A·P·U

ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

MODULE CODE: CT038-3-2-OODJ

INTAKE CODE: UC2F1609SE

ASSIGNMENT TITLE: FREIGHT MANAGEMENT SYSTEM

LECTURER NAME: DR. KADHAR BATCHA NOWSHATH

STUDENT NAME: SKRYPNYK BOHDAN

STUDENT ID: TP036337

Table of Contents

Use Case Diagram

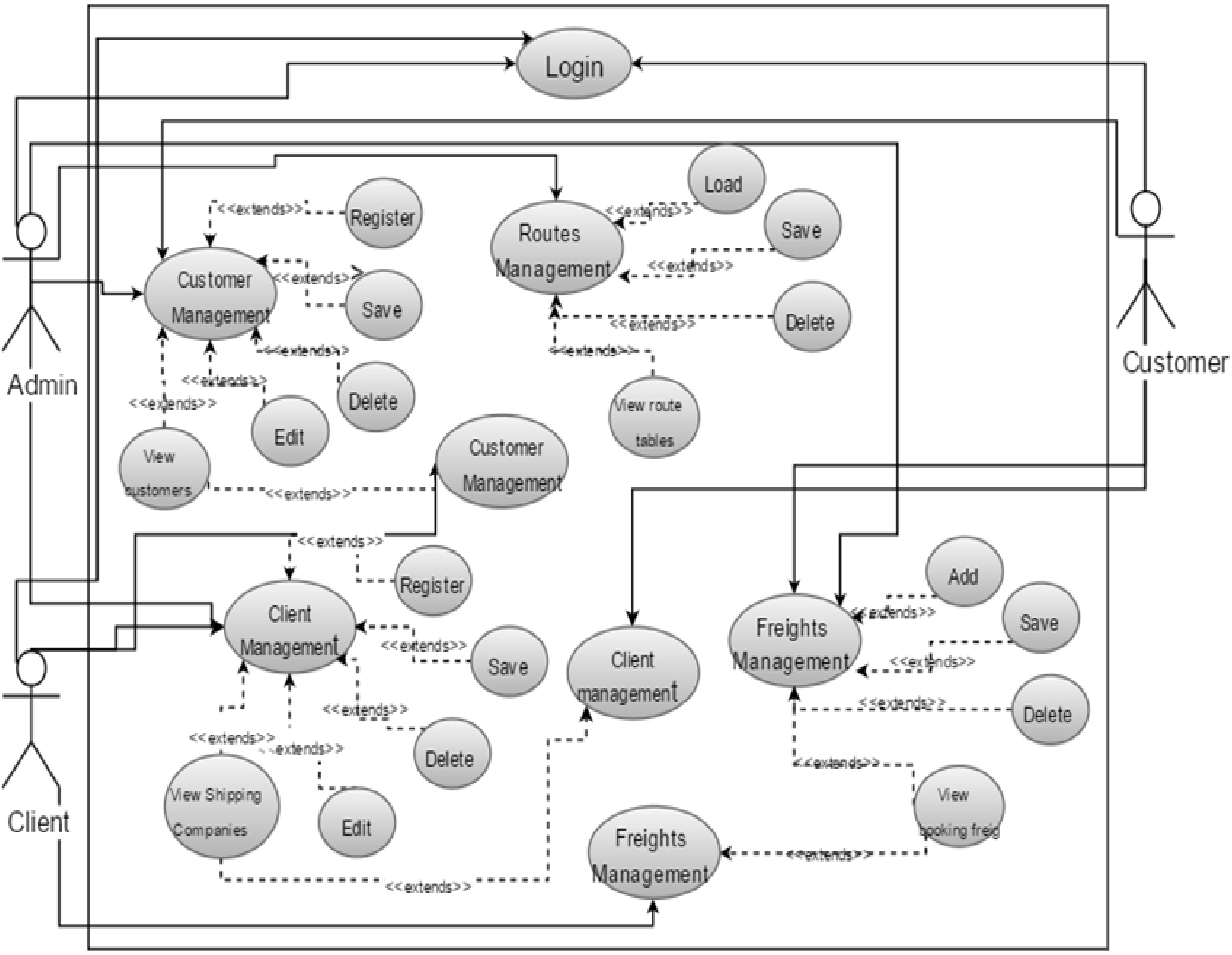


Figure 1 Use case Diagram

Use Case Description

Use case code	UC1
Title	Login
Description	User could log-in to the system
Actor(s)	Administrator, Client, Customer
Precondition	User must run the application
Post-condition	User logged in to the system
Normal Course of Events	1: User enters the credentials (Username and Password) 2: User press 'Enter' 3: User is in the system and it is showing available options
Extension(s)	No extension

Table 1 Description of the Use Case Diagram for UC1

Use case code	UC2
Title	Client Management
Description	Administrator / Client creates a new client
Actor(s)	Administrator / Client
Precondition	Administrator / Client must be logged in to the system
Post-condition	Administrator / Client created a new client
Normal Course of Events	1: Admin / Client select the "Client Management" option in the "Administration / Client module's" in console 2: System open management module 3: Admin / Client selects "Register Client" option 4: System shows information fields to be filled 5: Admin / Client fulfills the required fields 6: System outputs that the new client has been created
Extension(s)	1: Save 2: Delete client 3: Edit client information 4: View Shipping Companies

Table 2 Description of the Use Case Diagram for UC2

Use case code	UC3
Title	Customer Management
Description	Admin creates new customers
Actor(s)	Administrator
Precondition	Admin must be logged in to the system
Post-condition	Admin created new customers
Normal Course of Events	1: Admin select the "Customer Management" option in the "Administration module" in console 2: System open management module 3: Admin selects "Register Customer" option 4: System shows information fields to be filled

	5: Admin fulfills the required fields 6: System outputs that the new client has been created
Extension(s)	1: Edit customer information 2: Delete customer 3: View Customer

Table 3 Description of the Use Case Diagram for UC3

Use case code	UC4
Title	Routes Management
Description	Admin register new route
Actor(s)	Administrator
Precondition	Administrator must be logged in to the system
Post-condition	Admin register new route
Normal Course of Events	1: Administrator select the “Routes Management” option in the “Administration module” in console 2: System outputs the fields to be filled 3: Clerk/Manager inputs the required information 4: System outputs that the Registration is successful
Extension(s)	E1: Edit Customer E2: Delete Customer E3: Refresh Customer List

