CompSci 201, L2
Intro to Java

Course Staff Re-Intros

- Instructor: Brandon Fain
- Teaching Associate: Kate O'Hanlon
- Graduate Teaching Assistants:
  - Zhekun Wu
  - Yilun Song
- Undergraduate Teaching Assistants: Many! Will add pictures and names to website getting help page.

Logistics, Coming up

- This Friday, 9/2
  - First Discussion Section
- Next Monday, 9/5
  - OOP (object-oriented programming) in Java
- Next Wednesday 9/7
  - Interfaces, Implementations, ArrayList
  - First APT Exercises Due

Data Collection

Go to duke.is/zeqkj

Collecting a dataset for Project 0

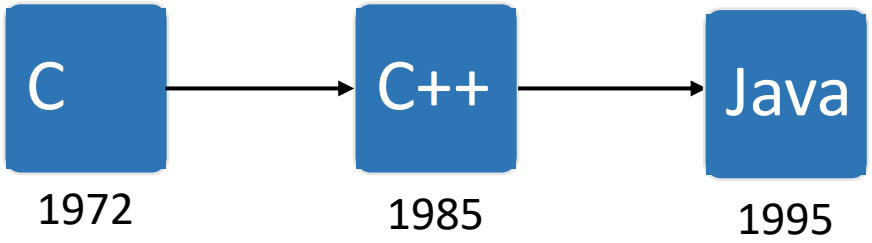
Info you enter will be in the dataset for everyone to see, don't enter anything you consider private or sensitive.

Algorithmic Problem Solving: Before Coding…

1. Understand the problem carefully
2. Work examples (small, by hand)
3. Gather insights (generalize examples?)
4. Make a plan (e.g., outline of algorithm)

Modeled by the upcoming CirclesCountry APT walkthrough in Discussion 1.

A very brief history of Java



- C. Streamlined language developed for writing operating systems and low-level systems utilities.
- C++. Can do everything in C (manual memory management), adds support for object-oriented programming (OOP).
- Java. Requires OOP, Automatic memory management, stronger compile time guarantees, more device independent.

Java is a compiled language

How is the program you write in source code translated into something instructions the machine can *execute?*

