CompSci 201, L2
Intro to Java
Course Staff Re-Intros

• Instructor: Brandon Fain
• Teaching Associate: Kate O'Hanlon
• Graduate Teaching Assistants:
  • Zhekun Wu
  • Yilun Song
• Undergraduate Teaching Assistants: Many! Will add pictures and names to website getting help page.
Logistics, Coming up

• This Friday, 9/2
  • First Discussion Section

• Next Monday, 9/5
  • OOP (object-oriented programming) in Java

• Next Wednesday 9/7
  • Interfaces, Implementations, ArrayList
  • First APT Exercises Due
Data Collection

Go to duke.is/zeqkj

Collecting a dataset for Project 0

Info you enter will be in the dataset for everyone to see, don’t enter anything you consider private or sensitive.
Algorithmic Problem Solving: Before Coding...

1. Understand the problem carefully
2. Work examples (small, by hand)
3. Gather insights (generalize examples?)
4. Make a plan (e.g., outline of algorithm)

Modeled by the upcoming CirclesCountry APT walkthrough in Discussion 1.
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 — vscodeTest

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

Compiling Hello.java

Creates Hello.class

Contains "bytecode" Not machine code

Can run it in JVM
## 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`. |
Pressing the “run” button in VS Code does these steps for you:

1. Compile the code.
2. Execute the compiled code.

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

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

• OOPs, we need some vocabulary:
  • Class, Object, object, fields, methods, constructor, mutators, accessors, instance variables, static, ...

• But first the basics:
  • Each Java program file contains a single class
  • The file is named <className>.java
  • 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 is strongly typed

Must be explicit about the type of every variable.

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

Prints 2

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

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
import java.util.StringTokenizer;

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>
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 Reference Types

• Need to use `new` and call constructor to create.

```java
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`.

```java
while (reader.hasNext()) {
    String word = reader.next();
}
```
Java uses {} and ; to denote blocks and 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?");
    }
}
```

; ends a statement /
/ denotes an operation

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

newline ends statement in Python

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

**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)
Java Strings: Concepts and Methods

Strings are objects that hold an array of characters.

H i C S 2 0 1 !

3  String  message  =  "Hi CS 201!";  10
4  System.out.println(message.length());
5  System.out.println(message.charAt(0));  ‘H’
6  System.out.println(message.substring(0, 4));  “Hi C”
7  System.out.println(message.equals("Hi CS 201!"));  True

Can even convert to char[] and back

9  char[]  letters  =  message.toCharArray();
10  String  originalMessage  =  new String(letters);
Note on Java characters

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

Values are how characters are *encoded* on a machine.
Java loops

Regular for

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

Enhanced for

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

while

```java
int i=0;
while (i < numbers.length) {
    System.out.println(numbers[i]);
    i++;
}
```
Go to duke.is/j7me4

Not graded for correctness, just participation. Try to answer without looking back at slides and notes.

Java methods

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

```
1 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."

```java
StaticExample.java > ...
1  public class StaticExample {
2   
3   public static void main(String[] args) {
4     String s = "Hello World!";
5     System.out.println(s.split(" ")[0]);
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.
Java API ArrayList

An import statement:

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

More on ArrayList next time, but the basics:

- Generic to specify type, 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));
}
```
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.”

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

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

An import statement:  
```java
import java.util.HashSet;
```

More on HashSet later, but the basics:

- Generic to specify type, does not store duplicates
- Uses add(), size(), contains()

```java
public static void main (String[] args) {
    HashSet<String> strSet = new HashSet<>();
    strSet.add("Hello");
    strSet.add("World");
    strSet.add("Hello");

    if(strSet.contains("World")) {
        System.out.println(strSet.size());
    }
}
```

Prints 2, no duplicates
API Documentation

Reading documentation is an important skill:

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

**Class ArrayList&lt;E&gt;**

java.lang.Object
java.util.AbstractCollection&lt;E&gt;
java.util.AbstractList&lt;E&gt;
java.util.ArrayList&lt;E&gt;

Type Parameters:
E - the type of elements in this list

All Implemented Interfaces:
Serializable, Cloneable, Iterable&lt;E&gt;, Collection&lt;E&gt;, List&lt;E&gt;, RandomAccess

Direct Known Subclasses:
AttributeList, RoleList, RoleUnresolvedList

public class ArrayList&lt;E&gt;
extends AbstractList&lt;E&gt;
implements List&lt;E&gt;, 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.
WOTO

Go to duke.is/mem3h

Not graded for correctness, just participation.

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

But do talk to your neighbors!
Comments on Java Style

Code blocks:

• Opening { ends first line of if, for, while, or method
• Indent every line inside the block
• Closing } on a separate line, last of block, not indented

Variable & method names:

• One-word names: lowercase
• Multi-word names: camelCase
• Should be informative
More comments on Java style

Class names:
• Capitalized & CamelCase
• MUST match name of .java file!

Comments:
• // for one line
• /* ... */ for multiple lines
PSVM: Public Static Void Main

Method that is:
• public – can call outside of class
• static – belongs to class, not an object
• void – no return
• main – starting point for a program to run

```java
1  public class MainExample {
2      public static void main(String[] args) {
3          for (String s : args) {
4              System.out.println(s);
5          }
6      }
7  }
```

`args` allows for command-line arguments

```sh
$javac MainExample.java
$java MainExample Hello World!
Hello World!
$
Do I need PSVM for APTs?

• No, generally writing a single regular method.
• PSVM method is supplied by the APT server that runs your code.
• Can write a PSVM to test your code locally, but...
• APT Server WILL NOT allow you to have static methods when you submit
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
• First Discussions, first sets of APTs, Projects P0 and P1 designed to help practice
• Peers, Ed discussion, Office hours, all can help
Fred Brooks, Why is programming fun?

- Duke ‘53
- Founded Compsci @ UNC
- Turing award winner, design
1. Sheer joy of making things
2. Pleasure of making things that are useful
3. Fascination of fashioning complex puzzle-like objects
4. Delight in working in such a tractable medium [like a poet]