education

Department:
Education
REPUBLIC OF SOUTH AFRICA

NATIONAL SENIOR CERTIFICATE

GRADE 12

INFORMATION TECHNOLOGY P1

EXEMPLAR 2008

MARKS: 120

TIME: 3 hours

This question paper consists of 39 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 permitted to leave the examination room before the end of the examination session.

2. Answer EITHER 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/educator will tell you where to find them on the hard drive of the workstation you are using or in a network folder.

DELPHI

QUESTION 1 DiseasesDatabaseU.pas
DiseasesDatabaseP.dpr
DiseasesDatabaseU.dfm
DiseasesDB.mdb
ConditionsTb.txt
WorkplacesTb.txt

QUESTION 2 TestCompanies_U.pas
TestCompanies_P.dpr
TestCompanies_U.dfm
Pollution.txt

QUESTION 3 OilSpill_U.pas
OilSpill_P.dpr
OilSpill_U.dfm

JAVA

QUESTION 1 DiseasesDatabase.java
TestDatabase.java
DiseasesDB.mdb
ConditionsTb.txt
WorkplacesTb.txt

QUESTION 2 TestCompany.java
Company.java
Pollution.txt

QUESTION 3 testOilSpill.java
OilSpill.java

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

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

6. 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.

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

8. Read ALL the questions carefully. Do only what is required.

9. At the end of this examination session you will be required to hand in the stiffy disk or CD with all the files with the work you have done or you must make sure that ALL the files with your work have been saved on the network as explained to you by the invigilator/educator. Ensure that ALL files can be read.

10. During the examination you may use the HELP functions of the software. **Java candidates may make use of the Java API files**. You may NOT refer to any other resource material.

11. Make printouts of the programming code of ALL the questions. Make sure that your examination number appears in the first line of each program as a comment.

    Arrange the pages of each question in the correct order and then the questions from QUESTION 1 to 3. Staple all the printouts of the questions (arranged correctly) in one batch to hand in.

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

SCENARIO:

Pollution is becoming a serious problem worldwide. The questions in this question paper deal with various aspects of this issue.

SECTION A: (Answer ALL the questions in this section ONLY if you studied Delphi.)

QUESTION 1: DELPHI PROGRAMMING AND DATABASE

A study is underway to establish if pollution is a contributory factor in the occurrence of diseases in workers in different work environments. The database, **DiseasesDB.mdb**, which contains data related to this topic, has been supplied to you in a folder named **Question 1 Delphi**.

Two text files have been supplied as well. If you cannot use the database provided, use the text files named **Conditions.txt** and **Workplaces.txt** to create your own database named **DiseasesDB** containing two tables named **ConditionsTb** and **WorkplacesTb**. Change the data types and the sizes of the fields of the tables to the specifications given below. Create a one-to-many relationship between the two tables.

The **ConditionsTb** table stores data on patients suffering from various diseases. The fields in this table are defined as follows:

Field Name	Type	Size	
PatientID	Text	5	
CdtnName	Text	30	(Name of the illness)
CdtnType	Text	20	(Category of the illness)
Age	Number	Byte	
WorkplaceID	Text	10	
DateAdmitted	Date/Time	ShortDate	(First time admitted)
HoursPerDay	Number	Byte	(Working hours per day)

The following table is an example of the data contained in the table named **ConditionsTb** in the database named **DiseasesDB.mdb**.

PatientID	CdtnName	CdtnType	Age	WorkplaceID	DateAdmitted	HoursPerDay
AB11	TB (Tuberculosis)	Lung	55	Fac001	2007/06/07	8
AQ67	Pneumonia	Lung	34	Fac001	2007/06/12	9
BF79	Tumor	Brain	35	Fac002	2007/05/15	10
BG57	Nephritis	Kidney	56	Fac002	2007/02/23	11
BY47	Aneurysm	Brain Circulatory	46	Fac003	2007/07/15	7
CX98	Black Lung	Lung	40	Off002	2007/03/23	10
DC37	High Cholesterol	Heart Circulatory	23	Off001	2007/04/20	8
DF34	Migraine	Brain	34	Fac001	2007/05/12	9
DF45	Abscess	Brain Circulatory	65	Off002	2007/02/17	7
DG35	Ulcer	Circulatory	57	Fac004	2007/08/28	8
DG44	Bronchitis	Lung	45	Sch001	2007/06/21	7
DG45	Varicose veins	Circulatory	29	Off002	2007/07/29	8

:
:

The **WorkplacesTb** table stores data on the places where patients are working. The fields in this table are defined as follows:

Field Name	Type	Size	
WorkPlaceID	Text	6	
WorkType	Text	20	(Type of workplace)
Town	Text	20	
PollutionRiskLevel	Text	10	(Can be high, medium or low)

The following table is an example of the data contained in the table named **WorkplacesTb** in the database named **DiseasesDB.mdb.**

WorkplaceID	WorkType	Town	PollutionRiskLevel
Fac001	Factory	Sasolburg	HIGH
Fac002	Factory	Johannesburg	HIGH
Fac003	Factory	Benoni	HIGH
Fac004	Factory	Sasolburg	MEDIUM
Fac005	Factory	Johannesburg	LOW
Fac006	Factory	Randfontein	MEDIUM
Fac007	Factory	Boksburg	LOW
Fac008	Factory	Germiston	MEDIUM
Fac009	Factory	Germiston	HIGH
Fac010	Factory	Randfontein	HIGH
Min001	Mine	Carletonville	MEDIUM
Min002	Mine	Randfontein	LOW
Min003	Mine	Carletonville	HIGH
Off001	Office	Johannesburg	MEDIUM
Off002	Office	Johannesburg	LOW
Off003	Office	Benoni	LOW
Sch001	School	Meyerton	MEDIUM
Sch002	School	Johannesburg	LOW
Sch003	School	Germiston	HIGH
Sch004	School	Germiston	LOW

You have also been supplied with an incomplete Delphi program with a unit named **DiseasesDatabaseU** and a project named **DiseasesDatabaseP** in the folder named **Question 1 Delphi**. Open the incomplete program.

The program displays ten buttons as well as a DBGrid that will be used as an output component.

Do the following:

- Change the **captions** of the buttons from top to bottom to match the screenshot given below (FIGURE 1.1).

- Change the **names** of the buttons from top to bottom as follows: **btnSelectAll**, **btnSelectAge**, **btnUpdate**, **btnInsert**, **btnCondition**, **btnMiddleAge**, **btnSubsidy**, **btnCountHeart** and **btnNotJhb**.

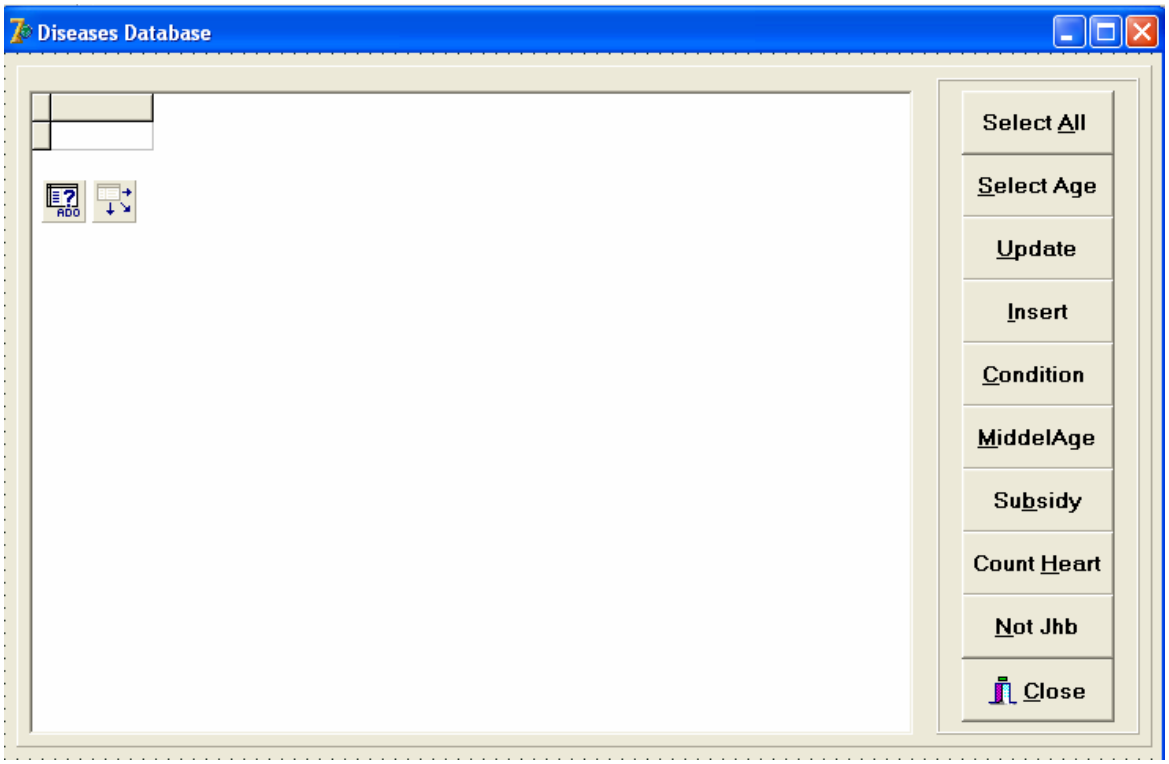


FIGURE 1.1

- The program should be able to connect to the database named **DiseasesDB.mdb**. When you do QUESTION 1.1 and you find that the connectivity is not in place, use the following steps to establish connection with the database:

    - Click on the ADOQuery component named **qryDiseases**.
    - Click on the Ellipse button (three dots) to the right of the Connection string property in the Object Inspector.
    - Click on the Build button which takes you to the Data Link Properties dialogue box.
    - Select Microsoft Jet 4.0 OLE DB Provider and click on Next.
    - The first option on the Connection tab sheet allows you to browse and find the **DiseasesDB.mdb** file.
    - Remove the user name Admin.
    - Click on the Test Connection button.
    - Click OK on each one of the open dialogue windows.

