MODULE NAME:	OBJECT ORIENTED DEVELOPMENT WITH JAVA
MODULE CODE:	CT038-3-2-OODJ
ASSIGNMENT TYPE:	INDIVIDUAL ASSIGNMENT
INTAKE CODE:	UC2F1711SE
HAND IN DATE:	5 MARCH 2018
LECTURER NAME:	DR. KADHAR BATCHA NOWSHATH
STUDENT ID:	TP040205
STUDENT NAME:	MUKABAK ORAZBEK

CONTENTS

INTRODUCTION

Alisdair stated that object-oriented programming languages have played enormous popularity since the 1990s, such that the object has largely overtaken the procedure and the module as the primitive means of breaking up large software systems (Alisdair, 2007). Historically, a program has been developed as a single logical procedure that takes input data, processes values, and generates output data.

Nowadays, developers implemented object-oriented techniques in variety computing areas, such as maintaining bank transactions, analysing stored data, designing video games, and developing desktop applications. The main advantage of object-oriented programming is it allows the developers to reuse objects and generates it during the runtime.

In this assignment, students are designed Retail Order Management System for small retailers. And there are two different stockholders, admin and customer. Admin has an ability to manage customers, manage products and manage orders. Based on the system requirements, customer can only manage orders.

UML DIAGRAM

Use Case Diagram

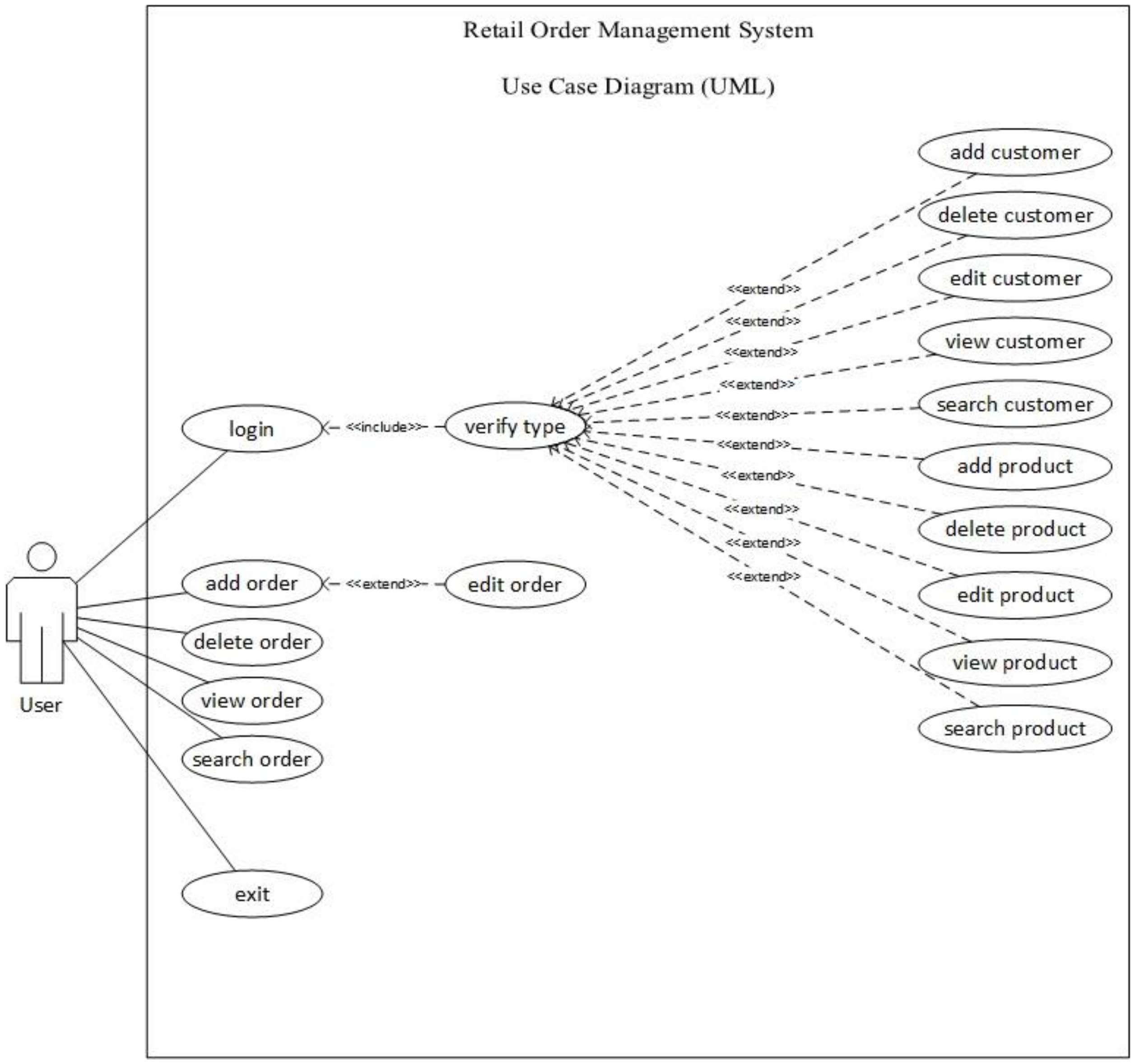


Figure 1: The use-case-diagram at the above used for representing Retail Order Management System.

The case scenario represented that there are two different types of users, and both of them have an ability to manage orders, such as add order, delete order, view order and search order. If user is an admin it means there are additional functionalities, namely manage products and manage orders. There is a login function in this use case diagram, and function is responsible to check login credentials and identify user types.

Class Diagram

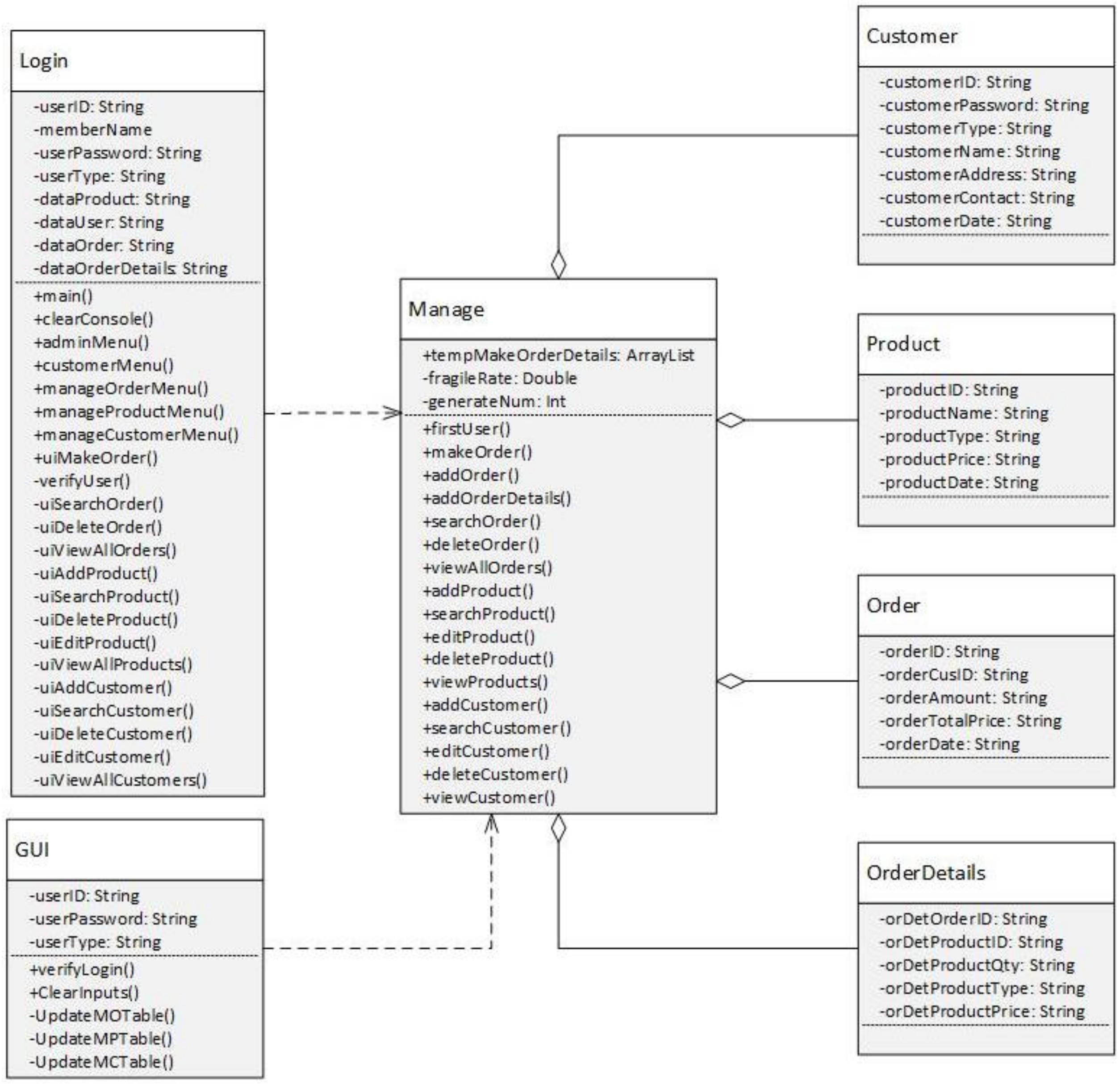


Figure 2: The class diagram at the above used for representing class relationships of Retail Order Management System.

Within the system, there are totally 6 different classes. And 4 classes are used to describe objects, namely customer object, product object, order object, order details object. In addition, all of the object behaviors or object related methods are located in manage class, as well as login class is dependent to manage class and responsible to support system user interactions.

Activity Diagram

Verify Login Function

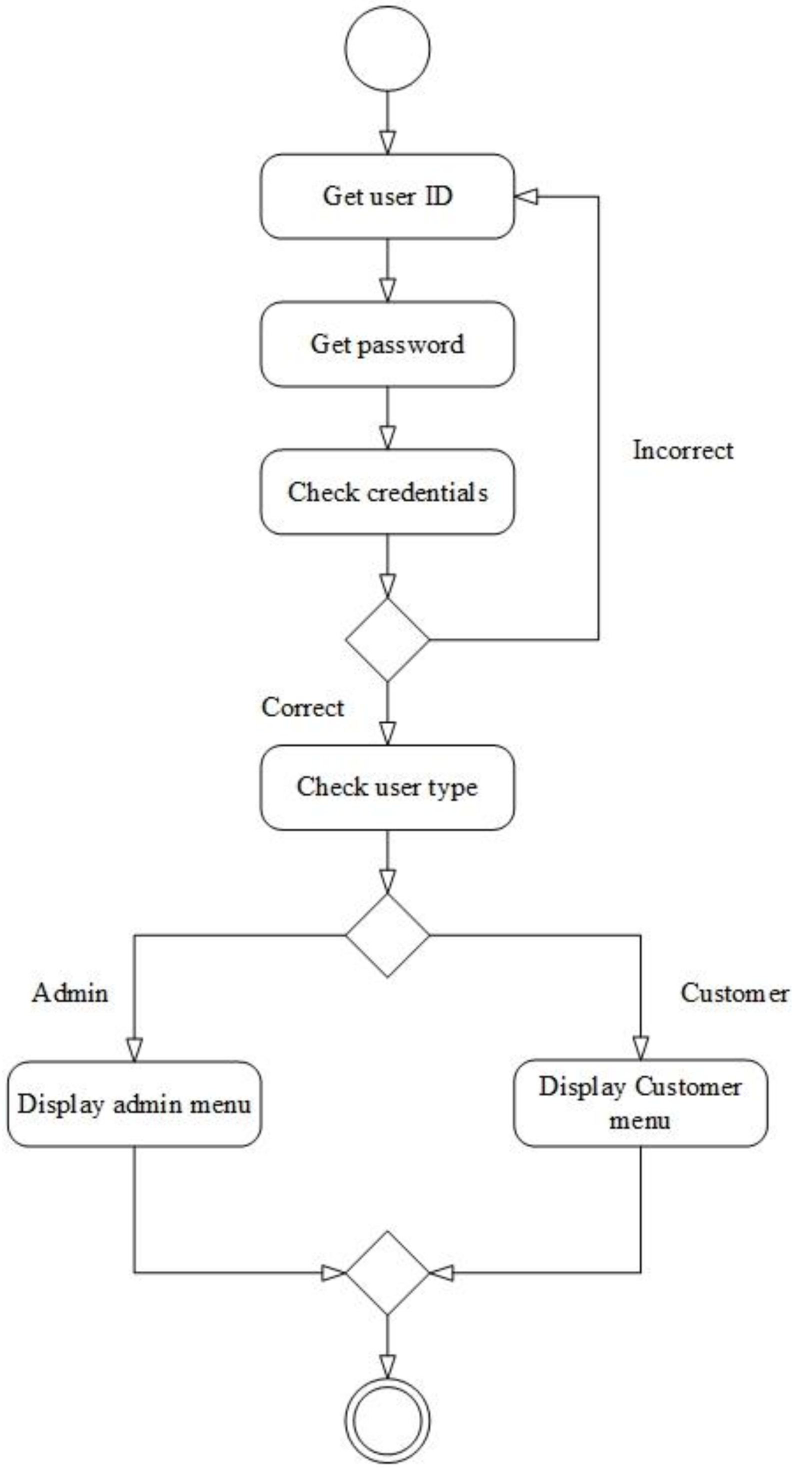


Figure 3: The activity diagram at the above represents verify login function.