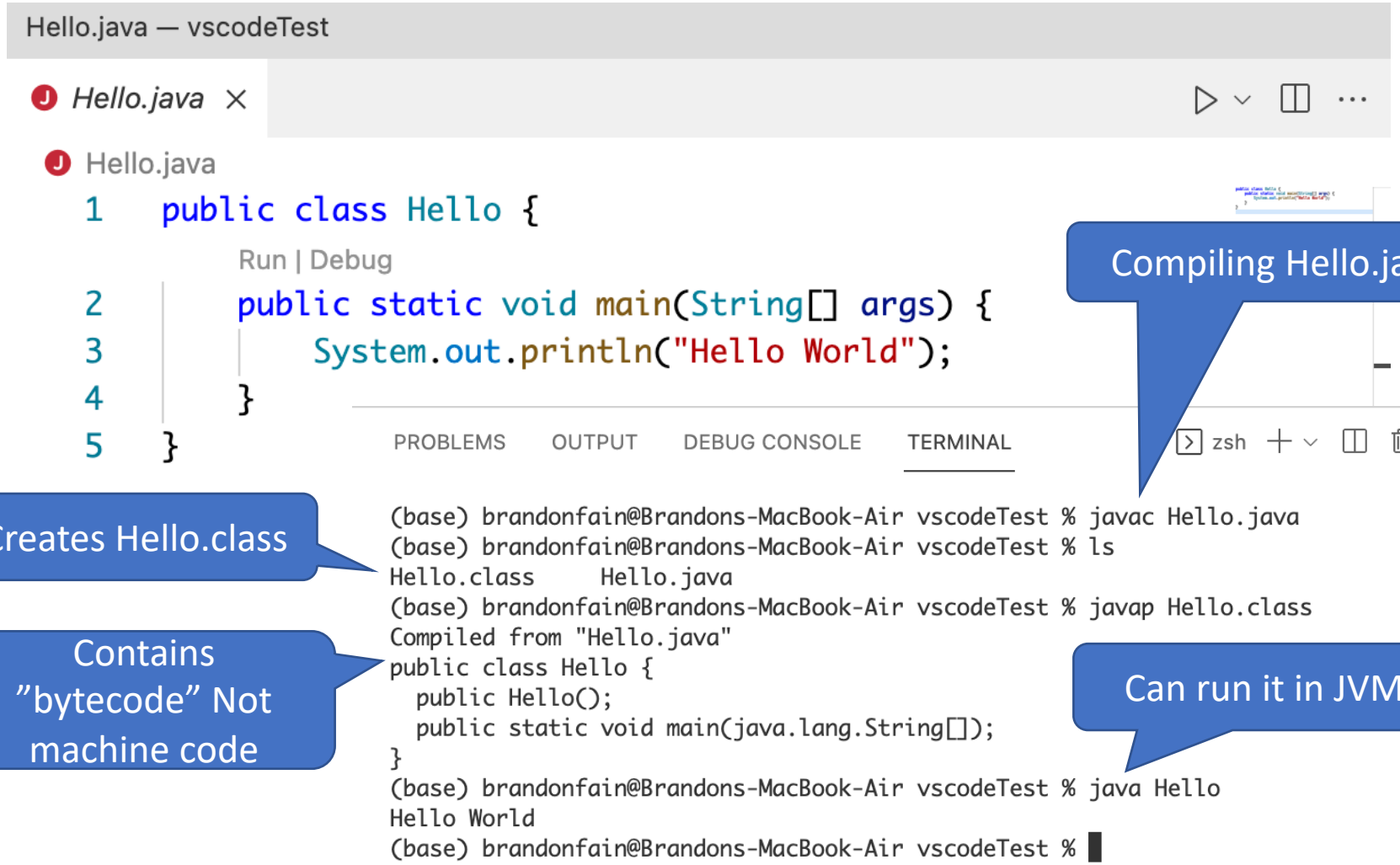
Compiled

- All at once
- Compiler is another program that translates source code into machine code*.
- Run the *executable*, the output of the compiler.

Interpreted

- Line at a time
- Interpreter is another program that translates *and* runs a program line by line.
- Python is an interpreted language.

The “Java Virtual Machine”



