Multi-Paradigm Programming Shop Assignment

Gerard O'Mahony

H.Dip in Data Analytics

November 17, 2019

Multi Paradigm Programming - Assignment One

Advised by: Dr Dominic Carr

Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)

Contents

About this assignment

Abstract This report outlines the concepts of the two software development paradigms, procedural and object-orientated programming. This is achieved by describing and analysing the development of a program which functions the same but is written in the procedural language *C* and object-oriented language *Java*. The two paradigms are compared and contrasted and conclusions are drawn as to the suitability of each to components of the programming assignment. Source code is provided in the appendices.

Authors Gerard O Mahony.

The GMIT LaTeX report template is courtesy of Dr Ian McLoughlin, Lecturer, GMIT

Chapter 1

Introduction

The purpose of this assignment was to write a program which performs the same function using two different programming paradigms, procedural and object orientated programming. The procedural program was developed in the *C* programming language whereas *Java* was used for the object orientated program.

1.1 *C* Programming Language

The *C* programming language was first developed by Dennis Ritchie, a computer scientist at AT&T's Bell Laboratories in the early 70's. It is a low level programming language in so far as it provides little or no abstraction of programming concepts and is very close to writing actual machine instructions[1].
The *C* language provides a means of storing data in derived data types called *structures*, and this is as close to objects as *C* gets. The language is very simple and you are required to handle operations such as memory allocation/deallocation, string and file operations which are abstracted away in higher level programming languages. This provides more power to the user but at the cost of the additional responsibility of handling these operations correctly.
The Shop assignment was written in C in a procedural paradigm where we wrote the program as it should be executed as a series of step-by-step instructions. This entailed writing the basic data structures firstly which then could be re-used in more complex data structures. The functions for the population of these data structures were created, also as a series of logical instructions. The final 'main' component of the program, which is what is executed when the program is compiled and run, also contains a sequen-

tial set of instructions which creates the data structures and then populates them with user input. This data is then processed as required in a sequential manner calling the functions as needed. These functions must exist in the program scope when called but are not necessarily bound to any data type. It is possible to access the data members in the structures throughout the program directly without having any specific methods to do so. The main Shop data types storing the shop products and price as well as the customers data are global to the program and need to be passed around and to various functions, or pointers to the data, so that it can be modified.

1.2 *Java* Programming Language

The *Java* programming language is considered to be an object-orientated programming language as it uses classes which are blueprints from which objects are created. It is a general purpose language that was designed to be run on any platform once compiled without the need to re-compile. The phrase 'Write once, run anywhere' describes the languages' philosophy succinctly.
The equivalent Shop program in *Java* was written as a collection of data classes and methods within these classes to act on the data to retrieve or modify it. The data contained within the classes are only acted on by specific methods and cannot be accessed by external means and the data is encapsulated in the class object when instantiated. This is a core concept of the object-orientated paradigm. This makes classes and objects very portable and forces the developer to think in a specific way to create these objects so they incorporate this encapsulation and modularity.

Chapter 2

Conclusion

The development of the Shop is *C* was much more difficult than in *Java* due to the level of the language and having to consider basic operations such as string splitting which are built into the string class in most object-orientated languages. In a sense, we still developed the *C* data with some object-orientated methodology in mind as the basic data types were developed initially and used as constituent data components in more complex structures. This emulates the object-orientated inheritance idea. Where *C* differed was the tracking of the data throughout the program and how it was getting modified in different functions. I would imagine that as a *C* program structure got larger this would become extremely difficult.
Using *Java* was easier as a language and the built in methods in the primitive types such as string.equals, string.format or object iterators facilitated this. It was more intuitive to consider the data as classes and the use of built in getters or setters and other methods to act on the data. The tight coupling of the data and methods made more intuitive sense whereas the *C* methods and functions were difficult to re-factor when necessary due to the looser coupling with the data and potential knock-on effects of modifying one function on another part of the program. It was much easier in object-orientated programming to deal with the state of the shop and customer objects and their interaction. The final component of the program, where interacting with the user, seemed more suited to procedural type programming.