Make Order Function

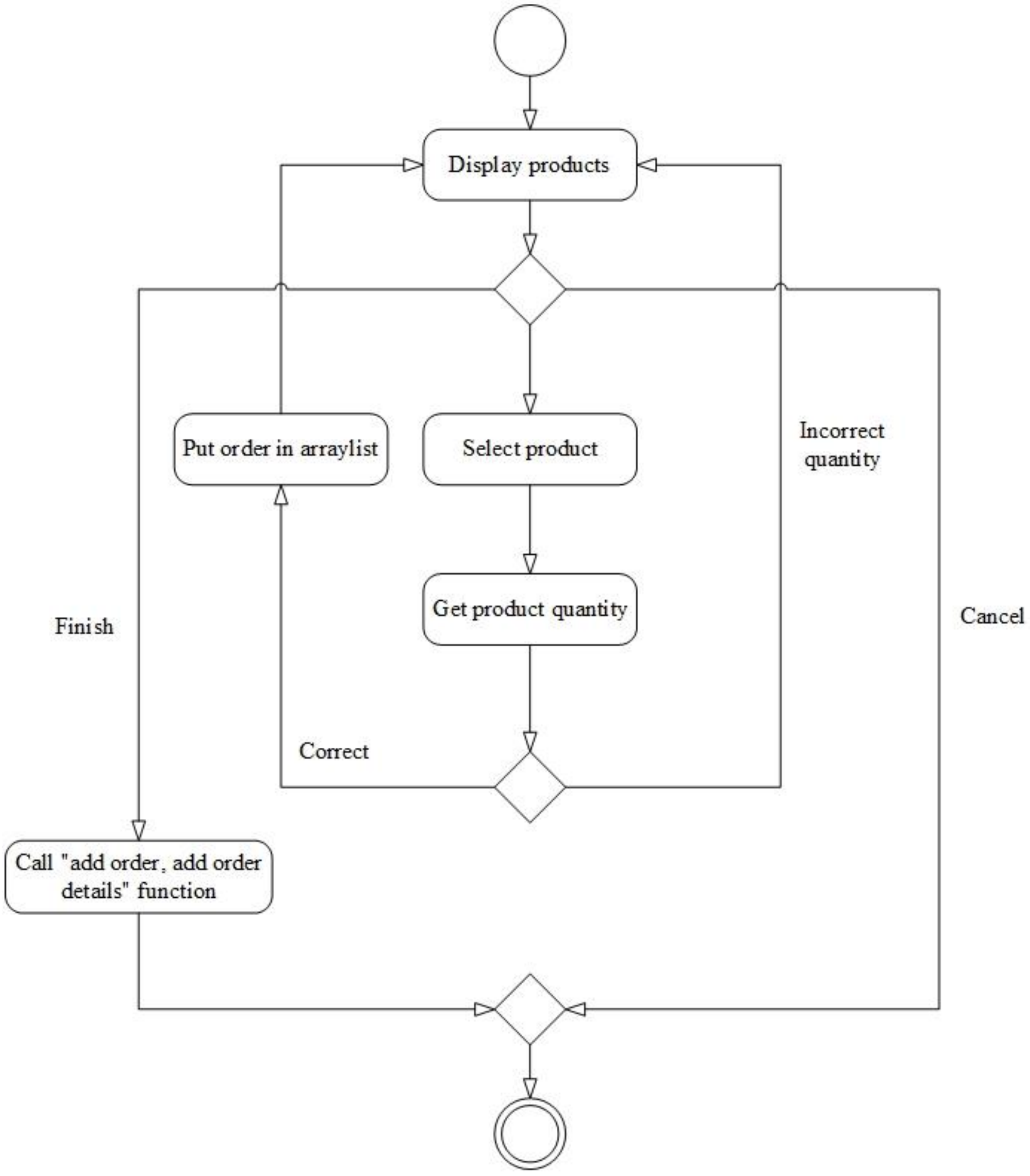


Figure 4: The activity diagram at the above used for representing make order function.

Search Order Function

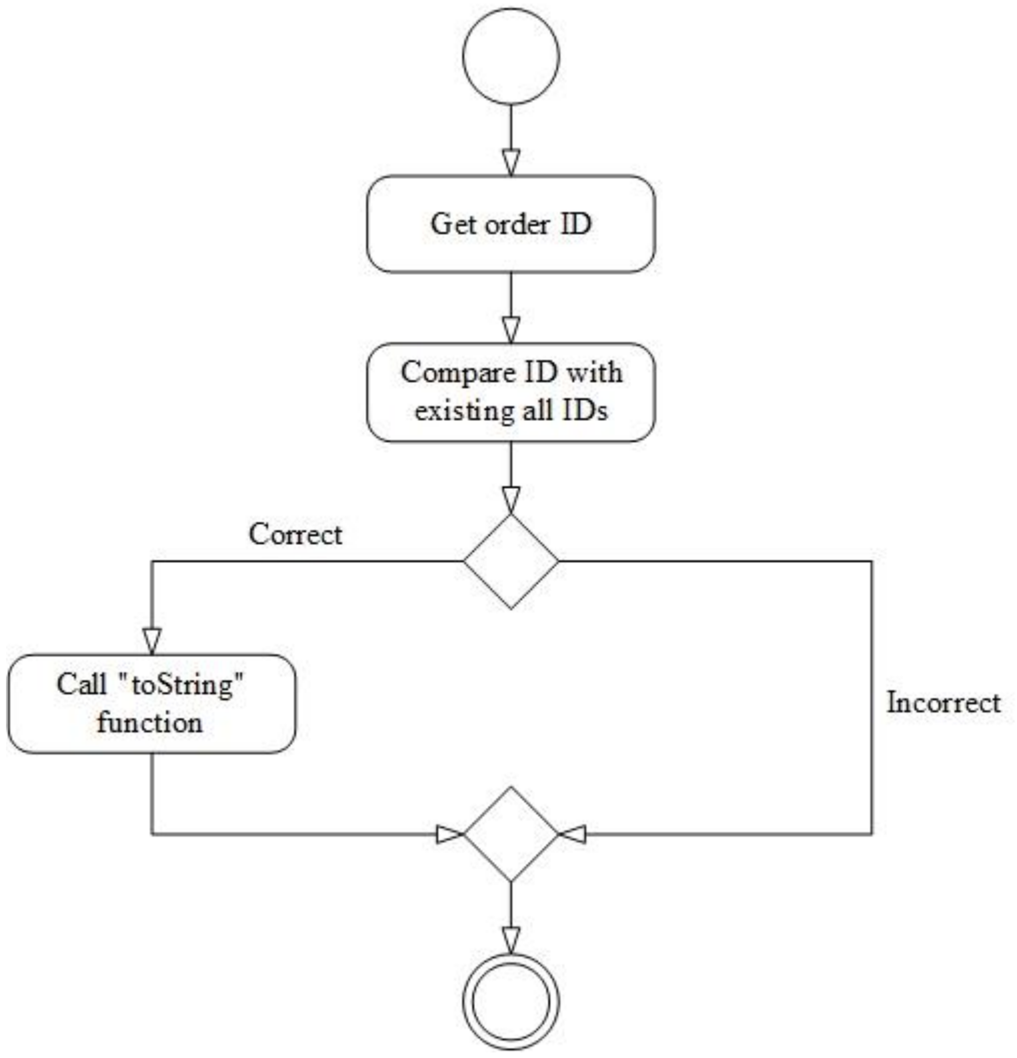


Figure 5: The activity diagram at the above used for representing search order function.

Delete Order Function

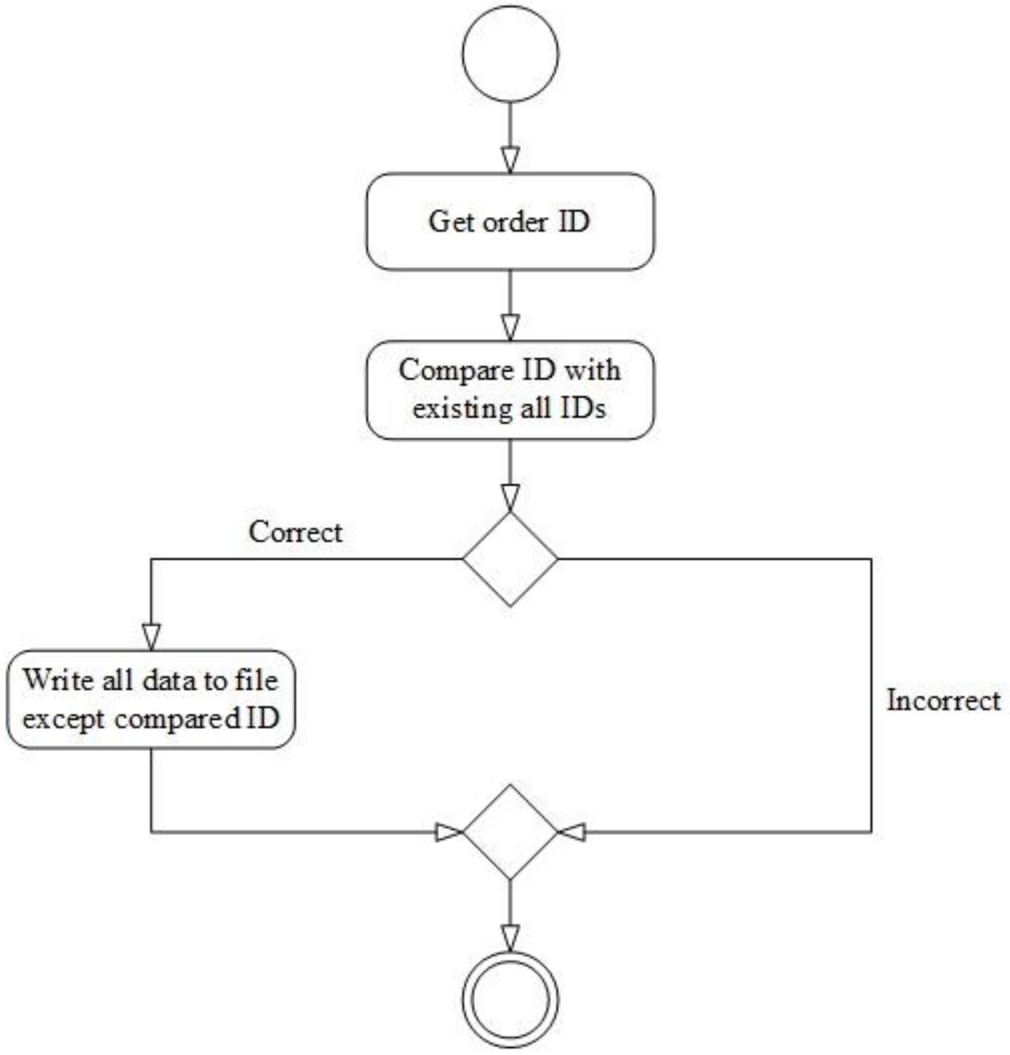

Get order ID
Compare ID with existing all IDs
Correct
Incorrect
Write all data to file except compared ID

Figure 6: The activity diagram at the above used for representing delete order function.

View All Orders Function

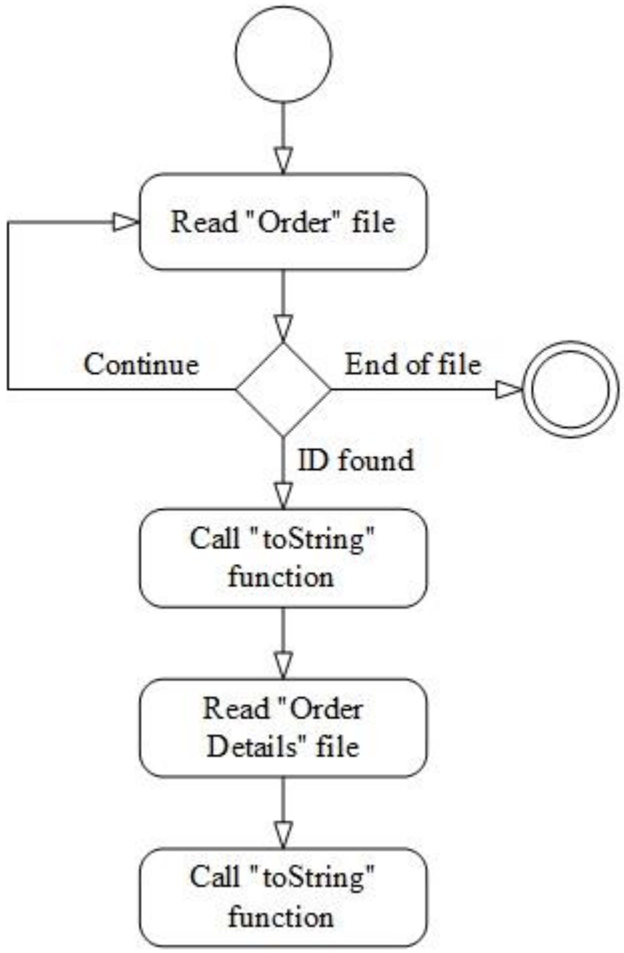


Figure 7: The activity diagram at the above used for representing view all orders function.

Add Product Function

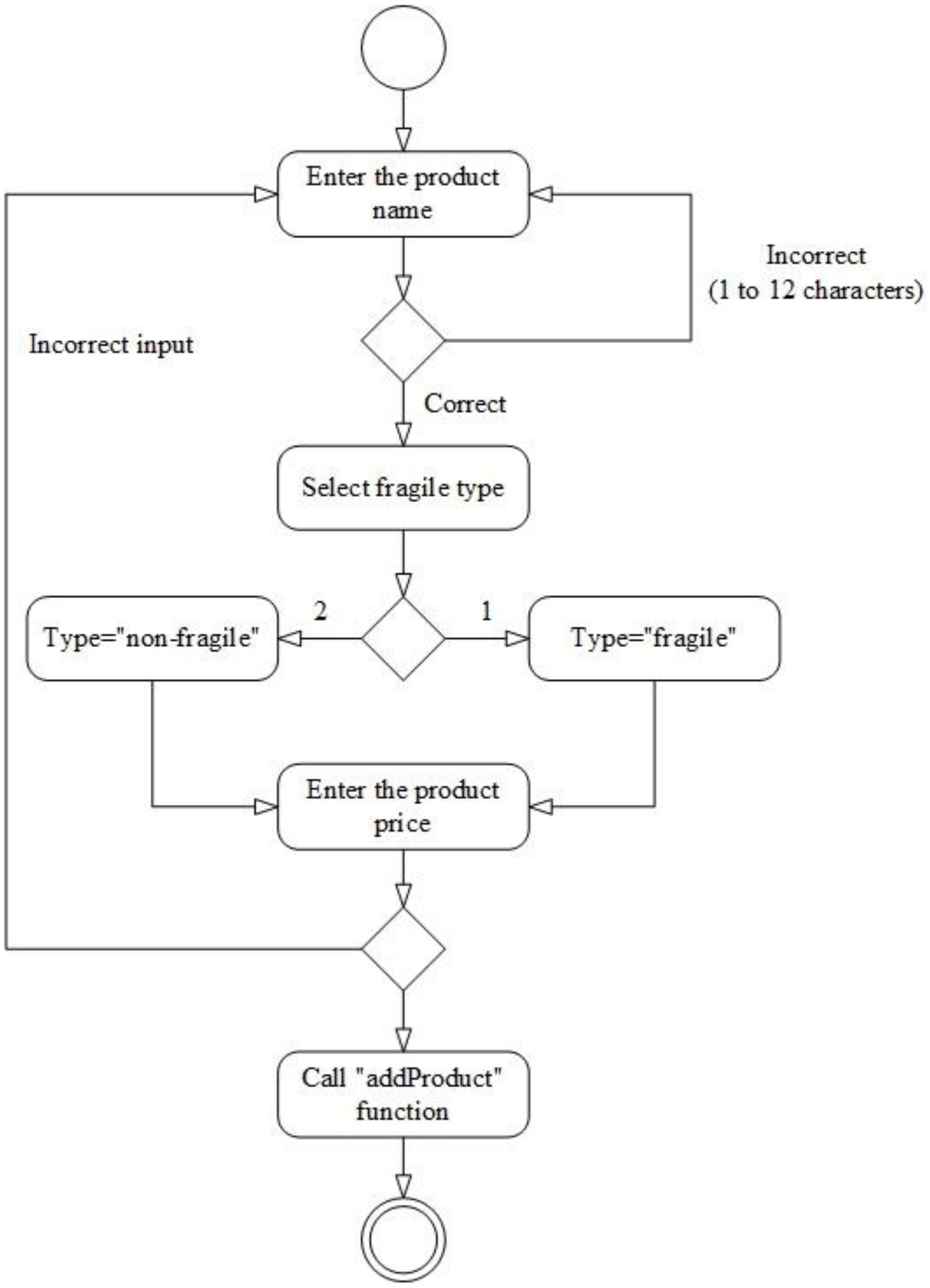


Figure 8: The activity diagram at the above used for representing add product function.