Table 4 Description of the Use Case Diagram for UC4

Use case code	UC5
Title	Freights Management
Description	Admin / Customers create new booking
Actor(s)	Administrator / Customers
Precondition	Administrator / Customers must be logged in to the system
Post-condition	Admin / Customers create new booking
Normal Course of Events	1: Administrator / Customers select the “Freights Management” option in the “Administration/ Customers module’s” in console 2: System outputs the fields to be filled 3: Administrator / Customers inputs the required information 4: System outputs that the Registration is successful
Extension(s)	E1: Add booking E2: Delete booking E3: View booking freights

Table 5 Description of the Use Case Diagram for UC5

Use case code	UC6
Title	View Customer
Description	Client views list of customers
Actor(s)	Client
Precondition	Client must be logged in to the system
Post-condition	Client carries out the checkup a list of customers

Normal Course of Events	1: Client select the “Customer Management” option in the “Client module” in console 2: Client views list of customers
Extension(s)	No extension

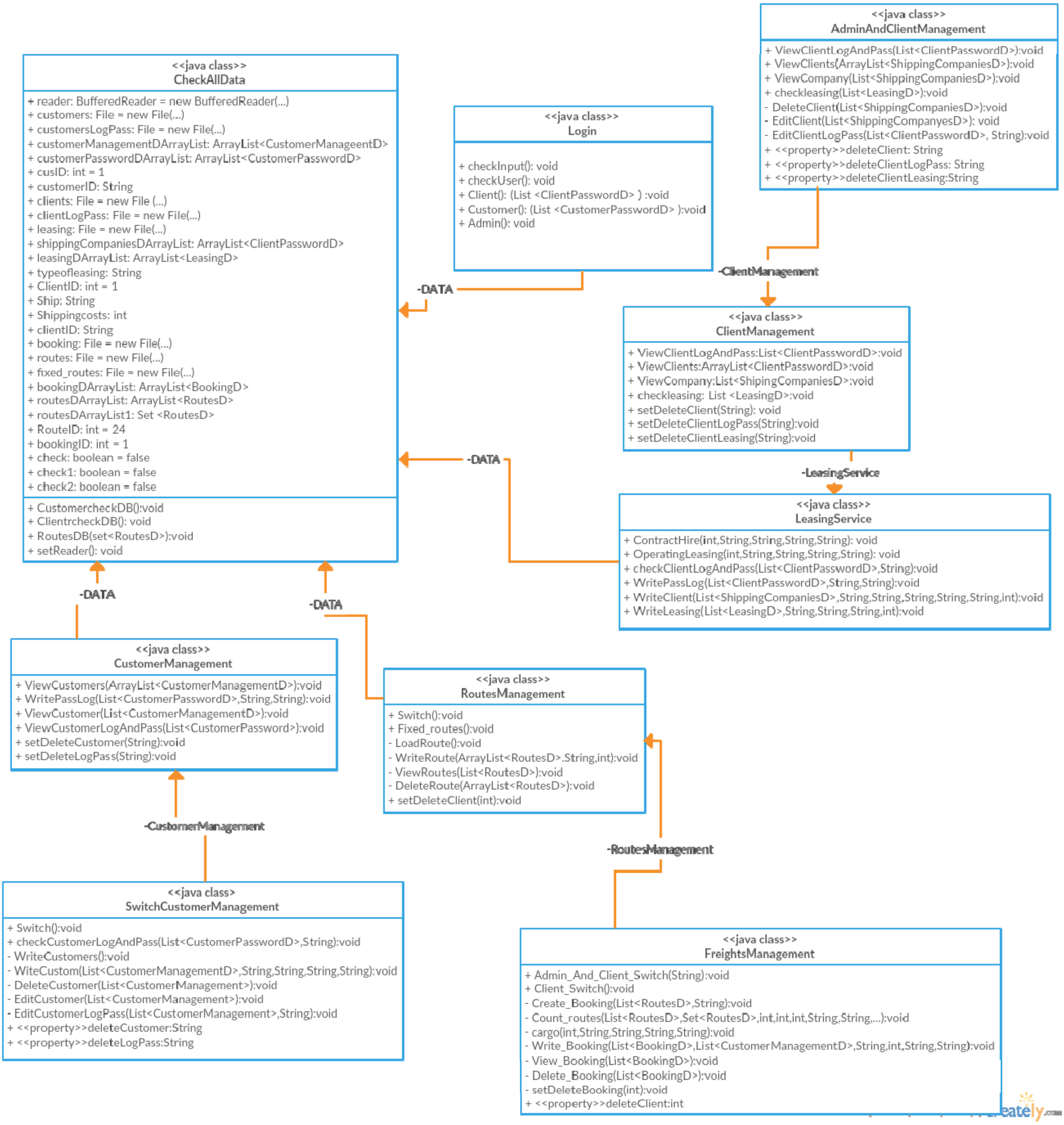
Table 6 Description of the Use Case Diagram for UC6

Use case code	UC7
Title	View freights
Description	Client views list of freights
Actor(s)	Client
Precondition	Client must be logged in to the system
Post-condition	Client carries out the checkup a list of freights
Normal Course of Events	1: Client select the “Freights Management” option in the “Client module” in console 2: Client views list of freights
Extension(s)	No extension

