basic education

Department:
Basic Education
REPUBLIC OF SOUTH AFRICA

NATIONAL SENIOR CERTIFICATE

GRADE 12

INFORMATION TECHNOLOGY P1

NOVEMBER 2010

MARKS: 120

TIME: 3 hours

This question paper consists of 34 pages, 3 annexures and an information sheet.

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 permitted to leave the examination room before the end of the examination session.

2. Answer SECTION A (for Delphi programmers) OR SECTION B (for Java programmers).

3. You require the files listed below in order to answer the questions. They are EITHER on a stiffy disk OR CD issued to you, OR the invigilator/teacher will tell you where to find them on the hard drive of the workstation you are using OR in a network folder:

QUESTION 1

Delphi:	**Java:**
Question1_P.dpr	Sightings.class
Question1_P.res	SightingsDB.mdb
Question1_U.dfm	tblRangers.txt
Question1_U.pas	tblSightings.txt
SightingsDB.mdb	TestSightings.java
tblRangers.txt	
tblSightings.txt	

QUESTION 2

Delphi:	**Java:**
Question2_P.dpr	Competitor.java
Question2_P.res	Sightings.txt
Question2_U.dfm	TestCompetitor.java
Question2_U.pas	
uCompetitor.pas	
Sightings.txt	

QUESTION 3:

Delphi:	**Java:**
Question3_P.dpr	TestDistances.java
Question3_P.res	
Question3_U.dfm	
Question3_U.pas	

If a disk (CD or stiffy) containing the above files was issued to you, write your examination number on the label.

4. Save your work at regular intervals as a precaution against power failures.

5. Save ALL your solutions in folders with the number of the question and your examination number as the name of the folder, for example Quest2_3020160012.

6. Type in your examination number as a comment in the first line of each program.

7. Read ALL the questions carefully. Do not do more than the questions require.

8. During the examination you may use the manuals originally supplied with the hardware and software. You may also use the HELP functions of the software. **Java candidates may make use of the Java API files. You may NOT use any other resource material.**

9. At the end of this examination session you will be required to hand in the disk with your work saved on it OR you must make sure that all your work has been saved on the hard drive/network as explained to you by the invigilator/teacher. Ensure that all files can be read.

10. You also have to hand in printouts of the programming code for all the questions that you did.

11. All printing of programming questions will take place within an hour of the completion of the examination.

12. Complete the information sheet attached to this question paper and hand it in at the end of this examination session.

SECTION A

Answer ALL the questions in this section only if you studied **Delphi**.

SCENARIO:

The Big Five Game Park is a new South African game park which protects a variety of South African wildlife. The game park's priorities are research, nature conservation, animal monitoring and public awareness. They require custom-developed computer software that will assist in performing some everyday tasks.

QUESTION 1: DELPHI PROGRAMMING AND DATABASE

The chief park ranger requires a program that will enable him/her to store personal information on the rangers (staff), as well as details of the sightings of animals being monitored. This information will assist visitors to the park to view details on animals on a daily basis. The program will also assist in monitoring the work carried out by the different rangers on different days. A database called **SightingsDB** has been developed. An incomplete program has been developed to process queries on the information in the given database. Your task will be to complete this program.

NOTE: The design of the tables in the **SightingsDB** database and sample data for this question can be found in **ANNEXURE A: Table Description Sheet**.

NOTE: If you cannot use the database in the provided format, follow the instructions in **ANNEXURE B** to create the database before you answer any of QUESTIONS 1.1 to 1.7.

NOTE: Make a copy of the **SightingsDB** database BEFORE you start with the solution. You will need a copy of the original database to be able to test your program thoroughly.

You have also been supplied with an incomplete Delphi project named **Question1_P.dpr** in the folder named **Question 1 Delphi**.

Do the following:

- Rename the folder **Question 1 Delphi** to **Quest1_X**, where X should be replaced with your examination number.
- Open Delphi and then open the file **Question1_P.dpr** in the folder **Quest1_X**. The program displays eight buttons as well as a DBGrid that will be used as an output component (see example on next page).
- Add your examination number to the right of 'Question 1 –' in the caption of the form.
- Go to File/Save As … and save the unit as **Question1_UXXXX** (where XXXX must be replaced by the last FOUR digits of your examination number).
- Go to File/Save Project As … and save the project as **Question1_PXXXX** (where XXXX must be replaced by the last FOUR digits of your examination number).

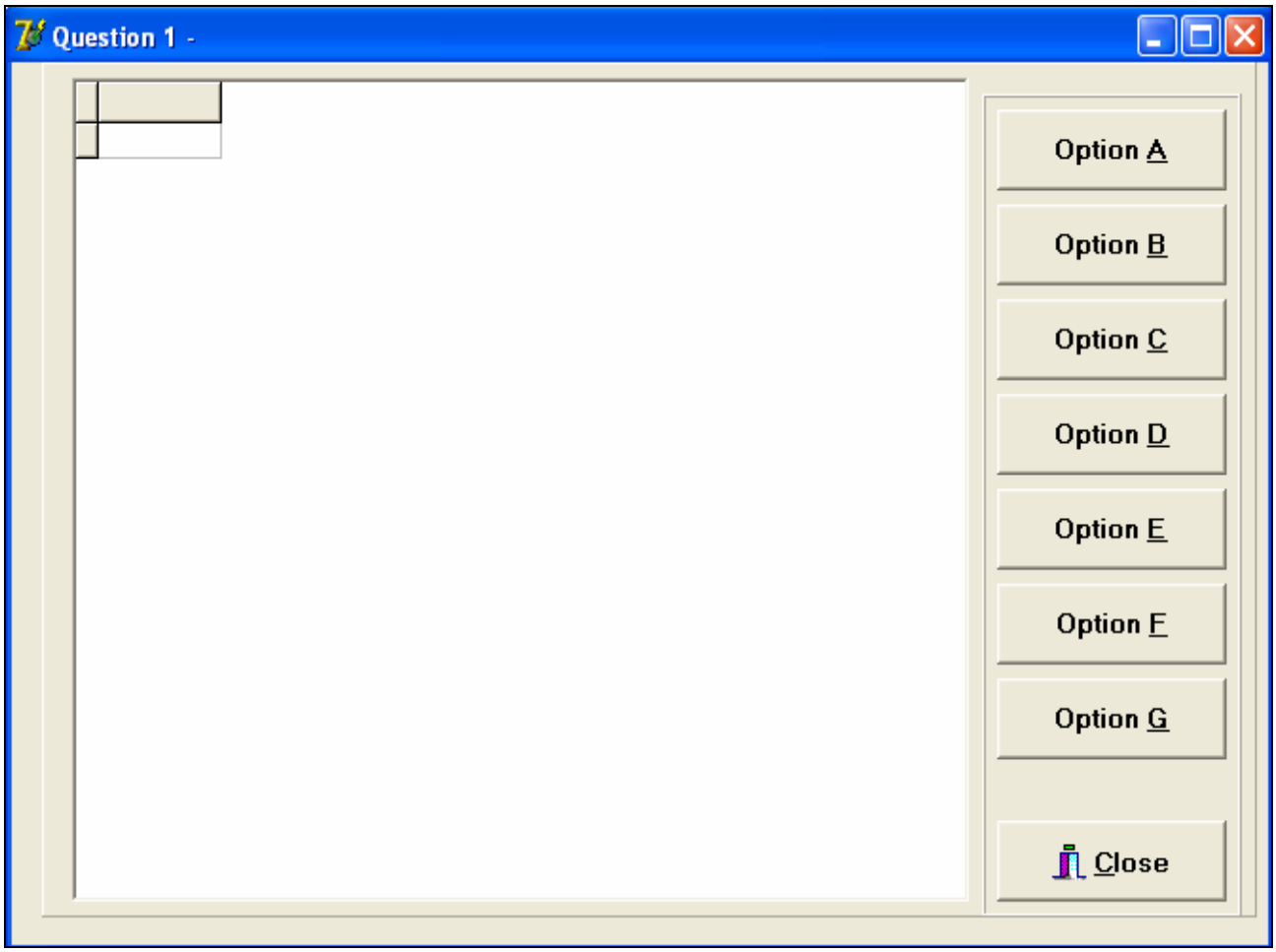
- The program should be able to connect to the database named **SightingsDB**. When you do QUESTION 1.1 (which follows below) and you find that the connectivity is not in place, use the steps supplied in **ANNEXURE C** to establish connection with the database.

