Compsci 201, L3: Object-Oriented Programming (OOP)

Logistics, Coming up

- Wednesday, 9/7
  - Interfaces vs. Implementations, ArrayList
  - First APT Exercises due
  - Complete at least 4 for full credit
- Friday, 9/9
  - Discussion 2: APTs, Sets, Strings, Git
- Monday 9/12
  - Prjoect 0: Person201 due (warmup project)

Course Policy Reminders

- Collaboration reminder: Can discuss projects and APTs conceptually, **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!
  - Course website 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.

Recapping some Java Themes

Comments on Java Style

Code blocks:

- Opening `{` ends first line of `if`, `for`, `while`, or method
- Indent every line inside the block
- Closing `}` on a separate line, last of block, not indented

```
16          int i=0;
17          while (i < numbers.length) {
18              System.out.println(numbers[i]);
19              i++;
20          }
```

Variable & method names:

- One-word names: `lowercase`
- Multi-word names: camelCase
- Should be informative

```
23          int index = 0;
24          int maxSize = 10;
```

More comments on Java style

Class names:

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

MethodExample.java ×

MethodExample.java > MethodExample

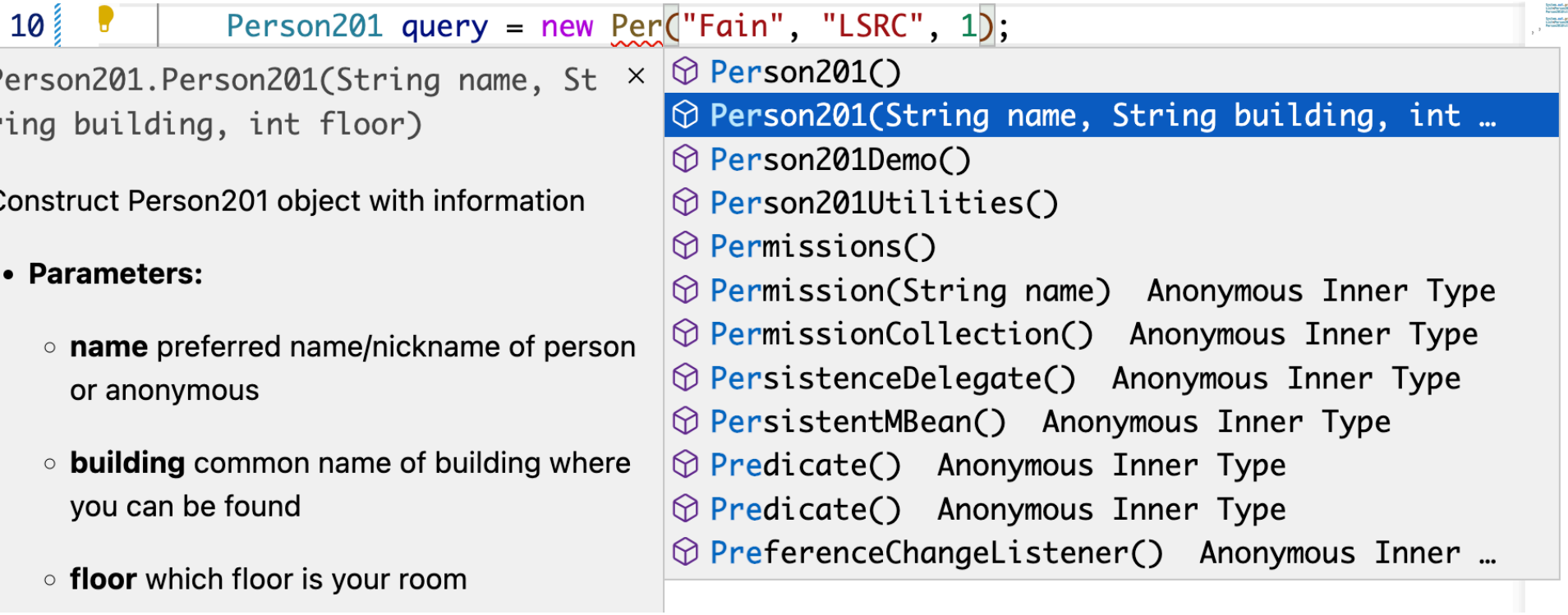
```
1   public class MethodExample {
```