Appendices

Appendix A

Shop *C* Source Code

```
#define _GNU_SOURCE
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <dirent.h>
#include <stdbool.h>
#include <math.h>

struct Product {
char* name;
double price;
};

struct ProductStock {
struct Product product;
int quantity;
};

struct Shop {
double cash;
struct ProductStock stock[20];
int index;
};

struct Customer {
char* name;
```

```
double budget;
struct ProductStock shoppingList[10];
int index;
};

struct CustomerQueue {
struct Customer customerQ[20];
int index;
};

void printProduct(struct Product p)
{
printf("%s, Price: %.2f, ", p.name, p.price);
}

void printCustomer(struct Customer c)
{
printf("CUSTOMER NAME: %s \nCUSTOMER BUDGET: %.2f\n", c.name, c.budget
printf("---------------\n");
int i;
for( i = 0; i < c.index; i++)
{
printProduct(c.shoppingList[i].product);
printf("%s ORDERS %d OF ABOVE PRODUCT\n", c.name, c.shoppingList[i].qu
double cost = c.shoppingList[i].quantity * c.shoppingList[i].product.p
printf("The cost to %s will be %.2f\n", c.name, cost);
}
printf("\n");
}

struct Shop createAndStockShop()
{

// struct Shop shop;
struct Shop shop = { 200 };

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);

shop.cash = atof(line);

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;
}
fclose(fp);

return shop;
}

void printShop(struct Shop* s)
{
printf("Shop: \n Cash=%.2f\n Stock:\n", s->cash);
int i;
for ( i = 0; i < s->index; i++)
{
printProduct(s->stock[i].product);
printf("Quantity: %d\n", s->stock[i].quantity);
}
}


struct Customer processCustomer(char *fileName)
{
FILE * fp;
```

```
char * line = NULL;
size_t len = 0;
ssize_t read;
char customerDirFileName[255] = "./customers/";
strcat(customerDirFileName,fileName);

fp = fopen(customerDirFileName, "r");
if (fp == NULL)
exit(EXIT_FAILURE);

// Get customer Name and budget from first line
getline(&line, &len, fp);

// Use string token function to split CSV values
// into name n and cash c string values
char *n = strtok(line, ",");
char *c = strtok(NULL, ",");

char *name = malloc(sizeof(char) * 50);
strcpy(name, n);

double cash = atof(c);

struct Customer newCustomer = { name, cash };

while ((read = getline(&line, &len, fp)) != -1)
{
char *p = strtok(line, ",");
char *q = strtok(NULL, ",");
int quantity = atoi(q);

char *productName = malloc(sizeof(char) * 50);
strcpy(productName, p);
struct Product product = { productName };
struct ProductStock stockItem = { product, quantity };
newCustomer.shoppingList[newCustomer.index++] = stockItem;
}

fclose(fp);

return newCustomer;
```

```
}

struct CustomerQueue queueCustomers()
{
struct CustomerQueue queue = {.index = 0};
size_t flen;
DIR *customerDir;
struct  dirent *file;
char customerFileName[255];
customerDir = opendir("./customers");
if (customerDir)
{
while ((file = readdir(customerDir)) != NULL)
{
strncpy(customerFileName, file->d_name, 254);
customerFileName[254] = '\0';

if (strstr(customerFileName, ".cust") != NULL)
{
struct Customer newCustomer = processCustomer(customerFileName);
queue.customerQ[queue.index++] = newCustomer;
}
}
closedir(customerDir);
}
return queue;
}

void printQueue(struct CustomerQueue q)
{
printf("The queue has %d customers\n", q.index);
int i;
for (i = 0; i < q.index; i++)
{
printCustomer(q.customerQ[i]);
}
}

void serveCustomer(struct Shop* s, struct CustomerQueue* c, bool q)
{
int stockQuantity;
```

```
int stockIndex;
double itemPrice;
double costToCust;
double remainingBudget;

int iCust;
for (iCust = 0; iCust < c->index; iCust++)
{
printf("==========\n");
if(q)
{
printf("Hello %s how can I help you today?\n", c->customerQ[iCust].nam
}

int iList;
for (iList = 0; iList < c->customerQ[iCust].index; iList++)
{
printf("You want %d of %s\n",
c->customerQ[iCust].shoppingList[iList].quantity,
c->customerQ[iCust].shoppingList[iList].product);

stockIndex = checkProductStock(s, c, &iCust, &iList, &stockQuantity, &
if (stockQuantity <= c->customerQ[iCust].shoppingList[iList].quantity)
{
printf("Sorry %s, we don't have enough stock to fulfill your order\n",
} else if (stockQuantity > 0) {
printf("We have %d of %s which cost %.2f each\n", stockQuantity,
c->customerQ[iCust].shoppingList[iList].product.name, itemPrice);
costToCust = itemPrice * c->customerQ[iCust].shoppingList[iList].quant
printf("%d of %s will cost you: \n",
c->customerQ[iCust].shoppingList[iList].quantity,
c->customerQ[iCust].shoppingList[iList].product);
printf("%.2f\n", costToCust);
remainingBudget = c->customerQ[iCust].budget - costToCust;

if (remainingBudget >= 0.0)
{
printf("You have %.2f left in your %.2f budget\n", remainingBudget,c->