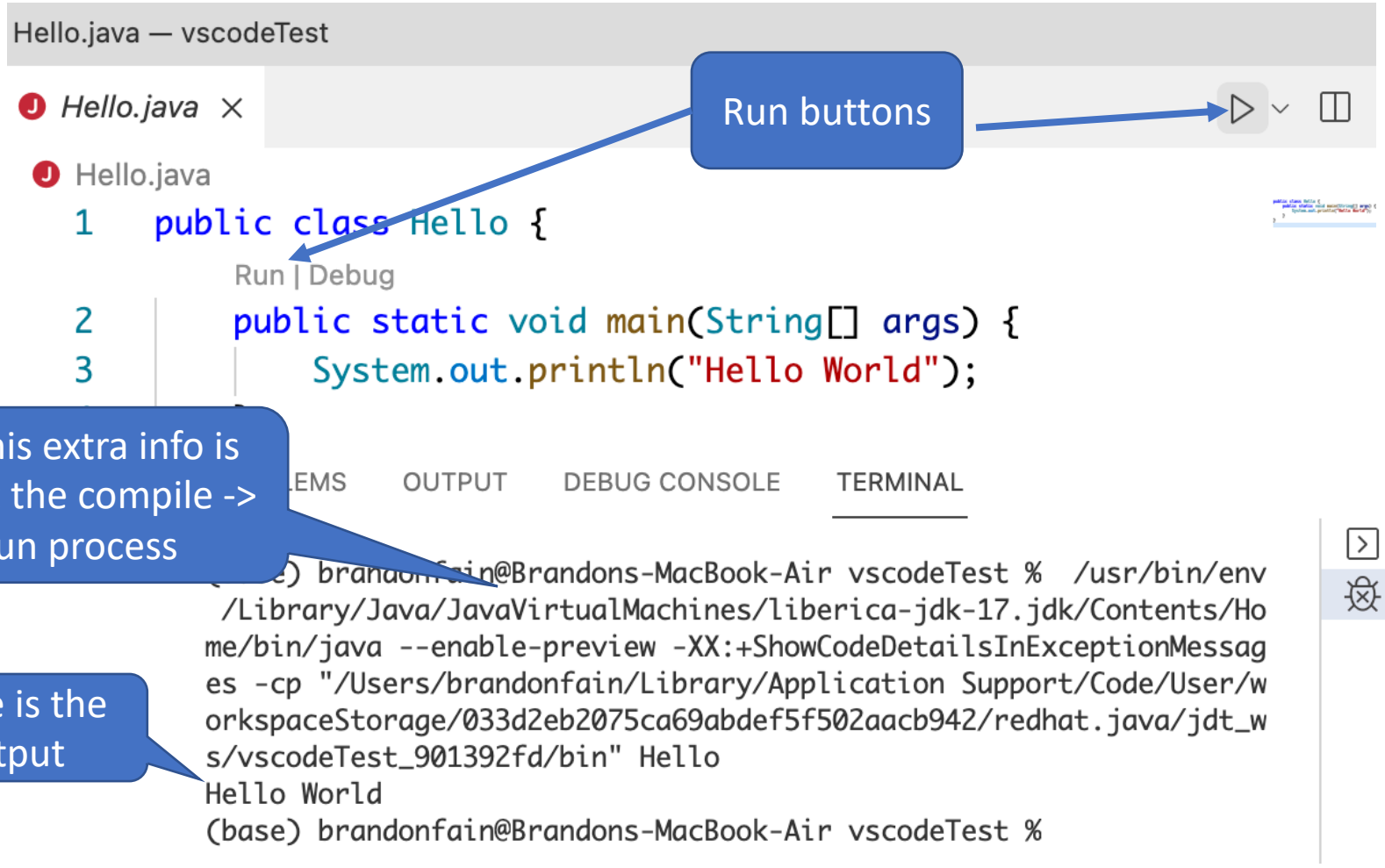
Interlude: Command Line?

Command	Meaning	Details
`pwd`	Print Working Directory	Shows the full file path to the directory you are currently in
`ls`	List Files	Shows all files and directories contained in the current directory
`cd`	Change Directory	• **`cd`** by itself goes to your home directory • **`cd directory`** goes to the specified directory • **`cd ..`** goes to the enclosing directory
`mkdir`	Make Directory	• **`mkdir directory`** creates a directory
`cp`	Copy	**`cp source target`** Copies the `source` file and names the result `target`.
`rm`	Remove	**`rm file`** deletes the specified file. No backups!!!

Interlude: Compile and Run Java

Command	Meaning	Details
`javac`	Compile .java files to .class files	• **`javac file.java`** compiles and creates `file.class` • **`javac *.java`** compiles **all** .java files in current directory to .class files.
`java`	Run java class files	**`java file`** executes the main method of `file.class`. Must have already been compiled from `file.java`.

Pressing the “run” button in VS Code does these steps for you



Java Basics

- OOPs, we need some vocabulary:
  - Class, Object, object, fields, methods, constructor, mutators, accessors, instance variables, static, …
- But first the basics:
  - Each Java program file contains a single `class`
  - The file is named `<className>.java`
  - To run a program, must have a `public static void main` (PSVM) method
- Larger projects have multiple classes / .java files, only one needs a PSVM to start program.

Hello.java ×

Hello.java

```
1   public class Hello {
        Run | Debug
2       public static void main(String[] args) {
3           System.out.println("Hello World");
4       }
5   }
```

Java is **strongly typed**

Must be explicit about the **type** of every variable.

