

Introduction to Java

CPSC 2150

Introduction

- In this course we will be using Java for our programming assignments
- The course is designed with the assumption that you have no experience with Java
 - But you do have experience with C++ and C
- Java is very similar to C++
 - Variables, loops, if statements, etc. all exist in Java
 - Just some syntax differences
- In the real world, you will be asked to work in an unfamiliar language.
 - All of my professional experience has been in a language I did not learn in class
 - In school you learn how to program in C++ because most languages are similar to C++
- This course will be your first experience learning a programming language on your own.
- This guide will help you get started.

Getting Started

- One of the strengths of Java is that it can run on any machine
 - Windows, Mac, Linux, etc. The code is the same
- When you run a Java program, you are actually running the Java Virtual Machine
 - This allows for cross platform compatibility
- In this course we will often use the IntelliJ IDE (Interactive **D**evelopment **E**nvironment)
 - Far more advanced than a Unix program
 - Many helpful features
 - You even get you use your mouse!
 - Get started with IntelliJ:
 - <https://www.jetbrains.com/help/idea/installation-guide.html>
- You will still need to make sure your programs run on Unix for this course
 - This should not be an issue at all, since Java has that cross-platform compatibility

Creating a Java Program (on Unix)

- Create a file just like you would with C++ on Unix but use the `.java` file extension
 - **Ex.** `HelloWorld.java`
 - It is common to see Java programs named with uppercase letters
- To compile the program, use the `javac` command
 - **Ex.** `javac HelloWorld.java`
 - Like there is with the C++ compilers, there are many flags and options for the Java compiler. We'll cover those as we need to
- To run your program, use the `java` command
 - **Ex.** `java HelloWorld`
 - **Note:** now there is no file extension
- See the IntelliJ instructions on how to create and run the program in the IDE
 - Then you can transfer the code files to Unix and compile and run there before submitting your work

Import statements

- At the top of our code file, we have our *import* statements
 - This is how we can connect to other files and existing classes
 - The same concept as the `#include` statements in C and C++
- Common ones you'll see
 - `import java.util.*;`
 - `import java.io;`
 - These are useful for a lot of basic functionality, such as reading in input and printing output

```
import java.util.*;
```

```
import java.io;
```

```
public class MyFirstApp {
```

```
    public static void main(String[] args) {
```

```
...
```

Your `main` class

- One important difference between Java and C++ is that everything in Java is part of a class
 - In C++ we would have our main program be in a code file, but not part of a class.
 - This is not the case in Java.
 - In Java every variable, constant and method must be inside a class.
- Your program itself will be a class, with a `public static` method called `main`
 - Most of the classes you have created so far have been entity objects, i.e., they represent some real-world entity or concept
 - **Examples:** car, driver, student, etc.
 - Your class that contains your `main` method may not be an entity object, which makes naming it a little trickier.
 - It's common to name that class after the program itself, or to concatenate "App" at the end of the name to make it clear that this is the class that contains the `main` method
 - EX: `HelloWorld` or `HelloWorldApp`

Your `main` method

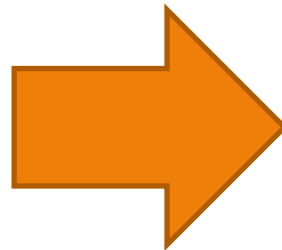
- Your `main` method will (in general):
 - Be inside some class
 - **Note:** every class can have a `main` method, even if they are part of the same program. You will just call one `main` method to start the program.
 - Be a `public static void` method
 - And have one argument, an array of strings
 - This will contain the command line arguments
 - In C++ and C we also had `argc` to let us know how many arguments we had. In Java we can call `args.length` to get that number
- **Note:** no semi-colon at the end of the class in java

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

Output in Java

- In C++ you defined a stream object and used the `<<` operator to print to the screen
- In Java we will use the `System.out` object and either `print()` or `println()` method
 - **Ex.** `System.out.println("Hello World");`
 - `println()` includes a new line symbol, while `print()` does not
 - Will accept strings, characters and numeric types.
- You can use the `+` operator to concatenate strings together
 - Remember, `String` is a defined class, so they overloaded the operator
 - As long as one operand is a `String`, then it will convert the other to a `String` first
 - **Ex:**

```
int x = 20;  
int y = 10;  
System.out.println("x: " + x);  
System.out.println("y: " + y);
```



```
x: 20  
y: 10
```

Input in Java

- First you will need to import `java.io.*` and `java.util.*`
 - Newer versions of Java don't require `java.io.*`
- Then create a `Scanner` object and pass `System.in` to the constructor
 - This tells it to read from the correct `InputStream`
- Now you can use the `nextLine()` method on your `Scanner` object to read in a line of input

- Returns a `String`

```
import java.io.*;  
import java.util.*;
```

```
public class MyConsoleIO {  
  
    public static void main(String[] args) {  
        System.out.println("Give me some input");  
        Scanner scanner = new Scanner(System.in);  
        String input = scanner.nextLine();  
    }  
}
```

Input in Java

- It can cause errors to have many `Scanner` objects reading from the same `InputStream` in the same program
- Often times programmers will have only one class that interacts with the keyboard
- In that class they will create a `private static Scanner` object as an attribute that can be used throughout the class, then define a `public` method to get the input.
- This also makes it easier to change how you interact with the user, but that conversation will come later
- You do not need to create a separate class for user input in this course, however it is generally beneficially to keep it all in the class that contains your `main` method

The end

- Well, the end of the basics
- View the next video on **Data Types** to continue learning about Java