NATIONAL SENIOR CERTIFICATE EXAMINATION
NOVEMBER 2023

INFORMATION TECHNOLOGY: PAPER I

MARKING GUIDELINES

Time: 3 hours 150 marks

These marking guidelines are prepared for use by examiners and sub-examiners, all of whom are required to attend a standardisation meeting to ensure that the guidelines are consistently interpreted and applied in the marking of candidates' scripts.

The IEB will not enter into any discussions or correspondence about any marking guidelines. It is acknowledged that there may be different views about some matters of emphasis or detail in the guidelines. It is also recognised that, without the benefit of attendance at a standardisation meeting, there may be different interpretations of the application of the marking guidelines.

SECTION A SQL

QUESTION 1.1 [4]

```
SELECT *
FROM tblParkings
WHERE PremiumParking = true
ORDER BY DailyRate
```

QUESTION 1.2 [5]

```
SELECT*
FROM tblCars
WHERE CarRegistration LIKE '*GP'
AND MID(CarRegistration, 3,1) = ' '
```

Accept % for mysql

JavaDB

```
AND SUBSTR (CarRegistration, 3,1) = ' '
```

QUESTION 1.3 [6]

```
SELECT Address, DailyRate, DateAdded, PremiumParking
FROM tblParkings
WHERE PremiumParking = True OR
(DailyRate>135 AND YEAR (dateAdded) = 2022)
```

QUESTION 1.4 [5]

```
SELECT *
FROM tblParkings
WHERE DailyRate = (SELECT MAX(DailyRate)
                   FROM tblParkings)
```

QUESTION 1.5 [4]

```
SELECT  tblCars.CarRegistration, Model
FROM tblCars
LEFT JOIN tblRentedParkings
ON tblRentedParkings.CarRegistration = tblCars.CarRegistration
WHERE tblRentedParkings.CarRegistration IS NULL
```

Accept ParkingID, StartDate or EndDate is NULL

```
SELECT *
FROM tblCars
WHERE tblCars.CarRegistration
           NOT IN (SELECT tblRentedParkings.CarRegistration
                   FROM tblRentedParkings)
```


QUESTION 1.6 [7]

```
SELECT Address , Count(*) AS TimesRented
FROM tblParkings, tblRentedParkings
WHERE tblParkings.ParkingID = tblRentedParkings.ParkingID
GROUP BY Address
HAVING Count(*) > 2
```

Accept

```
HAVING Count(*) >= 3
```

MySQL

```
HAVING TimesRented > 2
HAVING TimesRented >= 3
```

QUESTION 1.7 [9]

```
SELECT tblCars.CarRegistration, Owner, DailyRate, StartDate,
EndDate, (EndDate - StartDate) * DailyRate AS RentAmount
FROM tblCars, tblRentedParkings, tblParkings
WHERE tblCars.CarRegistration=tblRentedParkings.CarRegistration
And tblParkings.ParkingID=tblRentedParkings.ParkingID;
```

JAVADB

```
SELECT tblCars.CarRegistration, Owner, DailyRate, StartDate,
EndDate, {fn TIMESTAMPDIFF
(SQL_TSI_DAY,StartDate,ENDDATE)} * DailyRate) AS RentAmount
FROM tblCars✓, tblRentedParkings, tblParkings
WHERE tblCars.CarRegistration=tblRentedParkings.CarRegistration
And tblParkings.ParkingID=tblRentedParkings.ParkingID;
```

QUESTION 1.8 [6]

```
UPDATE tblCars
SET Model = 'VW' & RIGHT(Model, LEN(Model)-11)
WHERE Model LIKE 'Volkswagen*'
```

Accept

```
SET Model = 'VW' & RIGHT(Model, LEN(Model)-10)
WHERE Model LIKE 'Volkswagen%'
```

