CSCI 134 Fall 2021:

Introduction to Java

Dec 1, 2021

Shikha Singh, 9AM
Jeannie Albrecht, 10AM

Announcements & Logistics

• **Lab 9 Boggle**: Parts 3 & 4 due tonight/tomorrow @ 10pm
  • Lots of office hours: 1-5:30 pm today, 1-4:30 pm tomorrow
  • Come talk to us if you have questions!

• **HW 9** available today, due Mon 12/6 @ 10pm
  • Covers a lot of “advanced” topics from recent lectures
  • Good discussion topics for herd meetings

• **Lab 10 Selection Sort in Java** (next Mon/Tue)
  • No pre-lab work or video; hope most of you will start and finish during your lab session

• **Final exam reminder**: Dec 18 @ 9:30 am

• Course evals next Friday 12/10 (bring a laptop to class if possible)

Do You Have Any Questions?
Last Time

- Briefly reviewed searching algorithms:
  - $O(\log n)$: binary search runtime in a sorted array-based list
  - $O(n)$: linear searching runtime in an unsorted list
- Discussed two classic sorting algorithms:
  - $O(n \log n)$: merge sort runtime
  - $O(n^2)$: selection sort runtime
- What about (extra) space for sorting:
  - $O(n)$: naive merge sort
  - $O(1)$: selection sort
- Time-space tradeoff!
Today

• Begin discussing **Java**
  • Discuss how to **run programs in Java**
  • Learn about Java **syntax**
  • Take a closer look at **data types** in Java

• Goals of next 4 lectures:
  • Understand the key similarities and differences between Python and other programming languages (Java)
  • **Review basic features of Python** in preparation for final exam
  • Gain confidence in our programming abilities
  • Help ease the transition to CS 136 (and beyond!)
Python vs. Java

**Python**

- Nice language for beginner programmers
- Sparse and clear syntax
- Object-oriented with a consistent model for objects and variables
- Can write powerful, interesting programs without much “work”
- Informal, scripting language, can run interactively or as a script
- Dynamically/loosely typed

**Java**

- Better performance for (very large, complex) computations
- Maintainability (function of verbosity and the use of data types)
- Object-oriented with some primitive types
- Formal, industrial-strength language
- Must be compiled and run from terminal
- Statically/strongly typed
Hello, World!

Python in Week 1:

```python
1  print("Hello World")
```

```
bash-3.2$ python3 hello-simple.py
Hello World
```

Python in Week 11:

```python
def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

```
bash-3.2$ python3 hello.py
Hello, World!
```

Java:

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

```
bash-3.2$ javac Hello.java
bash-3.2$ java Hello
Hello, World!
```
Hello, World!

Python:
```python
def main():
    print("Hello, World!")
if __name__ == "__main__":
    main()
```
bash-3.2$ python3 hello.py
Hello, World!

Java:
```java
public class Hello {
    public static void main(String args[]) {
        System.out.println("Hello, World!");
    }
}
```
bash-3.2$ javac Hello.java
bash-3.2$ java Hello
Hello, World!
Hello, World!

Python:
```python
def main():
    print("Hello, World!")
if __name__ == "__main__":
    main()
```

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

bash-3.2$ python3 hello.py
Hello, World!
bash-3.2$ javac Hello.java
bash-3.2$ java Hello
Hello, World!
Running Our Code

• **Python** is an *interpreted* language
  • The Python *interpreter* runs through our code line by line and executes each command
  • Other interpreted languages: PHP, Ruby, and JavaScript

• **Java** is a *compiled* language*
  • The Java *compiler* converts our code into machine code that the processor can execute
  • Compiled languages required code to be **manually compiled** before execution
  • Other compiled languages: C, C++, Haskell, Rust, and Go

• Interpreted languages were once significantly slower than compiled languages. But that gap is shrinking.

*Technically Java is both interpreted and compiled, but we can ignore that detail for now.*
Using the Java Compiler

- The compiler converts our Java source code into compiled byte code which is faster to run (hence the performance benefits).
- Java source files are always named `<file>.java`.
- To compile, type: `javac <file>.java`.
- Compilers detect and report syntax errors before execution.
- Compiler creates class files: `<file>.class`.
- Code is executed by typing `java <file>` (without the .class extension).

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

