Assignment 8

Samuel Godwin, Computer Science BSc, K1630575

December 5, 2016

1 Introduction

The purpose of this assignment is to create a program in which different types of collections are used in order to model the scenario of shopping for many items. A customer is to be able to shop by adding a selection of products to their shopping basket from the collection of products available in the shop. As well as removing an item from their basket again, a customer may also purchase all of the items in their basket, a process in which a transaction of 'GoldCoin's takes place - a class I created in last week's assignment.

(Pseudocode on p2).

2 Pseudocode

GoldCoin Class

```
CLASS 'GoldCoin':
        Create final int field 'coinNumber'
        Create static int field 'numberOfCoins'
        METHOD 'GoldCoin':
                coinNumber = numberOfCoins++
        END METHOD.
        METHOD 'readCoin':
                Return coinNumber
        END METHOD.
END CLASS.
```

Product Class

Import ‘TreeMap’

CLASS ‘Product’:

    Create String field ‘name’

    Create int field ‘price’

    Create TreeMap<String, GoldCoin> ‘itemPrices’

    METHOD ‘Product' <name, price>:

        Set field ‘name’ to <name>

        Set field ‘price’ to <price>

        Initialise TreeMap ‘itemPrices’ as new TreeMap<String, GoldCoin>

    END METHOD.

    METHOD ‘getName’:

        Return name

    END METHOD.

    METHOD ‘getPrice’:

        Return price

    END METHOD.

    METHOD ‘toString’:

        Return name + price

    END METHOD.

END CLASS.

Shop Class

Import ‘TreeMap’

Import ‘ArrayList’

CLASS ‘Shop’:

Create String field ‘name’

Create TreeMap<String, Product> ‘products’

Create Product field ‘productToRemove’

Create ArrayList<GoldCoin> ‘coinBox’

Create TreeMap<String, Integer> ‘customerTotalSpend’

METHOD ‘Shop’ <name, coins>:

Set field ‘name’ to <name>

Initialise TreeMap ‘products’ as new TreeMap<String, Product>

Initialise ArrayList ‘coinBox’ as new ArrayList<GoldCoin>

Create ‘coins’ number of new ‘GoldCoin’s in ‘coinBox’

Initialise TreeMap ‘customerTotalSpend’ as new TreeMap<String, Integer>

END METHOD.

METHOD ‘addProduct’ <product>:

Put result of ‘getName’ in <product> into ‘products’ TreeMap, mapped to <product>

END METHOD.

METHOD ‘removeProduct’ <product>:

Set field ‘productToRemove’ to <product>

Remove item mapped to ‘getName’ in <product> from ‘products’ TreeMap

IF result of ‘getName’ in <product> == null THEN:

Return productToRemove

ELSE:

Print “Product was not properly removed”

Return null

END METHOD.

METHOD ‘searchProduct’ <s>:

Return item mapped to <s> in ‘product’

END METHOD.

METHOD ‘addGoldCoin’ <coin>:

Add <coin> into ‘coinBox’ ArrayList

END METHOD.

METHOD ‘updateTotalSpend’ <customer, amountOfCoins>:

IF <customer> in ‘customerTotalSpend’ TreeMap == null THEN:

Put result of ‘getName’ in <customer> into ‘customerTotalSpend’ TreeMap, mapped to <amountOfCoins>

ELSE:

Create int variable ‘temp’ = integer mapped to ‘getName’ in <customer> + <amountOfCoins>

Put result of ‘getName’ in <customer> into ‘customerTotalSpend’ TreeMap, mapped to ‘temp’

END METHOD.

METHOD ‘toString’:

Return name + size of ‘coinBox’ ArrayList + result of ‘toString’ in ‘products’

END METHOD.

METHOD ‘getName’:

Return name

END METHOD.

END CLASS.

Shop Customer

Import ‘TreeMap’

Import ‘ArrayList’

CLASS ‘Customer’:

Create String field ‘name’

Create ArrayList<Product> ‘shoppingBasket’

Create Product field ‘productToRemove’

Create Product field ‘productSearch’

Create TreeMap<String, Product> ‘ownedProducts’

Create ArrayList<GoldCoin> ‘purse’

Create int field ‘valueOfBasket’

Create int field ‘coinsSpent’

METHOD ‘Customer’ <name, coins>:

Set field ‘name’ to <name>

Initialise ArrayList ‘shoppingBasket’ as new ArrayList<Product>

Initialise TreeMap ‘ownedProducts’ as new TreeMap<String, Product>

Initialise ArrayList ‘purse’ as new ArrayList<GoldCoin>

Create ‘coins’ number of new ‘GoldCoin’s in ‘purse’

END METHOD.