Search Product Function

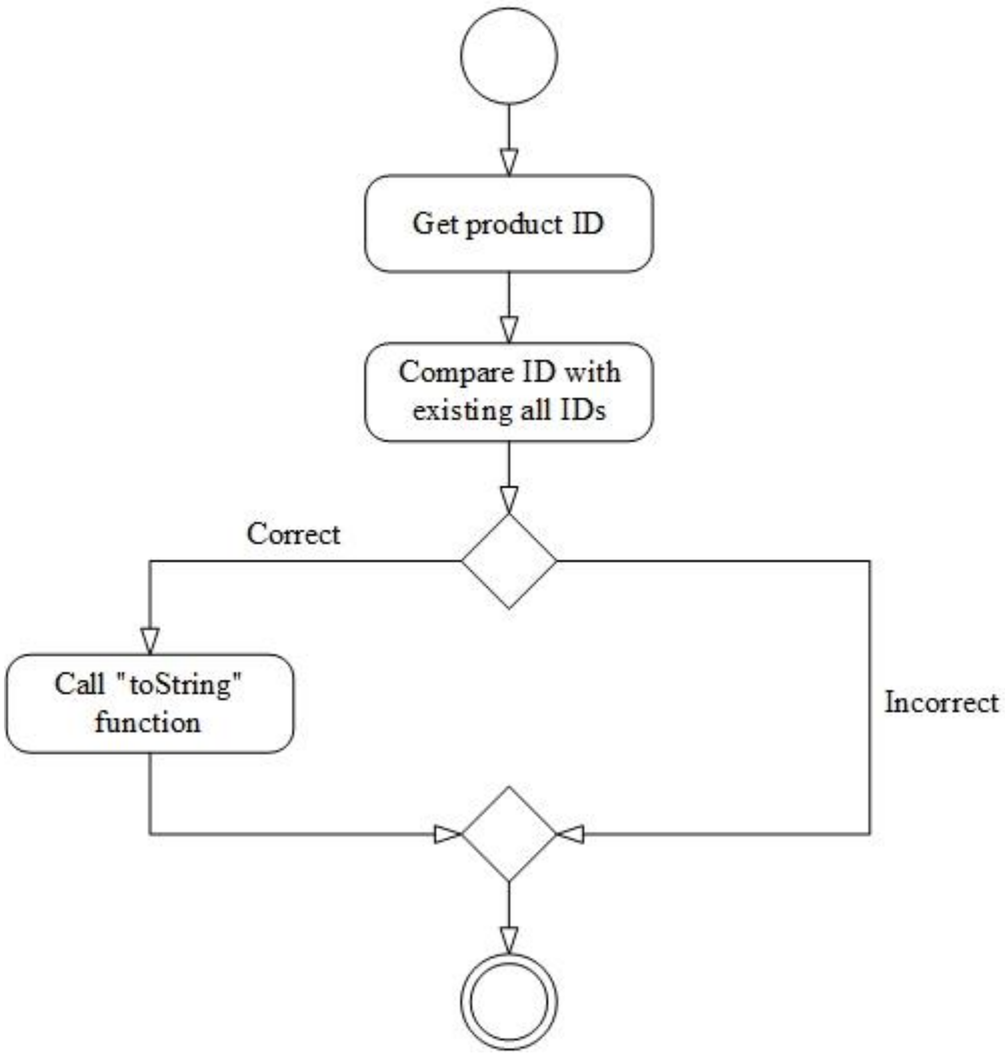


Figure 9: The activity diagram at the above used for representing search product function.

Edit Product Function

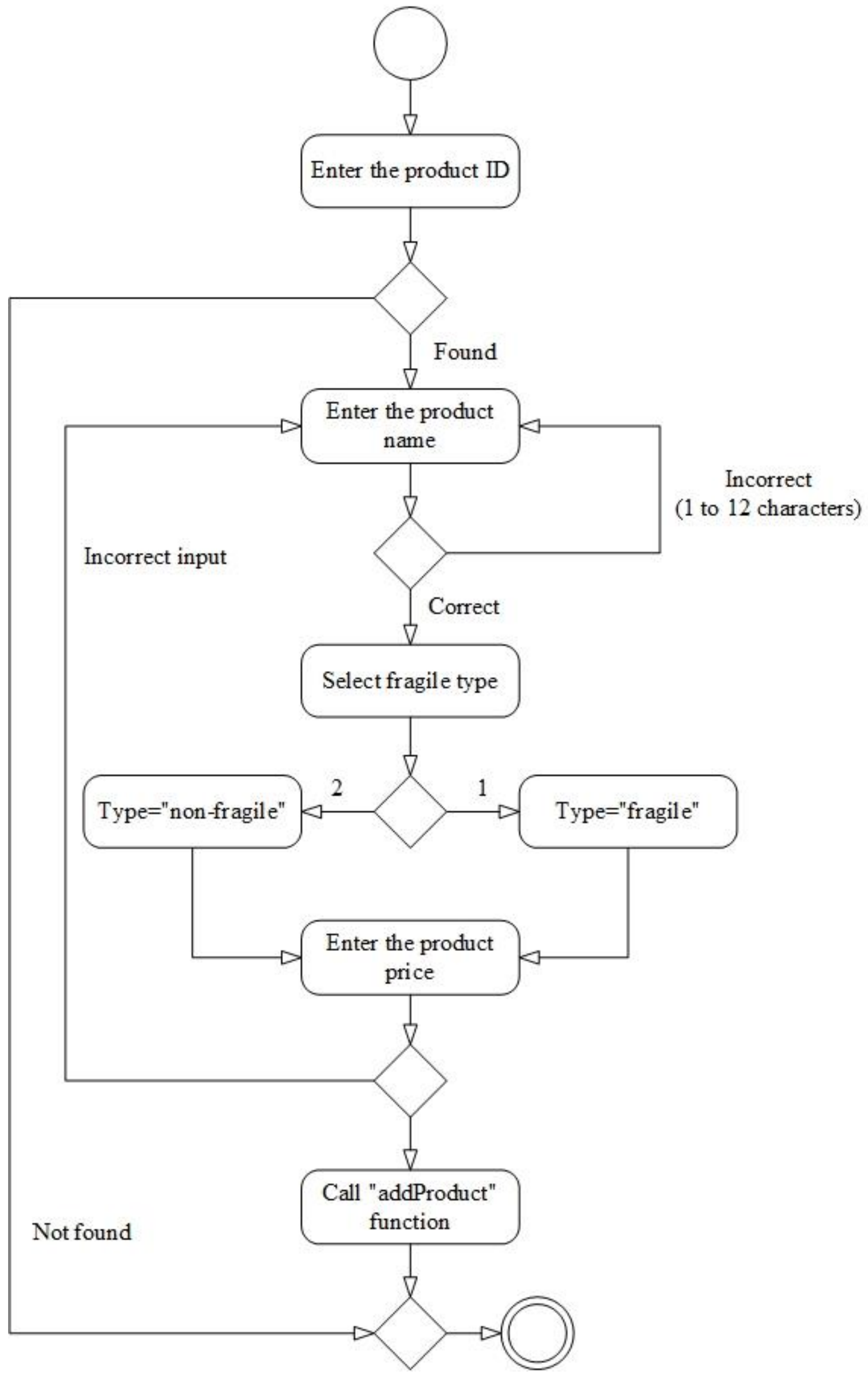


Figure 10: The activity diagram at the above used for representing edit product function.

Delete Product Function

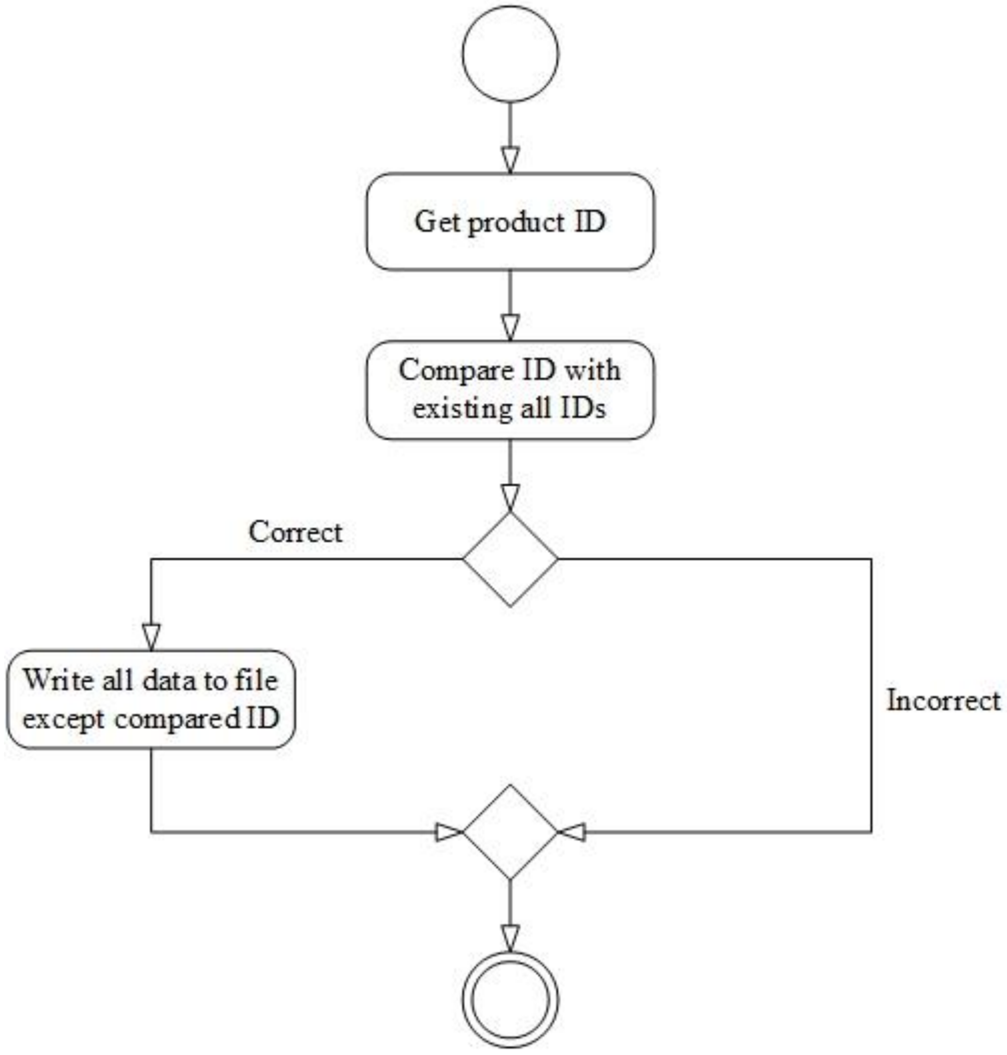

Get product ID
Compare ID with existing all IDs
Correct
Write all data to file except compared ID
Incorrect

Figure 11: The activity diagram at the above used for representing delete product function.

View All Products Function

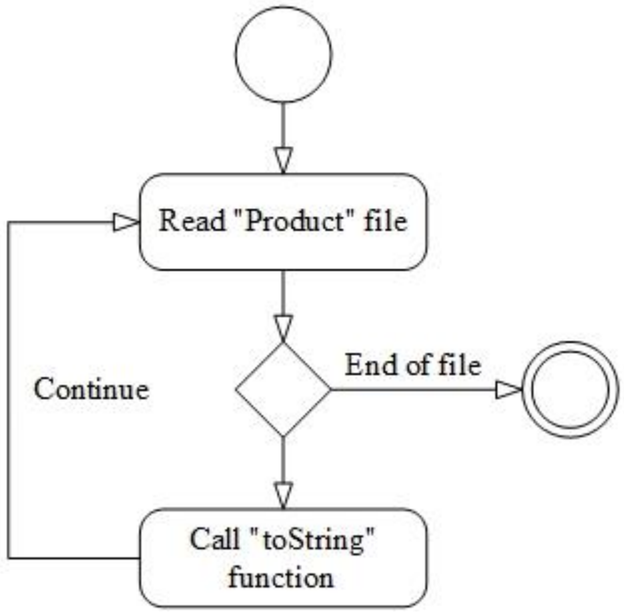


Figure 12: The activity diagram at the above used for representing view all products function.

