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.
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

1. Understand
   Understand the problem you’re trying to solve. Read carefully, do examples.

2. Plan
   Generalize insights, develop an algorithm.

3. Implement
   Translate the algorithm into code.

4. 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 --> source

Hello.java

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

Compiling Hello.java

```
Compiles Hello.class

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

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 (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 is **strongly typed**

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

Strong typing allows the compiler to help you avoid mistakes.
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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+</code>, <code>-</code></td>
<td>Add, subtract</td>
</tr>
<tr>
<td><code>*</code>, <code>/</code></td>
<td>Multiply, divide (careful with divide, 5/4 gives 1)</td>
</tr>
<tr>
<td><code>%</code></td>
<td>Modulus (remainder in int division, if % 2 == 0 then even, if % 2 == 1 then odd)</td>
</tr>
<tr>
<td><code>&lt;</code>, <code>&lt;=</code></td>
<td>Less than, less than or equal to</td>
</tr>
<tr>
<td><code>&gt;</code>, <code>&gt;=</code></td>
<td>Greater than, greater than or equal to</td>
</tr>
<tr>
<td><code>==</code></td>
<td>Equal (only for primitive types!!!)</td>
</tr>
<tr>
<td><code>!</code></td>
<td>Logical NOT (a means a must not be true)</td>
</tr>
<tr>
<td><code>&amp;&amp;</code></td>
<td>Logical AND (a &amp;&amp; b means both 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] = 20;
numbers[4] = 22;
```

**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:
  ```java
  String s = "Hello";
  String s = new String("Hello");
  ```

- `+` is **overloaded** to concatenate Strings:
  ```java
  String s = "Hello";
  String t = " World";
  System.out.println(s + t); // prints "Hello World"
  ```

- **NOT** an array, but can access i-th char:
  ```java
  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>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>i</td>
<td>C</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```java
String message = "Hi CS 201!";
System.out.println(message.length()); // prints 10
System.out.println(message.charAt(0)); // prints "H"
System.out.println(message.substring(0, 4)); // prints "Hi C"
System.out.println(message.equals("Hi CS 201!")); // prints true
```

Can even convert to `char[]` and back

```java
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.

```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](#).

---

Java conditionals

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

Condition must be in parentheses

{} to enclose block

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

---

Java loops

Regular for

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

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

Enhanced for, "for-each" loop

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

while

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

number takes each value in numbers in sequence
Note on Java characters

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

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

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.
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
import java.util.ArrayList;

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

Anatomy of a Java collections data structure

- An import statement: `import java.util.ArrayList;` - Goes outside the class, top of the file

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

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()`
## ArrayList methods reference

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

See the full ArrayList documentation.

---

### Live Coding

![Live Coding Image]