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

Marks will only be awarded for the programming code that contains the SQL statements in the unit named DiseasesDatabaseU.

- Complete the program by creating the necessary SQL statements in the buttons named **btnSelectAll**, **btnSelectAge**, **btnUpdate**, **btnInsert**, **btnCondition**, **btnMiddleAge**, **btnSubsidy**, **btnCountHeart** and **btnNotJhb** respectively as indicated in QUESTIONS 1.1 – 1.9.

1.1 Complete the code in the button named **btnSelectAll** by formulating an SQL statement to display all the fields in the **ConditionsTb** table sorted alphabetically according to the names of the diseases. Place the statement in the appropriate line of program code. Example of the output:

PatientID	CdtnName	CdtnType	Age	WorkplaceID	DateAdmitted	HoursPerDay
DF45	Abscess	Brain Circulatory	65	Off002	2007/02/17	7
FJ78	Alzheimers disease	Brain	68	Sch001	2007/03/16	7
NH78	Anemia	Kidney Circulatory	67	Fac009	2007/06/25	9
BY47	Aneurysm	Brain Circulatory	46	Fac003	2007/07/15	7
DK10	Angina	Heart	56	Fac004	2007/06/25	9
GG88	Arrhythmia	Heart	46	Sch001	2007/09/15	7
IO91	Arrhythmia	Heart	63	Off003	2007/08/15	10
ZN29	Asthma	Lung Respiratory	67	Fac006	2007/04/15	10
DR35	Asthma	Lung Respiratory	47	Fac004	2007/04/22	10
CX98	Black Lung	Lung	40	Off002	2007/03/23	10
VG57	Blood clot	Circulatory	76	Fac007	2007/01/26	9
GH67	Bornholm disease	Lung	46	Sch002	2007/09/21	6
DG44	Bronchitis	Lung	45	Sch001	2007/06/21	7
GF76	Bronchitis	Lung Respiratory	57	Sch002	2007/08/28	7
SX21	Bronchitis	Lung Respiratory	34	Fac006	2007/05/18	9
SD36	Buerger's Disease	Circulatory	24	Min002	2007/04/12	9
QW23	Collapsed Lung	Lung	37	Fac006	2007/06/21	10
EF56	Cyst	Kidney	45	Fac005	2007/05/24	10
SW21	Diabetes	Circulatory Brain	56	Fac008	2007/04/16	8
SF23	Dimentia	Brain	45	Min003	2007/06/12	10
GT66	Emphysema	Lung	56	Sch004	2007/05/12	7

(3)

1.2 Complete the code in the button named **btnSelectAge**, by formulating an SQL statement to display the **PatientID**, **CdtnName**, **CdtnType** and **Age** fields for patients who are over 50 years of age and who suffer from any type of lung disease. Place the statement in the appropriate line of program code. Example of the output:

PatientID	CdtnName	CdtnType	Age
AB11	TB (Tuberculosis)	Lung	55
FG29	TB (Tuberculosis)	Lung	56
FH67	TB (Tuberculosis)	Lung	53
GF76	Bronchitis	Lung Respiratory	57
GJ78	TB (Tuberculosis)	Lung	56
GT66	Emphysema	Lung	56
RF45	Lung Cancer	Lung	56
ZN29	Asthma	Lung Respiratory	67

(4)

1.3 Some of the entries in the field named **PollutionRiskLevel** in the **WorkplacesTb** table are entered as 'HIGH'. Complete the code in the button named **btnUpdate**, by formulating an SQL statement to replace all entries of 'HIGH' with 'SEVERE'. Place the statement in the appropriate line of program code. Display all the records and fields in the **WorkplacesTB** table after the change has been made. (3)

1.4 Complete the code in the button named **btnInsert**, by formulating an SQL statement to insert a new record into the **WorkplacesTb** table with the values 'Fac012', 'Factory', 'Sasolburg' and 'HIGH'. Place the statement in the appropriate line of program code. Display all the records and fields in the **WorkplacesTB** table after the new record has been added. (3)

1.5 Complete the code in the button named **btnCondition**, by asking the user to enter the type of condition (such as Lung) and the month that the patient was admitted (such as 6) as inputs. Formulate an SQL statement to display the **PatientID**, **CndtName**, **CdtnType** and **DateAdmitted** fields of the relevant patients. Place the statement in the appropriate line of program code.

Example of the output for the following input:
Lung as the type of condition and 6 as the month

PatientID	CdtnName	CdtnType	DateAdmitted
AB11	TB (Tuberculosis)	Lung	2007/06/07
AQ67	Pneumonia	Lung	2007/06/12
DG44	Bronchitis	Lung	2007/06/21
FH67	TB (Tuberculosis)	Lung	2007/06/23
QW23	Collapsed Lung	Lung	2007/06/21
QW40	TB (Tuberculosis)	Lung	2007/06/28

(6)

1.6 Complete the code in the button named **btnMiddleAge** by formulating an SQL statement to display the **PatientID**, **Age**, **WorkType** and **PollutionRiskLevel** fields of the patients within the age group between 30 and 45 years and who are working in a LOW- or MEDIUM-risk pollution area. Display readable headings. NB: You will need to link the tables with an appropriate ***where*** clause to be able to do this. Example of the output:

PatientID	Age	Type of workplace	PollutionRiskLevel
GF55	38	Office	LOW
CX98	40	Office	LOW
QW23	37	Factory	MEDIUM
QW40	34	Factory	MEDIUM
SX21	34	Factory	MEDIUM
DR45	34	School	LOW
GB83	37	Office	LOW
WE29	41	Factory	LOW

(6)

1.7 Complete the code in the button named **btnSubsidy** by formulating an SQL statement to list the **PatientID**, **Age**, **HoursPerDay** and **Subsidy** fields. **Subsidy** is a calculated field which reflects the contribution of the company towards the medical bill of the worker. The subsidy is calculated as follows: R100 times the working hours per day times the age of the person. The values in the **Subsidy** field must be neatly displayed including the currency. Example of the output:

PatientID	Age	HoursPerDay	Subsidy
AB11	55	8	R 44,000.00
AQ67	34	9	R 30,600.00
BF79	35	10	R 35,000.00
BG57	56	11	R 61,600.00
BY47	46	7	R 32,200.00
DC37	23	8	R 18,400.00
DF34	34	9	R 30,600.00
DF45	65	7	R 45,500.00
DG35	57	8	R 45,600.00

: (5)

1.8 Complete the code in the button named **btnCountHeart** by formulating an SQL statement that will determine and display the number of patients with heart conditions. Example of the output:

Total number of Patients with heart conditions
12

(4)

1.9 Complete the code in the button named **btnNOTJhb** by formulating an SQL statement that will list the **Town**, **CndtType** and **WorkType** fields of all the HIGH (or SEVERE) risk pollution areas outside Johannesburg. The headings should be user friendly. NB: You will need to link the tables with an appropriate ***where*** clause to be able to do this. Example of the output:

Town	Type of Condition	Type of work
Sasolburg	Lung	Factory
Sasolburg	Lung	Factory
Sasolburg	Brain	Factory
Sasolburg	Circulatory	Factory
Benoni	Brain Circulatory	Factory
Germiston	Heart Circulatrory	Factory
Germiston	Kidney Circulatory	Factory
Randfontein	Kidney Circulatory	Factory
Carletonville	Brain	Mine
Carletonville	Brain	Mine
Germiston	Circulatory	School
Germiston	Lung	School
Germiston	Heart	School

(6)

- Enter your examamination number as a comment in the first line of the **DiseasesDatabaseU** unit containing the SQL statements.
- Save the unit named **DiseasesDatabaseU** and the project named **DiseasesDatabaseP (File/Save All)**.
- Rename the folder **Question 1 Delphi** to **Quest1_X**, where X should be replaced with your examination number.
- Make a printout of the code in the **DiseasesDatabaseU** unit to hand in.

[40]

QUESTION 2: DELPHI PROGRAMMING

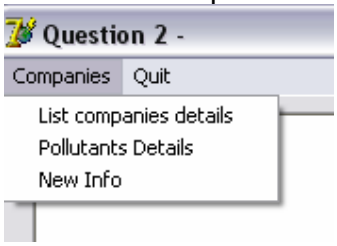
This question is intended to test object-oriented programming skills. You are required to produce a solution that includes all classes specified in the instructions. No marks will be allocated to any alternative solution such as one program not creating an object.

You are asked to examine the contents of a text file named **Pollution.txt** saved in a folder named **Question 2 Delphi**. The text file contains the names of the companies, as well as the pollution levels of three pollutants at each company, separated by the # character. The pollutants are carbon dioxide, lead and mercury (in this order). The content of the text file is shown below.

```
Alpha Corps#6#0#3
Beta Co#3#1#5
Delta Max#2#0#1
:
```

Do the following:

- Rename the folder **Question 2 Delphi** as **Quest2_X** (where X represents your examination number).
- Open Delphi and then open the file **testCompany program** in the folder **Quest2_X**.
- Go to File/Save As… and save the unit as **testCompany_Uxxxx** (where xxxx represents the last FOUR digits of your examination number).
- Go to File/Save Project As… and save the project as **testCompany_Pxxxx** (where xxxx represents the last FOUR digits of your examination number).
- Change the caption properties of the different options of the Menu component so that it corresponds with the figure shown below.