Type.java > ...

```java
public class Type {
    Run | Debug
    public static void main(String[] args) {
        int x = 5;
        System.out.println(x/2);
    }
}
```

Prints 2

type.py

```python
x = 5
print(x/2)
```

Prints 2.5

Type.java > ...

```java
public class Type {
    Run | Debug
    public static void main(String[] args) {
        int x = 5;
        System.out.println((double)x/2);
    }
}
```

Prints 2.5

Notice also that every method must specify the *type* of what it returns (void means nothing).

Can **cast** to convert types `(NewType) var`

Strong typing allows the compiler to help you avoid mistakes

StrongTyping.java 1 ×

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

```
1   public class StrongTyping {
2       public static String getFirstWord(String s) {
3           return s.split(" ")[0];
4       }
        Run | Debug
5       public static void main(String[] args) {
6           System.out.println(getFirstWord(201));
7       }
8   }
9
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL

```
(base) brandonfain@Brandons-MacBook-Air examples % javac StrongTyping.java
StrongTyping.java:6: error: incompatible types: int cannot be converted to
String
        System.out.println(getFirstWord(201));
                                        ^
```

Java primitive types

- Primitive types in Java: Don't need `new` to create.
  - `byte`, `short` (rarely used in this course)
  - `int`, `long` (common integer types)
  - `float`, `double` (common decimal number types)
  - `boolean` (true or false)
  - `char` (for example, `'a'` or `'x'`).

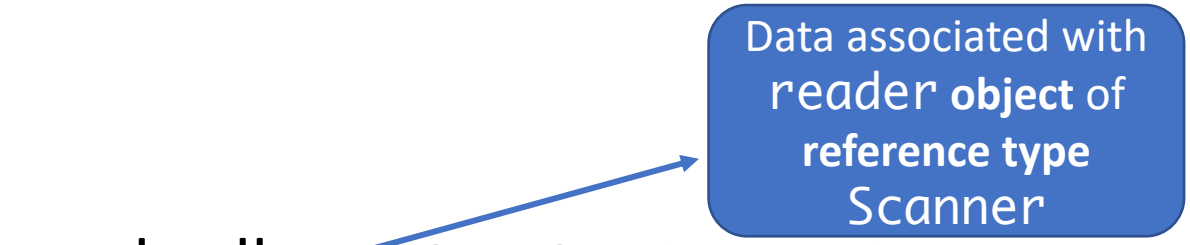
Java Basic Operators

+, -	Add, subtract
*, /	Multiply, divide (careful with divide, 5/4 gives 1)
%	Modulus (remainder in int division, if % 2 == 0 then even, if % 2 == 1 then odd)
<, <=	Less than, less than or equal to
>, >=	Greater than, greater than or equal to
==	Equal (only for primitive types!!!)
!	Logical NOT (!a means a must not be true)
&&	Logical AND (a && b means a and b need to be true)
||	Logical OR (a || b means a could be true, or b, or both)

Special Case: String

- NOT primitive, but can initialize in two ways:
  - `String s = “Hello”;` or `String s = new String(“Hello”);`
- + is overloaded to concatenate Strings:
  - `String s = “Hello”;`
  - `String t = “ World”;`
  - `System.out.println(s + t);` prints “Hello World”

Java Reference Types



- Need to use **new** and call **constructor** to create.

```
Scanner reader = new Scanner(
```

- Variable stores a **reference** to an **object**, i.e., a place in memory.
- Can access instance variables and method calls with the **dot operator**.