JavaDB

```
SET Model = 'VW' || SUBSTR(Model, 11, LENGTH(MODEL))
```

QUESTION 1.9 [4]

```
DELETE *
FROM tblParkings
WHERE DateAdded < #2018/01/01#
```

Accept

```
WHERE YEAR(DateAdded) < 2018
```


SECTION B OBJECT ORIENTATED PROGRAMMING

JAVA SOLUTION

QUESTION 2 Daily.java

```
//Q2.1 - 4
// class header
public class Daily
{
    // all fields private
    // typed correctly
    // named correctly
    private String registrationNumber;
    private LocalDateTime entryDateTime;
    private LocalDateTime exitDateTime;

    //Q2.2 - 2
    // fields declared as public
    // and final
    //named and assigned correctly
    public static final double HOURLYRATE = 24.50;
    public static final double FINEAMOUNT = 850;

    //Q2.3 - 3
    // header correct
    // parameters named and typed correctly
    public Daily(String inRN, LocalDateTime inEY,
                 LocalDateTime inET)
    {
        // fields assiged correctly
        registrationNumber = inRN;
        entryDateTime = inEY;
        exitDateTime = inET;
    }

    //Q2.4 - 2
    // All getter methods named correctly
    // All return correct type
    public String getRegistrationNumber()
    {
        return registrationNumber;
    }

    public LocalDateTime getEntryDateTime()
    {
        return entryDateTime;
    }

    public LocalDateTime getExitDateTime()
    {
        return exitDateTime;
    }
```



```
    //Q2.5 - 8
    public double getParkingFee()
    {

        // check if days are different
        // convert DatesTime to date
                //- accept any correct alternate answer
                // possible alternative
                // if (entryDateTime.getDayOfYear() !=
                // exitDateTime.getDayOfYear()) {
        if (!entryDateTime.toLocalDate().equals
           (exitDateTime.toLocalDate()))
        {
            // return FINEAMOUNT
            return FINEAMOUNT;
        } else
        {
            // get time diff
            // convert both DateTime object to Time objects
            // create appropriate object using diff
            Duration diff = Duration.between
                           (entryDateTime.toLocalTime(),
                            exitDateTime.toLocalTime());
            long secs = diff.toSeconds();
            LocalTime rDiff = LocalTime.ofSecondOfDay(secs);

            // calculate fee using HOURLYRATE
            double fee = HOURLYRATE * rDiff.getHour();
            // return fee
            return fee;
        }
    }

    //Q2.6 - 4
    // header and return correct
    public String toString()
    {
        // contains all fields
        // formating correct
        // calls getParkingFee
        return "Registration: " + registrationNumber +
               "\tFee: R" + getParkingFee() +
               "\nEntry: " + entryDateTime +
               " Exit: " + exitDateTime ;
    }

}
```

QUESTION 2 LongTerm.java

```
//Q3.1 - 2
// class named correctly
// extends Daily
public class LongTerm extends Daily {
    //Q3.2 - 1
    // field typed and named correctly
    private String parkingBay;

    //Q3.3 - 1
    // field declared public, final, typed and named correctly
    public static final double RATEPERDAY = 250;

    //Q3.4 - 4
    // constructor header correct
    public LongTerm(String inRN, LocalDateTime inEY, LocalDateTime
inET , String inPB)
    {
        // parent constructor called
        // correct parameters given
        super(inRN, inEY, inET);

        // child class field assigned using parameters
        parkingBay = inPB;
    }

    //Q3.5 - 1
    // headers correct and returning correct field
    public String getParkingBay()
    {
        return parkingBay;
    }

    //Q3.6 - 5
    // correct header name and return double
    public double getParkingFee()
    {
        // convert fields to Date objects
        // determine the difference in dates
        Period diff = Period.between(
                    getEntryDateTime().toLocalDate(),
                    getExitDateTime().toLocalDate());
        // calculation correct using RATEPERDAY
        // use correct method to access the number of days
        double fee = diff.getDays() * RATEPERDAY;
        return fee;
    }
```



```
    //Q3.7  - 3
    // header correct and return string
    public String toString()
    {
        // calling toString from parent
        // appends field
        return super.toString() + "\nParking Bay: " + parkingBay;
    }
}
```

