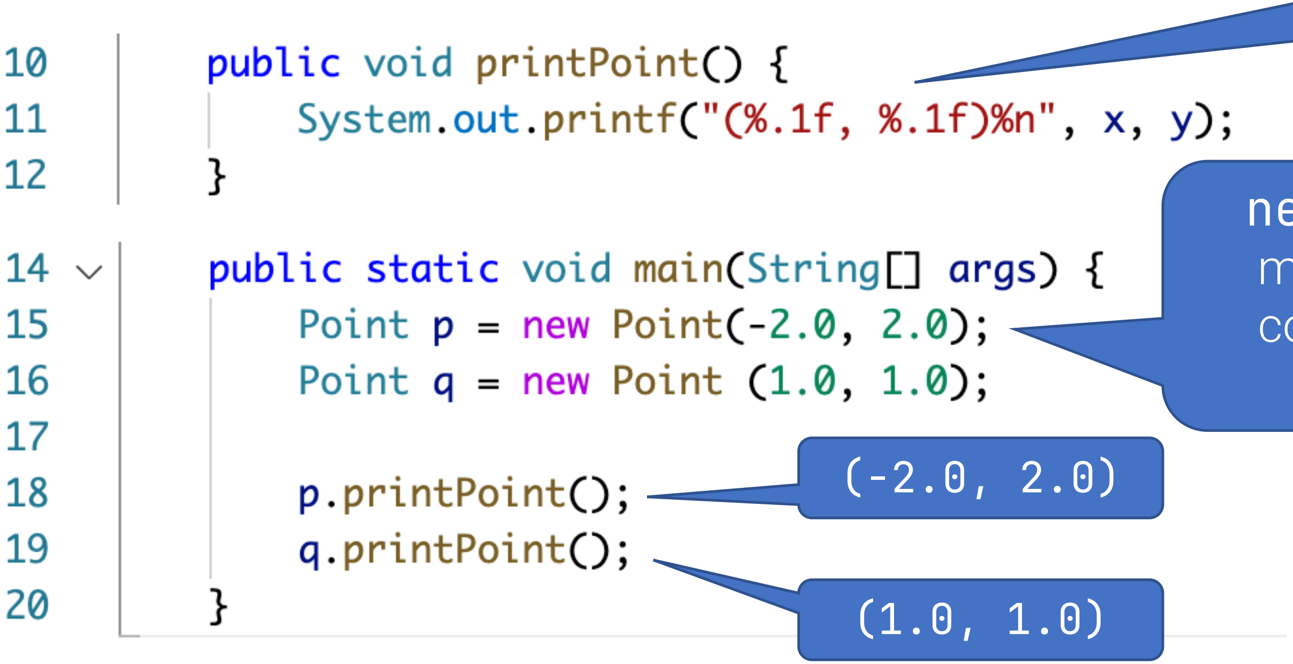
Instance variables. Each `Point` object has its own **x** and **y** value.

A **constructor** method specifies how to create a new Point object. Same name as class.

`this` keyword refers to object on which method is called.

. (dot) operator accesses instance variable or method of this object

Creating objects, calling methods



```java
    public void printPoint() {
        System.out.printf("(%.1f, %.1f)%n", x, y);
    }

    public static void main(String[] args) {
        Point p = new Point(-2.0, 2.0);
        Point q = new Point (1.0, 1.0);

        p.printPoint();
        q.printPoint();
    }
```

Method defined inside the point class

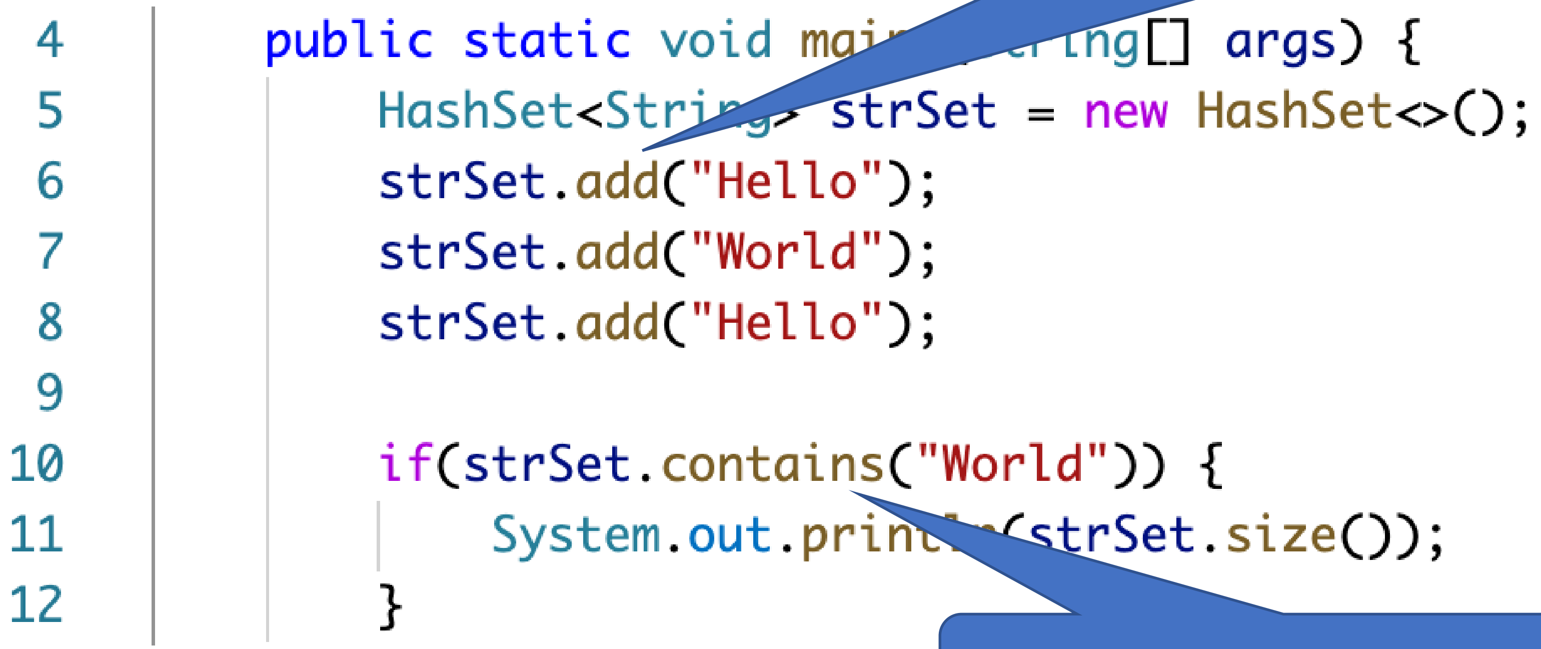
`new Point` allocates memory and calls the constructor to set the instance variables

```
(-2.0, 2.0)
```

```
(1.0, 1.0)
```

Note how the `printPoint()` method "knows" the correct value for `x` and `y` – they are stored with the objects on which we call the method as *instance variables*.

