L2: Intro to Java

Alex Steiger

CompSci 201: Spring 2024

1/17/24

1

Logistics, Coming up

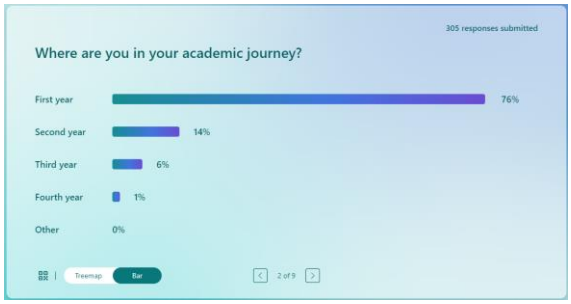
- This Friday, 1/19
  - First discussion section meetings
- Next Monday, 1/22
  - Intro to OOP (object-oriented programming) in Java
- Next Wednesday 1/24
  - Interfaces, Implementations, ArrayList data structure
  - First APT set (short programming exercises) due
    - Can discuss with peers, but **code must be your own.** Policies page

2

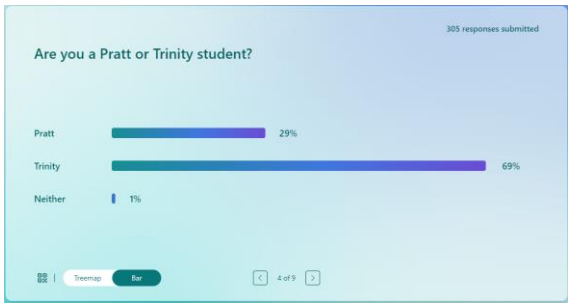
Helper Hours

- **What:** Drop-in time to ask TAs questions about course content (Concepts, Java, APTs, Projects).
- **When:** Sunday-Thursdays
- **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.

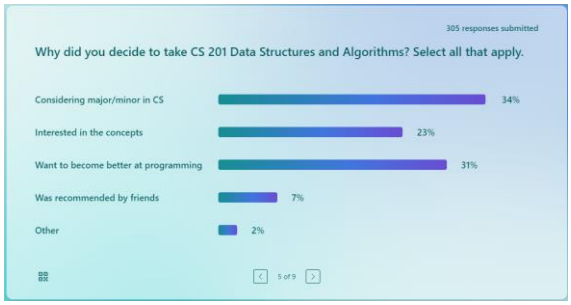
3


305 responses submitted
Where are you in your academic journey?
First year
76%
Second year
14%
Third year
6%
Fourth year
1%
Other
0%
Treemap
Bar
2 of 9

4

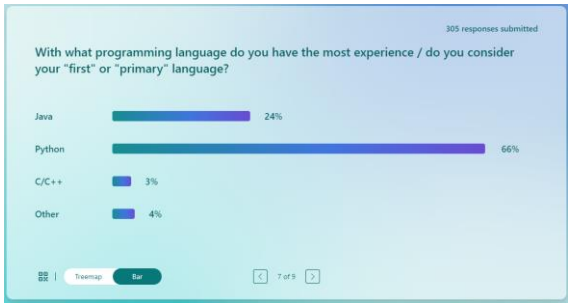

305 responses submitted
Are you a Pratt or Trinity student?
Pratt
29%
Trinity
69%
Neither
1%
Treemap
Bar
4 of 9

5


305 responses submitted
Why did you decide to take CS 201 Data Structures and Algorithms? Select all that apply.
Considering major/minor in CS
34%
Interested in the concepts
23%
Want to become better at programming
31%
Was recommended by friends
7%
Other
2%
5 of 9

6

7



8

Goals for 201?

- Become proficient in Java / coding
- Improve problem-solving skills
- Learn real-world applications to other fields
- Learn to better communicate and collaborate
- Decide if want to pursue/major in CS
- Build a foundation for more CS classes

9

Fred Brooks, why is programming fun?

- Duke '53
- Founded CompSci @ UNC '64
- Turing award '99

1. Sheer joy of making things.
2. Pleasure of making things that are useful.
3. Fascination of fashioning complex puzzle-like objects of interlocking moving parts.
4. Joy of always learning.
5. Delight in working in such a tractable medium.

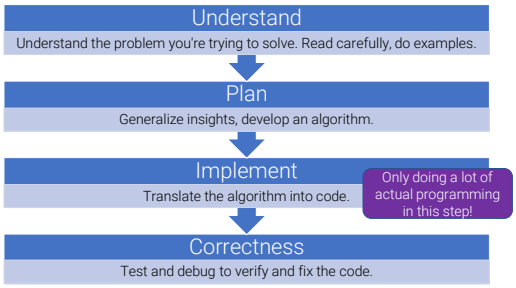
10

Fred Brooks, cont.

- …Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures…
- …[Programming] is fun because it gratifies creative longings built deep within us and delights sensibilities [we all have in common.]

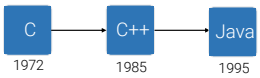
11

An Algorithmic Problem-Solving Process: UPIC



12

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.

13

Java is a compiled language

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

Compiled

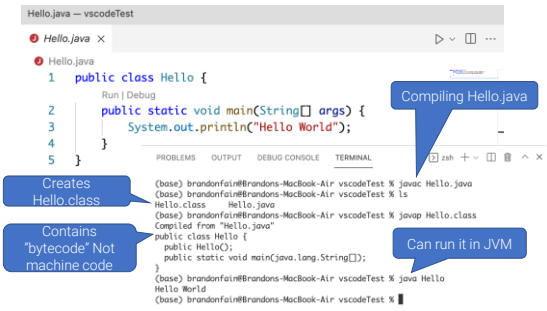
- All at once
- Compiler is another program that translates source code into machine code.
- Run the *executable*, the output of the compiler.

