### lmu_blakCourse Submission Cover Sheet Module: CS4001 Programming Engineering

### Assignment no: 001

### Weighting: 30% of module mark

### Deadline:

**Module Leader: Sandra Fernando Student ID:**

PLAGIARISM

You are reminded that there exist regulations concerning plagiarism. Extracts from these regulations are printed below. Please sign below to say that you have read and understand these extracts:

(Signature:) Date:

This header sheet should be attached to the work you submit. No work will be accepted without it.

Extracts from University *Regulations on*

Cheating, Plagiarism and Collusion

Section 2.3: "The following broad types of offence can be identified and are provided as indicative examples …..

1. **Cheating: including taking unauthorised material into an examination; consulting unauthorised material outside the examination hall during the examination; obtaining an unseen examination paper in advance of the examination; copying from another examinee; using an unauthorised calculator during the examination or storing unauthorised material in the memory of a programmable calculator which is taken into the examination; copying coursework.**
2. **Falsifying data in experimental results.**
3. Personation, where a substitute takes an examination or test on behalf of the candidate. Both candidate and substitute may be guilty of an offence under these Regulations.
4. **Bribery or attempted bribery of a person thought to have some influence on the candidate's assessment.**
5. Collusion to present joint work as the work solely of one individual.
6. Plagiarism, where the work or ideas of another are presented as the candidate's own.
7. Other conduct calculated to secure an advantage on assessment.
8. Assisting in any of the above.

Some notes on what this means for students:

1. Copying another student's work is an offence, whether from a copy on paper or from a computer file, and in whatever form the intellectual property being copied takes, including text, mathematical notation and computer programs.

Taking extracts from published sources *without attribution* is an offence. To quote ideas, sometimes using extracts, is generally to be encouraged. Stating an author’s argument and attributing it, perhaps by quoting, achieve quoting ideas

**TABLE OF CONTENTS:**

|  |  |
| --- | --- |
|  |  |
|  |  |
| **CLASS DIAGRAM** | **3** |
|  |  |
|  |  |
| **METHODS DESCRIPTION** | **4** |
|  |  |
|  |  |
| **TESTS** | **7** |
|  |  |
|  | **16** |
| **ERROR DETECTION AND CORRECTION** |  |
|  |  |
|  |  |
| **CONCLUSION** | **23** |
|  |  |
|  |  |
| **APPENDIX: PROJECT CODE** | **24** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**CLASS DIAGRAM**

|  |
| --- |
| **Car** |
| -String description; |
| -String customerName; |
|  |
| + getCarDescription() |
| + getCustomerName (); |
|  |
| + Void setCustomerName (String newCustomerName); |
| + Void display() |
|  |

|  |  |  |
| --- | --- | --- |
| **CarToBuy** |  | **CarToRent** |
|  |  |  |
| -Int price; |  | -String rentalDate; |
| -Int registrationYear; |  | -String returnDate; |
| -Int mileage; |  | -Int adminFee; |
| -Boolean sold; |  | -Int numberOfDays; |
|  |  | -Int dailyRate; |
| + Void setNewPrice (int newPriceCarToBuy); |  | -Int totalAccumulated; |
| +getPrice(); |  | -Boolean onLoan; |
| +getRegYear(); |  |  |
| +getMileage();  +getSaleStatus () |  | +Void TotalAccumulated (int dailyRate, int numberOfDays, int adminFee, Boolean onLoan) |
| +Void getSalesStatus(); |  | +getRentalDate(); |
| +Void buyCar (Sting customerName); |  | +getReturnDate(); |
| +Void display(); |  | +getAdmonFee(); |
|  |  | +getNumberOfDays(); |
|  |  | +getDailyRate(); |
|  |  | +getTotalAccumulated();  +getRentalStatus () |
|  |  | +Void getOnLoanStatus(); |
|  |  | +Void setDailyRate (int newDailyRate); |
|  |  | +Void setAdminFee(int newAdminFee); |
|  |  | +Void rentCar((String customerName, String newRentalDate, String newReturnDate, int newNumberOfDays); |
|  |  | +Void returnCar(); |
|  |  | +Void printCarDescription(); |
|  |  | +Void display(); |
|  |  |  |