Two reasons to call a method



```
    public static void main(String[] args) {
        HashSet<String> strSet = new HashSet<>();
        strSet.add("Hello");
        strSet.add("World");
        strSet.add("Hello");

        if(strSet.contains("World")) {
            System.out.println(strSet.size());
        }
```

For the *side effect*, what it did to the object

For the *return value*

WOTO

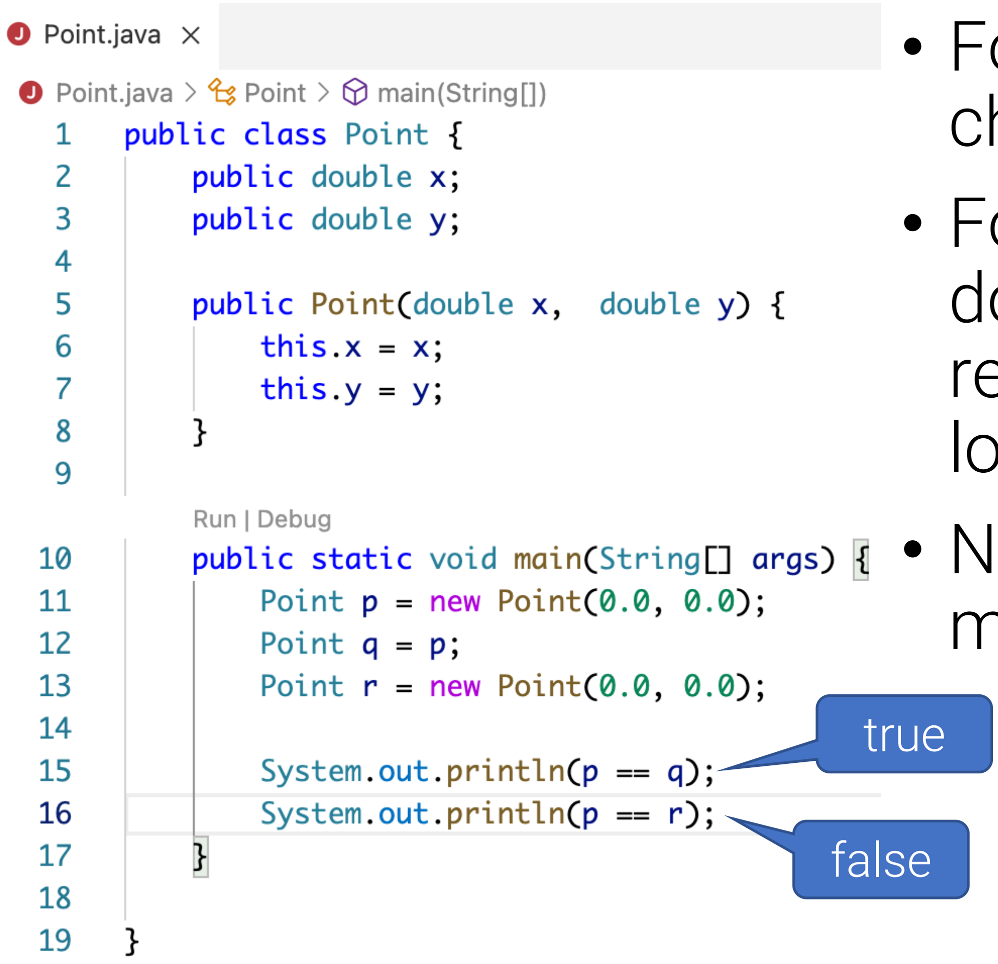
Go to

Not graded for correctness, just participation.

Try to answer *without* looking back at slides and notes.

But do talk to your neighbors!

== or `.equals()`?



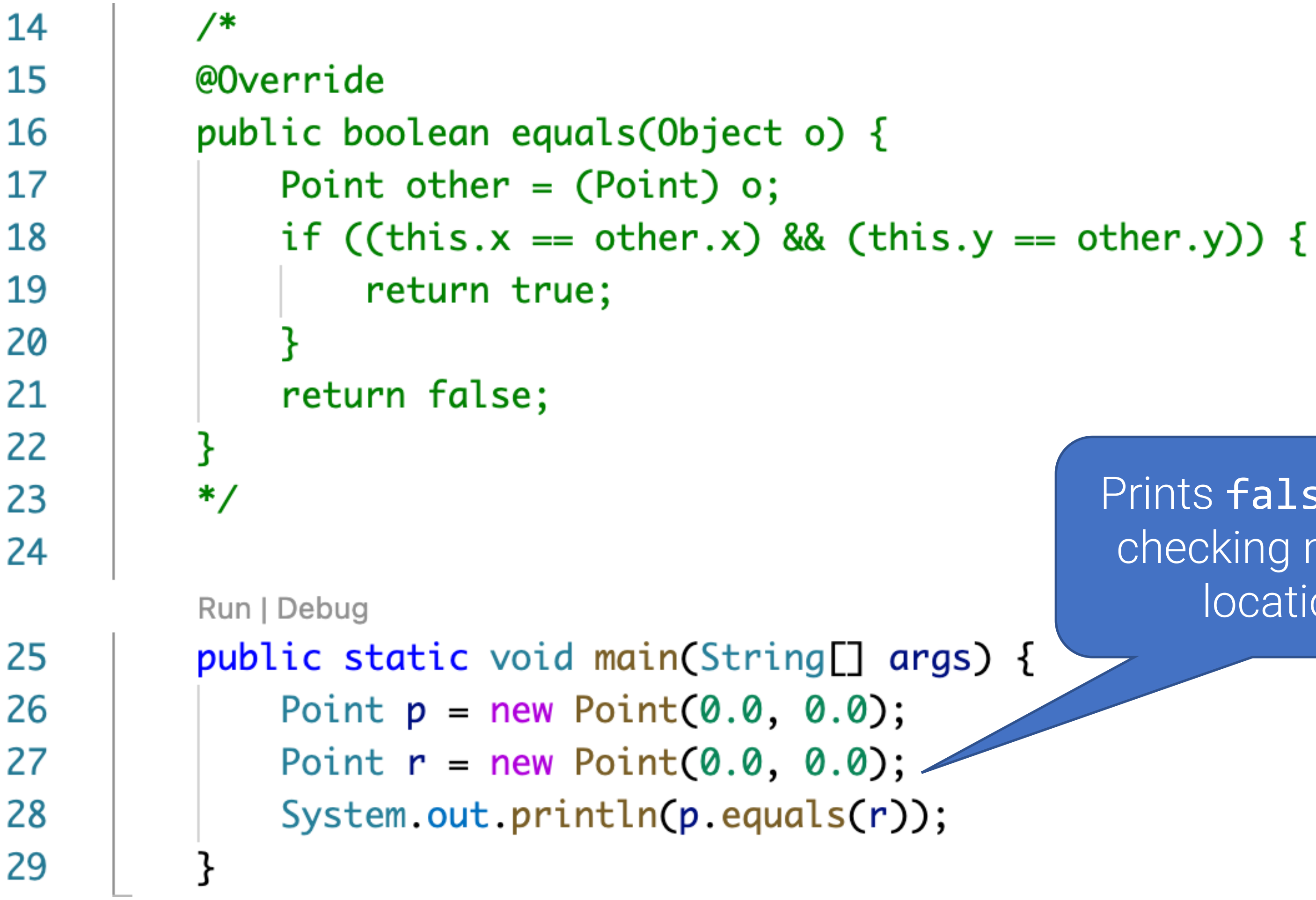
```java
public class Point {
    public double x;
    public double y;

    public Point(double x,  double y) {
        this.x = x;
        this.y = y;
    }

    public static void main(String[] args) {
        Point p = new Point(0.0, 0.0);
        Point q = p;
        Point r = new Point(0.0, 0.0);

        System.out.println(p == q);
        System.out.println(p == r);
    }

}
```