METHOD ‘addToShoppingBasket’ <product>:

Add <product> into ‘shoppingBasket’ ArrayList

Set field valueOfBasket = result of ‘getPrice’ in <product>

END METHOD.

METHOD ‘removeFromShoppingBasket <product>:

Set field productToRemove = <product>

Remove <product> from ‘shoppingBasket’ ArrayList

IF result of ‘getName’ in <product> == null THEN:

Set field valueOfBasket -= result of ‘getPrice’ in <product>

Return productToRemove

ELSE:

Print “Product was not properly removed”

```
            Return null
END METHOD.
METHOD 'searchShoppingBasket' <suppliedName>:
    Set field productSearch = null
    For each product in 'shoppingBasket':
        IF (suppliedName == result of 'getName' in product)
            productSearch = product
    Return productSearch
END METHOD.
METHOD 'addOwnedProduct <product>:
    Put result of 'getName' in <product> into 'ownedProducts' TreeMap, mapped to
                                                                    <product>
END METHOD.
METHOD 'addCoin' <coin>:
    Add <coin> into 'purse' ArrayList
END METHOD.
METHOD 'addCoin' <coin>:
    Add <coin> into 'purse' ArrayList
END METHOD.
METHOD 'purchaseProducts <shop, customer>:
    Create variable 'n' = 'valueOfBasket'
    IF (valueOfBasket > size of 'purse') THEN:
        Print "Purchase failed"
        Return false
    ELSE:
        Set field coinsSpent += valueOfBasket
        While (n != 0):
            Add GoldCoin to 'coinBox' in 'shop'
            Remove GoldCoin from 'purse'
            n --
        Add items in 'shoppingBasket' to 'OwnedProduct' TreeMap
```

```
                Remove items in 'shoppingBasket' from 'shoppingBasket'
                Go to 'updateTotalSpend' in 'shop',     set customer = <customer>,
                                                        set amountOfCoins = 'coinsSpent'
                Set field valueOfBasket = 0
                Return true
        END METHOD.
        METHOD 'getName':
                Return name
        END METHOD.
        METHOD 'toString':
                Return name + size of 'purse' ArrayList + result of 'toString' in 'shoppingBasket' +
                        valueOfBasket + result of 'toString' in 'ownedProducts' + coinsSpent
        END METHOD.
END CLASS.
```

ShoppingTrip Class

Import ‘Scanner’

CLASS ‘ShoppingTrip’:

METHOD ‘main’:

Create instance of ‘Scanner’ called ‘in’

Create object of ‘Product’ called ‘product1’<”Diamond”, 40>
Create object of ‘Product’ called ‘product2’<”Crown Jewels”, 100>,
Create object of ‘Product’ called ‘product3’<”Silver Locket”, 60>,
Print result of ‘toString’ in ‘product1’
Print result of ‘toString’ in ‘product2’
Print result of ‘toString’ in ‘product3’
Create object of ‘Shop’ called ‘shop’<”Hidden Hideaway”, 125>
Add ‘product1’ to ‘shop’
Add ‘product2’ to ‘shop’
Add ‘product3’ to ‘shop’
Print result of ‘toString’ in ‘shop’

Create object of ‘Customer called ‘customer’<”Blackbeard”, 100>

Print result of ‘toString’ in ‘customer’

Print welcome message

Create String variable ‘userInput’ = “”

Do:

Print “Enter an input”

Print result of ‘toString’ in each of ‘shop’ and ‘customer’

userInput = take user input

IF (userInput = “add product”) THEN:

Print “Enter a product to add to your basket”