**METHODS DESCRIPTION**

1. **Class: Car**

- public String getCarDescription():

an accessor method for car description, returns carDescription

* public String getCustomerName():

an accessor method for customer name, returns customerName

* public void setCustomerName (String newCustomerName):

a setter method, assigns variable customerName to a newCustomerName,

enables change of a customer name

* public void display():

a method displaying the car description, if available, a customer name

1. **Class: CarToBuy**

* public void setNewPrice(int newPriceCarToBuy):

if the car is available for sale:sets new car price, accepts new price as a parameter,

if the car has been sold, outputs an error message

* public int getPrice():

an accessor method for price, returns price

* public int getRegYear():

an accessor method for car registration year, returns registration year

* public int getMileage():

an accessor method for car mileage, returns mileage

* getSaleStatus ():

an accessor method checking if a car is available for sale;

* public void buyCar(String customerName):

a method for buying a car, checks if a car is available for sale,

if the car has been sold, displays relevant message,

if the car is available, sets a customer name, passed as a parameter,

changes the car status to sold

* public void display():

a method displaying car details,

makes a call to display method in Car class to obtain and display car description, if available, the customer name,

checks the car sale status,

if the car is available for sale, displays price, registration year and mileage

**Class: CarToRent**

* public void TotalAccumulated (int dailyRate, int numberOfDays, int adminFee, boolean onLoan):

shows total value of all rentals for the car using totalCarRental variable and adding it to the totalAccumulated so far,

totalCarRental is calculated by multiplying total of daily car rental value by number of days, and adding to that result admin fee

* public String getRentalDate():

an accessor method for rental date, returns rentalDate

* public String getReturnDate():

an accessor method for a rented car return date, returns returnDate

* public int getAdminFee():

an accessor method for a car admin fee, returns adminFee

* public int getNumberOfDays():

an accessor method for number of days the car is rented for, returns numberOfDays

* public int getDailyRate():

an accessor method for daily rental rate, returns daily rental rate

* public int getTotalAccumulated():

an accessor method for total accumulate value of the car rental, returns total accumulated value of rental for a particular car

* getRentalStatus ():

an accessor method for checking the rental status of a car

* public void setDailyRate (int newDailyRate):

a method setting new daily rental rate for the car

* public void setAdmiFee (int newAdminFee):

a method for setting a new admin fee

* public void rentCar (String customerName, String rentalDate, String returnDate, int numberOfDays):

a method for renting a car,

parameters: the customer's name, the rental date, the return date, the number of days the car is required for,

if the car is already rented, prints out the message, stating that the car has been rented and return date;

if the car is available for rent outputs customer name, the rental date, the return date and the number of days;

calls method to set the customer name to set the customer name ,

changes the on loan attribute to true (signifying rented );

the amount paid by the customer is added to the total accumulated

* public void returnCar():

a method for returning the rented car; checks the car rental status:

if the car is rented out, calls the method to set the customer name, passing an empty string as a parameter, sets the number of days is set to 0, sets the date of car rental and return to an empty strings, the on loan status is set to false;

if the car is available for rent, outputs an error message

* public void printCarDescription():

a method to print the car description,

includes its accumulated total from all rentals,

calls the description method in superclass

* public void display():

a method to display the car details,

calls to display method in superclass to bring in the details of the car

if the car is available for rent, displays admin fee, daily rental rate;

if the car is on loan displays the rental date, the return date and the number of days the car has been rented for

**TESTS**

1. Test 1: Inspect a car to buy, buy the car, re-inspect the car to buy

The inspector shows the car created:

**Graphical user interface, website

Description automatically generated**

The display message shows the car description, car price, registration year and mileage:**Graphical user interface, text, application, email

Description automatically generated**

The inspector shows the car data after the price change:

**Graphical user interface

Description automatically generated**

getPrice() method returns new price:

**Graphical user interface, application

Description automatically generated**

The display method outputs the message showing that the car has been sold:

**Graphical user interface, text, application, email

Description automatically generated**

The display method outputs the message after the car has been sold and is no longer available for sale:

**Graphical user interface, text, application, email

Description automatically generated**

1. Test 2: Display all the details of a car to buy

getMileage() returns mileage:

**Graphical user interface

Description automatically generated with medium confidence**

getPrice() method returns price:

**Graphical user interface, application

Description automatically generated**

getRegYear() method returns registration year:

**Graphical user interface, application

Description automatically generated**

1. Test 3: Inspect a car to rent, rent the car, re-inspect the car to rent

The inspector shows details of a car to rent:

**Graphical user interface

Description automatically generated**

The display method outputs details of the car for rent:

Graphical user interface, text, application

Description automatically generated

The inspector shows details of the car rented:

Graphical user interface

Description automatically generated

The display method shows the details of the car rented:

Graphical user interface, text, application

Description automatically generated

1. Test 4: Display the description and total accumulated, before and after renting the car

Description and total accumulated before renting the car:

**Graphical user interface, text, application

Description automatically generated**

Description and total accumulated after renting the car:

Text

Description automatically generated

1. Test 5: Display all the details of a car to rent

The inspector shows details of the car to rent (currently rented):

Graphical user interface

Description automatically generated

getAdminFee() method displays admin fee:

**Graphical user interface, application

Description automatically generated**

getDailyRate() method displays daily rental rate:

Graphical user interface, application

Description automatically generated

getNumberOfDays() method displays the number of days the car has been rented for:

Graphical user interface, application, Word

Description automatically generated

getRentalDate() method displays the date the rental begins:

Graphical user interface, application

Description automatically generated

getReturnDate() method displays the date the rental ends and the car is returned:

Graphical user interface, application

Description automatically generated

getTotalAccumulated() method displays total value of all rentals for the car:

Graphical user interface, application

Description automatically generated

printCarDescription() displays the car description, the customer name (as the car has been rented) and total accumulated value of rental for this car

Text

Description automatically generated

**6. Test 6: Inspect a car to rent, return the car, re-inspect the car to rent**

void ReturnCar() method returns the car and it becomes available for rent, again, as shown in inspector:

Graphical user interface

Description automatically generated

rentalDate, returnDate reset to empty string placeholders, numberOfDays goes back to 0 and the totalAccumulated holds the total value of the rental values for the car, so far

public void setAdminFee (int newAdminFee) sets the new value of admin fee for rental:

Graphical user interface, text, application

Description automatically generated

public void setDailyRate (int newDailyRate) sets the new value of daily rental fee:

Graphical user interface, text, application

Description automatically generated

The inspector shows car details with new value of admin fee and daily rate:

Graphical user interface, website

Description automatically generated

**ERROR DECTECTION AND CORRECTION**

1. **Using assign operator “=” instead of logical equals “==” in boolean expression checking a condition:**

public void setNewPrice (int newPriceCarToBuy)

{

if (sold = true){

System.out.println ("The price cannot be changed, as this car has already been sold.");

}

else {

price = newPriceCarToBuy;

}

}

The inspector displays the object being correctly created:

Graphical user interface, website

Description automatically generated

The display method prints out the details of the car created:

Graphical user interface, text, application, email

Description automatically generated

However, calling setNewPrice (int newPriceCarToBuy) method results in the following message being displayed:

Graphical user interface, text, application, email

Description automatically generated

This is because sold = true assigns the value true to the variable sold, as “=” is an assignment operator.

In order to check validity of a condition it is necessary to use logical equal: “==”, resulting in the following method code:

public void setNewPrice (int newPriceCarToBuy)

{

if (sold == true){

System.out.println ("The price cannot be changed, as this car has already been sold.");

}

else {

price = newPriceCarToBuy;

}

}

1. **No data validity checks:**

The only condition for the new price entered is that it is an integer. There are no checks whether the number entered is a real-life reasonable value. It is possible to enter a negative integer as a car price.

public void setNewPrice (int newPriceCarToBuy)

{

if (sold == true){

System.out.println ("The price cannot be changed, as this car has already been sold.");

}

else {

price = newPriceCarToBuy;

}

}

Without the checks, a negative value is accepted:

Graphical user interface, website

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

In order to avoid such cases, it is necessary to add conditional statements checking the validity of the data entered.

public void setNewPrice (int newPriceCarToBuy)

{

if (sold == true){

System.out.println ("The price cannot be changed, as this car has already been sold.");

}

else {

price = newPriceCarToBuy;

if (newPriceCarToBuy <= 0) {

System.out.println ("The number entered is invalid. Please, enter an integer greater than 0");

}

}

}

1. **Passing variables declared in class as parameters in a method:**

Even though, the method is public, the values passed through its parameters are private and not available outside the method. Passing variables declared at the top of the class in the method parameters results in the values initially assigned to those variables in the constructor being output every time the information about the object is displayed.

