# Java and C Comparison

## Introduction

For this project I developed two shop programs, one using a procedural programming language, C and the other using an object orientated language, Java. Java is a high level language and C is considered mid-level language as it is closer to the system.

### Data Structures

There were similarities from the beginning, for example defining a struct in C was similar to defining a class in Java. A struct groups fields together to be used later while in java a class represents a blueprint from which objects are later created, both data structures. These are defined at the beginning of both projects.

*struct* Product {

*char*\* name;

*double* price;

};

Figure 1: Example of product struct in C shop

public class Product {

    private *String* name;

    private *double* price;

Figure 2: Example of product class in Java shop

However, Java does have the additional functionality of getters and setters which come in quite useful later on.

public class Product {

    private *String* name;

    private *double* price;

    public Product(*String* *name*, *double* *price*) {

        this.name = name;

        this.price = price;

    }

    public *String* getName() {

        return name;

    }

    public *double* getPrice() {

        return price;

    }

Figure 3: Java getters

sname.getProduct().getName()

Figure 4: Java calling a getter

### Run order

C is blocks of code, and has to be executed in the correct order otherwise you may run into errors. Java is divided into separate pages and it more modular and doesn’t appear to have this problem.

### Reading in files

When reading in the files, C had to declare variables and initialize counters while Java make use of import java.io.File package which helps with inputs and outputs. C also needs to tackle memory allocation while pulling in the data from the new file which means additional lines of code has to be written. Making Java less verbose and easier to read.

FILE \* fp;

*char* \* line = NULL;

*size\_t* len = 0;

*ssize\_t* read;

*int* i = 0;

    fp = fopen("orders.csv", "r");

    // if file doesn't exist exit program

    if (fp == NULL)

        exit(EXIT\_FAILURE);

    getline(&line, &len, fp);

    // read customer name and budget

*char* \*n = strtok(line, ",");

*char* \*b = strtok(NULL, ",");

*char* \*name = malloc(sizeof(*char*)\*50);

    strcpy(name, n);

*double* budget = atoi(b);

*struct* Customer current ={ name, budget };

    //struct Shop shop = { cashInshop };

Figure 5: Reading in a file in C

shoppingList = new *ArrayList*<>();

*List*<*String*> lines = Collections.emptyList();

        try {

            lines = Files.readAllLines(Paths.get(fileName), StandardCharsets.UTF\_8);

*String*[] firstLine = lines.get(0).split(",");

            name = firstLine[0];

            budget = Double.parseDouble(firstLine[1]);

            // i am removing at index 0 as it is the only one treated differently

            lines.remove(0);

Figure 6: Reading in a file in Java

*char* \*name = malloc(sizeof(*char*)\*50);

Figure 7: Memory allocation makes C more verbose

### Printing out variables

In C, you need to know the data type of a variable before you can print it out. For example, %s for a string, %d for an integer, %2.f for a double. Thankfully this is not required in Java. This is a good example of C being lower level than Java.

printf("%s, in stock: %d \n",s.stock[i].product.name, s.stock[i].quantity );

Figure 8: Print out in C

System.out.println(sname.getProduct().getName()+ " " +sname.getQuantity());

Figure 9: Print out in Java

### User Input

There were similarities in the approaches to capturing the user inputs. Both used their print statements to ask the question. Once again C had to deal with memory allocation where java did not. Java used Scanner a built in class from a package java.util and C used scanf.

Here I ran into the same issue in both programs during the livemode part of the project, when asking what product the user wants if the product was made up of two strings then I ran into errors. While solving the issue in java was straightforward, solving it in C required an regular expression "%[^\n]s" to ensure the program read all the characters until it hit \n.

printf("What is your name?\n");

*char* \*name = malloc(sizeof(*char*)\*50);

    scanf(" %s", name);

    printf("What is your budget?\n");

*double* budget;

    scanf("%lf", &budget);

    printf("What would you like to buy?");

*char* \*p = malloc(sizeof(*char*)\*50);

*char* temp;

    scanf("%c", &temp);

    scanf("%[^\n]s", p);

    printf("How many?\n");

*int* quantity;

    scanf(" %d", &quantity);

*struct* Customer newcurrent ={name, budget};

Figure 10: User input in C

*Scanner* s = new Scanner(System.in);

        System.out.println("-----Live mode------");

        System.out.println(""+"What's your name?");

*String* name2 = s.nextLine();

        System.out.println("Please enter budget?");

*double* budget2 = Double.valueOf(s.nextLine());

        System.out.println("What product do you want to buy?");

*String* productname = s.nextLine();

        System.out.println("How many ?");

*int* quantity = Integer.valueOf(s.nextLine());

        //livemode();

*ArrayList*<*ProductStock*> shoppingList2 = new *ArrayList*<*ProductStock*>();

*Product* p = new Product(productname, 0);

*ProductStock* sn = new ProductStock(p, quantity);

                shoppingList2.add(sn);

s.close();

*Customer* newcustomer = new Customer( name2, budget2, shoppingList2);

        System.out.println(newcustomer);

Figure 11: User Input in Java

### Loops

It was the loops where I noticed a big difference in the approach. For example, with C you were required to have an index variable defined in the struct shop which acted as the counter for the ProductStock array. In Java, it seemed more intelligent and I was able to use an enhanced loop for (*ProductStock* product : shopList ) to loop through the ProductStock array and print out the products without a counter

for (*int* i = 0; i < s.index; i++)

    {

        // loop through customer order

        for (*int* j = 0; j < current.index; j++)

        {

*struct* Product product = s.stock[i].product;

*struct* Product productcustomer = current.shoppingList[j].product;

            // compare shop and order

            if (strcmp(product.name, productcustomer.name) ==0)

            {

                    printf("%s, Price %.2f \* Quantity %d\n", current.shoppingList[j].product.name, product.price, current.shoppingList[j].quantity);

Figure 12: Looping in C

for (*ProductStock* sname : shopstock ) {

                for (*ProductStock* product : shopList ) {

                    if(product.getProduct().getName().equals(sname.getProduct().getName())){

                        System.out.println(product.getProduct().getName()+ " "+ sname.getProduct().getPrice()+ " \* "+ product.getQuantity() );

Figure 13: Caption: looping in Java

## Conclusion

As a new programmer for both, Java and C, there was a huge learning curve during this project, however once I had completed the C part everything seemed to move at a quicker pace for the Java part. There were many similar aspects of C that I was able to reuse in the Java code. Once I knew how I was going to develop the program, it was just merely figuring out how to translate it into Java. I did find C more frustrating to develop against due to the memory allocations and declaring the datatypes. I would like to learn more about Java as I feel I’ve only touched the tip of the iceberg with this project and there is a lot more to learn.