Add Customer Function

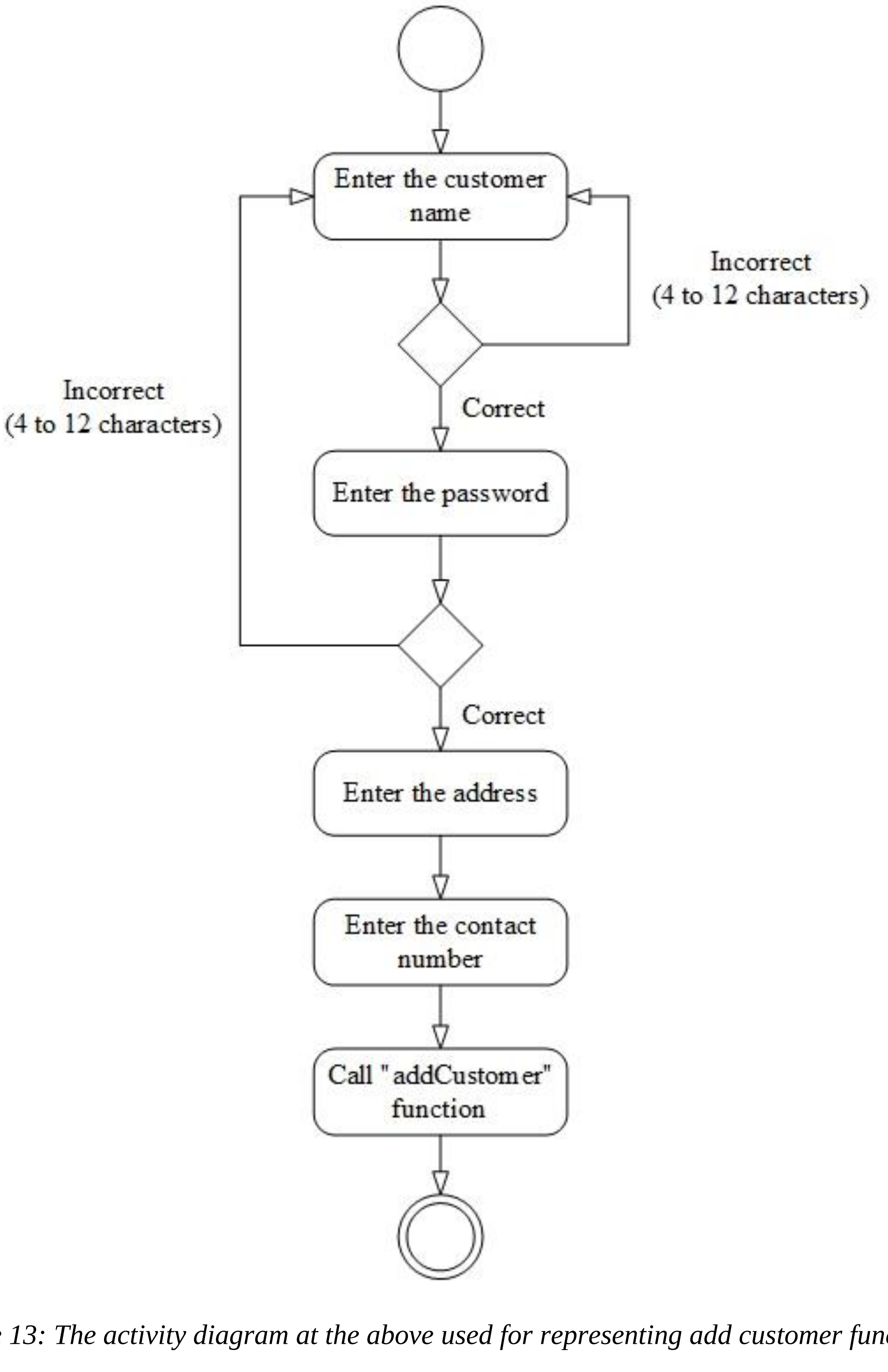


Figure 13: The activity diagram at the above used for representing add customer function.

Search Customer Function

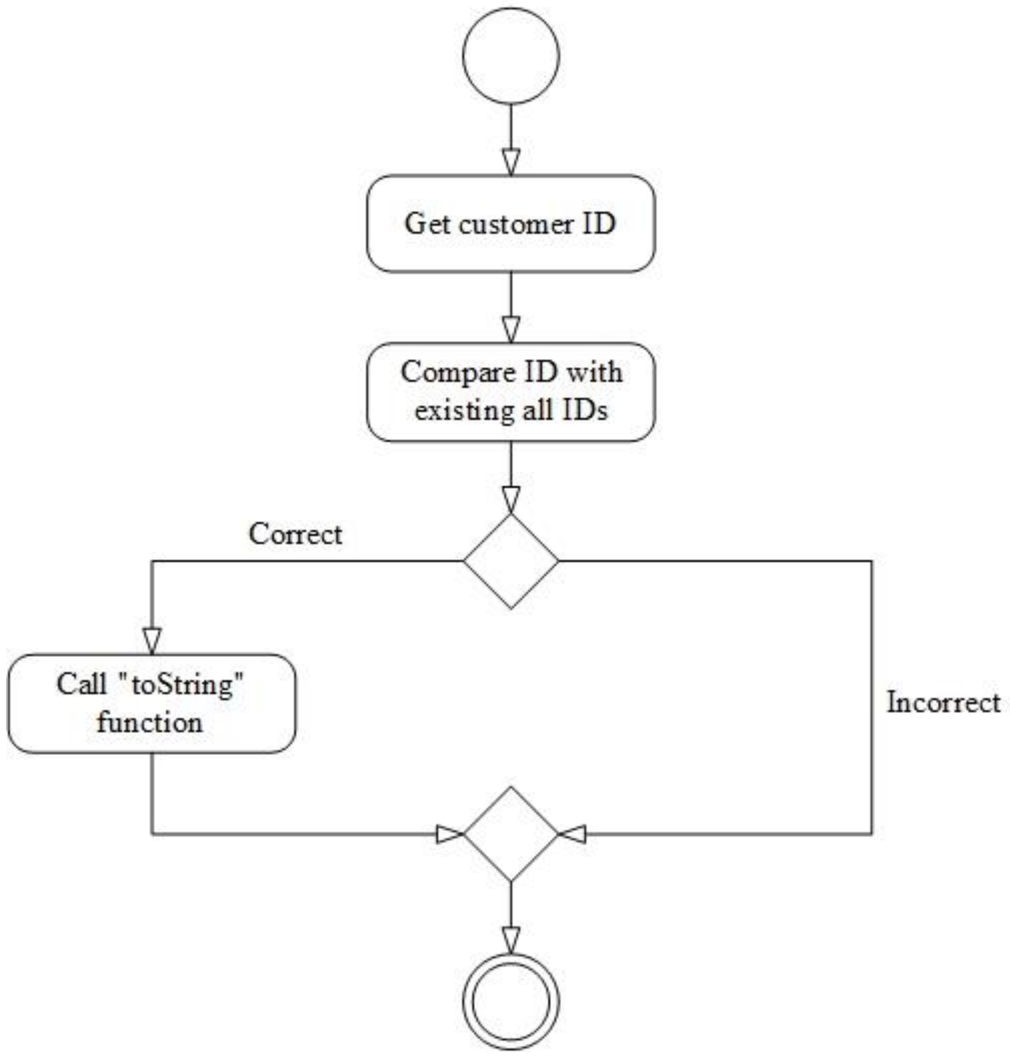


Figure 14: The activity diagram at the above used for representing search customer function.

Edit Customer Function

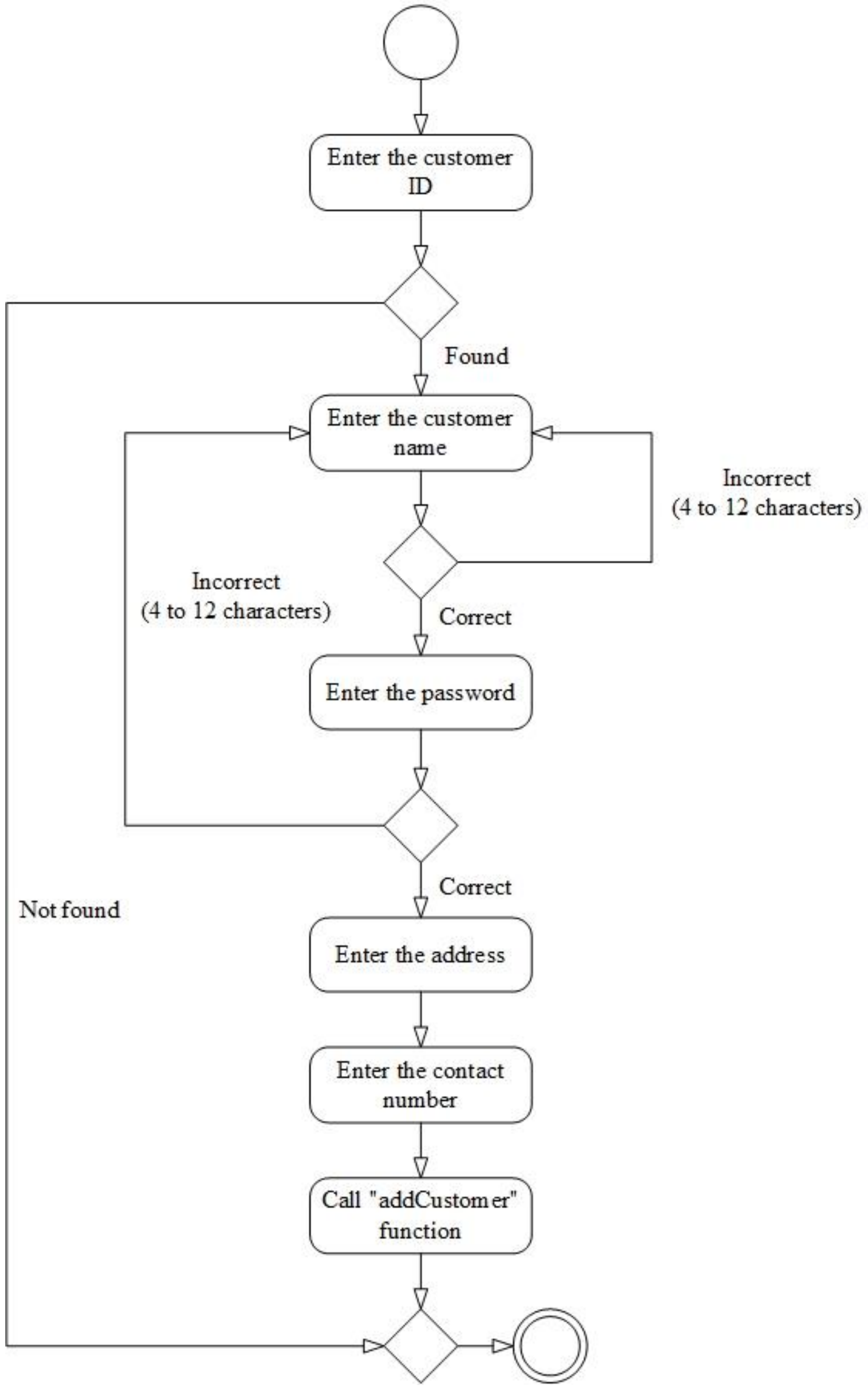


Figure 15: The activity diagram at the above used for representing edit customer function.

Delete Customer Function

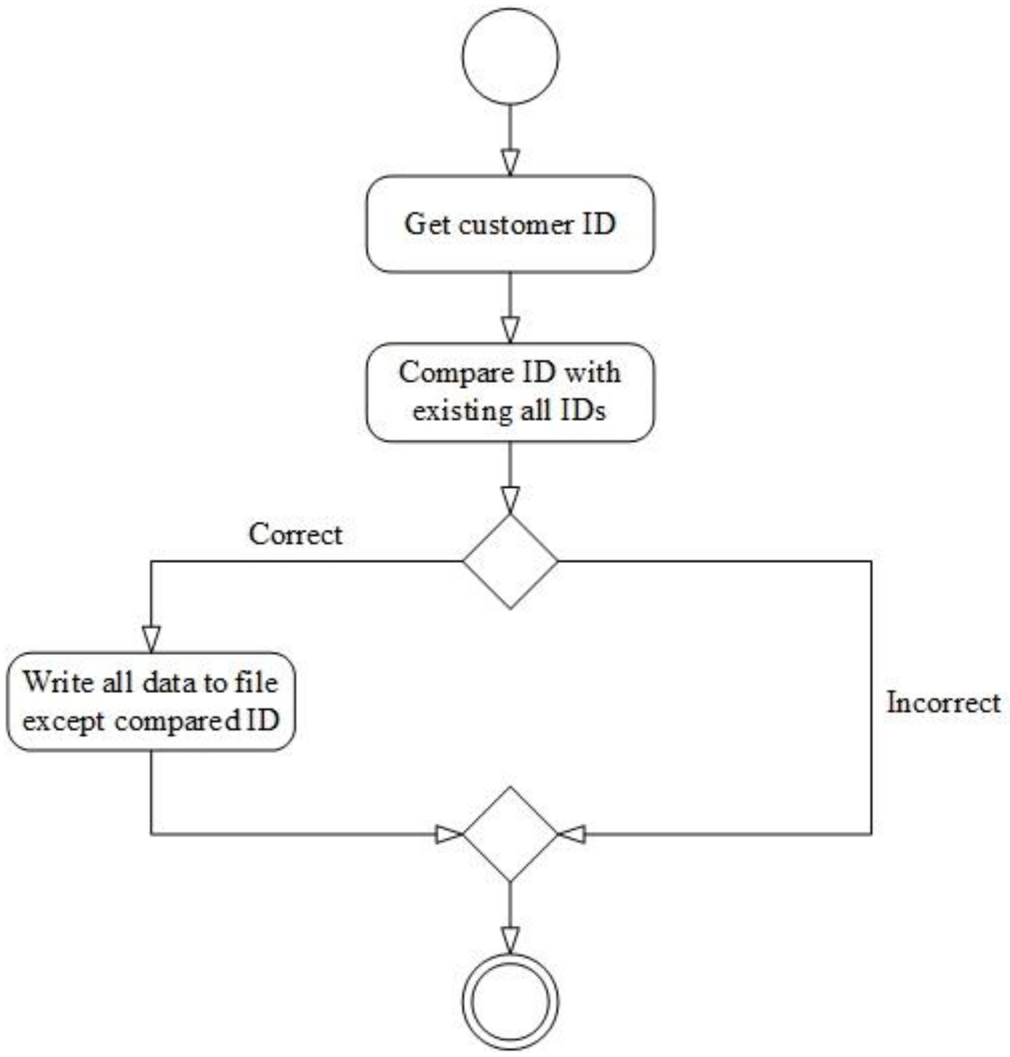

Get customer ID
Compare ID with existing all IDs
Correct
Write all data to file except compared ID
Incorrect

Figure 16: The activity diagram at the above used for representing delete customer function.