```
while (reader.hasNext()) {
    String word = reader.next();
```

Java uses {} and ; to denote blocks and statements

Block.java

```java
public class Block {
    Run | Debug
    public static void main(String[] args) {
        int x = 4;
        if (x % 2 == 0) {
            System.out.println("even");
        }
        else {
            System.out.println("odd");
        System.out.println("will this print?");
        }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
(base) brandonfain@Brandons-MacBook-Air examples % javac Bloc
(base) brandonfain@Brandons-MacBook-Air examples % java
even
```

; ends a *statement* / denotes an operation

{…} denotes a block of code, e.g., for an if statement, loop, or method

block.py

```python
x = 4
if (x % 2 == 0):
    print("even")
else:
    print("odd")
print("will this print?")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

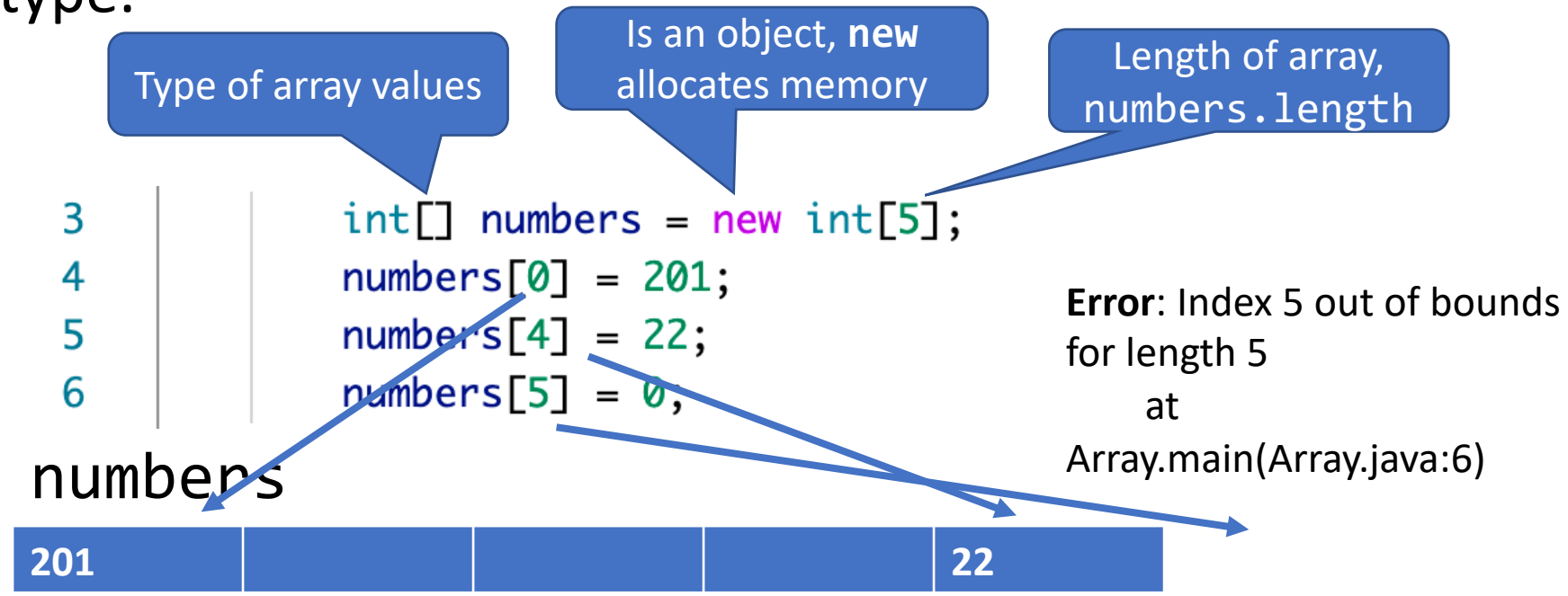
```
(base) brandonfain@Brandons-MacBook-Air examples % python3 block.py
even
will this print?
```

newline ends statement in Python

And indentation denotes blocks.
Still a style convention in Java!

Java arrays

An **array** holds a *fixed* number of values of a single type.



