CompSci 201, L2
Intro to Java

Logistics, Coming up

- This Friday, 1/20
  - First discussion section meetings
- Next Monday, 1/23
  - Intro to OOP (object-oriented programming) in Java
- Next Wednesday 1/25
  - Interfaces, Implementations, ArrayList data structure
  - First APT set (short programming exercises) due

Helper Hours

- **What:** Drop-in time to ask TAs questions about course content (concepts, Java, APTs, projects).
- **When:** Sunday-Thursdays starting this Sunday 1/22.
- **Where:** In-person, and virtual options.
- **How:**
  - Try / think on your own
  - OhHai queue to post your question
  - Talk with a TA for ~5-15 minutes
  - Iterate
- **Details:** See the **Getting Help page** of the website.

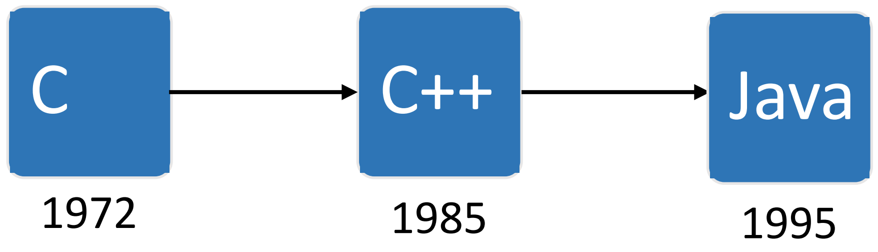
Person in CS: Fred Brooks

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

Why is programming fun?

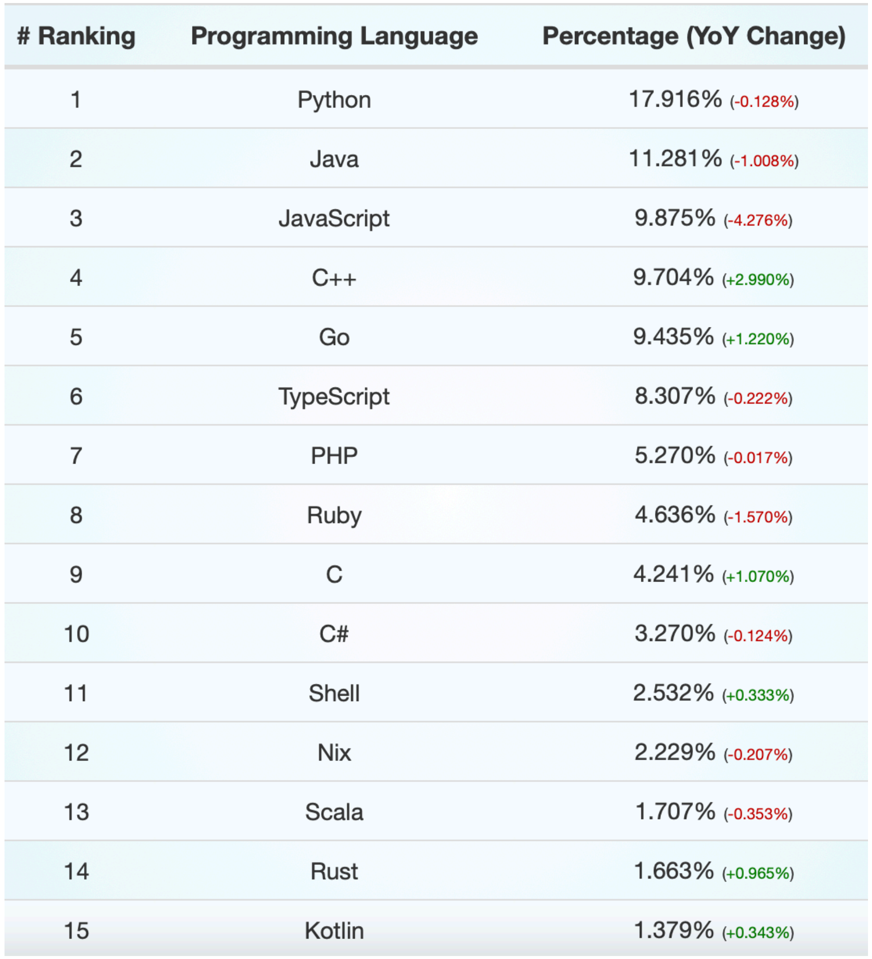
- “joy of making things…that are useful”
- “Fascination of fashioning complex puzzle-like objects”
- “Delight in working in such a tractable medium”

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 common language



# Ranking	Programming Language	Percentage (YoY Change)
1	Python	17.916% (-0.128%)
2	Java	11.281% (-1.008%)
3	JavaScript	9.875% (-4.276%)
4	C++	9.704% (+2.990%)
5	Go	9.435% (+1.220%)
6	TypeScript	8.307% (-0.222%)
7	PHP	5.270% (-0.017%)
8	Ruby	4.636% (-1.570%)
9	C	4.241% (+1.070%)
10	C#	3.270% (-0.124%)
11	Shell	2.532% (+0.333%)
12	Nix	2.229% (-0.207%)
13	Scala	1.707% (-0.353%)
14	Rust	1.663% (+0.965%)
15	Kotlin	1.379% (+0.343%)

- Based on an analysis of Github repositories.

Java is a compiled language

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

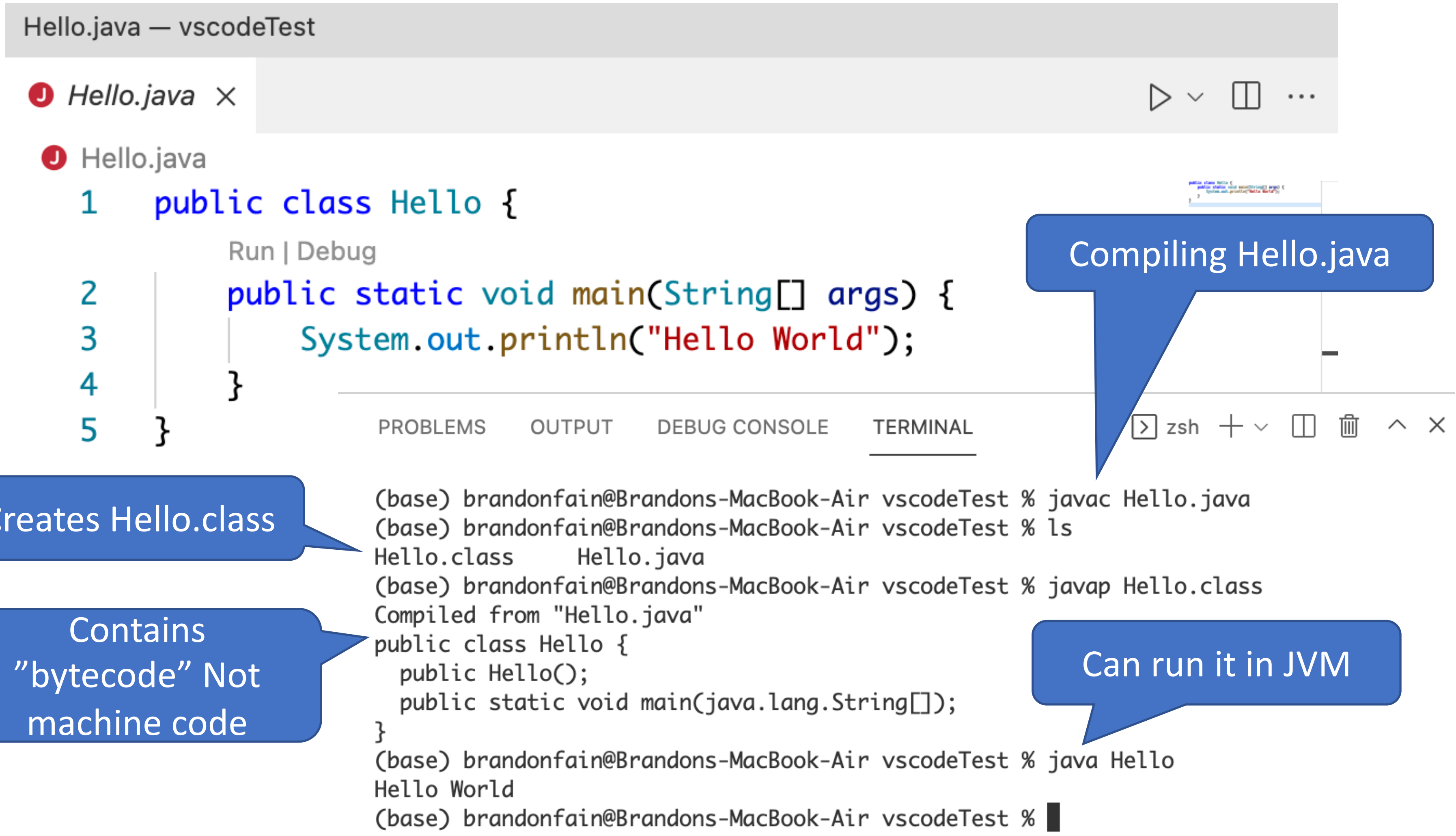
Compiled

- 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”