- Add your examination number to the caption of the form to the right of 'Question 2 –'.

2.1 Create an object class (another unit) named **CompanyXXXX** and save this unit as **CompanyXXXX** in your **Quest2_X** folder. (XXXX should be replaced by the last four digits of your examination number.) Write the following code as part of this class:

2.1.1 Define a class named **TCompany**. This class must contain the following private fields:

Company name
Level of carbon dioxide (CO_2)
Level of lead (Pb)
Level of mercury (Hg)

Ensure that you choose appropriate data types for these fields.
(4 ÷ 2) (2)

2.1.2 Create a parameterised constructor named **create** that will pass values for the fields in the class. These parameters should be used to initialise the fields of the class. (4 ÷ 2) (2)

2.1.3 Write a method of type string (a string function) named **toString** that returns information on a company in one string formatted as follows:

Name of company Carbon dioxide: x Lead: x Mercury: x

Example of return strings for the first two companies in the text file **Pollution.txt**:

Alpha Corps Carbon dioxide: 6 Lead: 0 Mercury: 3
Beta Co Carbon dioxide: 3 Lead: 1 Mercury: 5 (6 ÷ 2) (3)

2.1.4 Write a method to calculate the pollution factor of a company as follows:

PollutionFactor = CO_2_level + (2 * Pb_level) + (3* Hg_level)
(4 ÷ 2) (2)

2.1.5 Write a method to determine which one of the three pollutants is the highest in the company. Assume that at least one of the pollutants will have a positive value greater than 0.
(10 ÷ 2) (5)

2.1.6 Write a method that will receive three new values for the three pollutants at a company and change the current values of the pollutants to the new values. (6 ÷ 2) (3)

2.1.7 Write a method that will return the name of the company. (2 ÷ 2) (1)

2.2 Do the following in the **testCompany_Uxxxx** file (the main unit) in the given program:

2.2.1 Create an array named **arrComp** that keeps objects of **TCompany**. Write code in the **OnActivate Eventhandler** of the form to read information from the text file **Pollution.txt** according to the following steps:

(a) Open the text file and initialise a loop to read the data.

(b) Read a line of text from the text file.

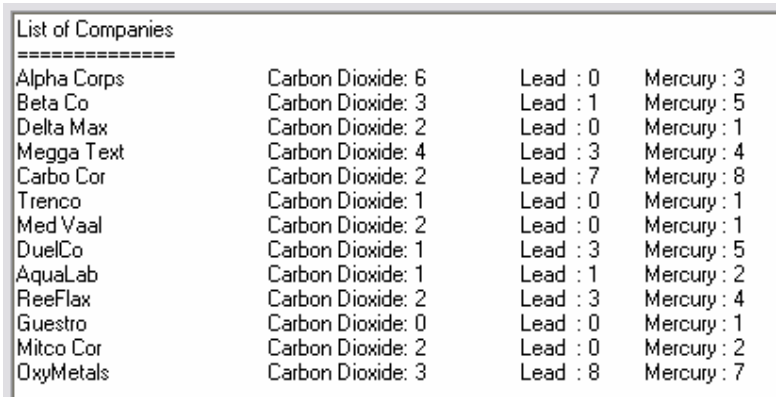
(c) Separate the text into the name of the company and the three pollutant figures.

(d) Use this information to create a new **TCompany** object and place the object into the array.

(e) Use a counter field to keep track of how many items there are in the array. (16 ÷ 2) (8)

2.2.2 Write code to complete the following options on the menu provided in the program. The methods in the **TCompany** class should be used where applicable. Invoke (Call) the relevant methods (procedures and functions) from the class.

List companies details: Display the names of the companies as well as the pollution levels of the three pollutants for each company. Call the **toString** method to display the information. Display a suitable heading. Example of the output:

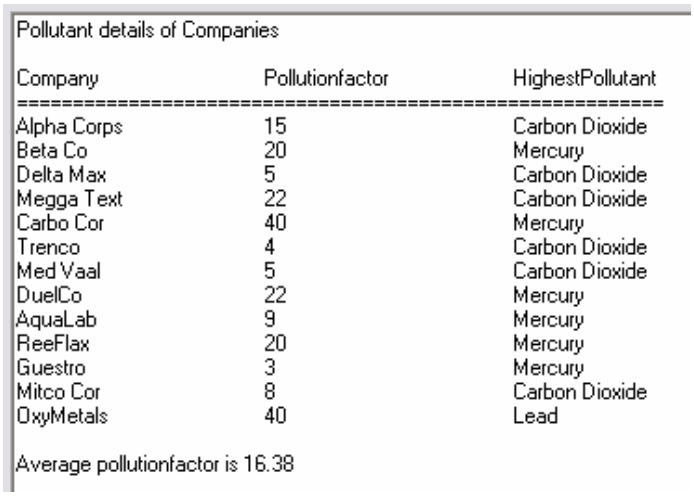


```
List of Companies
==============
Alpha Corps     Carbon Dioxide: 6    Lead : 0    Mercury : 3
Beta Co         Carbon Dioxide: 3    Lead : 1    Mercury : 5
Delta Max       Carbon Dioxide: 2    Lead : 0    Mercury : 1
Megga Text      Carbon Dioxide: 4    Lead : 3    Mercury : 4
Carbo Cor       Carbon Dioxide: 2    Lead : 7    Mercury : 8
Trenco          Carbon Dioxide: 1    Lead : 0    Mercury : 1
Med Vaal        Carbon Dioxide: 2    Lead : 0    Mercury : 1
DuelCo          Carbon Dioxide: 1    Lead : 3    Mercury : 5
AquaLab         Carbon Dioxide: 1    Lead : 1    Mercury : 2
ReeFlax         Carbon Dioxide: 2    Lead : 3    Mercury : 4
Guestro         Carbon Dioxide: 0    Lead : 0    Mercury : 1
Mitco Cor       Carbon Dioxide: 2    Lead : 0    Mercury : 2
OxyMetals       Carbon Dioxide: 3    Lead : 8    Mercury : 7
```

(4 ÷ 2) (2)

Pollutants Details: Display the detailed information on the pollution factor as well as the highest pollutant at each company. Display a suitable heading and subheadings. Calculate and display the average pollution factor of the companies.

Example of the output:



Pollutant details of Companies

Company	Pollutionfactor	HighestPollutant
Alpha Corps	15	Carbon Dioxide
Beta Co	20	Mercury
Delta Max	5	Carbon Dioxide
Megga Text	22	Carbon Dioxide
Carbo Cor	40	Mercury
Trenco	4	Carbon Dioxide
Med Vaal	5	Carbon Dioxide
DuelCo	22	Mercury
AquaLab	9	Mercury
ReeFlax	20	Mercury
Guestro	3	Mercury
Mitco Cor	8	Carbon Dioxide
OxyMetals	40	Lead

Average pollutionfactor is 16.38

(10 ÷ 2) (5)

New Info: Allow the user to enter the name of a company. Code an effective way to search for the name in the array by stopping the loop as soon as the name has been found.
Ask the user to enter the new values for carbon dioxide, lead and mercury for the company.
Display a message indicating the information has been updated or that the company has not been found. The name of the company must be part of the message. (20 ÷ 2) (10)

- Enter your examination number as a comment in the first line of the main unit (**testCompany_Uxxxx**) as well as the object unit (**CompanyXXXX**).
- Save the files (File/Save All).
- Make printouts of the code of the two units (**testCompany_Uxxxx** as well as **CompanyXXXX**) to hand in.

[43]

QUESTION 3: DELPHI PROGRAMMING

You have been asked to write a program that demonstrates how it might be possible to **simulate** an oil spill on the open sea. A two-dimensional array is used to represent an oil spill. The user will be asked to enter some information and thereafter characters are placed randomly into the 2D array representing the oil spill. The content of the array will be displayed as shown in FIGURE 3.1 below.

The "-" character indicates the open sea. The "+" character indicates oil.