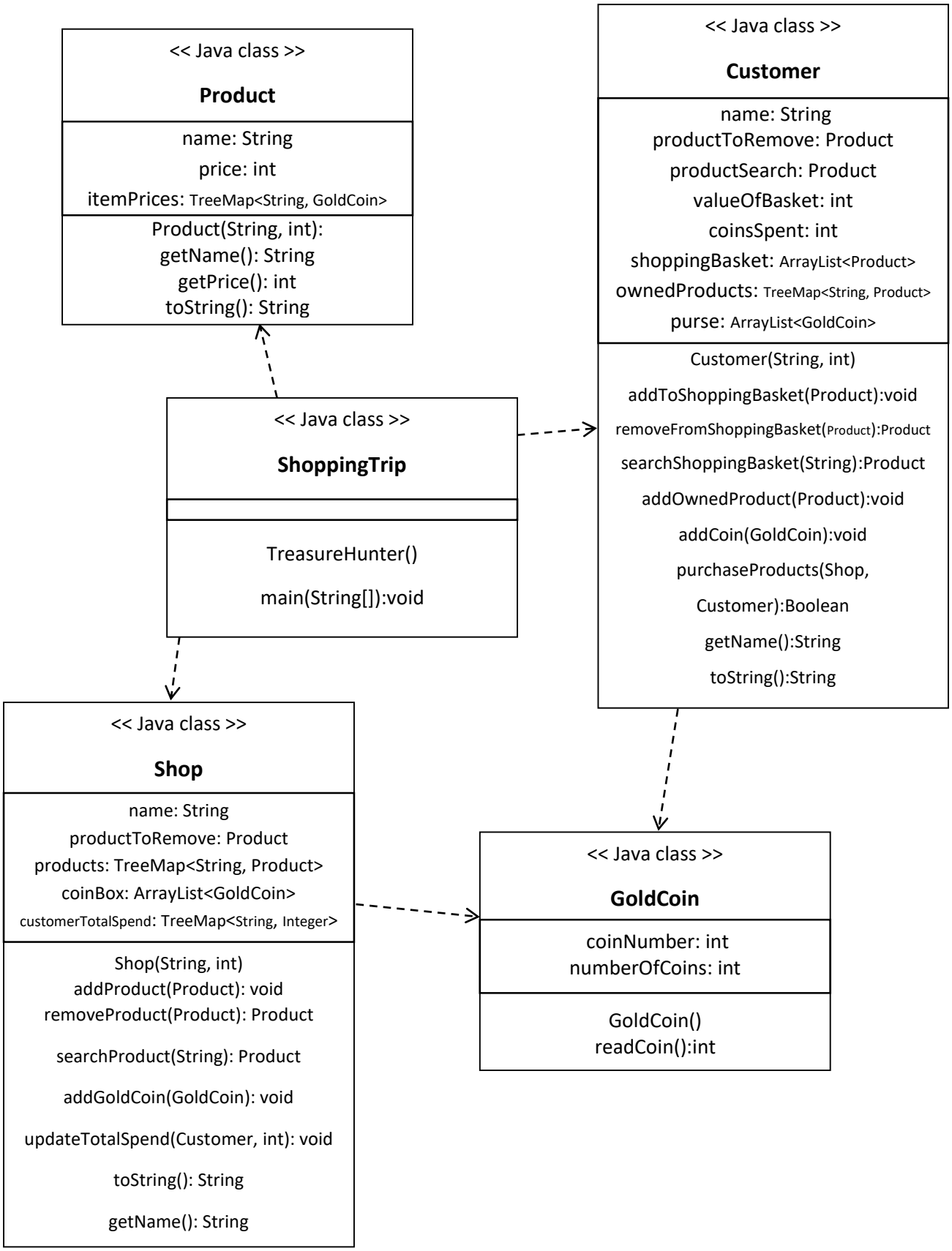
Create String variable ‘productToAdd’ = take user input

Add product to ‘shoppingBasket’ in ‘customer’ if it exists in ‘shop’

Remove product from shop

```
                ELSE IF (userInput = “remove product”) THEN:
                        Print “Enter a product to remove from your basket”
                        Create String variable ‘productToRemove’ = take user input
                        Remove product from ‘shoppingBasket’ if it exists in ‘shoppingBasket’
                        Remove product from ‘shoppingBasket’ in ‘customer’
                ELSE IF (userInput = “purchase”) THEN:
                        Go to ‘purchaseProducts’ in ‘customer’, set <shop> = shop,
                                                                set <customer> = customer
                ELSE:
                        Print “Please enter a valid input”
        While (userInput != “exit”)
        Call the ‘close’ method on my ‘Scanner’ instance
    END METHOD.
END CLASS.
```