QUESTION 4 & 6.1,6.2 ParkingManager.java

```
//Q4.1 - 1
// Class header correct
public class ParkingManager {

    //Q4.2 - 4
    // fields private
    // array of 50 Daily
    // type parent class
    // size field correct type
    private Daily pArr[] = new Daily[50];
    private int size = 0;

    //Q4.3 - 11
    //constructor
    public ParkingManager() {
        try {
            // open  and loop through file
            String reg, entSt, exSt, bay = "";
            Scanner scFile = new Scanner(new
                             File("parkings.txt"));
            while (scFile.hasNextLine()) {
                // read line from text file
                String line = scFile.nextLine();
                // from text file extract first 3 fields
                Scanner scLine = new Scanner(line).
                                 useDelimiter(";");
                reg = scLine.next();
                entSt = scLine.next();
                exSt = scLine.next();
                // from text file  use correct DateTime format
                DateTimeFormatter format =
                                  DateTimeFormatter.ofPattern
                                  ("yyyy/MM/dd HH:mm");
                // create DateTime objects
                LocalDateTime entry = LocalDateTime.parse
                                      (entSt, format);
                LocalDateTime exit = LocalDateTime.parse
                                     (exSt, format);
```

```
                // check for long term parking bay
                if (scLine.hasNext()) {
                    // get parking bay
                    bay = scLine.next();
                    // create LongTermParking object and
                    //add to array
                    pArr[size] = new LongTerm
                                        (reg, entry, exit, bay);
                } else {
                    // create Daily object and add to array
                    pArr[size] = new Daily(reg, entry, exit);
                }
                // increment size
                size++;
                scLine.close();
            }
            scFile.close();
        } catch (FileNotFoundException e) {
            System.out.println("File Missing");
        }
    }

    //Q4.4 - 4
    // header correct and return string
    public String toString() {
        String r = "";
        // loop through array
        for (int i = 0; i < size; i++) {
            // append to string
            // add a blank line
            r += pArr[i] + "\n\n";
        }
        return r;
    }

    //Q4.5 - 6
    public void sortByRegistration() {
        // outside loop correct
        for (int i = 0; i < size - 1; i++) {
            // inside loop correct
            for (int j = i + 1; j < size; j++) {
                // compare the correct elements
                // using registration number
                if (pArr[i].getRegistrationNumber().
                    compareToIgnoreCase
                    (pArr[j].getRegistrationNumber()) > 0) {
                    // store into temp with correct type
                    Daily temp = pArr[j];
                    // swap elements
                    pArr[j] = pArr[i];
                    pArr[i] = temp;
                }
            }
        }
```

```
    }
    //Q6.1 - 6
    // correct method header and return
    public String getRegistrationList() {
        String temp = "";
        //loop through array
        for (int i = 0; i < size; i++) {
            // check if car registration
            // is already in temp
            if (!temp.contains(pArr[i].getRegistrationNumber())) {
                // add registration to list
                // add a # character
                temp += pArr[i].getRegistrationNumber() + "#";
            }
        }
        return temp;
    }

//Q6.2 - 1
// method header correct and return
    public String generateBilling() {
        //Q6.2.1 - 1
        // call getRegistrationList and assign to cars
        String cars = getRegistrationList();

        String bay = "", output = "";
        LongTerm temp = null;

        //6.2.2
        //isolate registration numbers - 3
        // open scanner
        // Alternate method - use split to convert to an array
        Scanner scCars = new Scanner(cars).useDelimiter("#");
        String currentReg;
        // check for more registration numbers
        // loop through array to end at array.length field
        while (scCars.hasNext()) {
            // extract registration number
            // access array element
            currentReg = scCars.next();

            //create a string - 3*
            // *append the registration number before the entries
            output = output + currentReg + "\n";

            //search and add to total - 5
            // initialise total inside the loop
            double total = 0;
            boolean Daily = false;
            // loop through array
            for (int i = 0; i < size; i++) {
                // check if car registration already in list
                if (currentReg.equals(pArr[i].
                    getRegistrationNumber())) {
```

```
                    // add parking fee
                    // to total
                    total += pArr[i].getParkingFee();

                    // *append each entry details to output
                    output += pArr[i].getEntryDateTime() + " "
                            + pArr[i].getExitDateTime() + " "
                            + pArr[i].getParkingFee() + "\n";

                    //check if long-term parking bay - 4
                    // check if pArr[i]
                    // is a Long-Term object
                    if (pArr[i] instanceof LongTerm) {
                        Daily = false;
                        // cast pArr[i] to a LongTerm object
                        temp = (LongTerm) pArr[i];
                        // record parking bay number
                        bay = temp.getParkingBay();
                    } else {
                        Daily = true;
                        bay = "";
                    }
                }
            }
            // *append the parking fee
            output += "Total parking fees due: R" + total + "\n";

            //add the bay number - 2
            // check if the registration is a LongTerm object
            if (Daily == false) {
                // append the bay number
                output += "Long-term bay number: " + bay + "\n\n";

                //add message - 2
                // check if the total is above 1500
            } else if (total > 1500 && Daily == true) {
                // append the message
                output += "Consider renting long-term
                           parking\n\n";
            }
            else output += "\n\n";
        }
        return output;
    }
}
```

