

Name: Jason Gunn

Number: 06039966

Unit: Further Programming

Unit Code: CMP1024

Assignment Number: 2

Unit Lecturer: Stephen Rank

Contents

Page 1 – Front Cover
Page 2 – Contents
Page 3 to 7 – Use Case Diagram
Page 7 to 9 – Class Diagram
Page 10 to 43 – Source Code
Page 43 to 55 – Testing
Page 43 – Bibliography

Task 1 – UML Use Case Diagram

Step 1:

First I took the problem I was given, and highlighted the key actors, classes, etc.

Fitzroy Bailey, the owner of Rockall Vehicle Rental Agency, wishes to set up a stock-control system to manage hiring, invoicing, and customer records. Bailey should be able to determine how many vehicles are on hire at a given moment, and what the rental income to date is. The shop hires bicycles by the half-day, cars by the week (with an additional charge if a car is used for more than an average of 1,000 miles/week), and helicopters by the hour. When a customer hires a vehicle, an employee must record their name and address, and ensure that the hiring information is recorded, including the length of time for which the vehicle has been hired. Everyone who hires a vehicle must pay for it at the time of hiring.

Step 2:

I then took these words and sorted them into a logical order:

Rockall Vehicle Rental Agency

Shop

rental income

stock-control system

Fitzroy Bailey

Bailey

employee

customer

customer records

name

address

invoicing

Hire

Hiring

length

pay

vehicle

Vehicles

bicycle

cars

helicopters

Step 3:

Next I looked at the list, and took the most appropriate things to use:

[Shop](#)
[employee](#)
[customer records](#)
[name](#)
[address](#)
[invoicing](#)
[vehicle](#)
[bicycle](#)
[cars](#)
[helicopters](#)

Step 4:

I then needed to decided what each use case would be:

A Customer rents a vehicle

A Customer returns a vehicle

The length of time a vehicle was out was checked

Extra costs are issued if need be

When A vehicle is to be rented, stock must be checked

The customer needs to be invoiced

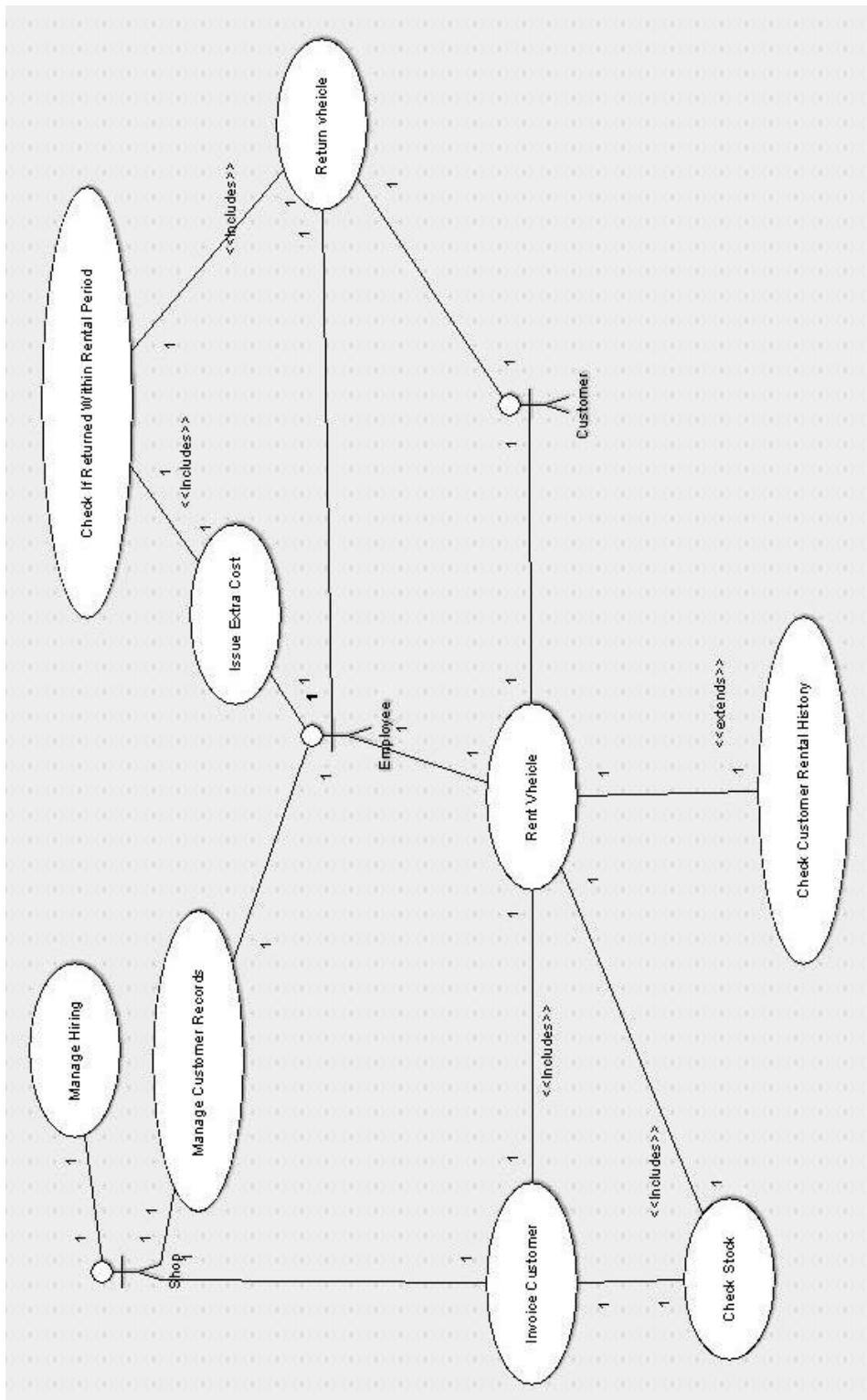
The hiring of staff needs to be managed

Customers records need to be managed

Check Customer Rental History

Step 5:

I then took my list, and created my first diagram:



“A customer rents a vehicle”. This is related to both the customer, who wishes to rent, and the employee who must arrange the rental. The employee will perform a “check on the customers rental history” to see if he or she is will be loaned a vehicle, this action extends the “Rent vehicle” use case.

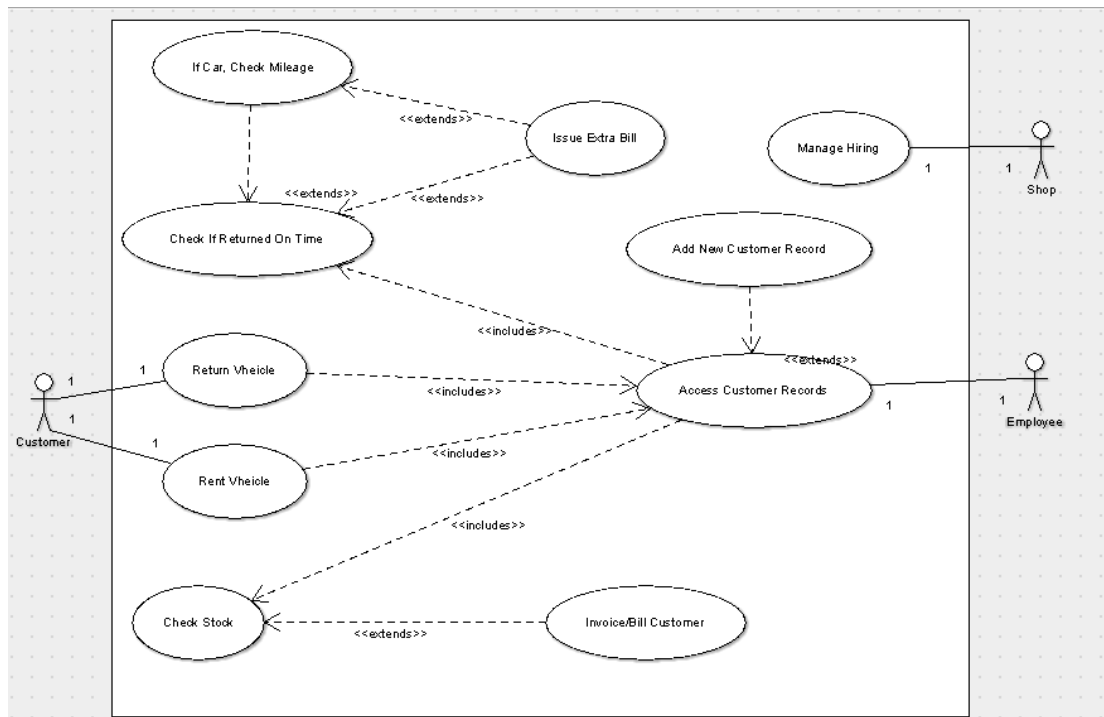
Once it has been decided that the customer will rent a vehicle, the action of renting the vehicle includes two actions, “invoicing the customer” for the cost of the rental (getting payment) and making sure they have what the “stock” the customer wants.

The customer also needs to “return the vehicle”, the employee again deals with this, the process of returning the vehicle included “checking how long the vehicle was out” and “issuing any extra costs” resulting in extended rentals, this will also be added to the “customers records”.

The employee will update all of the “customers records”, with details such as current rentals, reliability, return times etc. Finally the shop (or boss) will deal with hiring, and firing of staff, meaning they relate to the employee, they will also have a copy of the customer records, and be responsible for preparing the customer invoices.

Step 6:

I then took my initial (Messy) diagram, and refined it to comply with the UML Use Case Standards:



A customer, can “rent” or “return” a vehicle from the shop, each time they come in their record will be accessed, “if” they are a new customer, then a new record will be added.

If they are returning then the system will check if they returned the vehicle on time, it will also check its mileage if it is a car, and “if” need be an additional bill will be issued to the customer.

When the customer is renting, then first stock will be checked, and if the vehicle they want is in stock they will be invoiced/billed.

The employee is in charge of managing the customers records, adding additional information etc.

The shop manages the hiring of staff.

Task 2 – A UML Class Diagram

Step 1:

I again took my final list from Task 1, as these are the elements that should be used in my diagram:

Shop
employee
customer records
name
address
invoicing
vehicle
bicycle
cars
helicopters

Step 2:

I then needed to decide which items would be my classes, and which classes would have what belonging to them.