HINT: If your program cannot connect to the database, ensure that the database file **SightingsDB** is in the same folder as your program. Your program will not work if the database file is in a folder other than the folder containing your Delphi program. If this is the case, copy the database file **SightingsDB** into the same folder as your program.

NOTE: If you cannot establish connectivity with the database at all when you execute the program, you must still do the SQL code and submit it for marking.

Marks will only be awarded for the code that contain the SQL statements in the unit named Question1_UXXXX.

When you execute the program, the interface below will be displayed. When the buttons are clicked, an error will be displayed due to the incomplete SQL statements.



Do the following:

Complete the SQL statements in **Question1_UXXXX.pas** for each menu option as indicated in QUESTIONS 1.1 to 1.7 below. The code to execute the SQL statements and display the results on the DBGrid has been given to you.

1.1 The chief park ranger would like to view all details of animals sighted on a daily basis. Complete the code for the **Option A** button by formulating an SQL statement to display **all details** of the sightings stored in the **tblSightings** table. Display the output in descending order of **SightingID**.

Example of output for the first five sightings:

SightingID	SightingDate	Animal	NumAnimals	Young	RangerID
200	2010/05/07	Kudu	16	False	9
199	2010/05/31	Kudu	9	True	11
198	2010/04/04	Impala	21	True	4
197	2010/07/29	Cheetah	2	True	4
196	2010/01/19	Kudu	4	True	12

:

NOTE: The date on your output may be in a different format, depending on the settings on your computer. Any format of the date will be acceptable. (4)

1.2 A visiting international student is carrying out a survey and requires information, particularly about the different types of young animals that have been sighted at the park. Complete the code for the **Option B** button by formulating an SQL statement to display **only the names** of the different types of young animals that have been sighted.

NOTE: The name of each type of young animal should be displayed once only.

Example of output:

Animal
Aardvark
Cheetah
Elephant
Giraffe
Impala
Kudu
Lion
Rhino

(4)

1.3 A record needs to be kept to determine the number of years each ranger has been employed at the park. Complete the code for the **Option C** button by formulating an SQL statement to display the **RangerID**, **Name**, **Surname** and the number of years the ranger has been employed. Store the calculated field in **TotalYears**.

Example of output for the first five rangers:

RangerID	Name	Surname	TotalYears
1	Jada	Harrison	8
2	Kenyon	Carney	3
3	Dylan	Pollard	5
4	Sylvester	Walls	7
5	Urielle	Wynn	3

:

(6)

1.4 At the end of every month the chief ranger is required to print a report to show the average number of each type of animal sighted, using all listings in the **tblSightings** table. Complete the code for the **Option D** button by formulating an SQL statement that will display the **animal** name and the average number sighted, rounded off to TWO decimal points (name this field **AvgSighted**). The output must be grouped according to the animal field.

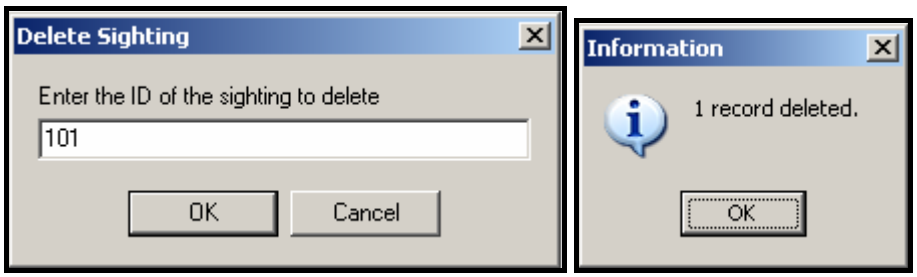
Example of output:

Animal	AvgSighted
Aardvark	13.00
Cheetah	17.46
Elephant	18.15
Giraffe	15.25
Impala	16.92
Kudu	16.18
Lion	15.54
Rhino	16.25

(6)

1.5 The results of sightings during poor weather conditions can sometimes be inaccurate. The park administrator is required to delete inaccurate sightings as they come up. Complete the code for the **Option E** button by allowing the user to enter the **SightingID** of a sighting and then formulate an SQL statement that will **delete** the record of the corresponding sighting.

Example of output:



HINT: Run **Option A** to verify that the record has been deleted. (4)

1.6 All rhinos sighted in the park happen to be white rhinos. Complete the code for the **Option F** button by formulating an SQL statement that will update the **animal** field to 'White Rhino' if the animal field contains the word 'Rhino'.

Example of output:

HINT: Run **Option A** to verify that the records have been updated. (5)

1.7 Information is needed regarding the details of rangers who sighted elephants after a given date. Complete the code for the **Option G** button by formulating an SQL statement to display the **SightingDate**, **Name** and **Surname** of the rangers that sighted elephants after **30/04/2010.**

Example of output:

SightingDate	Name	Surname
2010/05/29	Ivory	Frost
2010/05/31	Karleigh	Jones
2010/07/06	Jada	Harrison
2010/06/08	Odessa	Head

NOTE: The date on your output may be in a different format, depending on the settings on your computer. Any format of the date will be acceptable. (6)

- Enter your examination number as a comment in the first line of the file named **Question1_UXXXX.pas** containing the SQL statements.
- Save the unit **Question1_UXXXX** and the project **Question1_PXXXX** (File/Save All).
- A printout for the code of the **Question1_UXXXX.pas** file will be required. **[35]**

QUESTION 2: DELPHI – OBJECT-ORIENTED PROGRAMMING

The Big Five Game Park wants to launch a competition for the public. The competition will require each person to write down the type of animals that he/she sees (regardless of duplicates), during a day at the park. For the competition, the park divides animals into three categories, **large game**, **small game** and **birds**.

Each competitor will record his/her name and then his/her list of animals together with each animal's category.

Each competitor submits a text file with his/her sightings.

Each competitor's animal sightings will be processed to obtain the total number of animals sighted in each of the three categories. The park's management will award points to each competitor based on the animals they have sighted.

You are required to write a program (as indicated in QUESTION 2.1 and QUESTION 2.2) to process ONE such competitor text file.

The data stored in the text file named **Sightings.txt** in the folder **Question 2 Delphi** contains the information on the **sightings for a single competitor**. The format of the data in the file is as follows:

Name of competitor
Animal(Letter)
Animal(Letter)
etc.

where **Letter** represents one of the three categories of animals as defined above (**L** for large game, **S** for small game and **B** for birds).

Only these three categories (**L, S, B**) are valid. Any other category referred to in the text file, such as (**M**) for medium game, will be regarded as an invalid entry.

An example of the data in the text file:

Jane
Elephant(L)
Rhino(L)
Rhino(L)
Crocodile(M)
Eagle(B)
Owl(B)
Warthog(S)
Meerkat(S)
etc.