c->customerQ[iCust].budget = remainingBudget;
reduceProductStock(s, &stockIndex,
c->customerQ[iCust].shoppingList[iList].product.name,
```

```
c->customerQ[iCust].shoppingList[iList].quantity);
s->cash = s->cash + costToCust;
}
else
{
printf("Sorry %s, you don't have enough money to purchase %d of %s\n",
c->customerQ[iCust].name,
c->customerQ[iCust].shoppingList[iList].quantity,
c->customerQ[iCust].shoppingList[iList].product);
continue;
}
} else {
printf("Sorry %s, we are out of stock for %s\n",
c->customerQ[iCust].name, c->customerQ[iCust].shoppingList[iList].prod
}
}
}
}

int checkProductStock(struct Shop* s, struct CustomerQueue* c, int *iC
{
int compare;
int i;
for (i=0; i < s->index; i++)
{
compare = compare_string(s->stock[i].product.name,
c->customerQ[*iCust].shoppingList[*iList].product.name);
if (compare == 0)
{
*quantity = s->stock[i].quantity;
*price = s->stock[i].product.price;
return i;
}
}
*quantity = -1;
*price = 0.0;
return -1;
}

int reduceProductStock(struct Shop *s, int *stockIndex, char *customer
{
```

```
int compare;
compare = compare_string(s->stock[*stockIndex].product.name, customerP
if (compare == 0)
{
s->stock[*stockIndex].quantity = s->stock[*stockIndex].quantity - custo
return 1;
}
return 0;
}

int compare_string(char *first, char *second)
{
while (*first == *second)
{
if (*first == '\0' || *second == '\0')
break;

first++;
second++;
}
if (*first == '\0' && *second == '\0')
return 0;
else
return -1;
}

int checkUserInput()
{
int n;
double temp;

char value[MAX_INPUT] = "0";
char str[MAX_INPUT] = "";

// Precision for integer checking
double val = 1e-12;

fgets(value, 255, stdin); // Read input

// Check for integers.
if (sscanf(value, "%lf", &temp) == 1)
```

```
{
n = (int)temp; // typecast to int.
if (fabs(temp - n) / temp > val)
{
printf("Please enter an integer number\n");
return 0;
} else {
return n;
}
} else if (sscanf(value, "%s", str) == 1) {
printf("Please enter an integer number\n");
return 0;
} else {
printf("Please enter an integer number\n");
return 0;
}
}

void serveCounter(struct Shop* s, char *name)
{
struct CustomerQueue counterQ = {};
int productIndex=0;
char term;
char *productName = malloc(sizeof(char) * 50);
int quantity;
double budget;

printf("Hello %s\n", name);
printf("Please select something from our stocklist: \n");
printf("++++++++++++++++++++++++++++++++++++++++++++++++\n");
int i;
for (i = 0; i < s->index; i++)
{
printf("[%d]: %s\n", i+1, s->stock[i].product);
}
printf("++++++++++++++++++++++++++++++++++++++++++++++++\n");

while(true)
{
productIndex = checkUserInput();
```

```
if (productIndex > s->index+1)
{
printf("Please select a product index from the list\n");
continue;
} else if (productIndex == 0){
continue;

} else {
printf("++++++++++++++++++++++++++++++++++++++++++++++++\n");
break;
}
}
productIndex = productIndex -1;

strcpy(productName, s->stock[productIndex].product.name);

printf("What amount of %s would you like?\n", productName );
scanf("%d", &quantity);

printf("What's your budget today %s?\n", name);
scanf("%lf", &budget);

struct Customer counterCustomer = {
.name = name,
.budget = budget,
.index = 0};

struct Product product = {productName};
struct ProductStock stockItem = {product, quantity};
counterCustomer.shoppingList[counterCustomer.index++] = stockItem;

counterQ.customerQ[counterQ.index++] = counterCustomer;

serveCustomer(s, &counterQ, false);
}

int main(void)
{
struct Shop shop = createAndStockShop();
struct CustomerQueue queue = queueCustomers();
```

```
int answer;
int queueCleared = 0;
char customerName[256];
char *name;

printf("Welcome to the shop. What's your name?\n");
scanf("%s", &customerName);
name = customerName;

while (1)
{
printf("=============================================================\n");
printf("| Please select an option:
|\n");
printf("| ------------------------
|\n");
printf("| Allow the customer queue to go ahead:      [1] |\n");
printf("| Buy something yourself:                    [2] |\n");
printf("| Leave the shop:                            [3] |\n");
printf("| Print shop details:                        [4] |\n");
printf("=============================================================\n");
scanf("%d", &answer);
switch(answer)
{
case 1:
if (queueCleared == 0)
{
serveCustomer(&shop, &queue, true);
queueCleared = 1;
} else {
printf("The customer queue has alredy been served\n");
}
continue;

case 2:
serveCounter(&shop, name);
continue;

case 3:
printf("Thank you for your custom, please call again!\n");
printf("Goodbye\n");
```

```
return;

case 4:
printShop(&shop);
continue;

default:
return -1;
}
}

return 0;
}
```

