Multi-Paradigm Programming – Shop Assignment

Student name: Aidan O'Connor

Student number: G00364756

C and Java shop assignment programs information

C file name: "cshop.c" (Functions correctly for all specifications of the shop assignment)

Live version C file name: "live_shop.c" (Does not function correctly)

Java file name: "ShopVideoVersion" (Functions correctly for all specifications of the shop assignment)

Live version Java file name: "ShopVideoVersion_Live" (Does not function correctly)

Programming Paradigms

The term programming paradigm is the style or way of thinking about approaching problems. There are a number of different programming paradigms. The most common are:

1. Object Orientated
2. Functional
3. Imperative

Procedural Programming

Procedural programming is based on the concept of the procedure call. Procedures contain a series of computational steps to be carried out. The sequence of these computational steps is very important in procedural programming. The C programming language is one that follows the procedural programming paradigm.

Object-orientated Programming

Object-orientated programming is a paradigm which employs the concept of objects. Objects can be defined as data fields that have unique attributes and behaviour. The program is organised around the data rather than functions and logic. The Java programming language is one that follows the object-orientated programming paradigm.

Differences between Procedural and Object Orientated Programming

Order of functions and methods

Below is a snippet of the code I developed for the C programming part of the Shop assignment. It is the main function which is called upon execution of the file when compiled and run on the machine.

```c
int main(void)
{

    struct Shop shop = createAndStockShop();
    struct Customer customer = createCustomer(shop, customer);
    //printShop(shop);
    struct Customer *ptr_customer;
    ptr_customer = &customer;
    findProductPrice(shop, ptr_customer);
    printCustomer(customer, ptr_customer);
    struct Shop *ptr_shop;
    ptr_shop = &shop;
    enoughforPurchase(customer, shop, ptr_shop);
    overwrite_csv(shop, ptr_shop);

    return 0;
}
```

Figure 1: Main function, C program for Shop assignment

It is difficult to add functionality to the C program. Any additions need to be incorporated carefully as procedural code is a step by step approach. Inserting a new step in the incorrect place will have a knock-on effect to the whole program. The functions and structs must be read in the correct sequence otherwise the program will not run or will give an unintended output. I ran into problems several times in my development from defining a function before another function which relied on its input and it led to troubleshooting and reorganising the sequence of the functions.

This is not an issue in object orientated programming. Methods can be defined in any order within the program and called in the main method which will not throw up any errors. The sequence in which the methods are defined and called does not matter. Below are snippets of the Java program I developed to answer the shop assignment.

```java
public class Runner {

    public static void main(String[] args) {
        Shop shop = new Shop("src/ShopVideoVersion/stock.csv");
        Customer james = new Customer("src/ShopVideoVersion/customer.csv");
        shop.processOrder(james);
        shop.enoughtoBuy(james,shop);
        //shop.DiditSet(james);
    }
}
```

Figure 2: Java program, Main method Shop assignment

For example in the above main method the “enoughtoBuy()” method relies on numerous other methods defined in the class “Shop”. The method “modifyCSV” is defined after it is called in the “enoughtoBuy()” method however this does not matter in the object-oriented programming language Java as it retrieves the method wherever it is declared and runs the code.

```
185
186     public void modifyCSV(Shop s) {
187         String shopcash = String.valueOf(s.getCash());
188         List<String> test = new ArrayList<>();
189         for (ProductStock productStock : stock) {
190             Product stockp = productStock.getProduct();
191             int stockquantity = productStock.getQuantity();
192             test.add(stockp.getName() + ",");
193             test.add(stockp.getPrice() + ",");
194             test.add(String.valueOf(stockquantity));
195             test.add("\r\n");
196         }
197
198         try {
199         FileWriter fw = new FileWriter("/Users/aoconnor.CER/Desktop/52960--
200         BufferedWriter bw = new BufferedWriter(fw);
```

Figure 3: ModifyCSV method

```
145     public void enoughtoBuy(Customer c, Shop s) {
146         double result = 0;
147         double cash = getCash();
148         double budget = c.getBudget();
149         double total = c.getTotal();
150         if ( budget < total) {
151             System.out.println(c.getName()+" has a budget of €" + budget);
152             System.out.println(c.getName()+" does not have enough money to
153         }
154         else {
155             int trigger = returnTrigger(c,s); // develop method to determin
156             for (int z = 0; z < 1; z++) {
157                 if (trigger == 1)
158                 {
159                     cash = cash + total;
160                     s.setCash(cash);
161                     modifyQuantity(c);
162                     modifyCSV(s);
163                 }
```