true

false

- For primitive types: `==` checks for equal values.
- For objects, `==` generally does ***not***; compares references (memory locations)
- Need to use `.equals()` method for objects.
  - Correct way to compare `String` objects.
  - Must be implemented for the given Class!

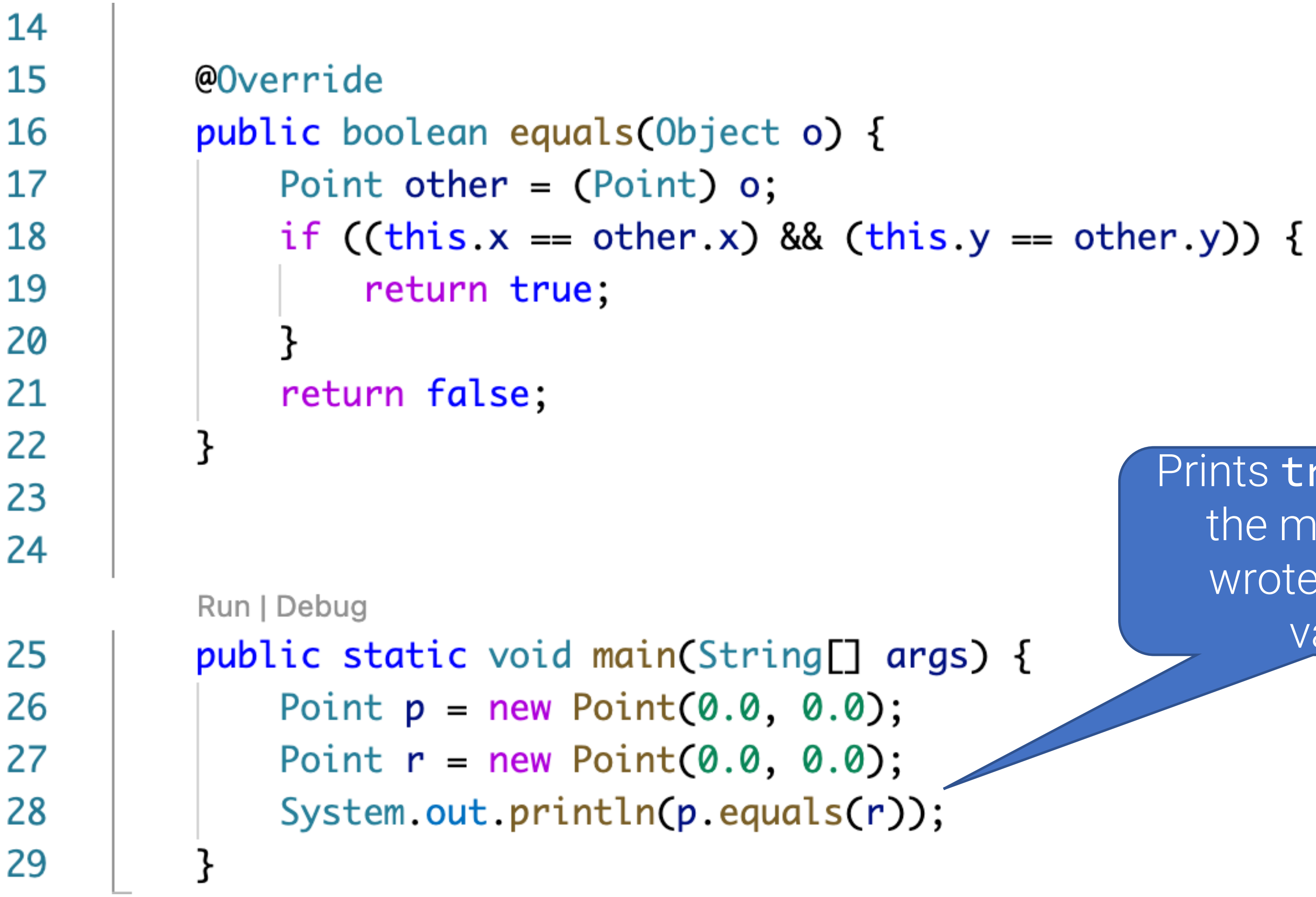
Default Object `.equals`



```
/*
@Override
public boolean equals(Object o) {
    Point other = (Point) o;
    if ((this.x == other.x) && (this.y == other.y)) {
        return true;
    }
    return false;
}
*/

public static void main(String[] args) {
    Point p = new Point(0.0, 0.0);
    Point r = new Point(0.0, 0.0);
    System.out.println(p.equals(r));
}
```

Prints **`false`**, is just checking memory locations

Overriding default Object `.equals`



```
@Override
public boolean equals(Object o) {
    Point other = (Point) o;
    if ((this.x == other.x) && (this.y == other.y)) {
        return true;
    }
    return false;
}
```

```
public static void main(String[] args) {
    Point p = new Point(0.0, 0.0);
    Point r = new Point(0.0, 0.0);
    System.out.println(p.equals(r));
}
```

Prints `true`, is using the method we wrote to check values

Object vs. object, Inheritance?

