NATIONAL SENIOR CERTIFICATE EXAMINATION
NOVEMBER 2021

INFORMATION TECHNOLOGY: PAPER I

Time: 3 hours 150 marks

PLEASE READ THE FOLLOWING INSTRUCTIONS CAREFULLY

1. This question paper consists of 15 pages. Please check that your question paper is complete.

2. This question paper is to be answered using object-oriented programming principles. Your program must make sensible use of methods and parameters.

3. This paper is divided into two sections. All candidates must answer both sections.

4. This paper is set in programming terms that are not specific to any particular programming language (Java/Delphi) or database (Access/MySQL/JavaDB).

5. Ensure that you answer the questions in the manner described because marks will be awarded for your solution according to the specifications that are given in the question.

6. Only answer what is asked in each question. For example, if the question does not ask for data validation, then no marks are awarded for it, and therefore no code needs to be written for data validation.

7. If you cannot get a section of code to work, comment it out so that it will not be executed and so that you can continue with the examination. If possible, try to explain the error to aid the marker.

8. When accessing files from within your code, DO NOT use full path names for the files, as this will create problems when the program is marked on a computer other than the one you are writing on. Merely refer to the files using their names and extensions, where necessary.

9. Your programs must be coded in such a way that they will work with any data and not just the sample data supplied or any data extracts that appear in the question paper. You are advised to look at the supplied data files carefully.

10. Make sure that routines such as searches, sorts and selections for arrays are developed from first principles, and that you do not use the built-in features of a programming language for any of these routines.

11. All data structures must be defined and declared by you, the programmer. You may not use components provided within the interface to store and later retrieve data.

12. Read the whole question paper before you choose a data structure. You may find that there could be an alternative method of representing the data that will be more efficient considering the questions that are asked in the paper.

13. You must save all your work regularly on the disk you have been given, or the disk space allocated to you for this examination. You should also create a backup of the original files before you start in case the original version is accidentally modified by your solution.

14. If your examination is interrupted by a technical problem such as a power failure, you will, when you resume writing, be given only the time that was remaining when the interruption began, to complete your examination. No extra time will be given to catch up on work that was not saved.

15. Make sure that your examination number appears as a comment in every program that you code as well as on every page of the hard copy that you hand in.

16. Print a code listing of all the programs/classes/output files that you code. Printing must be done after the examination. You will be given half an hour to print after the examination is finished. Your teacher will tell you what arrangements have been made for the printing of your work.

17. You should be provided with the following two folders (in bold) and files. These files are to be used as data for this examination. Note that the database files are provided in MS Access, JavaDB and MySQL format. Ensure that you are able to open the files with the packages that you will use to code your solutions to this examination.

    Section A:
    DigitalNimbus.mdb
    DigitalNimbus_JavaDB.sql
    DigitalNimbus_MySQL.sql
    SQLAnswerSheet.rtf
    SQLBrowser.exe

    Section B:
    servers.txt
    technicians.txt

SCENARIO:

Digital Nimbus is a Web Hosting company that specialises in hosting different website domains on their cloud servers. The name of the domain, the date the domain was added to Digital Nimbus and whether the domain is a VPN are stored. Digital Nimbus offers the companies different packages depending on operating system and budget. Digital Nimbus also boasts a 24-hour client assistance through their service agents. A service agent is a member of a department and has their experience recorded in years. When a client needs assistance with an aspect of their website, they can open a ticket, which is assigned to a service agent depending on the type of help required. Each ticket has a priority level (A being the most important and F being the least important) and the date and time the ticket was logged. If the ticket has been completed, a second date and time are recorded.

SECTION A SQL

QUESTION 1

All the data on domains registered with Digital Nimbus, the packages, the service agents and the tickets are stored in a database.

tblDomains contains the details of all the domains registered and hosted by Digital Nimbus.

FIELDS	DATA TYPE	DESCRIPTION
DomainID	INTEGER	A unique auto-numbered identification number for each domain.
DomainName	TEXT	The name of the domain.
VPN	BOOLEAN	Whether or not the domain is hosted using a Virtual Private Network.
DateSubscribed	DATE	The date the domain was registered.
PackageID	INTEGER	The PackageID chosen by the company. This is a foreign key to the tblPackages.