Comments:

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

```
2       // one line comment
3       /* a
4       block
5       comment
6       */
```

Javadoc use


10
Person201 query = new Per("Fain", "LSRC", 1);
Person201.Person201(String name, St
ring building, int floor)
Construct Person201 object with information
• Parameters:
◦ name preferred name/nickname of person or anonymous
◦ building common name of building where you can be found
◦ floor which floor is your room
Person201()
Person201(String name, String building, int …
Person201Demo()
Person201Utilities()
Permissions()
Permission(String name) Anonymous Inner Type
PermissionCollection() Anonymous Inner Type
PersistenceDelegate() Anonymous Inner Type
PersistentMBean() Anonymous Inner Type
Predicate() Anonymous Inner Type
Predicate() Anonymous Inner Type
PreferenceChangeListener() Anonymous Inner …

Writing Javadoc

```
24      /**
25       * Construct Person201 object with information
26       * @param name preferred name/nickname of person or anonymous
27       * @param building common name of building where you can be found
28       * @param floor which floor is your room
29       */
30      public Person201(String name, String building, int floor) {
31          myName = name;
32          myBuilding = building;
33          myFloor = floor;
34      }
```

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

Java API ArrayList Reminder

Import statement:

ArrayListExample.java > ...

```
1   import java.util.ArrayList;
```

Basic usage of `ArrayList`:

- List of values of same type in order, can grow
- Uses `add()`, `get()`, `size()`, `contains()`

```
4       public static void main(String[] args) {
5           ArrayList<Integer> intList = new ArrayList<>();
6           intList.add(1);
7           intList.add(2);
8           int sum = 0;
9
10          for (int i=0; i<intList.size(); i++) {
11              sum += intList.get(i);
12          }
13          System.out.println(intList.contains(5));
```

`.add()` Appends to end of list

`.get(i)` returns i'th index element

`.contains(x)` returns `true` if x in list

Java API HashSet Reminder

An import statement:

```
1   import java.util.HashSet;
```

Basic usage of `HashSet`:

- Unordered collection, does not store duplicates
- Uses `add()`, `size()`, `contains()`

```
4       public static void main (String[] args) {
5           HashSet<String> strSet = new HashSet<>();
6           strSet.add("Hello");
7           strSet.add("World");
8           strSet.add("Hello");
9
10          if(strSet.contains("World")) {
11              System.out.println(strSet.size());
12          }
```

Prints 2, no duplicates

Java API Collections and 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 an int object, 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