QUESTION 5, 6.3 ParkingUI.java

```
//Q5.1 - 1
// class header correct
public class ParkingUI
{
    public static void main(String[] args)
    {
        //Q5.2 - 2
        // ParkingTagManager object created
        // in correct place
        ParkingManager pm = new ParkingManager();

        //Q5.3 - 2
        // sort method called
        pm.SortByRegistration();
        // toString method called
        System.out.println(pm);

        //Q6.3 - 2
        // correct method name
        // in an output statement (typed method call)
        System.out.println(pm.generateBilling());
    }
}
```


DELPHI SOLUTION

QUESTION 2 uDaily.pas

```
unit uDaily;

interface
uses SysUtils, DateUtils;
//Q2.1 - 4
//class header
type TDaily = class
  //all fields private
  //typed correctly
  //named correctly
  private
    registrationPlate : string;
    entryDateTime : TDateTime;
    exitDateTime : TDateTime;


  public
     //Q2.2 - 2
    //field declared as public and final
    //named and assigned correctly
    const
      HOURLYRATE = 24.50;
      FINEAMOUNT = 850;

    constructor Create( inRN:string ; inEY: TDateTime; inET:
TDateTime ) ;
    function getRegistrationPlate() : string;
    function getEntryDateTime() : TDateTime;
    function getExitDateTime() : TDateTIME;
    function getParkingFee() : double; virtual;
    function toString() : string; virtual;

end;

implementation

{ TDaily }
//Q2.3 - 3
//header correct
//parameters named and typed correctly
constructor TDaily.Create(inRN: string; inEY, inET: TDateTime);
begin
     //fields assiged correctly
    registrationPlate := inRN;
    entryDateTime := inEY;
    exitDateTime := inET
end;
```



```
//Q2.4 - 2
//All getter methods named correctly
//All return correct type
function TDaily.getEntryDateTime: TDateTime;
begin
  Result := entryDateTime;
end;

function TDaily.getExitDateTime: TDateTIME;
begin
  Result := exitDateTime;
end;

function TDaily.getRegistrationPlate: string;
begin
  Result:= registrationPlate;
end;

//Q2.5 - 8
function TDaily.getParkingFee: double;
var
  fee : double;
  sBetween, hBetween , mBetween : integer;
begin
   //check if days are different
   // convert DatesTime to date
   //accept any correct alternate answer
    if NOT(DayOfTheYear(entryDateTime) =
DayOfTheYear(exitDateTime)) then
    begin
     //return FINEAMOUNT
         Result := FINEAMOUNT;
    end
    else
    begin
         //get time diff
         //convert both DateTime object to Time objects
         //convert to hours
         fee:= 0;

         sBetween := SecondsBetween( entryDateTime ,
exitDateTime);
         hBetween := sBetween div 3600;
         sBetween := sBetween - hBetween * 3600;
         mBetween := sBetween div 60;

          //calculate fee using HOURLY RATE
         fee := HOURLYRATE * hBetween;
           //return fee
         Result := fee;
    end;

end;
```



```
//Q2.6 - 4
//header and return correct
function TDaily.toString: string;
begin
   //contains all fields
  //formating correct
  //calls getParkingFee
   Result := 'Registation: ' + registrationPlate + #13#10 +
'Entry: ' + DateTimeToStr( entryDateTime ) + #13#10 + 'Exit: ' +
DateTimeToStr( exitDateTime);
end;

end.
```