Hello.java — vscodeTest

Hello.java

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

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

```
(base) brandonfain@Brandons-MacBook-Air vscodeTest % javac Hello.java
(base) brandonfain@Brandons-MacBook-Air vscodeTest % ls
Hello.class     Hello.java
(base) brandonfain@Brandons-MacBook-Air vscodeTest % javap Hello.class
Compiled from "Hello.java"
public class Hello {
  public Hello();
  public static void main(java.lang.String[]);
}
(base) brandonfain@Brandons-MacBook-Air vscodeTest % java Hello
Hello World
(base) brandonfain@Brandons-MacBook-Air vscodeTest % 
```

Compiling Hello.java

Creates Hello.class

Contains ”bytecode” Not machine code

Can run it in JVM

Interlude: Command Line?

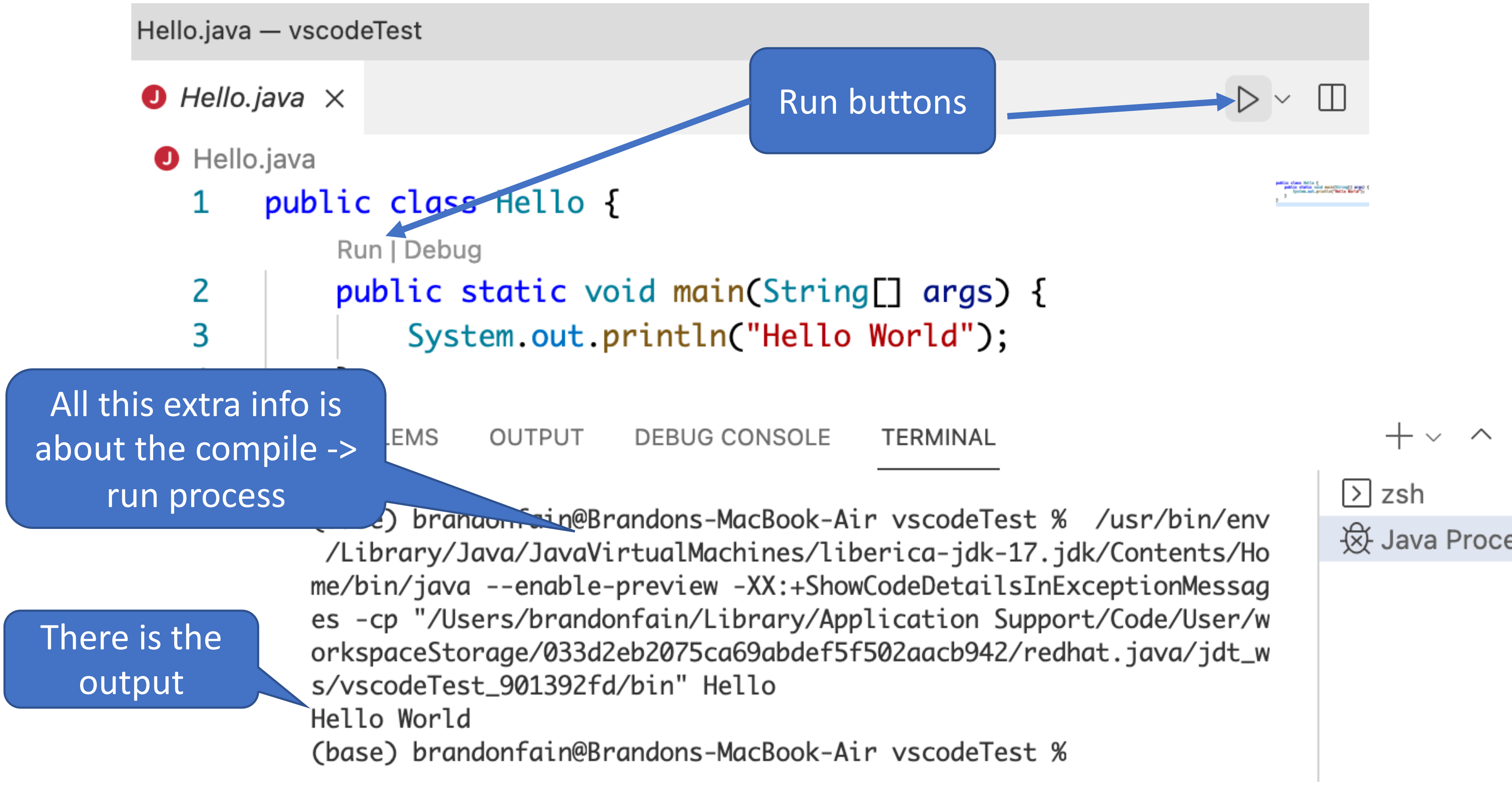
Command	Meaning	Details
`pwd`	Print Working Directory	Shows the full file path to the directory you are currently in
`ls`	List Files	Shows all files and directories contained in the current directory
`cd`	Change Directory	• **`cd`** by itself goes to your home directory • **`cd directory`** goes to the 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`.

See the javac documentation for more options

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

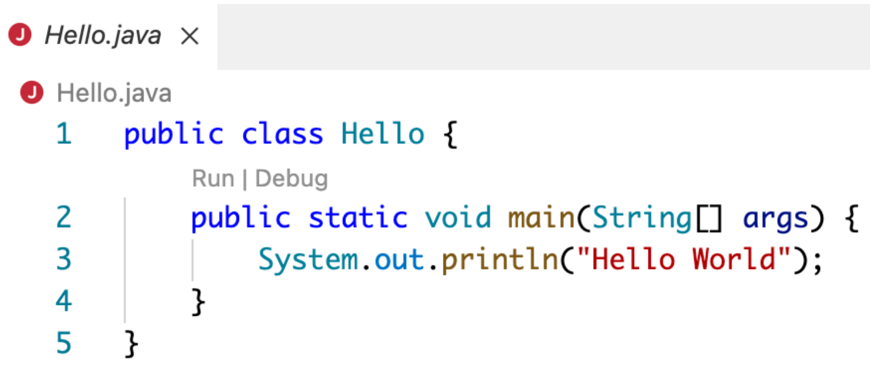


```java
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello World");
```

```
(base) brandonfain@Brandons-MacBook-Air vscodeTest %  /usr/bin/env
 /Library/Java/JavaVirtualMachines/liberica-jdk-17.jdk/Contents/Ho
me/bin/java --enable-preview -XX:+ShowCodeDetailsInExceptionMessag
es -cp "/Users/brandonfain/Library/Application Support/Code/User/w
orkspaceStorage/033d2eb2075ca69abdef5f502aacb942/redhat.java/jdt_w
s/vscodeTest_901392fd/bin" Hello
Hello World
(base) brandonfain@Brandons-MacBook-Air vscodeTest %
```

Basic anatomy of a Java program

- Each Java source code file `<className>.java` contains at least `public className`.



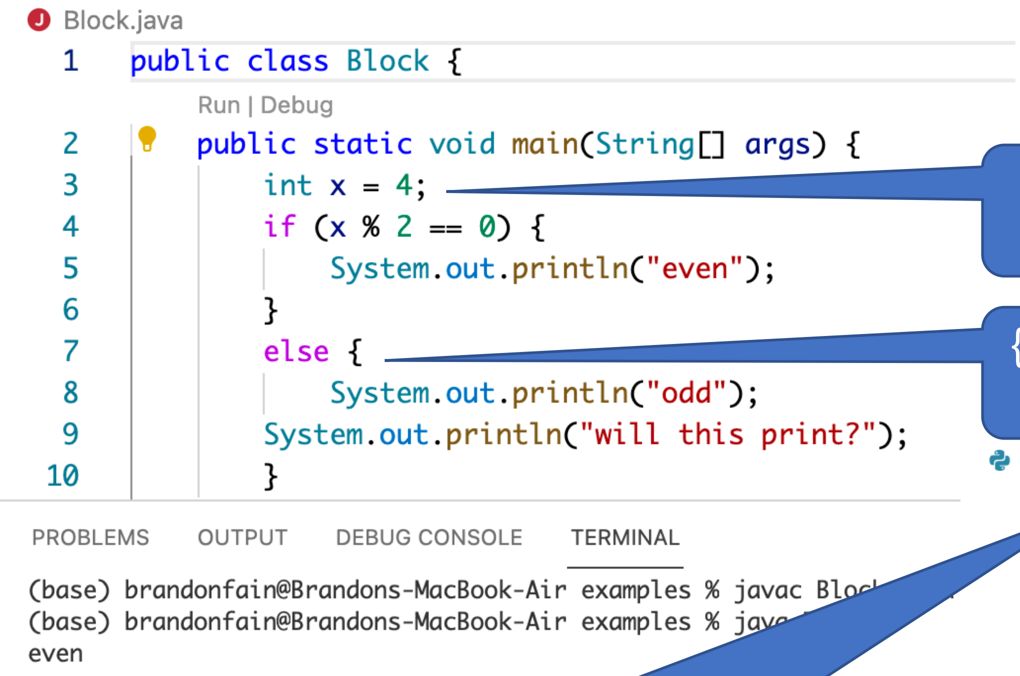
Hello.java ×

Hello.java

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

- 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.

