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

- Based on an analysis of Github repositories.

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

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

Compiling Hello.java

Can run it in JVM

Creates Hello.class

Contains “bytecode” Not machine code
### Interlude: Command Line?

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pwd</code></td>
<td>Print Working Directory</td>
<td>Shows the full file path to the directory you are currently in</td>
</tr>
<tr>
<td><code>ls</code></td>
<td>List Files</td>
<td>Shows all files and directories contained in the current directory</td>
</tr>
<tr>
<td><code>cd</code></td>
<td>Change Directory</td>
<td>• <code>cd</code> by itself goes to your home directory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>cd directory</code> goes to the specified directory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>cd ..</code> goes to the enclosing directory</td>
</tr>
<tr>
<td><code>mkdir</code></td>
<td>Make Directory</td>
<td>• <code>mkdir directory</code> creates a directory</td>
</tr>
<tr>
<td><code>cp</code></td>
<td>Copy</td>
<td><code>cp source target</code> Copies the source file and names the result <code>target</code></td>
</tr>
<tr>
<td><code>rm</code></td>
<td>Remove</td>
<td><code>rm file</code> deletes the specified file. No backups!!!</td>
</tr>
</tbody>
</table>
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](https://docs.oracle.com/en/java/javase/11/docs/api/java.compiler/package-summary.html) 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");
    }
}
```

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

There is the output:
```bash
Hello World
```

(You can also compile and run from the command line)

```
(brandofain@Brandons-MacBook-Air vscodeTest % /usr/bin/env /Library/Java/JavaVirtualMachines/liberica-jdk-17.jdk/Contents/Home/bin/java --enable-preview -XX:+ShowCodeDetailsInExceptionMessages -cp "/Users/brandofain/Library/Application Support/Code/User/workspaceStorage/033d2eb2075ca69abdef5f502aabc942/redhat.java/jdt_ws/vscodeTest_901392fd/bin" Hello)
Hello World
(brandofain@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?");
    }
}
```

```
block.py
1 x = 4
2 if (x % 2 == 0):
3     print("even")
4 else:
5     print("odd")
6 print("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.

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

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

Prints 2.5

```
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

<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
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

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>201</td>
<td>22</td>
<td></td>
</tr>
</tbody>
</table>

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

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

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>i</td>
<td>C</td>
<td>S</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Can even convert to char[] and back

```java
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"));
```

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

```java
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    }
```

- Condition must be in parentheses
- `{}` to enclose block
- Else statements optional, can chain else if else if ... else.
Java loops

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      }

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
Note on Java characters

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

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

- [ ] Line by line, the first line gets translated and executed, then the second...
- [x] 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

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

- [ ] When creating a copy
- [ ] When creating any variable
- [x] When creating a reference type variable
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]; }

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.

```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;
    }
}
```

Everything is inside a class, can have many methods in one class.

- **Parameter type**
- **Return type**
- **Parameter name**
- **Return statement**
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.”

```java
StaticExample.java  ➔ ...
1  public class StaticExample {
2      public static void main(String[] args) {
3          String s = "Hello World!";
4          System.out.println(s.split(" ")[0]);
5      }
6  }
7
8 }
```

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<>();
```
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>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](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/ArrayList.html)
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;
```
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);
}
```
Reading documentation is an important skill:

docs.oracle.com/en/java/javase/17/docs/api
First programming problems

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

• 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