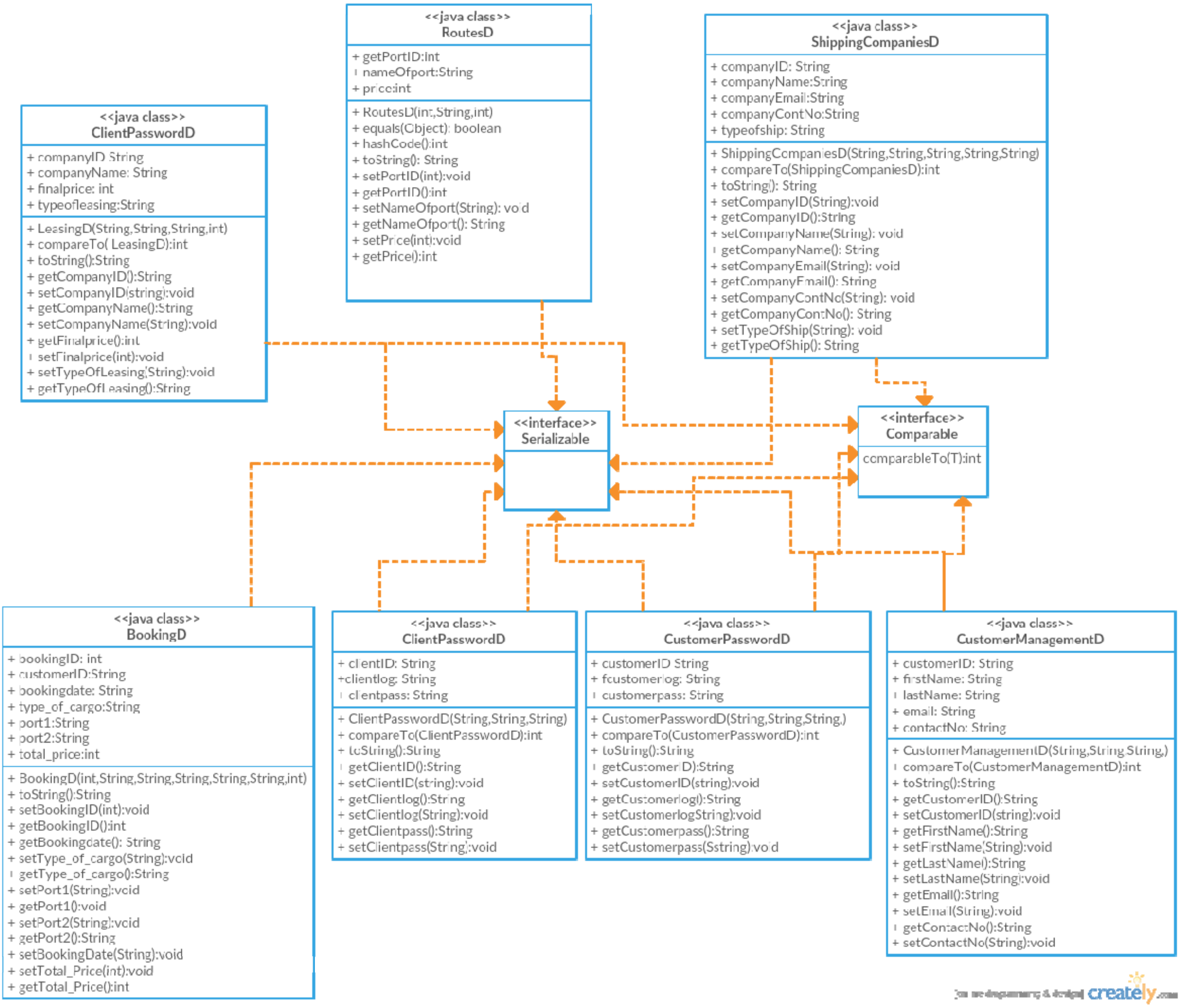
Table 7 Description of the Use Case Diagram for UC7

Use case code	UC8
Title	View Shipping Companies
Description	Customer view shipping companies
Actor(s)	Customer
Precondition	Customer must be logged in to the system
Post-condition	Customer carries out the checkup a list of freights
Normal Course of Events	1: Client select the “Client Management” option in the “Customer module” in console 2: Client views list shipping companies
Extension(s)	No extension

Class Diagram (Front End)


<<java class>>
AdminAndClientManagement
+ ViewClientLogAndPass(List<ClientPasswordD>):void
+ ViewClients(ArrayList<ShippingCompaniesD>):void
+ ViewCompany(List<ShippingCompaniesD>):void
+ checkleasing(List<LeasingD>):void
- DeleteClient(List<ShippingCompaniesD>):void
- EditClient(List<ShippingCompanyesD>): void
- EditClientLogPass(List<ClientPasswordD>, String):void
+ <<property>>deleteClient: String
+ <<property>>deleteClientLogPass: String
+ <<property>>deleteClientLeasing:String
<<java class>>
CheckAllData
+ reader: BufferedReader = new BufferedReader(...)
+ customers: File = new File(...)
+ customersLogPass: File = new File(...)
+ customerManagementDArrayList: ArrayList<CustomerManageentD>
+ customerPasswordDArrayList: ArrayList<CustomerPasswordD>
+ cusID: int = 1
+ customerID: String
+ clients: File = new File (...)
+ clientLogPass: File = new File(...)
+ leasing: File = new File(...)
+ shippingCompaniesDArrayList: ArrayList<ClientPasswordD>
+ leasingDArrayList: ArrayList<LeasingD>
+ typeofleasing: String
+ ClientID: int = 1
+ Ship: String
+ Shippingcosts: int
+ clientID: String
+ booking: File = new File(...)
+ routes: File = new File(...)
+ fixed_routes: File = new File(...)
+ bookingDArrayList: ArrayList<BookingD>
+ routesDArrayList: ArrayList<RoutesD>
+ routesDArrayList1: Set <RoutesD>
+ RouteID: int = 24
+ bookingID: int = 1
+ check: boolean = false
+ check1: boolean = false
+ check2: boolean = false
+ CustomercheckDB():void
+ ClientrcheckDB(): void
+ RoutesDB(set<RoutesD>):void
+ setReader(): void
<<java class>>
Login
+ checkInput(): void
+ checkUser(): void
+ Client(): (List <ClientPasswordD>) :void
+ Customer(): (List <CustomerPasswordD>):void
+ Admin(): void
-DATA
-ClientManagement
<<java class>>
ClientManagement
+ ViewClientLogAndPass:List<ClientPasswordD>:void
+ ViewClients:ArrayList<ClientPasswordD>:void
+ ViewCompany:List<ShipingCompaniesD>:void
+ checkleasing: List <LeasingD>:void
+ setDeleteClient(String): void
+ setDeleteClientLogPass(String):void
+ setDeleteClientLeasing(String):void
-DATA
-LeasingService
<<java class>>
LeasingService
+ ContractHire(int,String,String,String,String): void
+ OperatingLeasing(int,String,String,String,String): void
+ checkClientLogAndPass(List<ClientPasswordD>,String):void
+ WritePassLog(List<ClientPasswordD>,String,String):void
+ WriteClient(List<ShippingCompaniesD>,String,String,String,String,String,int):void
+ WriteLeasing(List<LeasingD>,String,String,String,int):void
-DATA
-DATA
<<java class>>
CustomerManagement
+ ViewCustomers(ArrayList<CustomerManagementD>):void
+ WritePassLog(List<CustomerPasswordD>,String,String):void
+ ViewCustomer(List<CustomerManagementD>):void
+ ViewCustomerLogAndPass(List<CustomerPassword>):void
+ setDeleteCustomer(String):void
+ setDeleteLogPass(String):void
<<java class>>
RoutesManagement
+ Switch():void
+ Fixed_routes():void
- LoadRoute():void
- WriteRoute(ArrayList<RoutesD>.String,int):void
- ViewRoutes(List<RoutesD>):void
- DeleteRoute(ArrayList<RoutesD>):void
+ setDeleteClient(int):void
-CustomerManagement
-RoutesManagement
<<java class>
SwitchCustomerManagement
+ Switch():void
+ checkCustomerLogAndPass(List<CustomerPasswordD>,String):void
- WriteCustomers():void
- WiteCustom(List<CustomerManagementD>,String,String,String,String):void
- DeleteCustomer(List<CustomerManagement>):void
- EditCustomer(List<CustomerManagement>):void
- EditCustomerLogPass(List<CustomerManagement>,String):void
+ <<property>>deleteCustomer:String
+ <<property>>deleteLogPass:String
<<java class>>
FreightsManagement
+ Admin_And_Client_Switch(String):void
+ Client_Switch():void
- Create_Booking(List<RoutesD>,String):void
- Count_routes(List<RoutesD>,Set<RoutesD>,int,int,int,String,String,...):void
- cargo(int,String,String,String,String):void
- Write_Booking(List<BookingD>,List<CustomerManagementD>,String,int,String,String):void
- View_Booking(List<BookingD>):void
- Delete_Booking(List<BookingD>):void
- setDeleteBooking(int):void
+ <<property>>deleteClient:int
eately.com