Java uses { } to denote blocks and ; to end statements

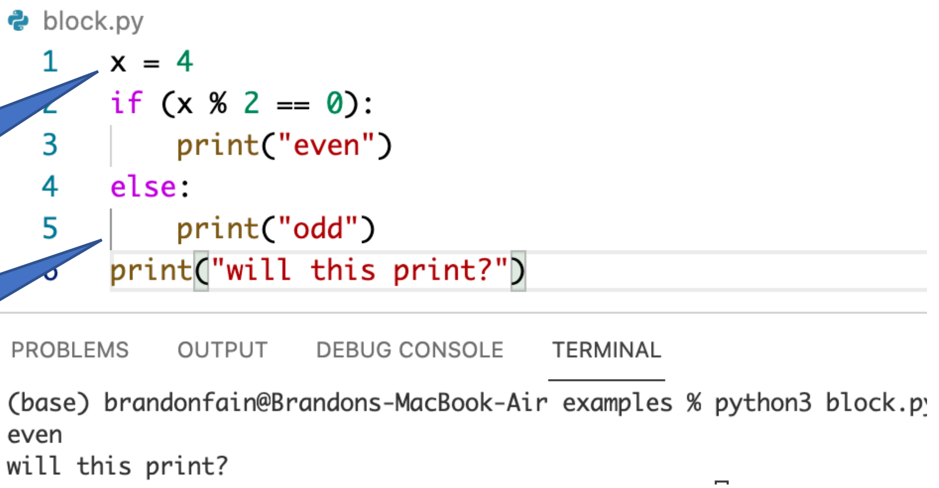


```java
public class Block {
    public static void main(String[] args) {
        int x = 4;
        if (x % 2 == 0) {
            System.out.println("even");
        }
        else {
            System.out.println("odd");
        System.out.println("will this print?");
        }
```

```
(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



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

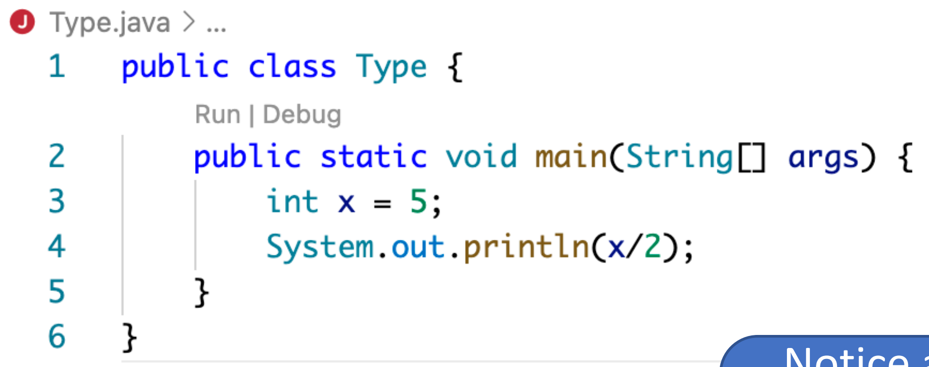
```
(base) brandonfain@Brandons-MacBook-Air examples % python3 block.p
even
will this print?
```

newline ends statement in Python

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

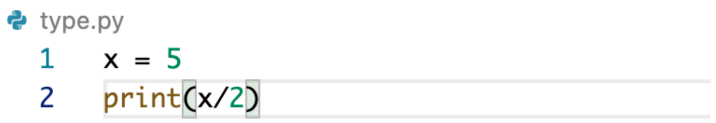
Java is **strongly typed**

Must be explicit about the **type** of every variable when declaring and in method signatures.



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

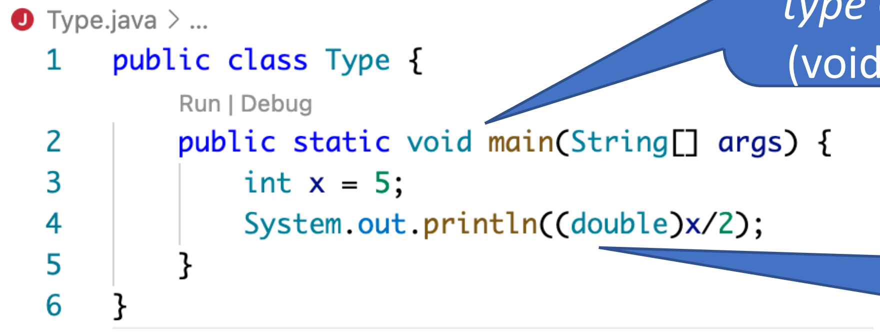
Prints 2



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

Prints 2.5

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

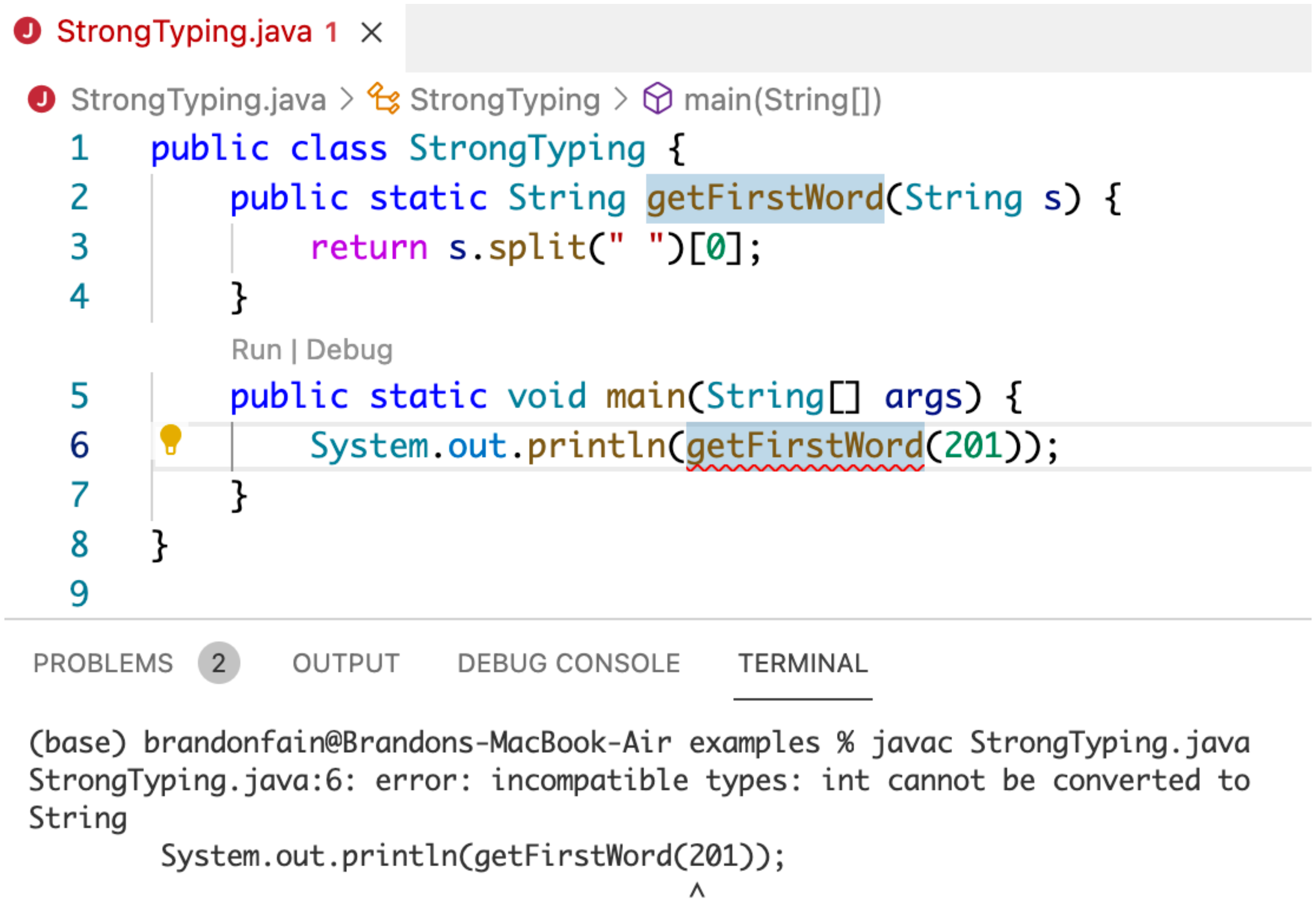