View All Customers Function

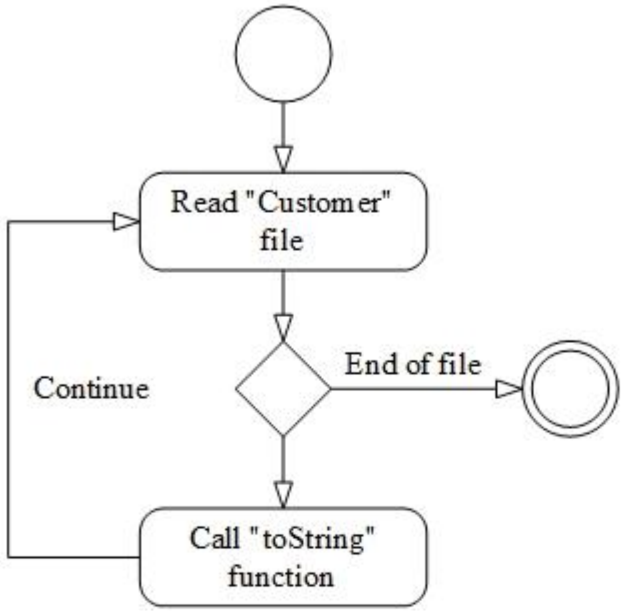


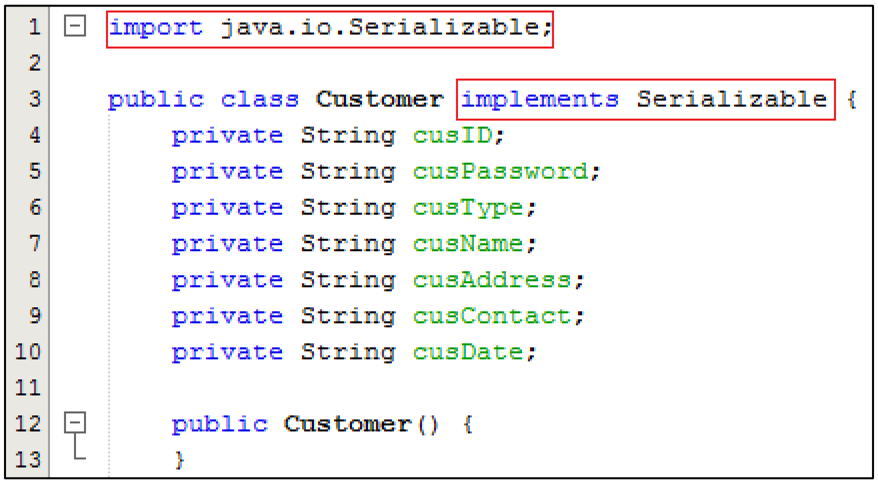
Figure 17: The activity diagram at the above used for representing view all customers function.

IMPLEMENTED TECHNIQUES

Serialization & Deserialization

The process of converting an object into a collection of bytes is called as serialization, and converted data can be stored to a disk or database or can be sent through streams. The deserialization is a reverse process of creating object from collection of bytes.

In this assignment, student implemented serialization & deserialization techniques to store list of objects into single file (Figure 18). There are four different types of objects: customer object, product object, order object and order details object. As well as, all stored objects are converted to sequences of bytes and highly protected.

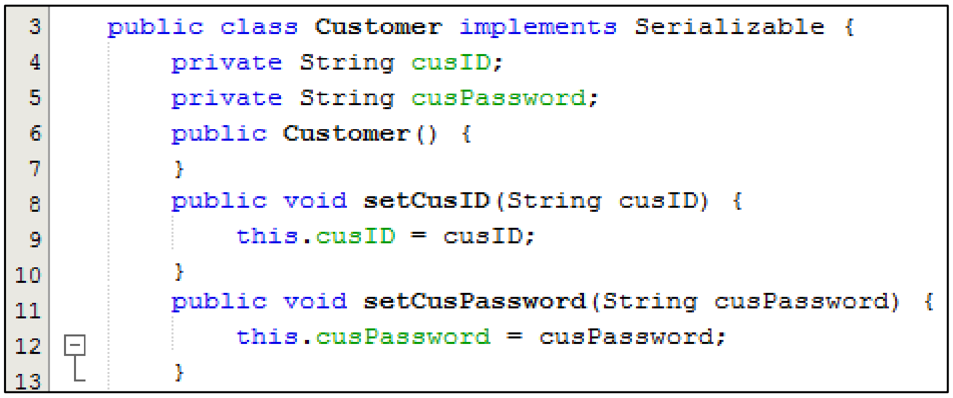


```java
import java.io.Serializable;

public class Customer implements Serializable {
    private String cusID;
    private String cusPassword;
    private String cusType;
    private String cusName;
    private String cusAddress;
    private String cusContact;
    private String cusDate;

    public Customer() {
    }
```

Figure 18: The serialization techniques implemented to customer class.

Encapsulation

Encapsulation is one of the vital fundamentals of OOP concepts. Encapsulation in java is a process of wrapping the variables and code acting on the methods together as a single unit. In result, the variables of a class will be invisible from different classes, and can be accessible only through the pubic methods of their current class. In this assignment, developer implemented encapsulation techniques to object properties (Figure 19), as consequence, all original properties are hidden from external classes.

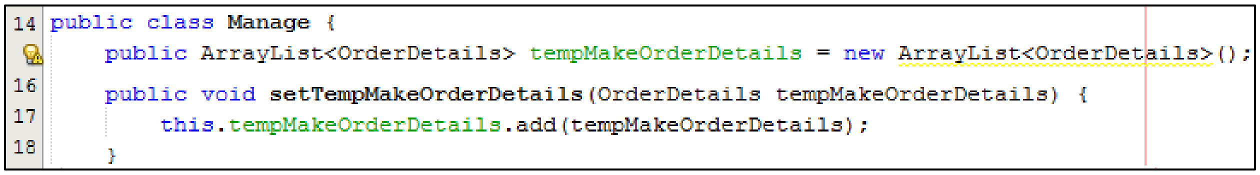


```java
public class Customer implements Serializable {
    private String cusID;
    private String cusPassword;
    public Customer() {
    }
    public void setCusID(String cusID) {
        this.cusID = cusID;
    }
    public void setCusPassword(String cusPassword) {
        this.cusPassword = cusPassword;
    }
```

Figure 19: The figure at the above represents implementation of encapsulation techniques.

ArrayList

ArrayList is a part of standard library and it is available through **java.util.ArrayList** package. An ArrayList provides dynamic non-primitive arrays, and it is clear that it is slower than normal primitive arrays but ArrayList can be helpful in programs to manipulate sequences of non-primitive objects. In this assignment, student implemented library ArrayList to store temporary objects, such as list of ordered objects (Figure 20).

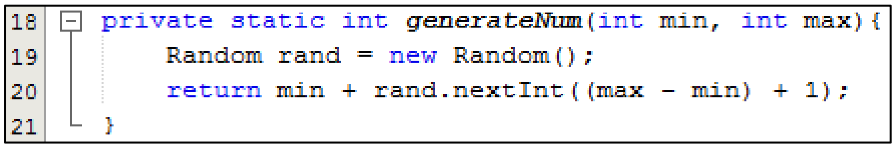


```
public class Manage {
    public ArrayList<OrderDetails> tempMakeOrderDetails = new ArrayList<OrderDetails>();
    public void setTempMakeOrderDetails(OrderDetails tempMakeOrderDetails) {
        this.tempMakeOrderDetails.add(tempMakeOrderDetails);
    }
```

Figure 20: The figure at the above represents implementation of ArrayList for order details.

Generate Random Numbers

In this assignment, students have to use unique values to describe objects, such as customer id, product id and order id. In this case, students should use java library collections to generate random number. This advantage helps with processing time and saves extra lines of code. If developer wants to use a certain feature, such as generating a random number, first need to import **java.util.Random** library into the code, then need to declare few lines of code at the beginning of the class or before the main function (Figure 21).

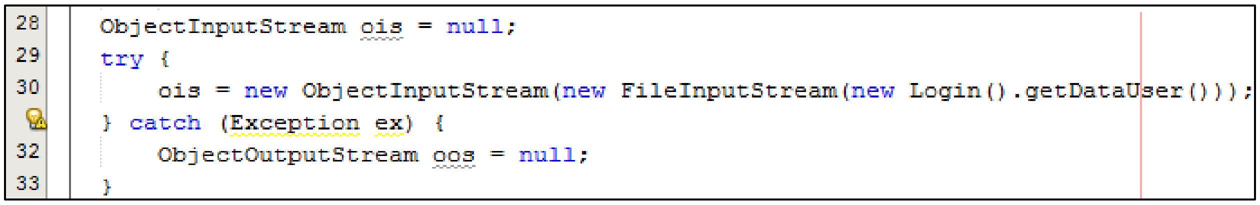


```java
private static int generateNum(int min, int max){
    Random rand = new Random();
    return min + rand.nextInt((max - min) + 1);
}
```

Figure 21: The figure at the above represents a generateNum function.

ObjectInputStream & ObjectOutputStream

The ObjectInputStream & ObjectOutputStream is a java class, and both of them are available under the standard java library collection (**java.io.ObjectOutputStream**). It enables users to store java objects to an OutputStream instead of just primitive bytes, and the InputStream is used to call the objects again. In this assignment, student used ObjectStream features to get access to output files, which hold user objects, product objects, order objects and order details objects (Figure 22).

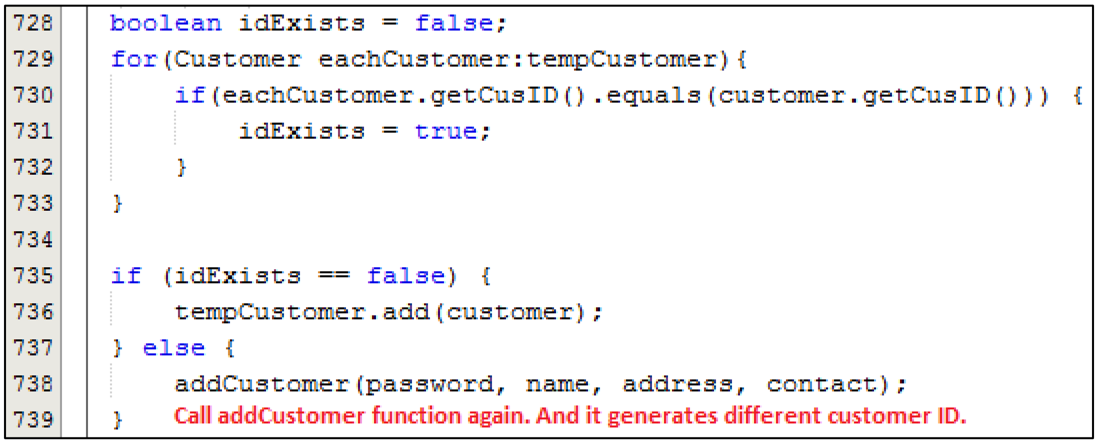


```
ObjectInputStream ois = null;
try {
    ois = new ObjectInputStream(new FileInputStream(new Login().getDataUser()));
} catch (Exception ex) {
    ObjectOutputStream oos = null;
}
```

Figure 22: The figure at the above represents implementation of ObjectInputStream, which is expected to read customer data.

No Duplication of IDs

Within the system, unique identification numbers are implemented to differentiate objects. Such as customer id (**UID000000**), product id (**PID000000**) and order id (**OID000000**). Each value is 9 characters long, 3 capital letters which based on object names, and 6 random generated numbers. During the registration process, if generated identification number is already used for describing certain objects, then if-else statement recalls whole function again, in result, system generates new object identification number.

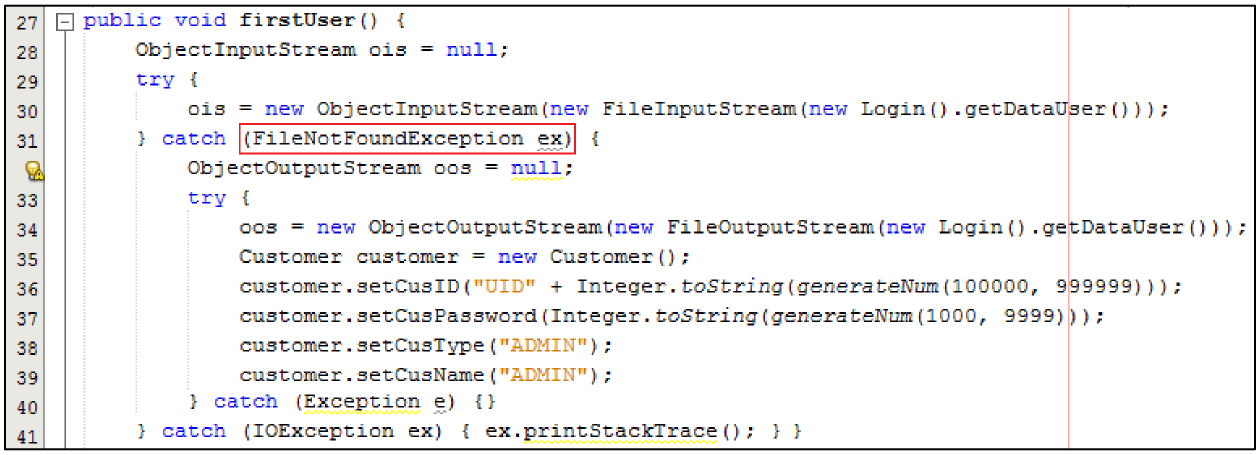


```java
boolean idExists = false;
for(Customer eachCustomer:tempCustomer){
    if(eachCustomer.getCusID().equals(customer.getCusID())) {
        idExists = true;
    }
}

if (idExists == false) {
    tempCustomer.add(customer);
} else {
    addCustomer(password, name, address, contact);
}    Call addCustomer function again. And it generates different customer ID.
```

Figure 23: The simple logic is implemented to prevent duplication of object identification number.

First User is Auto Generated

The figure 24 in the below represents simple logic which designed to generate first user or an admin. At the beginning, logic checks that is there an output file or not which contains user data (via **ObjectInputStream** and **FileNotFoundException**), if not exists then creates new output file, and stores auto generated first user, if an output file is already exists, then skips this procedure. During the generation process, system randomly creates credentials, such as user id and user password (format: UID000000, 0000).