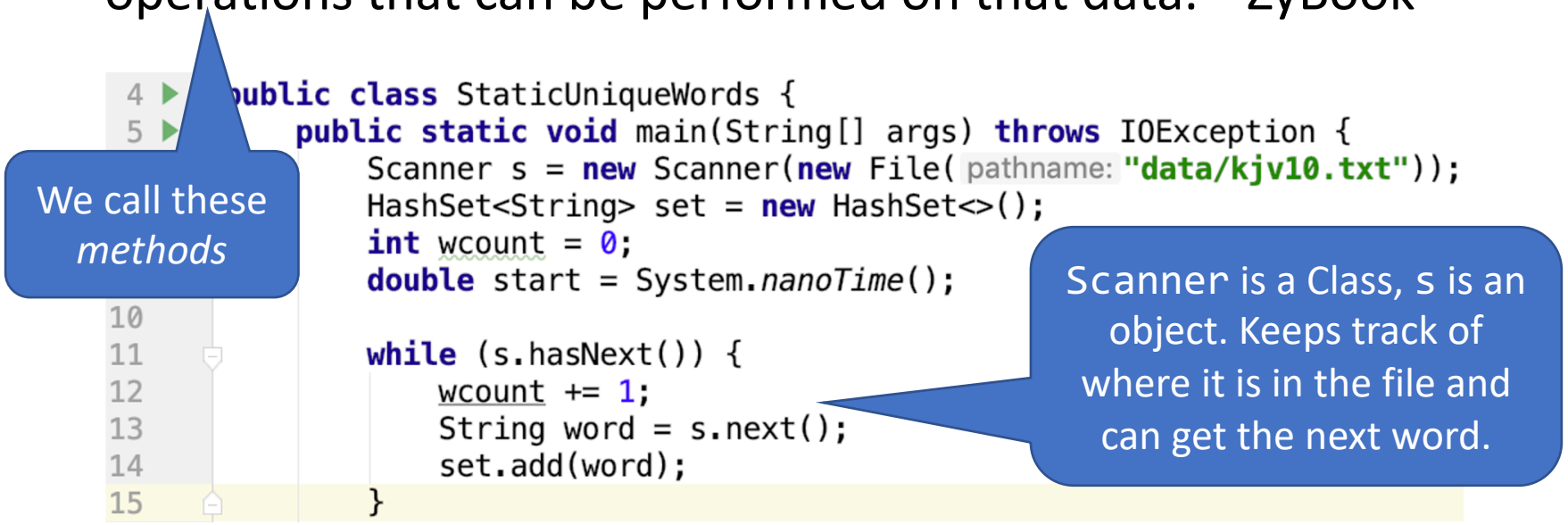
`ArrayList` <-> `Array` Conversion, Primitive Types

```
18      ArrayList<Integer> intList = new ArrayList<>();
19      int[] intArray = {2, 0, 1};
20
21      // Convert a int (or other primitive type) Array
22      // to a List by adding one at a time
23      for (int number : intArray) {
24          intList.add(number);
25      }
26
27      // Convert an Integer list to an int[] or
28      // other primitive type array one at a time
29      int[] newIntArray = new int[intList.size()];
30      for (int i=0; i<intList.size(); i++) {
31          newIntArray[i] = intList.get(i);
32      }
```

Object-Oriented Programming

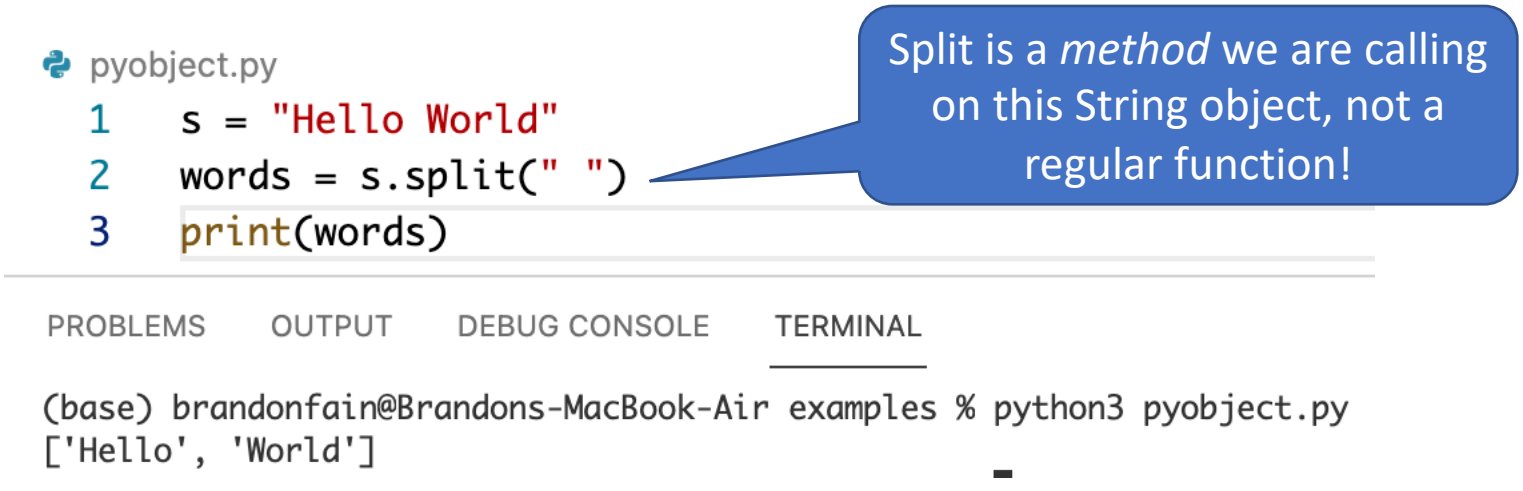
Java is object-oriented

- 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




```
4   public class StaticUniqueWords {
5       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();
10
11          while (s.hasNext()) {
12              wcount += 1;
13              String word = s.next();
14              set.add(word);
15          }
```

