```
import javax.swing.JOptionPane;

/*******************************************************************************************
 *  This class demonstrates the functionality of the CarRental class
 *  by allowing the user to enter information about a car rental then
 *  displaying summary information about the rental including its cost.
 *
 *  The user is prompted to input information regarding a car rental, the information
 *  is validated then a summary is displayed and the user is asked
 *  if the user wants to input another rental
 *
 *  CST 183 Programming Assignment 4
 *  @author Michael Clinesmith
 */

public class CarDriver
{
    /**
     * The main function is the driver of the program, requesting messages
     * to be displayed to the user and requesting input regarding a car rental.
     * with invalid rental input the user is requested for valid input and
     * summary information regarding valid rental input is displayed then the
     * user is requested if another rental is to be entered
     *
     * @param args String array - not used
     */
    public static void main(String[] args)
    {
        // declarations
        CarRental rental = new CarRental(); // required to initialize otherwise compiler error
        String outputString;
        char type;
        int days;
        double startMileage, endMileage;

        boolean validRental = false;        // flag for if rental data is valid
        char another;                       // flag to do another rental

        // introduction message

        outputString ="Welcome to the CarRental Program!\n\n" +
                "This program allows you to enter information regarding \n" +
                "a car rental, then will process the information and \n" +
                "produce a summary output including rental charges.\n\n" +
                "Program designed by Michael Clinesmith";
        JOptionPane.showMessageDialog(null, outputString);

        // process car rentals
        do                                  // loop allows user to enter multiple rentals, but at least once
        {
            validRental = false;            // flag to require valid data
            while (!validRental)            // loop until valid rental data
            {
                // request rental data

                outputString = "Enter the type of rental:\n" +
                        "------------------------\n" +
                        "Budget\n" +
                        "Daily\n" +
                        "Weekly\n";
```

30/30 points for Program 4

Great solution. Tested well.
Coding excellent overall. Nice job.

Testing Focus
- Driver file and CarRental class file
- Basic user interaction
- CarRental class with constructors, get/set methods
- Interaction with drive and CarRental object
- Accuracy and completeness of cost calculation method

