WESTERN CAPE EDUCATION DEPARTMENT

SEPTEMBER EXAMINATION – 2014

INFORMATION TECHNOLOGY

PAPER 1 – PRACTICAL

M E M O R A N D U M

Examiner: Max Brock

TIME: 3 HOURS MARKS: 150

This Memo consists of 17 pages.

INSTRUCTIONS and INFORMATION:

1. THIS IS A THREE-HOUR EXAMINATION. BECAUSE OF THE NATURE OF THIS EXAMINATION IT IS IMPORTANT TO NOTE THAT YOU WILL **NOT** BE ALLOWED TO LEAVE THE EXAMINATION ROOM BEFORE THE END OF THE EXAMINATION SESSION.
2. Log on and open your exam folder. You require the files listed below in order to answer the questions.

   Question 1: NetBeans project **SpeedingFineApp** – form **AverageSpeedProsecution**

   Question 2: NetBeans project **SpeedingFineApp** - form **ProcessingFines**, class **SpeedingFine**, and the **SpeedingOffences.txt** text file

   Question 3: NetBeans project **SpeedingFineApp** – form **MultipleFines**

   Question 4: NetBeans project **SpeedingFineApp** – form **Statistics**
3. The mark allocation for the 4 questions is indicated below. Use the mark allocation to plan your time wisely.

   Q1 – (33) Q2 – (56) Q3 – (29) Q4 – (32)
4. Save ALL your work in your exam folder under the required names.
   NB: Only programs saved in your exam folder will be marked!
5. Save your work regularly as a precaution against power failures.
6. You may make use of the Java's API files. You may NOT consult any other resource material.
7. Type your name and the question number as a comment in the first line of each program, e.g. // Joe Soap, Q. 2.
8. **Attempt ALL questions**.
9. Read the requirements for each program thoroughly and understand the details before starting to program. Do ONLY what is required by each question.
10. In ALL programs marks will be allocated to functional parts, even if the whole program does not work (or hasn't been completed). So make sure that you complete each step before starting the next one.
11. Good luck!

SCENARIO:

The Traffic Department of the Western Cape has commissioned an app to process various aspects of speeding and other traffic fines.

There are two types of speed measurement:

- **Average speed measurement**

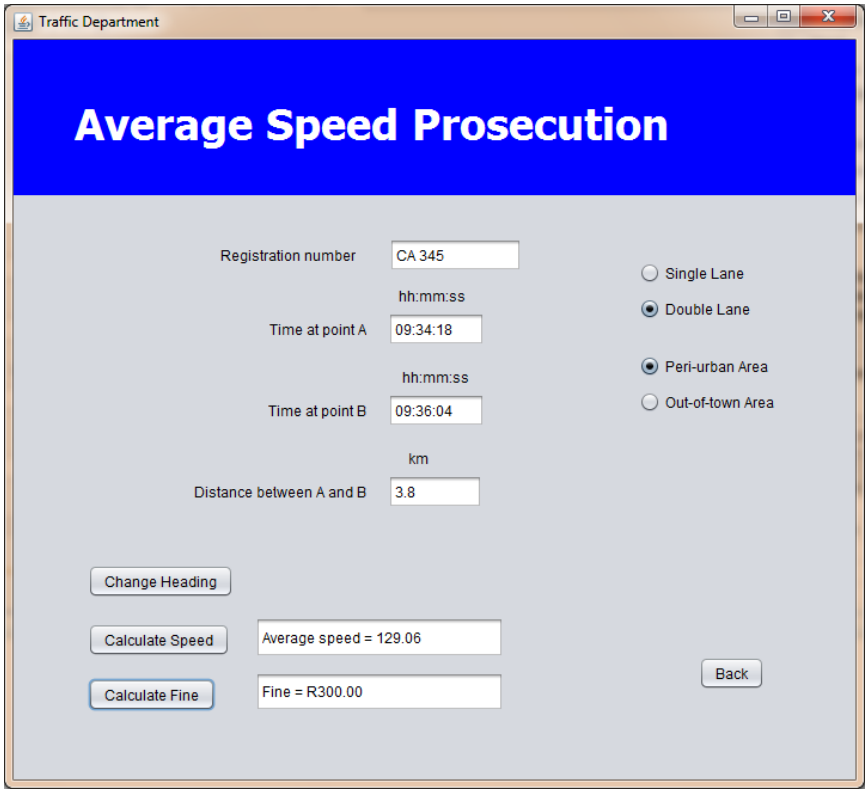
  A computer system reads vehicle registration plates and times at two or more fixed points along a road, usually hundreds of meters or even kilometres apart, then uses the known distance between them to calculate a vehicle's average speed.
- **Instantaneous speed measurement (the conventional "speed trap")**

  Instantaneous speed cameras measure the speed at a single point. These may either be a semi-permanent fixture or be established on a temporary basis. Usually radar speed guns utilize the time of flight of laser pulses to make a series of timestamped measurements of a vehicle's distance from the laser; the data is then used to calculate the vehicle's speed.

You have been tasked to complete the programming for the various tasks.

Question 1: Average Speed Prosecution – Java Programming

1. Please complete the first app where the average speed and possible fines according to **Average Speed Prosecution** are calculated.