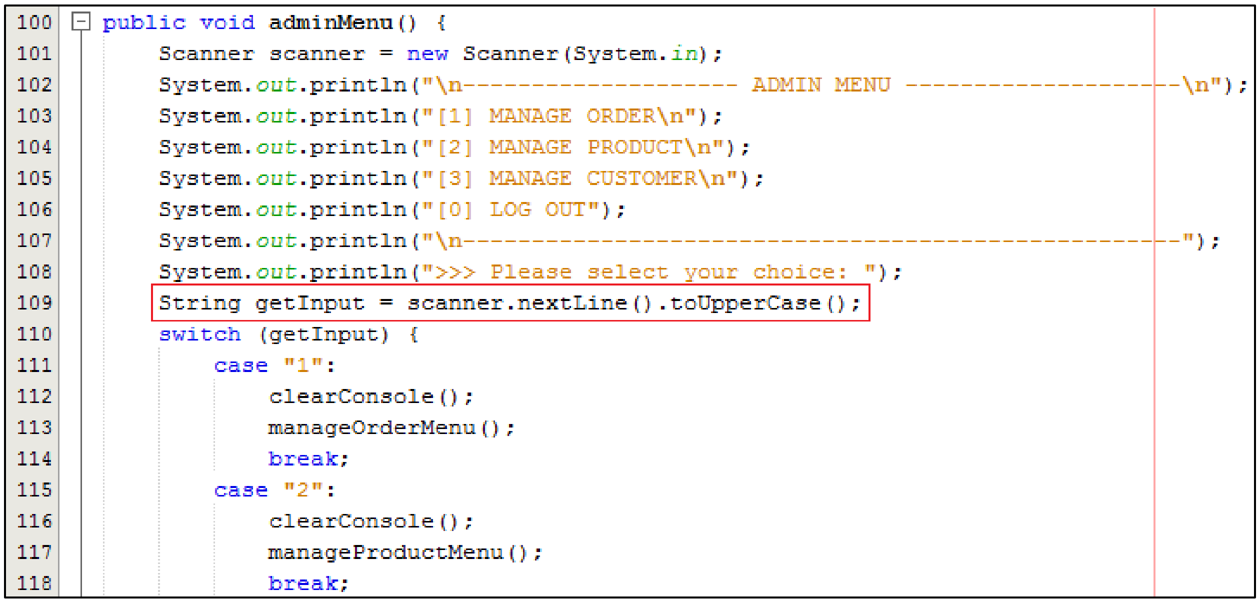


```
public void firstUser() {
    ObjectInputStream ois = null;
    try {
        ois = new ObjectInputStream(new FileInputStream(new Login().getDataUser()));
    } catch (FileNotFoundException ex) {
        ObjectOutputStream oos = null;
        try {
            oos = new ObjectOutputStream(new FileOutputStream(new Login().getDataUser()));
            Customer customer = new Customer();
            customer.setCusID("UID" + Integer.toString(generateNum(100000, 999999)));
            customer.setCusPassword(Integer.toString(generateNum(1000, 9999)));
            customer.setCusType("ADMIN");
            customer.setCusName("ADMIN");
        } catch (Exception e) {}
    } catch (IOException ex) { ex.printStackTrace(); } }
```

Figure 24: The simple logic is implemented to create first user (admin) if system has no any user.

No Case Sensitive for Input Values

In this assignment, system is designed as no case sensitive for all input values. For example, "admin1" and “ADMIN1” are same inputs. Because system automatically converts all inputs to uppercase letters before processing or storing it an output file (Figure 25). In this case, users do not have to think about difference between uppercase and lowercase letters, in one word, system is designed as user friendly, and easy to use.

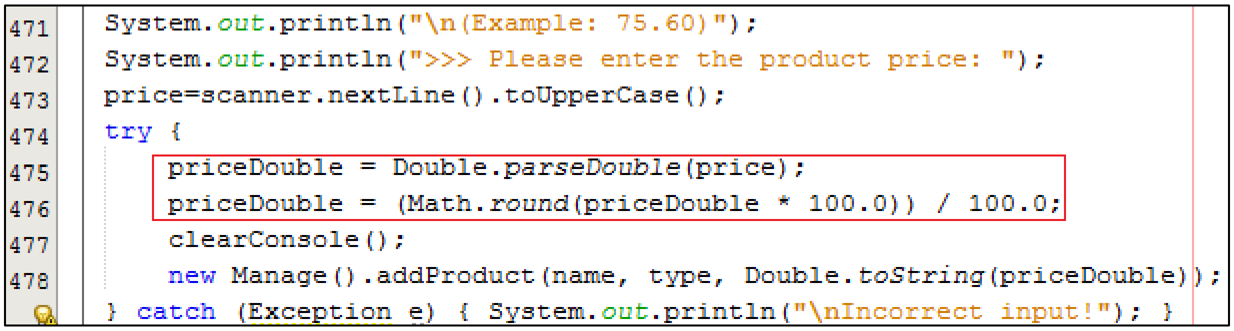


```
public void adminMenu() {
    Scanner scanner = new Scanner(System.in);
    System.out.println("\n-------------------- ADMIN MENU ---------------------\n");
    System.out.println("[1] MANAGE ORDER\n");
    System.out.println("[2] MANAGE PRODUCT\n");
    System.out.println("[3] MANAGE CUSTOMER\n");
    System.out.println("[0] LOG OUT");
    System.out.println("\n-----------------------------------------------------");
    System.out.println(">>> Please select your choice: ");
    String getInput = scanner.nextLine().toUpperCase();
    switch (getInput) {
        case "1":
            clearConsole();
            manageOrderMenu();
            break;
        case "2":
            clearConsole();
            manageProductMenu();
            break;
```

Figure 25: The figure at the above represents Admin Menu.

Math.round()

In this assignment, a Math.round() method is implemented to round product prices in 0.00 format (Math.round() is available under the standard java libraries). Because, if user enters unrealistic values such as 111.1111 or 12.99998 then system automatically converts it to double format, which is 111.11 and 13.0 respectively. In result, system become more accurate in calculations and become more user friendly.

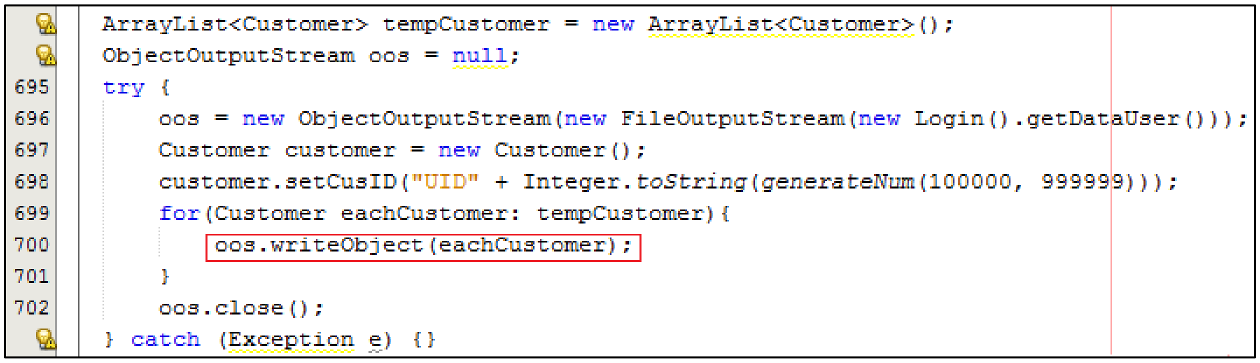


```
System.out.println("\n(Example: 75.60)");
System.out.println(">>> Please enter the product price: ");
price=scanner.nextLine().toUpperCase();
try {
    priceDouble = Double.parseDouble(price);
    priceDouble = (Math.round(priceDouble * 100.0)) / 100.0;
    clearConsole();
    new Manage().addProduct(name, type, Double.toString(priceDouble));
} catch (Exception e) { System.out.println("\nIncorrect input!"); }
```

Figure 26: The figure at the above represents implementation of Math.round method.

Objects are Implemented to Store Outputs

Objects are vital functionalities of object-oriented programing. Within the program, 4 different objects are implemented to support system (customer object, product object, order object and order details object), and they can be treated as variables, just need to declare it with the ***new*** keyword. The keyword invokes matched constructor defined in the class to create an object. Manoj stated that an object is a distinct entity instantiated from a class. Objects have fields to store values and methods for action. Fields and methods of one object are distinct from other objects of the same class (Manoj, 2016).

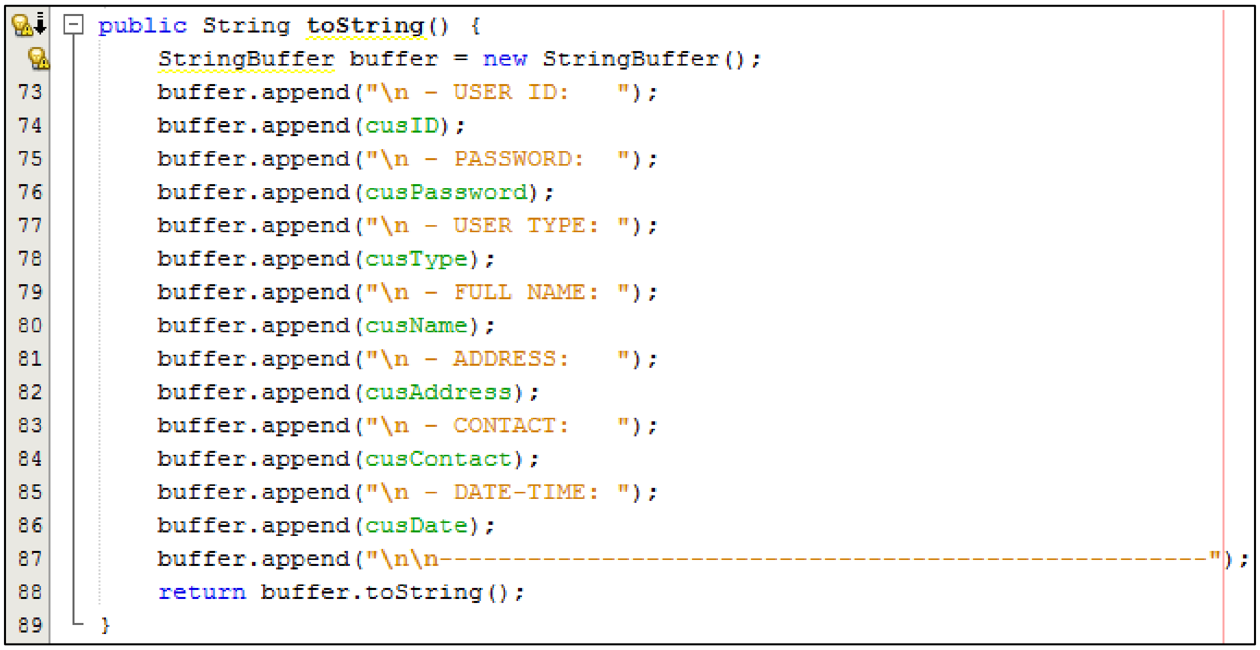


```
ArrayList<Customer> tempCustomer = new ArrayList<Customer>();
ObjectOutputStream oos = null;
try {
    oos = new ObjectOutputStream(new FileOutputStream(new Login().getDataUser()));
    Customer customer = new Customer();
    customer.setCusID("UID" + Integer.toString(generateNum(100000, 999999)));
    for(Customer eachCustomer: tempCustomer){
        oos.writeObject(eachCustomer);
    }
    oos.close();
} catch (Exception e) {}
```

Figure 27: The figure at the above represents procedure of writing objects to an output file.

toString() Method is Implemented

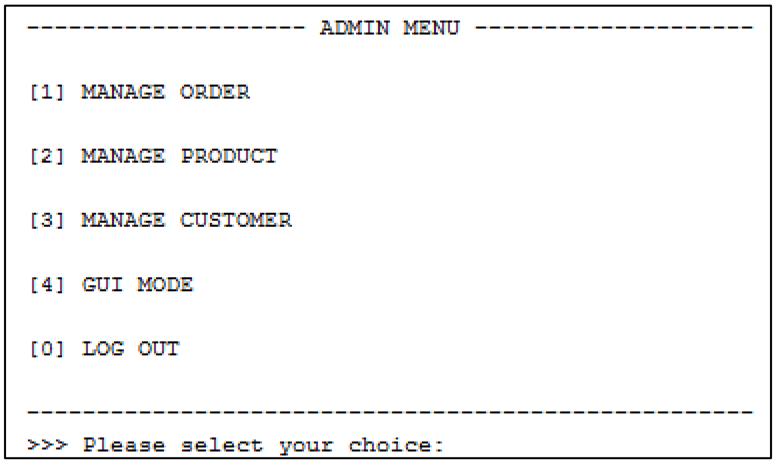
The java toString() method is useful when user need a string representation of an object. It is declared in Object class. And toString() method can be overridden to update or change the String representation of the Object. The figure 28 in the below represents implementation of toString() method, and method is located within the Customer class.



```
public String toString() {
    StringBuffer buffer = new StringBuffer();
    buffer.append("\n - USER ID:    ");
    buffer.append(cusID);
    buffer.append("\n - PASSWORD:   ");
    buffer.append(cusPassword);
    buffer.append("\n - USER TYPE:  ");
    buffer.append(cusType);
    buffer.append("\n - FULL NAME:  ");
    buffer.append(cusName);
    buffer.append("\n - ADDRESS:    ");
    buffer.append(cusAddress);
    buffer.append("\n - CONTACT:    ");
    buffer.append(cusContact);
    buffer.append("\n - DATE-TIME:  ");
    buffer.append(cusDate);
    buffer.append("\n\n-----------------------------------------------------");
    return buffer.toString();
}
```

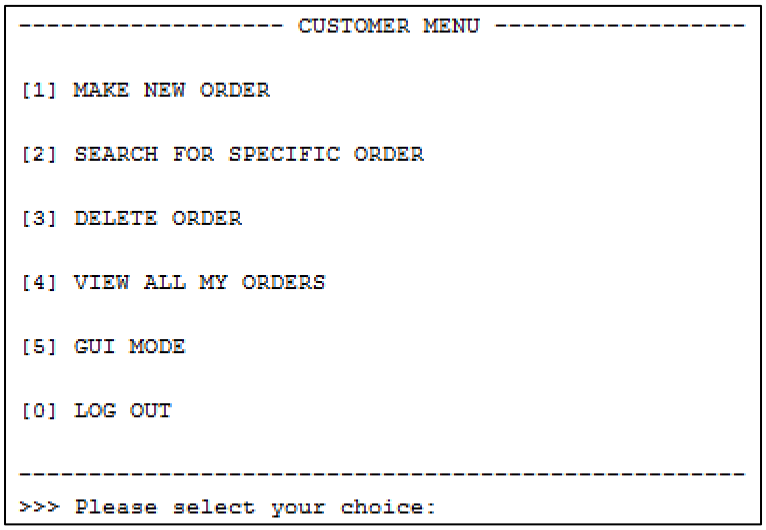
Figure 28: The figure at the above represents toString() method which is implemented to display customer details.