Class Diagram (Back End)



Activity Diagram

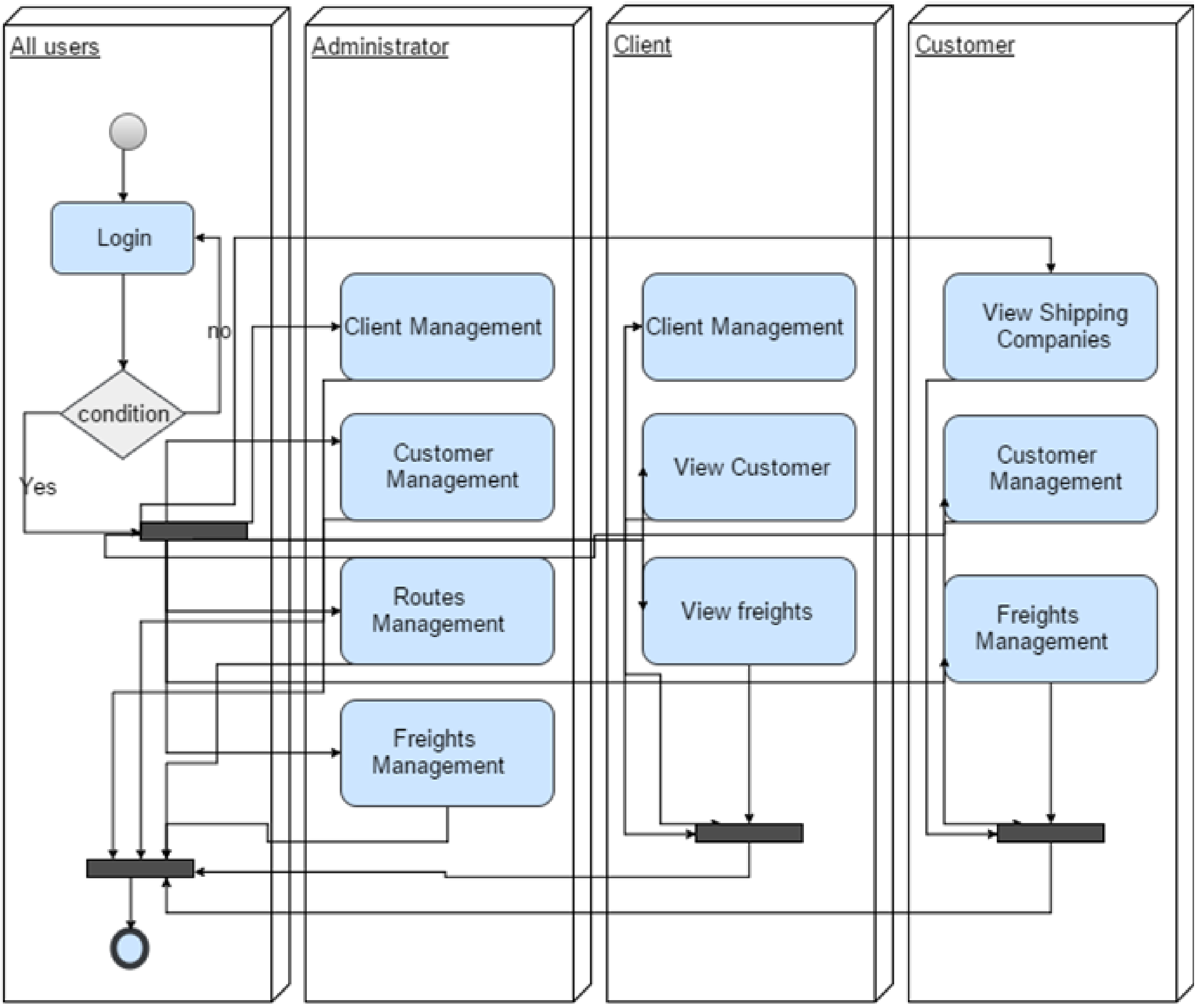


Figure 3 Activity Diagram

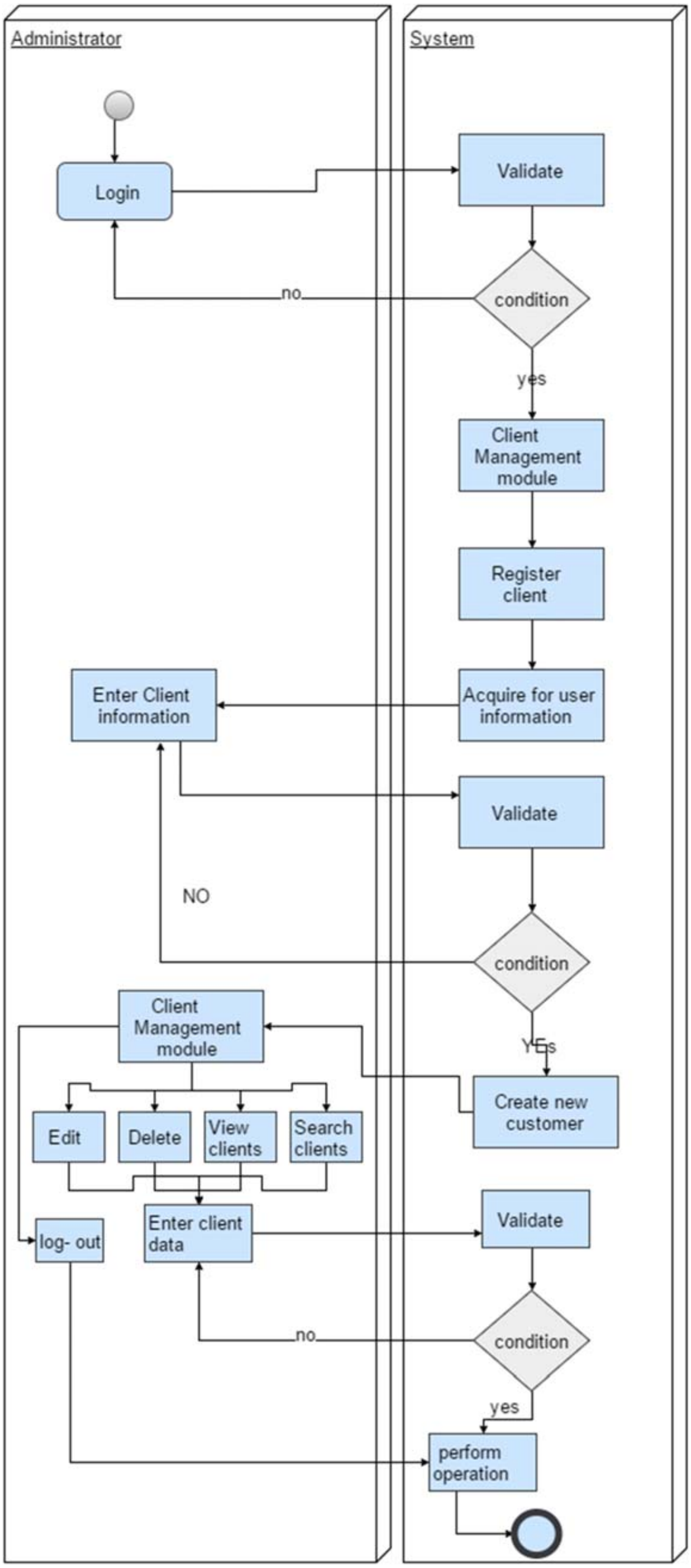


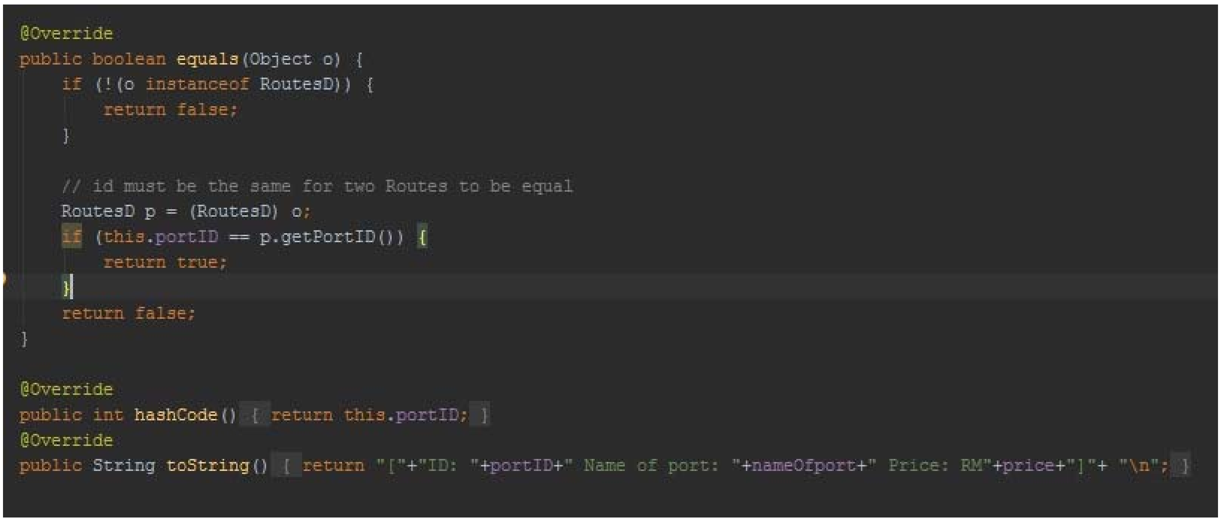
Figure 4 Activity Diagram

Implementing Polymorphism in Project

Polymorphism

Polymorphism in Java defines an object, which can perform single actions by different ways. So, we could say that polymorphism means ‘many forms’. It includes two major concepts, such as *Overriding* and *Overloading*. Let’s first consider the Overriding.