Java Strings: Concepts and Methods

Strings are objects that hold an array of characters.


```
3       String message = "Hi CS 201!";
4       System.out.println(message.length());
5       System.out.println(message.charAt(0));
6       System.out.println(message.substring(0, 4));
7       System.out.println(message.equals("Hi CS 201!"));
```

10

'H'

"Hi C"

True

Can even convert to `char[]` and back

```
9       char[] letters = message.toCharArray();
10      String originalMessage = new String(letters);
```

Note on Java characters

Java characters are ordered, comparable, correspond to integer values.

```
 9      for (char ch='a'; ch <= 'z'; ch++) {
10          System.out.printf("Char: %c, Val: %d%n", ch, (int)ch);
11      }
```

```
Char: a, Val: 97
Char: b, Val: 98
Char: c, Val: 99
Char: d, Val: 100
Char: e, Val: 101
Char: f, Val: 102
Char: g, Val: 103
Char: h, Val: 104
Char: i, Val: 105
Char: j, Val: 106
Char: k, Val: 107
Char: l, Val: 108
Char: m, Val: 109
Char: n, Val: 110
```

Values are how characters are *encoded* on a machine

Java loops

Creates an `int` variable, starting at 0, accessible only inside the loop block.

Loop while `i < numbers.length`

Increase `i` by 1 each time through loop

number takes each value in numbers in turn

Regular for

```
 8        for (int i=0; i<numbers.length; i++) {
 9            System.out.println(numbers[i]);
10        }
```

Enhanced for

```
12        for (int number : numbers) {
13            System.out.println(number);
14        }
```