- Object: ancestor of all classes
  - Default behavior that's not too useful, …
  - @Override for `.equals`

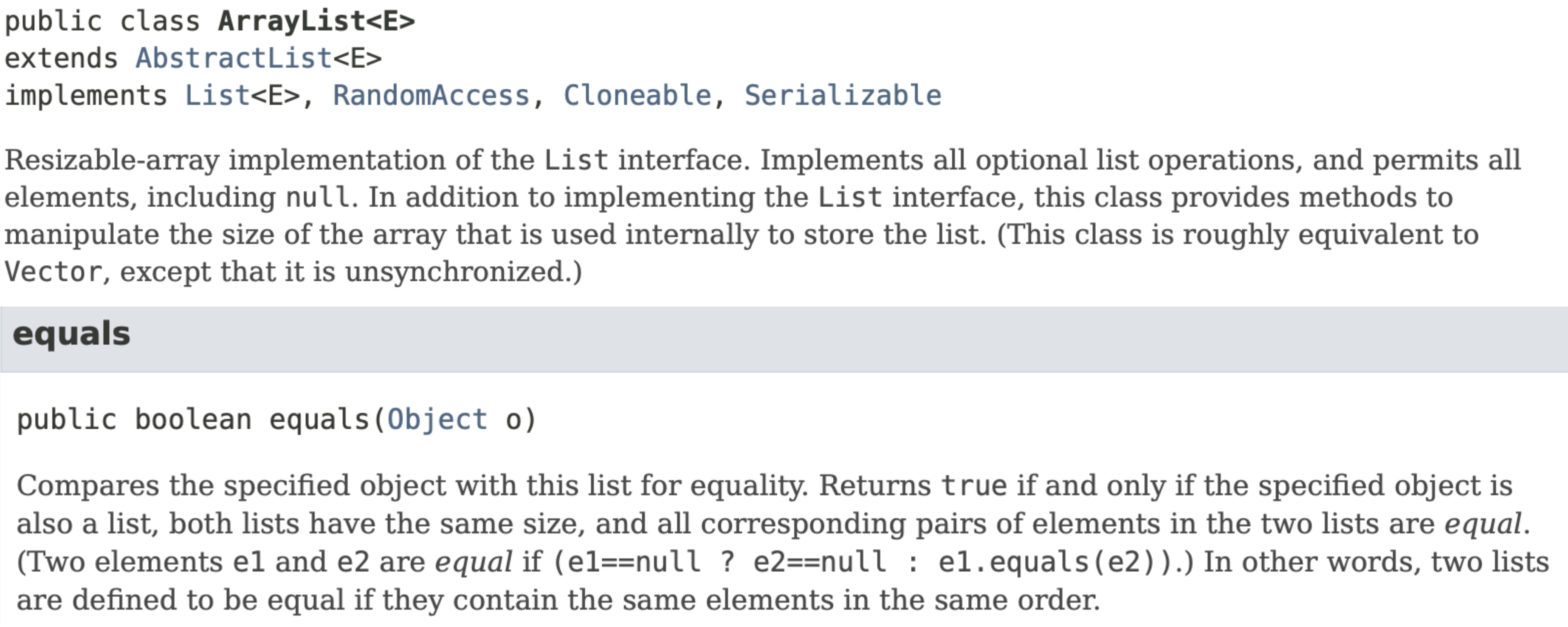
- object – synonym for instance of a class
  - What you get when you call `new` (technically, a *reference* to it)

- Inheritance is a major topic in object-oriented programming to which we will return!

How do I know what `.equals` does for Java API classes?

Read at the Java API documentation!!!

docs.oracle.com/en/java/javase/17/docs/



```
public class ArrayList<E>
extends AbstractList<E>
implements List<E>, RandomAccess, Cloneable, Serializable
```

Resizable-array implementation of the `List` interface. Implements all optional list operations, and permits all elements, including `null`. In addition to implementing the `List` interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to `Vector`, except that it is unsynchronized.)

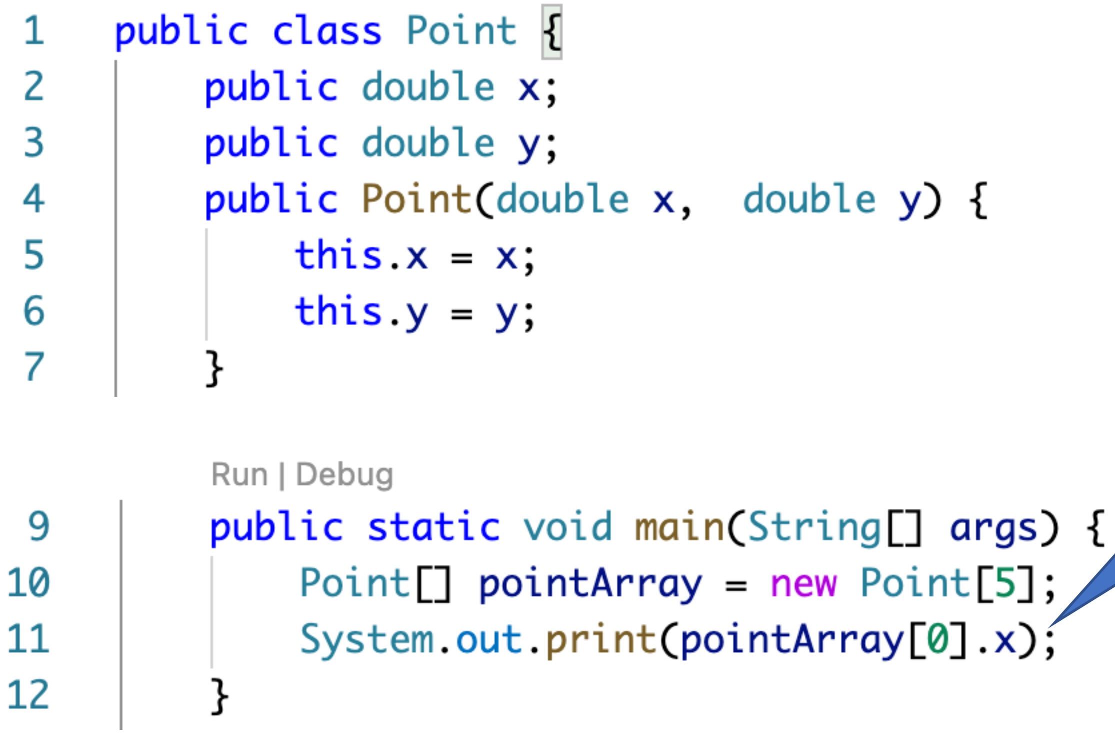
equals

```
public boolean equals(Object o)
```

Compares the specified object with this list for equality. Returns `true` if and only if the specified object is also a list, both lists have the same size, and all corresponding pairs of elements in the two lists are *equal*. (Two elements `e1` and `e2` are *equal* if `(e1==null ? e2==null : e1.equals(e2))`.) In other words, two lists are defined to be equal if they contain the same elements in the same order.

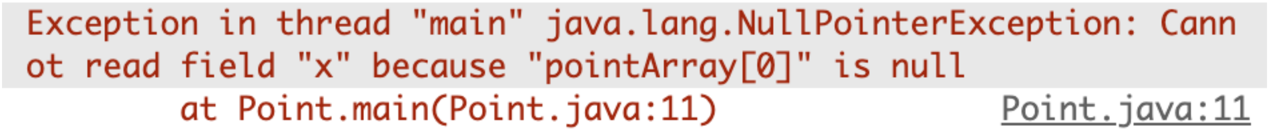
Q: When do I need **new**?