QUESTION 3 uLongTerm.pas

```
unit uLongTerm;

interface
 uses SysUtils, DateUtils, uDaily;
 //Q3.1 - 2
//class named correctly
//extends Daily
 type TLongTerm = class(TDaily)
  //Q3.2 - 1
    //field typed and named correctly
   private
    parkingBay : string;

   public
    //Q3.3 - 1
    //field declared public, final, typed and named correctly
    const
      RATEPERDAY = 250.0;

    constructor Create(inRN:string ; inEY: TDateTime; inET:
TDateTime; inPB : string);
    function getParkingBay() : string;
    function getParkingFee() : double ; override;
    function toString() : string ; override;
 end;

implementation

{ TLongTerm }
```

```
//Q3.4 - 4
//constructor header correct
constructor TLongTerm.Create(inRN: string; inEY : TDateTime; inET:
TDateTime; inPB: string);
begin
  //parent constructor called
  //correct parameters given
  Inherited Create( inRN, inEY, inET);
    //child class field assigned using parameters
  parkingBay := inPB;
end;
//Q3.5 - 1
//headers correct and returning correct field
function TLongTerm.getParkingBay: string;
begin
  Result := parkingBay;
end;

  //Q3.6 - 5
    //correct header name and return real/double
function TLongTerm.getParkingFee: double;
var
  diff : integer;
  fee : real;
begin
   //convert fields to Date objects
   //determine the difference in dates
   diff:= DaysBetween(getEntryDateTime(), getExitDateTime);
    //calculation correct using RATEPERDAY
    //use correct method to access the number of days
   fee := diff * RATEPERDAY;

   Result := fee;
end;

//Q3.7  - 3
//header correct and returns string
function TLongTerm.toString: string;
begin
   //calling toString from parent
   //appends field
  Result := Inherited toString + '' + #13#10 + 'ParkingBay ' +
parkingBay;
end;

end.
```


