L2: Intro to Java

Alex Steiger
CompSci 201: Spring 2024
1/17/24
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
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.
Where are you in your academic journey?

First year: 76%
Second year: 14%
Third year: 6%
Fourth year: 1%
Other: 0%
Are you a Pratt or Trinity student?

Pratt: 29%
Trinity: 69%
Neither: 1%
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%

305 responses submitted
What programming course did you last take?

CompSci 101 at Duke: 43%
EGR 103 at Duke: 9%
Online course: 3%
Course in high school (e.g. AP CS): 35%
No formal course: 4%
With what programming language do you have the most experience / do you consider your “first” or “primary” language?

- Java: 24%
- Python: 66%
- C/C++: 3%
- Other: 4%
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
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.
5. Delight in working in such a tractable medium.
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.]
An Algorithmic Problem-Solving Process: UPIC

**Understand**
Understand the problem you're trying to solve. Read carefully, do examples.

**Plan**
Generalize insights, develop an algorithm.

**Implement**
Translate the algorithm into code. Only doing a lot of actual programming in this step!

**Correctness**
Test and debug to verify and fix the code.
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”

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

- **Compiling Hello.java**
- **Creates Hello.class**
- **Contains “bytecode” Not machine code**
- **Can run it in JVM**
## Interlude: Compile and Run Java

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
<th>Details</th>
</tr>
</thead>
</table>
| 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:

1. The `public class` and `public static void main` statements are executed to create a new class and a main method.
2. The `System.out.println("Hello World");` statement is executed to print the message "Hello World" to the console.

All this extra info is about the compile -> run process. There is the output:

```
Hello World
(base) brandonfain@Brandons-MacBook-Air vscodeTest % /usr/bin/env
/Library/Java/JavaVirtualMachines/liberica-jdk-17.jdk/Contents/Home/bin/java --enable-preview -XX:+ShowCodeDetailsInExceptionMessages -cp "/Users/brandonfain/Library/Application Support/Code/User/workspaceStorage/03d2eb2075ca69abdef5f502aacb942/redhat.java/jdt_ws/vscodeTest_901392fd/bin" Hello
(base) brandonfain@Brandons-MacBook-Air vscodeTest %
```
Basic anatomy of a Java program

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

• To run a program, must have a `public static void main(String[] args)` 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?");
    }
}
```

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

```
OUTPUT
(base) brandonfain@Brandons-MacBook-Air examples % javac Block.java
(base) brandonfain@Brandons-MacBook-Air examples % java Block
even
```

```bash
(base) brandonfain@Brandons-MacBook-Air examples % python3 block.py
```

 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.

```
Type.java > ...
1 public class Type {
2     public static void main(String[] args) {
3         int x = 5;
4         System.out.println(x/2);
5     }
6 }
```

```
Type.java > ...
1 public class Type {
2     public static void main(String[] args) {
3         int x = 5;
4         System.out.println((double)x/2);
5     }
6 }
```

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

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’)

CompSci 201, Spring 2024, Java
## Java basic operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+, -</td>
<td>Add, subtract</td>
</tr>
<tr>
<td>*, /</td>
<td>Multiply, divide (careful with divide, (5/4) gives 1)</td>
</tr>
<tr>
<td>%</td>
<td>Modulus (remainder in int division, if (% \ 2 == 0) then even, if (% \ 2 == 1) then odd)</td>
</tr>
<tr>
<td>&lt;, &lt;=</td>
<td>Less than, less than or equal to</td>
</tr>
<tr>
<td>&gt;, &gt;=</td>
<td>Greater than, greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>Equal (only for primitive types!!!)</td>
</tr>
<tr>
<td>!</td>
<td>Logical NOT (!a means a must not be true)</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Logical AND (a &amp;&amp; b means a and b need to be true)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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.

```java
Scanner reader = new Scanner();

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

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

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

**Type of elements**

**Is an object, new allocates memory**

**Length of array, numbers.length**

**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”;
  • 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”
Java Strings: concepts and methods

Strings are objects that hold an array of characters.

<table>
<thead>
<tr>
<th>H</th>
<th>i</th>
<th>C</th>
<th>S</th>
<th>2</th>
<th>0</th>
<th>1</th>
<th>!</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

```
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!"));
```

Can even convert to `char[]` and back

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

True
More String methods: **split** and **join**

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

```java
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](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html)
Java conditionals

```
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    }
```
Java loops

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 | }

Loop while 
\[ i < \text{numbers.length} \]

Increase \( i \) by 1 each time through loop

number takes each value in \( \text{numbers} \) in sequence
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);
    }
```

Values are how characters are *encoded* on a machine (ASCII)
WOTO

Not graded for correctness, just participation.

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

But do talk to your neighbors!
Anatomy of Java methods

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

```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. 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.”

```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
Anatomy of a Java collections data structure

- An import statement:
  ```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

<table>
<thead>
<tr>
<th>Method</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add(element)</code></td>
<td>Appends <code>element</code> to end of list</td>
</tr>
<tr>
<td><code>get(index)</code></td>
<td>Returns the <code>index</code> position <code>element</code> (starting with 0)</td>
</tr>
<tr>
<td><code>contains(element)</code></td>
<td>Searches list, returns <code>true</code> if <code>element</code> is in the list, else <code>false</code>.</td>
</tr>
<tr>
<td><code>size()</code></td>
<td>Returns the (integer) number of elements in the list</td>
</tr>
<tr>
<td><code>set(index, element)</code></td>
<td>Assigns <code>element</code> to the <code>index</code> position (starting at 0), overwriting the previous value.</td>
</tr>
<tr>
<td><code>remove(index)</code></td>
<td>Remove the <code>index</code> position <code>element</code></td>
</tr>
</tbody>
</table>

See the full [ArrayList documentation](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html)
Live Coding