public void rentCar (String customerName, String rentalDate, String returnDate, int numberOfDays)  
    {  
        if (onLoan == true) {  
            System.out.println ("The car has already been rented out. The return date of the car is" + returnDate);  
        }  
        else {  
            setCustomerName (customerName);  
            onLoan = true;  
            int totalCarRental = ((dailyRate \* numberOfDays) + adminFee);  
            System.out.println ("This car has been rented by " + customerName + " on " + rentalDate + " for "  
            + numberOfDays + " days. It will be returned on " + returnDate + " The total rental cost is " + totalCarRental);  
            totalAccumulated = totalAccumulated + totalCarRental;  
             
        }    
    }

The inspector screen for the car rented shows empty strings instead of values for rentalDate and returnDate, and 0 for numberOfDays, as those are the values those variables were initialised to in constructor.

Graphical user interface

Description automatically generated

The display message outputs blanks for rent and return dates, and 0 for number of days the car has been rented for: Graphical user interface, text, application

Description automatically generated

In order to correct this, it is necessary to use different names for the method parameters and assigned them to the actual class attributes, making the values entered using the method available to other methods within the class.

The corrected method code is:

public void rentCar (String customerName, String newRentalDate, String newReturnDate, int newNumberOfDays)

{

if (onLoan == true) {

System.out.println ("The car has already been rented out. The return date of the car is "

+ returnDate);

}

else {

setCustomerName (customerName);

rentalDate = newRentalDate;

returnDate = newReturnDate;

numberOfDays = newNumberOfDays;

if (newNumberOfDays <=0){

System.out.println ("Please, enter a valid number of days, greater than 0.");

}

onLoan = true;

int totalCarRental = ((dailyRate \* numberOfDays) + adminFee);

TotalAccumulated (dailyRate, numberOfDays, adminFee, onLoan);

}

}

The inspector shows the car rented, including the actual values for rentalDate, returnDate and the numberOfDays variables.

Graphical user interface

Description automatically generated

The display method message shows the values for rentalDate, returnDate and the number of days the car has been rented:

Graphical user interface, text, application

Description automatically generated

**CONCLUSION**

As well as reviewing the material covered during the first term’s lectures, this coursework helped me understand the structure and working of the code. Although, by and large, it was bringing together the concepts learned through the term, it also posed some challenges.

The most problematic for me was trying to make the rentCar method make the values it received available in the display method in order to display the correct rental and return dates, and the number of days a car has been rented for.

I have used, as the method parameters , the variables initially declared in the class. This resulted in the values initially assigned to them being output every time the information about the car being rented was displayed. I struggled trying to work how to amend the method in order to obtain the correct result. The realisation that, although the method itself was public within the class, the values passed through its parameters were accessible only to the method itself, meant that it was necessary to find a way for those values to be accessible within the class. The answer was to rename the method parameters and assign their values to the initially declared class variables.

Although, I realise that my problem may seem trivial, and one that a more experienced student would most likely have spotted quickly, for me this was a valuable learning experience, as it helped me to understand the role of variables initially declared within a class, the way the method parameters work to pass the values, and how they interact with each other within a class.

**APPENDIX**

**PROJECT CODE**

1. **Class Car**

/\*\*

\* Class Car: a superclass creating an object car, which can be sold through subclass CarToBuy or rented through subclass CarToRent

\*

\*/

public class Car

{

// instance variables

private String description;

private String customerName;

/\*\*

\* Constructor for objects of class Car

\*/

public Car(String carDescription)

{

// initialise instance variables

description = carDescription;

customerName = " ";

}

/\*\*

\* an accessor method for car descrtiption

\* @return descrtiption

\*/

public String getCarDescription()

{

return description;

}

/\*\*

\* an accessor method for car name

\* @return customer name

\*/

public String getCustomerName()

{

return customerName;

}

/\*\*

\* a setter method asigning variable customerName to a newCustomerName, enabling change of a customer name

\* @param newCustomerName, text

\* @return newCustomerName

\*/

public void setCustomerName (String newCustomerName)

{

customerName = newCustomerName;

}

/\*\*

\* a display method outputs the description and, if the customer's name is not an empty string, the customer's name

\* @return car description and, if available, customer name

\*/

public void display()

{

System.out.println ("The car is " + description + ".");

if (customerName != " ") {

System.out.println ("The customer name is " + customerName);

}

}

}

1. **Class CarToBuy**

/\*\*

\* class CarToBuy is a subclass of the Car class.