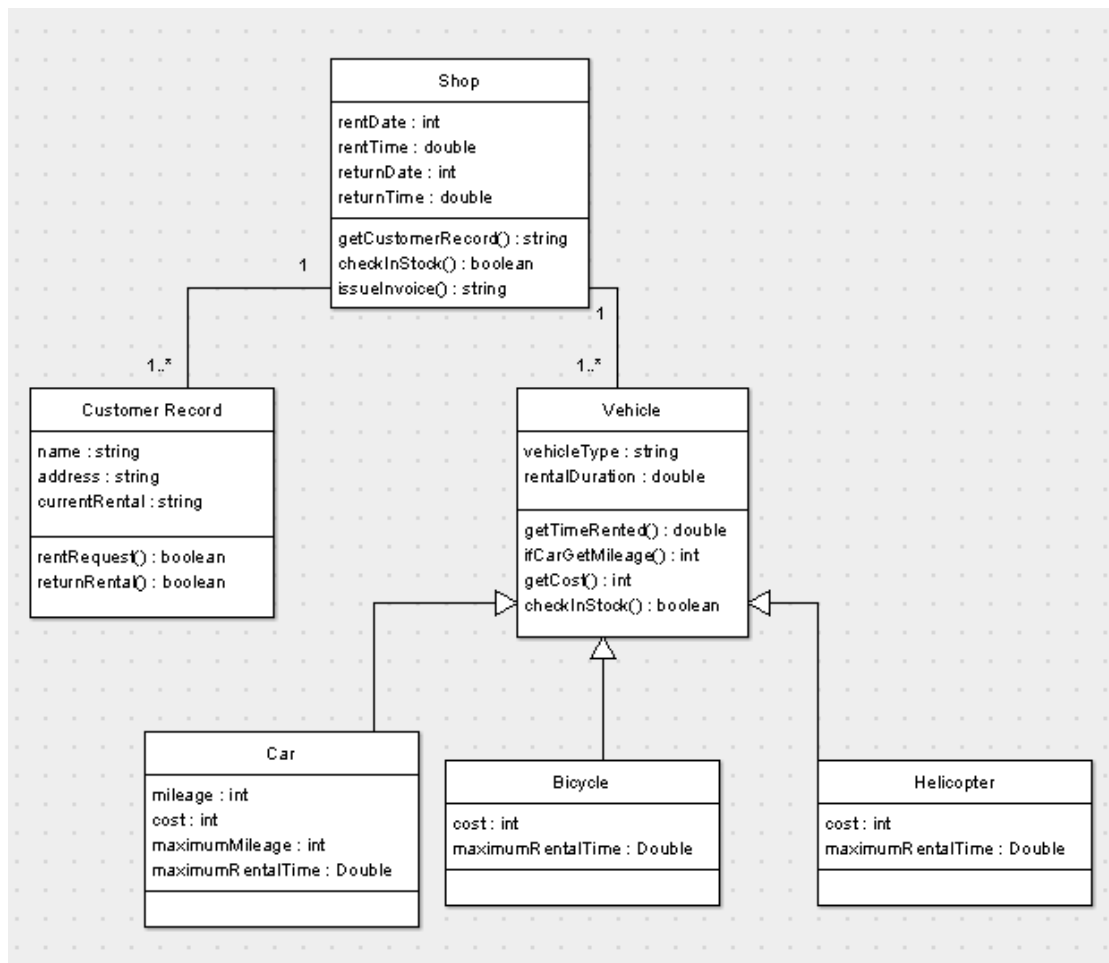
Shop
Employee
Invoicing

Customer Record
Name
Address

Vehicle
Bicycle
Cars
Helicopters

Step 3:

I then created my first diagram:



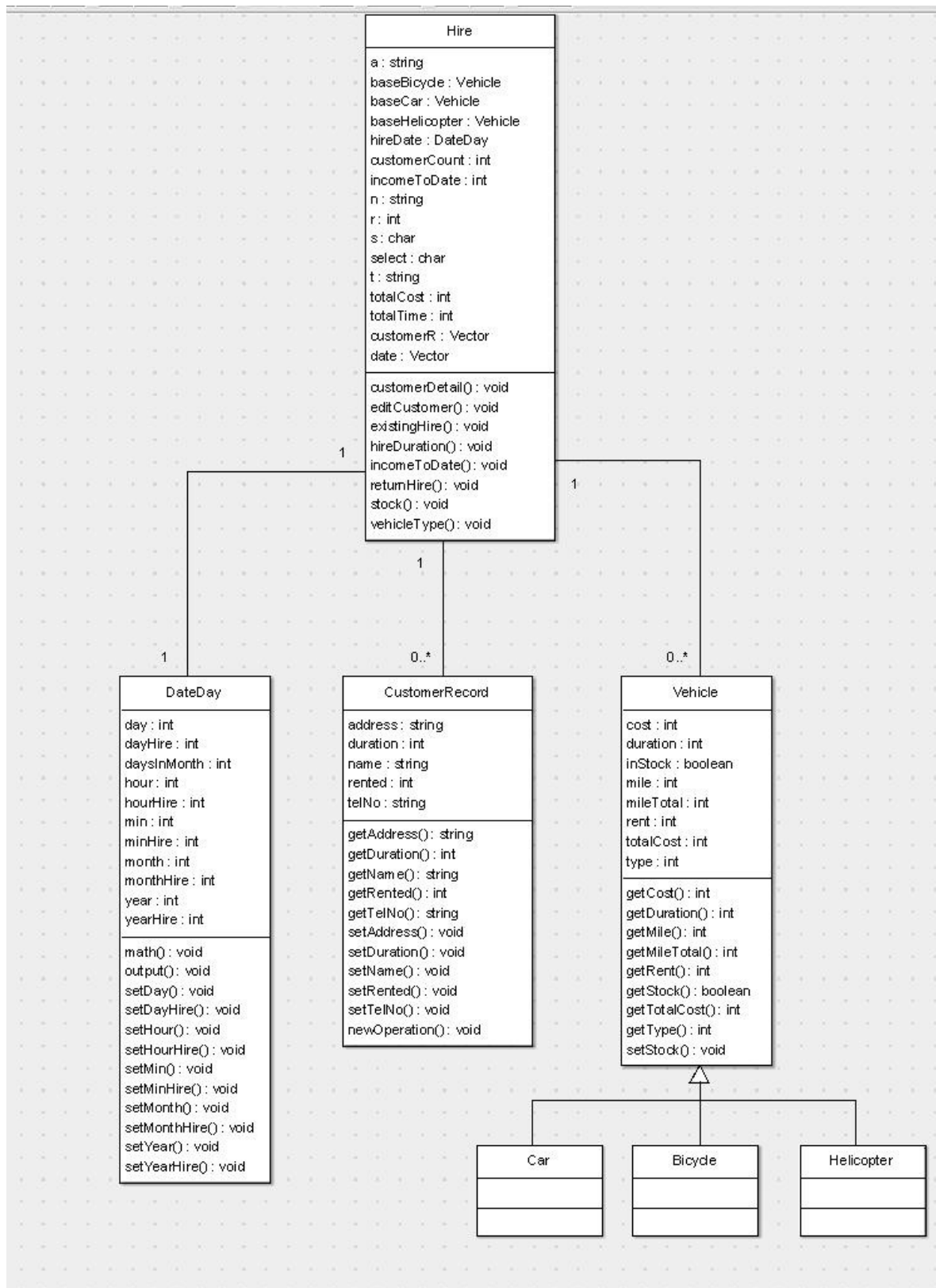
Shop accesses vehicle information and customer records. It stores the dates, and times in which vehicles are rented and returned, it also finds out if what the customer wants is in stock, and issues them with invoices.

Customer records holds information on the customer, their name, address and if they are currently renting anything, it also deals with rent requests and rent returns.

Vehicle stores information on the vehicles currently in stock, car, bicycle and helicopter, it finds out the mileage (If a car) and the rental duration. It also gets the cost of renting the vehicle for the set amount of time.

Step 4:

Thinking harder I realised that this was not correctly implementing Object Orientated programming, I also decided I would need more methods etc in order to create the program I intended to, thus I created this diagram:



Hire deals with customers, allowing them to rent a vehicle, and set the date in which they rent it. It also deals with returning a rental and updating customer details.

Vehicle holds all the information for vehicles, with car, bicycle and helicopter, inheriting this information, and giving them the ability to override the information.

Task 3 - Source Code

Main

```
/*This is the main class which calls methods from other classes, and gives the
program some functionality.*/

public class Main {
    public static void main(String[]args) { //This enables the program to be run in the
bluej console.
        int menu; //This is used in selecting options for the menu.
        int val = 0; //This is used to validate the menu.
        boolean loop = false; //This is used in a while loop.
        Hire newHire = new Hire(); //This creates a new hire object for use in the
program.

        while(!loop) { //This states that while loop is false, the loop will continue.
            System.out.println(""); //The following show the user what to input, and what
will happen when they input.
            System.out.println("To Use This System, Enter The Number You Want To
Use, Followed by Pressing Enter");
            System.out.println("Press 1# To Set Up A Rental For A New Customer");
            System.out.println("Press 2# To Set Up A Rental For An Existing Customer");
            System.out.println("Press 3# See What Vehicles Are In Stock");
            System.out.println("Press 4# If A Customer Is Returning Their Rental
Vehicle");
            System.out.println("Press 5# To Show The Income To Date");
            System.out.println("Press 6# To Edit An Existing Customers Details");
            System.out.println("Press 0# To Shut The System Down");
            System.out.println("");
            menu = TextIO.getInt(); //This gets into menu, and if appropriate will take
them to the corresponding attribute.
            if(menu == 1) { //This is what happens if the customer wishes to set up a new
rental for a new customer.
                newHire.vehicleType(); //This is run to select the type of vehicle the
customer wants.
                newHire.hireDuration(); //This is run to select how long they wish to rent
teh vehicle for.
                newHire.customerDetail(); //This allows the new customers details to be set
up, and stored in teh system.
                val = 1; //This sets val to 1, meaning other options on the main menu can
now be used.
            }
            else if(menu == 2 && val == 1) {
                newHire.existingHire(); //This will only occur if at least one customer has
been stored in the program, it will enable an existing customer to set up a new rental,
provided they do not have a current one.
            }
            else if(menu == 2 && val == 0) { //If there are no customers then this option
will throw an error to the user, and dump them back at the main menu.

```


protected char select; //This is used to manage which vehicle the customer wishes to rent.

protected char s; //This is also used to manage the vehicle the customer wishes to rent.

protected int totalCost; //This is used to work out the total cost.

protected int totalTime; //This is used to work out the total time of rental.

protected int incomeToDate; //This is used to store the total income to date.

protected String n; //The following strings are used in creating a new customer object.

protected String a;

protected String t;

protected int r; //This is used to set what type of vehicle is being rented, into the customer record.

Vector customerR = new Vector(); //This creates a vector for use with customer records.

Vector date = new Vector(); //This creates a vector for use with dates.

Car baseCar = new Car(0,0,0,0,true,0,0,0); //This creates a new car object.

Helicopter baseHelicopter = new Helicopter(0,0,0,0,true,0,0,0); //This creates a new helicopter object.

Bicycle baseBicycle = new Bicycle(0,0,0,0,true,0,0,0); //This creates a new bicycle object.

DateDay hireDate = new DateDay(1,1,1900,1,1,1,1,1900,1,1); //This creates a new date object.

public void stock() { //This class is used to show what vehicles are currently in stock, if a vehicle is in stock then a message will show this, if it is not in stock a message will also show this.

if(baseCar.getStock() == true) { //This finds out from the object if a value is true, or false, and then displays the appropriate message.