```java
public class Type {
    public static void main(String[] args) {
        int x = 5;
        System.out.println((double)x/2);
    }
}
```

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

Prints 2.5

Strong typing allows the compiler to help you avoid mistakes



```java
public class StrongTyping {
    public static String getFirstWord(String s) {
        return s.split(" ")[0];
    }
    public static void main(String[] args) {
        System.out.println(getFirstWord(201));
    }
}
```

```
(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));
                                        ^
```

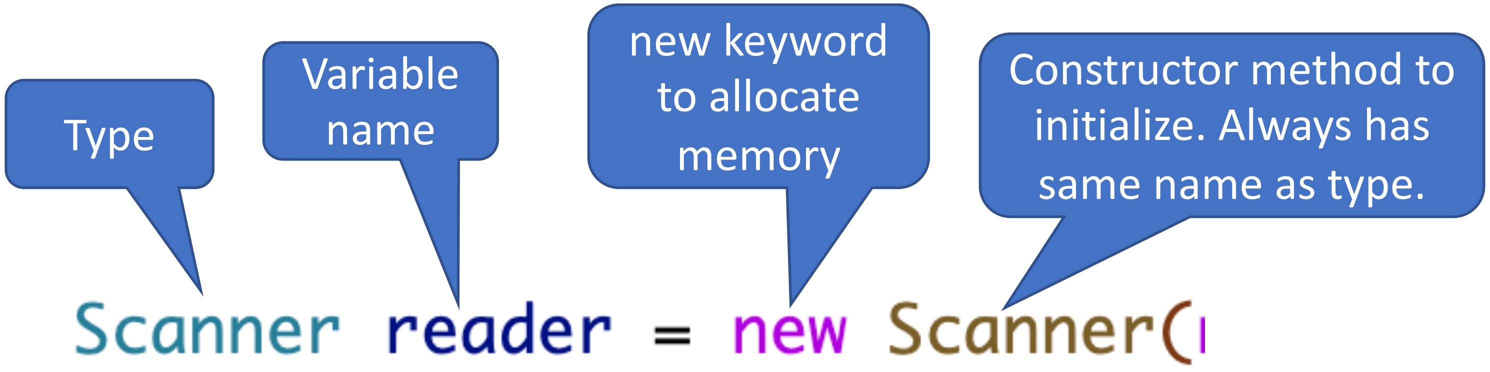
Java primitive types

- 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

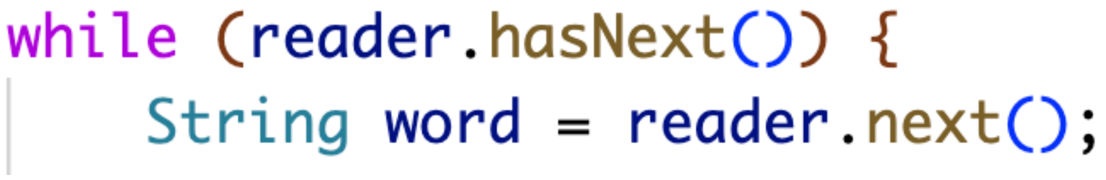
`+, -`	Add, subtract
`*, /`	Multiply, divide (careful with divide, 5/4 gives 1)
`%`	Modulus (remainder in int division, if % 2 == 0 then even, if % 2 == 1 then odd)
`<, <=`	Less than, less than or equal to
`>, >=`	Greater than, greater than or equal to
`==`	Equal (only for primitive types!!!)
`!`	Logical 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)

Java reference types


Type
Variable name
new keyword to allocate memory
Constructor method to initialize. Always has same name as type.

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

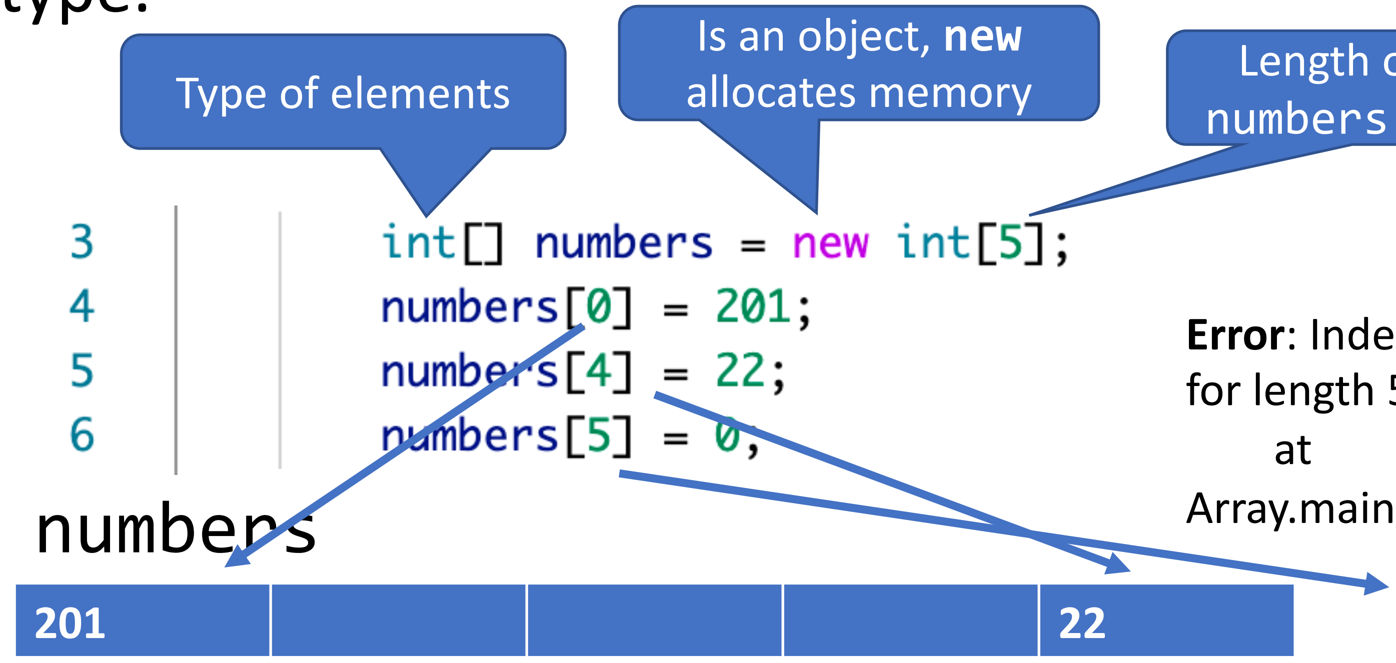
- Variable stores a **reference** to an **object**, i.e., a place in memory.
- Can access instance variables and method calls with the **dot operator**.



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

Java arrays

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



Type of elements
Is an object, new allocates memory
Length of array, numbers.length
numbers
201
22