\* it has four attributes: a price: a whole number, a registration year: a whole number, mileage: a whole number, a boolean value sold;

\*

public class CarToBuy extends Car

{

// instance variables

private int price;

private int registrationYear;

private int mileage;

private boolean sold;

/\*\*

\* Constructor for objects of class CarToBuy;

\* accepts four parameters; the description, the price, the registration year, the mileage;

\* a call to superclass constructor is made with the parameter: carDescription;

\*/

public CarToBuy(String carDescription, int priceCarToBuy, int regYearCarToBuy, int mileageCarToBuy)

{

// initialise instance variables

super(carDescription);

price = priceCarToBuy;

registrationYear = regYearCarToBuy;

mileage = mileageCarToBuy;

sold = false;

}

/\*\*

\* a method settting price;

\* accepts a new price as a parameter;

\* if the car has already been sold: a message informing of this and that the price cannot be changed is displayed;

\* otherwise the new value is assgned to the price attribute;

\*

\* @param newPriceCarToBuy: a whole number

\*/

public void setNewPrice (int newPriceCarToBuy)

{

if (sold == true)

System.out.println ("The price cannot be changed, as this car has already been sold.");

else {

price = newPriceCarToBuy;

if (newPriceCarToBuy <= 0) {

System.out.println ("The number entered is invalid. Please, enter an integer greater than 0. ");

}

}

}

/\*\*

\* an accessor method for price

\* @return descrtiption

\*/

public int getPrice()

{

return price;

}

/\*\*

\* an accessor method for car registration year;

\* @return registration year

\*/

public int getRegYear()

{

return registrationYear;

}

/\*\*

\* an accessor method for car mileage

\* @return car mileage

\*/

public int getMileage ()

{

return mileage;

}

/\*\*

\* an accessor method for car sale status

\*/

public void getSaleStatus ()

{

if (sold == true) {

System.out.println ("The car has been sold. ");

}

else {

System.out.println ("The car is available for sale. ");

}

}

/\*\*

\* a method for buying a car

\* @param customerName: text

\*/

public void buyCar (String customerName)

{

if (sold == true) {

System.out.println ("This car has already been sold and is no longer available.");

}

else {

setCustomerName (customerName);

sold = true;

}

}

/\*\*

\* a method displaying car details;

\* calls on the display method in superclass Car;

\* displays the description and, if available, the customer name

\* if the car has not been sold, displays price, registration year and mileage

\*/

public void display(){

super.display();

if (sold == false) {

System.out.println ("The car price is " + price + ". \nThe registration year is " + registrationYear

+ ". \nThe mileage is " + mileage + ".");

}

else {

System.out.println ("The car has been sold and is no longer available for sale.");

}

}

}

1. **Class CarToRent**

/\*\*

\* class CarToRent is a sublacss of the Car class.

\*

\*it has seven attributes: rental date: a string of text, return date: a string of text, admin fee: a whole number, number of days:

\*a whole number, daily rate: a whole number, total accumulated: a whole number, boolean value on loan

\*/

public class CarToRent extends Car

{

// instance variables

private String rentalDate;

private String returnDate;

private int adminFee;

private int numberOfDays;

private int dailyRate;

private int totalAccumulated;

private boolean onLoan;

/\*\*

\* Constructor for objects of class CarToRent

\* the construcor accepts three parameters: description, the admin fee and the daily rate;

\* a call to superclass constructor is made with the parameter: the description;

\*/

public CarToRent(String carDescription, int adminFeeRental, int dailyRateRental)

{

// initialise instance variables

super(carDescription);

adminFee = adminFeeRental;

dailyRate = dailyRateRental;

rentalDate = " ";

returnDate = " ";

numberOfDays = 0;

totalAccumulated = 0;

onLoan = false;

}

/\*\*

\* TotalAccumulated method shows total value of all rentals for the car

\* using totalCarRental variable and adding it to the totalAccumulated so far

\* totalCarRental is a total of daily car rental value multiplied by number of days, and adding admin fee

\*

\* @param dailyRate: whole number, numberOfDays: whole number, adminFee: whole number, onLoan: boolean value;

\* \*/

public void TotalAccumulated (int dailyRate, int numberOfDays, int adminFee, boolean onLoan){

if(onLoan == true)

{

int totalCarRental = ((dailyRate \* numberOfDays) + adminFee);

totalAccumulated = totalAccumulated + totalCarRental;