1.1 When you open the form/frame you will see that the heading is still "Heading" and the colour of the font is black while the colour of the upper panel is standard grey.

Please add code for the **Change Heading** button so that when it is pressed the following changes occur:

- The panel's colour changes to blue.
- The heading's colour changes to white.
- The text of the heading changes to "Average Speed Prosecution".

NB: You will get NO marks if you merely make these changes in the Properties sheet of the components. (3)

```
lblHeading.setText("Average Speed Prosecution");   ✓
lblHeading.setForeground(Color.white);             ✓
pnlHeading.setBackground(Color.blue);              ✓
```

1.2 When the **Calculate Speed** button is pressed, the average speed that the car is travelling at between points A and B must be calculated. The times recorded at points A and B are times of day (as on a watch). Please add code to for the Calculate Speed button to read the two times and the distance as entered by the user, and then calculate the average speed of the car.

Average speed = distance in km/(Time at B – Time at A) in hours

(**HINT**: Convert everything to seconds before subtraction, and then into a decimal fraction of hours before division. There are 3600 seconds in 1 hour).

When the data in the screenshot above is used the time difference expressed in hours is 0.029444444444444443.

Display the calculated average speed in the text field next to the "Calculate Speed" button rounded to 2 decimal places. (15)

```
String sTime1 = txfTime1.getText(); // in format hh:mm:ss
String[] time1 = sTime1.split(":");
int hr1 = Integer.parseInt(time1[0]);
int min1 = Integer.parseInt(time1[1]);
int sec1 = Integer.parseInt(time1[2]);
int secT1 = (hr1 * 60 + min1) * 60 + sec1;
String sTime2 = txfTime2.getText(); // in format hh:mm:ss
String[] time2 = sTime2.split(":");
int hr2 = Integer.parseInt(time2[0]);
int min2 = Integer.parseInt(time2[1]);
int sec2 = Integer.parseInt(time2[2]);
int secT2 = (hr2 * 60 + min2) * 60 + sec2;

int secDiff = secT2 - secT1;
```

reading time from correct text field✓

extracting the hr, min and sec by ANY method✓✓✓ (Scanner, split, substring, other)

converting to seconds✓✓✓

ONE mark for repeat of methods✓ [Mark this one if previous is wrong and this is more correct]

subtract times (seconds)✓

Alternatively a more complicated method of subtracting with carry-overs (60!) can be used for the bracketed code (any correct method/result should get all 6 marks PLUS one for the repeat PLUS 1 for getting the difference in times)

```
double hrDiff = secDiff / 3600.0;
double distance = Double.parseDouble(txfDistance.getText()); // in km
aveSpeed = distance / hrDiff; // km/h
txfSpeed.setText("Average speed = " + String.format("%.2f", aveSpeed));
```

convert seconds to a decimal fraction of hr✓

read in km and parse✓✓

divide✓

formatting✓ and display in correct text area✓

1.3 When the **Calculate Fine** button is pressed, the fine must be calculated if the speed is above the speed limit. Please add code for the Calculate Fine button to calculate the fine for the driver. As Average Speed Prosecutions can only take place on open straight roads, it does not lend itself to denser urban areas with stops and/or robots.

For the applicable speed limit the type of road and the type of area are important:

Criterion 1: Number of lanes of the road: Double lane or Single lane

Criterion 2: Where the speeds were taken: in an Urban area or an Out-of-town area

Lane/area codes:		
	SU	Single lane through Urban area
	SO	Single lane through Out-of-town area
	DU	Double lane through Urban area
	DO	Double lane through Out-of-town area

The speed limits are: SU = 80, SO = 120, DU = 100, DO = 120

Fines are R100 per 10 km above the speed limit. A speed is always rounded up to the nearest ten km.

In the example above the car drove at an average speed of 129km/h in a 100 km/h area (Double lane Urban), rounded up that is 30km/h too fast. Therefore the fine is calculated to R300.00.

This fine has to be displayed in the text field next to the "Calculate Fine" button using currency formatting ("R" in front and rounded to 2 decimals). (15)

```
String code = "";
if (rbtSingleLane.isSelected()) {
   code += 'S';  }
 else {  // or  if (rbtDoubleLane.isSelected())
   code += 'D';  }
if (rbtUrban.isSelected()) {                          build up code from radio buttons✓✓✓✓
   code += 'U';  }
else {  //  or  if (rbtOutOfTown.isSelected())
   code += 'O';
}
int speedLimit = 0;
switch(code)   {
        case "SU": speedLimit = 80; break;
        case "SO": speedLimit = 120; break;           set speed limit according to code✓✓✓✓
        case "DU": speedLimit = 100; break;           [can also use if or if..else – ANY method that
        case "DO": speedLimit = 120; break;                                  works correctly]
     }
int overSpeedLimit = (int) (aveSpeed - speedLimit);   calculate how much over limit✓ cast to int✓
int multiplier = overSpeedLimit / 10 + 1;             calculate "multiplier" rounded up to 10✓✓
double fine = multiplier * 100;                       calculate fine by multiplication✓
txfFine.setText(String.format("%s R%.2f", "Fine =", fine));      format✓ and display✓
```

Total Question 1 = [33]

Question 2: Speeding Fines Processing – Object-oriented Programming

The second app is designed to process all the speeding fines as recorded in a text file. The "speed trapping" was done using the instantaneous speed measurement method. In order to assist in the process you must complete and use the **SpeedingFine** class.