System.out.println("The Car Is In Stock");

}

else if(baseCar.getStock() == false) {

System.out.println("Sorry But The Car Is Not In Stock");

}

if(baseHelicopter.getStock() == true) {

System.out.println("The Helicopter Is In Stock");

}

else if(baseHelicopter.getStock() == false){

System.out.println("Sorry But The Helicopter Is Not In Stock");

}

if(baseBicycle.getStock() == true) {

System.out.println("The Bicycle Is In Stock");

}

else if(baseBicycle.getStock() == false) {

System.out.println("Sorry But The Bicycle Is Not In Stock");

}

}

```
public void vehicleType() { //This code is used to select which vehicle a new
customer wishes to rent.

    boolean breakLoop = false; //This is used to create a loop.

    while(!breakLoop){ //The user needs to input what type of vehicle they wish to
rent, a car, helicopter or bike, if it is in stock then the program will allow it to be
rented, else if it is not it will throw an error message.
        System.out.println("Please Select The Type Of Vehicle, C For Car, H For
Helicopter, B For Bicycle");
        s = TextIO.getChar(); //This is used to store which type of vehicle the
customer wants.
        select = s;

        if(s == 'c' && baseCar.getStock() == true){
            System.out.println("Car Cost To Rent £" + baseCar.getCost() + " Mileage "
+ baseCar.getMile() + " Max rental time per rental " + baseCar.getRent() + " Hours");
            r = 1;
            breakLoop = true;
        }
        else if(s == 'c' && baseCar.getStock() == false) {
            System.out.println("I'm Sorry The Car Is Not Currently Available For
Hire");
            breakLoop = true;
        }
        else if(s == 'h' && baseHelicopter.getStock() == true){
            System.out.println("Helicopter Cost To Rent £" + baseHelicopter.getCost()
+ " Max rental time per rental " + baseHelicopter.getRent() + " Hour");
            r = 2;
            breakLoop = true;
        }
        else if(s == 'h' && baseHelicopter.getStock() == false) {
            System.out.println("I'm Sorry The Helicopter Is Not Currently Available
For Hire");
            breakLoop = true;
        }
        else if(s == 'b' && baseBicycle.getStock() == true){
            System.out.println("Bicycle Cost To Rent £" + baseBicycle.getCost() + "
Max rental time per rental " + baseBicycle.getRent() + " Hours");
            r = 3;
            breakLoop = true;
        }
        else if(s == 'b' && baseBicycle.getStock() == false) {
            System.out.println("I'm Sorry The Bicycle Is Not Currently Available For
Hire");
            breakLoop = true;
        }
        else { //If the customer enters invalid data, an error message will be thrown,
and the loop will restart.
```

```

        System.out.println("Sorry You Have Entered Invalid Data, Please Try
Again!");
        breakLoop = false;
    }
}
}
public void hireDuration() {

```

/*This section of code is used to find out how long the customer wants to hire a vehicle for.

* The system allows only a multiple of set rental periods to be entered, for example if a customer wishes to rent a bike for 24 hours, then the number of times they will rent it will be 2.

```
*/
```

```
boolean breakLoop = false; //Again this is a loop used to validate the code.
```

while(!breakLoop) { //If the vehicle wanted is not in stock, then this loop will be broken.

```

    if(select == 'c' && baseCar.getStock() == false) {
        breakLoop = true;
    }
    else if(select == 'h' && baseHelicopter.getStock() == false) {
        breakLoop = true;
    }
    else if(select == 'b' && baseBicycle.getStock() == false) {
        breakLoop = true;
    }
}

```

else { //This section of code gets the input from the user.

```

    System.out.println("Please Enter How Long The Customer Wishes To Rent
A Vehicle For, This Can Only Be In Multiples Of The Time We Offer, E.G 2x On A
Bicycle = 24 Hours. Lowest Multiple 1, Highest 10");

```

```

    int d;
    int c;
    int m;
    int r;
    d = TextIO.getInt();

```

```

    if(d > 0 && d < 11) {

```

if(select == 'c' && baseCar.getStock() == true) { //For each section of code, the total duration, mile limit and cost will be calculated, and a new object will be created. The total cost, and time will also be stored, and the income to date calculated.

```

        c = baseCar.getCost()*d;
        m = baseCar.getMile()*d;
        r = baseCar.getRent()*d;
        Car customerQuoteCar = new Car(0,0,0,0,true,r,c,m);

```

```

        System.out.println("Total To Pay £" +
customerQuoteCar.getTotalCost() + " The Customer Can Use Up To " +

```

```

customerQuoteCar.getMileTotal() + " Miles" + " They Must Return The Car In " +
customerQuoteCar.getDuration() + " Hours");
    totalCost = customerQuoteCar.getTotalCost();
    totalTime = customerQuoteCar.getDuration();
    incomeToDate = incomeToDate + customerQuoteCar.getTotalCost();
    breakLoop = true;
}
else if(select == 'h' && baseHelicopter.getStock() == true) {
    c = baseHelicopter.getCost()*d;
    m = baseHelicopter.getMile()*d;
    r = baseHelicopter.getRent()*d;
    Helicopter customerQuoteHelicopter = new
Helicopter(0,0,0,0,true,r,c,m);
    if(d == 1){
        System.out.println("Total To Pay £" +
customerQuoteHelicopter.getTotalCost() + " They Must Return The Helicopter In " +
customerQuoteHelicopter.getDuration() + " Hour");
        totalCost = customerQuoteHelicopter.getTotalCost();
        totalTime = customerQuoteHelicopter.getDuration();
        incomeToDate = incomeToDate +
customerQuoteHelicopter.getTotalCost();
        breakLoop = true;
    }
    else {
        System.out.println("Total To Pay £" +
customerQuoteHelicopter.getTotalCost() + " They Must Return The Bicycle In " +
customerQuoteHelicopter.getDuration() + " Hours");
        totalCost = customerQuoteHelicopter.getTotalCost();
        totalTime = customerQuoteHelicopter.getDuration();
        incomeToDate = incomeToDate +
customerQuoteHelicopter.getTotalCost();
        breakLoop = true;
    }
}
else if(select == 'b' && baseBicycle.getStock() == true) {
    c = baseBicycle.getCost()*d;
    m = baseBicycle.getMile()*d;
    r = baseBicycle.getRent()*d;
    Bicycle customerQuoteBicycle = new Bicycle(0,0,0,0,true,r,c,m);
    System.out.println("Total To Pay £" +
customerQuoteBicycle.getTotalCost() + " They Must Return The Car In " +
customerQuoteBicycle.getDuration() + " Hours");
    totalCost = customerQuoteBicycle.getTotalCost();
    totalTime = customerQuoteBicycle.getDuration();
    incomeToDate = incomeToDate +
customerQuoteBicycle.getTotalCost();
    breakLoop = true;
}
}
}
}

```

else { //If the user enters an invalid amount the loop will reset, and throw an error message.

```
System.out.println("Sorry You Have Entered Invalid Data, Please Try Again!");
```

```
breakLoop = false;
```

} //This section of code allows the user to input the hire date and time for a vehicle.

```
System.out.println("Please Enter The Month Of Hire");
```

```
hireDate.setMonthHire();
```

```
System.out.println("Please Enter The Day Of Hire");
```

```
hireDate.setDayHire();
```

```
System.out.println("Please Enter The Year Of Hire");
```

```
hireDate.setYearHire();
```

```
System.out.println("Please Enter The Hour Of Hire");
```

```
hireDate.setHourHire();
```

```
System.out.println("Please Enter The Minute Of Hire");
```

```
hireDate.setMinHire();
```

```
}
```

```
}
```

```
}
```

public void customerDetail() { //This section of code gets the customers details, address, name, telephone number etc, It will also store the vehicle rented, and total durations into their record.

```
int val;
```

```
int val2;
```

```
int val3;
```

```
int c = 1;
```

```
int b = 1;
```

```
int ab = 1;
```

boolean loopBreak = false; //These are used for validation in for loops to make sure the system is as bug free as possible..

```
boolean loopBreak2 = false;
```

```
boolean loopBreak3 = false;
```

```
customerCount++;
```

while(!loopBreak3) { //If the vehicle the customer wants is not in stock the loop will be broken.

```
if(select == 'c' && baseCar.getStock() == false) {
```

```
loopBreak3 = true;
```

```
}
```

```
else if(select == 'h' && baseHelicopter.getStock() == false) {
```

```
loopBreak3 = true;
```

```
}
```

```
else if(select == 'b' && baseBicycle.getStock() == false) {
```

```
loopBreak3 = true;
```

```
}
```

```
else {
```

```

        for(int counter = ab;counter<2;counter++) { //This is a for loop, used in
validation, it will get the customers name, and check that it is correct. If not the user
will be able to try again.
            System.out.println("Please Input The Customers Name");
            n = TextIO.getLnString();
            if(n == null){
                n = TextIO.getLnString();
            }
            else {
                n = TextIO.getLnString();
            }
            System.out.println("The Name You Entered Is: " + n);
            System.out.println(" Is This Correct? Enter 1 For Yes Or 0 For No!");
            ab = 2;
        }
        val3 = TextIO.getInt();
        if(val3 == 1) {
            System.out.println("Thank You!");
            loopBreak3 = true;
            ab = 2;
        }
        else if(val3 == 0) {
            System.out.println("Please Enter The Name Again!");
            ab = 1;
            loopBreak3 = false;
        }
        else { //If the customerr enters invalid data, an error message will be
thrown, and the loop will be reset.
            System.out.println("I'm Sorry The Letter/Number You Entered Is Not
Valid, Please Try Again!");
            loopBreak3 = false;
        }
    }
}
while(!loopBreak) { //This loop is broken again, if the requested vehicle is out of
stock.
    if(select == 'c' && baseCar.getStock() == false) {
        loopBreak = true;
    }
    else if(select == 'h' && baseHelicopter.getStock() == false) {
        loopBreak = true;
    }
    else if(select == 'b' && baseBicycle.getStock() == false) {
        loopBreak = true;
    }
    else {
        for(int counter = c;counter<2;counter++) { //This section of code gets the
customers address, and allows the user to input it again should they make a mistake.
            System.out.println("Please Input The Customers Address");
            a = TextIO.getLnString();

```

```

        if(a == null){
            a = TextIO.getLnString();
        }
        else {
            a = TextIO.getLnString();
        }
        System.out.println("The Address You Entered Is: " + a);
        System.out.println(" Is This Correct? Enter 1 For Yes Or 0 For No!");
        c = 2;
    }
    val = TextIO.getInt();
    if(val == 1) {
        System.out.println("Thank You!");
        loopBreak = true;
        c = 2;
    }
    else if(val == 0) {
        System.out.println("Please Enter The Address Again!");
        c = 1;
        loopBreak = false;
    }
    else { //If invalid data is input the loop will restart, and an error message
thrown.
        System.out.println("i'm Sorry The Letter/Number You Entered Is Not
Valid, Please Try Again!");
        loopBreak = false;
    }
}
}
}
while(!loopBreak2) { //If the vehicle requested is not in stock, the loop will be
broken.
    if(select == 'c' && baseCar.getStock() == false) {
        loopBreak2 = true;
    }
    else if(select == 'h' && baseHelicopter.getStock() == false) {
        loopBreak2 = true;
    }
    else if(select == 'b' && baseBicycle.getStock() == false) {
        loopBreak2 = true;
    }
    else {
        for(int counter = b;counter<2;counter++) { //This gets the customers phone
number, and if entered wrong can be input again due to validation.
            System.out.println("Please Input The Customers Phone Number");
            t = TextIO.getLnString();
            if(t == null){
                t = TextIO.getLnString();
            }
            else {
                t = TextIO.getLnString();
            }
        }
    }
}