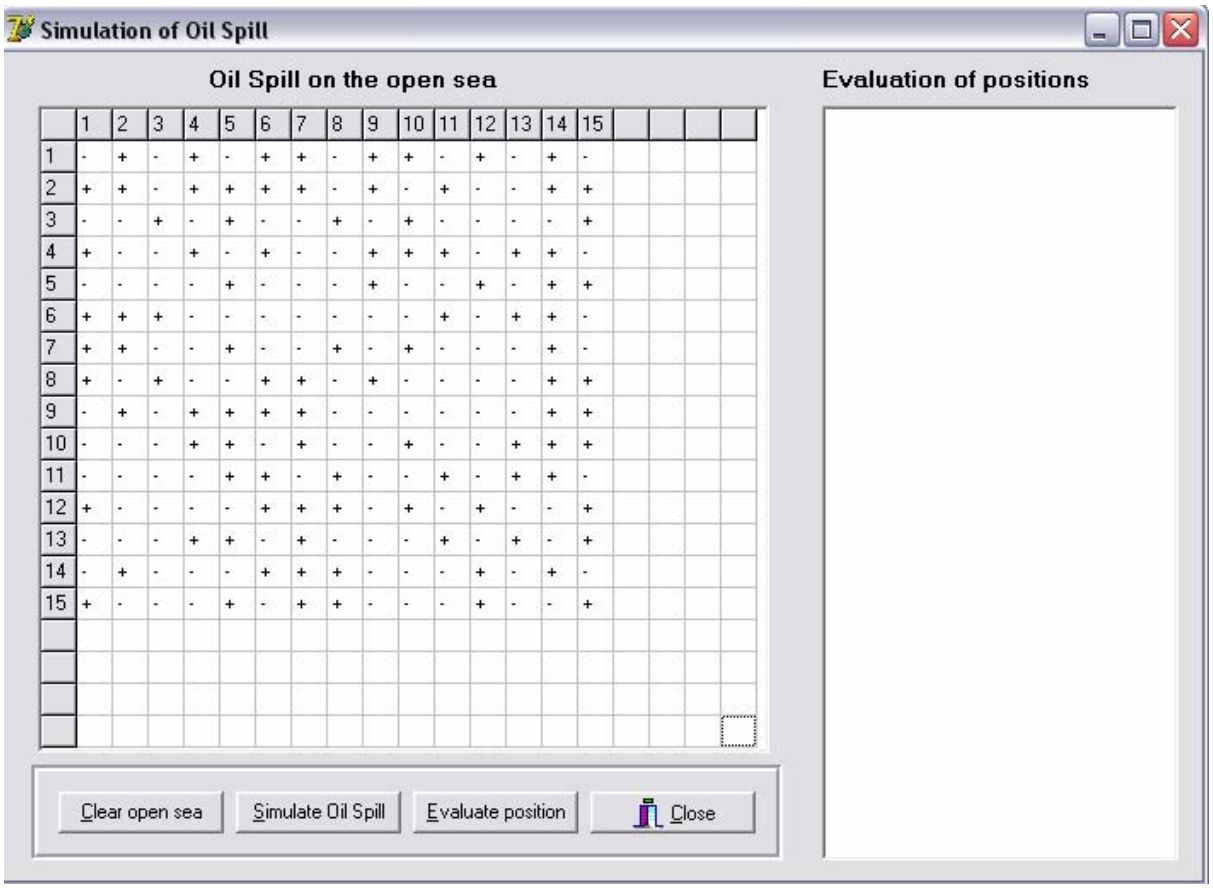


FIGURE 3.1

You have been supplied with an incomplete program in the folder named **Question 3 Delphi** to simulate an oil spill. The program displays a **stringGrid** and a **RichEdit** as output components as well as four buttons as indicated in FIGURE 3.1.

Do the following:

- Rename the folder named **Question 3 Delphi** to **Quest3_X**. Replace X with your examination number.
- Open the project in this folder.
- Save the unit as **OilSpillUxxxx** (File/Save As...) and the project as **OilSpillPxxxx** (File/Save Project As...) inside the folder (xxxx should be replaced with the last four digits of your examination number).
- Change the **captions** of the buttons to match those of the buttons in FIGURE 3.1.

You are required to do the following:

3.1 Declare a two-dimensional array and a size variable for the number of rows and columns of the array. The maximum number of rows and columns of the array will be 20. The number of rows and columns of the array (and the grid) will be the same.

Write a procedure named **clearSea** to initialise the 2D array to contain characters representing the open sea ("-") and to clear the stringGrid as well as the RichEdit. (8 ÷ 2) (4)

3.2 Write a procedure named **DisplayGrid** that will display the contents of the 2D array in the format indicated in FIGURE 3.1. Code must be included to display the numbering of the rows and columns according to the size of the array. The size of the grid will vary according to the value for the size of the array entered by the user in the Simulate Oil Spill button. A suitable heading should be displayed. (8 ÷ 2) (4)

3.3 Write a function that can be used to validate a value. The function must receive the lower boundary, the upper boundary and the value that should be validated. The function should return a Boolean value indicating whether the value is valid or not. Use this function whenever input values should be validated. (10 ÷ 2) (5)

3.4 In the **Simulate Oil Spill** button:

3.4.1 Request the user to enter a suitable size for the 2D array. This entry must be a value between 10 and 20 (both included). The valid boundaries of the value must be indicated to the user. Remember that the number of rows and columns of the array should be the same. The program should not continue until the input value is within the requested range. Use the function written in QUESTION 3.3 to validate the input value. Call the **clearSea** procedure to initialise the array with characters to represent the open sea ("-"). (8 ÷ 2) (4)

3.4.2 Request the user to enter the level of seriousness of the spill in the range of 1 to 10. The valid boundaries of the value must be indicated to the user. The program should not continue until the input value is within the requested range. Use the function written in QUESTION 3.3 to validate the input value. The level of seriousness of the spill affects the number of "-" characters in the 2D array that will be changed to "+" characters in the following way:

If the user enters a level of seriousness of 2, then 20 (2 x 10) random positions should be changed to "+" characters. If the user enters 3, then 30 (3 x 10) positions should be changed, and so on.

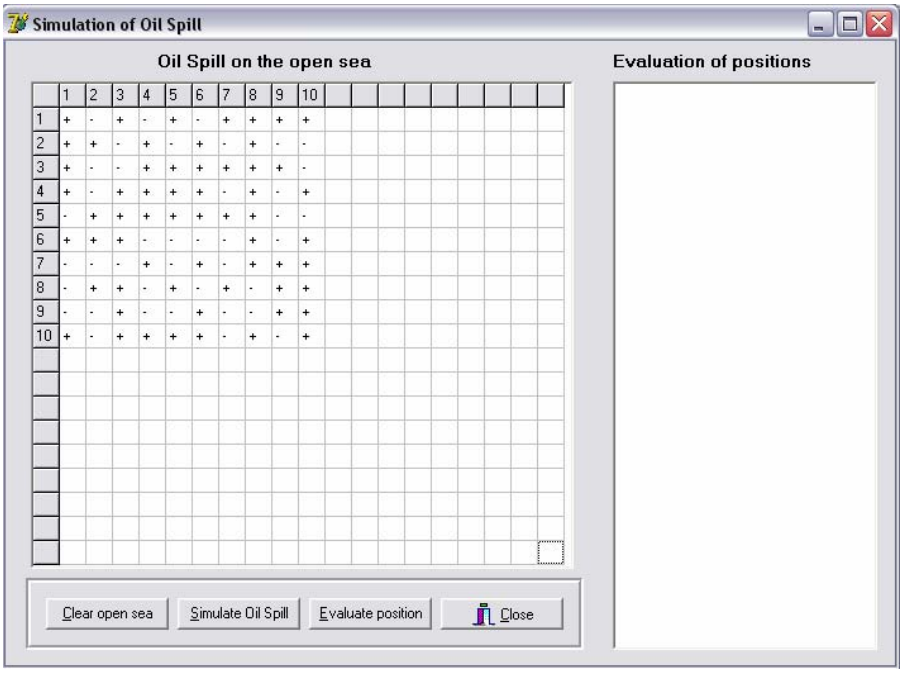
NOTE:

- The "+" characters should only be placed in positions that are not polluted yet. If a random position in the array is already occupied by a "+", another random position should be generated until an available position ("-") has been found.
- Positions should not be chosen outside the array boundaries.
- Call the procedure **DisplayGrid** to display the content of the 2D-array.
- Enable the Evaluate button since evaluation can only take place once the oil spill has taken place.

Examples of the output of test runs:

The following input values were used for the first test run:

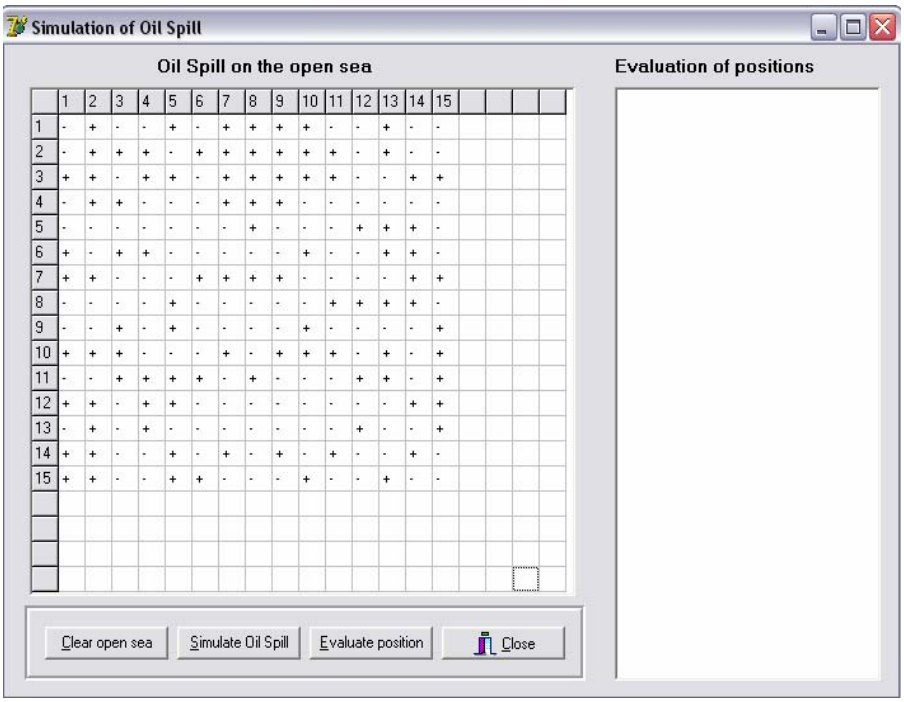
Size of the grid: 10
Level of seriousness of the oil spill: 6



NOTE: The positions of "+" characters displayed will vary because of the random selection of oil spill positions.