2.1 In the NetBeans project you have been provided with a class named **SpeedingFine**.

This class has the following attributes:

```
String registrationNumber;
char laneCode;
char areaCode;
char roadCode;
int speed;
```

The accessor methods (GETs) as well as the isFinable method have also been provided. Please add the following methods to the **SpeedingFine** class:

2.1.1 Write a ***parameterised constructor*** for the all 5 attributes. The parameters should be used to initialise the fields of the class. (1)

```
public SpeedingFine(String registrationNumber, char laneCode, char areaCode, char roadCode,
                                                                        int speed)
    this.registrationNumber = registrationNumber;
    this.laneCode = laneCode;
    this.areaCode = areaCode;
    this.roadCode = roadCode;
    this.speed = speed;
```

Can be done with NetBean's wizard, ONE mark only✓

2.1.2 The speed limit for instantaneous speed measurement is determined by a more complex formula than for average speed prosecution, and there are more factors to take into consideration:

The lane codes are: D = double lane, S = single lane

The area codes are: U = urban, P = peri-urban (i.e. on the urban fringe), R = residential, O = out-of-town,

Road code (not always mentioned): N = narrow and winding, no clear views ahead

Limits are: R = always 60, SU = 60, SP = 80, SO = 120, DU = always 80, DP = always 100, DO = always 120, SON = 100, SPN = 70

Write code for the **getSpeedLimit-**method that determines and returns the speed limit according to the 3 possible codes: the lane code, the area code, and the (optional) road code.

If one or both of the first two codes (lane and area codes) are wrong, e.g. a wrong letter in a wrong category, like an 'E' or 'F', then the speed limit must be returned as an impossible 1000.

But as the third code (the road code) is optional, any character other than 'N' must just be ignored. (10)

```
public int getSpeedLimit() {
      switch (areaCode) {
         case 'R':..return 60;
         case 'U': {
            if (laneCode == 'S') {
               return 60;      }
            if (laneCode == 'D') {
               return 80;      }
         }
         case 'P': {
            if (laneCode == 'S' && roadCode ==
'N') {
               return 70;      }
            if (laneCode == 'S') {
               return 80;      }
            if (laneCode == 'D') {
               return 100;      }
         }
         case 'O': {
            if (laneCode == 'S' && roadCode ==
'N') {
               return 100;     } // or one else
            if (laneCode == 'S') {
               return 120;      }
            if (laneCode == 'D') {
               return 120;      }
         }     }
      return 1000;    }
```

The whole speed limit determination can of course be done with just if or if..else.

Learners can use either an int variable for the speed limit and return that at the end, or return several times as in the example on the right.

Either characters of the different codes can be used separately (as in the sample code on the right, or they can be joined into Strings and evaluated with the .equals method.
E.g. if (code.equals("DO")
speedLimit = 120;

As there are 9 different code combinations to evaluate plus the default, the question will count 10 marks (9 plus 1 for else = 1000).

Look out for any logic errors! The 'M' must be reachable.

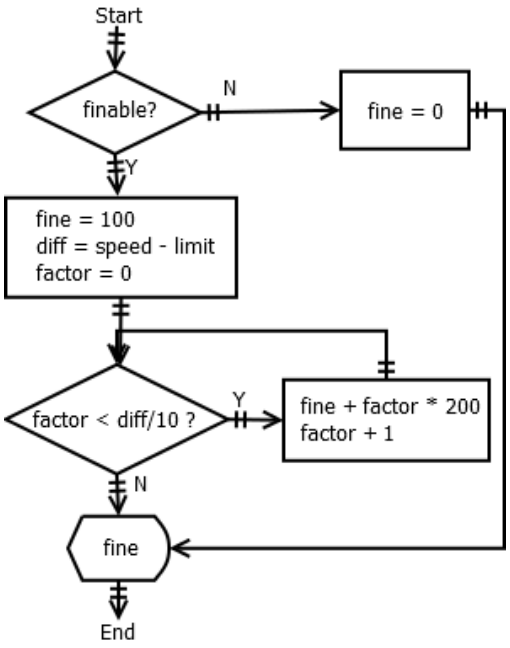
R = always 60✓
SU = 60✓
SP = 80✓
SO = 120✓
DU = always 80✓
DP = always 100✓
DO = always 120✓
SON = 100✓
SPN = 70✓
Else = 1000✓

2.1.3 The fines for instantaneous speed measurement increase steeply:

Speed over limit	≥10	≥20	≥30	≥40	≥50	≥60	≥70	≥80	Etc.
Increase		+200	+400	+600	+800	+1000	+1200	+1400	Etc.
Fine	100	300	700	1300	2100	3100	4300	5700	Etc.

Write code for the **getFine-**method that

1. tests if the driver is finable and, if so,
2. calculates and returns the fine using the fine escalation information above and the flowchart on the right,
3. and if not, returns a zero fine.