```
3        int[] numbers = new int[5];
4        numbers[0] = 201;
5        numbers[4] = 22;
6        numbers[5] = 0;
```

Error: Index 5 out of bounds for length 5
    at
Array.main(Array.java:6)

Shorthand for pre-initialized Array: `int[] myArray = {1, 2, 3};`

Special Case: String

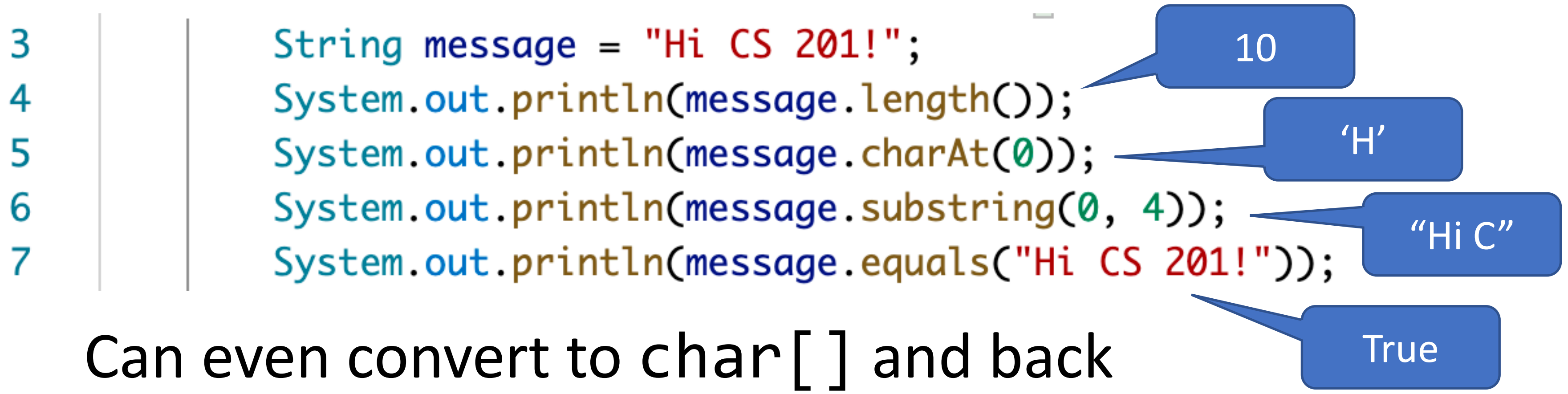
- NOT primitive, but can initialize in two ways:
  - `String s = “Hello”;` or `String s = new String(“Hello”);`

- + is overloaded to concatenate Strings:
  - `String s = “Hello”;`
  - `String t = “ World”;`
  - `System.out.println(s + t);` prints “Hello World”

Java Strings: concepts and methods

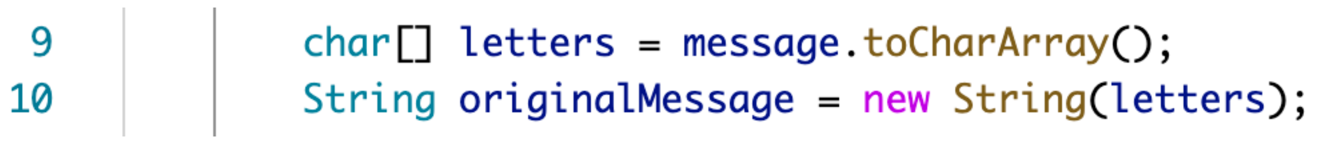
Strings are objects that hold an array of characters.

H	i		C	S		2	0	1	!
0	1	2	3	4	5	6	7	8	9



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

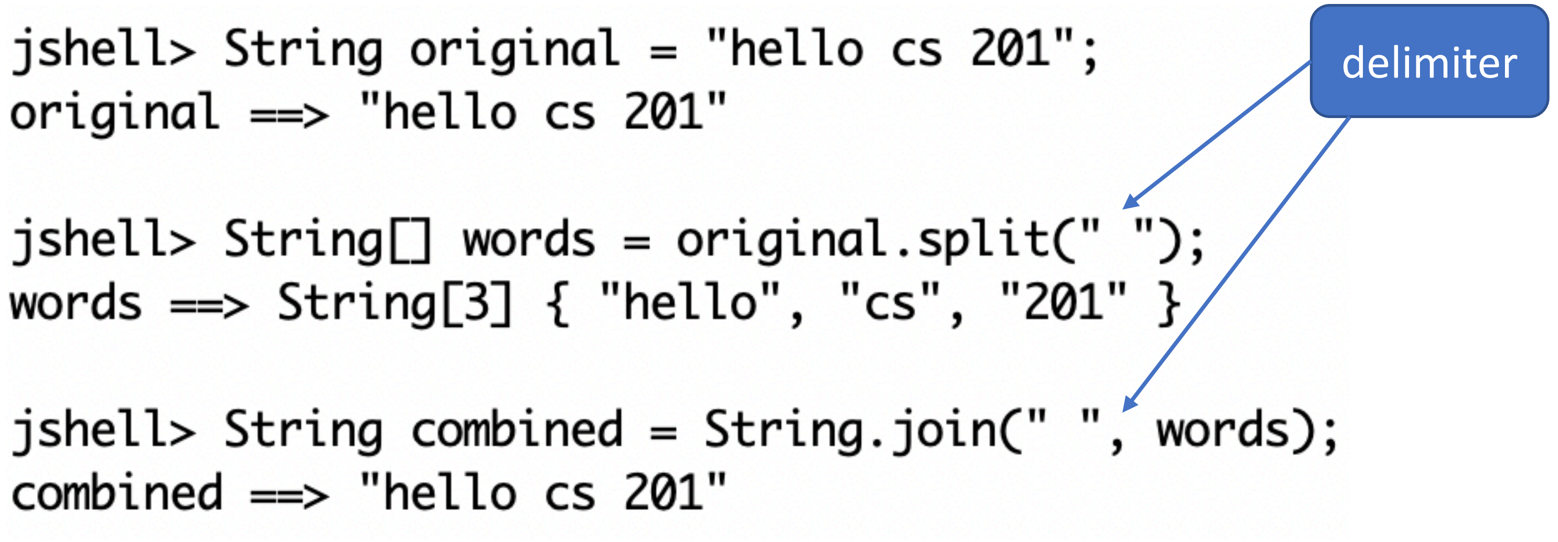
Can even convert to char[ ] and back



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

More String methods: `split` and `join`

Can `split` a String into an array of Strings or `join` an array of Strings to one String.



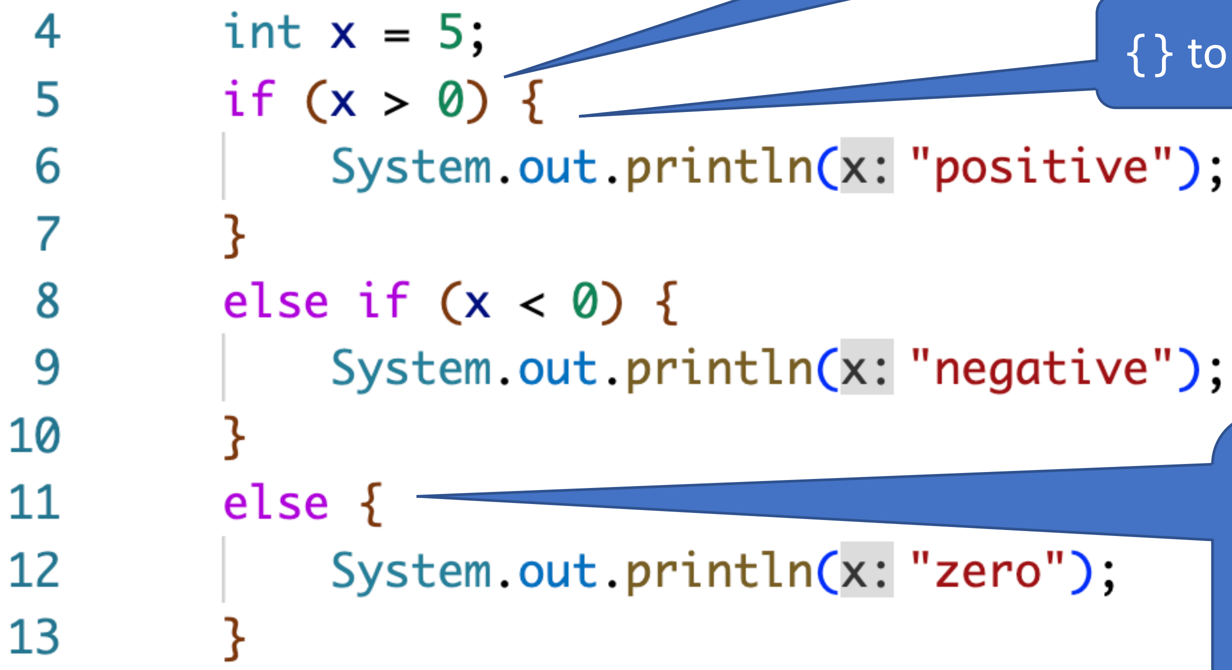
```
jshell> String original = "hello cs 201";
original ==> "hello cs 201"

jshell> String[] words = original.split(" ");
words ==> String[3] { "hello", "cs", "201" }

jshell> String combined = String.join(" ", words);
combined ==> "hello cs 201"
```

See the full [String documentation here]

Java conditionals

Condition must be in parentheses

{} to enclose block