```

```

    }
    System.out.println("The Number You Entered Is: " + t);
    System.out.println(" Is This Correct? Enter 1 For Yes Or 0 For No!");
    b = 2;
}
val2 = TextIO.getInt();
if(val2 == 1) {
    System.out.println("Thank You!");
    loopBreak2 = true;
    b = 2;
}
else if(val2 == 0) {
    System.out.println("Please Enter The Number Again!");
    b = 1;
    loopBreak2 = false;
}
else { //Iff the customer inputs invalid data the loop will restart and an error
will be given.
    System.out.println("i'm Sorry The Letter/Number You Entered Is Not
Valid, Please Try Again!");
    loopBreak2 = false;
}
}
}
}
if(select == 'c' && baseCar.getStock() == true) { //If the customer wishes to hire
a car, and it is in stock, then a new customer record will be created.
    CustomerRecord customer = new CustomerRecord(n, a, t, r, totalTime);
    customerR.add(customer); //The customers record will be added to the array.
    System.out.println("The Customers Name Is:"); //Details about the customer
will be printed out.
    System.out.println(customer.getName());
    System.out.println("The Customers Address Is:");
    System.out.println(customer.getAddress());
    System.out.println("The Customers Telephone Number Is:");
    System.out.println(customer.getTelNo());
    date.add(hireDate); //The date the customer hired the vehicle will also be
stored in an array.
    System.out.println("They Need To Pay A Total Of: £" + totalCost); //The total
amount to pay will be output, and the user will then process this externally.
    System.out.println("There Customer Number Is: " + customerCount + " Please
Inform Them To Hold Onto This For Use In Returning Their Vehicle"); //This gives
the customer number, which is needed when the customer is returning their vehicle, or
wanting to set up a new rental.
    baseCar.setStock(false);
}
else if(select == 'h' && baseHelicopter.getStock() == true) { //This code works
as above, however is used if the customer wishes to rent a helicopter, and the stock is
available.
    CustomerRecord customer = new CustomerRecord(n, a, t, r, totalTime);
    customerR.add(customer);

```

```

        System.out.println("The Customers Name Is:");
        System.out.println(customer.getName());
        System.out.println("The Customers Address Is:");
        System.out.println(customer.getAddress());
        System.out.println("The Customers Telephone Number Is:");
        System.out.println(customer.getTelNo());
        date.add(hireDate);
        System.out.println("They Need To Pay A Total Of: £" + totalCost);
        System.out.println("There Customer Number Is: " + customerCount + " Please
Inform Them To Hold Onto This For Use In Returning Their Vehicle");
        baseHelicopter.setStock(false);
    }
    else if(select == 'b' && baseBicycle.getStock() == true) { //This code also works
as above, but is used if the customer wishes to rent a Bicycle and it is in stock.
        CustomerRecord customer = new CustomerRecord(n, a, t, r, totalTime);
        customerR.add(customer);
        System.out.println("The Customers Name Is:");
        System.out.println(customer.getName());
        System.out.println("The Customers Address Is:");
        System.out.println(customer.getAddress());
        System.out.println("The Customers Telephone Number Is:");
        System.out.println(customer.getTelNo());
        date.add(hireDate);
        System.out.println("They Need To Pay A Total Of: £" + totalCost);
        System.out.println("There Customer Number Is: " + customerCount + " Please
Inform Them To Hold Onto This For Use In Returning Their Vehicle");
        baseBicycle.setStock(false);
    }
    else {
    }
}
public void existingHire() { //This section of code is used if an existing customer
wishes to set up a new rental, it is very similar to the above code, with some changes
so that existing records are updated, and not new ones created.

```

```

    int selectNo = -1;
    int validate;
    boolean loop = false;
    boolean loopBreak = false;
    boolean breakLoop = false;
    boolean breakLoop2 = false;

```

```

    while(!loop) { //This section of code gets the customers number, and if invalid
will ask for it to be input again.
        System.out.println("Please Enter The Customer's Customer Number");
        selectNo = TextIO.getInt();
        if(selectNo > customerCount) {
            System.out.println("i'm Sorry This Customer Does Not Exist, Please Try
Again!");
            loop = false;

```

```

    }
    else if(selectNo <0) {
        System.out.println("i'm Sorry This Customer Does Not Exist, Please Try
Again!");
        loop = false;
    }
    else {
        loop = true;
    }
}

```

//This section of code calls the customers details from the vector, based on their customer number, and then prints out their details.

```

CustomerRecord customer = (CustomerRecord) customerR.get(selectNo);
System.out.println("Customer Name: " + customer.getName());
System.out.println("Customer Address: " + customer.getAddress());
System.out.println("Customer Telephone Number: " + customer.getTelNo());
while(!loopBreak) {
    System.out.println("Are These Details Correct? Enter 1 For Yes, Or 0 For
No");
    validate = TextIO.getInt();
    if(validate == 1) { //If the customer says the details are correct, then the
system will check if the customer currently has anything on hire, if yes then the loop
will end.
        if(customer.getRented() == 1) {
            System.out.println("Sorry But A Customer Can Only Rent One Vehicle
At A Time");
            loopBreak = true;
        }
        if(customer.getRented() == 2) {
            System.out.println("Sorry But A Customer Can Only Rent One Vehicle
At A Time");
            loopBreak = true;
        }
        if(customer.getRented() == 3) {
            System.out.println("Sorry But A Customer Can Only Rent One Vehicle
At A Time");
            loopBreak = true;
        }
        else if(customer.getRented() == 4) {
            while(!breakLoop){ //This code will then allow the user to input what
vehicle is wanted. And check if it is in stock.
                System.out.println("Please Select The Type Of Vehicle, C For Car, H
For Helicopter, B For Bicycle");
                s = TextIO.getChar();
                select = s;

                if(s == 'c' && baseCar.getStock() == true){

```

```

        System.out.println("Car Cost To Rent £" + baseCar.getCost() + "
Mileage " + baseCar.getMile() + " Max rental time per rental " + baseCar.getRent() +
" Hours");
        r = 1;
        breakLoop = true;
    }
    else if(s == 'c' && baseCar.getStock() == false) {
        System.out.println("i'm Sorry The Car Is Not Currently Available
For Hire");
        breakLoop = true;
    }
    else if(s == 'h' && baseHelicopter.getStock() == true){
        System.out.println("Helicopter Cost To Rent £" +
baseHelicopter.getCost() + " Max rental time per rental " + baseHelicopter.getRent()
+ " Hour");
        r = 2;
        breakLoop = true;
    }
    else if(s == 'h' && baseHelicopter.getStock() == false) {
        System.out.println("i'm Sorry The Helicopter Is Not Currently
Avaliable For Hire");
        breakLoop = true;
    }
    else if(s == 'b' && baseBicycle.getStock() == true){
        System.out.println("Bicycle Cost To Rent £" +
baseBicycle.getCost() + " Max rental time per rental " + baseBicycle.getRent() + "
Hours");
        r = 3;
        breakLoop = true;
    }
    else if(s == 'b' && baseBicycle.getStock() == false) {
        System.out.println("i'm Sorry The Bicycle Is Not Currently
Avaliable For Hire");
        breakLoop = true;
    }
    else {
        System.out.println("Sorry You Have Entered Invalid Data, Please
Try Again!");
        breakLoop = false;
    }
}
while(!breakLoop2) {
    if(select == 'c' && baseCar.getStock() == false) {
        breakLoop2 = true;
    }
    else if(select == 'h' && baseHelicopter.getStock() == false) {
        breakLoop2 = true;
    }
    else if(select == 'b' && baseBicycle.getStock() == false) {
        breakLoop2 = true;
    }
}

```

```

    }
    else { //This section of code allows the duration of rent to be input.
        System.out.println("Please Enter How Long The Customer Wishes
To Rent A Vehicle For, This Can Only Be In Multiples Of The Time We Offer, E.G
2x On A Bicycle = 24 Hours. Lowest Multiple 1, Highest 10");
        int d;
        int c;
        int m;
        int r;
        d = TextIO.getInt();

        if(d > 0 && d < 11) {
            if(select == 'c' && baseCar.getStock() == true) {
                c = baseCar.getCost()*d;
                m = baseCar.getMile()*d;
                r = baseCar.getRent()*d;
                Car customerQuoteCar = new Car(0,0,0,0,true,r,c,m);
                System.out.println("Total To Pay £" +
customerQuoteCar.getTotalCost() + " The Customer Can Use Up To " +
customerQuoteCar.getMileTotal() + " Miles" + " They Must Return The Car In " +
customerQuoteCar.getDuration() + " Hours");
                totalCost = customerQuoteCar.getTotalCost();
                totalTime = customerQuoteCar.getDuration();
                incomeToDate = incomeToDate +
customerQuoteCar.getTotalCost();
                customer.setRented(1); //This sets the record to show the
customer is renting a car.
                customer.setDuration(r);
                    baseCar.setStock(false);
                breakLoop2 = true;
            }
            else if(select == 'h' && baseHelicopter.getStock() == true) {
                c = baseHelicopter.getCost()*d;
                m = baseHelicopter.getMile()*d;
                r = baseHelicopter.getRent()*d;
                Helicopter customerQuoteHelicopter = new
Helicopter(0,0,0,0,true,r,c,m);
                if(d == 1){
                    System.out.println("Total To Pay £" +
customerQuoteHelicopter.getTotalCost() + " They Must Return The Helicopter In " +
customerQuoteHelicopter.getDuration() + " Hour");
                    totalCost = customerQuoteHelicopter.getTotalCost();
                    totalTime = customerQuoteHelicopter.getDuration();
                    incomeToDate = incomeToDate +
customerQuoteHelicopter.getTotalCost();
                    customer.setRented(2); //This sets the record to show the
customer is renting a helicopter.
                    customer.setDuration(r);
                        baseHelicopter.setStock(false);
                    breakLoop2 = true;
                }
            }
        }
    }
}

```

```

    }
    else {
        System.out.println("Total To Pay £" +
customerQuoteHelicopter.getTotalCost() + " They Must Return The Bicycle In " +
customerQuoteHelicopter.getDuration() + " Hours");
        totalCost = customerQuoteHelicopter.getTotalCost();
        totalTime = customerQuoteHelicopter.getDuration();
        incomeToDate = incomeToDate +
customerQuoteHelicopter.getTotalCost();
        customer.setRented(2); //This sets the record to show the
customer is renting a helicopter.
        customer.setDuration(r);
        baseHelicopter.setStock(false);
        breakLoop2 = true;
    }
}
else if(select == 'b' && baseBicycle.getStock() == true) {
    c = baseBicycle.getCost()*d;
    m = baseBicycle.getMile()*d;
    r = baseBicycle.getRent()*d;
    Bicycle customerQuoteBicycle = new
Bicycle(0,0,0,0,true,r,c,m);
    System.out.println("Total To Pay £" +
customerQuoteBicycle.getTotalCost() + " They Must Return The Car In " +
customerQuoteBicycle.getDuration() + " Hours");
    totalCost = customerQuoteBicycle.getTotalCost();
    totalTime = customerQuoteBicycle.getDuration();
    incomeToDate = incomeToDate +
customerQuoteBicycle.getTotalCost();
    customer.setRented(3); //This sets the record to show the
customer is renting a bicycle.
    customer.setDuration(r);
    baseBicycle.setStock(false);
    breakLoop2 = true;
}
}
else {
    System.out.println("Sorry You Have Entered Invalid Data, Please
Try Again!");
    breakLoop2 = false;
} //This section of code calls the current stored date from the date
vector, based on the customer number, and asks for a new rental date to be input.
    DateDay hireDate = (DateDay) date.get(selectNo);
    System.out.println("Please Enter The Month Of Hire");
    hireDate.setMonthHire();
    System.out.println("Please Enter The Day Of Hire");
    hireDate.setDayHire();
    System.out.println("Please Enter The Year Of Hire");
    hireDate.setYearHire();
    System.out.println("Please Enter The Hour Of Hire");