The following input values were used for the second test run:

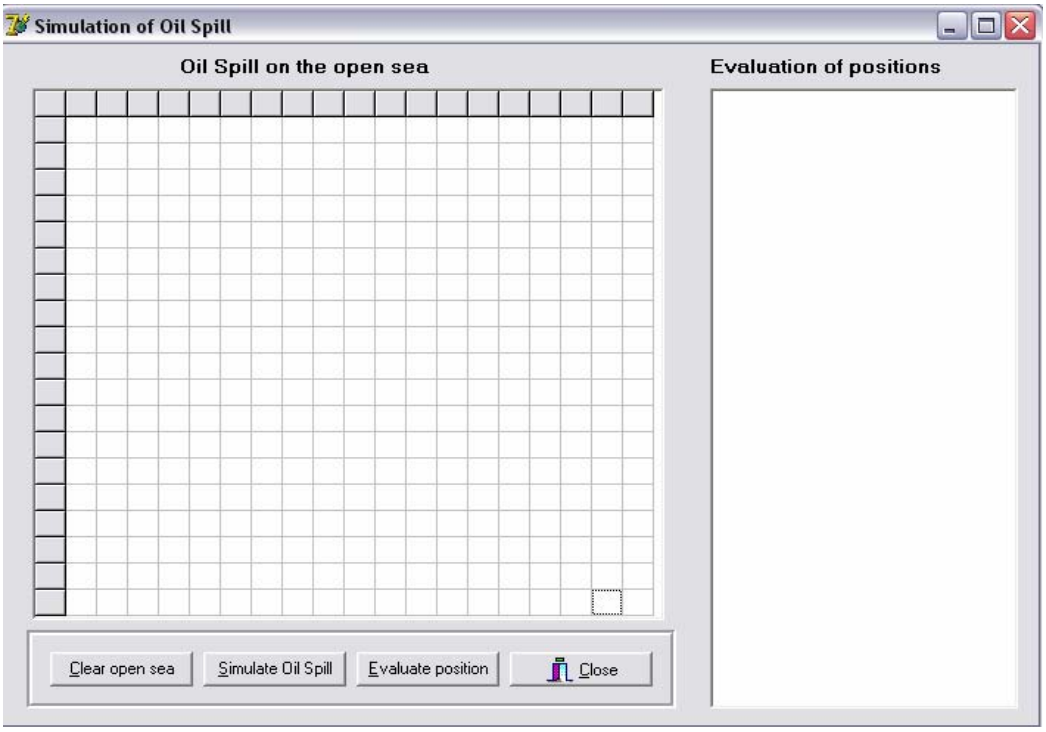
Size of the grid: 15
Level of seriousness of the oil spill: 10



(14 ÷ 2) (7)

3.5 In the **Clear Open Sea** button:
Call the relevant procedures to:

- initialise the 2D array to contain characters representing the open sea ("-") and clear the output components.
- display the **stringGrid** component as well as the **RichEdit** component. Example of the output:



(2 ÷ 2) (1)

3.6 Do the following in the **Evaluate position** button:

3.6.1 Request the user to enter two values (row and column) representing a ship at sea. The user must be requested to enter values that do not exceed the size of the array. Use the function written in QUESTION 3.3 to validate the values. The program should only continue when the input values are valid (within the boundaries of the array). (8 ÷ 2) (4)

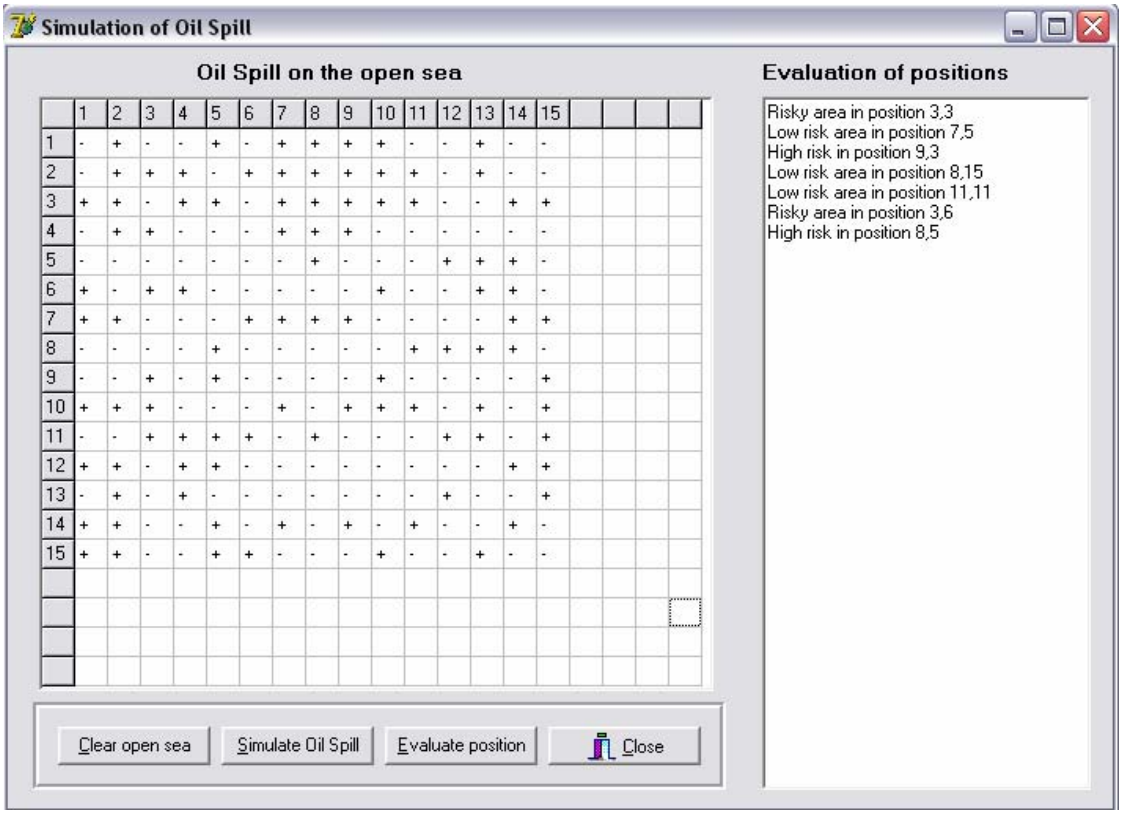
3.6.2 Write code to evaluate the position entered by the user, according to the rules that follow. A message should be displayed indicating whether the position is situated in a high risk, risky or low risk area. The message should also indicate the position entered. Display the message in the RichEdit component. The risk level of the area is determined as follows:

- 'High risk' area: The position entered by the user is occupied by a "+" in the 2D array.
- 'Risky' area: The position entered by the user is occupied by a "-". 4 or more of the spots surrounding the selected position is polluted ("+"). The selected position is regarded as a 'Risky' area. The following positions in FIGURE 3.2 are examples of 'Risky' areas:
  row 1 and column 2 (four "+" characters surrounding this position),
  row 4 and column 7 (seven "+" characters surrounding this position),
  row 2 and column 8 (five "+" characters surrounding this position),
  row 6 and column 10 (four "+" characters surrounding this position), et cetera.
- 'Low risk' area: The position entered by the user is occupied by a "-". Less than 4 of the spots surrounding the selected position are polluted. The selected position is regarded as a 'Low risk' area. The following positions in FIGURE 3.2 are examples of 'Low risk' areas:
  row 1 and column 3 (three "+" characters surrounding this position),
  row 3 and column 3 (three "+" characters surrounding this position),
  row 8 and column 1 (three "+" characters surrounding this position),
  row 3 and column 10 (two "+" characters surrounding this position), et cetera.

	1	2	3	4	5	6	7	8	9	10
1	+	-	-	-	+	+	-	+	+	+
2	+	+	+	+	+	+	+	-	-	+
3	+	-	-	-	+	+	+	+	-	-
4	+	-	-	-	+	+	-	+	-	+
5	-	+	+	+	-	+	+	-	+	+
6	+	-	+	-	+	-	+	+	+	-
7	+	+	+	-	+	+	+	+	+	-
8	-	-	+	-	+	-	-	-	+	+
9	+	-	-	+	+	+	-	-	+	+
10	-	-	+	-	-	+	+	-	+	+

FIGURE 3.2

The user must be able to evaluate as many positions as he/she wants in one specific oil spill simulation.
Example of the output when testing a few positions in an oil spill:



(16 ÷ 2) (8)

- Enter your examination number as a comment in the first line of the unit **OilSpill_Uxxxx**.
- Save the unit and the project (File/Save All).
- Make a printout of the code in the unit **OilSpill_Uxxxx** to hand in. **[37]**

TOTAL SECTION A: 120

SECTION B: (Answer ALL the questions in this section ONLY if you studied Java.)

QUESTION 1: JAVA PROGRAMMING AND DATABASE

A study is underway to establish if pollution is a contributory factor in the occurrence of diseases in workers in different work environments. The database, **DiseasesDB.mdb**, which contains data related to this topic, has been supplied to you in a folder named **Question 1 Java**.

Two text files have been supplied as well. If you cannot use the database provided, use the text files named **Conditions.txt** and **Workplaces.txt** to create your own database named **DiseasesDB** containing two tables named **ConditionsTb** and **WorkplacesTb**. Change the data types and the sizes of the fields of the tables to the specifications given below. Create a one-to-many relationship between the two tables.

The **ConditionsTb** table stores data on patients suffering from various diseases. The fields in this table are defined as follows:

Field Name	**Type**	**Size**	
PatientID	Text	5	
CdtnName	Text	30	(Name of the illness)
CdtnType	Text	20	(Category of the illness)
Age	Number	Byte	
WorkplaceID	Text	10	
DateAdmitted	Date/Time	ShortDate	(First time admitted)
HoursPerDay	Number	Byte	(Working hours per day)

The following table is an example of the data contained in the table named **ConditionsTb** in the database named **DiseasesDB.mdb**.