```
bash-3.2$ ls Hello.*
Hello.java
bash-3.2$ javac Hello.java
bash-3.2$ ls Hello.*
Hello.class Hello.java
bash-3.2$ java Hello
Hello, World!
```
Important Java Rules

• Every Java program must define a **class**, and all code is inside a class.

• The **file name** must be the same as the class name (**Hello.java**).

• Every object in Java must have an explicit **type**.

• Every Java program that we want to execute must have a main method: `public static void main(String[] args)`

• Blocks of code within `{}` (versus indentation in Python)

• Statements end with `;` (versus new line in Python)

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

- Every Java program must define a **class**, and all code is inside a class.
- The **file name** must be the same as the class name (**Hello.java**).
- Every object in Java must have an explicit **type**.
- Every Java program that we want to execute must have a main method: `public static void main(String[] args)`
- Blocks of code within `{}` (versus indentation in Python)
- Statements end with `;` (versus new line in Python)

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

Define a class called Hello. Notice the curly brace.

This curly brace closes the one on line 1.
Important Java Rules

- Every Java program must define a **class**, and all code is inside a class.
- The **file name** must be the same as the class name (**Hello.java**).
- Every object in Java must have an explicit **type**.
- Every Java program that we want to execute must have a main method: `public static void main(String[] args)`
- Blocks of code within `{}` (versus indentation in Python)
- Statements end with `;` (versus new line in Python)

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

- Defines the main method. Similar to saying if `__name__ == "__main__"` in Python.
- Opening curly brace
- Closing curly brace
Important Java Rules

• Every Java program must define a **class**, and all code is inside a class.
• The **file name** must be the same as the class name (**Hello.java**).
• Every object in Java must have an explicit **type**.
• Every Java program that we want to execute must have a main method: `public static void main(String[] args)`
• Blocks of code within `{}` (versus indentation in Python)
• Statements end with `;` (versus new line in Python)

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

Print "Hello, World!" to the terminal. Statements end with a ;
Public, Private, Protected

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

- **public** indicates to the Java compiler that this is a method that anyone can call
- Java enforces several levels of security on methods (also variables and classes): **public**, **protected**, and **private**
- Similar to _ and __ methods in Python, but more strictly enforced
static

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

- `static` indicates that this is a method that is part of the class, but is not a method for any one instance of the class (static exists in both Java and Python!)

- Most methods we used in Python required an instance of the class in order for the method to be called:
  - Example: `s.upper()` (where s is a string and `upper()` is a method in the string class)
  - With a static method, the object to the left of the . is a class, not an instance of the class.
  - For example the way that we would call the main method directly is: `Hello.main(...)`. 
  - Similar to Python modules (such as random) that don't require an instance
  - Example: `random.randint(0,15)`
void

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

- void tells the Java compiler that this method will not return a value
- void means “no type”
- Roughly analogous to omitting the return statement in a Python method (or having an implicit return)
Our main method takes as input an array (denoted by []) of Strings called args.

This is used for handling command-line arguments but we won't worry about that now.

Since everything in Java must have a type, we also have to tell the compiler that the types of values stored in our array are Strings.

Recall that arrays are a lot like lists in Python.
System.out and System.in

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

- **System** is a Java class
- Within the System class we find the object named **out**
- The **out** object is the **standard output stream** for this program. The **in** object is the **standard input stream**. We’ll come back to that soon.
- The **println** method prints a string with a newline character at the end
- Anywhere in Python that you used the `print(...)` function you will use the `System.out.println(...)` method in Java
Moving on...
Programming Language Features

• **Basic features:**
  • Data Types
  • Reading user input
  • Loops
  • Conditionals

• **Advanced topics:**
  • Classes
  • Interfaces
  • Collections
  • Graphical User Interface Programming

We have extensively studied all of these features in Python. Let's compare and contrast with Java.
Programming Language Features

• **Basic features:**
  - Data Types
  - Reading user input
  - Loops
  - Conditionals

• **Advanced topics:**
  - Classes
  - Interfaces
  - Collections
  - Graphical User Interface Programming

Let’s start with data types and reading user input.
Basic Data Types

• All **data types** in Python are **objects**
  • Implemented using **classes** and **methods** just like our LinkedList
• Two types of data types in Java: **primitive** (non-objects) and **Objects**
  • Example: **int** (lowercase) and **Integer** (uppercase)
  • The benefit of primitive data types is fast operations
  • We’ll mostly use the Object versions and let the compiler handle conversions to primitives for us

• Java data types:

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>float</td>
<td>Float</td>
</tr>
<tr>
<td>double</td>
<td>Double</td>
</tr>
<tr>
<td>char</td>
<td>Char</td>
</tr>
<tr>
<td>boolean</td>
<td>Boolean</td>
</tr>
</tbody>
</table>
Consider this Python script: `temp.py`

What does it do?
Simple Example