Aside: Python uses objects too



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

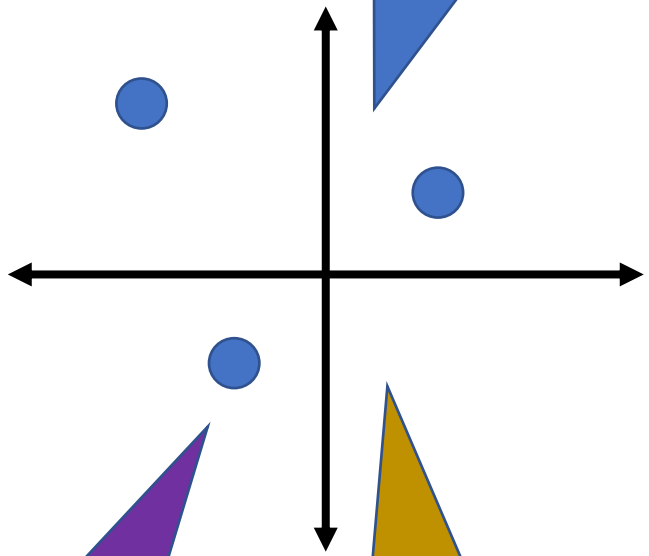
Object Concept

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

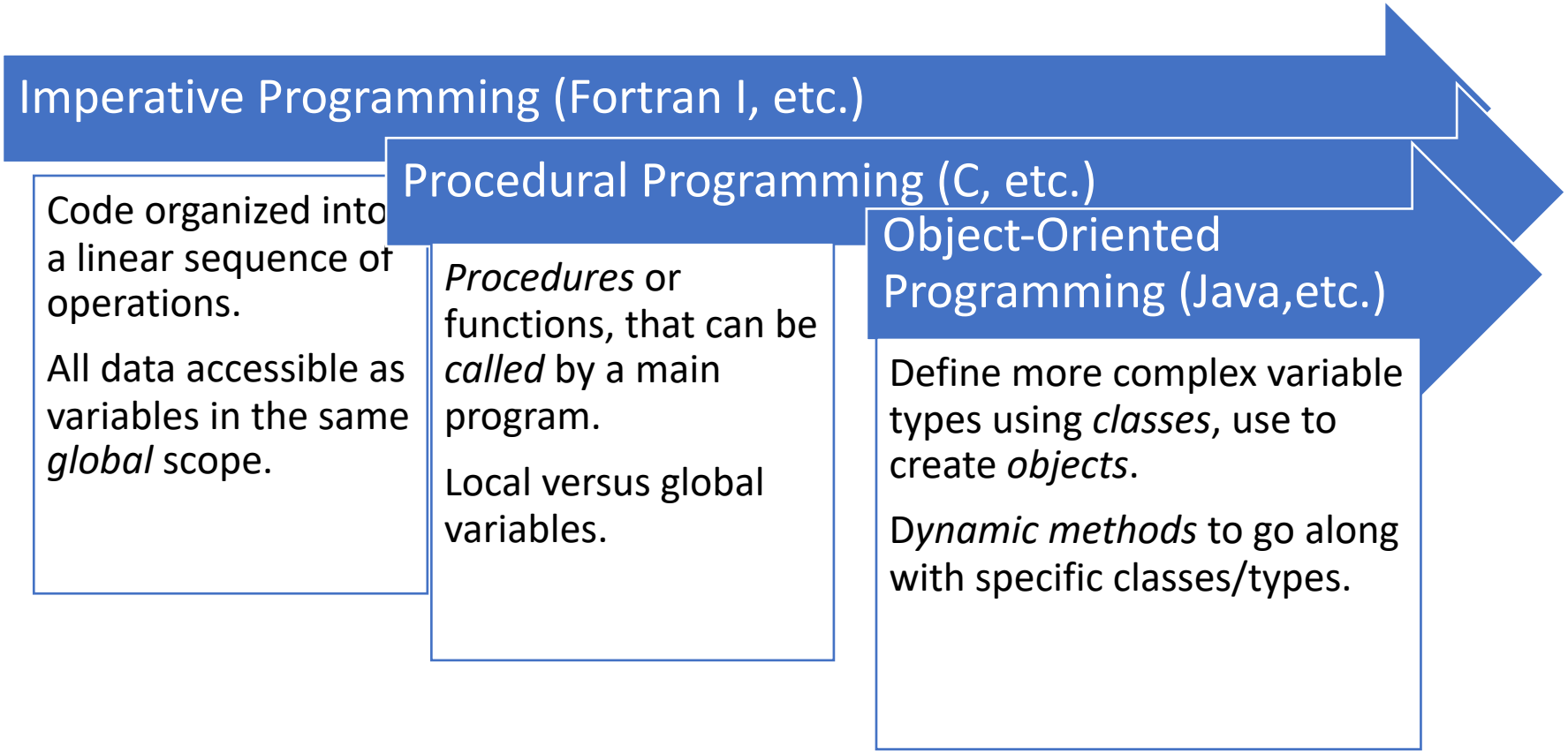


All different objects, but each of the same class

Each point object has its own x and y value.

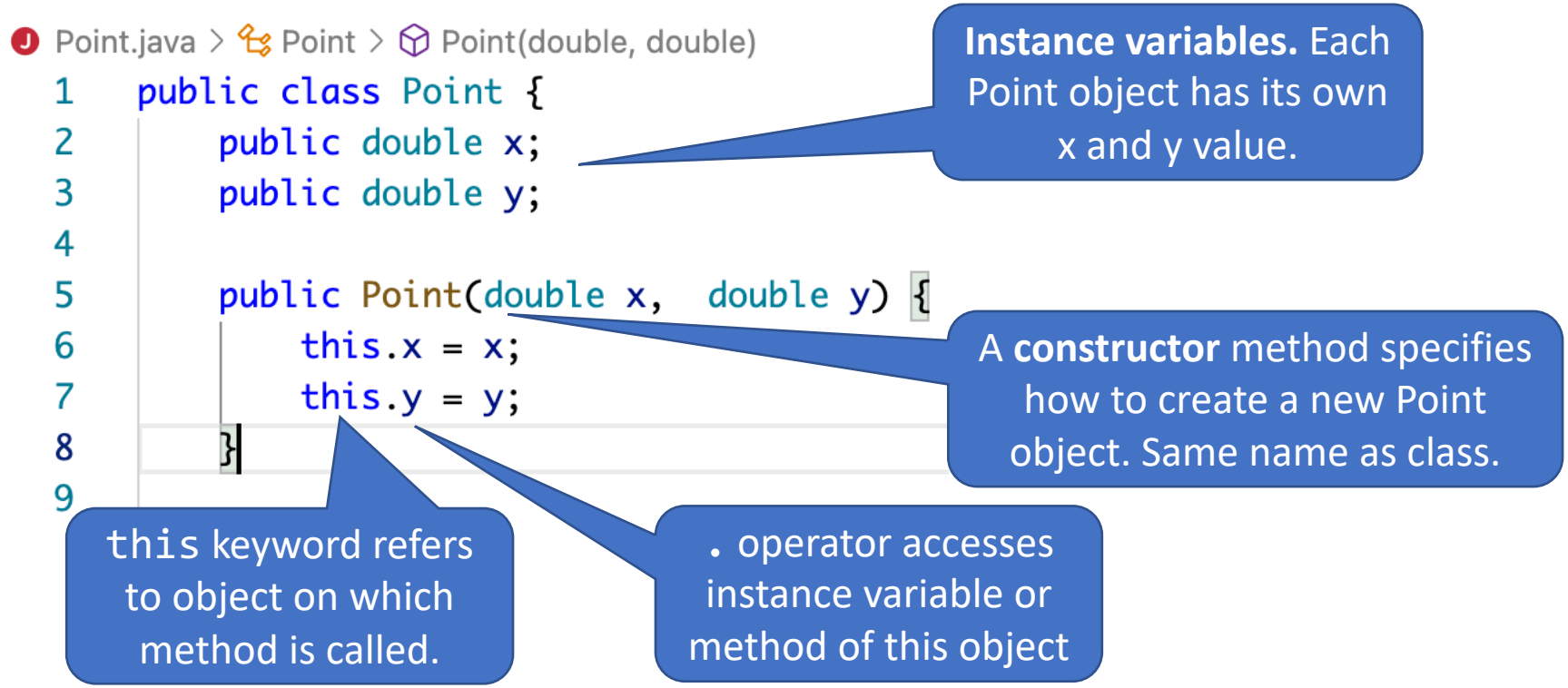
Methods should be able to operate on a particular point

