Cody McMahon

TCSS 480

**INTRO**

In this paper I will be examining the differences between the code organization, basic syntax, and built-in types ANSII C 99 and Java 7. For the rest of this paper I will be referring to ANSII C 99 as “C” and Java 7 as “Java”. First there are some things I would like to go over about these languages; first of all they are both imperative languages, second of all Java is heavily reliant on classes while C does not have a direct built-in version of a class, third of all C has a preprocessor that does select amounts of processing before the final file is created(Ernst 2012) Java does not have an equivalent to this, and finally Java is ran on a virtual machine(Gupta 2014) while C is intended to be ran directly on the machine. Although none of these things directly have anything to do with the topic they greatly affect the syntax indirectly and I will be bringing this up a lot later. In a lot of ways these two languages are very similar but in even more ways they are entirely different.

**THE OVERALL OGANIZATION OF A PROGRAM**

It is very important to remember at this point that Java runs on a virtual machine and relies heavily on the constant uses of classes (Gupta 2014) while C is very different on both of these regards. In C, the compiler will search through all the code until it finds main and not look any further so every function that you wish to call during your program must have been declared above your main by either having all your functions above your main or by using prototypes. A prototype in C is basically a “this function will appear later on” declaration which looks like int num**(**int**);,** It is common to find these in in header files. A header file in C is an additional file to accompany your .c file and is labeled with .h, this file is not directly compiled and only preprocessed so you cannot have any declaration that requires “on the fly processing” such as x **=** x **+** 1. A normal C file ends up with a bunch of library includes constants, globals, and other goodies at the top of the file most of these can be stuck into a header file, then there is a list of functions with the “main” function usually at the bottom. There can only be one main function in an entire program any more will cause a compilation error.

The overall organization of a Java file is very different than this, although it does have all the library includes at the top of the file this is followed by the defining of a class. All java code must be contained into classes, inside of each class is usually organized with constants, globals, and fields, followed by a list of methods the position of the main method is entirely arbitrary but there can only be one per class. It is also possible to have classes inside of other classes these generally follow the organization as any other class. A Java program can have any number of main functions as long as there is only one per class and this is because each class is compiled separately and only one class is designated to run and the rest are loaded through the class loader(Gupta 2014).

**VERY BASIC SYNTAX AND KEYWORDS**

C and Java are actually very similar in this area here are two examples that prove my point.

The classic swap

int x **=** 5**;**

int y **=** 2**;**

int tempVar**;**

tempVar **=** x**;**

x **=** y**;**

y **=** tempVar**;**

And the mesmerizing XOR swap

int x **=** 5**;**

int y **=** 2**;**

x **=** x **^** y;

y **=** y **^** x;

x **=** x **^** y;

Both of these snippets of code just swap the values of two ints which seems mundane but the fascinating thing is that they both could be a code snippet of either C or Java. Both languages have most of the same “primitive data types” such as char, short, int, long, float, and double. Both C and Java need to declare variables with a type and a name before giving it a value and you need to assign it a value before you can perform mathematical equations on it. Both C (Unnamed 2003) and Java (Oracle 2015) have the same basic mathematical operators which are [+, −, \*, /, %, &, |, ^, ~]. I won’t go into detail of each one but the best way to sum up the difference between the use of these operators in C and Java is that Java is a little more type safe than C. Methods and functions are quite similar in C and Java they start with the data type they will return followed by the name of the function followed by their accepted parameters surrounded by parenthesis and finally the function itself is surround by curly braces.

int thing**(**int param**){**

**return** param**;**

**}**

The syntax of the function itself is pretty lax so there is no reason to go over it. Java has lots of keywords such as Public, Private, Abstract, Interface, Extend etc. there is really no point to go over these because without doing some sort of work around all C functions are public and a struct (the closest thing C has to a class) does not have anything that resembles inheritance. You only need to use the keyword class once when defining the specific class but in C the keyword struct becomes part of the structs name unless you use the define or typedef keywords.

struct lawyer**{**

int number**;**

**};**

//struct name is "struct lawyer"

**typedef** struct lawyer**{**

int number**;**

**}**dog**;**

//struct name is “dog”

The define and typedef keywords are essentially used to make one thing look like another for example the code above can be written as.

#define int1 ;

#define teeeefansnj int

#define PUT\_CURLY\_HERE }

struct lawyer**{**

teeeefansnj number int1

PUT\_CURLY\_HERE int1

//this is good code according to a compiler but no one else

Improper or what might seem almost like malicious use of these make code extremely hard to read which makes it not surprising that nothing like it is in Java. Finally my absolute favorite keywords which are continue, break, and goto which are all widely discouraged, break and continue work in both languages exactly the same. The hated goto function only works in C however it is still highlighted as a keyword in Java for future use.

**BUILT IN DATA TYPES**

In Java and C it seems that on the surface many of their primitive data types are the same but they are different if you take a very close look. Java’s are much more intuitive where a character can only take in an ascii character such as ‘a’ ‘b’ or ‘c’. The integers are short, int and long they are 2 bytes 4 bytes and 8 bytes respectively. Then the floating point numbers which are float and double, they are 4 and 8 bytes respectively and they do follow the 754 IEEE standard (Oracle 2016). Then there are booleans with a value of true or false. Non-primitive data types such as classes are saved as addresses and cannot be altered by mathematic operators at all. All of these can of course be held in arrays. C works has a bit more freedom when working with built in data types where a char is basically a 1 byte int, and despite popular belief short, int, and long are not the same size by definition in C the official requirements are

“The 8-bit signed char … in the range of −128 to 127 … The 16-bit short int … in the range of −32,768 to 32,767 … The 32-bit int … in the range of −2,147,483,648 to 2,147,483,647 … The 32-bit long … in the range of at least −2,147,483,648 to 2,147,483,647 … The 64-bit long long int … in the range of −9,223,372,036,854,775,808 to 9,223,372,036,854,775,807” (GNU 2008).