```python
1  def main ():
2      fahr = float(input("Enter the temperature in F: ") )
3      cel = (fahr - 32) * 5.0 / 9.0
4      print("The temperature in C is:", cel)

5       if __name__ == "__main__":
6             main()
```

- Consider this Python script: `temp.py`
- What does it do?
  - Asks user to enter a temperature in Fahrenheit and converts the string input to float
  - Does the computation to convert temperature to Celsius
  - Prints result
Simple Example

```java
// this is a comment in Java
import java.util.Scanner;

public class TempConv {
    public static void main (String args[]) {
        Double fahr;
        Double cel;
        Scanner in;

        in = new Scanner (System.in);
        System.out.print("Enter the temperature in F: ");
        fahr = in.nextDouble ();

        cel = (fahr - 32) * 5.0 / 9.0;
        System.out.println("The temperature in C is: " + cel);
    }
}
```

- Same program in Java: `TempConv.java`
Simple Example

```java
// this is a comment in Java
import java.util.Scanner;

public class TempConv {
    public static void main (String args[]) {
        Double fahr;
        Double cel;
        Scanner in;

        in = new Scanner (System.in);
        System.out.print("Enter the temperature in F: ");
        fahr = in.nextDouble ();

        cel = (fahr - 32) * 5.0 / 9.0;
        System.out.println("The temperature in C is: " + cel);
    }
}
```

• Same program in Java: TempConv.java
Simple Example

Java uses import statements to tell the compiler what classes to use.

Java import statements are similar to `from module import xxx` statements in Python.

```java
// this is a comment in Java
import java.util.Scanner;

public class TempConv {
    public static void main (String args[]) {
        Double fahr;
        Double cel;
        Scanner in;

        in = new Scanner (System.in);
        System.out.print("Enter the temperature in F: ");
        fahr = in.nextDouble ();

        cel = (fahr - 32) * 5.0 / 9.0;
        System.out.println("The temperature in C is: " + cel);
    }
}
```
Java is **statically typed**. Thus, all variables must be **declared** with a name and type before they are used. Common convention is to declare variables at the top of our methods/classes.

### Simple Example

```java
// this is a comment in Java
import java.util.Scanner;

public class TempConv {
    public static void main (String args[]) {
        Double fahr;
        Double cel;
        Scanner in;

        in = new Scanner (System.in);
        System.out.print("Enter the temperature in F: ");
        fahr = in.nextDouble ();

        cel = (fahr - 32) * 5.0 / 9.0;
        System.out.println("The temperature in C is: "+ cel);
    }
}
```

Lines 6-8 are **variable declarations**, which define the name and type of our variables. Once declared, the types cannot be changed.
Simple Example