Overriding

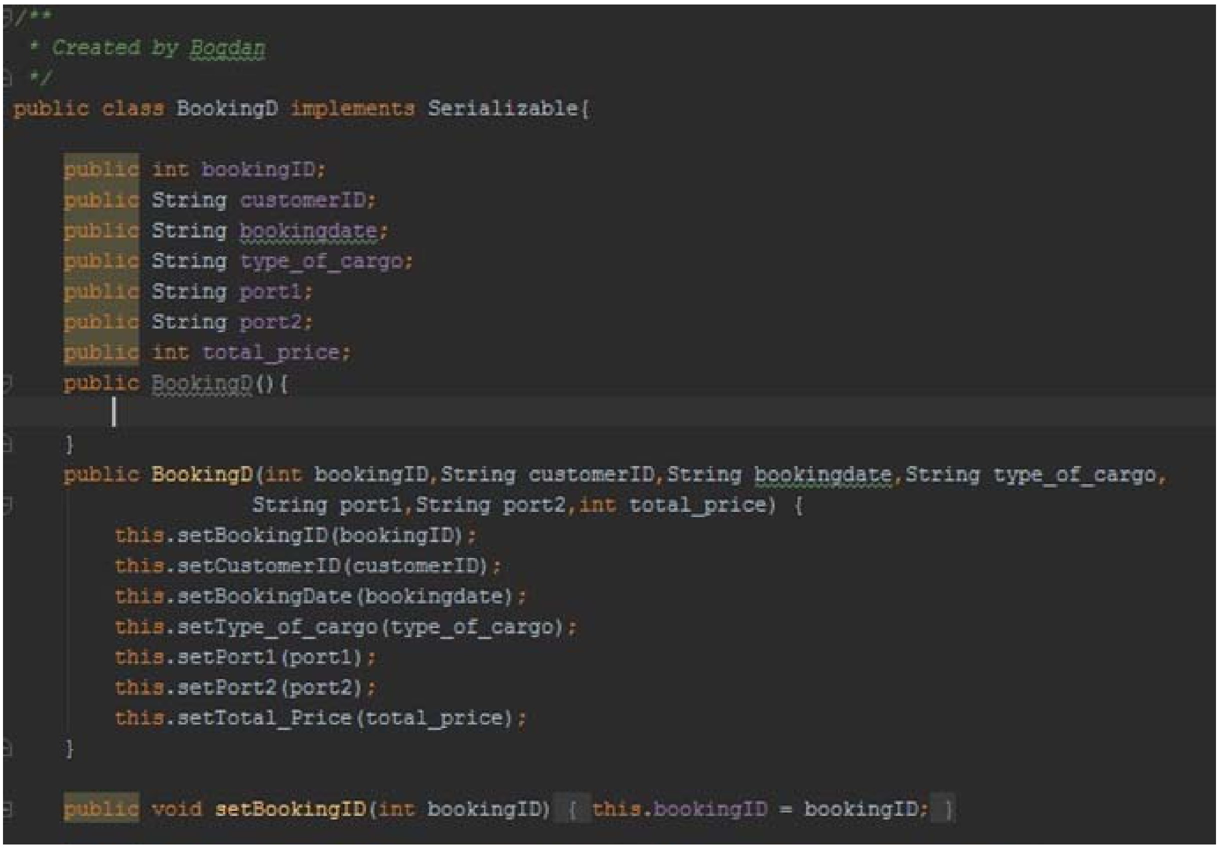


```java
@Override
public boolean equals(Object o) {
    if (!(o instanceof RoutesD)) {
        return false;
    }

    // id must be the same for two Routes to be equal
    RoutesD p = (RoutesD) o;
    if (this.portID == p.getPortID()) {
        return true;
    }
    return false;
}

@Override
public int hashCode() { return this.portID; }
@Override
public String toString() { return "["+"ID: "+portID+" Name of port: "+nameOfport+" Price: RM"+price+"]"+ "\n"; }
```