(9)

```
if (!isFinable()) {                                  use of method✓
   return 0.0;                                       return 0✓
}
double fine = 100.0;
double speedDifference = speed - getSpeedLimit();    calculate difference in speed✓
int fine = 100;
int factor = 0;                                      calculate "factor"✓
while (factor < diff / 10) {        //or for-loop    calculating fine with loop as per
    fine = fine + (factor * 200);                    flowchart (4 marks)✓✓✓✓
    factor++;                                        [only award 2 marks if fine is deter-
    }                                                mined by if..else or switch..case]
return fine;   }                                     returning correct value✓
```

2.1.4 Complete the ***toString***-method that displays the basic information for a SpeedingFine object as follows (here with data from a test record):

Registration number <spaces> area code <spaces> speed <spaces> fine

e.g. `CA 123 456  U  81  300.00`

OR if not finable

Registration number <lot of paces> "No fine"

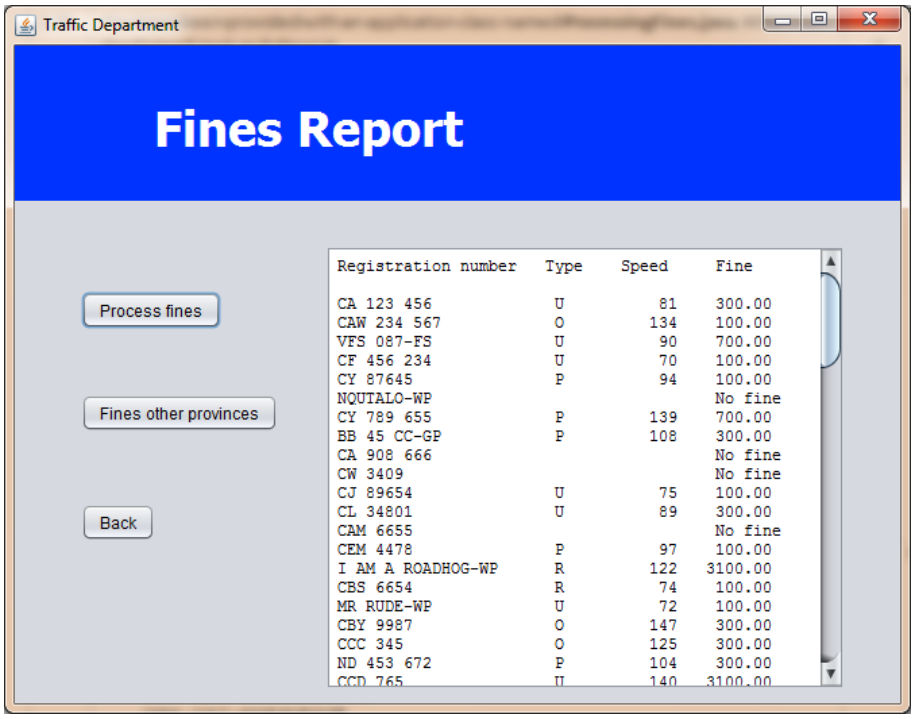
e.g. `CA 908 666  No fine`

NB: The fine must be rounded off to two decimal places. (6)

```
if (isFinable()) {                                   test finable✓
   return String.format("%-20s%4s%12.2f%10.2f%n",    formatting (or tab)✓
      registrationNumber, areaCode, getSpeed(), getFine());   4 elements✓
  }                                                  format fine R and 2 decs✓
   return String.format("%-20s%27s%n",               else or 2nd return✓
      registrationNumber,"No
fine");
                                                     Reg num and "No fine"✓
```

Subtotal 2.1 [26]

2.2 You have been provided with an application class named **ProcessingFines.java**. When run the GUI will look as follows:



2.2.1 Provide code for the ***Process Fines*** button to read the data from the text file **SpeedingOffences.txt**, use the SpeedingFine object to calculate the fine and display the information in columns in the text area. The first 3 records from the text file look as follows:

```
CA 123 456#SUV#81
CAW 234 567#DO#134
VFS 087-FS#SU#90
```

Each record is made up as follows:

Registration number, #, codes (2 or 3 characters!), #, speed

Do the following:

- Test whether the text file exists
- If not, display a message
- If yes, then create a file-reading object
- Display the headings in the text area
- Read a line at a time
    - Separate the data
    - Use the data to instantiate a SpeedingFine object
    - Use the toString method of the object to display the information