Do the following:

- Rename the folder **Question 2 Delphi** as **Quest2_X** (where X must be replaced by your examination number).
- Open Delphi and then open the file **Question2_P.dpr** in the folder **Quest2_X**.
- Go to File/Save As … and save the unit as **Question2_UXXXX** (where XXXX must be replaced by the last FOUR digits of your examination number).
- Open the unit **uCompetitor.pas**.
- Go to File/Save As … and save the unit as **uCompetitorXXXX.pas** (where XXXX must be replaced by the last FOUR digits of your examination number).
- Go to File/Save Project As … and save the project as **Question2_PXXXX** (where XXXX must be replaced by the last FOUR digits of your examination number).

2.1 The object class named **uCompetitorXXXX.pas** will represent the results for a single competitor, **TCompetitor**, including his/her name and how many animals in each category he/she sighted. Note the following:

- All fields in this class should be private and all methods public.
- Parts of this class have been inserted as comments in order to get the class to compile.
- In addition to modifying the given methods, you will be required to add code for new methods as described below.

Do the following in the **uCompetitorXXXX.pas** file:

2.1.1 Create private fields **with the following names** to hold the data. You should choose appropriate data types for these fields:

- **name** – name of competitor
- **largeGameCount** – total number of large game animals sighted
- **smallGameCount** – total number of small game animals sighted
- **birdCount** – total number of birds sighted

It is important that you use the field names given in bold above in order for the given code to operate correctly. (3)

2.1.2 (a) You have been provided with a **default constructor method**. Write an **additional constructor method** that has one parameter for the competitor's name. Initialise the name field using the parameter value and initialise the other fields to zero. (2)

(b) You have been provided with three methods named **spotLarge**, **spotSmall** and **spotBird**. Remove the comment symbols from the code provided in these methods. (1)

2.1.3 You have been provided with an incomplete method (function method) with a defined return data type called **calculatePoints** that should return the total number of points for a competitor. The points are awarded as follows:

- Five points for each large game animal sighted
- Three points for each small game animal sighted
- Two points for each bird sighted

The method contains an incomplete statement that has been blocked out as a comment. The total points are supposed to be calculated by calculating the sum of the number of large game sighted multiplied by five, the number of small game sighted multiplied by three and the number of birds sighted multiplied by two.

Remove the comment symbols from the given statement and complete the code so that the method returns the correct result. (3)

2.1.4 Write a method (function method) called **totalAnimals** which returns the total number of animals sighted as an integer. The total is calculated by calculating the sum of the number of large game, small game and birds sighted. (2)

2.1.5 Write a 'get' method called **getName** to return the name of the competitor. (2)

2.1.6 Write a method (function method) called **mostSpotted** that will determine and return the category of animal ("Large Game", "Small Game" or "Bird") that the competitor spotted the most. (4)

2.1.7 You have been provided with a method (function method) called **toString** that constructs and returns a string with the name and sighting results of a competitor. However, the code provided in the method is incomplete and has been commented out.

Remove the comment symbols so that the given statements will execute and complete the code so that it returns the information in the following format:

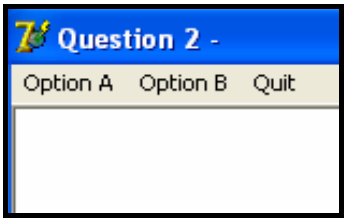
```
Competitor : name
Large : largeGameCount, Small : smallGameCount, Bird : birdCount
Total Animals : <tab>totalAnimals
```

Example of the output when the returned string is displayed:

```
Competitor : Jane
Large : 9  Small : 6  Bird : 7
Total Animals :      22
```

(5)

2.2 In the **Question2_UXXXX.pas** file (the main unit) you have been given code to display the following menu when the program is executed:



Open the **Question2_UXXXX.pas** file (the main unit) and do the following:

- Add your examination number to the caption of the form to the right of 'Question 2 –'.
- Write code to do the following in the **Question2_UXXXX.pas** file (the main unit) in the given program:

2.2.1 Write code in the **OnActivate** event handler of the form to read information from the text file **Sightings.txt** according to the following steps:

(a) Test if the text file exists. Display a suitable message if the file does not exist and terminate the program. Continue with the remainder of the steps if the file exists.

(b) Read the first line from the text file and store this as the competitor's name.

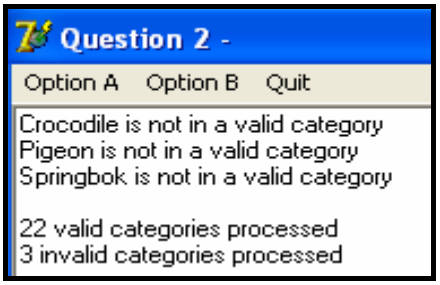
(c) Using the competitor's name, create a **single object** of type **TCompetitor**. You do not need to create an array of these objects as **only one competitor will be processed** each time the program is run.

(d) Use a loop to do the following:

- Read a line of text (one animal) from the text file.
- Test if the category of the animal that appears in brackets after the name of the animal is valid or not. Only the letters L (for large game), S (for small game) and B (for birds) are valid. Display an appropriate message which includes the name of the animal for an invalid category.
- If the category is valid, then call one of the following methods from your **TCompetitor** class: **spotLarge**, **spotSmall**, **spotBird**. These methods increase the number of large game, small game and bird sightings respectively each time the method is called.

(e) Use two counter variables to keep track of how many valid and invalid categories were recorded.

(f) Display the total number of invalid and valid categories recorded as shown below:

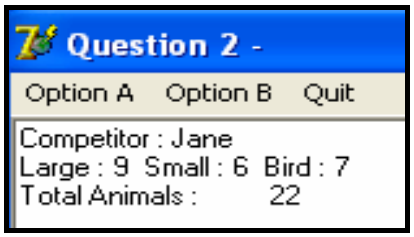


(20)

2.2.2 Menu Option A

When the user selects this menu option, the program must display the name and results for the competitor, by using the **toString** method from the **TCompetitor** class.

Example of output:



(2)

2.2.3 Menu Option B

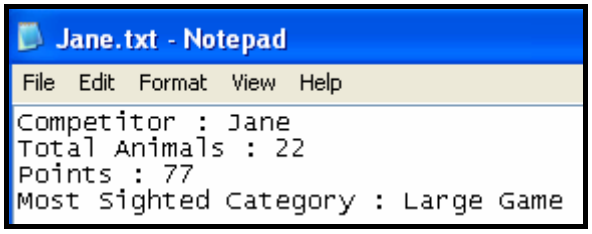
When the user selects this menu option the program must do the following:

- Create a new text file to save the name and results for the competitor. Construct a name for the text file that will contain the name of the competitor.

NOTE: Do not hardcode 'Jane.txt' as the file name since the name of the file should vary according to the name of the competitor.

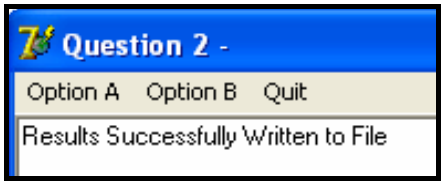
- Call the appropriate methods from the **TCompetitor** class and save the information to the created file.

Example of the contents of the text file:



- Display a message indicating that the information was successfully written to the file.

Example of output:



(5)

- Make sure that your examination number is a comment in the first line of the main class **Question2_UXXXX.pas**, as well as the object class **uCompetitorXXXX.pas**.
- Save all the files (File/Save All).
- Printouts of the code for the classes **Question2_UXXXX.pas** and **uCompetitorXXXX.pas** will be required.