Figure 5 Method Overriding

In java all classes are inherit from the Object class, directly or indirectly. The Object class has some basic methods like *clone(), toString(), equals(),*.. etc. The default *toString()* method in Object prints “class name @ hash code”. We can override *toString()* method in our class to print proper output. For example, in the following code *toString()* is overridden to print form with different variables. The @Override annotation is optional and indicates that this is expected to be overriding.

Overloading

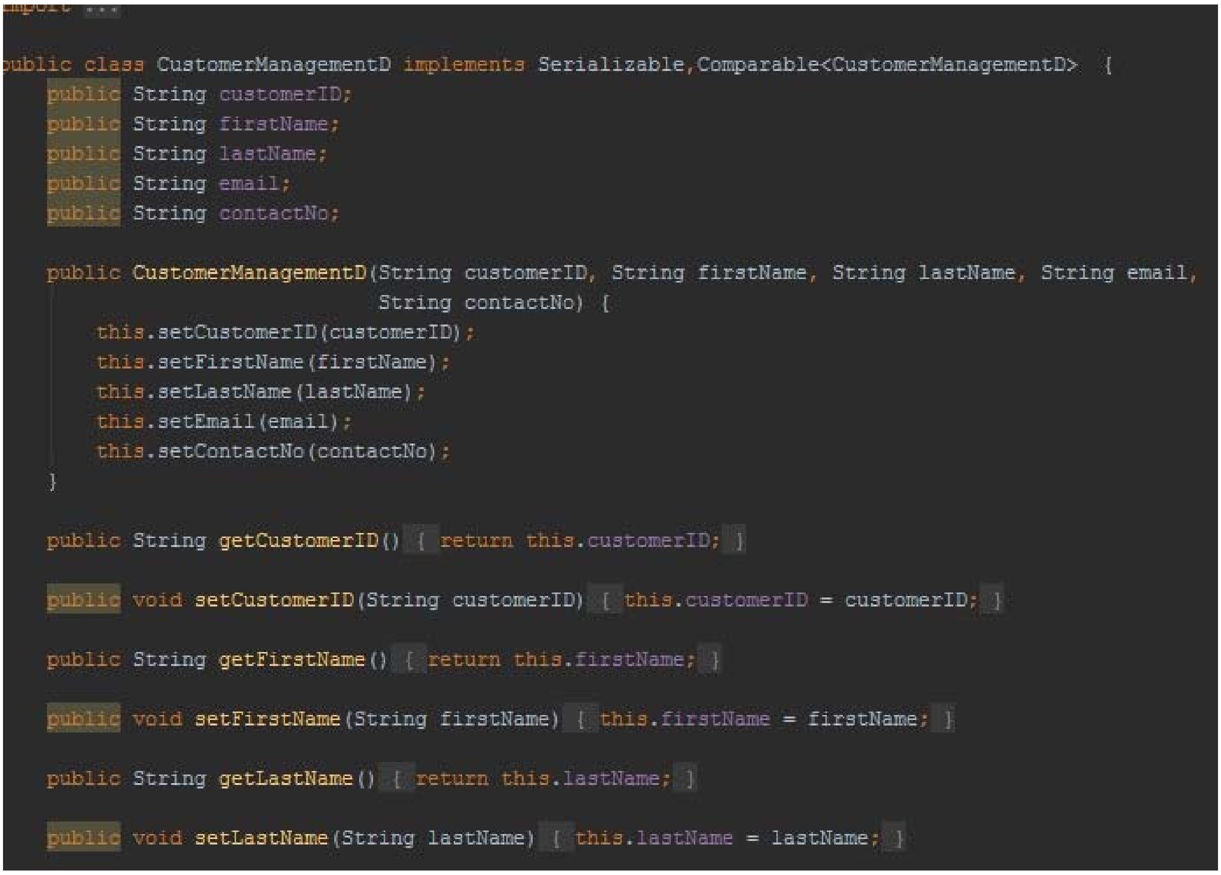


```
/**
 * Created by Bogdan
 */
public class BookingD implements Serializable{

    public int bookingID;
    public String customerID;
    public String bookingdate;
    public String type_of_cargo;
    public String port1;
    public String port2;
    public int total_price;
    public BookingD(){

    }
    public BookingD(int bookingID,String customerID,String bookingdate,String type_of_cargo,
                    String port1,String port2,int total_price) {
        this.setBookingID(bookingID);
        this.setCustomerID(customerID);
        this.setBookingDate(bookingdate);
        this.setType_of_cargo(type_of_cargo);
        this.setPort1(port1);
        this.setPort2(port2);
        this.setTotal_Price(total_price);
    }

    public void setBookingID(int bookingID) { this.bookingID = bookingID; }
```

Figure 6 Overloading

On the above figure in the *public class BookingD* there to constructor being overloaded. First method is empty, whereas second method with the same name *BookingD*, but it has acquiring the parameters, namely: *int bookingID*, *String customerID*, *String bookingdate, String category, String type_of_cargo*, *String port1*, *String port2*, *int total_price.* That is how the Overloading happens.