```java
int x = 5;
if (x > 0) {
    System.out.println(x: "positive");
}
else if (x < 0) {
    System.out.println(x: "negative");
}
else {
    System.out.println(x: "zero");
}
```

Else statements optional, can chain else if else if … else.

Java loops

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

Loop while i < numbers.length

Regular for

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

Increase i by 1 each time through loop

Enhanced for

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

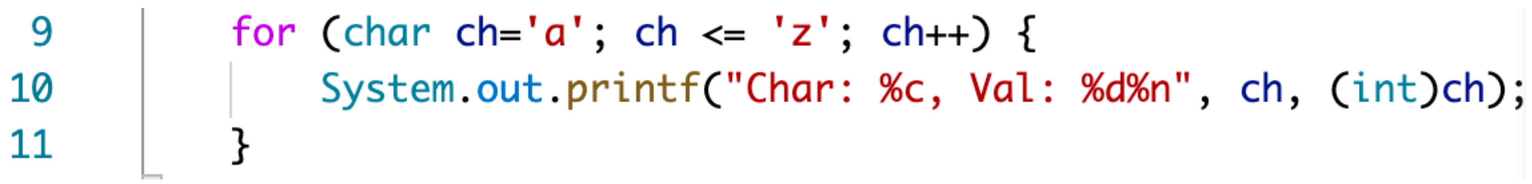
number takes each value in numbers in turn

while

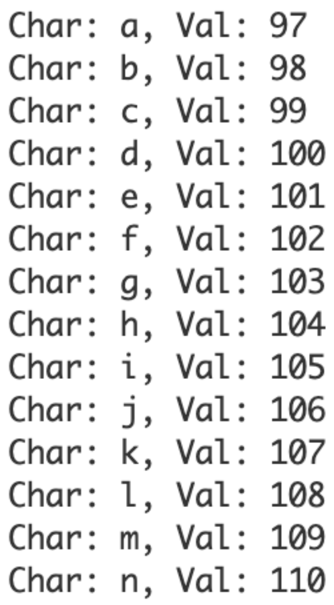
```
int i=0;
while (i < numbers.length) {
    System.out.println(numbers[i]);
    i++;
}
```

Note on Java characters

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



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



```
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

WOTO

Go to duke.is/gwcs5

Not graded for correctness, just participation.

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

But do talk to your neighbors!

2

Which of the following best describes how a Java program runs? *

- ○ Line by line, the first line gets translated and executed, then the second...
- [✓ The whole program is translated to bytecode that has to be run in a Java virtual machine]
- ○ The whole program is compiled into 0s and 1s then the machine runs that directly
- ○ Magic

3

In java, one generally needs to use the `new` keyword... *

- ○ When creating a copy
- ○ When creating any variable
- ○ When creating a data structure
- [✓ When creating a reference type variable]

4

Which of the following for loops correctly prints the sum of all the elements in an int[] called **values**? Select all that apply. *

[✓ for (int i : values) { sum += i; }]

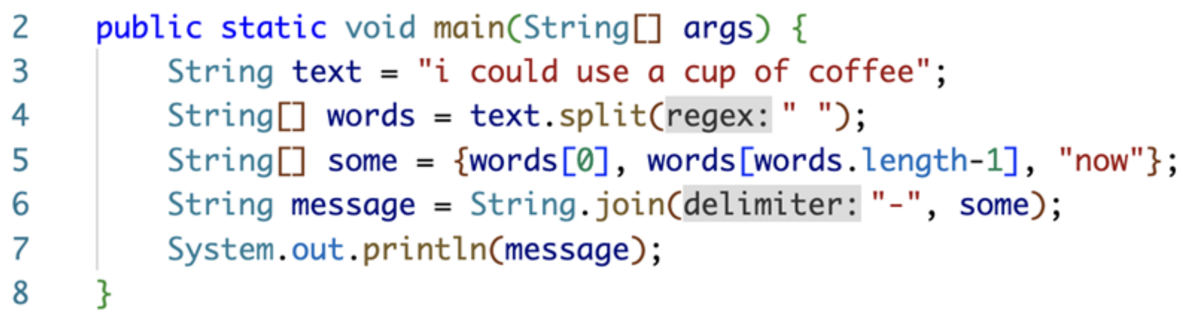
for (int i : values) { sum += values[i]; }

for (int i=0; i<values.length; i++) { sum += i; }

[✓ for (int i=0; i<values.length; i++) { sum += values[i]; }]

5

What will be printed by the following java program? *

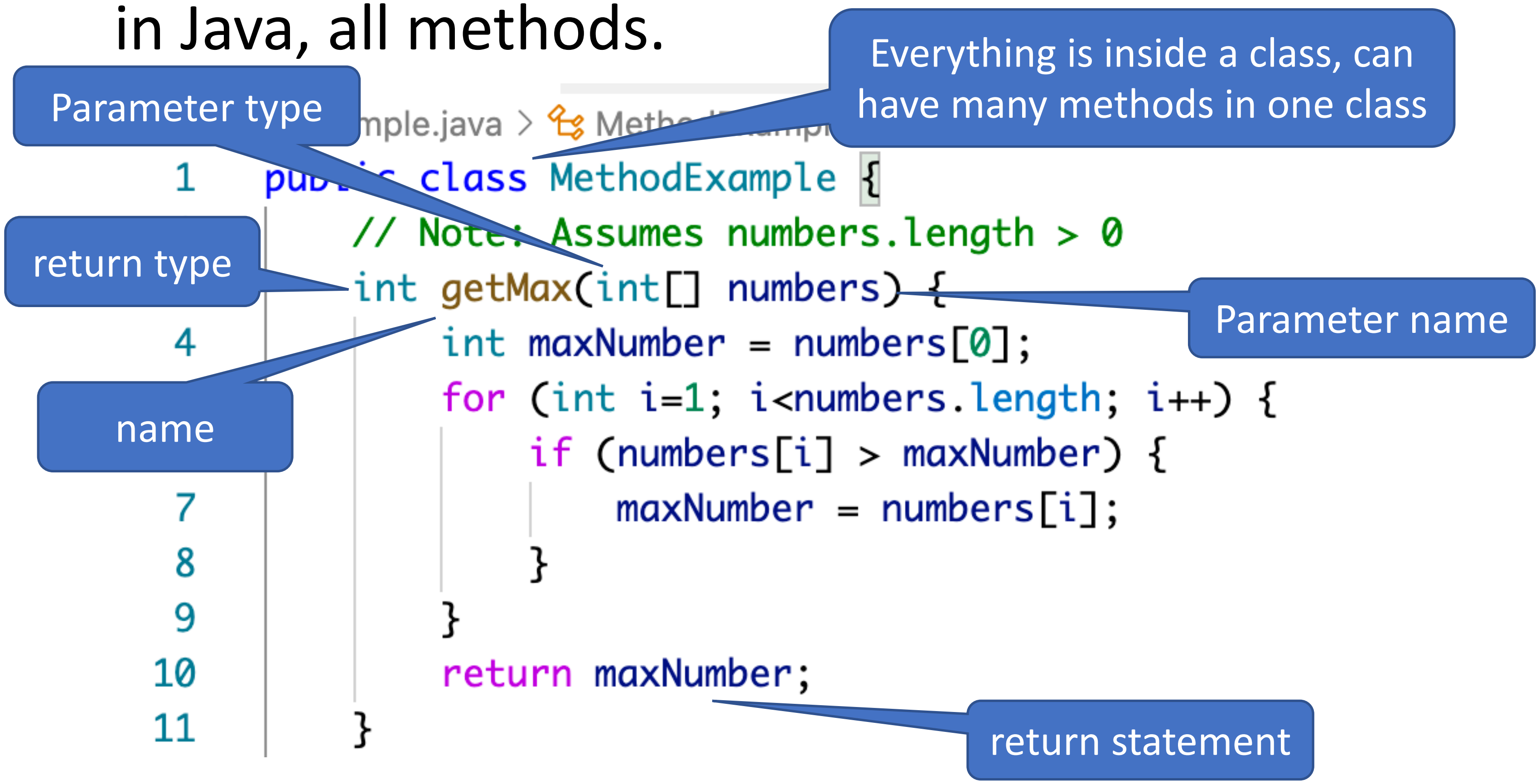


```java
public static void main(String[] args) {
    String text = "i could use a cup of coffee";
    String[] words = text.split(regex: " ");
    String[] some = {words[0], words[words.length-1], "now"};
    String message = String.join(delimiter: "-", some);
    System.out.println(message);
}
```