Test Cases

1) B 2 days Odom: 1111 - 1234
   Expected output: $110.75

2) D 6 days Odom: 1111 - 1234
   Expected output: $360.00

3) W 22 days Odom: 1111 - 6666
   Expected output: $1160.00

4) Various error-checking tests

```
            type = inputChar(outputString);

            outputString =  "Enter the whole number of days for the rental:\n" +
                            "Rentals may be made to a maximum of 60 days.";
            days = inputInt(outputString);

            outputString = "Enter the beginning mileage:";
            startMileage = inputDouble(outputString);

            outputString = "Enter the ending mileage:";
            endMileage = inputDouble(outputString);

            rental = new CarRental(type, days, startMileage, endMileage);

            validRental = rental.isValidRental();
            if (!validRental)           // if data not valid, user needs to try again
            {
                outputString =  "The data for the car rental is not valid.\n" +
                                "Please try again with valid data.";
                JOptionPane.showMessageDialog(null, outputString, "ERROR",
                        JOptionPane.ERROR_MESSAGE);
            }
        }

        // display rental information
        outputString = rental.rentalSummaryToString();
        JOptionPane.showMessageDialog(null, outputString);

        outputString = "Do you want to process another rental? (Y/N)";
        another = inputChar(outputString);
    }
    while (another == 'Y' || another == 'y');

    // ending message

    outputString ="Thank you for using the CarRental Program!\n";
    JOptionPane.showMessageDialog(null, outputString);

}

/**
 *  inputChar() gets a char of input from a user using dialog box
 *  This method validates the input to prevent runtime errors
 *
 * @param outputString  The string to be displayed to the user to get the required char
 * @return  a char from the first character entered by the user
 */
public static char inputChar (String outputString)
{
    String input;
    String messageString;
    char inputChar = ' ';
    boolean isValid = false;

    while (!isValid)
    {

        input = JOptionPane.showInputDialog(outputString);
        if (input == null)
        {
```

```
            messageString = "Proper input not entered.\n" +
                            "Please enter input in the correct format.";

            JOptionPane.showMessageDialog(null, messageString, "ERROR",
                    JOptionPane.ERROR_MESSAGE);
        }
        else if (input.length()==0)
        {
            messageString = "Proper input not entered.\n" +
                    "Please enter input in the correct format.";

            JOptionPane.showMessageDialog(null, messageString, "ERROR",
                    JOptionPane.ERROR_MESSAGE);
        }
        else
        {
            inputChar = input.charAt(0);
            isValid = true;
        }

    }
    return inputChar;
}

/**
 *  inputInt() gets a int of input from a user using dialog box
 *  This method validates the input to prevent runtime errors
 *
 * @param outputString  The string to be displayed to the user to get the required int
 * @return  an int entered by the user
 */
public static int inputInt (String outputString)
{
    String input;
    String messageString;
    int inputInt = 0;
    boolean isValid = false;

    while (!isValid)
    {

        try                     // used to catch bad input data exception
        {
            input = JOptionPane.showInputDialog(outputString);
            if (input == null)
            {
                messageString = "Proper input not entered.\n" +
                        "Please enter input in the correct format.";

                JOptionPane.showMessageDialog(null, messageString, "ERROR",
                        JOptionPane.ERROR_MESSAGE);
            } else if (input.length() == 0)
            {
                messageString = "Proper input not entered.\n" +
                        "Please enter input in the correct format.";

                JOptionPane.showMessageDialog(null, messageString, "ERROR",
                        JOptionPane.ERROR_MESSAGE);
            } else
            {
```

```
                inputInt = Integer.parseInt(input);
                isValid = true;
            }
        }
        catch (NumberFormatException e)     // catches exception where user inputs improperly formatted data
        {
            messageString = "Proper input not entered.\n" +
                    "Please enter input in the correct format.";

            JOptionPane.showMessageDialog(null, messageString, "ERROR",
                    JOptionPane.ERROR_MESSAGE);

        }
    }
    return inputInt;
}

/**
 *  inputDouble() gets a double of input from a user using dialog box
 *  This method validates the input to prevent runtime errors
 *
 * @param outputString  The string to be displayed to the user to get the required double
 * @return  a double entered by the user
 */
public static double inputDouble (String outputString)
{
    String input;
    String messageString;
    double inputDouble = 0.0;
    boolean isValid = false;

    while (!isValid)
    {
        try                 // used to catch bad input data exception
        {
            input = JOptionPane.showInputDialog(outputString);
            if (input == null)
            {
                messageString = "Proper input not entered.\n" +
                        "Please enter input in the correct format.";

                JOptionPane.showMessageDialog(null, messageString, "ERROR",
                        JOptionPane.ERROR_MESSAGE);
            } else if (input.length() == 0)
            {
                messageString = "Proper input not entered.\n" +
                        "Please enter input in the correct format.";

                JOptionPane.showMessageDialog(null, messageString, "ERROR",
                        JOptionPane.ERROR_MESSAGE);
            } else
            {
                inputDouble = Double.parseDouble(input);
                isValid = true;
            }
        }
        catch (NumberFormatException e)     // catches exception where user inputs improperly formatted data
        {
            messageString = "Proper input not entered.\n" +
                    "Please enter input in the correct format.";
```

```
                JOptionPane.showMessageDialog(null, messageString, "ERROR",
                        JOptionPane.ERROR_MESSAGE);
            }
        }
        return inputDouble;
    }
}
```

```
/*******************************************************************************
 * This class implements basic functionality for a car rental class
 * storing the type, number of days, and starting and ending mileage for a rental
 * and also being able to calculate the cost for the rental
 *
 *
 * CST 183 Programming Assignment 4
 * @author Michael Clinesmith
 *
 ******************************************************************************/

public class CarRental
{
    // field declarations
    private char type;          // the classification for the rental
    private int days;           // the number of days rented
    private int startMile;      // the beginning mileage for the rental
    private int endMile;        // the ending mileage for the rental

    // price constants which are the same for all elements of the class, hence the static keyword
    private static final double  BUDGET_BASE = 40.0, DAILY_BASE = 60.0, WEEKLY_BASE = 190.0;
    private static final double  MILEAGE_RATE = .25;
    // mileage charge ranges are hard coded into the dailyMileage and weeklyMileage methods


    /**
     * Constructor with no arguments
     */
    public CarRental()
    {
        type = 'B';
        days = 1;
        startMile = 0;
        endMile = 10;
    }

    /**
     * Constructor with given parameters including mileages as integers
     * @param t  character representing the classification of the rental
     * @param d  int representing the number of days rented
     * @param s  int representing the starting mileage
     * @param e  int representing the ending mileage
     */
    public CarRental(char t, int d, int s, int e)
    {
        type = t;
        days = d;
        startMile = s;
        endMile = e;
    }

    /**
     * Constructor with given parameters but with mileages as doubles
     * @param t  character representing the classification of the rental
     * @param d  int representing the number of days rented
     * @param s  double representing the starting mileage
     * @param e  double representing the ending mileage
     */
    public CarRental(char t, int d, double s, double e)
    {
```