Appendix B

Shop *Java* Source Code

B.1 Customer Class

```
package ShopApp;

import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Customer {
// Customer class

// Class members
private String name;
private double budget;
private ArrayList<ProductStock> shoppingList;

// Class method to initalise customer
public Customer(String fileName) {

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]);
lines.remove(0);
for (String line : lines) {
```

```
String[] arr = line.split(",");
String name = arr[0];
int quantity = Integer.parseInt(arr[1].trim());
Product p = new Product(name, 0);
ProductStock s = new ProductStock(p, quantity);
shoppingList.add(s);
}

}

catch (IOException e) {
e.printStackTrace();
}
}

public String getName() {
return name;
}


public double getBudget() {
return budget;
}

public void setBudget(double budget){
this.budget = budget;
}


public ArrayList<ProductStock> getShoppingList() {
return shoppingList;
}


@Override
public String toString() {
String ret = "Customer:\n Name:" + name + "\n Budget:" + String.format("%.2f", b
for (ProductStock productStock : shoppingList) {
ret+= productStock.getProduct().getName() + " Quantity:" + productStock.getQuantit
}
return ret;
}

}
```

B.2 CustomerFileFinder Class

```
package ShopApp;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

public class CustomerFileFinder {
// Class for finding customer files in a directory
List<String> inputFileinDir(String directory) {
List<String> inputFileinDir = new ArrayList<String>();
File dir = new File(directory);
for (File file : dir.listFiles()) {
if (file.getName().endsWith((".cust"))) {
inputFileinDir.add(file.getName());
}
}
return inputFileinDir;
}
}
```

B.3 CustomerFileWriter Class

```
package ShopApp;

import java.io.IOException;
import java.io.PrintWriter;

public class CustomerFileWriter {

public void toFile(String cName, double cBudget, String cProduct, int cQuantity){
try {
PrintWriter writer = new PrintWriter(cName + ".cust", "UTF-8");
writer.println(cName + "," + String.format("%.2f", cBudget));
writer.println(cProduct + "," + String.format("%d", cQuantity));
writer.close();
} catch (IOException e) {
e.printStackTrace();
}
}
}
```

B.4 Product Class

```
package ShopApp;

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;
}


public void setPrice(double price) {
this.price = price;
}

@Override
public String toString() {
return "Product [name=" + name + ", price=" + price + "]";
}



}
```

B.5 ProductStock Class

```
package ShopApp;

public class ProductStock {
private Product product;
private int quantity;

public ProductStock(Product product, int quantity) {
super();
this.product = product;
```

```
this.quantity = quantity;
}

public Product getProduct() {
return product;
}

public int getQuantity() {
return quantity;
}

public void setQuantity(int quantity){
this.quantity = quantity;
}

@Override
public String toString() {
return "ProductStock [product=" + product + ", quantity=" + quantity + "]";
}



}
```