Terminal User Interface



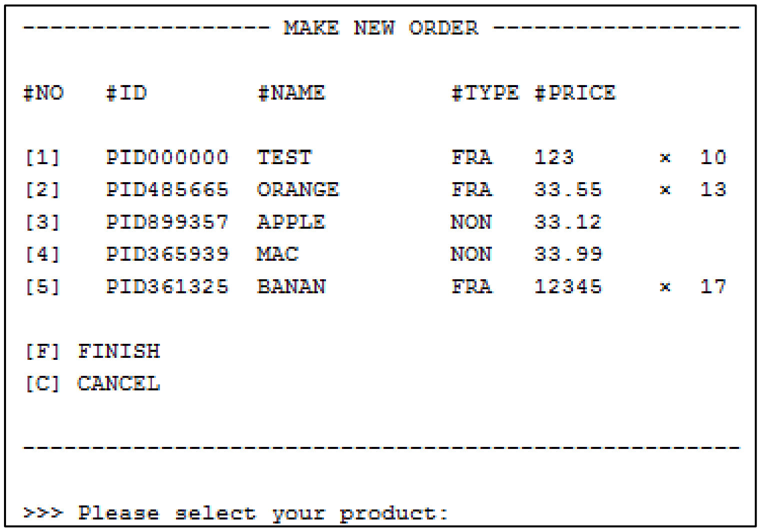
```
-------------------- ADMIN MENU --------------------

[1] MANAGE ORDER

[2] MANAGE PRODUCT

[3] MANAGE CUSTOMER

[4] GUI MODE

[0] LOG OUT

----------------------------------------------------
>>> Please select your choice:
```

Figure 29: The figure at the above represents the admin menu.



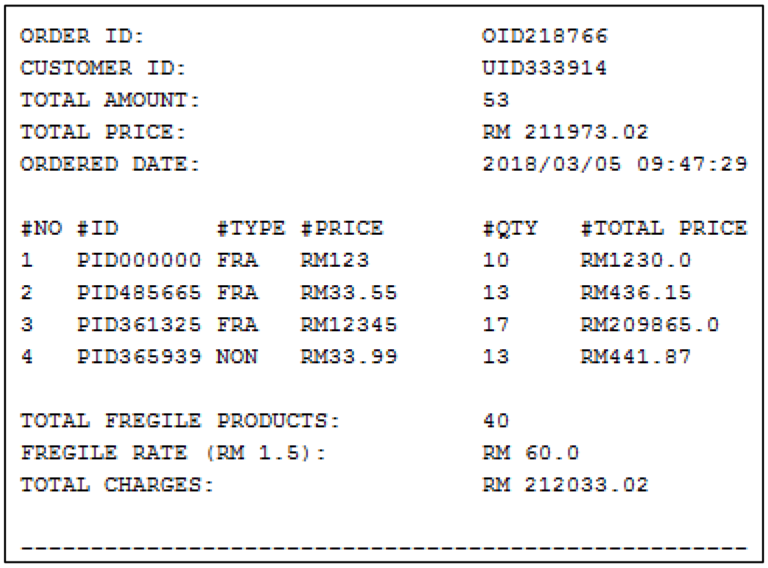
```
------------------- CUSTOMER MENU -------------------

[1] MAKE NEW ORDER

[2] SEARCH FOR SPECIFIC ORDER

[3] DELETE ORDER

[4] VIEW ALL MY ORDERS

[5] GUI MODE

[0] LOG OUT

-----------------------------------------------------
>>> Please select your choice:
```

Figure 30: The figure at the above represents the customer menu.



```
------------------ MAKE NEW ORDER ------------------

#NO   #ID        #NAME         #TYPE #PRICE

[1]   PID000000  TEST          FRA   123      ×  10
[2]   PID485665  ORANGE        FRA   33.55    ×  13
[3]   PID899357  APPLE         NON   33.12
[4]   PID365939  MAC           NON   33.99
[5]   PID361325  BANAN         FRA   12345    ×  17

[F] FINISH
[C] CANCEL

----------------------------------------------------

>>> Please select your product:
```