[49]

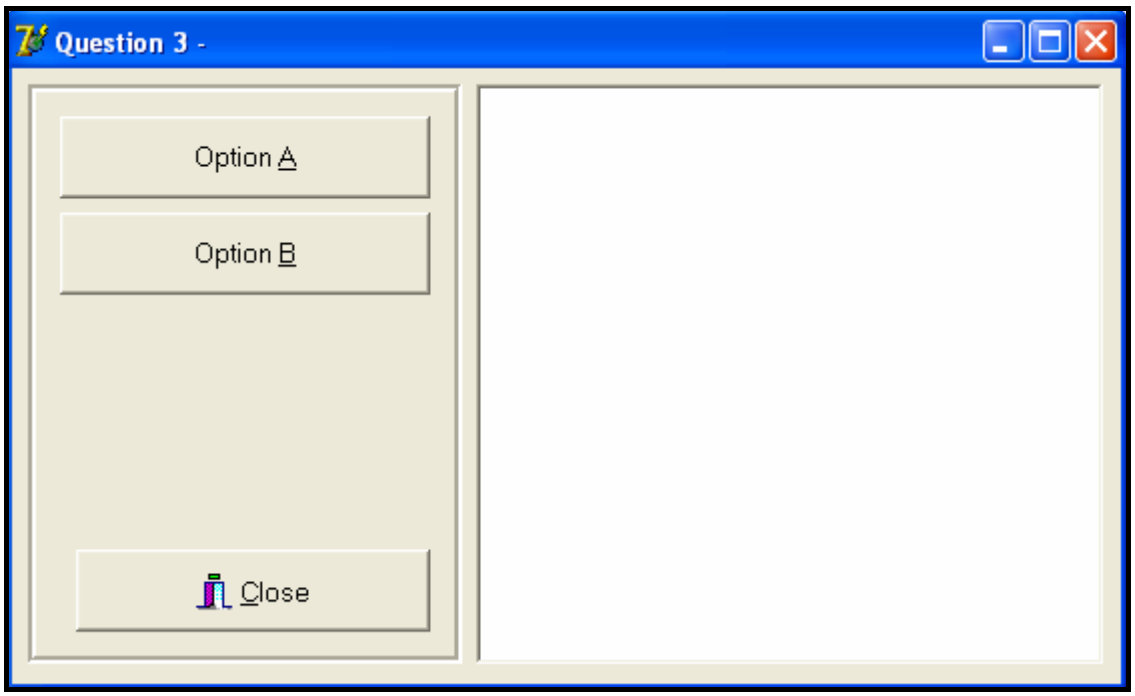
QUESTION 3: DELPHI – PROGRAMMING

The park rangers are doing some research on the movement of a **specific group of animals** relative to a **watering hole** situated at a specific location. They have monitored and recorded the locations of this group of animals over a specific period of time.

You have been given an incomplete program in the folder named **Question 3 Delphi**.

Do the following:

- Rename the folder named **Question 3 Delphi** to **Quest3_X** (where X should be replaced with your examination number).
- Open the Delphi program in this folder.
- Save the unit as ('File/Save As') **Question3_UXXXX** and the project as ('File/Save Project As') **Question3_PXXXX** inside the folder (XXXX should be replaced by the last FOUR digits of your examination number).
- Add your examination number to the caption of the form to the right of 'Question 3 –'.
- Execute the program. A menu with the following options will be displayed:



3.1 Do the following:

- Declare an array called **arrEntries** that must contain a maximum of 12 strings.
- Remove the comment symbols from the code supplied to assign strings to the **arrEntries** array.
- Use the array appropriately in your program to answer the questions that follow.

Each array entry consists of a string indicating the coordinates of the location of the group of animals relative to the watering hole at a specific time. Each string has the following format:

X-Coordinate,Y-Coordinate:Time recorded

Example of the first five strings assigned to the array:

```
arrEntries[1]  := '12,15:02h00';
arrEntries[2]  := '13,10:05h00';
arrEntries[3]  := '9,20:06h00';
arrEntries[4]  := '10,15:09h00';
arrEntries[5]  := '7,8:10h00';
```

NOTE:
In the first entry:
- The x-coordinate is 12
- The y-coordinate is 15
- The time is 02h00

(3)

3.2 The coordinates of the watering hole are as follows:

X-coordinate = 10
Y-coordinate = 10

Complete the code for the **Option A** button as follows:

- Declare TWO variables to store the coordinates of the watering hole and assign values to the variables.
- For each recording in the **arrEntries** array, extract the x- and y-coordinates and the time recorded. Use the x- and y- coordinates to calculate the distance between the location of the animals and the location of the watering hole. The distance should be an integer which has been rounded off. Use the following formula to calculate the distance:

$$\text{Distance} = \sqrt{(\text{AnimalXpos} - \text{WaterXpos})^2 + (\text{AnimalYpos} - \text{WaterYpos})^2}$$

where

AnimalXpos represents the x-coordinate of where the animal was spotted
AnimalYpos represents the y-coordinate of where the animal was spotted
WaterXpos represents the x-coordinate of where the watering hole is situated
WaterYpos represents the y-coordinate of where the watering hole is situated

- For each entry in the **arrEntries** array display the time, the distance from the watering hole and the x- and y-coordinates. Use a suitable heading and subheadings.

Example of output (on the next page):

Distances from the watering hole			
Time	Distance(km)	X-pos	Y-pos
02h00	5	12	15
05h00	3	13	10
06h00	10	9	20
09h00	5	10	15
10h00	4	7	8
11h00	0	10	10
14h00	8	12	18
17h00	9	7	18
19h00	3	11	7
20h00	0	10	10
23h00	12	2	1
24h00	7	12	17

(17)

3.3 The rangers had difficulty in tracking the animals. To facilitate the tracking of animals, it has been decided to create a tag for each animal in order to locate them more easily.

You are required to write code for the **Option B** button that will allow you to do the following:

- Enter the number of different types of animals in a mixed group of animals.
- Enter each type of animal in the group followed by the number of that type of animal in the group.

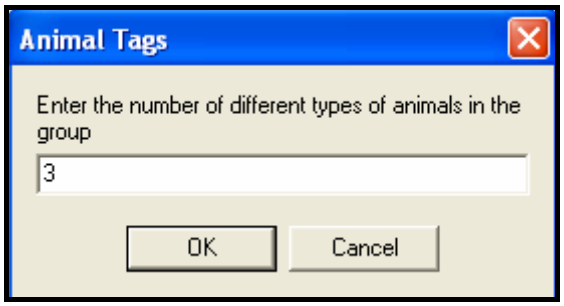
Your code must generate a tag for each of the animals in the group. The tag is generated as follows:

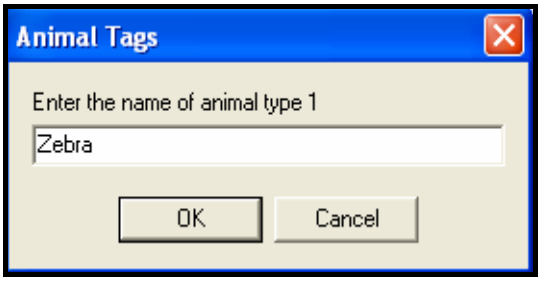
- Extract the first two letters from the name of the type of animal, for example "Rh" will be extracted from Rhino.
- Extract the last letter from the name of the type of animal, for example "o" will be extracted from Rhino.
- Randomly generate a three-digit even number per animal type.
- Combine the extracted letters and the number generated to form the first part of the tag.
- For each animal of a specific type, add a hyphen and a unique number beginning at 1 for the first animal of a specific type to the tag as indicated in the sample output.