```

```

        hireDate.setHourHire();
        System.out.println("Please Enter The Minute Of Hire");
        hireDate.setMinHire();
        date.add(hireDate); //This adds the new date back into the
vector.
        System.out.println("The Customer Needs To Pay A Total Of:
£" + totalCost);
    }
}
    loopBreak = true;
}
}
    else if(validate == 0) { //If the details are not correct, the loop will break, and
the user will have to try again.
        System.out.println("Please Try Again!");
        loopBreak = true;
    }
    else { //If thee user inputs anything other than 1 or 0, an error message will be
given and the loop will restar.t
        System.out.println("Please Enter 1 For Yes, Or 0 For No");
        loopBreak = false;
    }
}
}
}
    public void returnHire() { //This section of code is used when the customer wishes
to return the vehicle they are hiring.
        int validate;
        int selectNo = -1;
        boolean loop = false; //This again is used in validation.
        int loopBreak = 0;

        while(!loop) { //This section of code asks for the customers number, and if that
customer does not exist, then an error will be given, if they do exist then the code will
continue.
            System.out.println("Please Enter The Customer's Customer Number");
            selectNo = TextIO.getInt();
            if(selectNo >customerCount) {
                System.out.println("i'm Sorry This Customer Does Not Exist, Please Try
Again!");
                loop = false;
            }
            else if(selectNo <0) {
                System.out.println("i'm Sorry This Customer Does Not Exist, Please Try
Again!");
                loop = false;
            }
            else {
                loop = true;
            }
        }
    }
}

```

```
CustomerRecord customer = (CustomerRecord) customerR.get(selectNo); //This
calls the record of the customer, based on the number entered above.
if(customer.getRented() == 4) { //This section of code is called if the customer is
currently renting nothing, and an error is shown.
    System.out.println("This Customer Currently Has Nothing On Hire!");
}
else { //If the customer is renting something, then all their details will be given to
the user.
    System.out.println("Current Rental Details:");
    System.out.println("Customer Name: " + customer.getName());
    System.out.println("Customer Address: " + customer.getAddress());
    System.out.println("Customer Telephone Number: " + customer.getTelNo());
    if(customer.getRented() == 1) { //If the customer is renting a car, information
about the rental will be given.
        System.out.println("The Customer Is Renting A Car, They Rented It For: "
+ customer.getDuration() + " Hours");
    }
    else if(customer.getRented() == 2) { //If the customer is renting a helicopter,
information about the rental will be given.
        System.out.println("The Customer Is Renting A Helicopter, They Rented It
For: " + customer.getDuration() + " Hour(s)");
    }
    else if(customer.getRented() == 3) { //If the customer is renting a bicycle,
information about the rental will be given.
        System.out.println("The Customer Is Renting A Bicycle, They Rented It
For: " + customer.getDuration() + " Hours");
    }
    else { //If for some reason the customer had no rental details, then this error
would be thrown.
        System.out.println("The Customer Is Not Renting Anything! Please Try
Again");
    }
    System.out.println("Is This Correct? If Yes Press 1, If No Press 0");
    validate = TextIO.getInt(); //This section of code is used to confirm if these
details are correct.
    if(validate == 1) { //If the details are correct, then the date will be called from
the date vector, and the user will have to input the return date and time.
        DateDay hireDate = (DateDay) date.get(selectNo);
        System.out.println("Please Enter The Month Of Return");
        hireDate.setMonth();
        System.out.println("Please Enter The Day Of Return");
        hireDate.setDay();
        System.out.println("Please Enter The Year Of Return");
        hireDate.setYear();
        System.out.println("Please Enter The Hour Of Return");
        hireDate.setHour();
        System.out.println("Please Enter The Minute Of Return;");
        hireDate.setMin();
    }
}
```



```

boolean loop = false;

while(!loopBreak) { //This section of code is used to get the customers number,
and if it is invalid throw an error message, and ask the user to input it again.
    System.out.println("Please Enter The Customer's Customer Number");
    selectNo = TextIO.getInt();
    if(selectNo >customerCount) {
        System.out.println("i'm Sorry This Customer Does Not Exist, Please Try
Again!");
        loopBreak = false;
    }
    else if(selectNo <0) {
        System.out.println("i'm Sorry This Customer Does Not Exist, Please Try
Again!");
        loopBreak = false;
    }
    else {
        loopBreak = true;
    }
}

/*The folowing code calls the customers information from the customer vector,
based on their customer number.
* It then displays their information, name etc.*/
CustomerRecord customer = (CustomerRecord) customerR.get(selectNo);
System.out.println("Customer Name: " + customer.getName());
System.out.println("Customer Address: " + customer.getAddress());
System.out.println("Customer Telephone Number: " + customer.getTelNo());

while(!loop) { //This section of code generates a menu to deal with the customers
details.
    System.out.println("");
    System.out.println("Press 1# To Update The Customers Name");
    System.out.println("Press 2# To Update The Customers Address");
    System.out.println("Press 3# To Update The CustomersTelephone Number");
    System.out.println("Press 4# To Return To The Main Menu");
    System.out.println("");
    menu = TextIO.getInt();
    if(menu == 1) { //If the user selects 1, then they are given the option to edit
the customers name, and then returned to the menu.
        System.out.println("Please Input The Customers Updated Name");
        set = TextIO.getLnString();
        customer.setName(TextIO.getLnString()); //This sets the customers new
name, and it is then printed out.
        System.out.println("Customer Updated Name: " + customer.getName());
    }
    else if(menu == 2) { //If the user selects 2, then they will be able to enter a
new address for the customer, then return to the menu.
        System.out.println("Please Input The Customers Updated Address");
        set = TextIO.getLnString();

```

```

        customer.setAddress(TextIO.getLnString()); //This allows the customers
address to be changed, and the new address is then printed out.
        System.out.println("Customer Updated Address: " +
customer.getAddress());
    }
    else if(menu == 3) { //If the user selects 3, they will be able to update the
customers telephone number before being returned to the menu.
        System.out.println("Please Input The Customers Updated Telephone
Number");
        set = TextIO.getLnString();
        customer.setTelNo(TextIO.getLnString()); //This allows the telephone
number to be updted, and the new number is then displayed.
        System.out.println("Customer Updated Telephone Number: " +
customer.getTelNo());
    }
    else if(menu == 4) { //If the user selects 4, they then the loop is broken.
        loop = true;
    }
    else { //If the user inputs anything else then they will receive an error message,
and the loop will restart.
        System.out.println("Please Input A Valid Number!");
        loop = false;
    }
}
}
}
public void incomeToDate() { //This section of code displays the income to date,
for the company.
    System.out.println("The Income To Date Is: £" + incomeToDate);
}
}
}