while

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

WOTO

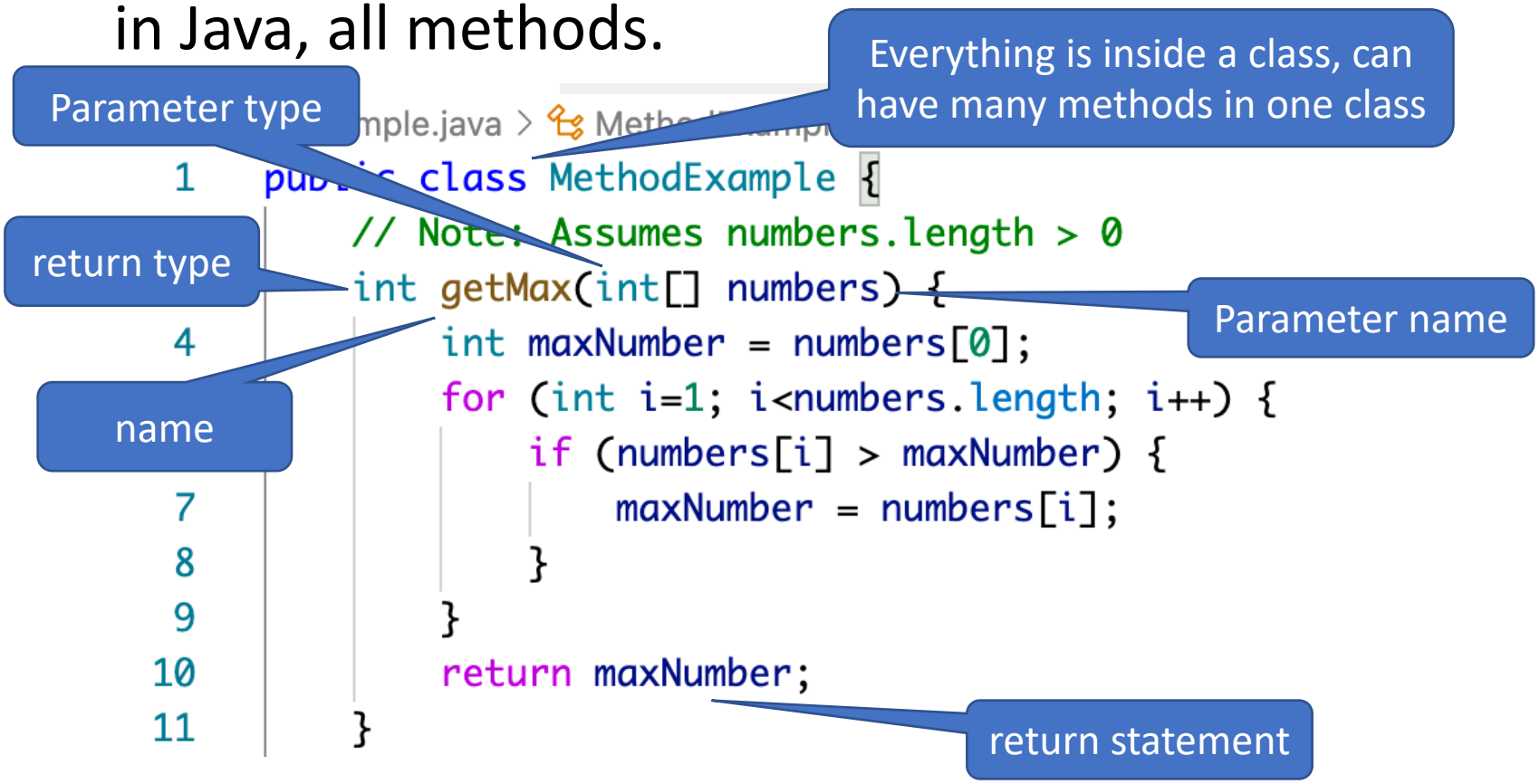
Go to duke.is/j7me4

Not graded for correctness, just participation. Try to answer *without* looking back at slides and notes.

Research indicates this kind of *retrieval practic* is one of the most powerful ways to learn. See Roediger, H. L., & Karpicke, J. D. (2006). Test-Enhanced Learning: Taking Memory Tests Improves Long-Term Retention. *Psychological Science*, *17*(3), 249–255. https://doi.org/10.1111/j.1467-9280.2006.01693.x

Java methods

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



Static vs. Dynamic Methods

- Dynamic methods are called on a created **object.** Has access to object data *and* arguments.
- Static methods are called on the **class**. Only has access to arguments. 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

Java API ArrayList

An import statement:

ArrayListExample.java > ...

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

More on `ArrayList` next time, but the basics:

- Generic to specify type, can grow dynamically
- 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));
```

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

Java API HashSet

An import statement:

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

More on `HashSet` later, but the basics:

- Generic to specify type, does not store duplicates
- Uses `add()`, `size()`, `contains()`

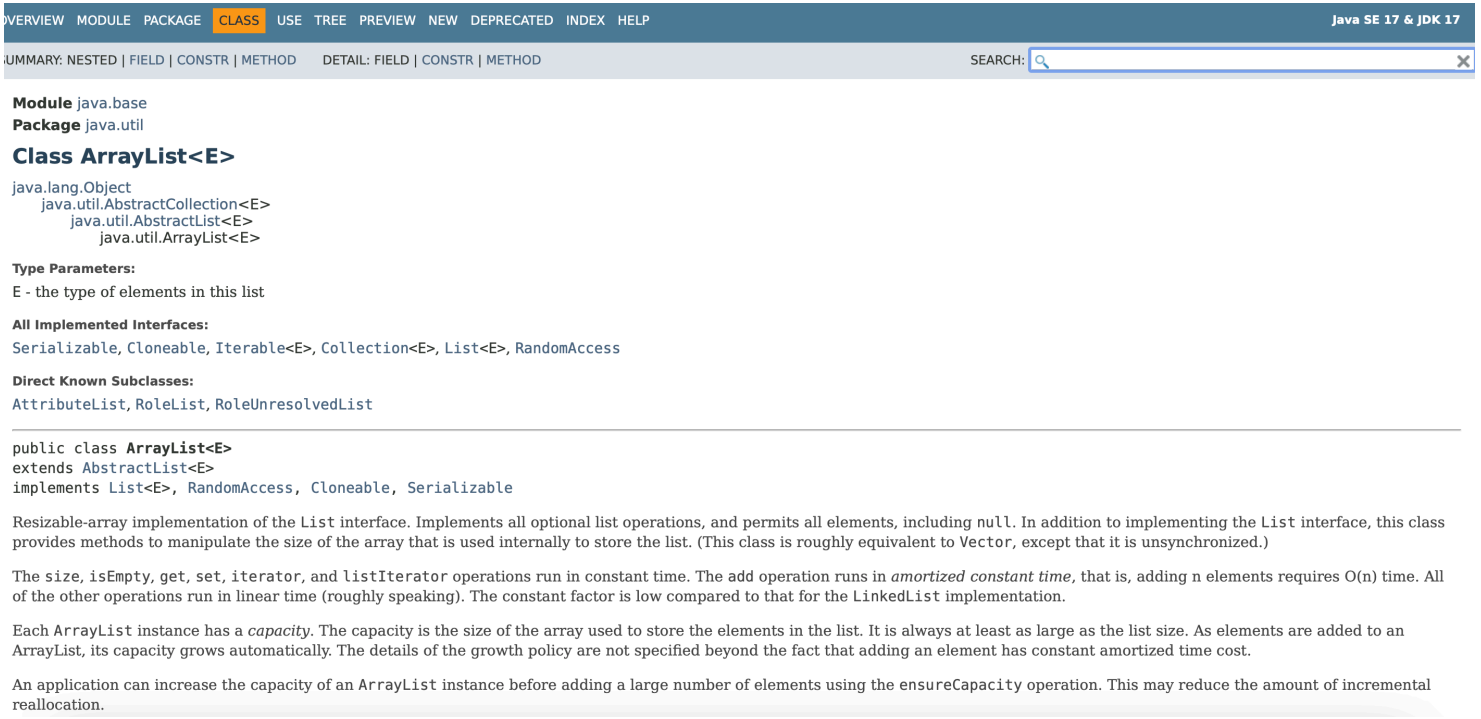
```
4       public static void main (String[] args) {
5           HashSet<String> strSet = new HashSet<>();
6           strSet.add("Hello");
7           strSet.add("World");
8           strSet.add("Hello");
9
10          if(strSet.contains("World")) {
11              System.out.println(strSet.size());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

Prints 2, no duplicates

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

UMMARY: 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 duke.is/mem3h

Not graded for correctness, just participation.

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

But do talk to your neighbors!

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:

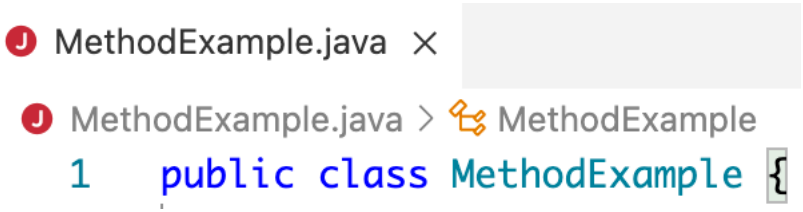
- One-word names: `lowercase`
- Multi-word names: camelCase
- Should be informative

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

More comments on Java style

Class names:

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



Comments:

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

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

PSVM: Public Static Void Main

Method that is:

- public – can call outside of class
- static – belongs to class, not an object
- void – no return
- 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!
$
```

Do I need PSVM for APTs?

- No, generally writing a single regular method.
- PSVM method is supplied by the APT server that runs your code.
- Can write a PSVM to test your code locally, but…
- APT Server WILL NOT allow you to have static methods when you submit

It’s going to be ok

For many of you:

- Java has new *syntax* to learn, and
- Object-oriented programming is a new *paradigm*

It’s normal for it to feel “strange” at first!

Resources:

- ZyBook, optional chapters 1-7 are intro java review
- First Discussions, first sets of APTs, Projects P0 and P1 designed to help practice
- Peers, Ed discussion, Office hours, all can help

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]