```java
// this is a comment in Java
import java.util.Scanner;

public class TempConv {
    public static void main(String args[]) {

        Scanner in = new Scanner(System.in);
        System.out.print("Enter the temperature in F: ");
        double fahr = in.nextDouble();

        double cel = (fahr - 32) * 5.0 / 9.0;
        System.out.println("The temperature in C is: "+ cel);
    }
}
```

- Let's try to compile: `javac TempConv.java`
The compiler will report several errors (sometimes repeatedly) when we try to compile our program after removing our variable declarations.
Simple Example

```java
1 // this is a comment in Java
2 import java.util.Scanner;

3 public class TempConv {
4     public static void main(String[] args) {
5         Double fahr;
6         Double cel;
7         Scanner in;
8         in = new Scanner(System.in);
9         System.out.print("Enter the temperature in F: ");
10        fahr = in.nextDouble();
11        cel = (fahr - 32) * 5.0 / 9.0;
12        System.out.println("The temperature in C is: "+cel);
13     }
14 }
```

- After declaring a `Scanner` object named `in`, we also have to initialize it before using it (like calling `__init__()` in Python).
Simple Example

```java
// this is a comment in Java
import java.util.Scanner;

public class TempConv {
    public static void main(String[] args) {
        Double fahr;
        Double cel;
        Scanner in;

        in = new Scanner(System.in);
        System.out.print("Enter the temperature in F: ");
        fahr = in.nextDouble();

        cel = (fahr - 32) * 5.0 / 9.0;
        System.out.println("The temperature in C is: " + cel);
    }
}
```

- System.out.print and System.out.println are like print in Python.
- `in.nextDouble()` automatically reads the user input as a Double (like using `input()` in Python and then converting to `float`)
An Aside: Using the Java Scanner

- Since Java is **strongly typed**, we have to be extra careful when reading input from the user to make sure it is of the expected type.

- The **Scanner** class provides methods for making sure the next value (like an iterator!) is of the expected type.

- Here are some methods for the Java **Scanner** class:

<table>
<thead>
<tr>
<th>Method</th>
<th>Computes</th>
</tr>
</thead>
<tbody>
<tr>
<td>nextBoolean()</td>
<td>reads and converts next token to a boolean value</td>
</tr>
<tr>
<td>nextInt()</td>
<td>reads and converts next token to an integer value</td>
</tr>
<tr>
<td>nextLong()</td>
<td>reads and converts next token to a long value</td>
</tr>
<tr>
<td>nextDouble()</td>
<td>reads and converts next token to a double value</td>
</tr>
<tr>
<td>nextString() or next()</td>
<td>reads next token and returns it as a String</td>
</tr>
<tr>
<td>nextLine()</td>
<td>reads until the next new line and returns a String</td>
</tr>
<tr>
<td>hasNextBoolean()</td>
<td>returns true iff the next token is either “true” or “false”</td>
</tr>
<tr>
<td>hasNextInt()</td>
<td>returns true iff the next token is an integer</td>
</tr>
<tr>
<td>hasNextLong()</td>
<td>returns true iff the next token is a long</td>
</tr>
<tr>
<td>hasNextDouble()</td>
<td>returns true iff the next token is a real number</td>
</tr>
<tr>
<td>hasNextString() or hasNext()</td>
<td>returns true iff there is at least one more token of input</td>
</tr>
<tr>
<td>hasNextLine()</td>
<td>returns true iff there is another line of input</td>
</tr>
</tbody>
</table>
Simple Example

On Line 14 we perform the calculation to convert. On Line 15 we print the results.