QUESTION 4 & 6.1,6.2 uParkingManager.pas

```
unit uParkingManager;

interface
uses SysUtils, DateUtils, uDaily, uLongTerm;
//Q4.1 - 1
//class header correct
type tParkingManager = class
    //Q4.2 - 4
  //fields private
  //array of 50 Daily
  //type parent class
  //size field correct type
  private
    pArr : array[1..50] of tDaily;
    size : integer;

  public
    constructor Create();
    function toString : string;
    procedure SortByRegistration();
    function getRegistrationList() : string;
    function generateBilling(): string;
end;

implementation

{ tParkingManager }

//Q4.3 - 11
//constructor

constructor tParkingManager.Create;
var
  inFile : textfile;
  line, reg, entSt, exSt, bay , date , d , m , y , h , mi,d2 , m2
, y2 , h2 , mi2  : string;
  entryDT, exitDT : TDateTime;
begin
//open and loop through file
  if FileExists('parkings.txt') <> true then
    begin
      WriteLn('File Missing');
    end
  else
    begin
      AssignFile(inFile, 'parkings.txt');
      Reset(inFile);

      size:=0;

      while NOT EOF(inFile) do
        begin
```

```
         //read line  from text file
           ReadLn(inFile, line);
        //increment size
           Inc(size);

           //extract first 3 fields
           //use correct DateTime format
           reg := Copy(line , 1 , Pos(';', line) - 1);
           Delete(line , 1 , Pos(';',line));

           entST := Copy(line , 1 , Pos(';', line) - 1);
           Delete(line , 1 , Pos(';',line));

           exST := line;


           y := Copy(entSt,1 , Pos('/', entSt) - 1);
           Delete(entSt,1 , Pos('/', entSt));

           m := Copy(entSt,1 , Pos('/', entSt) - 1);
           Delete(entSt,1 , Pos('/', entSt));

           d := Copy(entSt,1 , Pos(' ', entSt) - 1);
           Delete(entSt,1 , Pos(' ', entSt));

           h := Copy(entSt,1 , Pos(':', entSt) - 1);
           Delete(entSt,1 , Pos(':', entSt));
           mi := entSt;
            //create DateTime objects
           entryDT := EncodeDateTime(StrToInt(y), StrToInt(m) ,
StrToInt(d) , StrToInt(h) , StrToInt(m), 0 , 0);


           y := Copy(exST ,1 , Pos('/', exST ) - 1);
           Delete(exST ,1 , Pos('/', exST ));

           m := Copy(exST ,1 , Pos('/', exST ) - 1);
           Delete(exST ,1 , Pos('/', exST ));

           d := Copy(exST ,1 , Pos(' ', exST ) - 1);
           Delete(exST ,1 , Pos(' ', exST ));

           h := Copy(exST ,1 , Pos(':', exST ) - 1);
           Delete(exST ,1 , Pos(':', exST ));
           mi := exST;


           exitDT := EncodeDateTime(StrToInt(y), StrToInt(m) ,
StrToInt(d) , StrToInt(h) , StrToInt(m), 0 , 0);
             //check for long term parking bay
           if Pos(';', line) > 0 then
           begin
```

```
            Delete(line , 1 , Pos(';',line));
            //get parking bay
            bay := line;
             //create LongTermParking object and add to array
            pArr[size] := TLongTerm.Create(reg,entryDT, exitDT,
bay);
          end
          else
          begin
            //create Daily object and add to array
            pArr[size] := TDaily.Create(reg,entryDT, exitDT);
          end;




        end;
    end;
end;
//Q4.4 - 4
//header correct and return string

function tParkingManager.toString: string;
var
  i : integer;
  output : string;
begin
   output := '';
   //loop through array
  for i := 1 to size do
    begin
      //appending to string
      //add a blank line
      output := output + pArr[i].toString() + #13#10 + #13#10;
    end;

  Result := output;
end;

//Q4.5 - 6
procedure tParkingManager.SortByRegistration;
var
  i , j : integer;
  temp : TDaily;
begin
  //outside for loop correct
  for i := 1 to size do
  begin
    //inside for loop correct
    for j := 1 to size-1 do
      begin
        //compare the correct elements
        //using registration number
```

```
        if CompareStr(pArr[j].getRegistrationPlate() ,
pArr[j+1].getRegistrationPlate()) > 0 then
        begin
             //store into temp with type correct for array
             temp := pArr[j];
             //swap elements
             pArr[j] := pArr[j+1];
             pArr[j+1] := temp;
        end;

      end;
  end;
end;