A: Every time I want to create an object, not automatic!



```java
1   public class Point {
2       public double x;
3       public double y;
4       public Point(double x,  double y) {
5           this.x = x;
6           this.y = y;
7       }

        Run | Debug
9       public static void main(String[] args) {
10          Point[] pointArray = new Point[5];
11          System.out.print(pointArray[0].x);
12      }
```

We created the array, but it is filled with **null**'s; did not call new for the individual Point objects.



```
Exception in thread "main" java.lang.NullPointerException: Cann
ot read field "x" because "pointArray[0]" is null
        at Point.main(Point.java:11)                    Point.java:11
```

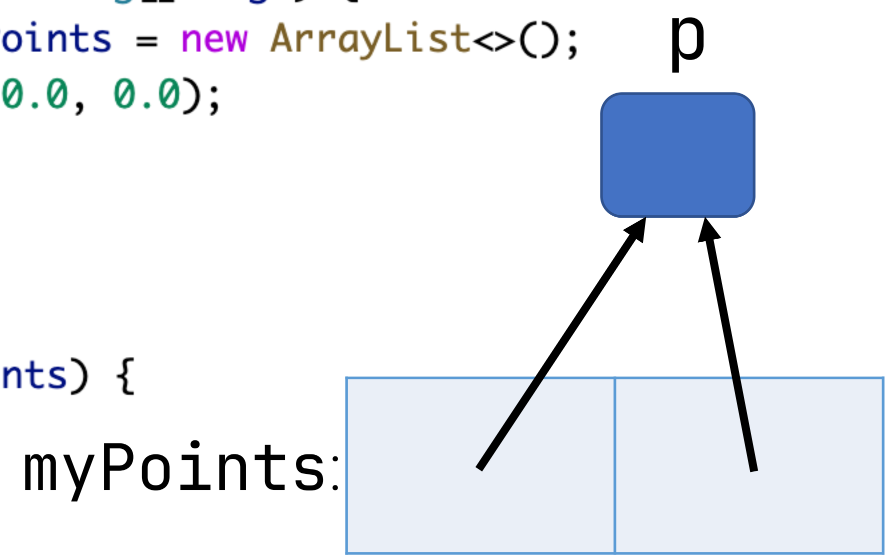
When do I need new again? For every object you want to create!

An even stranger error… creating one object but multiple references to it.

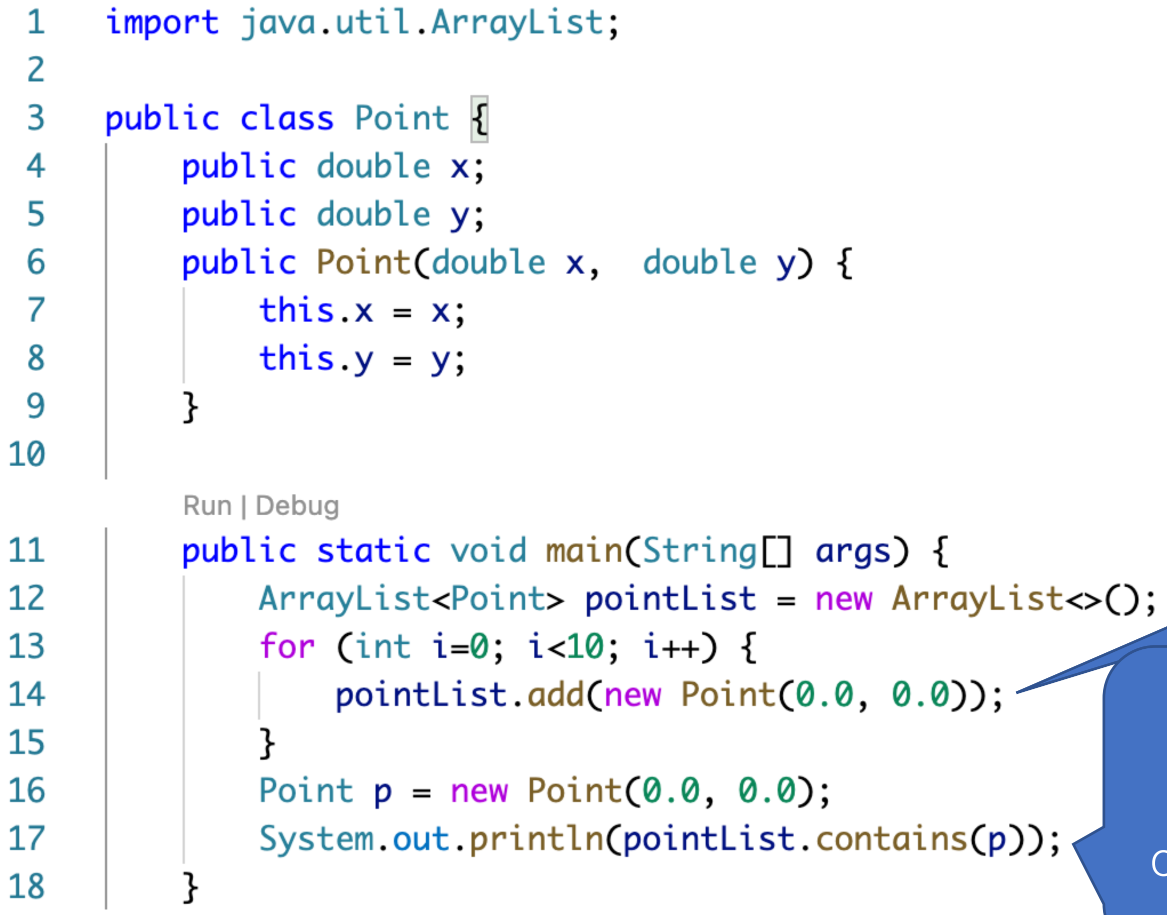
```java
    public static void main(String[] args) {
        ArrayList<Point> myPoints = new ArrayList<>();
        Point p = new Point(0.0, 0.0);
        myPoints.add(p);
        p.x = 2.0;
        myPoints.add(p);

        for (Point q : myPoints) {
            q.printPoint();
        }
```

Prints

```
(2.0, 0.0)
(2.0, 0.0)
```



Creating a List of points; **`contains`** uses **`equals`**



```java
import java.util.ArrayList;

public class Point {
    public double x;
    public double y;
    public Point(double x,  double y) {
        this.x = x;
        this.y = y;
    }

    public static void main(String[] args) {
        ArrayList<Point> pointList = new ArrayList<>();
        for (int i=0; i<10; i++) {
            pointList.add(new Point(0.0, 0.0));
        }
        Point p = new Point(0.0, 0.0);
        System.out.println(pointList.contains(p));
    }
```

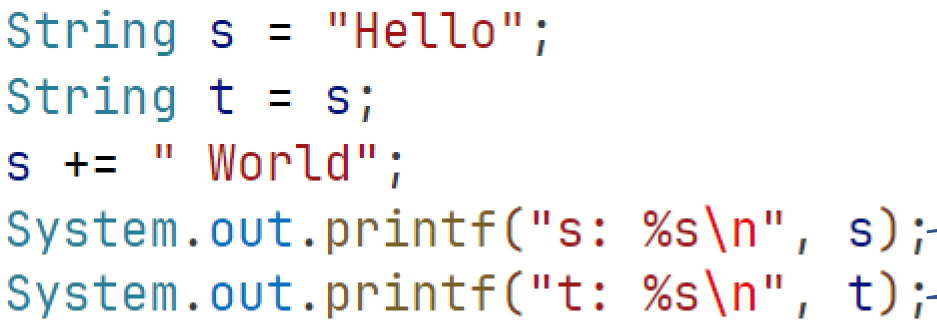
Good, we called **`new`** for every **`Point`** object we want to create.