}

}

/\*\*

\* an accessor method for rental date

\* @return rentalDate

\*/

public String getRentalDate()

{

return rentalDate;

}

/\*\*

\* an accessor method for return date

\* @return returnDate

\*/

public String getReturnDate()

{

return returnDate;

}

/\*\*

\* an accessor method for car admin fee

\* @return adminFee

\*/

public int getAdminFee()

{

return adminFee;

}

/\*\*

\* an accessor method for number of dayst the car is rented for

\* @return numberOfDays

\*/

public int getNumberOfDays ()

{

return numberOfDays;

}

/\*\*

\* an accessor method for daily rental rate

\* @return dailyRate

\*/

public int getDailyRate ()

{

return dailyRate;

}

/\*\*

\* an accessor method for total accumulated value of the car rental

\* @return totalAccumulated

\*/

public int getTotalAccumulated ()

{

return totalAccumulated;

}

/\*\*

\* an accessor method for checking if a car has been rented or is avaialbe for rent

\*/

public void getRentalStatus ()

{

if (onLoan == false){

System.out.println ("The car is avaialble for rent. ");

}

else {

System.out.println ("The car has already been rented. ");

}

}

/\*\*

\* an method setting new daily rental rate for the car

\* @param newDailyRate, whole number

\*/

public void setDailyRate (int newDailyRate)

{

dailyRate = newDailyRate;

}

/\*\*

\* a method for setting a new admin fee

\* @param newAdminFee, a whole number

\*/

public void setAdminFee (int newAdminFee)

{

adminFee = newAdminFee;

}

/\*\*

\* a method for renting a car

\* accepts as parameters: the customer's name, the rental date, the return date, the number of days the car is required

\* if a car is already rented, prints out the message, stating that the car has been rented and the return date

\* if the car is avaialable for rent outputs customer name,the rental date, the return date and the number of days;

\* the customer name is set through calling the method to set the customer name

\* the on loan atribute is changed to true, signifying that the car has been rented

\* the amount paid by the customer is added to the total accumulated

\* @param customerName, text, newRentalDate, text, newReturnDate, text, newNumberOfDays, whole number

\*/

public void rentCar (String customerName, String newRentalDate, String newReturnDate, int newNumberOfDays)

{

if (onLoan == true) {

System.out.println ("The car has already been rented out. The return date of the car is "

+ returnDate + ".");

}

else {

setCustomerName (customerName);

rentalDate = newRentalDate;

returnDate = newReturnDate;

numberOfDays = newNumberOfDays;

if (newNumberOfDays <=0){

System.out.println ("Please, enter a valid number of days, greater than 0.");

}

onLoan = true;

int totalCarRental = ((dailyRate \* numberOfDays) + adminFee);

TotalAccumulated (dailyRate, numberOfDays, adminFee, onLoan);

}

}

/\*\*

\* a method for returning the rented car

\* if the car is rented out, the method to set the customer name is called and an empty string is passed as a parameter

\* the number of days is set to 0, the date of car rental and return are set to empty strings, the on loan status is set to false

\* otherwise, a relevant error message is output

\*/

public void returnCar (){

if (onLoan == true) {

setCustomerName (" ");

numberOfDays = 0;

rentalDate = " ";

returnDate = " ";

onLoan = false;

System.out.println ("The car has been returned. ");

}

else {

System.out.println ("This car has not been rented, so cannot be returned.");

}

}

/\*\*

\* a method to print the car descrtiption

\* includes its accumulated totatl from all rentals

\* calls the description method in superclass

\*/

public void printCarDescription () {

super.getCarDescription();

super.display();

getTotalAccumulated();

System.out.println (" The total accumulated is " + totalAccumulated + ".");

}

/\*\*

\* a method to display the car details

\* calls to display method in superclass to bring in the details of the car and customer name, if available

\* displays admin fee, daily rental rate

\* if the car is on loan the rental date,the return date and the number of days are also displayed

\*/

public void display() {

super.display();

System.out.println ("The admin fee is " + adminFee + ". \nThe daily rate is " +

dailyRate + ".");

if (onLoan == true ) {

getNumberOfDays ();

System.out.println ("The rental date is " + rentalDate + ". \nThe return date is "

+ returnDate + ". \nThe car has been rented for " + numberOfDays + ".");

}

}

}