//Q6.1 - 6
//correct method header and return
function tParkingManager.getRegistrationList: string;
var
  i : integer;
  temp : string;
begin
     temp := '';
       //loop through array
     for i := 1 to size do
       begin
        //check if car registration
        //is already in temp
         if NOT( temp.Contains(pArr[i].getRegistrationPlate) )
then
         begin
          //add registration to list
          //add a # character
           temp := temp + pArr[i].getRegistrationPlate + '#';
         end;
       end;

     Result:=temp;
end;
//Q6.2 - 1
//method header correct and return
function tParkingManager.generateBilling: string;
var
  cars, bay, output , currentReg : string;
  temp : TLongTerm;
  total : real;
  i : integer;
  Daily : boolean;

begin
  //Q6.2.1 - 1
    //call getRegistrationList and assign to cars
    cars := getRegistrationList();
    bay := '';
    output := '';
```

```
        //isolate registration numbers - 3
        //open scanner

        //check for more registration numbers
    while cars.Length > 0 do
    begin
    //extract registration number
      currentReg :=  Copy(cars , 1 , Pos('#', cars) - 1);
      Delete(cars , 1 , Pos('#',cars));
      WriteLn(currentReg);
      WriteLn(currentReg.Length);
        //create a string - 3*
       //append the registration number before the entries
      output := output + currentReg  + #13#10;

       //search and add to total - 5
      //initialise total inside the loop
      total := 0;
      Daily := true;
      //loop through array
      for i := 1 to size do
      begin
            //check if car registration already in list
          if  ( CompareStr(pArr[i].getRegistrationPlate() ,
                          currentReg) = 0 ) then
          begin
              //add parking fee
              //to total
              total := total + pArr[i].getParkingFee();

              // append each entry details to output
              output := output  +
DateTimeToStr(pArr[i].getEntryDateTime()) + ' ' +
DateTimeToStr(pArr[i].getExitDateTime()) + ' ' +
FloatToStr(pArr[i].getParkingFee()) + #13#10;

               //check if long-term parking bay - 4
               //check if pArr[i]
               //is a Long-Term object
              if pArr[i]  is TLongTerm then
              begin
                Daily := false;
                //cast pArr[i] to a LongTerm object
                temp := pArr[i] as TLongTerm;
                //record parking bay number
                bay := temp.getParkingBay();
              end
              else
              begin
                Daily := true;
                bay := '';
              end;
```

```
          end;
      end;
        //append the parking fee
      output := output + 'Total parking fees due ' +
FloatToStr(total)  + #13#10;
       //add bay number - 2
       //check if the registration is a LongTerm object
      if NOT(Daily) then
      begin
       //append the bay number
          output :=  output  + 'Long Term Bay: ' + bay  + #13#10 +
#13#10
      end
      else
      //add message - 2
      //check if the total is above 1500
        if (total > 1500 ) AND (Daily = true) then
        begin
        //append the message
          output := output +  'Consider renting a long term
parking' + #13#10 + #13#10;
        end;
     Result:= output;




    end;


end;




end.
```

QUESTION 5, 6.3 ParkingUI.pas

```
//Q5.1 - 1
//class header correct
program ParkingUI;

{$APPTYPE CONSOLE}

{$R *.res}

uses
  System.SysUtils,
  DateUtils,
  uDaily in 'uDaily.pas',
  uLongTerm in 'uLongTerm.pas',
  uParkingManager in 'uParkingManager.pas';

var
  pm : tParkingManager;
begin
  try
    { TODO -oUser -cConsole Main : Insert code here }
    //Q5.2 - 2
    //ParkingTagManager object created
    //in correct place
    pm := tParkingManager.Create();
    //Q5.3 - 2
    //sort method called
    pm.SortByRegistration();
     //toString method called
    WriteLn(pm.toString());

   //Q6.3 - 2
   //correct method name
   //in an output statement (typed method call)
    WriteLn(pm.generateBilling());



    ReadLn;
  except
    on E: Exception do
      Writeln(E.ClassName, ': ', E.Message);
  end;
end.
```