Great job with the class. Very thorough work.
Professional-level coding.

```
    type = t;
    days = d;
    startMile = (int) Math.ceil(s);
    endMile = (int) Math.ceil(e);
}


/*************************************************************
 * Note that IntelliJ created the setter and getter methods
 * Then it later squeezed them into one line for some strange reason
 *************************************************************/

/**
 * Setter for classification of rental with 'B' for budget, 'D' for daily, and 'W' for weekly
 * @param type  character representing rental classification
 */
public void setType(char type)
{
    this.type = type;
}

/**
 * Setter for setting number of days for rental
 * @param days  int for number of days for rental
 */
public void setDays(int days)
{
    this.days = days;
}

/**
 * Setter for setting the starting mileage
 * @param startMile int for beginning mileage
 */
public void setStartMile(int startMile)
{
    this.startMile = startMile;
}

/**
 * Overloaded setter for setting starting mileage if the user gives a double
 * @param startMile double for beginning mileage (to be rounded up)
 */
public void setStartMile(double startMile)
{
    this.startMile = (int) Math.ceil(startMile);
}

/**
 * Setter for ending mileage
 * @param endMile int for ending mileage
 */
public void setEndMile(int endMile)
{
    this.endMile = endMile;
}

/**
 * Overloaded setter for ending mileage if the user gives a double
 * @param endMile   double for ending mileage (to be rounded up)
```

```
 */
public void setEndMile(double endMile)
{
    this.endMile = (int) Math.ceil(endMile);
}

/**
 * getter for the rental classification with 'B' for budget, 'D' for daily, and 'W' for weekly
 * @return char indicating rental classification
 */
public char getType()
{
    return type;
}

/**
 * getter for the number of days rented
 * @return  int for the number of days rented
 */
public int getDays()
{
    return days;
}

/**
 * getter for the starting mileage
 * @return  int for the starting mileage
 */
public int getStartMile()
{
    return startMile;
}

/**
 * getter for the ending mileage
 * @return  int for the ending mileage
 */
public int getEndMile()
{
    return endMile;
}

/**
 * the method checks whether the rental information is valid
 * @return boolean value true if the record if valid, false if it contains invalid elements
 */
public boolean isValidRental()
{
    boolean valid = true;       // assume true and check if false
    if ( type !='B' && type !='b' && type !='D' && type !='d' && type !='W' && type !='w')
    {
        valid = false;
    }
    else if (days < 0 || days > 60)
    {
        valid = false;
    }
    else if (startMile < 0)
    {
        valid = false;
```

```
    }
    else if (endMile <= startMile)
    {
        valid = false;
    }

    return valid;
}

/**
 * This method calculates the cost of the rental
 * @return  double (with 2 decimal places) for the rental cost
 */
public double rentalCost()
{
    double billCharge;

    if(this.isValidRental())
    {
        billCharge = rentalBaseCost() + rentalMileageCost();

        billCharge = dollars(billCharge);   // rounds charge up to two decimal places if necessary

    }
    else
    {
        billCharge = -1.0;              //  return -1 if invalid rental
    }

    return billCharge;
}

/**
 * This method calculates the cost of the rental based on the plan and number of days rented
 * @return double representing the cost of the rental based on the plan and number of days rented
 */
public double rentalBaseCost()
{
    double baseCost;

    if (type == 'B' || type == 'b')
    {
        baseCost = budgetBase();
    }
    else if (type == 'D' || type == 'd')
    {
        baseCost = dailyBase();
    }
    else if (type == 'W' || type == 'w')
    {
        baseCost = weeklyBase();
    }
    else
    {
        baseCost = -1.0;
    }

    return dollars(baseCost);
}
```

```
/**
 * This method calculates the cost of the rental based on the plan and the distance travelled
 * @return double representing the cost of the rental based on the plan and distance travelled
 */
public double rentalMileageCost()
{
    double mileageCost;

    if (type == 'B' || type == 'b')
    {
        mileageCost = budgetMileage(endMile-startMile);
    }
    else if (type == 'D' || type == 'd')
    {
        mileageCost = dailyMileage(endMile-startMile);
    }
    else if (type == 'W' || type == 'w')
    {
        mileageCost = weeklyMileage(endMile-startMile);
    }
    else
    {
        mileageCost = -1.0;
    }