```

DateDay

//THIS IS A UPDATED, AND MODIFIED VERSION OF DATE CODE FROM
PREVIOUS ASSIGNMENT

//This class is used to calculate the difference between two dates and two times, in
days, hours and mins.

```
public class DateDay{
```

```
    int[] daysInMonth = {-1,31,28,31,30,31,30,31,31,30,31,30,31}; //This is an array,
used to represent the number of days in each month, 1 - 12, it starts at -1, so that the
user can input 1 - 12, rather than 0 - 11.
```

```
    protected int monthHire; //This is used to represent the Hire month the user will
input.
```

```
    protected int dayHire; //This is used to represent the Hire day the user will input.
```

```
    protected int yearHire; //This is used to represent the Hire year the user will input.
```

```
protected int month; //This is used to represent the return month that the user will
input.
protected int day; //This is used to represent return day that the user will input.
protected int year; //This is used to represent return year that the user will input.

protected int hourHire; //This is used to represent the rental's time, hour the user
will input.
protected int minHire; //This is used to represent the rental's time, min the user will
input.

protected int hour; //This is used to represent the return's time, hour the user will
input.
protected int min; //This is used to represent the return's time, min the user will
input.

public DateDay(int m, int d, int y, int n, int b, int c,int hr, int mr, int ho, int mi){
//public Date is used to get input from the user, and to set both dates to the correct
values they enter.
    monthHire = m; //This defines monthHire as = to m.
    dayHire = d; //This defines dayHire as = to d.
    yearHire = y; //This defines yearHire as = to y.

    month = n; //This defines month as = to n.
    day = b; //This defines day as = to b.
    year = c; //This defines year as = to c.

    hourHire = hr; //This defines hourHire as = to hr.
    minHire = mr; //This defines minHire as = to mr.

    hour = ho; //This defines hour as = to ho.
    min = mi; //This defines min as = to mi.
}
public void setMonthHire(){ //This section of code is used to get input into
monthHire, and to validate it. Also to return error messages where needed.

    boolean monthHireNull = false; //For each validation method, I have created a
while loop, which will loop continually while ever my boolean is set to false, however
as soon as it is set to true, the loop will break, and the next part of code will activate.
    while(!monthHireNull){
        monthHire = TextIO.getInt();//This section of code is used to get input from
the user and store it into monthHire.
        if(monthHire >12){//This section of code states that if the month is greater
than 12, then the loop should restart as it is invalid.
            System.out.println("Im Sorry This Month Is Invalid (To High), Please Try
Again!");
            monthHireNull = false;
        }
        else if(monthHire <1){//This section of code states that if the month is less
than 1, the loop should restart as the code is invalid.
```

```
        System.out.println("Im Sorry This Month Is Invalid (To Low), Please Try
Again!");
        monthHireNull = false;
    }
    else{//This section of code states that if the above are not the case, then the
date is valid, and the loop should break.
        monthHireNull = true;
    }
}
}
}
public void setDayHire(){//This section of code is used to get input into dayHire,
and to validate it. Also to return error messages where needed.

    boolean dayHireNull = false;

    while(!dayHireNull){
        dayHire = TextIO.getInt();//This section of code gets input from the user and
stores it in dayHire
        if(monthHire == 1 || monthHire == 3 || monthHire == 5 || monthHire == 7 ||
monthHire == 8 || monthHire == 10 || monthHire == 12){//This section of code takes
all the months with 31 days in, and gives them validation.
            if(dayHire >31){//This states that if a month has 31 days, and a value higher
than this is entered then it will not be accepted.
                System.out.println("Im Sorry This Month Is Invalid (To High), Please
Try Again!");
                dayHireNull = false;
            }
            else if(dayHire <1){//This states that if the value is less than 1, it is invalid.
                System.out.println("Im Sorry This Month Is Invalid (To Low), Please Try
Again!");
                dayHireNull = false;
            }
            else{//This states that if the data is between 1 and 31 then it is valid and the
loop shall be broken.
                dayHireNull = true;
            }
        }
        else if(monthHire == 2){//This states that if the month is not one of the above,
and if it is 2 that it will go into validation for the 2nd month.
            if(dayHire >28){//This states that if the user enters a value greater than 28
then it will not be accepted.
                System.out.println("Im Sorry This Month Is Invalid (To High), Please
Try Again!");
                dayHireNull = false;
            }
            else if(dayHire <1){//This states that if the user enters a value less than 1, it
will not be accepted.
                System.out.println("Im Sorry This Month Is Invalid (To Low), Please Try
Again!");
                dayHireNull = false;
            }
        }
    }
}
```

```
    }
    else{//If the data entered is valid then the loop will be broken.
        dayHireNull = true;
    }
}
else if(monthHire == 4 || monthHire == 6 || monthHire == 9 || monthHire ==
11){//This is for the months with 30 days, if none of the above is true this will
activate.
    if(dayHire >30){//This states that if the user enters a value more than 30,
then it is not valid.
        System.out.println("Im Sorry This Month Is Invalid (To High), Please
Try Again!");
        dayHireNull = false;
    }
    else if(dayHire <1){//This states that if the user enters a value less than 1
then it is not valid.
        System.out.println("Im Sorry This Month Is Invalid (To Low), Please Try
Again!");
        dayHireNull = false;
    }
    else{//If the user enters data that is valid then the loop will be broken.
        dayHireNull = true;
    }
}
}
}
}
}
public void setYearHire(){//This section of code is used to get input from the user
and store it in yearHire. Also to return error messages where needed.
    boolean yearHireNull = false;
    while(!yearHireNull){
        yearHire = TextIO.getInt();
        if(yearHire >9999){//This states that if the user enters a value over 9999 that it
is not valid.
            System.out.println("Im Sorry This Year Is Invalid (To High), Please Try
Again!");
            yearHireNull = false;
        }
        else if(yearHire <1){//This states that if the user enters a value of less than 1,
that it is not valid.
            System.out.println("Im Sorry This Year Is Invalid (To Low), Please Try
Again!");
            yearHireNull = false;
        }
        else{//If the user enters valid data then the loop will be broken.
            yearHireNull = true;
        }
    }
}
}
}
public void setHourHire() { //This section of code is used to get input from the user
and store it in hourHire. Also to return error messages where needed.
```

```
boolean hourHireNull = false; //For each validation method, I have created a
while loop, which will loop continually while ever my boolean is set to false, however
as soon as it is set to true, the loop will break, and the next part of code will activate.
while(!hourHireNull){
    hourHire = TextIO.getInt();//This section of code is used to get input from the
user and store it into monthHire.
    if(hourHire >24){//This section of code states that if the hour is greater than
24, then the loop should restart as it is invalid.
        System.out.println("Im Sorry This Hour Is Invalid (To High), Please Try
Again!");
        hourHireNull = false;
    }
    else if(hourHire <1){//This section of code states that if the hour is less than 1,
the loop should restart as the code is invalid.
        System.out.println("Im Sorry This Hour Is Invalid (To Low), Please Try
Again!");
        hourHireNull = false;
    }
    else{//This section of code states that if the above are not the case, then the
time is valid, and the loop should break.
        hourHireNull = true;
    }
}
}
}
}
public void setMinHire() { //This section of code is used to get input from the user
and store it in minHire. Also to return error messages where needed.
    boolean minHireNull = false; //For each validation method, I have created a
while loop, which will loop continually while ever my boolean is set to false, however
as soon as it is set to true, the loop will break, and the next part of code will activate.
while(!minHireNull){
    minHire = TextIO.getInt();//This section of code is used to get input from the
user and store it into minHire.
    if(minHire >60){//This section of code states that if the min is greater than 60,
then the loop should restart as it is invalid.
        System.out.println("I'm Sorry This Minute Is Invalid (To High), Please Try
Again!");
        minHireNull = false;
    }
    else if(minHire <0){//This section of code states that if the min is less than 0,
the loop should restart as the code is invalid.
        System.out.println("I'm Sorry This Minute Is Invalid (To Low), Please Try
Again!");
        minHireNull = false;
    }
    else{//This section of code states that if the above are not the case, then the
time is valid, and the loop should break.
        minHireNull = true;
    }
}
}
}
```

```
public void setMonth(){//This section of code is used to get input from the user and
store it in month. Also to return error messages where needed.
    boolean monthNull = false;
    while(!monthNull){
        month = TextIO.getInt();
        if(month >12){//This states that if the month entered is over 12, it is invalid.
            System.out.println("I'm Sorry This Month Is Invalid (To High), Please Try
Again!");
            monthNull = false;
        }
        else if(month <1){//This states that if the month entered is less than 1, it is
invalid.
            System.out.println("I'm Sorry This Month Is Invalid (To Low), Please Try
Again!");
            monthNull = false;
        }
        else{//If the data is valid then the loop will be broken.
            monthNull = true;
        }
    }
}

public void setDay(){//This section of code is used to get input from the user and
store it in day. Also to return error messages where needed.
    boolean dayNull = false;
    while(!dayNull){
        day = TextIO.getInt();
        if(month == 1 || month == 3 || month == 5 || month == 7 || month == 8 || month
== 10 || month == 12){//This section of code looks at all months with 31 days.
            if(month >31){//If the user inputs over 31 then it is an invalid entry.
                System.out.println("I'm Sorry This Month Is Invalid (To High), Please
Try Again!");
                dayNull = false;
            }
            else if(day <1){//If the user inputs less than 1 then it is an invalid entry.
                System.out.println("I'm Sorry This Month Is Invalid (To Low), Please
Try Again!");
                dayNull = false;
            }
            else{//If the user inputs between 1 and 31 then the entry is valid and the
loop will break.
                dayNull = true;
            }
        }
        else if(month == 2){//If the month is not one of the above, and is the 2'nd then
this will validate month 2.
            if(day >28){//If the user enters a day over 28, then it is invalid.
                System.out.println("I'm Sorry This Month Is Invalid (To High), Please
Try Again!");
                dayNull = false;
            }
        }
    }
}
```

```

        else if(day <1){//If the user enters a day less than 1, then it is invalid.
            System.out.println("I'm Sorry This Month Is Invalid (To Low), Please
Try Again!");
            dayNull = false;
        }
        else{//If the user enters valid data then the loop will break.
            dayNull = true;
        }
    }
    else if(month == 4 || month == 6 || month == 9 || month == 11){//This section
of code is used if the month is not one of the above.
        if(day >30){//If the user inputs a day over 30, it is invalid.
            System.out.println("I'm Sorry This Month Is Invalid (To High), Please
Try Again!");
            dayNull = false;
        }
        else if(day <1){//If the user inputs a day under 1, then it is invalid.
            System.out.println("Im Sorry This Month Is Invalid (To Low), Please Try
Again!");
            dayNull = false;
        }
        else{//Else if the user inputs valid data the loop will break.
            dayNull = true;
        }
    }
}

public void setYear(){//This section of code is used to get input from the user and
store it in year. Also to return error messages where needed.
    boolean yearNull = false;
    while(!yearNull){
        year = TextIO.getInt();
        if(year >9999){//If the user inputs a year over 9999,it is invalid.
            System.out.println("I'm Sorry This Year Is Invalid (To High), Please Try
Again!");
            yearNull = false;
        }
        else if(year <1){//If the user inputs a year under 1, it is invalid.
            System.out.println("I'm Sorry This Year Is Invalid (To Low), Please Try
Again!");
            yearNull = false;
        }
        else if(year <yearHire) { //If the user inputs a year that is lower than the year
of hire an error will be displayed.
            System.out.println("I'm Sorry It Is Not Valid To Enter A Return Date
Before The Hire Date, Please Try Again!");
            yearNull = false;
        }
        else{//If the user inputs valid data the loop will break.
            yearNull = true;
        }
    }
}

```

```
    }  
  }  
}  
public void setHour() { //This section of code is used to get input from the user and  
store it in hour. Also to return error messages where needed.  
    boolean hourNull = false; //For each validation method, I have created a while  
loop, which will loop continually while ever my boolean is set to false, however as  
soon as it is set to true, the loop will break, and the next part of code will activate.  
    while(!hourNull){  
        hour = TextIO.getInt();//This section of code is used to get input from the user  
and store it into hour.  
        if(hour >24){//This section of code states that if the hour is greater than 24,  
then the loop should restart as it is invalid.  
            System.out.println("I'm Sorry This Hour Is Invalid (To High), Please Try  
Again!");  
            hourNull = false;  
        }  
        else if(hour <1){//This section of code states that if the hour is less than 1, the  
loop should restart as the code is invalid.  
            System.out.println("I'm Sorry This Hour Is Invalid (To Low), Please Try  
Again!");  
            hourNull = false;  
        }  
        else{//This section of code states that if the above are not the case, then the  
date is valid, and the loop should break.  
            hourNull = true;  
        }  
    }  
}  
public void setMin() { //This section of code is used to get input from the user and  
store it in min. Also to return error messages where needed.  
    boolean minNull = false; //For each validation method, I have created a while  
loop, which will loop continually while ever my boolean is set to false, however as  
soon as it is set to true, the loop will break, and the next part of code will activate.  
    while(!minNull){  
        min = TextIO.getInt();//This section of code is used to get input from the user  
and store it into min.  
        if(min >60){//This section of code states that if the min is greater than 60, then  
the loop should restart as it is invalid.  
            System.out.println("I'm Sorry This Minute Is Invalid (To High), Please Try  
Again!");  
            minNull = false;  
        }  
        else if(min <0){//This section of code states that if the min is less than 0, the  
loop should restart as the code is invalid.  
            System.out.println("I'm Sorry This Minute Is Invalid (To Low), Please Try  
Again!");  
            minNull = false;  
        }  
    }  
}
```

```

        else{//This section of code states that if the above are not the case, then the
date is valid, and the loop should break.

```

```

            minNull = true;

```

```

        }

```

```

    }

```

```

    public void math(){//This section of the code is used to calculate the number of
days between date 1, and date 2.

```

```

        int monthTotal = 0; //This int is used to represent monthTotal for the Hire/Start
date.

```

```

        int monthTotalFinal = 0; //This int is used to represent the monthTotal for the
End/Today's date.

```

```

        int dayTotal = 0; //This is used to represent the total number of days for the
end/today's date.

```

```

        int yearTotal = daysInMonth[1] + daysInMonth[2] + daysInMonth[3] +
daysInMonth[4] + daysInMonth[5] + daysInMonth[6] + daysInMonth[7] +
daysInMonth[8] + daysInMonth[9] + daysInMonth[10] + daysInMonth[11] +
daysInMonth[12]; //This int is used to calculate the total number of days in a year.

```

```

        int yearHireCalc = 0; //This int will be used to store the total number of days
over a set number of years for the Hire/Start Date.

```

```

        int yearCalc = 0; //This is used to store the total number of days over a set
number of years for the End/Today's Date.

```

```

        int hireTotal = 0; //This int is used to hold the total number of days for Hire/start
date.

```

```

        int leapYearTarget = yearHire; //This is used in a for loop to set leapYearTarget
as = to yearHire.

```

```

        int leapYearTarget2 = year; //This is used in a for loop to set leapYearTarget2
as = to year.

```

```

        int hourFinal = 0; //This is used to calculate the total number of hours.

```

```

        int minFinal = 0; //This is used to calculate the total number of mins.

```

```

        int hourHireTotal = hourHire; //This is used in calculating the hours.

```

```

        int minHireTotal = minHire; //This is used in calculating the mins.

```

```

        int hourTotal = hour; //This is used in calculating the hours.

```

```

        int minTotal = min; //This is used in the calculation of mins.

```

```

        for(int counter=leapYearTarget;counter<leapYearTarget2;counter++){//This for
loop will loop while ever counter is less than leapYearTarget2.

```

```

            leapYearTarget++; //This is used to add one year to the date each time, so that
eventually leapYearTarget and leapYearTarget2 will be of the same value.

```

```

            if(leapYearTarget % 4 == 0 && leapYearTarget % 100 != 0 ){ //This section
of code is used to calculate if the year is a leap year, to do this is first checks that the
year is divisible by 4, and then checks that it is not divisible by 100.

```

```

                dayTotal++; //If the above is true one day is added to dayTotal.

```

```

            }

```

```

            else if(leapYearTarget % 400 == 0){//This finds out if the year is divisible by
400, and if yes adds one to the dayTotal.

```

```

        dayTotal++;
    }
}

```

/*The following for loops works out how many days there are between the hire date, and the month the user enters.