Prints **`false`**. **`ArrayList .contains`** loops over list checking **`.equals()`**, but only default implementation here!

(Im)mutability

- An object is **immutable** if you cannot change it after creation. Methods that change objects are called **mutators.**

- Java Strings are immutable, even though you can “append” to them. Creates a new String and assigns it every time!



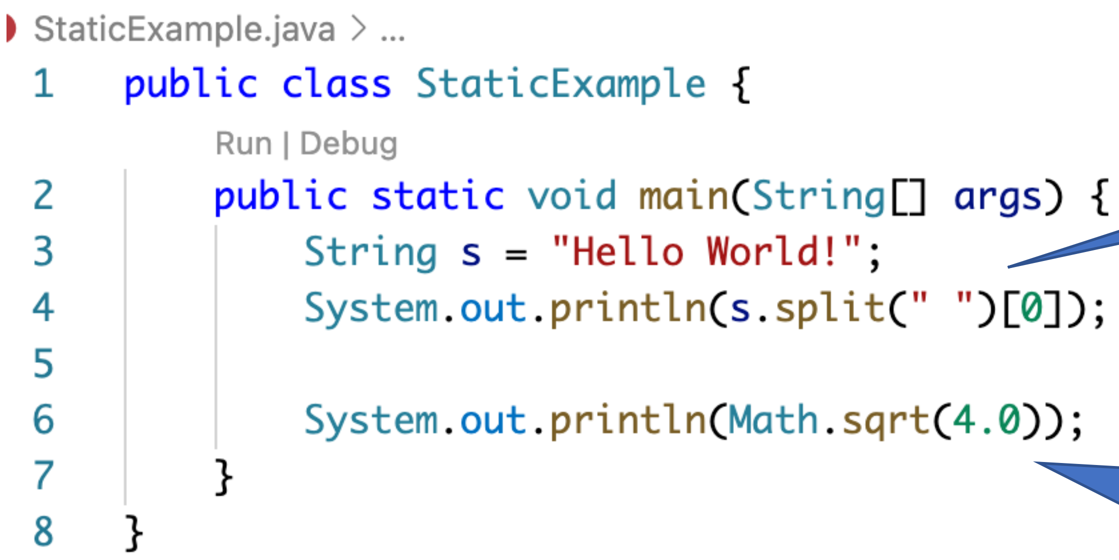
```
String s = "Hello";
String t = s;
s += " World";
System.out.printf("s: %s\n", s);
System.out.printf("t: %s\n", t);
```

Prints “s: Hello World”

Prints “t: Hello”

Static belongs to the class

- Non-static methods are called on an object, can use non-static instance variables (belong to object)
- Static methods are called on the class, *cannot* use non-static instance variables.
  - Often called utility “functions”



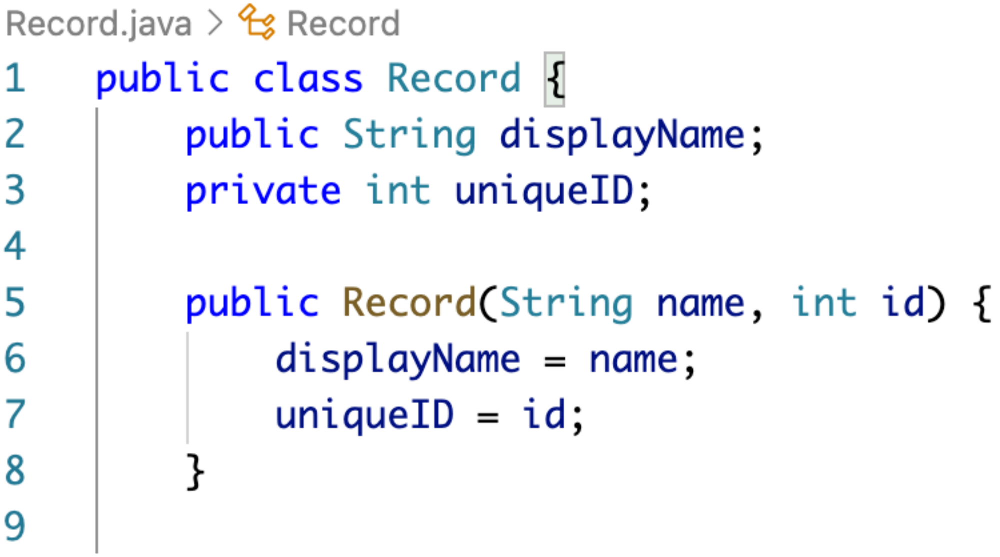
```java
public class StaticExample {
    public static void main(String[] args) {
        String s = "Hello World!";
        System.out.println(s.split(" ")[0]);

        System.out.println(Math.sqrt(4.0));
    }
}
```

Note that `split` is called on a String object

Whereas `sqrt` is called on the `Math` class

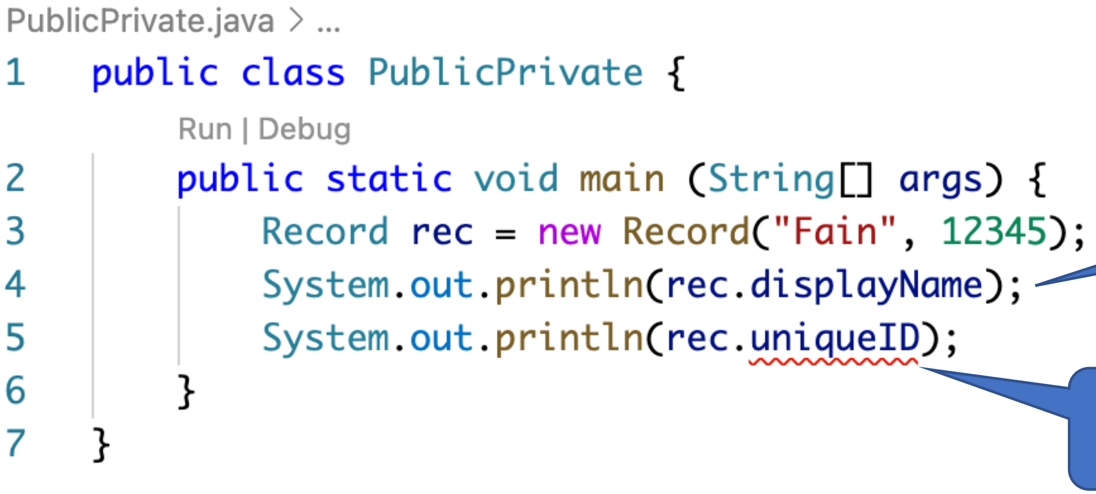
Public vs. Private

- **Public** – Can be accessed by code *outside* of the class.
- **Private** – Can *only* be accessed by code *inside* of the class.