B.6 Shop Class

```
package ShopApp;

import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Shop {

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), StandardCharsets.UTF_8);
```

```
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 differently
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);
}

}

catch (IOException e) {

// do something
e.printStackTrace();
}
}

public double getCash() {
return cash;
}

public ArrayList<ProductStock> getStock() {
return stock;
}

@Override
public String toString() {
String retn = "Shop:\n Cash=" + String.format("%.2f", cash) + "\n Stock:\n";
for (ProductStock item : stock) {
retn+= item.getProduct().getName() + ", Price:" +
String.format("%.2f", item.getProduct().getPrice()) +
", Quantity:" + item.getQuantity() +  "\n";
}
return retn;
}

public static void main(String[] args) {
// Shop shop = new Shop("src/ShopApp/stock.csv");
}

private double findPrice(String name) {

for (ProductStock productStock : stock) {
```

```
Product p =productStock.getProduct();
if (p.getName().equals(name)) {
return p.getPrice();
}
}

return -1;
}

public void processOrder(Customer c) {
ArrayList<ProductStock> custList = c.getShoppingList();

System.out.println("Hello " + c.getName() + " how can I help you today? \n");
System.out.println("==========================================================");

CUSTOMER_LOOP:
for (ProductStock listItem : custList) {

System.out.println("\n");

System.out.println("You want " + listItem.getQuantity() + " of " + listItem.getProd

for (ProductStock productStock : stock) {
Product p = productStock.getProduct();

if(p.getName().equals(listItem.getProduct().getName())){
int quantity = productStock.getQuantity();
double price = findPrice(p.getName());
double costToCustomer = price * listItem.getQuantity();
double remainingBudget = c.getBudget() - costToCustomer;

System.out.println("We have " + quantity +
" of " + p.getName() + " which will cost " + String.format("%.2f", price) + " each
System.out.println(listItem.getQuantity() + " of " + p.getName() + " will cost you

if(quantity >= listItem.getQuantity()){

if( costToCustomer <= c.getBudget()){
// Customer can afford item
System.out.println("You have " + String.format("%.2f", remainingBudget) + " left o
+ String.format("%.2f", c.getBudget()) + " budget");
// Reduce their budget accordingly
c.setBudget(remainingBudget);
// Reduce shop stock
productStock.setQuantity(quantity - listItem.getQuantity());
// Add to shop balance
cash = cash + costToCustomer;

continue CUSTOMER_LOOP;
```

```
} else {
// Customer cannot afford item
System.out.println("Sorry " + c.getName() +
", you don't have enough money to purchase " + listItem.getQuantity()
+ " of " + listItem.getProduct().getName() );
}
} else if (quantity == 0){
// No stock in shop
System.out.println("Sorry " + c.getName() +
", we are out of stock for " + listItem.getProduct().getName() );
continue CUSTOMER_LOOP;
}else {
// Not enough stock in shop
System.out.println("Sorry " + c.getName() +
", we don't have enough stock to fulfill your order ");
continue CUSTOMER_LOOP;
}
}
}
}
}
}
```

B.7 Runner Class

```
package ShopApp;

import java.util.List;
import java.util.Scanner;

public class Runner {

public static void main(String[] args) {

int queueCleared = 0;

Shop shop = new Shop("src/ShopApp/stock.csv");
Scanner scan = new Scanner(System.in);

System.out.println("Welcome to the shop, Whats your name?");
String customerName = scan.nextLine();

while (true){
Scanner scanWhile = new Scanner(System.in);

System.out.println("==========================================================");
```

```
System.out.println("| Please select an option:                    |");
System.out.println("| ---------------------------                |");
System.out.println("| Allow the customer queue to go ahead: [1]  |");
System.out.println("| Buy something yourself:               [2]  |");
System.out.println("| Leave the shop:                       [3]  |");
System.out.println("| Print shop details:                   [4]  |");
System.out.println("==============================================");

String answer = scanWhile.nextLine();

switch(answer){
case "1":
if( queueCleared > 0){
System.out.println("The customer queue has alredy been served\n");
} else {
List<String> fileList = new CustomerFileFinder().inputFileinDir("src/ShopApp/custon
for (String item : fileList) {
Customer nextCustomer = new Customer("src/ShopApp/customers/" + item);
// System.out.println(nextCustomer);
shop.processOrder(nextCustomer);
queueCleared = 1;
System.out.println("==============================================\n");
}
continue;
}

case "2":
// ask the user for what they want to buy and save as string
System.out.println("What product do you want to buy?");
List <ProductStock> currentStockList = shop.getStock();
for (ProductStock listItem : currentStockList) {
System.out.println(listItem.getProduct().getName());
}
System.out.println("==============================================\n");
String productName = scanWhile.nextLine();

// Ask how many they want and save as a integer
System.out.println("How many " + productName + " do you want?");
int amount = scanWhile.nextInt();

// find out how much money they have to pay you with
System.out.println("What's your budget today?");
double cash = scanWhile.nextDouble();

CustomerFileWriter fw = new CustomerFileWriter();
fw.toFile(customerName, cash, productName, amount);
Customer localCustomer = new Customer(customerName +".cust");
shop.processOrder(localCustomer);
continue;
```

```
case "3":
System.out.println("Thank you for your custom, please call again!\n Godbye \n");
scanWhile.close();
scan.close();
return;

case "4":
System.out.println(shop);
continue;

default:
scanWhile.close();
break;
}
scan.close();
}
}
}
```

Bibliography

[1] E. Programming, *C Programming Language: The Ultimate Beginner's Guide*. Scotts Valley, California, United States: CreateSpace Independent Publishing Platform, 2016.