These are not hard requirements but minimums that each datatype can hold and can change based on the operating system. There are no booleans in C, all boolean operations are done with other data types with a binary 0 being equivalent to false and anything else being equivalent to true. Then there are the floating point numbers in C which work very similar to how they do in Java they also follow the 754 IEEE standard (Thomas 2001). Then there are pointers which are an address made up with enough bits as need to fully address memory so in most current computers is 8 bytes. Every one of these data types can be operated on by any mathematical operator. Much like Java all Built in data types can be used in an array.

**CLASSES VS STRUCTS**

There is enough to discuss here that you could write a book on it unfortunately this paper is only on syntax so there really isn’t way too much to discuss. In Java it is much simpler to initialize a class because it forces you to dynamically create it like

lawyer dog **=** **new** lawyer**();**

or in C if dynamically made it, it would look like

lawyer**\*** dog **=** malloc**(sizeof(**dog**));**

but you can also make it similar to how primitive data is made like

lawyer dog;

Of course if you want to actually use the struct the act of initializing it is more complex. When calling a field or method attached to a Java class it is always lawyer**.**number, where in C it is the same but more often the not you will see dog**->**number and that is because dogis an address and -> dereferences the struct and then accesses it’s field the same process can be written as **(\***dog**).**number. There are a lot more syntactical differences but I don’t feel like it’s worth writing about because of just how different a struct is from a class.

**FUNCTION VS METHOD**

I think this is the area where most novice programmers have trouble wrapping their heads around because of how similar it looks on the outside but how different it really is when you start looking at it. In Java every method is attached directly to a class and by defaulted can only be called by instances of that class, so without an instance of the class it is impossible to call that method. This is fixed by making the method static which makes the method visible without a specific instance of the class being initialized; this is especially important in the case of a main method. This all sounds really simple but all of this needs to be taken into consideration when writing the simplest program take this program for example.

public class HelloWorld **{**

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

// Prints "Hello, World" to the terminal window.

System**.**out**.**println**(**"Hello, World"**);**

**}**

**}(** Sedgewick 2015)

I know I wouldn’t want to personally explain to someone who is programming for the first time how “the HelloWorld class is not initialized so you need main to be static in order for the program to run”. In C every function is very similar to a static Java method; it can be seen globally, does not need anything to be initialized, and all share the same namespace. Writing a C program is very similar to writing a Java program where every method is static. One weird problem with C is that there is actually a finite number of function names, because they all share a common namespace, but there are enough possible names to get by in most programs.

Both Java methods and C functions can only return one data type, and for the parameter they can both take numerous data types as well variable numbers of arguments implemented through argument lists, although in Java argument lists can only take one type of data while C can take various datatypes in the same argument list. For Java there is a lot of type checking when it comes to parameters, this is to make sure you are passing the right thing in the right place but for C there is next to no type checking, it is the same story for the return statements.

One big feature that Java has that C does not is overloaded functions where you can essentially have multiple different functions in the same name space as long as they take different sets of parameters. Trying this in C will lead to an immediate syntax error. One big feature that C has that Java doesn’t is function pointers which just treats a function as a variable and allows you to pass it around through other functions (Milanova 2003), Java does not have an equivalent feature.

**CONCLUSION**

Although C and Java have similar looking syntax and is generally written similarly you could spend an astronomical amount of time discussing how they are different. The big theme is that C generally has much less rules as far as syntax is concerned and Java is very restrictive syntactically.

**CITATIONS**

1. Sumit Gupta, Nargish Gupta, Rishabh Gupta (January 2014). International Journal of Computer Applications (0975 –8887)Volume 86–No 11. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.429.682&rep=rep1&type=pdf
2. Michael D. Ernst, Greg J. Badros, and David Notkin (December 2002). An Empirical Analysis of C Preprocessor Use. Retrieved from <https://homes.cs.washington.edu/~mernst/pubs/c-preprocessor-tse2002.pdf>
3. Unnamed (summer 2003). Bitwise Operators in C. Retrieved from <https://www.cs.umd.edu/class/sum2003/cmsc311/Notes/BitOp/bitwise.html>
4. Oracle (2015). Bitwise and Bit Shift Operators. Retrieved from <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/op3.html>
5. Oracle (2016). Class Float. Retrieved from <https://docs.oracle.com/javase/7/docs/api/java/lang/Float.html>
6. Jim Thomas(2001). IEEE 754 Support in C99. Retrieved from <http://grouper.ieee.org/groups/754/meeting-materials/2001-07-18-c99.pdf>
7. GNU(November 2008). The GNU C Reference Manual. Retrieved from <http://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html#GNU-Free-Documentation-License>
8. Robert Sedgewick, Kevin Wayne(October 2015). HelloWorld.java. Retrieved from <http://introcs.cs.princeton.edu/java/11hello/HelloWorld.java.html>
9. Ana Milanova, Atanas Rountev, Barbara G. Ryder(April 2003). Precise Call Graphs for C Programs with Function Pointers. Retrieved from <http://www.cs.rpi.edu/~milanova/docs/paper_kluw.pdf>