In the end the GUI will look like in the screenshot above. (20)

```
txaOutput.setText(String.format("%-22s%-8s%-8s%6s%n%n", "Registration number", "Type",
"Speed", "Fine"));                                    display heading✓
File f = new File("SpeedingOffences.txt");           declaring File✓
if(f.exists())  {                                    test if file exists (or try)✓
    try {
     Scanner fileReader = new Scanner(f);            declare file reading object✓
     while (fileReader.hasNext())    {               while loop to read file✓
        String line = fileReader.nextLine();         read one line✓
        Scanner lineBreaker = new Scanner(line).useDelimiter("#");
                                                declare line breaker object or use split(“#”)✓
        String registrationNumber = lineBreaker.next();        assign first string✓
        String codes = lineBreaker.next();
        char laneType = codes.charAt(0);             assign string and divide into chars✓✓
        char areaType = codes.charAt(1);
        char roadType = ' ';                         declase a default char for road type✓
                                                            (can be any char NOT used)
        if (codes.length()>2)  {                             test whether there is a third char✓
           roadType=  codes.charAt(2);      }                assign third char to roadType✓
        int speed = lineBreaker.nextInt();                   assign int✓
        SpeedingFine sf                                      declare SpeedingFine object✓
            = new SpeedingFine(registrationNumber, laneType,
                    areaType, roadType, speed);              with correct parameters✓
        txaOutput.append(sf.toString());                     append to txa✓ using toString✓
        txaOutput.setCaretPosition(0);                       (optional)
        } }    }
    catch (Exception e) {
        JOptionPane.showMessageDialog(rootPane, "File reading error");
        e.printStackTrace();
    }      }
else {
        JOptionPane.showMessageDialog(rootPane, "File not found");    error message✓
}   }                                                    getting column formatting (almost) right✓
```

2.2.2 Code the **Fines other provinces** button to determine the number of fines and the total value of the fines from cars from other provinces. Make use of the 2 class variables declared at the top of the class: **counterFinesOtherProvinces** and **totalFinesOtherProvinces**. In the PREVIOUS actionPerformed method (***btnProcessFinesActionPerformed()***) add code to

- Initialise the 2 variables
- Inside the while loop, while you have the registration number handy, check whether it is from inside the Western Cape or not (**Hint**: WC numbers either start with a “C…” or end with “-WP”). If it is NOT a WC number then increase the counter and add the fine to the total.
- In the ***btnFinesFromOtherProvincesActionPerformed()*** method you only have to clear the text area and display the details of other provinces by making use of the attributes. (10)

In btnProcessFinesActionPerformed() at the beginning:

```
counterFinesOtherProvinces = 0;
totalFinesOtherProvinces = 0.0;                    initialising both attributes✓
```

In btnProcessFinesActionPerformed() in the while loop AFTER instantiating the SpeedingFine object:

```
if !(sf.getRegistrationNumber().startsWith("C")                    test NOT starts with 'C'✓
    || sf.getRegistrationNumber().endsWith("WP"))                  test NOT ends with 'WP'✓
   {                                                               correct logic✓
                                    [Either !(cond1 || cond2)   OR   !(cond1) &&
!(cond2)]
 // OR (sf.getRegistrationNumber().charAt(0) != 'C' &&
            sf.getRegistrationNumber().charAt(sf.getRegistrationNumber().length()-2) !='W' )
      counterFinesOtherProvinces++;                                incrementing counter✓
      totalFinesOtherProvinces += sf.getFine();                    adding fine to total✓
    }
```

private void btnFinesFromOtherProvincesActionPerformed(java.awt.event.ActionEvent evt) {

```
txaOutput.setText("Fines incurred by cars from other provinces\n");        display heading✓
txaOutput.append( "Number = " + counterFinesOtherProvinces + "\n");        display counter✓
txaOutput.append ("Total fines = " + String.format("R%.2f%n", totalFinesOtherProvinces));
                                                  display total✓ and format fine to currency✓
}
```

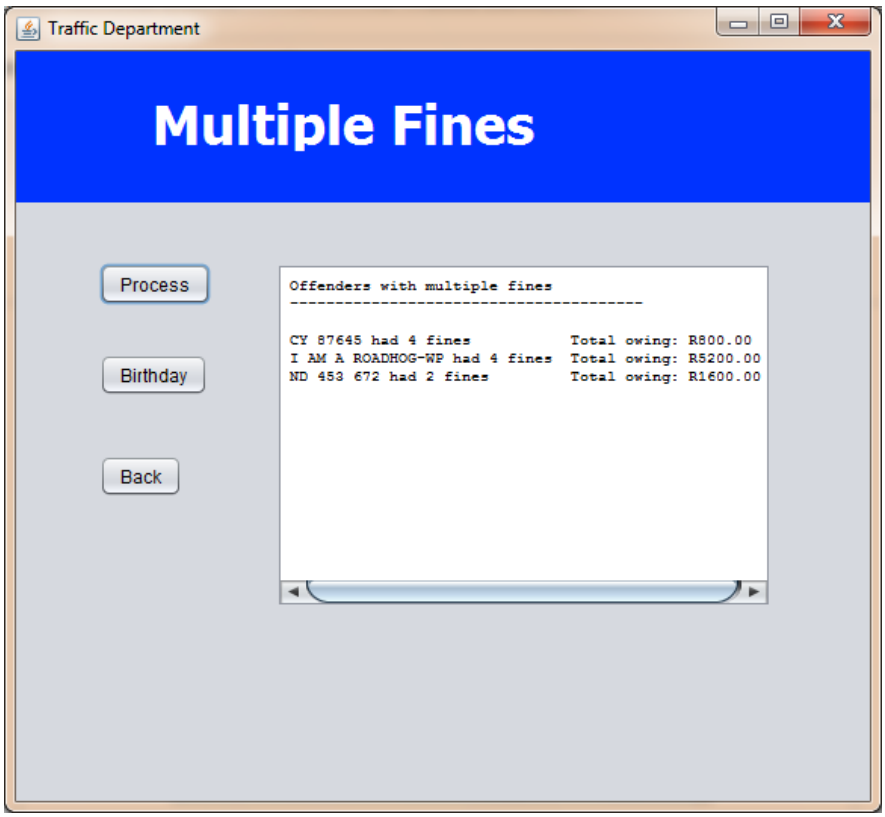
Subtotal 2.2 [30]

Total Question 2 = [56]

Question 3: Multiple Speeding Fine Processing – Problem Solving

The third app, **MultipleFines**, is designed to interact with two arrays recording the speeding fines. The first array ("arrRegNums") is an array of car's registration numbers and the second array ("arrFines") contains the fines that these cars recorded in the first array have incurred. All data has been declared for you.