i-coffee-now

Anatomy of Java methods

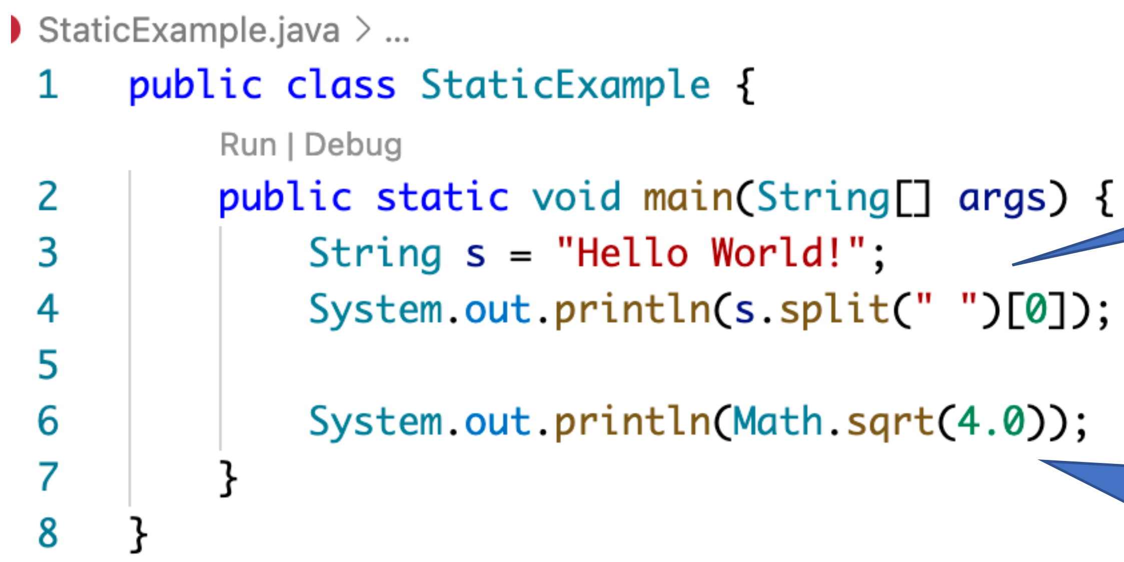
A function defined in a class. No “regular” functions in Java, all methods.


Parameter type
Everything is inside a class, can have many methods in one class
return type
Parameter name
name
return statement
MethodExample.java > MethodExample

```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;
    }
```

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 > ...

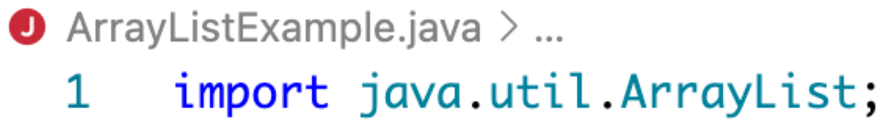
```java
public class StaticExample {
    Run | Debug
    public static void main(String[] args) {
        String s = "Hello World!";
        System.out.println(s.split(" ")[0]);

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

Note that `split` is called on a String object

Whereas `sqrt` is called on the Math class

Anatomy of a Java collections data structure

- An import statement:



ArrayListExample.java > ...

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

  - Goes outside the class, top of the file

```java
ArrayList<Integer> list = new ArrayList<>();
```

Collections type

Element type

Variable name

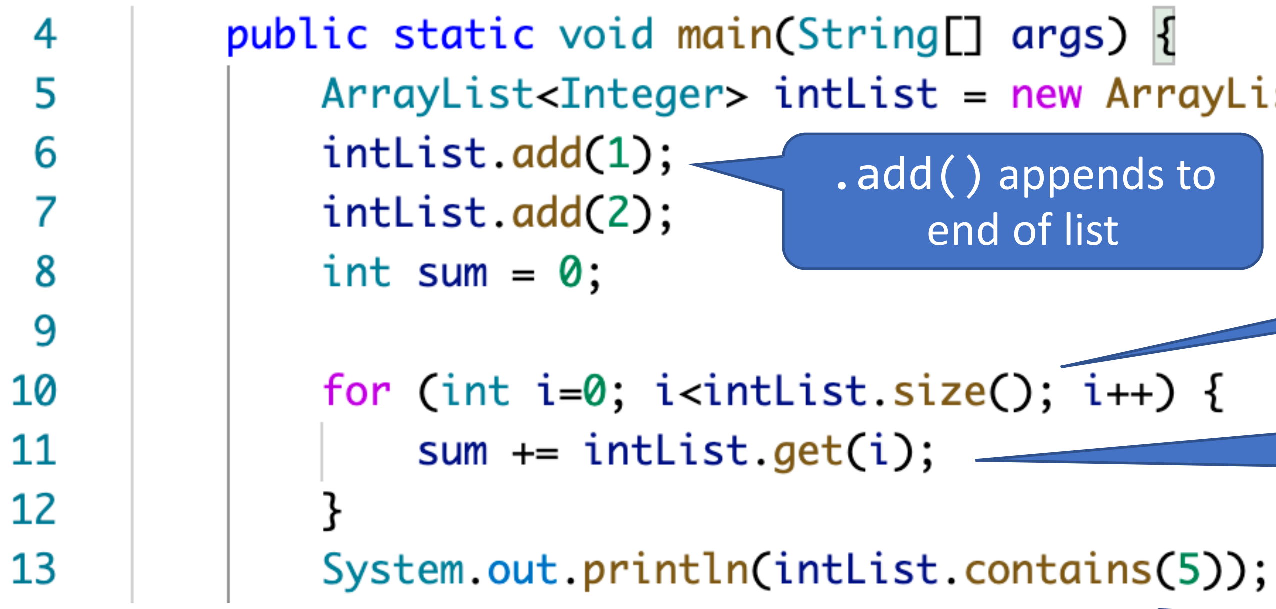
Allocate memory

Call constructor method to initialize

Java API ArrayList data structure

`ArrayList` is most like a Python list.

- Access by index access but can grow dynamically
- Uses `add()`, `get()`, `size()`, `contains()`



```java
    public static void main(String[] args) {
        ArrayList<Integer> intList = new ArrayList<>();
        intList.add(1);
        intList.add(2);
        int sum = 0;