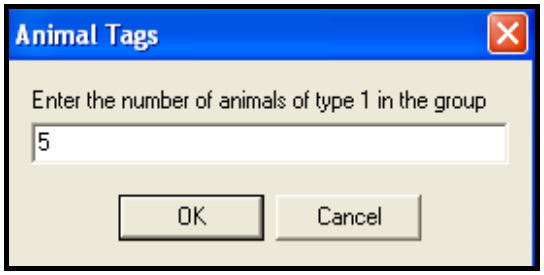
You are required to display the name of the type of animal as part of a heading and a numbered list with a tag number for each animal of the specific type.

Example of input and output for a group of animals consisting of three different types of animals, for example five zebra, four black wildebeest and ten impala (on the next page):

Input:



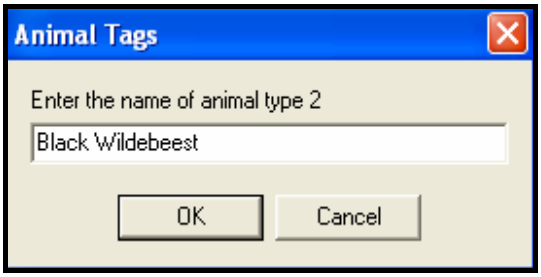


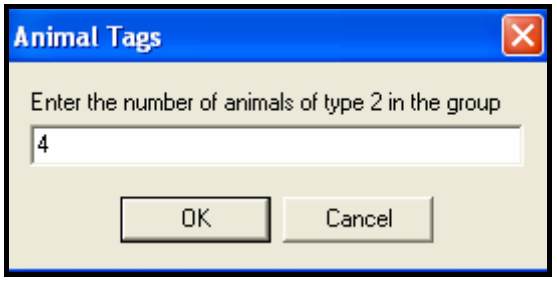


Output:

Zebra	Tag number
1.	Zea782-1
2.	Zea782-2
3.	Zea782-3
4.	Zea782-4
5.	Zea782-5

Input:

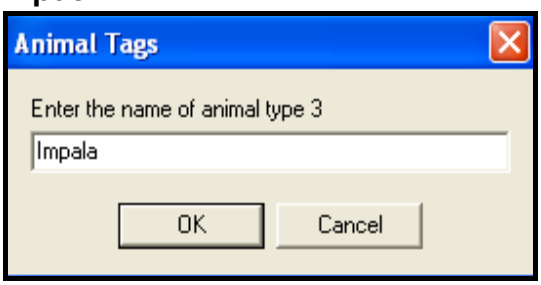


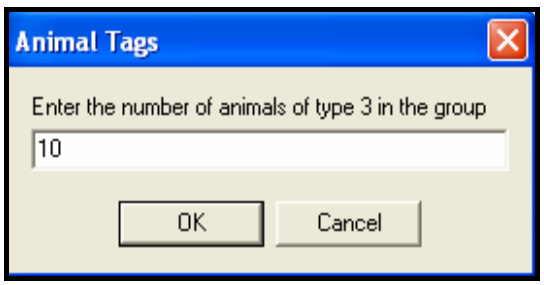


Output:

Black Wildebeest	Tag number
1.	Blt548-1
2.	Blt548-2
3.	Blt548-3
4.	Blt548-4

Input:





Output:

Impala	Tag number
1.	Ima658-1
2.	Ima658-2
3.	Ima658-3
4.	Ima658-4
5.	Ima658-5
6.	Ima658-6
7.	Ima658-7
8.	Ima658-8
9.	Ima658-9
10.	Ima658-10

NOTE: Different random numbers will be generated for each execution of the program. (16)

- Enter your examination number as a comment in the first line of the unit **Question3_UXXXX**, as well as any other unit(s) you may have created.
- Save the unit(s) and the project ('File/Save All').
- A printout of the code for the unit **Question3_UXXXX**, as well as any other unit(s) you may have created, will be required. **[36]**

TOTAL SECTION A: 120

SECTION B

Answer ALL the questions in this section only if you studied **Java**.

SCENARIO:

The Big Five Game Park is a new South African game park which protects a variety of South African wildlife. The game park's priorities are research, nature conservation, animal monitoring and public awareness. They require custom-developed computer software that will assist in performing some everyday tasks.

QUESTION 1: JAVA PROGRAMMING AND DATABASE

The chief park ranger requires a program that will enable him/her to store personal information on the rangers (staff), as well as details of the sightings of animals being monitored. This information will assist visitors to the park to view details on animals on a daily basis. The program will also assist in monitoring the work carried out by the different rangers on different days. A database called **SightingsDB** has been developed. An incomplete program has been developed to process queries on the information in the given database. Your task will be to complete this program.

NOTE: The design of the tables in the **SightingsDB** database and sample data for this question can be found in **ANNEXURE A: Table Description Sheet**.

NOTE: If you cannot use the database in the provided format, follow the instructions in **ANNEXURE B** to create the database before you answer any of QUESTIONS 1.1 to 1.7.

NOTE: Make a copy of the **SightingsDB** database BEFORE you start with the solution. You will need a copy of the original database to be able to test your program thoroughly.

You have also been supplied with an incomplete Java program, in the folder named **Question 1 Java**, with a test class named **TestSightings.java** and an object class named **Sightings.class** which will display the results of the queries.

Do the following:

- Rename the folder **Question 1 Java** as **Quest1_X**, where X should be replaced with your examination number.
- Rename the **TestSightings.java** file in the folder **Quest1_X** to **TestSightingsXXXX.java** (where XXXX must be replaced by the last FOUR digits of your examination number).
- Open the **TestSightingsXXXX.java** file.
- Change the name of the class to **TestSightingsXXXX** (where XXXX must be replaced by the last FOUR digits of your examination number).

NOTE: If you compile and run the **TestSightingsXXXX.java** file, the following menu will be displayed (see on the next page). However, if you enter any of the options A to G, the program will not work because of the incomplete SQL statements.

```
     MENU

  Option A
  Option B
  Option C
  Option D
  Option E
  Option F
  Option G

  Q - QUIT
```

The connectivity code, as well as the code to display the results, are contained in the file named **Sightings.class**.

HINT: If your program cannot connect to the database, ensure that the database file **SightingsDB** is in the same folder as the files **TestSightingsXXXX.java** and **Sightings.class**. Your program will not work if the database file is in a folder other than the folder containing your Java program. If this is the case, copy the database file **SightingsDB** into the same folder as your program.

NOTE: If you cannot establish connectivity with the database at all when you execute the program, you must still do the SQL code and submit it for marking.

Marks will only be awarded for the programming code which contains the SQL statements in the program named TestSightingsXXXX.java.

Do the following:

Complete the SQL statements in **TestSightingsXXXX.java** for each menu option as indicated in QUESTIONS 1.1 to 1.7 below. The code to pass the SQL statements to the relevant methods in the **Sightings.class** file has already been programmed for you.

1.1 The chief park ranger would like to view all details of animals sighted on a daily basis. Complete the code for menu **Option A** by formulating an SQL statement to display **all details** of the sightings stored in the **tblSightings** table. Display the output in descending order according to **SightingID**.

Example of output of the first five sightings:

SightingID	SightingDate	Animal	NumAnimals	Young	RangerID
200	2010-05-07	Kudu	16	False	9
199	2010-05-31	Kudu	9	True	11
198	2010-04-04	Impala	21	True	4
197	2010-07-29	Cheetah	2	True	4
196	2010-01-19	Kudu	4	True	12
:					

NOTE: The date on your output may be in a different format, depending on the settings on your computer. Any format of the date will be acceptable. (4)

1.2 A visiting international student is carrying out a survey and requires information particularly about the different types of young animals that have been sighted at the park. Complete the code for menu **Option B** by formulating an SQL statement to display **only the names** of the different types of young animals that have been sighted.