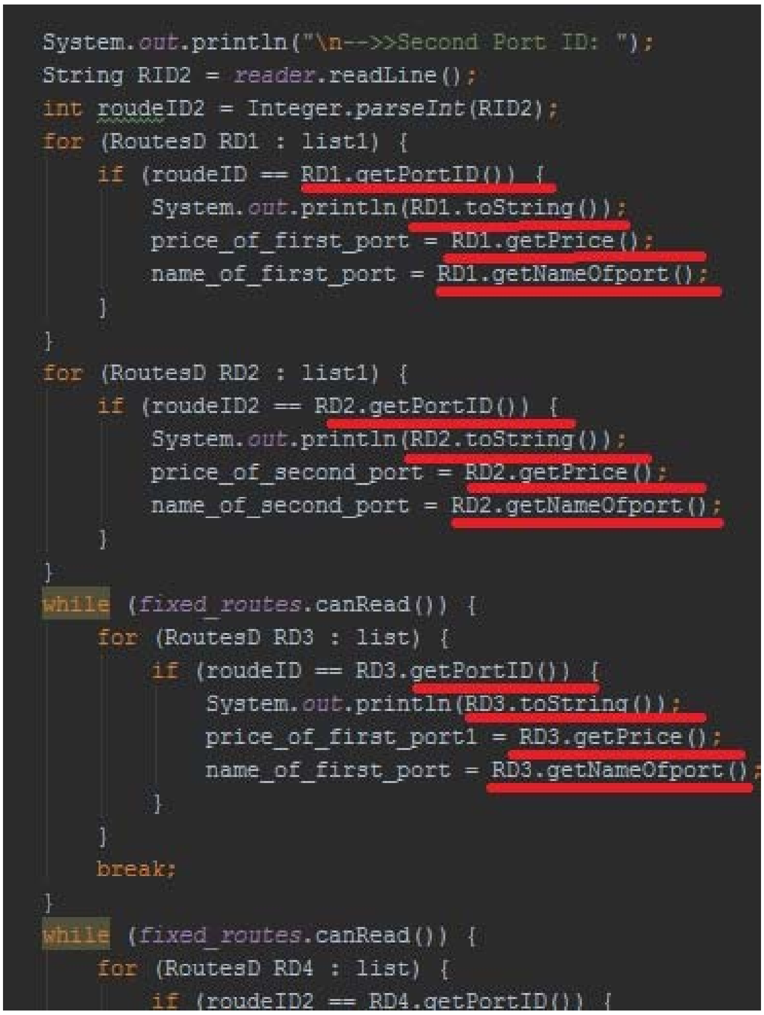
Encapsulation Implementation in Project



```
import ...

public class CustomerManagementD implements Serializable,Comparable<CustomerManagementD>  {
    public String customerID;
    public String firstName;
    public String lastName;
    public String email;
    public String contactNo;

    public CustomerManagementD(String customerID, String firstName, String lastName, String email,
                               String contactNo) {
        this.setCustomerID(customerID);
        this.setFirstName(firstName);
        this.setLastName(lastName);
        this.setEmail(email);
        this.setContactNo(contactNo);
    }

    public String getCustomerID() { return this.customerID; }

    public void setCustomerID(String customerID) { this.customerID = customerID; }

    public String getFirstName() { return this.firstName; }

    public void setFirstName(String firstName) { this.firstName = firstName; }

    public String getLastName() { return this.lastName; }

    public void setLastName(String lastName) { this.lastName = lastName; }
```

Figure 7 Encapsulation

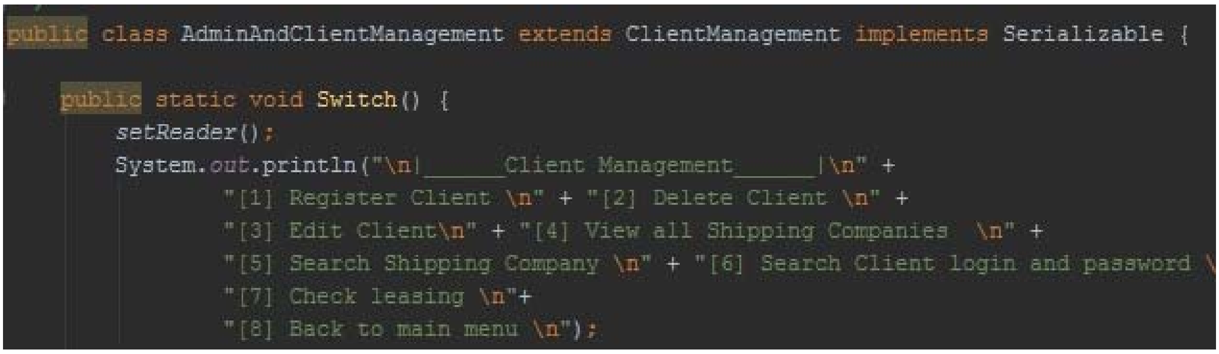
Encapsulation it's fundamental part of Object-oriented. Programmers use encapsulation for protecting data from abuse by the outside world, also it called 'information hiding' or 'data hiding'. I think that encapsulation has a number of advantages that increase the reusability, flexibility and maintainability of the code. As you can see in the above examples, encapsulation is implemented in Java using classes, access modifiers, setters and getters. We use getter and setters when develop a system to be able to refer to them later, furthermore we use just methods which being created, without direct access to the variable. The *figure 8* below represents how I used the getter and setter methods in other class to refer to the variables declared above.



```
System.out.println("\n-->>Second Port ID: ");
String RID2 = reader.readLine();
int roudeID2 = Integer.parseInt(RID2);
for (RoutesD RD1 : list1) {
    if (roudeID == RD1.getPortID()) {
        System.out.println(RD1.toString());
        price_of_first_port = RD1.getPrice();
        name_of_first_port = RD1.getNameOfport();
    }
}
for (RoutesD RD2 : list1) {
    if (roudeID2 == RD2.getPortID()) {
        System.out.println(RD2.toString());
        price_of_second_port = RD2.getPrice();
        name_of_second_port = RD2.getNameOfport();
    }
}
while (fixed_routes.canRead()) {
    for (RoutesD RD3 : list) {
        if (roudeID == RD3.getPortID()) {
            System.out.println(RD3.toString());
            price_of_first_port1 = RD3.getPrice();
            name_of_first_port = RD3.getNameOfport();
        }
    }
    break;
}
while (fixed_routes.canRead()) {
    for (RoutesD RD4 : list) {
        if (roudeID2 == RD4.getPortID()) {
```