3 Class Diagram



4 Description

My solution to this assignment involves the creation of a program in which my driver class 'ShoppingTrip', which contains my 'main' method, creates objects of three of my other classes, 'Shop', 'Product' and 'Customer'. It does not, however, create any objects of 'GoldCoin' as these are created and handled within each of the classes 'Shop' and 'Customer'. I initially import 'Scanner' for scanning of user input in my driver class and then, in my 'main' method, I create a 'Scanner' object and then three 'Product' objects. These 'Products's are of varying names and prices - which I pass as I create the objects - since this information is handled, and the corresponding fields set, within the constructors for these classes. I make an object of 'Shop' called shop and pass it a name "Hidden Hideaway" and an initial number of coins for its 'coinBox'. In 'Shop' and 'Customer', when an integer value is passed as a parameter to the constructor of either class, a 'for' loop I built creates the appropriate number of GoldCoins, ultimately increasing the number of GoldCoins and giving each one their own unique 'coinNumber'.

Next, in my driver class, I call the 'addProduct' through my shop object three times, passing the product instances made previously and thereby adding each of these three products to the shop stock. Finally, I create a 'Customer' object called 'customer' and pass it the string value "Blackbeard" and integer value 100, setting its name and number of coins. This number is for 'purse' in 'customer', which is an ArrayList of GoldCoins. Before a do-while loop, I set my 'userInput' field in my driver class simply to "" - an empty string. This is due to the fact it does not have a default value.

At this point in my driver class I build a do-while loop, initially asking the user for input. I also display the information from within 'shop' and 'customer', the details of which I will discuss later.

Although my 'Product' class also contains one of the following types of collections, I will elaborate on my collections mainly in terms of their use within my 'Shop' and 'Customer' classes. Both of these classes contains collections of either the 'TreeMap' or 'ArrayList' type. 'Shop' contains a TreeMap mapping String values to 'Product' instances, and this TreeMap is called 'products'. It also contains a TreeMap called 'customerTotalSpend', mapping String values to integer values, called 'customerTotalSpend'. Additionally, 'Shop' contains an ArrayList called coinBox, which is a collection of instances of my class 'GoldCoin'. Similarly, my 'Customer' class also has an ArrayList, 'purse'. Which is a collection of instances of my 'GoldCoin' class. This class contains a second ArrayList, 'shoppingBasket', which is a collection of instances of my 'Product' class, as touched upon earlier when I spoke about the creation of my objects within my driver class. Lastly, this class contains a TreeMap, 'ownedProducts', which maps String values to Product instances.

In each of 'Shop' and 'Customer', I include a 'toString' method in which I return the above collections, as well as some of the other fields (such as 'name') in the classes.

I take user input in my do-while loop via my 'Scanner' object. This is stored in my 'userInput' string variable created earlier, which I then compare, using '.equals', to each of the possible scenarios that I already have code ready for. These include whether the user input is equal to "add product", in which case I add the product which corresponds to the user input to the customer's 'shoppingBasket' ArrayList and, subsequently, remove it from 'products' in my shop object.

An element of my code I am particularly impressed with is how I maintained use of a TreeMap type for my 'products' collection, even though I struggled to implement it and use its syntax correctly initially. In my initial implementation of my program, I tried using TreeMap as my collection type for

‘products’ with the intention of making searching for a product (via its name) easier – however, I ran into numerous errors. I later worked through and fixed each of these issues.

What made my success here particularly useful is the ability, for example, of a user using my program to be able to retrieve a ‘Product’ object simply by searching the string value associated with it according to its mapping (i.e. its name). This is possible in a particularly efficient number of lines of code. It spared me the need for the alternative option, also, of searching through the ‘name’ field of every object in my ‘products’ collection testing whether each name matches the user’s string input. An example of exactly where this comes in useful is my ‘searchProduct’ method in my ‘Shop’ class, which has a String value passed to it, as a parameter.

The condition for my do-while loop is for it to repeat so long as the string input entered by the user does not equal “exit” – this will end the loop. I chose to implement a do-while loop rather than a while loop as no matter what, I always want the contents inside of the loop to execute at least once. At the end of my ‘main’ method, I close my Scanner to avoid an error.