Language History: A story of increasing abstraction and organization



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.



Creating objects, calling methods

```
10      public void printPoint() {
11          System.out.printf("(%.1f, %.1f)%n", x, y);
12      }

14      public static void main(String[] args) {
15          Point p = new Point(-2.0, 2.0);
16          Point q = new Point (1.0, 1.0);
17
18          p.printPoint();
19          q.printPoint();
20      }
```

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

```
4       public static void main(String[] args) {
5           HashSet<String> strSet = new HashSet<>();
6           strSet.add("Hello");
7           strSet.add("World");
8           strSet.add("Hello");
9
10          if(strSet.contains("World")) {
11              System.out.println(strSet.size());
12          }
```

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

For the **return value**, no change to object

WOTO

Go to duke.is/5vtee

Not graded for correctness, just participation.

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

But do talk to your neighbors!

== or .equals()?

Point.java

Point.java > Point > main(String[])

```
1   public class Point {
2       public double x;
3       public double y;
4
5       public Point(double x,  double y) {
6           this.x = x;
7           this.y = y;
8       }
9
        Run | Debug
10      public static void main(String[] args) {
11          Point p = new Point(0.0, 0.0);
12          Point q = p;
13          Point r = new Point(0.0, 0.0);
14
15          System.out.println(p == q);
16          System.out.println(p == r);
17      }
18
19  }
```

true

false

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

Default Object `.equals`

```
14      /*
15      @Override
16      public boolean equals(Object o) {
17          Point other = (Point) o;
18          if ((this.x == other.x) && (this.y == other.y)) {
19              return true;
20          }
21          return false;
22      }
23      */
24
        Run | Debug
25      public static void main(String[] args) {
26          Point p = new Point(0.0, 0.0);
27          Point r = new Point(0.0, 0.0);
28          System.out.println(p.equals(r));
29      }
```

Prints `false`, is just checking memory locations

Overriding default Object `.equals`

```
14
15      @Override
16      public boolean equals(Object o) {
17          Point other = (Point) o;
18          if ((this.x == other.x) && (this.y == other.y)) {
19              return true;
20          }
21          return false;
22      }
23
24
        Run | Debug
25      public static void main(String[] args) {
26          Point p = new Point(0.0, 0.0);
27          Point r = new Point(0.0, 0.0);
28          System.out.println(p.equals(r));
29      }
```

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
- 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/api

```
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.

When do I need new? Every time I create an object, not automatic!

```
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 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
```

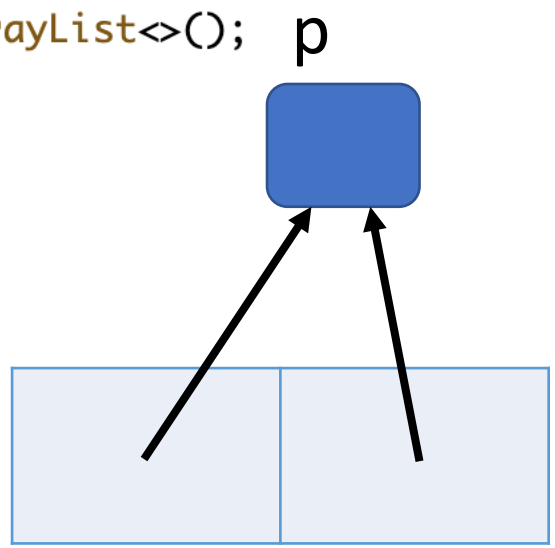
When do I need new again? For every object you want to create!

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