The provincial traffic department is very concerned that some drivers have incurred speeding fines again and again. They want to trace these drivers and take them to court.



3.1 When the **Process** button is pressed, the data in the arrays is searched for duplicates and the total owing is added. Add code to produce the output pictured above.

Don't worry if at first your output looks like this:

```
CY 87645 had 4 fines           Total owing: R800.00
I AM A ROADHOG-WP had 4 fines  Total owing: R5200.00
ND 453 672 had 2 fines         Total owing: R1600.00
CY 87645 had 3 fines           Total owing: R700.00
I AM A ROADHOG-WP had 3 fines  Total owing: R2100.00
I AM A ROADHOG-WP had 2 fines  Total owing: R1400.00
CY 87645 had 2 fines           Total owing: R600.00
```

The trick is creating a data structure to record or store which cars have already been found with multiple speeding fines. These structures could be – amongst others – a list, an array, a text file, or an array of boolean. You can then incorporate that into your test and avoid listing a car more than once. (16)

```
boolean[] multiple = new boolean[max];          declaring and initialising Boolean
                                                array (or any other data structure
for (int i = 0; i < multiple.length; i++) {     to assist with tracing the multiple offenders.
   multiple[i] = false;                         There are even solutions of replacing the
}                                               numbers with "X" and no extra data structure
                                                has to be created – test it and give the 2
                                                marks if correct) ✓✓
txaOutput.setText("Offenders with multiple fines"
          + "\n---------------------------------------\n\n");      display heading✓
for (int i = 0; i < max - 1; i++) {             loop through all entries✓
   int count = 1;                               initialise a counter✓  (to keep track
                                                of duplicates (can be replaced with
                                                any operation to keep track of
                                                multiple offenders)
   double total = arrFines[i];                  initialise total✓
   for (int j = i + 1; j < max; j++) {          nested loop starting at i + 1✓
      if (arrRegNums[i].equals(arrRegNums[j])   testing whether regnum is equal✓
            && multiple[j] == false) {          && (AND) logic✓
                                        ANY test whether they have been processed already✓
         count++;                               increment counter✓
         total += arrFines[j];                  add fine to total✓
         multiple[j] = true;                    set Boolean to true to indicate that
      }                                         this record has been processed or
   }                                            add to the data structure declared
                                                above✓

   if (count > 1) {                             test count > 1 indicating multiple fines✓ (or
                                                check whether in data structure of duplicates)
      String out = String.format("%s%s%d%s", arrRegNums[i], " had ", count, " fines");
      txaOutput.append (String.format ("%-31s%sR%.2f%n",out, "Total owing: ", total));
                                                display information reasonably formatted
                                                      (columns NOT required✓✓
}    }
```

3.2 The Traffic Chief's birthday is this week. To mark this occasion he wants to cancel ONE driver's fine. This is activated when the **Birthday** button is pressed.

Add code to the ***Birthday***-button to randomly select ONE registration number which has a fine of only R100.00 to be let off provided it is not one of the multiple offenders displayed in 3.1. If the random choice falls on a car with a higher fine or a repeat offender then the program must randomise another number until one with a R100.00 fine is found.

At the bottom of the frame a panel has been provided (called **pnlBirthdayMessage**) but nothing further. You must instantiate a new label and add it to the panel. Use setBounds(30, 20, 500, 30) to place the button on the panel. The message should be "Happy Birthday Chief! <SOMEONE> will not have to pay his fine." where <SOMEONE> is the registration number of one of the cars in the array that the program has randomly selected.

You may add colour to the label and change the font. In the screenshot below the "Papyrus" font was chosen, the label was painted red and the writing white. You can only get ONE mark for any ONE feature, so don't waste any time on it.

Finally, you must disable the Birthday button, otherwise the user could press it several times until a registration number of his/her choice would come up. (13)


```
Random random = new Random();
int ranNum = 0;
do {
   ranNum = random.nextInt(max);
} while (arrFines[ranNum] != 100.0
      || multiple[ranNum] == true);
```

ANY random method used correctly✓
initialise number outside loop✓
do..while loop to repeat✓
randomise between 0 and 81✓
fine must be R100.00 (not more)✓
must not be one of the multiple offenders✓

```
JLabel message = new JLabel();
pnlBirthdayMessage.add(message);
message.setBounds(30, 10, 500, 30);
message.setText("  Happy Birthday Chief!   "
      +arrRegNums[ranNum]
      +" will not have to pay his fine!");
message.setFont(new Font("Papyrus", 3, 12));
message.setBackground(Color.red);
message.setForeground(Color.white);
message.setOpaque(true);
btnBirthday.setEnabled(false);
```

dynamic instantiation of label✓
add label to panel✓
setBounds as instructed✓
message text as instructed✓
registration number as randomised added into message✓
any ONE formatting feature✓
disabling Birthday button✓