The first 10 records of tblDomains:

tblDomains table				
DomainID	DomainName	VPN	DateSubscribed	PackageID
1	herculentertainment.co.cn	FALSE	30-Oct-15	6
2	kiwical.net	FALSE	09-Sep-10	2
3	triplelionresort.co.za	TRUE	17-Nov-14	6
4	asu.org	FALSE	29-Nov-13	3
5	yelloworks.co.ru	FALSE	24-Oct-19	7
6	honeywheels.org	FALSE	28-Aug-19	5
7	libertyresort.co.br	FALSE	17-Feb-19	5
8	surgesecurity.co.uk	FALSE	02-Apr-15	6
9	dynamico.co.cn	FALSE	26-Aug-15	8
10	outnip.co.ru	TRUE	20-Feb-16	3

tblPackages contains the details of the different packages that customers can choose.

FIELDS	DATA TYPE	DESCRIPTION
PackageID	INTEGER	A unique auto-numbered identification number for each package.
Platform	TEXT	The name of the platform of the hosting server.
Package	TEXT	The type of package.
Cost	DOUBLE	The monthly cost of the package.

All the records in tblPackages:

tblPackages table			
PackageID	Platform	Package	Cost
1	linux	bronze	120.00
2	linux	silver	250.00
3	linux	gold	400.00
4	linux	platinum	650.00
5	windows	bronze	460.00
6	windows	silver	740.00
7	windows	gold	1150.00
8	windows	platinum	1450.00

tblServiceAgents contains the details for each service agent.

FIELDS	DATA TYPE	DESCRIPTION
AgentID	INTEGER	A unique auto-numbered identification number for each agent.
Firstname	TEXT	The first name of the agent.
Lastname	TEXT	The surname of the agent.
Experience	INTEGER	The agent's experience measured in years.
Department	TEXT	The department of the agent.

The first 10 records of tblServiceAgents:

tblServiceAgents table				
AgentID	Firstname	Lastname	Experience	Department
1	Lindani	Kolisi	1	Hosting
2	Fezeka	Langeni	5	Maintenance
3	Ethan	Dixon	4	Web Dev
4	Tobias	Moore	6	Web Dev
5	Timothy	Ross	7	Web Dev
6	John	Botha	7	Hosting
7	Stanley	Edwards	3	Admin
8	Phumzile	Mbuli	9	Web Dev
9	Zola	Cibane	5	Admin
10	Lindikhaya	Mbeki	6	Web Dev

tblTickets contains the details of the tickets logged by the agents for the customers.

FIELDS	DATA TYPE	DESCRIPTION
TicketID	INTEGER	A unique auto-numbered identification number for each ticket.
DomainID	TEXT	The DomainID that the ticket relates to. This is a foreign key to the tblDomains.
AgentID	TEXT	The AgentID that opened the ticket. This is a foreign key to the tblServiceAgents
PriorityLevel	DOUBLE	The priority level; A being the highest and F the lowest.
DateLogged	DATE	The date and time the ticket was logged.
DateCompleted	DATE	The date and time the ticket was completed. This value can be null if the ticket has not been completed.

The first 10 records of tblTickets:

tblTickets table					
TicketID	DomainID	AgentID	PriorityLevel	DateLogged	DateCompleted
1	16	23	C	2021/10/08 6:33 AM	2021/10/13 12:53 PM
2	37	37	C	2021/10/08 6:34 PM	2021/10/14 2:08 PM
3	19	37	A	2021/10/08 7:20 PM	2021/10/13 1:18 AM
4	3	7	E	2021/10/08 10:47 PM	2021/10/13 12:59 AM
5	28	23	E	2021/10/09 4:39 AM	2021/10/14 5:57 PM
6	33	29	D	2021/10/09 7:19 AM	2021/10/13 5:34 PM
7	3	34	C	2021/10/10 5:19 AM	2021/10/16 6:06 PM
8	36	26	D	2021/10/10 9:25 AM	2021/10/15 11:15 PM
9	15	31	C	2021/10/10 12:45 PM	2021/10/16 4:31 AM
10	35	12	E	2021/10/10 7:08 PM	2021/10/16 5:52 PM

1.1 Display all the information about the domains that are hosted on a VPN in alphabetical order based on the domain name. Sample output is given below. Note that the secondary ordering may differ depending on the database package used.