Figure 4: enoughtoBuy() method

Abstraction

C has a low level of abstraction which means certain aspects need to be managed manually by the developer rather than automatically like other programming languages with a higher level of abstraction (e.g. Java). An aspect that needs manual management is memory. This lower level of abstraction can be cumbersome for developers. I found this aspect difficult as I ran into problems often forgetting to declare variables for memory. In the code below variables needed to be declared for allocation of memory [char *name = malloc(sizeof(char) * 50);] before data could be assigned to the variables.

```c
struct Shop createAndStockShop()
{
    FILE * fp;
    char * line = NULL;
    size_t len = 0;
    ssize_t read;

    fp = fopen("stock.csv", "r");
    if (fp == NULL)
        exit(EXIT_FAILURE);

    getline(&line, &len, fp);
    double cashInShop = atof(line);
    struct Shop shop = { cashInShop };

    while ((read = getline(&line, &len, fp)) != -1)
    {
        char *n = strtok(line, ",");
        char *p = strtok(NULL, ",");
        char *q = strtok(NULL, ",");
        int quantity = atoi(q);
        double price = atof(p);
        char *name = malloc(sizeof(char) * 50);
        strcpy(name, n);
        struct Product product = { name, price };
        struct ProductStock stockItem = { product, quantity };
        shop.stock[shop.index++] = stockItem;
    }
    return shop;
}
```

Figure 5: Function to create and stock the shop in C.

Object-oriented programming has a higher level of abstraction meaning time does not need to be given to cumbersome tasks such as memory management. This does mean efficiency losses but modern computers are so fast that the efficiency losses are not a problem for the average developer.

```java
public class Shop {
    // String newLine = System.getProperty("line.separator"); // Putting in
    private double cash;
    private ArrayList<ProductStock> stock;

    public Shop(String fileName) {
        stock = new ArrayList<>();
        List<String> lines = Collections.emptyList();
        try {
            lines = Files.readAllLines(Paths.get(fileName), StandardCharset
            System.out.println(lines.get(0));
            cash = Double.parseDouble(lines.get(0));
            // i am removing at index 0 as it is the only one treated diffe
            lines.remove(0);
            for (String line : lines) {
                String[] arr = line.split(",");
                String name = arr[0];
                double price = Double.parseDouble(arr[1]);
                int quantity = Integer.parseInt(arr[2].trim());
                Product p = new Product(name, price);
                ProductStock s = new ProductStock(p, quantity);
                stock.add(s);
            }
```

Figure 6: Java method for creating and stocking shop

Even though 2 different programming paradigms are employed to produce the shop assignment they can both produce the same output given that they are programmed to do so.

Output of C program – Shop Assignment

Below are the csv inputs for my C program.

	A	B	C
1	200		
2	Coke Can	1.1	100
3	Bread	0.7	30
4	Spaghetti	1.2	100
5	Tomato Sa	0.8	100
6	Big Bags	2.5	4
7			

Figure 8: Stock csv input

	A	B
1	James	40
2	Coke Can	10
3	Bread	3
4	Spaghetti	5
5	Tomato Sa	5
6	Big Bags	4
7		

Figure 7: Order csv input

Below is the console output to my C program for the above inputs.

```
<terminated> Runner [Java Application] C:\Program Files\Java\jdk-13.0.1\bin\javaw.exe (15 Nov 2019, 23:22:53)
The shop has €200 in cash before any purchase is processed.


The shop's product pricelist:
Coke Can costs 1.1
Bread costs 0.7
Spaghetti costs 1.2
Tomato Sauce costs 0.8
Big Bags costs 2.5
The total cost of James's order is €33.10

Shop stock quantity:     100
Customer order quantity: 10
Shop stock quantity:     30
Customer order quantity: 3
Shop stock quantity:     100
Customer order quantity: 5
Shop stock quantity:     100
Customer order quantity: 5
Shop stock quantity:     4
Customer order quantity: 4


The shop has the remaining inventory:
Coke Can,1.1,90
Bread,0.7,27
Spaghetti,1.2,95
Tomato Sauce,0.8,95
Big Bags,2.5,0
```

Figure 9: Console output for C program for the shop assignment

Below is the csv output to my C program for the above inputs. The shop's state is consistent as the csv is overwritten.