PatientID	CdtnName	CdtnType	Age	WorkplaceID	DateAdmitted	HoursPerDay
AB11	TB (Tuberculosis)	Lung	55	Fac001	2007/06/07	8
AQ67	Pneumonia	Lung	34	Fac001	2007/06/12	9
BF79	Tumor	Brain	35	Fac002	2007/05/15	10
BG57	Nephritis	Kidney	56	Fac002	2007/02/23	11
BY47	Aneurysm	Brain Circulatory	46	Fac003	2007/07/15	7
CX98	Black Lung	Lung	40	Off002	2007/03/23	10
DC37	High Cholesterol	Heart Circulatory	23	Off001	2007/04/20	8
DF34	Migraine	Brain	34	Fac001	2007/05/12	9
DF45	Abscess	Brain Circulatory	65	Off002	2007/02/17	7
DG35	Ulcer	Circulatory	57	Fac004	2007/08/28	8
DG44	Bronchitis	Lung	45	Sch001	2007/06/21	7
DG45	Varicose veins	Circulatory	29	Off002	2007/07/29	8

:
:

The **WorkplacesTb** table stores data on the places where patients are working. The fields in this table are defined as follows:

Field Name	Type	Size	
WorkPlaceID	Text	6	
WorkType	Text	20	(Type of workplace)
Town	Text	20	
PollutionRiskLevel	Text	10	(Can be high, medium or low)

The following table is an example of the data contained in the table named **WorkplacesTb** in the database named **DiseasesDB.mdb**.

WorkplaceID	WorkType	Town	PollutionRiskLevel
Fac001	Factory	Sasolburg	HIGH
Fac002	Factory	Johannesburg	HIGH
Fac003	Factory	Benoni	HIGH
Fac004	Factory	Sasolburg	MEDIUM
Fac005	Factory	Johannesburg	LOW
Fac006	Factory	Randfontein	MEDIUM
Fac007	Factory	Boksburg	LOW
Fac008	Factory	Germiston	MEDIUM
Fac009	Factory	Germiston	HIGH
Fac010	Factory	Randfontein	HIGH
Min001	Mine	Carletonville	MEDIUM
Min002	Mine	Randfontein	LOW
Min003	Mine	Carletonville	HIGH
Off001	Office	Johannesburg	MEDIUM
Off002	Office	Johannesburg	LOW
Off003	Office	Benoni	LOW
Sch001	School	Meyerton	MEDIUM
Sch002	School	Johannesburg	LOW
Sch003	School	Germiston	HIGH
Sch004	School	Germiston	LOW

You have also been supplied with an incomplete Java program in the folder named **Question 1 Java** containing the files named **DiseasesDatabase.java** and **TestDatabase.java**. Open the incomplete program and run it.

The program displays a simple text menu with ten options.

Do the following:

- Change the options on the menu from top to bottom to match the screenshot given below (FIGURE 1.1).

```
    MENU

A - Display sorted records
B - Display Lung Conditions in over 50 age group
C - Change 'HIGH' pollution level entry to 'SEVERE'
D - Insert one new record
E - Display certain conditions
F - Display middle age group
G - Display subsidy
H - Count heart conditions
I - Display Low pollution area outside Jhb
Q - QUIT
Your Choice -
```

FIGURE 1.1

- Change the names of the methods in the two Java files as follows:

method A to selectAllQuery(),
method B to selectAgeQuery(),
method C to updateQuery(),
method D to insertQuery(),
method E to getConditionQuery(),
method F to getMiddleAgeQuery(),
method G to getSubsidyQuery(),
method H to countHeartQuery(),
method I to getNotJhbQuery().

- The connectivity code has already been written as part of the given code in the file named **DiseasesDatabase.java**. When you run the program, you have to enter the exact path where the database **DiseasesDB.mdb** has been stored.

  HINT: Copy the database into the root directory of the drive that you are working on. The path to enter should be short, for example E:/DiseasesDB.mdb.

  HINT: Instead of entering the path every time that you run the program, you can change the input to a constant string containing the exact location of the database, for example

  String file name = 'E:/DiseasesDB.mdb'

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

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

- Complete the programming code in the file named **DiseasesDatabase.java** by creating the necessary SQL statements in the methods named **selectAllQuery()**, **selectQuery()**, **updateQuery()**, **insertQuery()**, **getConditionQuery()**, **getMiddleAgeQuery()**, **getSubsidyQuery()**, **countHeartQuery()** and **getNotJhbQuery()** respectively as indicated in QUESTIONS 1.1 – 1.9.

1.1 Complete the code in the method named **selectAllQuery()** by formulating an SQL statement to display all the fields in the **ConditionsTb** table sorted alphabetically according to the names of the diseases. Place the statement in the appropriate line of program code. Example of the output:

PatientID	Condition Name	Type of condition	Age	WorkPlaceID	DateSubmitted	Hours per Days
DF45	Abscess	Brain Circulatory	65	Off002	2007-02-17	7
FJ78	Alzheimers disease	Brain	68	Sch001	2007-03-16	7
NH78	Anemia	Kidney Circulatory	67	Fac009	2007-06-25	9
BY47	Aneurysm	Brain Circulatory	46	Fac003	2007-07-15	7
DK10	Angina	Heart	56	Fac004	2007-06-25	9
GG88	Arrhythmia	Heart	46	Sch001	2007-09-15	7
IO91	Arrhythmia	Heart	63	Off003	2007-08-15	10
ZN29	Asthma	Lung Respiratory	67	Fac006	2007-04-15	10
DR35	Asthma	Lung Respiratory	47	Fac004	2007-04-22	10
CX98	Black Lung	Lung	40	Off002	2007-03-23	10
VG57	Blood clot	Circulatory	76	Fac007	2007-01-26	9
GH67	Bornholm disease	Lung	46	Sch002	2007-09-21	6
DG44	Bronchitis	Lung	45	Sch001	2007-06-21	7
GF76	Bronchitis	Lung Respiratory	57	Sch002	2007-08-28	7
SX21	Bronchitis	Lung Respiratory	34	Fac006	2007-05-18	9
SD36	Buerger's Disease	Circulatory	24	Min002	2007-04-12	9
QW23	Collapsed Lung	Lung	37	Fac006	2007-06-21	10
EF56	Cyst	Kidney	45	Fac005	2007-05-24	10
SW21	Diabetes	Circulatory Brain	56	Fac008	2007-04-16	8
SF23	Dimentia	Brain	45	Min003	2007-06-12	10

: (3)

1.2 Complete the code in the method named **selectAgeQuery()** by formulating an SQL statement to display the **PatientID**, **CdtnName**, **CdtnType** and **Age** fields for patients who are over 50 years of age and who suffer from any type of lung disease. Place the statement in the appropriate line of program code. Example of the output:

PatientID	Condition Name	Condition Type	Age
AB11	TB (Tuberculosis)	Lung	55
FG29	TB (Tuberculosis)	Lung	56
FH67	TB (Tuberculosis)	Lung	53
GF76	Bronchitis	Lung Respiratory	57
GJ78	TB (Tuberculosis)	Lung	56
GT66	Emphysema	Lung	56
RF45	Lung Cancer	Lung	56
ZN29	Asthma	Lung Respiratory	67

(4)

1.3 Some of the entries in the field named **PollutionRiskLevel** are entered as 'HIGH' in the **WorkPlacesTB** table. Complete the code in the method named **updateQuery()** by formulating an SQL statement to replace all entries of 'HIGH' with 'SEVERE'. Place the statement in the appropriate line of program code. Display all the records and fields in the **WorkplacesTb** table after the change has been made. (3)

1.4 Complete the code in the method named **insertQuery()** by formulating an SQL statement to insert a new record into the **WorkplacesTB** table with the values 'Fac012', 'Factory', 'Sasolburg' and 'HIGH'. Place the statement in the appropriate line of program code. Display all the records and fields in the **WorkplacesTb** table after the new record has been added. (3)

1.5 Complete the code in the method named **getConditionQuery**, by asking the user to enter the type of condition (such as Lung) and the month that the patient was admitted (such as 6) as inputs. Formulate an SQL statement to display the **PatientID**, **CndtName**, **CndtType** and **DateAdmitted** fields of the relevant patients. Place the statement in the appropriate line of program code.

Example of the output:
Input: Lung as the type of condition and 6 as the month

PatientID	Condition Name	Condition Type	Date Admitted
AB11	TB (Tuberculosis)	Lung	2007-06-07
AQ67	Pneumonia	Lung	2007-06-12
DG44	Bronchitis	Lung	2007-06-21
FH67	TB (Tuberculosis)	Lung	2007-06-23
QW23	Collapsed Lung	Lung	2007-06-21
QW40	TB (Tuberculosis)	Lung	2007-06-28

(6)

1.6 Complete the code in the method named **getMiddleAgeQuery** by formulating an SQL statement to display the **PatientID**, **Age**, **WorkType** and **PollutionRiskLevel** fields of the patients within the age group between 30 and 45 years and who are working in a LOW- or MEDIUM-risk pollution area. Display readable headings. NB: You will need to link the tables with an appropriate ***where*** clause to be able to do this. Example of the output:

PatientID	Age	WorkType	PollutionRiskLevel
GF55	38	Office	LOW
CX98	40	Office	LOW
QW23	37	Factory	MEDIUM
QW40	34	Factory	MEDIUM
SX21	34	Factory	MEDIUM
DR45	34	School	LOW
GB83	37	Office	LOW
WE29	41	Factory	LOW

(6)

1.7 Complete the code in the method named **getSubsidyQuery** to list the **PatientID**, **HoursPerDay**, **Age** and **Subsidy** fields. **Subsidy** is a calculated field which reflects the contribution of the company towards the medical bill of the worker. The subsidy is calculated as follows: R100 times the working hours per day times the age of the person. The values in the **Subsidy** field must be neatly displayed, including the currency. Example of the output:

PatientID	Age	HoursPerDay	Subsidy
AB11	55	8	R 44,000.00
AQ67	34	9	R 30,600.00
BF79	35	10	R 35,000.00
BG57	56	11	R 61,600.00
BY47	46	7	R 32,200.00
DC37	23	8	R 18,400.00
DF34	34	9	R 30,600.00
DF45	65	7	R 45,500.00
DG35	57	8	R 45,600.00
DG44	45	7	R 31,500.00
DG45	29	8	R 23,200.00
DJ76	67	7	R 46,900.00
DK10	56	9	R 50,400.00
DR35	47	10	R 47,000.00
DR45	34	7	R 23,800.00
DR55	45	8	R 36,000.00

: (5)

1.8 Complete the code in the method named **countHeartQuery** by formulating an SQL statement that will determine and display the number of patients with heart conditions. Example of the output:

```
Number of patients with heart conditions: 12
```

(4)

1.9 Complete the code in the method named **getNOTJhbQuery** by formulating an SQL statement that will list the **Town, CndtType** and **WorkType** fields of all the HIGH (or SEVERE) risk pollution areas outside Johannesburg. NB: You will need to link the tables with an appropriate ***where*** clause to be able to do this. Example of the output:

Town	Type of Condition	Type of Workplace
Sasolburg	Lung	Factory
Sasolburg	Lung	Factory
Sasolburg	Brain	Factory
Sasolburg	Circulatory	Factory
Benoni	Brain Circulatory	Factory
Germiston	Heart Circulatory	Factory
Germiston	Kidney Circulatory	Factory
Randfontein	Kidney Circulatory	Factory
Carletonville	Brain	Mine
Carletonville	Brain	Mine
Germiston	Circulatory	School
Germiston	Lung	School
Germiston	Heart	School

(6)

- Enter your examination number as a comment in the first line of the file named **DiseasesDatabase.java** containing the SQL statements.
- Save the **TestDatabase.java** and the **DiseasesDatabase.java** files.
- Rename the folder **Question 1 Java** to **Quest1_X**, where X should be replaced with your examination number.
- Make a printout of the code of the **DiseasesDatabase** file to hand in.

[40]

QUESTION 2: JAVA PROGRAMMING

This question is intended to test object-oriented programming skills. You are required to produce a solution that includes all classes specified in the instructions. No marks will be allocated to any alternative solution such as one program not creating an object.

You are asked to examine the contents of a text file named **Pollution.txt** saved in a folder named **Question 2 Java**. The text file contains the names of the companies, as well as the pollution levels of three pollutants at each company, separated by the # character. The pollutants are carbon dioxide, lead and mercury (in this order). The content of the text file is shown below.

```
Alpha Corps#6#0#3
Beta Co#3#1#5
Delta Max#2#0#1
:
```

Do the following:

- Rename the folder **Question 2 Java** as **Quest2_X** (where X represents your examination number).
- Open the **testCompany** file (class) in the folder **Quest2_X**.
- Change the options on the menu so that it corresponds with the menu shown below.

```
    Menu

A - List All Companies
B - Pollutants Details
C - New Info
Q - QUIT

Your Choice? :
```

- Add your examination number as a comment in the first line of the **testCompany** class.

2.1 Create an object class named **CompanyXXXX.java** and save this file as **CompanyXXXX.java** in your **Quest2_X** folder. (XXXX should be replaced by the last four digits of your examination number.) This class should include code to do the following:

2.1.1 Define the following private fields:

Company name
Level of carbon dioxide (CO_2)
Level of lead (Pb)
Level of mercury (Hg)

Ensure that you choose appropriate data types for these fields. (4 ÷ 2) (2)

2.1.2 Create a parameterised constructor named **create** that will pass values for the fields in the class. These parameters should be used to initialise the fields of the class. (4 ÷ 2) (2)

2.1.3 Write a method of type string named **toString** that returns information on a company in one string formatted as follows:

Name of company Carbon dioxide: x Lead: x Mercury: x

Example of return strings for the first two companies in the **Polution.txt** text file:

Alpha Corps Carbon dioxide: 6 Lead: 0 Mercury: 3
Beta Co Carbon dioxide: 3 Lead: 1 Mercury: 5 (6 ÷ 2) (3)

2.1.4 A method to calculate the pollution factor of a company as follows:

PollutionFactor = CO_2_level + (2 * Pb_level) + (3* Hg_level)
(4 ÷ 2) (2)

2.1.5 Write a method to determine which one of the three pollutants is the highest at the company. Assume that at least one of the pollutants will have a positive value greater than 0. (10 ÷ 2) (5)

2.1.6 Write a method that will receive three new values for the three pollutants of a company and change the current values of the pollutants of the object to the new values. (6 ÷ 2) (3)

2.1.7 Write a method that will return the name of the company. (2 ÷ 2) (1)

2.2 Do the following in the **testCompany** class in the given program:

2.2.1 Create an array named **arrComp** that keeps objects of **CompanyXXXX**. Write code to read information from the text file **Polution.txt** according to the following steps:

(a) Open the text file and initialise a loop to read the data.

(b) Read a line of text from the text file.

(c) Separate the text into the name of the company and the three pollutant figures.

(d) Use this information to create a new **CompanyXXXX** object and place the object into the array.

(e) Use a counter field to keep track of how many items there are in the array. (16 ÷ 2) (8)

2.2.2 Write code to complete the options on the menu provided in the program as indicated below. The methods in the **CompanyXXXX** class should be used where applicable. Invoke (Call) the relevant methods from the class.

List All Companies: Display the names of the companies as well as the pollution levels of the three pollutants for each company. Call the **toString** method to display the information. Display a suitable heading. Example of the output:

```
List of All Companies
=====================
Alpha Corps             Carbon Dioxide: 6       Lead: 0         Mercury: 3
Beta Co                 Carbon Dioxide: 3       Lead: 1         Mercury: 5
Delta Max               Carbon Dioxide: 2       Lead: 0         Mercury: 1
Megga Text              Carbon Dioxide: 4       Lead: 3         Mercury: 4
Carbo Cor               Carbon Dioxide: 2       Lead: 7         Mercury: 8
Trenco                  Carbon Dioxide: 1       Lead: 0         Mercury: 1
Med Vaal                Carbon Dioxide: 2       Lead: 0         Mercury: 1
DuelCo                  Carbon Dioxide: 1       Lead: 3         Mercury: 5
AquaLab                 Carbon Dioxide: 1       Lead: 1         Mercury: 2
ReeFlax                 Carbon Dioxide: 2       Lead: 3         Mercury: 4
Guestro                 Carbon Dioxide: 0       Lead: 0         Mercury: 1
Mitco Cor               Carbon Dioxide: 2       Lead: 0         Mercury: 2
OxyMetals               Carbon Dioxide: 3       Lead: 8         Mercury: 7
```

(4 ÷ 2) (2)

Pollutants Details: Display the detailed information on the pollution factor as well as the highest pollutant at each company. Display a suitable heading and subheadings. Calculate and display the average pollution factor of the companies. Example of the output:

```
List of pollutant detail

Company             Pollution Factor    Highest pollutant
=========================================================
Alpha Corps         15                  Carbon Dioxide
Beta Co             20                  Mercury
Delta Max           5                   Carbon Dioxide
Megga Text          22                  Carbon Dioxide
Carbo Cor           40                  Mercury
Trenco              4                   Carbon Dioxide
Med Vaal            5                   Carbon Dioxide
DuelCo              22                  Mercury
AquaLab             9                   Mercury
ReeFlax             20                  Mercury
Guestro             3                   Mercury
Mitco Cor           8                   Carbon Dioxide
OxyMetals           40                  Lead

The average pollution factor is      16.38
```

(10 ÷ 2) (5)

New Info: Allow the user to enter the name of a company. Code an effective way to search for the name in the array by stopping the loop as soon as the name has been found.
Ask the user to enter the new values for carbon dioxide, lead and mercury for the company.
Display a message indicating the information has been updated or that the company has not been found. The name of the company must be part of the message. (20 ÷ 2) (10)

- Make sure that your examination number appears as a comment in the first line of the **CompanyXXXX** class as well as the **testCompanyXXXX** class.
- Save the **CompanyXXXX** class and the **testCompanyXXXX** class.
- Make printouts of the code of the two files to hand in.

[43]

QUESTION 3: JAVA PROGRAMMING

You have been asked to write a program that demonstrates how it might be possible to **simulate** an oil spill on the open sea. A two-dimensional array is used to represent an oil spill. The user will be asked to enter some information and thereafter characters are placed randomly into the 2D array representing the oil spill. The content of the array will be displayed as shown in FIGURE 3.1 below.

The "-" character indicates the open sea. The "+" character indicates oil.

```
   1  2  3  4  5  6  7  8  9  10
1  +  +  -  +  +  +  +  -  +  +
2  -  -  +  +  +  -  -  -  -  +
3  -  +  -  -  -  +  +  +  -  -
4  +  -  +  +  -  -  +  -  -  -
5  -  -  +  -  +  +  -  -  -  +
6  -  +  +  -  -  -  -  -  -  -
7  +  +  +  +  +  -  +  +  -  +
8  -  -  +  -  +  -  +  +  -  +
9  +  -  +  +  +  -  -  +  +  -
10 +  -  +  +  -  -  -  +  +  -
```

FIGURE 3.1

You have been supplied with an incomplete program in the folder named **Question 3 Java** to simulate an oil spill similar to the output shown in FIGURE 3.1.

Do the following:

- Rename the folder named **Question 3 Java** to **Quest3_X**. Replace X with your examination number.
- Open the program in this folder.
- Save the testOilSpill class as **testOilSpillXXXX** and the OilSpill class as **OilSpillXXXX** inside the folder named **Quest3_X**. (XXXX should be replaced with the last four digits of your examination number.)
- The program displays a simple text menu with four options. Change the options on the menu to those indicated in the menu shown below.

```
    MENU

A - Clear open sea
B - Simulate oil spill
C - Evaluate positions
Q - QUIT
Your Choice -
```

You are required to write the following methods in the **OilSpillXXXX** class. Call the methods from the menu in the **testOilSpillXXXX** class to test your program.