Note only the first six records are shown.

DomainID	DomainName	VPN	DateSubscribed	PackageID
11	blixin.org	Yes	2018/10/05	1
34	cannonics.co.ru	Yes	2017/07/05	1
17	ironwood.com	Yes	2011/12/06	7
39	modestpierhotel.co.uk	Yes	2012/06/02	4
19	outlookhotel.co.uk	Yes	2018/05/25	6
10	outnip.co.ru	Yes	2016/02/20	3

(4)

1.2 Display a list of all service agents who have from 2 to 5 (inclusive) years of experience in the Maintenance or Admin departments. The correct output is shown below:

AgentID	Firstname	Lastname	Experience	Department
2	Fezeka	Langeni	5	Maintenance
7	Stanley	Edwards	3	Admin
9	Zola	Cibane	5	Admin
23	Dansa	Ndlovu	3	Admin
24	Zonke	Vilakazi	5	Maintenance
28	Siphokazi	Gumede	5	Admin
33	John	Parker	3	Maintenance

(5)

1.3 Calculate and display how many domains have a ".co.za" domain extension. Name this field **RSACompanies**. The correct output is shown below:

RSACompanies
2

(4)

1.4 Every month the company would like to see a list of all domains that subscribed in the current month and who are not on the platinum package. The company would like to offer a reduced upgrade fee by increasing their current fee by 50% instead of the set cost of the upgrade. Display the domain name, current package and the current cost with a 50% increase. The correct output is shown below:

DomainName	Package	BargainPrice
herculentertainment.co.cn	silver	1110
yelloworks.co.ru	gold	1725
blixin.org	bronze	180
extraousco.in	bronze	180
mirrorhotel.co.cn	gold	600

(7)

1.5 Display a list showing each department and the average amount of experience held by its service agents. Name this field **AvgExperience**. Show only those departments where the average experience is greater than 6. The correct output is shown below:

Department	AvgExperience
Admin	6.11111111111111
Web Dev	6.08333333333333

(7)

1.6 Due to an import error the domains of companies in India need to be corrected. Update all the domains that end with ".in" to become ".co.ind"

(6)

1.7 At the beginning of each day a list showing the incomplete tickets must be displayed. Display the list showing the agentID, domain name, the first and last name of the agent assigned to the ticket, the priority level and the date and time logged. The list should be ordered first by agentID in ascending order then by priority in alphabetical order. This list should only show the tickets that have not been completed yet.

Note only the first five records are shown.

DomainName	Firstname	Lastname	AgentID	PriorityLevel	DateLogged
cannonics.co.ru	Fezeka	Langeni	2	E	2021/10/14 5:11:28 PM
crowtechnologies.co.ru	Tobias	Moore	4	D	2021/10/14 6:54:07 PM
yelloworks.co.ru	Tobias	Moore	4	F	2021/10/14 7:50:42 PM
dynamico.co.cn	John	Botha	6	B	2021/10/14 12:39:59 AM
whirlwindsports.com	John	Botha	6	F	2021/10/14 8:28:12 AM

(8)

1.8 All the companies that have Russian domain names would like to create backup websites. For each domain that ends in a ".ru", insert a matching record in tblDomain. The backup domain name must be the existing domain name preceded by a random number in the range (1 to 5 inclusive). These new backup domains will be hosted on VPN servers with the current date as a subscription date. They will subscribe to the same packages. A sample of all the inserted records are shown below:

DomainID	DomainName	VPN	DateSubscribed	PackageID
81	1yelloworks.co.ru	Yes	2021/10/19	7
82	4outnip.co.ru	Yes	2021/10/19	3
83	1betaaid.co.ru	Yes	2021/10/19	5
84	2corecords.co.ru	Yes	2021/10/19	7
85	4crowtechnologies.co.ru	Yes	2021/10/19	5
86	3cannonics.co.ru	Yes	2021/10/19	1
87	1quietpetalresort.co.ru	Yes	2021/10/19	1

(9)

50 marks

SECTION B OBJECT-ORIENTATED PROGRAMMING

The company, Digital Nimbus, hosts many remote servers for clients. These servers are located in a large room and each server fulfils one function such as a file server, print server, email server or a custom server. The servers are maintained by technicians employed by Digital Nimbus. The location of the server in the server room is recorded to help the technicians find the server. When a server experiences a fault, it is stored in a text file called **servers.txt** along with the type of fault.