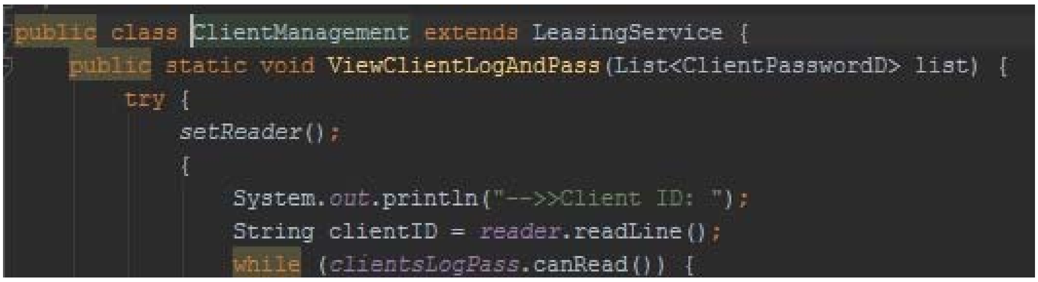
Figure 8 setters and getters encapsulation

Inheritance



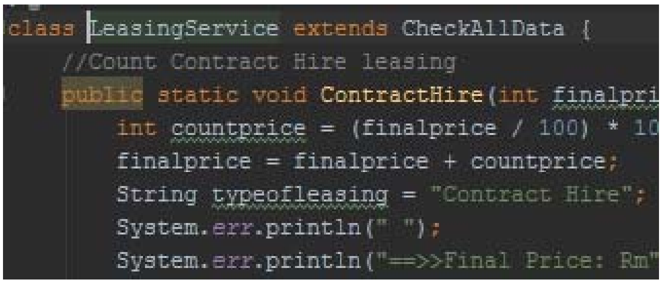
```
public class AdminAndClientManagement extends ClientManagement implements Serializable {

    public static void Switch() {
        setReader();
        System.out.println("\n|______Client Management______|\n" +
                "[1] Register Client \n" + "[2] Delete Client \n" +
                "[3] Edit Client\n" + "[4] View all Shipping Companies  \n" +
                "[5] Search Shipping Company \n" + "[6] Search Client login and password \
                "[7] Check leasing \n"+
                "[8] Back to main menu \n");
```

Figure 9 above shows the implementation of Inheritance, where class AdminClienManagement extends the ClientManagement.



```
public class ClientManagement extends LeasingService {
    public static void ViewClientLogAndPass(List<ClientPasswordD> list) {
        try {
            setReader();
            {
                System.out.println("-->>Client ID: ");
                String clientID = reader.readLine();
                while (clientsLogPass.canRead()) {
```

Figure 10 Inheritance



```
class LeasingService extends CheckAllData {
    //Count Contract Hire leasing
    public static void ContractHire(int finalpri
        int countprice = (finalprice / 100) * 10
        finalprice = finalprice + countprice;
        String typeofleasing = "Contract Hire";
        System.err.println(" ");
        System.err.println("==>>Final Price: Rm"
```

Figure 11 Inheritance

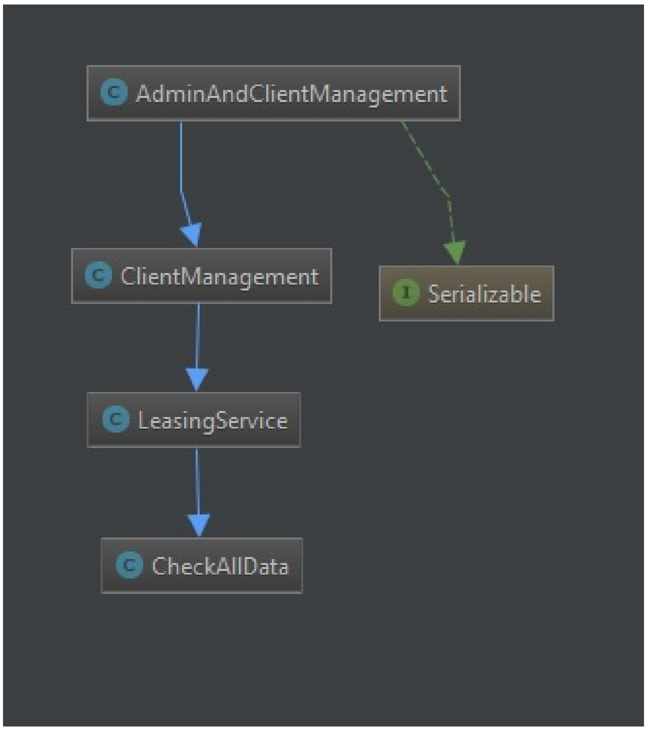


Figure 12 Inheritance

Above figures illustrate implementation of multiple inheritance. The *Figure 12* shows that one class inherits the methods from three classes and one interface. The code on *Figures 9-11* display that *public class AdminAndClientManagement extend ClientManagement,* and inherits Serializable methods of an Interface. Next from the Page class *ClientManagement extend LisingService* and finaly *LeasingService extend CheckAllData*. So I created multiple inheritance to acquiring all the attributes from the *CheckAllData* class to *AdminAndClientManagement* class, because in *AdminAndClientManagement class* I use some methods from *ClientManagement* class. At the same time *ClientManagement* class using methods from *LeasingService* class and finally all above classes use methods from *CheckAllData* class. This is the way how inheritance works in Java. Regarding the Multiple Inheritance, our Assignment Requirements were enough to built a Single Inheritance, so it was not very important to implement the Multiple Inheritance feature.

Conclusion

To sum up, with this assignment I become understand Object Oriented Programming in Java more deep and my coding skills also increased. However, I faced different limitations while create freight management system, such as implementing Serialization, logic flow, because Serialization was new topic for me and it was quite hard to comprehend it. Occasionally, logic failure led to an incorrect implementation of the system, so I've learned how to debug and eliminate errors. Finally, I understand how to work with Polymorphism, Encapsulation, Inheritance etc. Unfortunately, any system has some limitation. In my opinion, limitation of my system is that wasn't implemented GUI and somewhere not clearly written code.

Reference

Eckel, B. (2006). *Thinking in Java*. 1st ed. Upper Saddle River, NJ: Prentice Hall.

development, J. (2016). *Intro to Java programming*. [online] Ibm.com. Available at: https://www.ibm.com/developerworks/learn/java/intro-to-java-course/index.html [Accessed 16 Dec. 2016].

Stackoverflow.com. (2016). *Newest 'java' Questions*. [online] Available at: http://stackoverflow.com/questions/tagged/java [Accessed 16 Dec. 2016].

Stravaganzastravaganza.blogspot.my. (2016). *CATERING SYSTEMS*. [online] Available at: http://stravaganzastravaganza.blogspot.my/2012/03/catering-systems.html [Accessed 17 Dec. 2016].

Docs.oracle.com. (2017). *Moved*. [online] Available at: https://docs.oracle.com [Accessed 23 Jan. 2017].

Horstmann, C. (n.d.). *Java SE 8 for the really impatient*. 1st ed.

Urma, R., Fusco, M. and Mycroft, A. (n.d.). *Java 8 in action*. 1st ed.