*for example if the user enters 1, and 12, all the days in months 1 - 11 will be added. But not the days from month 12, as this has not yet passed.

```

*/
for(int counter=1;counter<monthHire;counter++) { //This is used to work out the
days in monthTotal.
    monthTotal = monthTotal+daysInMonth[counter];
}
for(int counter=1;counter<month;counter++) { //This is used to work out the
days in monthTotalFinal.
    monthTotalFinal = monthTotalFinal+daysInMonth[counter];
}

```

//The following code calculates how many days total thee are for yearHire. It calculates the years by yearHire*yearTotal so that it can work out the number of days total, and these can then be taken away from the following section to return the correct value it does the same with year/month and day.

yearHireCalc = yearHire*yearTotal;//This returns the value of days between year 0 and the year entered.

hireTotal = yearHireCalc+monthTotal+dayHire;//This returns the total number of days including the above.

yearCalc = year*yearTotal; //This returns the value of days between year 0 and the year entered.

dayTotal = dayTotal+day+monthTotalFinal+yearCalc-hireTotal; //This calculates the total number of days.

```
System.out.println("Total Is: " + dayTotal);
```

```
hourFinal = hourTotal - hourHireTotal; //This calculates the total hours.
```

```
minFinal = minTotal - minHireTotal; //This calculates the total mins.
```

if(hourFinal <0 && minFinal <0) { //The following is validation code, it will validate what happens at the beginging and end of days.

```
    dayTotal--;
```

```
    hourFinal = 24 + hourFinal;
```

```
}
```

else if(hourFinal <1 && minFinal == 60) { //This works out what happens if hour is less than 1, but mins are == to 60.

```
    hourFinal = hourFinal + 1;
```

```
    minFinal = 0;
```

```
}
```

else if(hourFinal == 24) { //If hours are 24 then day = day + 1 and hour = 0,

```
    day++;
```

```
    hourFinal = 0;
```

```
}
```

```
else if(hourTotal <1 && minFinal >0) {
```

```

    day++;
    hourFinal = 0;
}
if(minFinal <0) { //This works out what happens if mins = 0.
    hourFinal--;
    minFinal = 60 + minFinal;
}
if(dayTotal <0) { //This validates what happen if the day total is less than 0. It
makes the readout positive, instead of negative.
    dayTotal = dayTotal * -1;
    dayTotal--;
}
System.out.println("There Are " + dayTotal + " Day(s) " + hourFinal + " Hour(s) " +
minFinal + " Minute(s) Between Those Dates/Times"); //This section of code shows
the user how many days, hours and mins there are between the 2 dates.
    dayTotal = dayTotal * 24; //This section of code is used to calculate the total
number of hours, by taking the number of days and multiplying them by 24.
    hourFinal = hourFinal + dayTotal; //This section of code calculates the final
amount of hours between the two dates and times entered, by adding hoursFinal to
dayTotal.
    System.out.println(hourFinal + " Total"); //This prints out the total number of
hours.
}
public void output() { //This section of code prints out the dates, and times the user
has entered.
    System.out.print("The Customer Rented The Vehicle On: ");
    System.out.println(dayHire+ "/" + monthHire+ "/" + yearHire);
    System.out.print("At: ");
    System.out.println(hourHire + ":" + minHire);
    System.out.print("The Customer Has Returned The Vehicle On: ");
    System.out.println(day + "/" + month + "/" + year);
    System.out.print("At: ");
    System.out.println(hour + ":" + min);
}
}
}

```

Vehicle

/*This class is the parent class for the vehicles car, bike and helicopter, it sets several methods.*/

```

public class Vehicle {

    protected int cost; //This int is used in getCost.
    protected int rent; //This int is used in getRent.
    protected int mile; //This int is used in getMile.
    protected int type; //This int is used in getType.
    protected boolean inStock; //This boolean is used in getStock and setStock.
    protected int duration; //This int is used in getDuration.
    protected int totalCost; //This int is used in getTotalCost.

```

```
protected int mileTotal; //This int is used in getMileTotal.

public Vehicle(int c, int r, int m, int t, boolean i, int d, int tc, int mm) {
    cost = c; //This defines cost as c.
    rent = r; //This defines rent as r.
    mile = m; //This defines mile as m.
    type = t; //This defines type as t.
    inStock = i; //This defines inStock as i.
    duration = d; //This defines duration as d.
    totalCost = tc; //This defines totalCost as tc.
    mileTotal = mm; //This defines mileTotal as mm.
}
public int getCost() { //This will get the cost of a vehicle.
    return this.cost;
}
public int getRent() { //This will get the amount of time for one vehicle rental
period.
    return this.rent;
}
public int getMile() { //This will return the max allowed miles per rental period,
where applicable.
    return this.mile;
}
public int getType() { //This is used in defining what type of vehicle a vehicle is.
    return this.type;
}
public boolean getStock() { //This is used to check whether a specific vehicle is in
stock.
    return this.inStock;
}
public void setStock(boolean inStock) { //This is used to update the stock
information of a vehicle, true for instock, false for out of stock.
    this.inStock = inStock;
}
public int getDuration() { //This is used to get the total duration a vehicle is being
rented for.
    return this.duration;
}
public int getTotalCost() { //This is used to get the total cost of a rental.
    return this.totalCost;
}
public int getMileTotal() { //This is used to get the total allowed miles, where
aplicable.
    return this.mileTotal;
}
}
```

Bicycle

/*This class inherits properties from the parent class vehicle, and sets its own values where needed.*/

```
public class Bicycle extends Vehicle {  
  
    public Bicycle(int cost, int rent, int mile, int type, boolean inStock, int duration, int totalCost, int mileTotal) {  
        super(cost, rent, mile, type, inStock, duration, totalCost, mileTotal);  
    }  
    public int getCost() { //This sets the price per rental period to 25.  
        return (25);  
    }  
    public int getRent() { //This sets the maximum rental time per rental period to 12.  
        return (12);  
    }  
    public int getMile() { //This is not applicable to bicycles, so is 0.  
        return (0);  
    }  
    public int getType() { //This sets teh type of vehicle, 3 = bicycle  
        return 3;  
    }  
}
```

Helicopter

/*This class extends the parent class vehicle, inheriting some of its properties, and setting its own where needed.*/

```
public class Helicopter extends Vehicle {  
  
    public Helicopter(int cost, int rent, int mile, int type, boolean inStock, int duration, int totalCost, int mileTotal) {  
        super(cost, rent, mile, type, inStock, duration, totalCost, mileTotal);  
    }  
    public int getCost() { //This sets the cost per rental period to 500.  
        return (500);  
    }  
    public int getRent() { //This sets the maximum time per rental period to 1.  
        return (1);  
    }  
    public int getMile() { //This vehicle is not bound by a mile limit, so is set to 0.  
        return (0);  
    }  
    public int getType() { //This sets the type of vehicle, 2 = helicopter.  
        return 2;  
    }  
}
```

Car

/*This class extends the vehicle class, inheriting properties from it, and defining its own values where appropriate.*/

```
public class Car extends Vehicle {  
  
    public Car(int cost, int rent, int mile, int type, boolean inStock, int duration, int  
totalCost, int mileTotal) {  
        super(cost, rent, mile, type, inStock, duration, totalCost, mileTotal);  
    }  
    public int getCost() { //This sets the cost of one rental period to 250.  
        return (250);  
    }  
    public int getRent() { //This sets the max time per rental period to 168.  
        return (168);  
    }  
    public int getMile() { //This sets the max number of miles per rental period to 1000.  
        return (1000);  
    }  
    public int getType() { //This defines what type of vehicle this is.  
        return 1;  
    }  
}
```

Customer Record

/*This Class Is Used To Manage Customer Records, It Enables The User To Set, And Get The Customers Address, Name, Telephone Number, As Well As Showing What They Are Renting, And How Long For.*/

```
public class CustomerRecord {  
    protected String name; //This String Is Used In getName And setName.  
    protected String address; //This String Is Used In getAddress And setAddress.  
    protected String telNo; //This String Is Used In getTelNo and setTelNo.  
    protected int rented; //This Int Is Used In getRented and setRented.  
    protected int duration; //This Int Is Used In setDuration and getDuration.  
  
    public CustomerRecord(String n,String a, String t, int r, int d) {  
        name = n; //This Defines String name As n.  
        address = a; //This Defines String address As a.  
        telNo = t; //This Deifines String telNo As t.  
        rented = r; //This Defines int rented As r.  
        duration = d; //This Defines int duration As d.  
    }  
    public String getName() { //This returns the customers name.  
        return this.name;  
    }  
    public void setName(String name) { //This lets the customers name be edited.  
        this.name = name;  
    }  
}
```

```
public String getAddress() { //This returns the customers address.
    return this.address;
}
public void setAddress(String address) { //This lets the customers address be
edited.
    this.address = address;
}
public String getTelNo() { //This returns the customers telephone number.
    return this.telNo;
}
public void setTelNo(String telNo) { //This allows the customers telephone number
to be edited.
    this.telNo = telNo;
}
public int getRented() { //This will show what the customer is currently renting.
    return this.rented;
}
public void setRented(int rented) { //This enables the program to update what hte
customer is renting.
    this.rented = rented;
}
public int getDuration() { //This shows how long the customer is renting their
current rental.
    return this.duration;
}
public void setDuration(int duration) { //This will be updated to show how long
they are renting a vehicle for.
    this.duration = duration;
}
}
```

Task 4 – Testing And Evaluation

Testing Note:

If I have used code more than once, or code that is very similar, such as my validation for/while loops I will not test them in each class, to save repeating the same tests over again in my work.

Vectors

I have created to vectors in my hire class, one to store customer records and one to store dates, I have tested that they work be creating items, storing them onto the vectors, and then getting them off, and testing that the information is correct. I have also modified information, and it has been correctly stored back onto the vectors.

Validation

I have written validation rules for user input, so that users cannot input inappropriate data, for example when selecting an option, they cannot enter an option that is not available, instead they will get an error message?

I have tested each validation method, and made sure that they all work, therefore I am relatively sure that the system is secure for users, and that they should not experience to many problems with input.

Hire

This class contains several methods, all of which are called in my main class, using main I have tested each method.

customerDetail()

To test this class, I input the required data several times:

Please Input The Customers Name

Entering - Bob

The Name You Entered Is: Bob

Is This Correct? Enter 1 For Yes Or 0 For No!

Pressing 0:

Please Enter The Name Again!

Please Input The Customers Name

Entering a new name Bobby

The Name You Entered Is: Bobby

Is This Correct? Enter 1 For Yes Or 0 For No!

Pressing 1:

Thank You!

Please Input The Customers Address

I then repeated this process for the rest of the code, entering several different names, addresses, and telephone numbers, Each time it correctly showed me what I had just entered.

When it is asking if it is the correct data, entering anything other than 1 or 0 returns an error message:

Entering 3:

Im Sorry The Letter/Number You Entered Is Not Valid, Please Try Again!

Entering 400:

Im Sorry The Letter/Number You Entered Is Not Valid, Please Try Again!

Entering – 1.233444444

**** Error in input: Illegal integer input.*

**** Expecting: Integer in the range -2147483648 to 2147483647*

**** Discarding Input:*

Please re-enter:

Entering – Test

**** Error in input: Illegal integer input.*

**** Expecting: Integer in the range -2147483648 to 2147483647*

**** Discarding Input: Test*

Please re-enter:

Entering – T

**** Error in input: Illegal integer input.*

**** Expecting: Integer in the range -2147483648 to 2147483647*

**** Discarding Input: T*

Evaluation:

This method works as intended, it does not allow the user to input invalid data, and if they enter incorrect data gives them the ability to change it. This method is easy to use, and relatively fool proof.

editCustomer()

My main class will not let this be run until at least one customers details has been put into the system. This function allows you to edit a customers details:

I am asked for the customers number, if I input an invalid number I get the following message:

i'm Sorry This Customer Does Not Exist, Please Try Again!