The details of the technicians together with their assigned servers are stored in a text file called **technicians.txt**.

The management at Digital Nimbus needs your help to create a program that will manage the technicians' assignments for the different servers. Since the server room is so large, the technicians require a physical map to guide them to the correct server using the server's location in the room. The **Technician** and **Server** classes will need to be created to represent the data given in the text files:

Technician Class

This class will represent the technician's basic details:

- **techID** – the employee identification code for the technician.
- **name** – the full name of the technician.
- **experience** – the amount of experience in years the technician has at the company.
- **roleSpecialty** – the type of server this technician specialises in. This will match one of the four basic roles (file server, print server, email server or a custom server) a server can perform.

The details of the technicians are stored in a text file called **technicians.txt.** with each line of the file representing the data of an individual technician in the following format:

`<techID> # <name> # <experience> # <role speciality>`

The complete text file is shown below:

```
T-A1#Lindani Kolisi#1#File
T-FE#Fezeka Langeni#5#Print
T-EF#Ethan Dixon#4#Email
T-D8#Tobias Moore#6#Custom
T-D1#Timothy Ross#7#Custom
T-F8#John Botha#7#Email
T-D0#Stanley Edwards#3#Print
T-BD#Phumzile Mbuli#9#File
T-C6#Zola Cibane#5#Print
T-A9#Lindikhaya Mbeki#6#Custom
```

Server Class

- **serverID** – the identification code for the server.
- **location** – the location of the server in the server room. (The location contains a letter and a number. The letter represents the row and the number of the column where the server can be found. Rows range from A to J (inclusive) and columns from 1 to 15 (inclusive).)

- **fault** – the basic issue that has been reported for the server. The current faults that can be reported are Power issues, Temperature warnings, Update failures, Hardware failures and Security breaches. Their descriptions may vary.
- **role** – the basic role that the server was designed to provide on the network. There are four basic roles: **File**, **Print** and **Email** with all other roles specified as **Custom**.
- **assignedTech** – this is the name of the technician that will be assigned to resolve the fault for the server.

A text file called **servers.txt** contains all the details about each server that has reported a fault. These servers will be assigned to technicians in Question 6. Each line of the text file contains the data about a server and its reported fault in the following format:

<serverID> # **<location>** # **<fault>** # **<role>**

Below are the first ten lines of the text file:

```
B674#C10#Security#Email
DF8F#A12#Updates#File
6B1A#I2#Security#Print
5F67#B15#Updates#Custom
CD71#A13#Power#Email
8AE6#G12#Power#Custom
E1B0#C8#Hardware#Email
C23F#E3#Temp#Custom
47EC#C5#Power#Email
DC97#I11#Power#Custom
```


QUESTION 2

Use the class diagram below to create a new class called **Technician**. This class will be used to store the details of a technician.

Technician
Fields: - techID : string - name : string - experience : integer - roleSpeciality : string
Methods: + Constructor(inTID : string, inN ; string, inE : integer, inR : string) + getTechID() : string + getName() : string + getExperience() : integer + getRoleSpeciality() : string + toString() : string

2.1 Create a new class named **Technician** with the **techID**, **name**, **experience** and **roleSpecialty** fields as shown in the class diagram. (3)

2.2 Add a parameterised constructor method to the class that will assign the values to the fields of the class. (4)

2.3 Write code to create accessor methods for the **techID**, **name**, **experience** and **roleSpecialty** fields of the class. (2)

2.4 Code a **toString** method to return a string combining all the fields in the following format:

`name,<space>techID,<space>experience<space>year(s)`

`<space>[roleSpecialty]`

For example: `Lindani Kolisi, T-A1, 1 year(s) [File]`

(4)
[13]

QUESTION 3

Use the class diagram below to create a new class called **Server**. This class will be used to store the details of a server. The diagram below indicates the field and methods that are required.