NOTE: The name of each type of young animal should be displayed once only.

Example of output:

```
Animal
==========
Aardvark
Cheetah
Elephant
Giraffe
Impala
Kudu
Lion
Rhino
```

(4)

1.3 A record needs to be kept to determine the number of years each ranger has been employed at the park. Complete the code for menu **Option C** by formulating an SQL statement to display the **RangerID**, **Name**, **Surname** and the number of years the ranger has been employed. Store the calculated field in **TotalYears**.

Example of output for the first five rangers:

```
RangerID        Name                Surname             TotalYears
=================================================================
1               Jada                Harrison            8
2               Kenyon              Carney              3
3               Dylan               Pollard             5
4               Sylvester           Walls               7
5               Urielle             Wynn                3
:
```

(6)

1.4 At the end of every month, the chief ranger is required to print a report to show the average number of each type of animal sighted, using all listings in the **tblSightings** table. Complete the code for menu **Option D** by formulating an SQL statement that will display the **animal** name and the average number sighted, rounded off to TWO decimal points (name this field **AvgSighted**). The output must be grouped according to the animal field.

Example of output (on the next page):

Animal	AvgSighted
Aardvark	13.0
Cheetah	17.46
Elephant	18.15
Giraffe	15.25
Impala	16.92
Kudu	16.18
Lion	15.54
Rhino	16.25

(6)

1.5 The results of sightings during poor weather conditions can sometimes be inaccurate. The park administrator is required to delete inaccurate sightings as they come up. Complete the code for menu **Option E** by allowing the user to enter the **SightingID** of a sighting and then formulate an SQL statement that will **delete** the record of the corresponding sighting.

Example of output:

```
Enter the ID of the sighting to delete
101

1 record deleted
```

HINT: Run **Option A** to verify that the record has been deleted. (4)

1.6 All rhinos sighted in the park happen to be white rhinos. Complete the code for menu **Option F** by formulating an SQL statement that will update the **animal field** to 'White Rhino' if the animal field contains the word 'Rhino'.

Example of output:

```
12 records updated
```

HINT: Run **Option A** to verify that the records have been updated. (5)

1.7 Information is needed regarding the details of rangers who sighted elephants after a given date. Complete the code for menu **Option G** by formulating an SQL statement to display the **SightingDate**, **Name** and **Surname** of the rangers that sighted elephants after **30/04/2010**.

Example of output:

SightingDate	Name	Surname
2010-05-29	Ivory	Frost
2010-05-31	Karleigh	Jones
2010-07-06	Jada	Harrison
2010-06-08	Odessa	Head

NOTE: The date on your output may be in a different format, depending on the settings on your computer. Any format of the date will be acceptable. (6)

- Enter your examination number as a comment in the first line of the file named **TestSightingsXXXX.java** containing the SQL statements.
- Save the file **TestSightingsXXXX.java**.
- A printout for the code of the **TestSightingsXXXX.java** file will be required. **[35]**

QUESTION 2: JAVA – OBJECT-ORIENTED PROGRAMMING

The Big Five Game Park wants to launch a competition for the public. The competition will require each person to write down the type of animals that he/she sees (regardless of duplicates), during a day at the park. For the competition, the park divides animals into three categories, **large game**, **small game** and **birds**.

Each competitor will record his/her name and then his/her list of animals together with each animal's category.

Each competitor submits a text file with his/her sightings.

Each competitor's animal sightings will be processed to obtain the total number of animals sighted in each of the three categories. The park's management will award points to each competitor based on the animals they have sighted.

You are required to write a program (as indicated in QUESTION 2.1 and QUESTION 2.2) to process ONE such competitor text file.

The data stored in the text file named **Sightings.txt** in the folder **Question 2 Java** contains the information on the **sightings for a single competitor**. The format of the data in the file is as follows:

Name of competitor
Animal(Letter)
Animal(Letter)
etc.

where **Letter** represents one of the three categories of animals as defined above (**L** for large game, **S** for small game and **B** for birds).

Only these three categories (**L, S, B**) are valid. Any other category referred to in the text file, such as (**M**) for medium game, will be regarded as an invalid entry.

An example of the data in the text file:

Jane
Elephant(L)
Rhino(L)
Rhino(L)
Crocodile(M)
Eagle(B)
Owl(B)
Warthog(S)
Meerkat(S)
etc.

Do the following:

- Rename the folder **Question 2 Java** as **Quest2_X** (where X must be replaced by your examination number).
- Rename the **Competitor.java** file in the folder **Quest2_X** to **CompetitorXXXX** (where XXXX must be replaced by the last FOUR digits of your examination number).
- Open the **CompetitorXXXX.java** file.
- Change the **class name and constructor method** to **CompetitorXXXX** (where XXXX must be replaced by the last FOUR digits of your examination number).
- Add your examination number as a comment in the first line of the **CompetitorXXXX.java** class. Save the file.
- Rename the **TestCompetitor.java** file in the folder **Quest2_X** to **TestCompetitorXXXX** (where XXXX must be replaced by the last FOUR digits of your examination number).
- Open the **TestCompetitorXXXX.java** file.
- Change the **class name** to **TestCompetitorXXXX** (where XXXX must be replaced by the last FOUR digits of your examination number). Save the file.

2.1 The object class named **CompetitorXXXX.java** will represent the results for a single competitor including his/her name and how many animals in each category he/she sighted. Note the following:

- All fields in this class should be private and all methods public.
- Parts of this class have been inserted as comments in order to get the class to compile.
- In addition to modifying the given methods, you will be required to add code for new methods as described below.

Do the following in the **CompetitorXXXX.java** file:

2.1.1 Create private fields **with the following names** to hold the data. You should choose appropriate data types for these fields:

- **name** – name of competitor
- **largeGameCount** – total number of large game animals sighted
- **smallGameCount** – total number of small game animals sighted
- **birdCount** – total number of birds sighted

It is important that you use the field names given in bold above in order for the given code to operate correctly. (3)

2.1.2 (a) You have been provided with a **default constructor method**. Write an **additional constructor method** that has one parameter for the competitor's name. Initialise the name field using the parameter value and initialise the other fields to zero. (2)

(b) You have been provided with three methods named **spotLarge**, **spotSmall** and **spotBird**. Remove the comment symbols from the code provided in these methods. (1)

2.1.3 You have been provided with an incomplete typed method called **calculatePoints** that should return the total number of points for a competitor. The points are awarded as follows:

- Five points for each large game animal sighted
- Three points for each small game animal sighted
- Two points for each bird sighted

The method contains an incomplete statement that has been blocked out as a comment. The total points are supposed to be calculated by calculating the sum of the number of large game sighted multiplied by five, the number of small game sighted multiplied by three and the number of birds sighted multiplied by two.

Remove the comment symbols from the given statement and complete the code so that the method returns the correct result. (3)

2.1.4 Write a method called **totalAnimals** which returns the total number of animals sighted as an integer. The total is calculated by calculating the sum of the number of large game, small game and birds sighted. (2)

2.1.5 Write a 'get' method called **getName** to return the name of the competitor. (2)

2.1.6 Write a method called **mostSpotted** that will determine and return the category of animal ("Large Game", "Small Game" or "Bird") that the competitor spotted the most. (4)

2.1.7 You have been provided with a method called **toString** that constructs and returns a string with the name and sighting results of a competitor. However, the code provided in the method has been inserted as comments.