```
5       public static void main(String[] args) {
6           ArrayList<Point> myPoints = new ArrayList<>();
7           Point p = new Point(0.0, 0.0);
8           myPoints.add(p);
9           p.x = 2.0;
10          myPoints.add(p);
11
12          for (Point q : myPoints) {
13              q.printPoint();
14          }
```

Prints

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



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

```
1   import java.util.ArrayList;
2
3   public class Point {
4       public double x;
5       public double y;
6       public Point(double x,  double y) {
7           this.x = x;
8           this.y = y;
9       }
10
        Run | Debug
11      public static void main(String[] args) {
12          ArrayList<Point> pointList = new ArrayList<>();
13          for (int i=0; i<10; i++) {
14              pointList.add(new Point(0.0, 0.0));
15          }
16          Point p = new Point(0.0, 0.0);
17          System.out.println(pointList.contains(p));
18      }
```

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!

WOTO

Go to duke.is/n5r6b

Not graded for correctness, just participation.

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

But do talk to your neighbors!

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

```
1   public class Record {
2       public String displayName;
3       private int uniqueID;
4
5       public Record(String name, int id) {
6           displayName = name;
7           uniqueID = id;
8       }
9
```

PublicPrivate.java > ...

```
1   public class PublicPrivate {
        Run | Debug
2       public static void main (String[] args) {
3           Record rec = new Record("Fain", 12345);
4           System.out.println(rec.displayName);
5           System.out.println(rec.uniqueID);
6       }
7   }
```

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`.

Record.java > Record

```
1   public class Record {
2       public String displayName;
3       private int uniqueID;
4
5       public Record(String name) {
6           displayName = name;
7       }
8
9       public void setID(int id) {
10          if (id < 0) {
11              System.out.println("Must be nonnegative");
12          }
13          else {
14              uniqueID = id;
15          }
16      }
```

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

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

(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";
s += " World";
```

More like

```
String sOld = "Hello";
String sNew = "" + sOld + " World";
```

(and then get rid of `sOld`)

Static belongs to the class

- Regular instance variables and methods are called on an object.
- Static methods are called on the class, do not use any instance variables. Often utility “functions”

StaticExample.java > ...

```
1   public class StaticExample {
        Run | Debug
2       public static void main(String[] args) {
3           String s = "Hello World!";
4           System.out.println(s.split(" ")[0]);
5
6           System.out.println(Math.sqrt(4.0));
7       }
8   }
```

Note that `split` is called on a String object

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

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

`args` allows for command-line arguments

MainExample.java > ...

```
1   public class MainExample {
        Run | Debug
2       public static void main(String[] args) {
3           for (String s : args) {
4               System.out.println(s);
5           }
6       }
7   }
```

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

APT and OOP, making a PSVM method

Suppose you're working on the SandwichBar APT.

```
1   public class SandwichBar {
2           public int whichOrder(String[] available, String[] orders){
3              // fill in code here
4              return 0;
5           }
6       }
```

Remember what you know about Java OOP:

- whichOrder is a regular 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

```
1   public class SandwichBar {
2           public int whichOrder(String[] available, String[] orders){
3              // fill in code here
4              return 0;
5           }
6
            Run | Debug
7           public static void main(String[] args) {
8               String[] testAvailable = { "ham", "cheese", "mustard" };
9               String[] testOrders = { "ham cheese" };
10              SandwichBar testInstance = new SandwichBar();
11              int testResult = testInstance.whichOrder(testAvailable, testOrders);
12              System.out.println(testResult);
13          }
14      }
```

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.

Fred Brooks, Why is programming fun?

- Duke ‘53
- Founded Compsci @ UNC
- Turing award winner, design

1. Sheer joy of making things
2. Pleasure of making things that are useful
3. Fascination of fashioning complex puzzle-like objects
4. Delight in working in such a tractable medium [like a poet]