Server
Fields: - serverID : string - location : string - role : string - fault : string - assignedTech : Technician + ROLETYPE_EMAIL : string = "Email" + ROLETYPE_FILE : string = "File" + ROLETYPE_PRINT : string = "Print" + ROLETYPE_CUSTOM : string = "Custom"
Methods: + Constructor(inSID : string, inLo : string, inRo : string, inFa : string) + getServerID() : string + getLocation() : string + getRole() : string + getFault() : string + getAssignedTech() : Technician + setAssignedTech(inTech : Technician) + toString() : string

3.1 Create a new class named **Server** with the **serverID**, **location**, **role**, **fault** and **assignedTech** fields as shown in the class diagram. Note that **assignedTech** is of type **Technician** created in Question 2. (5)

3.2 Create four constants ROLETYPE_EMAIL, ROLETYPE_FILE, ROLETYPE_PRINT and ROLETYPE_CUSTOM as shown in the class diagram. These constants must be available outside the class and can be either static or non-static. (4)

3.3 Write code to create a parameterised constructor method for the class that will accept parameters for the **serverID**, **location**, **role** and **fault**. The parameter for the role must be checked to see if it matches one of the constants. This check must be case insensitive. If the parameter matches, then assign the parameter to the role field. If it does not match, then the role should be set to ROLETYPE_CUSTOM. (6)

3.4 Create accessor methods for the **serverID**, **location**, **role** and **fault** fields. (2)

3.5 Create a getter and a setter method for the **assignedTech** field. (2)

3.6 Create a **toString** method that will return a string that combines the values of the fields in the **Server** class. The method should also append the technician's details using the **toString** method of the **Technician** class. If a technician has not been assigned, the message "none assigned" should be added. The format should be as follows:

```
Server:<space>serverID(Role:<space>role)<newline>
Fault:<space>fault @ <location>
Assigned<space>to:<space>technician
```

For example:

```
Server: B764(Role: Email)
Fault: Security @ C10
Assigned to: Ethan Dixon, T-EF, 4 year(s) [Email]
```

However, if no technician has been assigned, the string would be:

```
Server: 2DD7(Role: Custom)
Fault: Temp @ C6
Assigned to: none assigned
```

(6)
[25]

QUESTION 4

4.1 Create a class named **ServerManager**. (1)

4.2 Create two private fields for the class. The first must be an array called **sArr** to store 50 **Server** objects. The second must be a counter called **size** to keep track of the number of **Server** objects added to the array called **sArr**. The fields should not be accessible outside the class. (4)

4.3 Create a constructor method that will read all the contents of the text file named **servers.txt** containing the information for servers and their faults. Each line read from the file will add one **Server** object to the array. The method should do the following:

- Check if the file **servers.txt** exists. Display an error message if the file does not exist.
- Open the file for reading.
- Loop through all the lines of the text file. In each iteration of the loop:
  - Read the line and split the data for a **Server** into the separate parts.
  - Create a **Server** object using the split data and store the object in the next available position in the array called **sArr**.
- Update the counter variable called **size**. (9)

4.4 Write code to create a method named **allServers**. This method should return a string that contains the information of all the servers. Use the object's **toString** methods that you have created previously. Each server's details should be separated by a blank line. (5)

4.5 Write code to create a method named **countServers**. This method should accept two string parameters one being to represent a fault and the second the role type. The checks must be case insensitive. The method should return an integer containing a count of how many servers match the fault type and role type. (5)
[24]