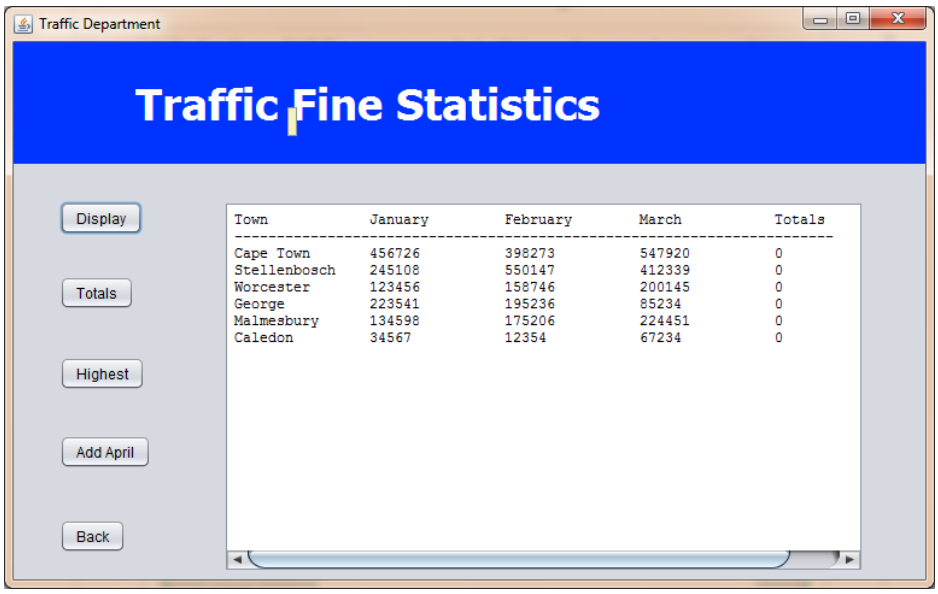
Total Question 3 = [29]

Question 4: Traffic Fine Statistics – Problem-Solving

In the fourth app, **Statistics**, you are provided with 3 one-dimensional arrays as well as one two-dimensional array. The two-dimensional array contains traffic fine totals for the 3 months January to March as reported to the provincial traffic department by 6 of the district offices. The two-dimensional array of values will thus be 6 x 3 (see below).

The other arrays contain the headings, the names of the district offices, and the last array is an empty array to be used for the totals for each district office.

4.1 When the Display button is pressed the statistics must display the statistics neatly in columns. Add code to the ***Display***-button to display the information as in the screen shot below.



As you can see, the totals for each district office have not been calculated yet. The Display button acts as a refresher for the screen. So, when the totals have been calculated, the Display button must be pressed again. (12)

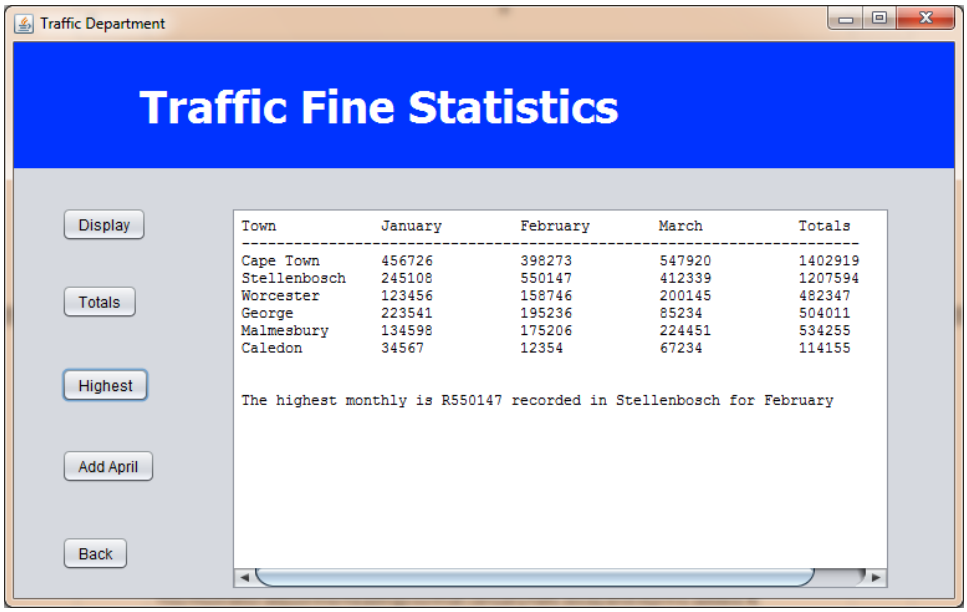
```
txaOutput.setText(""); // clear                                  clear txa✓
    for (int i = 0; i < headings.length; i++) {                  loop to display headings✓
      txaOutput.append(headings[i] + "\t");                      headings appended with \t (or other)✓
      if (headings[i].length() < 8) {
        txaOutput.append("\t");                                  extra spaces for small headings✓
      }    }
    txaOutput.append("\n----------------------------------------------------------------------\n");    line✓
    for (int i = 0; i < stats.length; i++) {                     outer loop to 6✓
      txaOutput.append(towns[i] + "\t");                         append towns' names✓
      if (towns[i].length() < 8) {
        txaOutput.append("\t");     }                            extra spaces for small towns✓
      for (int j = 0; j < stats[i].length; j++) {                inner loop to 3✓
        txaOutput.append(stats[i][j] + "\t\t");     }            append numbers✓
      txaOutput.append(totals[i] + "\n");    }                   append totals✓ plus new line✓
```