    return dollars(mileageCost);
}

/**
 * rentalTypeString() converts the character representing the type of rental to a
 * string stating the type of rental
 * @return String stating the type of rental
 */
public String rentalTypeString()
{
    String strType;

    if (type == 'B' || type == 'b')
    {
        strType = "Budget Rental";
    }
    else if (type == 'D' || type == 'd')
    {
        strType = "Daily Rental";
    }
    else if (type == 'W' || type == 'w')
    {
        strType = "Weekly Rental";
    }
    else
    {
        strType = "Invalid Type";
    }

    return strType;
}

/**
 * rentalSummaryToString() takes data regarding an CarRental object and saves it as a string
 * @return  a String containing a summary of a CarRental object
```

```
 */
public String rentalSummaryToString()
{
    String outputString;

    outputString =  "Rental Type: " + rentalTypeString() + "\n" +
                    "Beginning Mileage: " + getStartMile() + "\n" +
                    "Ending Mileage: " + getEndMile() + "\n" +
                    "Miles Driven: " + distance() + "\n" +
                    "Days Rented: " + getDays() + "\n" +
                    "Rental Base Charge: $" + String.format("%.2f", rentalBaseCost()) + "\n" +
                    "Rental Mileage Charge: $" + String.format("%.2f", rentalMileageCost()) + "\n" +
                    "Total Rental Charge: $" + String.format("%.2f", rentalCost());

    return outputString;
}

/**
 * This method calculates the total mileage for the rental
 * @return int representing the miles travelled
 */
public int distance()
{
    return endMile - startMile;
}

/*-------------------------------------------------------------------------
The following methods are private and not accessible outside of the class
 -------------------------------------------------------------------------*/

/**
 * This method calculates the daily charge when doing a budget rental
 * @return  the daily cost for the budget rental
 */
private double budgetBase()
{
    return BUDGET_BASE * days;
}

/**
 * This method calculates the daily charge when doing a daily rental
 * @return  the daily cost for the daily rental
 */
private double dailyBase()
{
    return DAILY_BASE * days;
}

/**
 * This method calculates the weekly charge when doing a weekly rental
 * @return  the weekly cost for the weekly rental
 */
private double weeklyBase()
{
    return WEEKLY_BASE * weeks();
}

/**
 * This method returns the number of weeks from the number of days of the rental
 * @return  the number of weeks of the rental
```

```
 */
private int weeks()
{
    int Weeks;
    Weeks = days / 7;
    if (days % 7 != 0)                // if a partial week, charge for it
    {
        Weeks++;
    }
    return Weeks;
}

/**
 * This method returns the mileage charge for a budget rental
 * @param miles int representing the number of miles travelled
 * @return the mileage cost of the budget rental
 */
private double budgetMileage(int miles)
{
    return MILEAGE_RATE * miles;
}

/**
 * This method returns the mileage charge for a daily rental
 * This method includes a hard coded range for no mileage charge for the rental
 *
 * @param miles int representing the number of miles travelled
 * @return the mileage cost of the daily rental
 */
private double dailyMileage(int miles)
{
    double cost;

    if (miles <= (100 * days))                    // miles less than 100 per day
    {
        cost = 0;
    }
    else
    {
        cost = MILEAGE_RATE * (miles - (100 * days));
    }

    return cost;
}

/**
 * This method returns the mileage charge for a weekly rental
 * This method includes a hard coded ranges for no mileage charge for the rental
 *
 * @param miles int representing the number of miles travelled
 * @return the mileage cost of the weekly rental
 */
private double weeklyMileage(int miles)
{
    int Weeks = weeks();
    double cost;
    if (miles <= (900 * Weeks))
    {
        cost = 0;
    }
```

```
        else if (miles <= (1500 * Weeks))
        {
            cost = 100 * Weeks;
        }
        else
        {
            cost = 200 * Weeks + MILEAGE_RATE * (miles - (1500 * Weeks));
        }

        return dollars(cost);
    }

    /**
     *  This method converts a double into a double with only two decimal places, rounding up if necessary
     * @param doub a double that is to be represented as a money amount
     * @return the money equivalant of the doub variable (rounded up)
     */
    private double dollars(double doub)
    {
        return Math.ceil(doub * 100) / 100.0;
    }

}
```