Record.java > Record

```java
public class Record {
    public String displayName;
    private int uniqueID;

    public Record(String name, int id) {
        displayName = name;
        uniqueID = id;
    }

```



PublicPrivate.java > ...

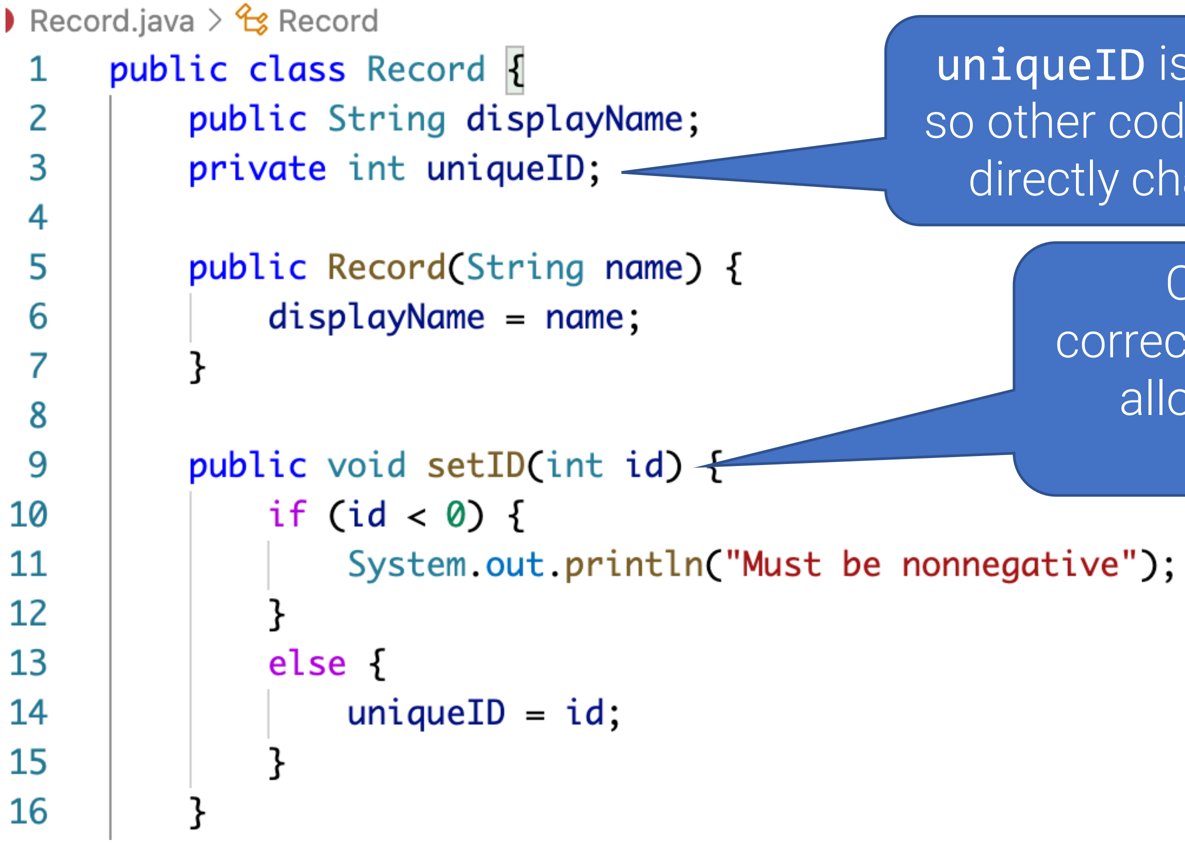
```java
public class PublicPrivate {
        Run | Debug
    public static void main (String[] args) {
        Record rec = new Record("Fain", 12345);
        System.out.println(rec.displayName);
        System.out.println(rec.uniqueID);
    }
}
```

Can access this **public** instance variable

Cannot access this **private** instance variable

The value of privacy

Suppose your entire system crashes terribly if some code is called on a negative **`uniqueID`**.



```java
public class Record {
    public String displayName;
    private int uniqueID;

    public Record(String name) {
        displayName = name;
    }

    public void setID(int id) {
        if (id < 0) {
            System.out.println("Must be nonnegative");
        }
        else {
            uniqueID = id;
        }
    }
```

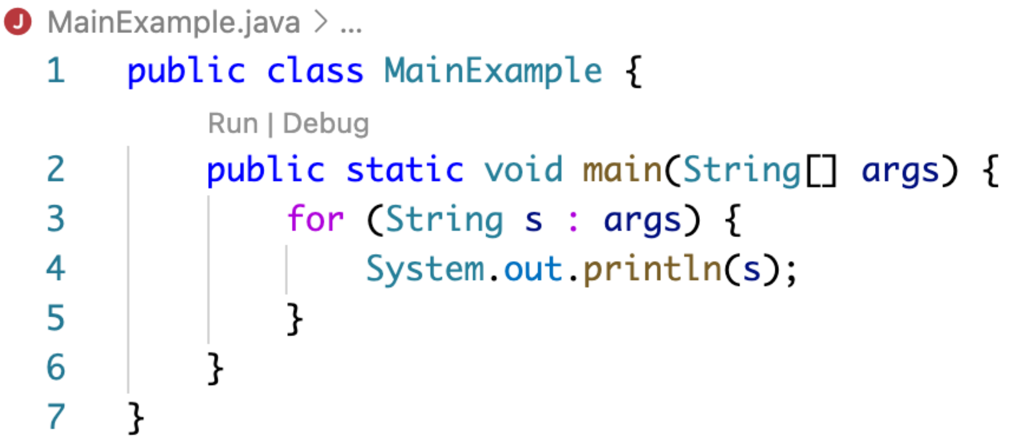
`uniqueID` is private, so other code cannot directly change it

Can check for correctness in only code allowed to change `uniqueID`

PSVM: Public Static Void Main

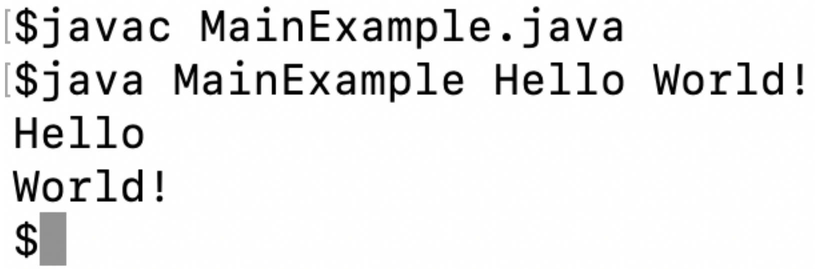
Method that is:

- `public` – can call outside of class
- `static` – belongs to class, not an object
- `void` – no return value
- `main` – starting point for a program to run



```java
public class MainExample {
    public static void main(String[] args) {
        for (String s : args) {
            System.out.println(s);
        }
    }
}
```

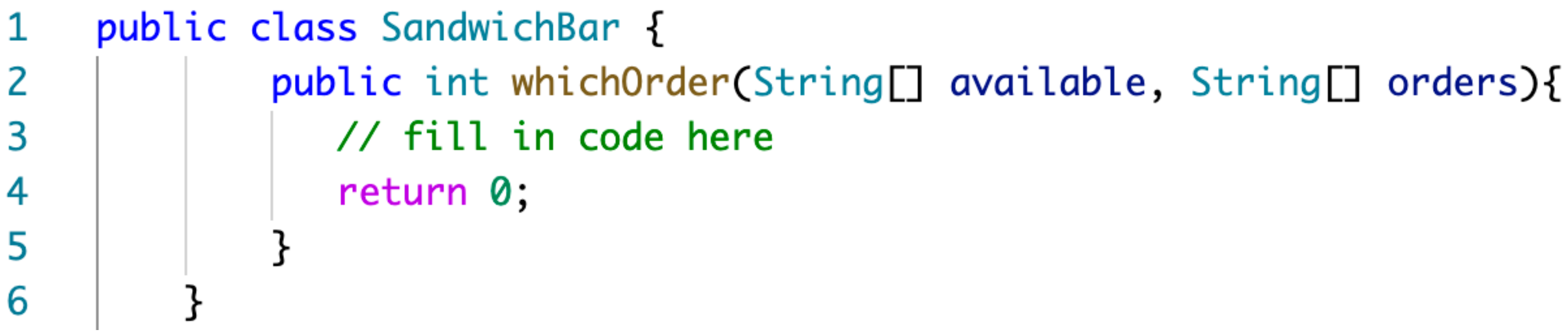
`args` allows for command-line arguments



```
[$javac MainExample.java
[$java MainExample Hello World!
Hello
World!
$
```

APT and OOP, making a PSVM method

Suppose you’re working on the SandwichBar APT.

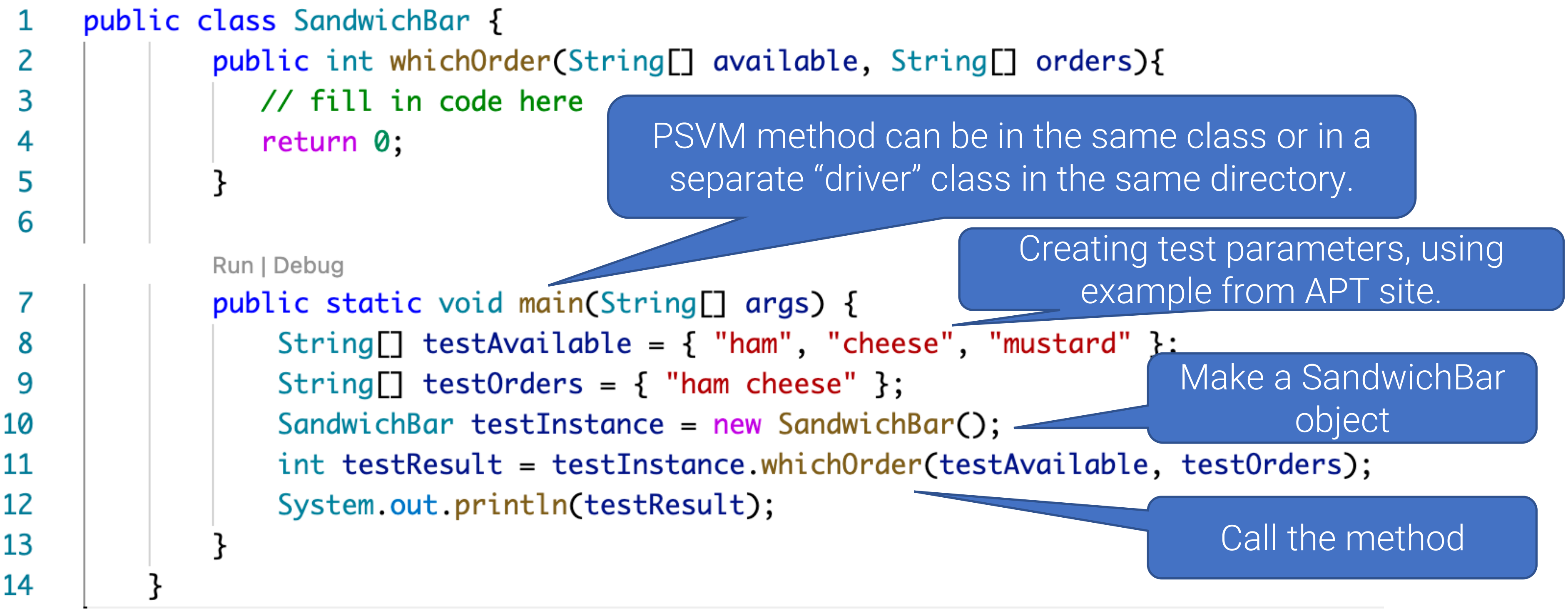


```java
public class SandwichBar {
        public int whichOrder(String[] available, String[] orders){
            // fill in code here
            return 0;
        }
}
```

Remember what you know about Java OOP:

- **`whichOrder`** is a non-static method, need to call on an *object* of the **`SandwichBar`** class.
- **`whichOrder`** has parameters, need to supply those.
- All java programs must begin in a PSVM method.

APT and OOP: Making a PSVM method



```
public class SandwichBar {
        public int whichOrder(String[] available, String[] orders){
            // fill in code here
            return 0;
        }

        Run | Debug
        public static void main(String[] args) {
            String[] testAvailable = { "ham", "cheese", "mustard" };
            String[] testOrders = { "ham cheese" };
            SandwichBar testInstance = new SandwichBar();
            int testResult = testInstance.whichOrder(testAvailable, testOrders);
            System.out.println(testResult);
        }
    }
```

PSVM method can be in the same class or in a separate “driver” class in the same directory.

Creating test parameters, using example from APT site.

Make a SandwichBar object

Call the method

Why use Classes/objects?

- Because you must in Java
- Formal specification for complex data structures
- Convenience and ease of correct programming
- Composition, Interfaces, & Implementations, Extending & Inheritance – More later!

It’s ok to not be fully “convinced” yet. But OOP has proven itself to be a powerful paradigm for designing complex, scalable software.