Please Enter The Customer's Customer Number

Entering a valid number brings up that customers details

Customer Name: 1

Customer Address: 1

Customer Telephone Number: 1

I can then choose to edit their name, address, or telephone number, or return to the main menu.

I tested this class by entering several different combinations of names, addresses etc, all tests were successful, here is the outcome of a test.

? 1

Please Input The Customers Updated Name

This Is

Customer Updated Name: This Is

Press 1# To Update The Customers Name

Press 2# To Update The Customers Address

Press 3# To Update The CustomersTelephone Number

Press 4# To Return To The Main Menu

2

Please Input The Customers Updated Address

A

Customer Updated Address: A

Press 1# To Update The Customers Name

Press 2# To Update The Customers Address

Press 3# To Update The CustomersTelephone Number

Press 4# To Return To The Main Menu

3

Please Input The Customers Updated Telephone Number

Test

Customer Updated Telephone Number: Test

Press 1# To Update The Customers Name

Press 2# To Update The Customers Address

Press 3# To Update The Customers Telephone Number

Press 4# To Return To The Main Menu

Evaluation

Again this class is relatively fool proof, the user cannot enter invalid data, and if they enter an incorrect detail they will be able to see and amend it easily.

existingHire()

This class again can only be used if there is already a customer in the system, it takes a customer number, and then continues the code when valid, I repeated the testing that I have done before for this section of the code until I was satisfied that it would not let me enter an invalid number.

Customer Name: This Is

Customer Address: A

Customer Telephone Number: Test

Are These Details Correct? Enter 1 For Yes, Or 0 For No

If the customer is already renting a vehicle then the system does not let them set up a new rental:

Sorry But A Customer Can Only Rent One Vehicle At A Time

If they are not then I can set up a new rental:

Please Select The Type Of Vehicle, C For Car, H For Helicopter, B For Bicycle

Entering anything other than C, H or B returns an error:

Sorry You Have Entered Invalid Data, Please Try Again!

Please Enter How Long The Customer Wishes To Rent A Vehicle For, This Can Only Be In Multiples Of The Time We Offer, E.G 2x On A Bicycle = 24 Hours. Lowest Multiple 1, Highest 10

I tried inputting several different numbers in this class, and again it would not accept anything that was invalid.

I then have to enter information about the date, and time the vehicle is rented, I am unable to enter any invalid month, day, year, or time, due to the validation rules I have written, if I do enter something invalid I get an error message thrown at me.

Once the hire has been set up I get the message:

The Customer Needs To Pay A Total Of: £500

The amount alters correctly dependant on the amount of rental multiples I enter, in this case I entered 2.

Evaluation:

After entering many different types of data, I am happy with this method, it will not let a user break it, which makes it relatively safe for use, and it is self explanatory in use.

hireDuration()

This class is used to calculate the total duration of rental a customer wishes to have, they can enter a multiple, of the minimum rental time, e.g 2, and it will display how long they have it for.

I have entered many multiples into this, and all times it either gives me a validation error (If inappropriate data) or stores the correct duration:

If Total Time Is Greater Than The Allowed Rental Time Of: 336 Hours And The Mileage Is Greater Than The Allowed Miles Issue Late Return Fine

I again entered many different values, negative numbers e.g. -23344, positive e.g. 256, fractions e.g. 2.334454545, characters etc, until I was satisfied it was working correctly

Evaluation

This class works well, allowing the user to rent a vehicle for longer than the set duration if they wish, but not forcing it, it does not allow the user to input inappropriate data, so should not break the overall program.

incomeToDate()

This is an extremely simple method with only one line of code, it simply prints out the income to date integer, I tested that the sum correctly works by renting different vehicles:

Renting a car for 250

The Income To Date Is: £250

Renting a bicycle for 2 rental periods

The Income To Date Is: £300

I continued adding vehicle rentals until I was satisfied that this was working correctly.

Evaluation

This method requires no user input, so cannot be broken easily, it works well, and shows the desired output.

returnHire()

This method is used to return a rented vehicle.

As tested below, I set up, and returned several rentals to ensure that the stock system was working correctly, setting a vehicle setStock() to false when rented, and true when returned, and found this function to work well.

I also tested that the date calculation function was working correctly, by entering several different customers at any one time, and calling a specific one, and I found that the calculation from hire date and return date were correct.

I also updated customers, with a second rental and this still worked as expected, giving the number of days, hours and minutes to the user.

It will also display to the user what vehicle has been rented, and how long for, after testing, I managed to get this working correctly, so that it would display for each type of vehicle and update that specific details status.

After all input has been entered you get the following display:

There Are 1 Day(s) 0 Hour(s) 0 Minute(s) Between Those Dates/Times

24 Total

If Total Time Is Greater Than The Allowed Rental Time Of: 168 Hours And The Mileage Is Greater Than The Allowed Miles Issue Late Return Fine

Dependant on the dates, and other values you have entered during the course of the program.

Evaluation

This works as expected, it does not allow the user to input inappropriate data, and correctly returns the vehicle to in stock as well as updating the customers details.

stock()

This method displays what vehicles are currently in stock, when you first run the system:

The Car Is In Stock

The Helicopter Is In Stock

The Bicycle Is In Stock

After renting 2 vehicles:

Sorry But The Car Is Not In Stock

Sorry But The Helicopter Is Not In Stock

The Bicycle Is In Stock

After returning a vehicle:

The Car Is In Stock

Sorry But The Helicopter Is Not In Stock

The Bicycle Is In Stock

I continued setting up rentals, and returning them, until I was satisfied the system was working as intended.

Evaluation

I am happy with this system, it enables the user to see what is currently in stock, by checking with the vehicle objects.

vehicleType()

I tested this by selecting the type of vehicle the customer is renting, and then looking at that customers details, as this writes the details to a variable and then stores them in the CustomerRecord object, in my vector.

Very quickly I could see that this was working correctly, as there were only the options of car, helicopter and bicycle, so testing was relatively easy for this method.

Inputting c, gave this message when checking the customers details for example:

The Customer Is Renting A Car, They Rented It For: 168 Hours

Evaluation

This method enables me to store what type of vehicle a customer is renting, and thus my program to work correctly, for example if a customer rented a car, but claimed they were renting a bike, the system would prove otherwise.

Vehicle – Car – Helicopter - Bicycle

These classes correctly use inheritance, and overriding, for example, the cost to rent a car is 250, and this will always be true. These classes work correctly, and I have written very basic unit tests to ensure that the overriding works correctly. I have also tested “setStock()” and my program correctly alters the value from true to false, when the vehicle has been rented, and to true again when it is returned automatically without any user input.

Car

Testing Code:

```
//This is a testing class for Car.
```

```
public class CarTest extends junit.framework.TestCase
{
    //Sets up a new Car object.
    Car test = new Car(0,0,0,0,true,0,0,0);
    public CarTest()
    {
    }
    protected void setUp()
    {
    }
    protected void tearDown()
    {
    }
    //Is test.
    public void testCarData() {
        //Checks that values are correct.
        assertTrue(test.getCost() == 250);
        assertTrue(test.getRent() == 168);
        assertTrue(test.getMile() == 1000);
        assertTrue(test.getType() == 1);
    }
}
```

Helicopter

Testing Code:

```
//This is a testing class for Helicopter.
```

```
public class HelicopterTest extends junit.framework.TestCase
{
```

```
//Sets up new Helicopter object.
Helicopter test = new Helicopter(0,0,0,0,true,0,0,0);
public HelicopterTest()
{
}
protected void setUp()
{
}
protected void tearDown()
{
}
//Is test.
public void testHelicopterData() {
    //Checks that values are correct.
    assertTrue(test.getCost() == 500);
    assertTrue(test.getRent() == 1);
    assertTrue(test.getMile() == 0);
    assertTrue(test.getType() == 2);
}
}
```

Bicycle

Testing Code:

```
public class BicycleTest extends junit.framework.TestCase
{
    //Sets up a new Bicycle object.
    Bicycle test = new Bicycle(0,0,0,0,true,0,0,0);
    public BicycleTest()
    {
    }
    protected void setUp()
    {
    }
    protected void tearDown()
    {
    }
    //Is test.
    public void testBicycleData() {
        //Checks that values are correct.
        assertTrue(test.getCost() == 25);
        assertTrue(test.getRent() == 12);
        assertTrue(test.getMile() == 0);
        assertTrue(test.getType() == 3);
    }
}
```

DateDay

This code is a modified version of the code submitted for the last assignment, I have tidied it up removing several large If statements and converting them to for loops, I have also added the ability for the code to calculate the number of hours, and minutes between two dates and times.

I added several validation rules to the hour and minute classes, meaning that they cannot be anything other than valid, a user cannot enter an hour less than 1, or more than 24, nor can they enter a minute less than 1 or over 60.

I have also made it so that the math function correctly validates so that it can handle the end and start of days and hours, and display the correct amount of days/hours, I have tested this by entering dozens of combinations of dates, and checking that the calculations were correct, to start with I found that they were not however after much editing I managed to get the code functioning to a level I am happy with.

Evaluation

The method itself is easy to use, allowing users to input start and end dates, and giving them the correct feedback of days and hours between two dates.

CustomerRecord

After thorough testing of my code I found the CustomerRecord class to work correctly, it allows me to create new CustomerRecord objects, and set the values of each “get” method, I can also update Name, Address, Telephone Number, Rented and Duration by using my “public void set” methods.

Evaluation

This class works as intended, functioning well with the rest of the program, and allowing customer details to be managed.

Testing Code:

```
//Testing class for CustomerRecord.
```

```
import java.util.Vector; //This imports the vecotor utility.
```

```
public class CustomerRecordTest extends junit.framework.TestCase
{
    //Creates new vector.
    Vector testV = new Vector();
    public CustomerRecordTest() {
    }
    protected void setUp() {
    }
    protected void tearDown()
```

```
{
}
//Is test.
public void testCreateVectorObjects() {
    //Sets up Strings.
    String a = "The Test Name";
    String b = "The Test Address";
    String c = "The Test Telephone Number";
    //Creates new CustomerRecord object.
    CustomerRecord test = new CustomerRecord(a,b,c,0,0);
    //Loop to insert objects into vector.
    for(int counter = 0;counter<10;counter++) {
        testV.add(test);
    }
    //Loop to get objects from vector, and test they are not null.
    for(int counter = 0;counter<10;counter++) {
        assertNotNull(testV.get(counter));
    }
}
}
```

Evaluation Of Program

My program started very simple, allowing only one customer at a time, however I then expanded this using more advanced functions such as inheritance, and vectors making the system more useful in the real world.

From intensive testing I have found my program to be relatively fool proof, generally not accepting any data that is invalid.

I realise that mixing I/O into my hire class was a mistake, as this would have been much better in a class of its own, and would of allowed easier modification of the program in future, if for example I wanted to add a GUI.

Sadly due to time constraints I have not been able to alter my code, so have attempted to make it as safe as possible for the user, and implement as much testing as possible.

I have entered several basic unit tests, to demonstrate the more advanced testing functions available, however could not implement as much as I would of liked due to the way I have written my program.

Conclusion

I have learnt a great deal from this project, and if I were to have had more time, or if I were to undertake this project again I would separate my I/O from my methods, and implement more advanced testing methods. However this having been said I have created a working program, and tested it under several conditions, and hope that this will be enough to have met the criteria in the brief.

Bibliography

Sources Used:

Lincoln Virtual Campus

Resources Used:

TextIO.Java - <http://www.faqs.org/docs/javap/source/TextIO.java>