4.2 Add code to the ***Totals***-button: You must use the two-dimensional array to calculate the totals for each district office and store the totals in the totals[] array.

The user will have to press the Display button again to make the totals visible. (5)

```
for (int i = 0; i < totals.length; i++) {
      totals[i] = 0;      }                          initialise totals to 0✓
   for (int i = 0; i < stats.length; i++) {          outer loop to 6✓
     for (int j = 0; j < stats[i].length; j++) {     inner loop to 3✓
       totals[i] += stats[i][j];      }      }       add correct number✓ to correct total✓
```

4.3 Add code to the ***Highest***-button to determine the highest figure in the two-dimensional array and append it to the bottom of the statistics. State BOTH for which town and for which month this highest figure applies. (8)




```
int xIndexHighest = 0;                                initialise variables to store highest,
int yIndexHighest = 0;                                town, month or use indexes✓✓
for (int i = 0; i < stats.length; i++) {              outer loop to 6✓
   for (int j = 0; j < stats[i].length; j++) {        inner loop to 3✓
     if (stats[i][j] > stats[xIndexHighest][yIndexHighest]) {  test highest against current value✓
       xIndexHighest = i;                             if higher, store details✓
       yIndexHighest = j;
     }       }     }
txaOutput.append("\n\nThe highest monthly is R"
     + stats[xIndexHighest][yIndexHighest]            build up string with value, town and
     + " recorded in " + towns[xIndexHighest]         month✓ and append to txa✓
     + " for "+ headings[yIndexHighest+1]);
```

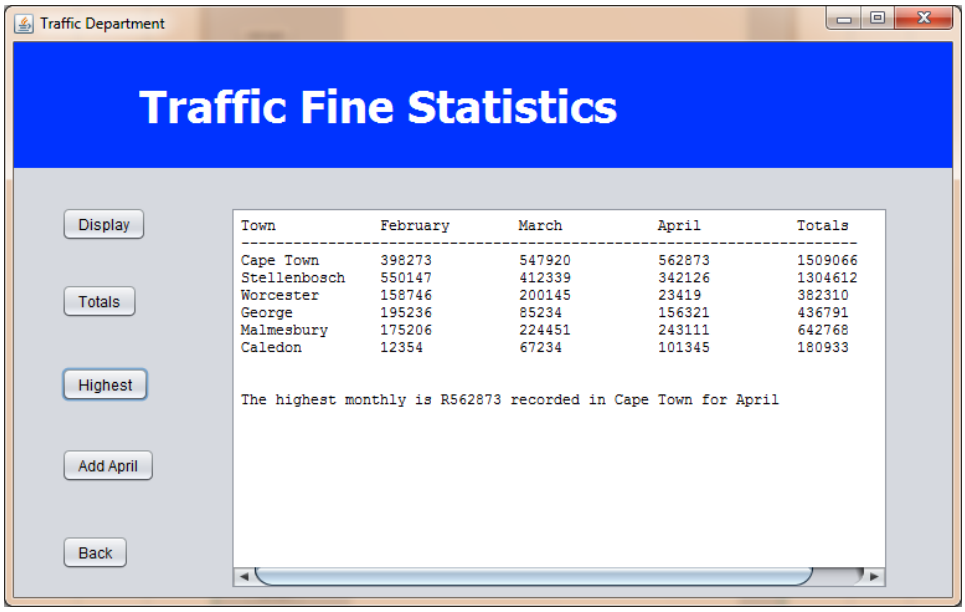
4.4 The figures for April have just come in. They are

April = 562873, 342126, 23419, 156321, 243111, 101345 for the 6 towns respectively.

Add code to the ***AddApril***-button: You must add the April figures to the statistics. As the statistics ONLY contain figures for the LAST THREE months, January's figures must fall away.

You must also adjust the headings so that January falls away and April is added.

When the totals have been recalculated and the display has been refreshed, the information displayed in the text area looks as follows: (7)



Note that when you press the Highest button now, a different value is the highest, with a different town and month.

```
int[] april = {562873, 342126, 23419,
                156321, 243111, 101345};      new array for April's figures✓✓
headings[1] = headings[2];
headings[2] = headings[3];                    move every month up one position✓
headings[3] = "April";                        add "April"✓
for (int i = 0; i < stats.length; i++) {      loop to 6✓
   stats[i][0] = stats[i][1];
   stats[i][1] = stats[i][2];                 move every figure up one position✓
   stats[i][2] = april[i];       }            add April's figure✓
```

Total Question 4 = [32]

GRAND TOTAL = 150