Figure 31: The figure at the above represents make new order page.



```
ORDER ID:                         OID218766
CUSTOMER ID:                      UID333914
TOTAL AMOUNT:                     53
TOTAL PRICE:                      RM 211973.02
ORDERED DATE:                     2018/03/05 09:47:29

#NO #ID       #TYPE #PRICE        #QTY   #TOTAL PRICE
1   PID000000 FRA   RM123         10     RM1230.0
2   PID485665 FRA   RM33.55       13     RM436.15
3   PID361325 FRA   RM12345       17     RM209865.0
4   PID365939 NON   RM33.99       13     RM441.87

TOTAL FREGILE PRODUCTS:           40
FREGILE RATE (RM 1.5):            RM 60.0
TOTAL CHARGES:                    RM 212033.02

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

Figure 32: The figure at the above represents report page.

Graphical User Interface

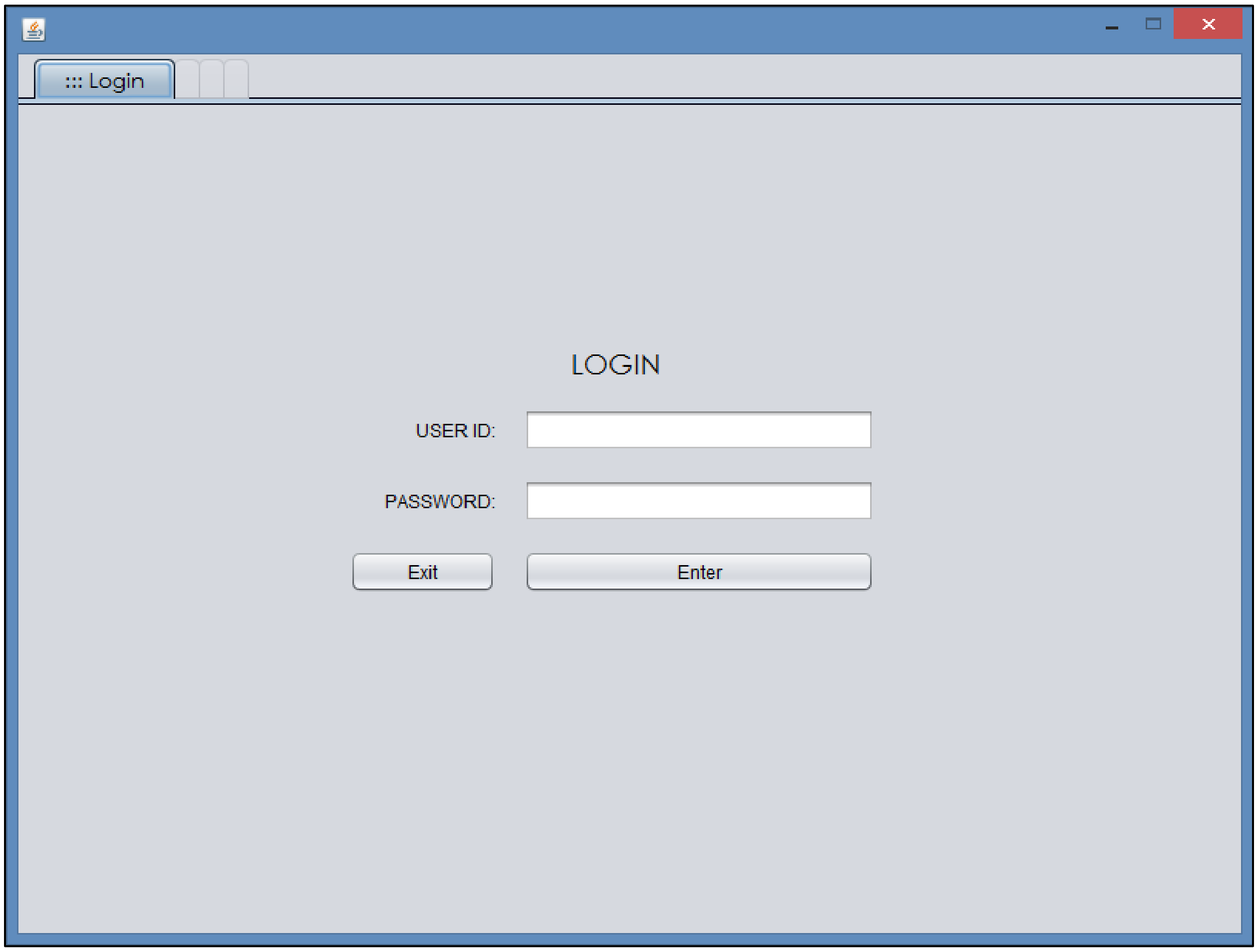
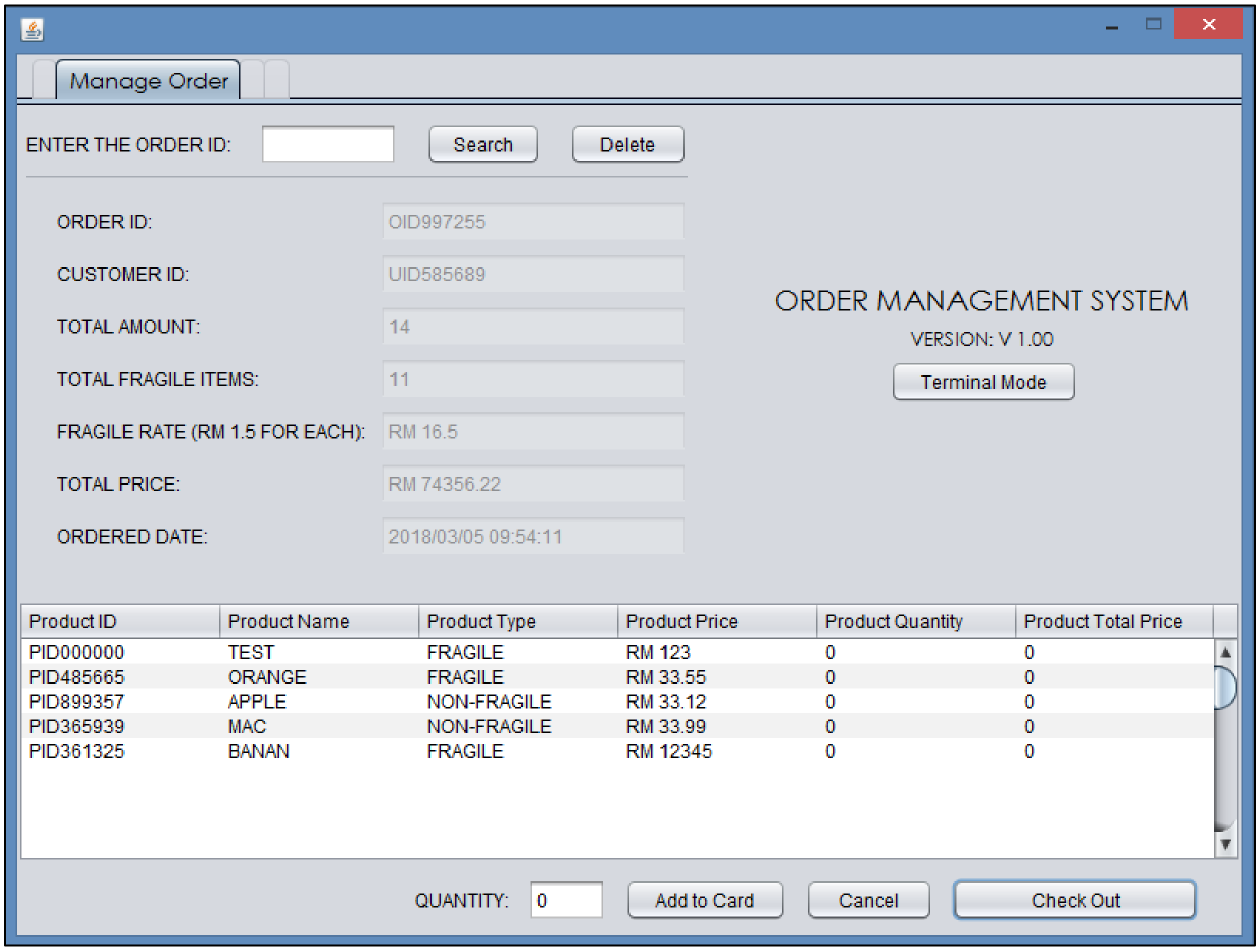


Figure 33: The GUI mode, the login page.



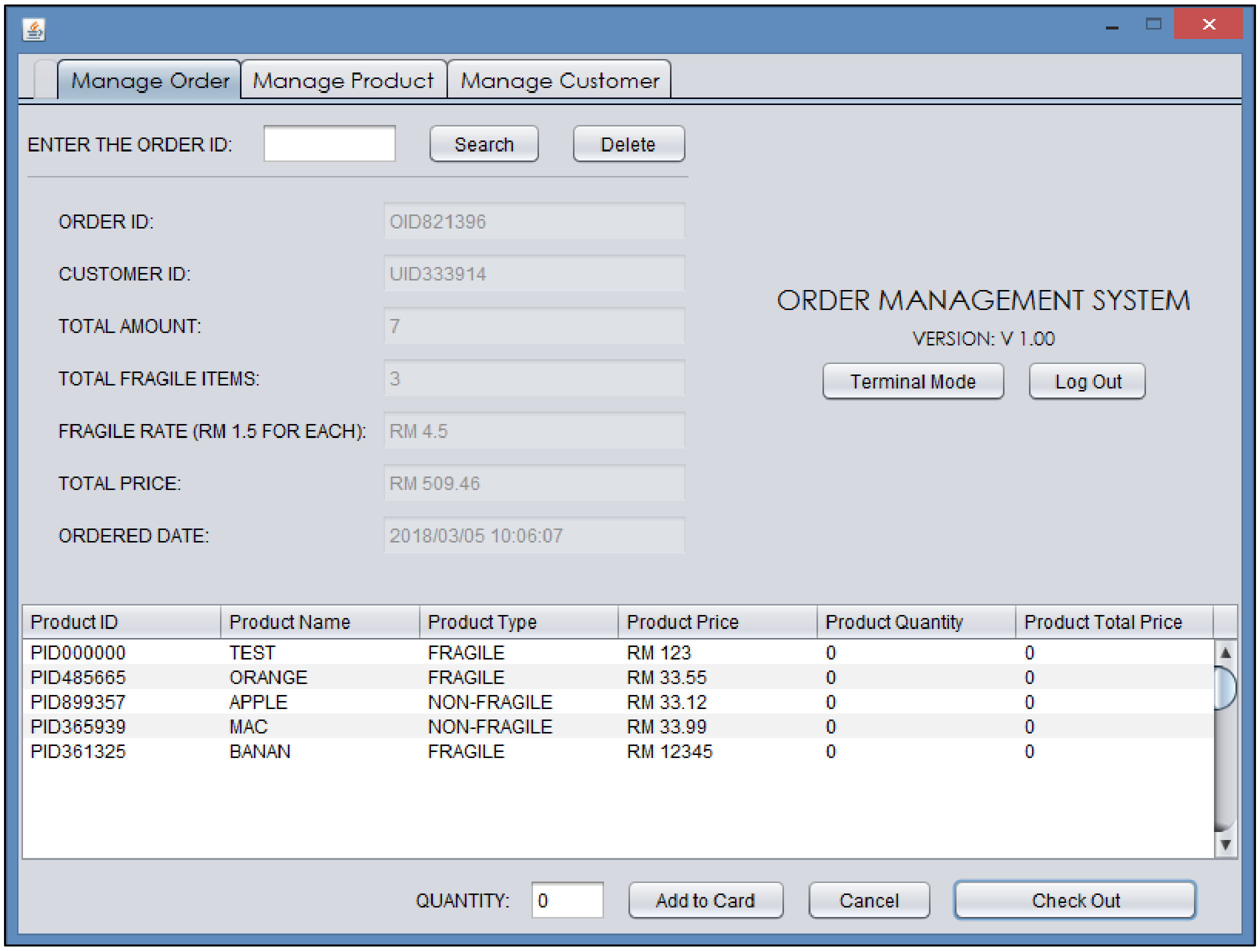
ENTER THE ORDER ID:

ORDER ID:	OID997255
CUSTOMER ID:	UID585689
TOTAL AMOUNT:	14
TOTAL FRAGILE ITEMS:	11
FRAGILE RATE (RM 1.5 FOR EACH):	RM 16.5
TOTAL PRICE:	RM 74356.22
ORDERED DATE:	2018/03/05 09:54:11

Product ID	Product Name	Product Type	Product Price	Product Quantity	Product Total Price
PID000000	TEST	FRAGILE	RM 123	0	0
PID485665	ORANGE	FRAGILE	RM 33.55	0	0
PID899357	APPLE	NON-FRAGILE	RM 33.12	0	0
PID365939	MAC	NON-FRAGILE	RM 33.99	0	0
PID361325	BANAN	FRAGILE	RM 12345	0	0

QUANTITY: 0

Figure 34: The GUI mode, user type is customer.



ENTER THE ORDER ID:

ORDER ID:	OID821396
CUSTOMER ID:	UID333914
TOTAL AMOUNT:	7
TOTAL FRAGILE ITEMS:	3
FRAGILE RATE (RM 1.5 FOR EACH):	RM 4.5
TOTAL PRICE:	RM 509.46
ORDERED DATE:	2018/03/05 10:06:07

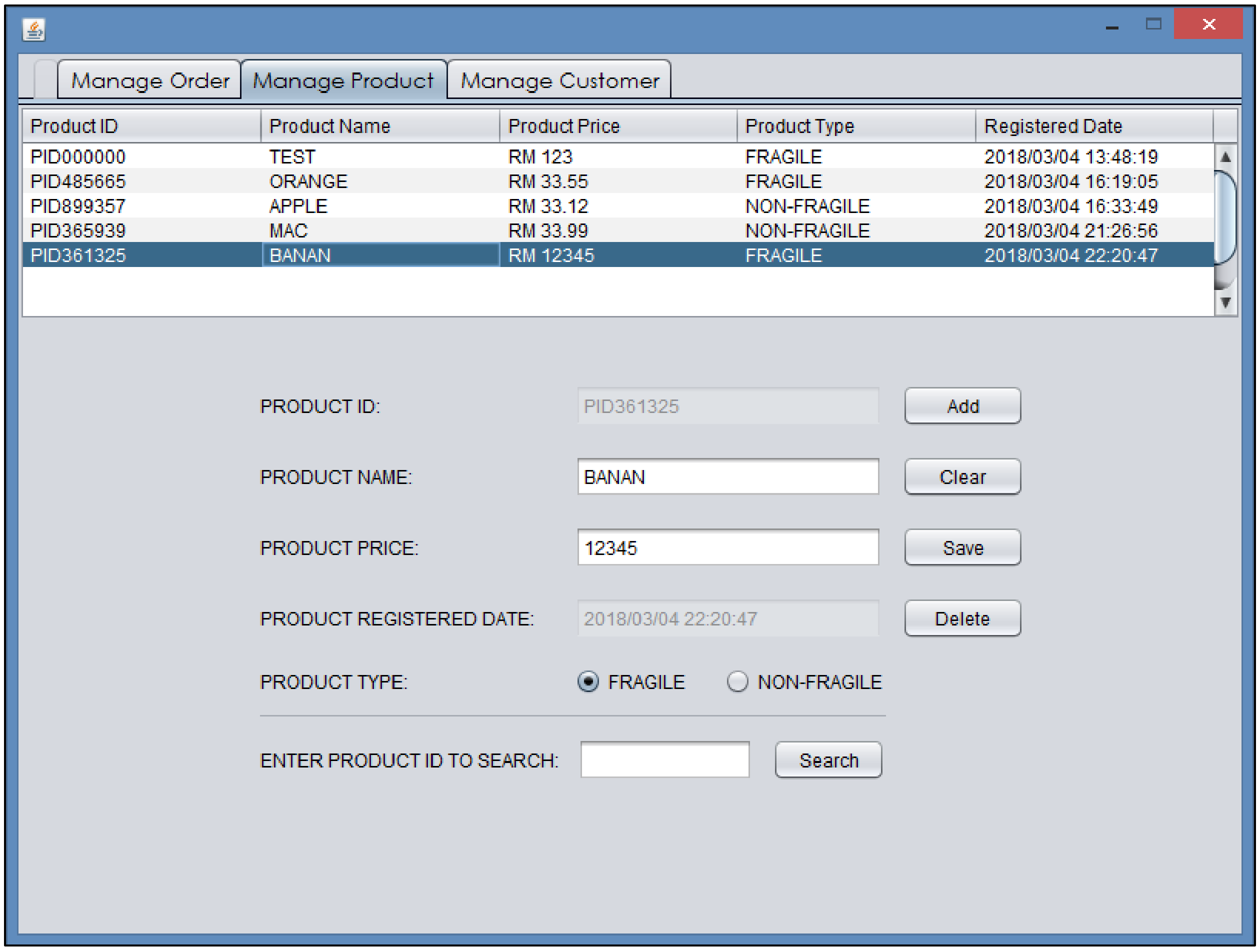
ORDER MANAGEMENT SYSTEM

VERSION: V 1.00

Product ID	Product Name	Product Type	Product Price	Product Quantity	Product Total Price
PID000000	TEST	FRAGILE	RM 123	0	0
PID485665	ORANGE	FRAGILE	RM 33.55	0	0
PID899357	APPLE	NON-FRAGILE	RM 33.12	0	0
PID365939	MAC	NON-FRAGILE	RM 33.99	0	0
PID361325	BANAN	FRAGILE	RM 12345	0	0

QUANTITY: 0

Figure 35: The GUI mode, the manage order page.



Product ID	Product Name	Product Price	Product Type	Registered Date
PID000000	TEST	RM 123	FRAGILE	2018/03/04 13:48:19
PID485665	ORANGE	RM 33.55	FRAGILE	2018/03/04 16:19:05
PID899357	APPLE	RM 33.12	NON-FRAGILE	2018/03/04 16:33:49
PID365939	MAC	RM 33.99	NON-FRAGILE	2018/03/04 21:26:56
PID361325	BANAN	RM 12345	FRAGILE	2018/03/04 22:20:47

PRODUCT ID: PID361325

PRODUCT NAME: BANAN

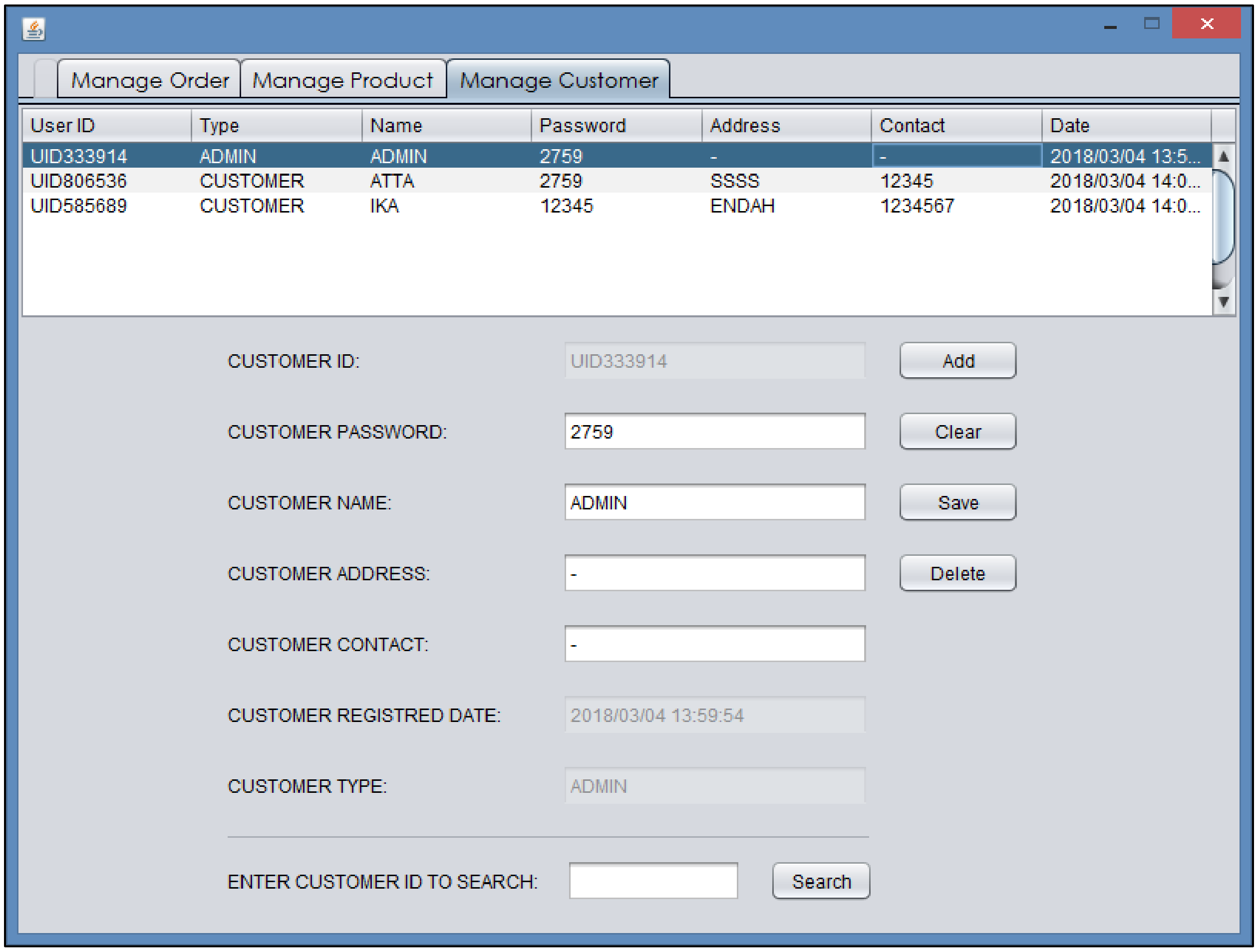
PRODUCT PRICE: 12345

PRODUCT REGISTERED DATE: 2018/03/04 22:20:47

PRODUCT TYPE: [✓ FRAGILE] NON-FRAGILE

ENTER PRODUCT ID TO SEARCH:

Figure 36: The GUI mode, the manage product page.



User ID	Type	Name	Password	Address	Contact	Date
UID333914	ADMIN	ADMIN	2759	-	-	2018/03/04 13:5...
UID806536	CUSTOMER	ATTA	2759	SSSS	12345	2018/03/04 14:0...
UID585689	CUSTOMER	IKA	12345	ENDAH	1234567	2018/03/04 14:0...

CUSTOMER ID:	UID333914
CUSTOMER PASSWORD:	2759
CUSTOMER NAME:	ADMIN
CUSTOMER ADDRESS:	-
CUSTOMER CONTACT:	-
CUSTOMER REGISTRED DATE:	2018/03/04 13:59:54
CUSTOMER TYPE:	ADMIN

ENTER CUSTOMER ID TO SEARCH:

Figure 37: The GUI mode, the manage customer page.

CONCLUSION

There is no doubt that object oriented programming has vital functionalities in designing large systems. It allows to developers to divide tasks in small unities, then it gives possibilities to reuse it or integrate it with other part of system. In this assignment, students learn basic fundamentals of object oriented programming, such as inheritance, polymorphism, encapsulation, and abstraction (Raymond, 2005). On the other hand, students got chance to differentiate between variety data types and its unique features, for example Vectors and ArrayLists, then implemented it to their assignments. As well as, students learn how to allocate resources effectively (memory), or how to design an optimal algorithms (timely efficient). And it was clear that all researches and practices are definitely helpful for future software engineers.

REFERENCES

Alisdair, W. (2007) *Relationships for Object-oriented Programming Languages.* University of Cambridge Computer Laboratory. United Kingdom.

Robertson, L.A. (2007) *Simple Program Design: A Step-by-Step Approach.* Fifth edition. Thomson. Nelson Australia Pty Limited.

LearningLad. (2015) *Java Object Oriented Programming Video Tutorials: Video 1 – 12.* [Online] Available at: https://www.youtube.com/watch?v=0wGq6yZNn60&list=PLfVsf4Bjg79DeKFtT3TvgcviYB0R860Df

[Accessed: 10/01/2018].

Stackoverflow.com. (2009) *What is Object Serialization?* [Online]

Available at: https://stackoverflow.com/questions/447898/what-is-object-serialization

[Accessed: 10/01/2018].

W3resource.com. (2017) *Inheritance (IS-A) vs. Composition (HAS-A) Relationship.* [Online]

Available at: https://www.w3resource.com/java-tutorial/inheritance-composition-relationship.php

[Accessed: 10/01/2018].

Martin, G. (2017) *Random Number Generation* [Online]. Available at: https://study.com/academy/lesson/java-generate-random-number-between-1-100.html

[Accessed: 27/02/2018].

Jenkov, J. (2015) *Java IO: ObjectOutputStream* [Online].

Available at: http://tutorials.jenkov.com/java-io/objectoutputstream.html

[Accessed: 27/02/2018].

Raymond, L. (2005) *4 Major Principles of Object-Oriented Programming* [Online]. Available at: http://codebetter.com/raymondlewallen/2005/07/19/4-major-principles-of-object-oriented-programming/

[Accessed: 27/02/2018].

Manoj, D. (2016) *Understanding Java Objects and Java Classes: Overview of Java Objects* [Online]. Available at: https://www.developer.com/java/data/understanding-java-objects-and-java-classes.html

[Accessed: 28/02/2018].