	A	B	C
1	233.1		
2	Coke Can	1.1	90
3	Bread	0.7	27
4	Spaghetti	1.2	95
5	Tomato Sa	0.8	95
6	Big Bags	2.5	0
7			

Figure 10: Output to Stock CSV after the C program has run

If the inputs are changed so that the customer does not have enough money to purchase the order or the shop does not have enough stock to fill the order an appropriate error message will show in the console.

Output of Java program – Shop Assignment

Below are the csv inputs for my Java program.

	A	B	C
1	200		
2	Coke Can	1.1	100
3	Bread	0.7	30
4	Spaghetti	1.2	100
5	Tomato Sa	0.8	100
6	Big Bags	2.5	4
7			

Figure 11: Stock csv input

	A	B
1	James	40
2	Coke Can	10
3	Bread	3
4	Spaghetti	5
5	Tomato Sa	5
6	Big Bags	4

Figure 12: Order csv input

Below is the terminal output to my Java program for the above inputs.

```
aoconnor@CERW10LT270 ~/cfiles
$ gcc cshop.c

aoconnor@CERW10LT270 ~/cfiles
$ ./a.exe
------------
CUSTOMER NAME: James
------------
CUSTOMER BUDGET: 40.00
------------
CUSTOMER ORDER: 10 of Coke Can at a price of 1.10
------------
CUSTOMER ORDER: 3 of Bread at a price of 0.70
------------
CUSTOMER ORDER: 5 of Spaghetti at a price of 1.20
------------
CUSTOMER ORDER: 5 of Tomato Sauce at a price of 0.80
------------
CUSTOMER ORDER: 4 of Big Bags at a price of 2.50
------------
The cost of this order is: 33.10
------------

 shop quantity: 100

 customer quantity: 10

 shop quantity: 30

 customer quantity: 3

 shop quantity: 100

 customer quantity: 5

 shop quantity: 100

 customer quantity: 5

 shop quantity: 4

 customer quantity: 4

------------
```

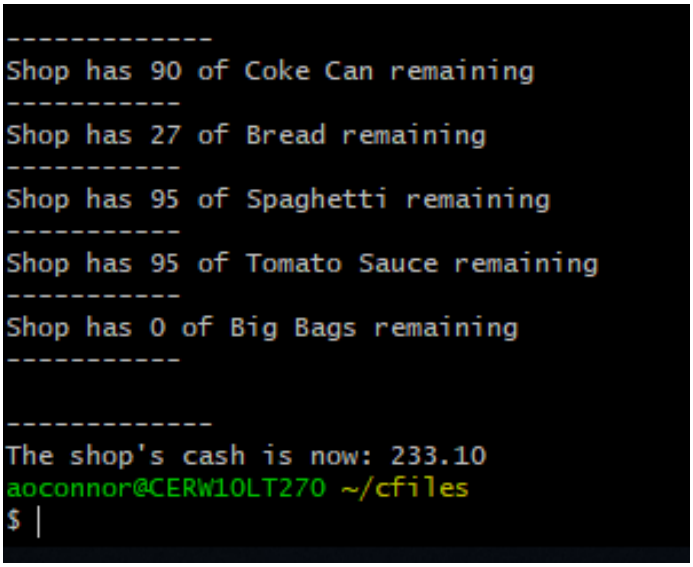

Figure 13: Output to terminal, Java program for Shop assignment

Below is the csv output to my Java program for the above inputs. The shop's state is consistent as the csv is overwritten.

	A	B	C
1	233.1		
2	Coke Can	1.1	90
3	Bread	0.7	27
4	Spaghetti	1.2	95
5	Tomato Sa	0.8	95
6	Big Bags	2.5	0
7			

Figure 14: Output to Stock CSV after Java program runs

If the inputs are changed so that the customer does not have enough money to purchase the order or the shop does not have enough stock to fill the order an appropriate error message will show in the console.

Conclusions

There are a number of programming paradigms available for developers to chose from. Each paradigm has their own attributes and characteristics which can be beneficial for certain applications and not so beneficial for others. Procedural programming such as C programming is a very efficient language by design which can be used if efficiency is important. However, if efficiency is not as much a concern and ease of adapting functionality is more of a concern an object orientated program such as Java would be more applicable. It took me much longer to code my C program than my Java program, I found it easier to develop code using the Java programming language.