Remove the comment symbols so that the given statements will execute and complete the code so that it returns the information in the following format:

```
Competitor : name
Large : largeGameCount, Small : smallGameCount, Bird : birdCount
Total Animals : <tab>totalAnimals
```

Example of the output when the returned string is displayed (on the next page):

```
Competitor : Jane
Large : 9  Small : 6  Bird : 7
Total Animals :        22
```

(5)

2.2 In the **TestCompetitorXXXX.java** file (the main class) you have been given code to display the following menu when the program is executed:

```
          Menu

    Option A
    Option B

    Q - QUIT

    Your choice? :
```

Open the **TestCompetitorXXXX.java** file (the main class) and do the following:

- Add your examination number as a comment in the first line of the **TestCompetitorXXXX.java** class.
- Write code to do the following in the **TestCompetitorXXXX.java** file (the main class) in the given program:

2.2.1 Read information from the text file **Sightings.txt** according to the following steps:

(a) Test if the text file exists. Display a suitable message if the file does not exist and terminate the program. Continue with the remainder of the steps if the file exists.

(b) Read the first line from the text file and store this as the competitor's name.

(c) Using the competitor's name, create a **single object** of type **CompetitorXXXX**. You do not need to create an array of these objects as **only one competitor will be processed** each time the program is run.

(d) Use a loop to do the following:

- Read a line of text (one animal) from the text file.
- Test if the category of the animal that appears in brackets after the name of the animal is valid or not. Only the letters L (for large game), S (for small game) and B (for birds) are valid. Display an appropriate message which includes the name of the animal for an invalid category.
- If the category is valid, then call one of the following methods from your **CompetitorXXXX** class: **spotLarge**, **spotSmall**, **spotBird**. These methods increase the number of large game, small game and bird sightings respectively each time the method is called.

(e) Use two counter variables to keep track of how many valid and invalid categories were recorded.

(f) Display the total number of invalid and valid categories as shown below.

```
Crocodile is not in a valid category
Pigeon is not in a valid category
Springbok is not in a valid category

22 valid categories processed
3 invalid categories processed
```

(20)

2.2.2 Menu Option A

When the user selects this menu option, the program must display the name and the results for the competitor by using the **toString** method from the **CompetitorXXXX** class.

Example of output:

```
Competitor : Jane
Large : 9  Small : 6  Bird : 7
Total Animals :         22
```

(2)

2.2.3 Menu Option B

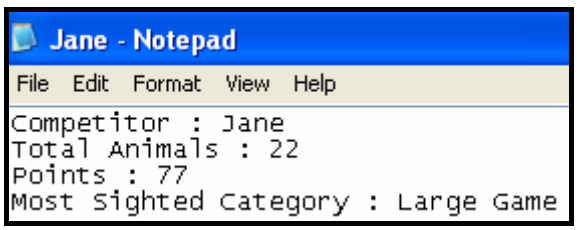
When the user selects this menu option the program must do the following:

- Create a new text file to save the name and results for the competitor. Construct a name for the text file that will contain the name of the competitor.

NOTE: Do not hardcode 'Jane.txt' as the file name since the name of the file should vary according to the name of the competitor.

- Call the appropriate methods from the **CompetitorXXXX** class to save the information to the created file.

Example of the contents of the text file:



- Display a message indicating that the information was successfully written to the file.

Example of output:

```
Results Successfully Written to File
```

(5)

- Make sure that your examination number is entered as a comment in the first line of the main class **TestCompetitorXXXX.java** as well as the object class **CompetitorXXXX.java**.
- Save all the files (File/Save All).
- Printouts of the code for the classes **TestCompetitorXXXX.java** and **CompetitorXXXX.java** will be required. **[49]**

QUESTION 3: JAVA – PROGRAMMING

The park rangers are doing some research on the movement of a **specific group of animals** relative to a **watering hole** situated at a specific location. They have monitored and recorded the locations of this group of animals over a specific period of time.

You have been given an incomplete program in the folder named **Question 3 Java**.

Do the following:

- Rename the folder named **Question 3 Java** to **Quest3_X** (where X should be replaced with your examination number).
- Rename the file **TestDistances** in this folder to **TestDistancesXXXX.java** (XXXX should be replaced by the last FOUR digits of your examination number).
- Open the file (incomplete program) **TestDistancesXXXX.java**. Change the name of the class to **TestDistancesXXXX.java**.
- Add your examination number as a comment in the first line of the program.
- Execute the program. A menu with the following options will be displayed:

```
Menu

Option A
Option B

Q - QUIT
```

NOTE: You may use one or more classes for this solution.

3.1 Do the following:

- Declare an array called **arrEntries** that must contain a maximum of 12 strings.
- Use the provided assignment statements to initialise the **arrEntries** array.
- Use the array appropriately in your program to answer the questions that follow.

Each array entry consists of a string indicating the coordinates of the location of the group of animals relative to the watering hole at a specific time. Each string has the following format:

X-Coordinate,Y-Coordinate:Time Recorded

Example of the first five strings assigned to the array (on the next page):

```
arrEntries[0]  := "12,15:02h00";
arrEntries[1]  := "13,10:05h00";
arrEntries[2]  := "9,20:06h00";
arrEntries[3]  := "10,15:09h00";
arrEntries[4]  := "7,8:10h00";
```

NOTE:
In the first entry:
- The x-coordinate is 12
- The y-coordinate is 15
- The time is 02h00

(3)

3.2 The coordinates of the watering hole are as follows:

X-coordinate = 10
Y-coordinate = 10

Complete the code for the option **Option A** as follows:

- Declare TWO variables to store the coordinates of the watering hole and assign values to these variables.

- For each recording in the **arrEntries** array, extract the x- and y-coordinates and the time recorded. Use the x- and y-coordinates to calculate the distance between the location of the animals and the location of the watering hole. The distance should be an integer which has been rounded off. Use the following formula to calculate the distance:

$$\text{Distance} = \sqrt{(\text{AnimalXpos} - \text{WaterXpos})^2 + (\text{AnimalYpos} - \text{WaterYpos})^2}$$

where

AnimalXpos represents the x-coordinate of where the animal was spotted
AnimalYpos represents the y-coordinate of where the animal was spotted
WaterXpos represents the x-coordinate of where the watering hole is situated
WaterYpos represents the y-coordinate of where the watering hole is situated

- For each entry in the **arrEntries** array, display the time, the distance from the watering hole and the x- and y-coordinates. Use a suitable heading and subheadings.

  Example of output (on next page):

Distances from the watering hole			
Time	Distance(km)	X-pos	Y-pos
02h00	5	12	15
05h00	3	13	10
06h00	10	9	20
09h00	5	10	15
10h00	3	7	8
11h00	0	10	10
14h00	8	12	18
17h00	8	7	18
19h00	3	11	7
20h00	0	10	10
23h00	12	2	1
24h00	7	12	17

(17)

3.3 The rangers had difficulty in tracking the animals. To facilitate the tracking of animals, it has been decided to create a tag for each animal in order to locate them more easily.

You are required to write code for **Option B** that will allow you to do the following:

- Enter the number of different types of animals in a mixed group of animals.
- Enter each type of animal in the group followed by the number of that type of animal in the group.

Your code must generate a tag for each of the animals in the group. The tag is generated as follows:

- Extract the first two letters from the name of the type of animal, for example "Rh" will be extracted from Rhino.
- Extract the last letter from the name of the type of animal, for example "o" will be extracted from Rhino.
- Randomly generate a three-digit even number per animal type.
- Combine the extracted letters and the number generated to form the first part of the tag.
- For each animal of a specific type, add a hyphen and a unique number beginning at 1 for the first animal of a specific type to the tag as indicated in the sample output.

You are required to display the name of the type of animal as part of a heading and a numbered list with a tag number for each animal of the specific type.