Interpreted

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

14

The "Java Virtual Machine"



15

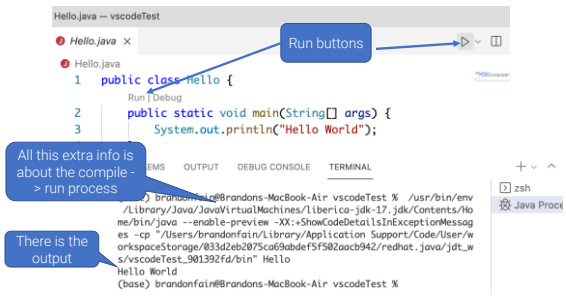
Interlude: Compile and Run Java

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

16

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



17

Basic anatomy of a Java program

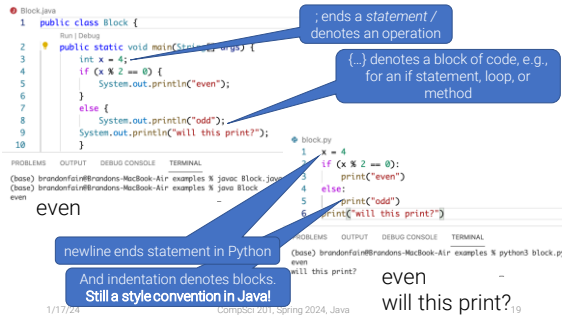
- Each Java source code file `<className>.java` contains at least `public className`.

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

- 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.

18

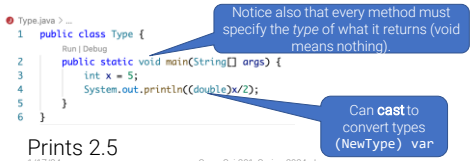
Java uses { } to denote blocks and ; to end statements



19

Java is **strongly typed**

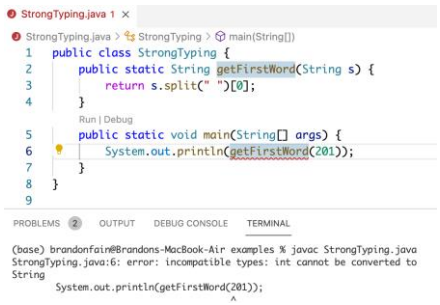
Must be explicit about the **type** of every variable.



Prints 2.5

20

Strong typing allows the compiler to help you avoid mistakes



21

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'`)

22

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 at **a, b, or both** need to be true)

23

Java reference types

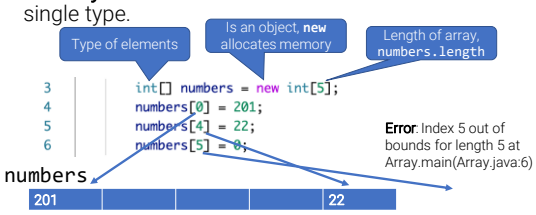
- Variable stores a ***reference*** to an ***object***, i.e., a place in memory.
- Can access instance variables and method calls with the ***dot operator***.

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


24

Java arrays

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



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

25

Special Case: String

- **NOT** primitive, but can initialize in two ways:
  - `String s = "Hello";`
  - `String s = new String("Hello");`
- `+` is *overloaded* to concatenate Strings:
  - `String s = "Hello";`
  - `String t = " World";`
  - `System.out.println(s + t);` prints "Hello World"
- **NOT** an array, but can access i-th char:
  - `char c = t.charAt(1);`
  - `System.out.println(c);` prints "W"

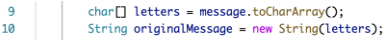
26

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

Can even convert to `char[]` and back



27

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"
```

delimiter

See the full String documentation here

28

Java conditionals

Condition must be in parentheses

{} to enclose block

```
4   int x = 5;
5   if (x > 0) {
6       System.out.println(x: "positive");
7   }
8   else if (x < 0) {
9       System.out.println(x: "negative");
10  }
11  else {
12      System.out.println(x: "zero");
13  }
```

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

29

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 sequence

Regular `for`

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

Enhanced `for`, "for-each" loop

```
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      }
```


30

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 (ASCII)

31

WOTO

Not graded for correctness, just participation.

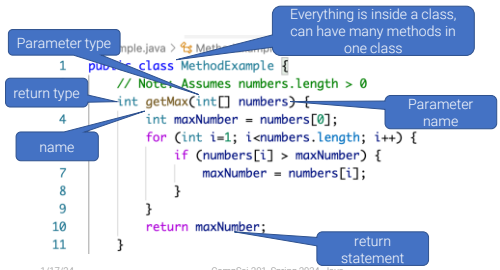
Try to answer *without* looking back at slides and notes.

But do talk to your neighbors!

32

Anatomy of Java methods

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



34

Static vs. Non-static Methods

- Non-static 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

35

Anatomy of a Java collections data structure

- An import statement:

ArrayListExample.java > ...

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

  - Goes outside the class, top of the file

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

Collections type | Element type | Variable name | Allocate memory | Call constructor method to initialize

36

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()`

```
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

`.size()` returns number of elements

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

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

37

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

38

Live Coding

48