3.1 Declare a two-dimensional array and a size variable for the number of rows and columns of the array (maximum 20). The number of rows and columns of the array (and the grid) will be the same.

Write a method named **clearSea** to initialise the array with characters representing the open sea ("-") to the current size. (4 ÷ 2) (2)

3.2 Write a method named **displayGrid** that will display the content of the 2D array in the format indicated in FIGURE 3.1. Code must be included in this method to display the numbering of the rows and columns according to the size of the array. The size of the grid will vary according to the value for the size of the array entered by the user in the Simulate Oil Spill option on the menu. A suitable heading should be displayed. (12 ÷ 2) (6)

3.3 Write a method that can be used to validate a value. The method must receive the lower boundary, the upper boundary and the value that should be validated. Return a Boolean value indicating whether the value is valid or not. Invoke this method whenever input values should be validated. (10 ÷ 2) (5)

3.4 Write a method named **simulateOilSpill** to do the following:

3.4.1 Request the user to enter a suitable size for the 2D array. This entry must be a value between 10 and 20 (both included). The valid boundaries of the value must be indicated to the user. Remember that the number of rows and columns of the array will be the same. The program should not continue until the input value is within the requested range. Invoke the function written in QUESTION 3.3 to validate the input value. Invoke the **clearSea** method to initialise the array with characters to represent the open sea ("-"). (8 ÷ 2) (4)

3.4.2 Request the user to enter the level of seriousness of the spill in the range of 1 to 10. The valid boundaries of the value must be indicated to the user. The program should not continue until the input value is within the requested range. Use the function written in QUESTION 3.3 to validate the input value. The level of seriousness of the spill affects the number of "-" characters in the 2D array that will be changed to "+" characters in the following way:

If the user enters a level of seriousness of 2, then 20 (2 x 10) random positions should be changed to "+" characters. If the user enters 3, then 30 (3 x 10) positions should be changed, and so on.

NOTE:

- The "+" characters should only be placed in the positions that are not polluted yet. If a random position in the array is already occupied by a "+", another random position should be generated until an available position ("-") has been found.
- Positions should not be chosen outside the array boundaries.
- Invoke the **displayGrid** method to display the content of the 2D array.

Examples of the output of test runs:

The following input values were used for the first test run:
Size of the grid: 10
Level of seriousness of the oil spill: 6

```
How big is grid for the simulation? (10 - 20)10
How serious is the spill (1-10)?6
    Oil Spill on the open sea

   1  2  3  4  5  6  7  8  9  10
1  -  -  +  +  -  -  -  +  +  +
2  +  +  +  -  -  +  +  +  +  +
3  +  +  +  -  +  +  +  +  -  +
4  +  -  +  -  +  -  -  +  -  +
5  +  +  -  +  +  +  +  +  -  +
6  +  +  +  -  -  +  +  +  +  -
7  +  +  +  -  -  -  +  -  -  -
8  +  +  -  -  -  -  -  -  -  +
9  +  -  +  +  +  +  +  -  -  +
10 -  +  +  -  -  -  +  +  +  -
```

NOTE: The positions of '+' characters displayed will vary because of the random selection of oil spill positions.

The following input values were used for the second test run:
Size of the grid: 15
The level of seriousness of the oil spill: 10

```
How big is grid for the simulation? (10 - 20)15
How serious is the spill (1-10)?10
    Oil Spill on the open sea

   1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
1  -  -  +  +  +  +  +  +  -  -  -  +  +  +  -
2  +  -  -  +  -  +  +  -  +  -  -  +  -  -  +
3  -  +  -  -  -  -  -  -  -  +  +  -  -  +  -
4  -  +  -  +  -  -  -  -  -  +  +  +  +  -  +
5  -  +  +  -  +  +  -  +  -  +  +  +  +  -  -
6  -  -  +  +  -  +  -  -  +  -  -  -  +  +  +
7  -  +  -  -  -  +  +  -  +  -  -  +  -  +  -
8  +  +  -  -  -  +  -  -  +  -  -  -  +  -  +
9  +  +  -  -  +  -  -  -  -  +  +  -  -  +  -
10 +  -  -  +  -  -  +  +  +  +  +  -  +  -  +
11 -  -  -  -  -  -  -  -  -  +  -  +  +  -  +
12 +  +  +  +  -  -  +  +  -  -  +  +  +  -  +
13 +  -  -  +  -  -  +  -  -  +  +  -  -  -  -
14 +  -  -  +  -  +  +  -  -  -  -  -  -  -  -
15 -  +  -  -  -  -  +  +  +  +  +  -  -  -  +
```

(14 ÷ 2) (7)

3.5 In the **Clear Sea** option:
Invoke the relevant methods to:

- initialise the 2D array with characters representing the open sea ("-").
- display a grid representing the open sea.

Example of output:

```
    Oil Spill on the open sea

   1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20
1  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
2  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
3  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
4  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
5  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
6  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
7  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
8  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
9  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
10 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
11 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
12 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
13 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
14 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
15 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
16 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
17 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
18 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
19 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
20 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
```

(2 ÷ 2) (1)

3.6 Write a method named **evaluatePosition** to do the following:

3.6.1 Request the user to enter two values (row and column) representing a ship at sea. The user must be requested to enter values that do not exceed the size of the array. Invoke the function written in QUESTION 3.3 to validate the values. The program should only continue when the input values are valid (within the boundaries of the array). (8 ÷ 2) (4)

3.6.2 Write code to evaluate the position in the 2D array entered by the user according to the rules that follow. A message should be displayed indicating whether the position is situated in a high risk, risky or low risk area, as well as the position entered. Display the message below the grid on the screen. The risk level is determined as follows:

- 'High risk' area: The position entered by the user is occupied by a "+" in the 2D array.
- 'Risky' area: The position entered by the user is occupied by a "-". 4 or more of the spots surrounding the selected position is polluted ("+"). The selected position is regarded as a 'Risky' area. The following positions in FIGURE 3.2 are examples of 'Risky' areas:
  row 2 and column 4 (four "+" characters surrounding this position),
  row 5 and column 9 (six "+" characters surrounding this position),
  row 7 and column 10 (four "+" characters surrounding this position),
  row 8 and column 6 (seven "+" characters surrounding this position), et cetera.
- 'Low risk' area: The position entered by the user is occupied by a "-". Less than 4 of the spots surrounding the selected position are polluted. The selected position is regarded as a 'Low risk' area. The following positions in FIGURE 3.2 are examples of 'Low risk' areas:
  row 2 and column 5 (two "+" characters surrounding this position),
  row 3 and column 8 (two "+" characters surrounding this position),
  row 1 and column 2 (three "+" characters surrounding this position),
  row 1 and column 1 (one "+" character surrounding this position), et cetera.


```
      Oil Spill on the open sea

     1  2  3  4  5  6  7  8  9  10
  1  -  -  +  -  -  -  -  -  +  -
  2  -  +  +  -  -  +  -  -  +  -
  3  +  +  +  +  -  -  -  -  -  +
  4  +  +  +  -  +  +  +  -  -  +
  5  -  +  +  +  +  +  +  +  -  +
  6  +  +  -  +  +  -  +  +  +  +
  7  -  -  -  +  +  +  +  +  +  -
  8  +  +  -  +  +  -  +  -  -  +
  9  -  -  +  +  -  +  +  +  +  +
  10 +  +  +  -  +  +  +  +  -  -
```

FIGURE 3.2

The user must be able to evaluate as many positions as he/she wants in one simulation.
Example of the output when testing a few positions:

```
    Oil Spill on the open sea

   1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
1  +  -  +  +  +  -  -  +  +  -  -  +  +  +  -
2  +  +  -  +  +  -  +  -  -  -  +  +  +  -  -
3  -  -  +  +  -  -  -  +  +  -  -  +  -  -  +
4  +  +  -  +  +  -  -  -  -  -  -  -  -  -  +
5  +  +  -  -  -  +  -  -  -  -  -  +  +  -  +
6  -  +  +  -  +  -  -  +  -  -  -  -  -  -  -
7  +  -  -  +  +  -  +  +  -  -  +  -  +  +  +
8  +  -  -  +  -  -  -  -  -  -  -  +  -  -  +
9  +  +  -  -  -  -  +  -  -  -  -  -  +  -  +
10 -  -  +  +  -  -  +  -  -  -  -  +  +  -  +
11 -  +  -  -  +  -  +  +  +  +  -  +  +  +  -
12 +  -  +  -  -  +  +  +  +  +  -  +  -  -  -
13 +  -  -  +  -  -  -  +  +  -  +  +  -  -  +
14 -  -  +  -  +  -  +  +  -  +  +  -  +  -  +
15 -  -  -  +  +  -  -  +  -  -  +  -  +  +  -

Enter a number for the row and the column (1 - 15)
Row?4
Column?6
Low risk area in position 4,6

Another position (Y/N)Y
Enter a number for the row and the column (1 - 15)
Row?4
Column?3
Risky area in position 4,3

Another position (Y/N)N
```

(16 ÷ 2) (8)

- Enter your examamination number as a comment in the first line of the **TestOilSpillXXXX** and the **OilSpillXXXX** classes.
- Save the two classes.
- Make printouts of the code of the **TestOilSpillXXXX** class and the **OilSpillXXXX** class to hand in. **[37]**

TOTAL SECTION B: 120

GRAND TOTAL: 120