Example of input and output for a group of animals consisting of three different types of animals, for example five zebra, four black wildebeest and ten impala (on the next page):

```
Enter the number of different types of animals in the group : 3

Enter the name of animal type 1 : Zebra
Enter the number of animals of type 1 in the group : 5

Zebra                Tag number
1.                   Zea794-1
2.                   Zea794-2
3.                   Zea794-3
4.                   Zea794-4
5.                   Zea794-5

Enter the name of animal type 2 : Black Wildebeest
Enter the number of animals of type 2 in the group : 4

Black Wildebeest     Tag number
1.                   Blt780-1
2.                   Blt780-2
3.                   Blt780-3
4.                   Blt780-4

Enter the name of animal type 3 : Impala
Enter the number of animals of type 3 in the group : 10

Impala               Tag number
1.                   Ima726-1
2.                   Ima726-2
3.                   Ima726-3
4.                   Ima726-4
5.                   Ima726-5
6.                   Ima726-6
7.                   Ima726-7
8.                   Ima726-8
9.                   Ima726-9
10.                  Ima726-10
```

NOTE: Different random numbers will be generated for each execution of the program. (16)

- Enter your examination number as a comment in the first line of the class **TestDistancesXXXX.java**, as well as any other class(es) you may have created.
- Save the class(es).
- A printout of the code for the class **TestDistancesXXXX.java**, as well as any other class(es) you may have created, will be required. **[36]**

TOTAL SECTION B: 120
GRAND TOTAL: 120

ANNEXURE A: Table description sheet

This sheet shows the data structure and sample data for the tables used in the **SightingsDB** database in **Question 1.**

tblSightings Table Structure

tblSightings : Table

Field Name	Data Type	Description
SightingID	Number	A unique ID given to each sighting
SightingDate	Date/Time	The date of the sighting
Animal	Text	The animal that was sighted
NumAnimals	Number	The number of animals that were sighted
Young	Yes/No	If there were any young sighted
RangerID	Number	The ID of the ranger that sighted the animals

tblRangers Table Structure

tblRangers : Table

Field Name	Data Type	Description
RangerID	Number	A unique ID for each Ranger
Name	Text	The name of the ranger
Surname	Text	The surname of the ranger
Rank	Text	The rank of the ranger
DateAppointed	Date/Time	The date the ranger was appointed

tblSightings Table – Data Sample

Microsoft Access - [tblSightings : Table]

File Edit View Insert Format Records Tools Window Help

SightingID	SightingDate	Animal	NumAnimals	Young	RangerID
101	5/3/2010	Giraffe	2	True	3
102	4/5/2010	Impala	23	False	5
103	1/15/2010	Lion	13	False	7
105	6/2/2010	Aardvark	1	True	8
106	4/3/2010	Elephant	1	False	3
107	6/21/2010	Lion	23	False	13
108	4/21/2010	Elephant	32	False	1
109	5/8/2010	Rhino	13	False	2
110	6/23/2010	Rhino	13	False	3
111	5/29/2010	Elephant	10	False	15
112	3/25/2010	Aardvark	7	True	19
113	3/26/2010	Giraffe	14	True	20
114	2/12/2010	Aardvark	25	True	14
115	4/1/2010	Cheetah	14	False	18
116	7/2/2010	Giraffe	7	False	20
117	5/31/2010	Elephant	17	False	14
118	1/19/2010	Impala	8	False	18
119	6/14/2010	Lion	8	True	7
120	1/22/2010	Giraffe	15	False	3

tblRangers Table – Data Sample

Microsoft Access - [tblRangers : Table]

File Edit View Insert Format Records Tools Window Help

RangerID	Name	Surname	Rank	DateAppointed
1	Jada	Harrison	Park	10/18/2002
2	Kenyon	Carney	Park	4/30/2007
3	Dylan	Pollard	Field	5/13/2005
4	Sylvester	Walls	Senior	11/1/2003
5	Urielle	Wynn	Park	12/1/2007
6	Giselle	Head	Field	6/27/2003
7	Amos	Roach	Recruit	6/26/2005
8	Tobias	Paul	Park	9/19/2004
9	Odessa	Head	Section	6/7/2003
10	Charles	Buckner	Section	8/7/2008
11	Ariel	Hooper	Recruit	8/31/2002
12	Hammett	Gates	Field	7/2/2005
13	Tamara	Frazier	Field	1/21/2004
14	Karleigh	Jones	Recruit	8/23/2003
15	Ivory	Frost	Field	1/3/2009
16	Caryn	Gill	Senior	6/3/2006
17	Denise	Thornton	Field	5/23/2006
18	Lev	Sparks	Recruit	11/11/2008
19	Alyssa	Jones	Section	9/20/2005
20	Conan	Sheppard	Section	7/22/2002

ANNEXURE B: Instructions to create the database SightingsDB.mdb

If you cannot use the database provided, do the following:

- Use the two text files named **tblSightings.txt** and **tblRangers.txt** supplied. Create your own database named **SightingsDB** with a table named **tblSightings** and another table named **tblRangers** in the folder called **Question 1 Delphi** or **Question 1 Java**.
- Change the data types and the sizes of the fields in the two tables to the specifications given below.

The **tblSightings** table stores data on the sightings of animals for the game park. The fields in the **tblRangers** table are defined as follows:

Field Name	Type	Size	Comment
SightingID	Number	Integer	A unique ID given to each sighting
SightingDate	Date/Time	ShortDate	The date of the sighting
Animal	Text	20	The animal that was sighted
NumAnimals	Number	Integer	The number of animals that were sighted
Young	Yes/No	Boolean	If there were any young sighted
RangerID	Number	Integer	The ID of the ranger that sighted the animals

See **ANNEXURE A:** Example of the data contained in the **tblSightings** table.

The fields in the **tblRangers** table are defined as follows:

Field Name	Type	Size	Comment
RangerID	Number	Integer	A unique ID for each ranger
Name	Text	20	The name of the ranger
Surname	Text	20	The surname of the ranger
Rank	Text	20	The rank of the ranger
DateAppointed	Date/Time	ShortDate	The date the ranger was appointed

See **ANNEXURE A:** Example of the data contained in the **tblRangers** table.

ANNEXURE C: Instructions to connect to the database in Delphi

In Delphi: If you cannot use the database provided, do the following:

- Click on the ADOQuery component named **qrySightings**.
- Click on the Ellipse button (three dots) to the right of the 'ConnectionString' property in the Object Inspector.
- Click on the Build button which takes you to the Data Link Properties dialogue box.
- Click on the Provider tab to open the provider tab sheet and select Microsoft Jet 4.0 OLE DB Provider. Click on the Next button.
- The Connection tab sheet will be displayed. The first option on the Connection tab sheet provides an Ellipse button (three dots) that allows you to browse and look for the **SightingsDB** file. You will find this file in the **Question 1 Delphi** folder. Once you have found it, select the **SightingsDB** file and click on the Open button.
- Remove the user name Admin.
- Click on the Test Connection button.
- Click OK on each one of the open dialogue windows.

120

INFORMATION TECHNOLOGY PAPER 1
NOVEMBER 2010

INFORMATION SHEET *(to be completed by candidates)*

NAME OF PROVINCE _____

CENTRE NUMBER _____

EXAMINATION NUMBER _____

WORK STATION NUMBER _____

DATE OF EXAMINATION _____

Programming language used
(Mark appropriate box with a cross (X).)

Delphi	Java

FOLDER NAME _____

Enter the file name used for each answer and tick if saved.

Question number	File names	Saved (*tick* ✓)	Maximum mark	Mark achieved	Marker initial/ code
1			35		
2			49		
3			36		
TOTAL			**120**		

Comment (*for official use only)*

_____

_____

_____

_____

_____

_____