QUESTION 5

5.1 Write code to create a text-based user interface called **ServerUI** that will allow simple input and output. (1)

5.2 Declare and instantiate a **ServerManager** object in an appropriate place in the code. (1)

5.3 Write code that will do the following by calling the appropriate methods in the **ServerManager** class:
Display a list of all the **Server** objects. Example output of the first two and the last two servers is shown below:

```
Server: B674(Role: Email)
Fault: Security @ C10
Assigned to: none assigned

Server: DF8F(Role: File)
Fault: Updates @ A12
Assigned to: none assigned

…

Server: 78BA(Role: Print)
Fault: Temp @ C13
Assigned to: none assigned

Server: 5932(Role: Print)
Fault: Security @ A9
Assigned to: none assigned
```

(1)

5.4 Show a count of how many servers have experienced a **Temp** (temperature) fault and have a **Custom** server role type. You must use the appropriate constant in the **Server** class for the role type.

```
Number of servers with a Temp fault and Custom Role: 5
```

(3)
[6]

QUESTION 6

6.1 Code a method named **assignTechnicians** in the **ServerManager** class that will read all the data from the text file **technicians.txt**. The method should create a **Technician** object from each line of the file and assign the technician to the appropriate **Server** objects. To ensure that technicians do not overwork themselves the following criteria are used when assigning a technician.

- A technician must be assigned to a maximum of 4 servers.
- Technicians must only work on servers that have the same role as the technician's role speciality.
- The servers do not have any priority and will be assigned in the order they appear in the **servers.txt** file if it meets the technician's requirements.

Note this does mean that some servers may not be assigned a technician. (11)

6.2 Call the **assignTechnicians** method you created above in an appropriate class and redisplay all the details of the servers. An example of the output for the first two and the last three servers is shown below:

```
Server: B674(Role: Email)
Fault: Security @ C10
Assigned to: Ethan Dixon, T-EF, 4 year(s) [Email]

Server: DF8F(Role: File)
Fault: Updates @ A12
Assigned to: Lindani Kolisi, T-A1, 1 year(s) [File]

…

Server: F5FF(Role: Custom)
Fault: Updates @ I9
Assigned to: none assigned

Server: 78BA(Role: Print)
Fault: Temp @ C13
Assigned to: Stanley Edwards, T-D0, 3 year(s) [Print]

Server: 5932(Role: Print)
Fault: Security @ A9
Assigned to: Stanley Edwards, T-D0, 3 year(s) [Print]
```

(2)
[13]

QUESTION 7

To help the technicians find the servers in the server room, you have been requested to create a map displaying the location of the servers assigned to each technician. The map will need to show the row letters and the column numbers. Each technician will input their **techID** to create a customised map of their allocated servers in the server room.

The map will need to be saved to a text file named after the technician so that the technician has the option to print a hard copy of the map later. The servers that are allocated to a technician often change, so the current date and time needs to be included as the first line of the text file, in the format YYYY/MM/DD HH:MM:SS.

```
2021/10/19 11:55:42

   1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
A  *  *  X  X  *  *  *  *  *  *  *  *  *  *  *
B  *  *  *  *  *  *  *  *  *  X  X  *  *  *  *
…
J  *  *  *  *  *  *  *  *  *  *  *  *  X  *  *
```

7.1 Write code in the **ServerManager** class to create a method named **printMap**. This method should accept one string parameter representing the **techID** of the technician who needs a map printed. The method should identify all locations of the technician's allocated servers and denote these locations on the map. For each of the technician's allocated servers the map should display the letter "X" in the correct position. However, for servers not assigned to the technician the symbol "*" should be shown.

The map generated should be saved to a text file with the same name as the technician's ID and the extension **.txt**. For example, **T-D1.txt**. The method should return a **string** representing the map that was sent to the text file. (17)

7.2 Write code in the **ServerUI** to call the **printMap** method and display the map for the technician with the **techID** of **"T-D1"**. The correct output is shown below:

```
2021/10/19 11:55:42

   1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
A  *  *  *  *  *  X  *  *  *  *  *  *  *  *  *
B  *  *  *  *  *  *  *  *  *  X  *  *  *  *  *
C  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
D  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
E  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
F  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
G  *  *  *  *  *  *  *  *  *  X  *  *  *  *  *
H  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
I  *  *  X  *  *  *  *  *  *  *  *  *  *  *  *
J  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
```

(2)
[19]

100 marks

Total: 150 marks