        for (int i=0; i<intList.size(); i++) {
            sum += intList.get(i);
        }
        System.out.println(intList.contains(5));
```

`.add()` appends to end of list

`.size()` returns number of elements

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

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

ArrayList methods reference

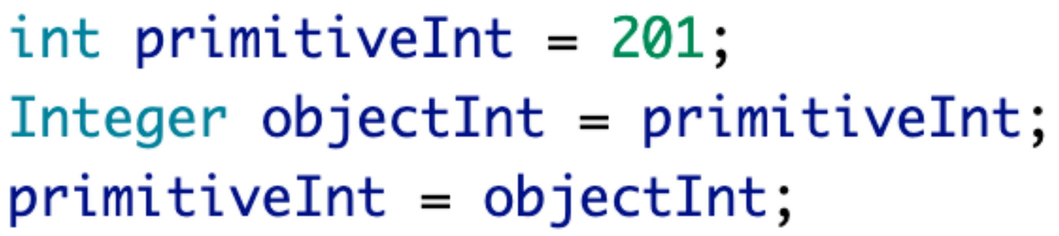
Method	Notes
`add(element)`	Appends `element` to end of list
`get(index)`	Returns the `index` position element (starting with 0)
`contains(element)`	Searches list, returns `true` if `element` is in the list, else `false`.
`size()`	Returns the (integer) number of elements in the list
`set(index, element)`	Assigns `element` to the `index` position (starting at 0), overwriting the previous value.
`remove(index)`	Remove the `index` position element

See the full ArrayList documentation

Java API Collections and Primitive vs. object types

Why `ArrayList<Integer>` ... instead of `ArrayList<int>`...?

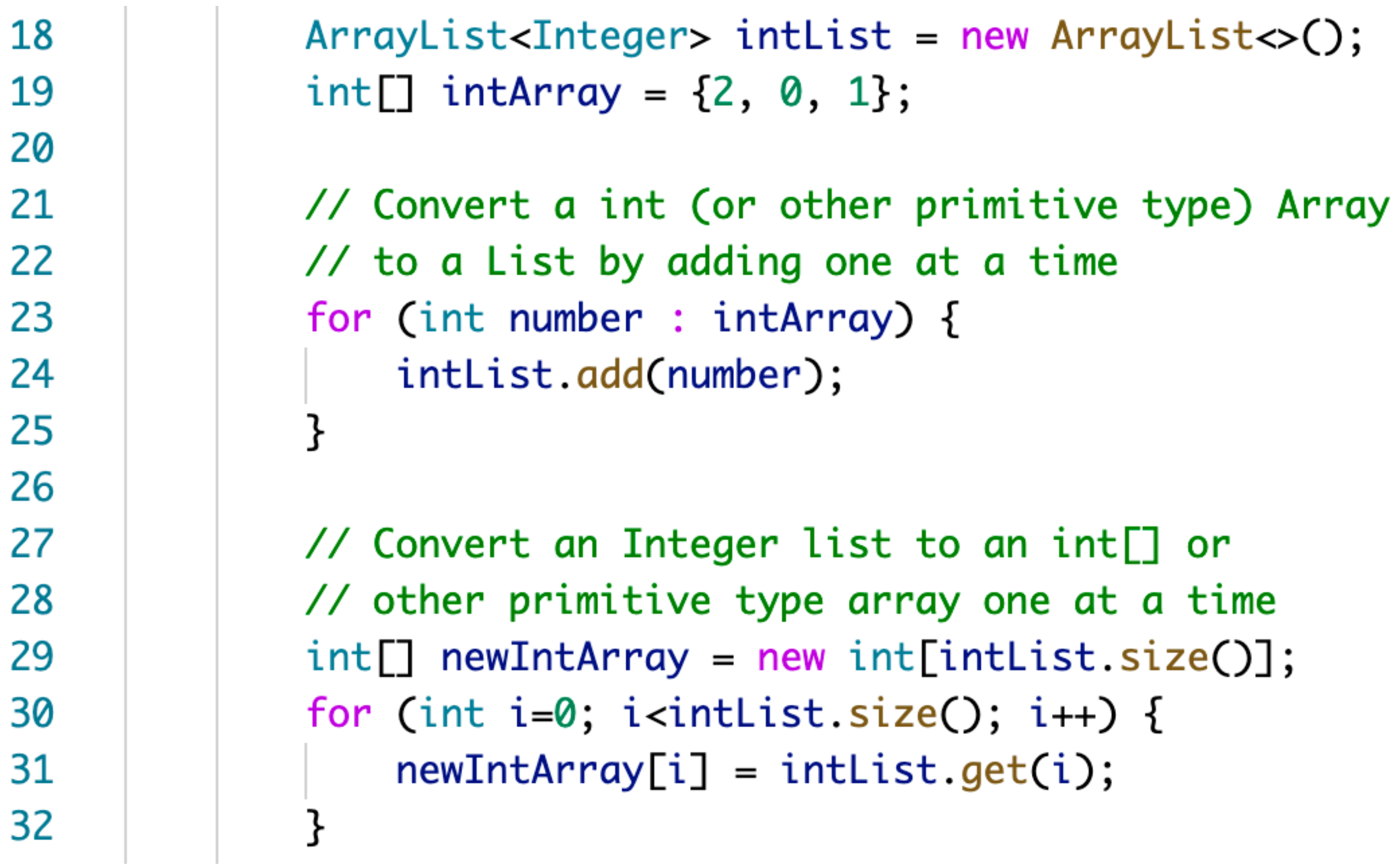
- Java API Collections (ArrayList, HashSet, ...) only store *reference types*, not primitive types.
- `Integer` is an int object, can convert back and forth "automatically."



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

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

Converting Arrays and ArrayLists

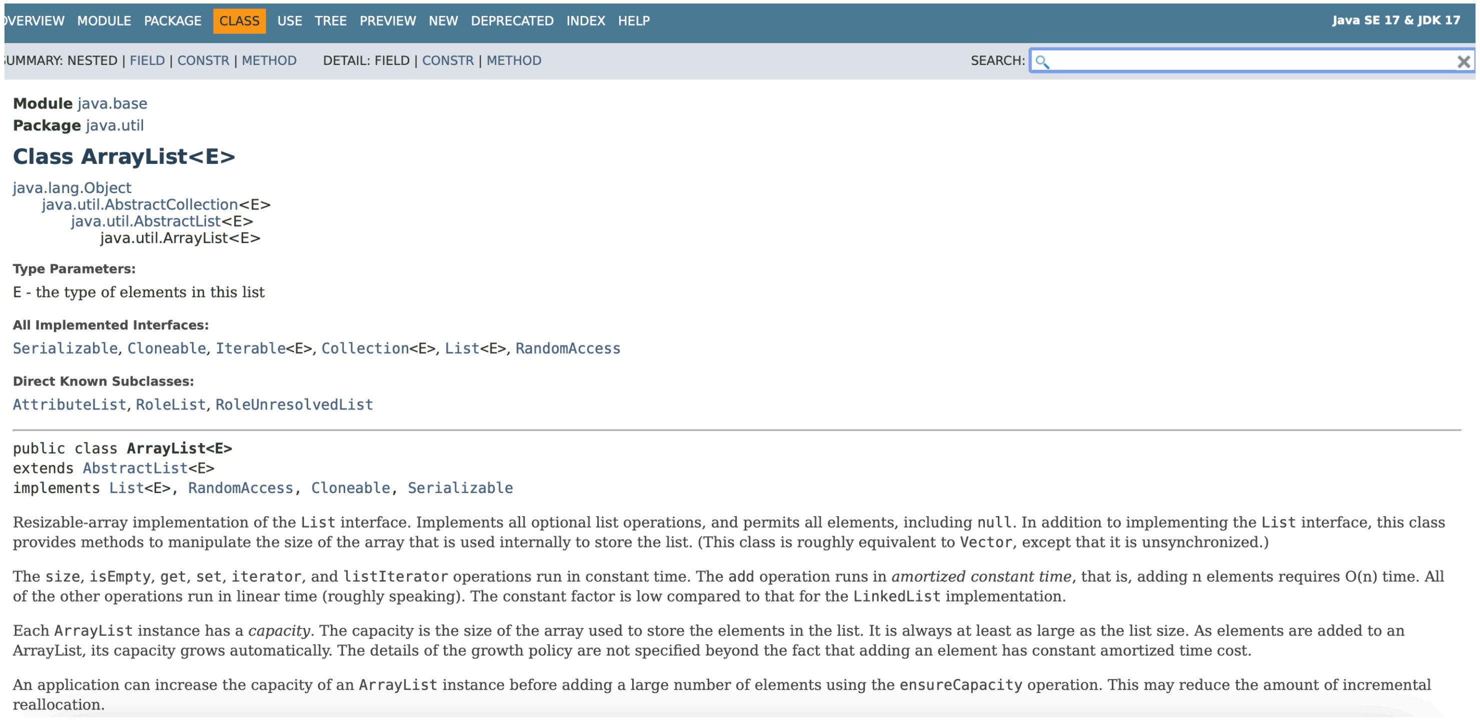


```java
ArrayList<Integer> intList = new ArrayList<>();
int[] intArray = {2, 0, 1};

// 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);
}
```

API Documentation

Reading documentation is an important skill:

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


VERVIEW 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.)

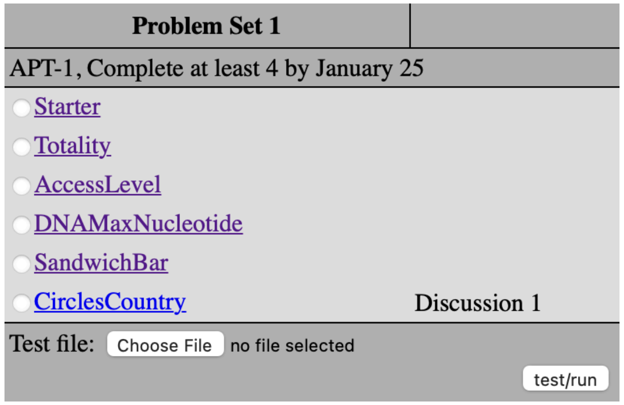
The `size`, `isEmpty`, `get`, `set`, `iterator`, and `listIterator` operations run in constant time. The add operation runs in *amortized constant time*, that is, adding n elements requires O(n) time. All of the other operations run in linear time (roughly speaking). The constant factor is low compared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

First programming problems

- APT 1 due next Wednesday 1/25, access server from course website schedule.



Problem Set 1

APT-1, Complete at least 4 by January 25

- Starter
- Totality
- AccessLevel
- DNAMaxNucleotide
- SandwichBar
- CirclesCountry — Discussion 1

Test file: Choose File no file selected

test/run

- Write 1 method per problem, ~10-30ish lines of code.
- No static methods allowed, no main needed.
- Automatic testing, submit as many times as needed.
- See walkthrough video of submitting an APT

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
- Java4Python resource on website
- First Discussions, first sets of APTs, Projects P0 and P1 designed to help practice
- Peers, Ed discussion, helper hours, all can help