```java
public class TempConv {
    public static void main (String args[]) {
        Double fahr;
        Double cel;
        Scanner in;

        in = new Scanner (System.in);
        System.out.print("Enter the temperature in F: ");
        fahr = in.nextDouble ();

        cel = (fahr - 32) * 5.0 / 9.0;
        System.out.println("The temperature in C is: "+ cel);
    }
}
```

- Arithmetic calculations in Java and Python are very similar wrt syntax
- When we print, we use the + operator to perform string concatenation
Simple Example

Before running our program, we compile using `javac`

```
bash-3.2$ javac TempConv.java
```

To run, we use `java`

```
bash-3.2$ java TempConv
Enter the temperature in F: 98.6
The temperature in C is: 37.0
bash-3.2$ java TempConv
Enter the temperature in F: 32
The temperature in C is: 0.0
```

• Before running our program, we compile using `javac`

  `javac TempConv.java`

• To run, we use `java`

  `java TempConv`
Recap: Python vs. Java

Java:

```java
in = new Scanner (System.in);
System.out.print("Enter the temperature in F: ");
fahr = in.nextDouble ();

cel = (fahr - 32) * 5.0 / 9.0;
System.out.println("The temperature in C is: "+ cel);
```

Python:

```python
fahr = float(input("Enter the temperature in F: "))
cel = (fahr - 32) * 5.0 / 9.0
print("The temperature in C is:", cel)
```

- Step 1: Prepare to read input from user.
Recap: Python vs. Java

Java:

```java
in = new Scanner(System.in);
System.out.print("Enter the temperature in F: ");
fahr = in.nextDouble();

cel = (fahr - 32) * 5.0 / 9.0;
System.out.println("The temperature in C is: "+ cel);
```

Python:

```python
fahr = float(input("Enter the temperature in F: "))
cel = (fahr - 32) * 5.0 / 9.0
print("The temperature in C is:", cel)
```

• Step 2: Prompt user for input.
Recap: Python vs. Java

Java:
```java
in = new Scanner (System.in);
System.out.print("Enter the temperature in F: ");
fahr = in.nextDouble () ;
cel = (fahr - 32) * 5.0 / 9.0;
System.out.println("The temperature in C is: " + cel);
```

Python:
```python
fahr = float(input("Enter the temperature in F: "))
cel = (fahr - 32) * 5.0 / 9.0
print("The temperature in C is:", cel)
```

- Step 3: Read user input and convert to float/double (that is, a number with a decimal point).
Recap: Python vs. Java

Java:
```java
in = new Scanner (System.in);
System.out.print("Enter the temperature in F: ");
fahr = in.nextDouble ();

cel = (fahr - 32) * 5.0 / 9.0;
System.out.println("The temperature in C is: " + cel);
```

Python:
```python
fahr = float(input("Enter the temperature in F: "))

cel = (fahr - 32) * 5.0 / 9.0
print ("The temperature in C is:", cel)
```

• Step 4: Perform conversion to Celsius.
Recap: Python vs. Java

- Step 5: Print result.

Java:

```java
in = new Scanner (System.in);
System.out.print("Enter the temperature in F: ");
fahr = in.nextDouble ();

cel = (fahr - 32) * 5.0 / 9.0;
System.out.println("The temperature in C is: " + cel);
```

Python:

```python
fahr = float(input( "Enter the temperature in F: ")

Cel = (fahr - 32) * 5.0 / 9.0

print("The temperature in C is:", cel)
```
An Aside: Java GUls

- Java has more built-in support for making GUls and supporting graphical objects
- Here is a graphical version of our program

```java
import javax.swing.*;

public class TempConvGUI {
    public static void main(String args[]) {
        Double fahr, cel;
        String fahrString;

        fahrString = JOptionPane.showInputDialog("Enter the temperature in F: ");
        fahr = Double.valueOf(fahrString);

        cel = (fahr - 32) * 5.0 / 9.0;
        JOptionPane.showMessageDialog(null, "The temperature in C is " + cel);
    }
}
```