NATIONAL SENIOR CERTIFICATE EXAMINATION
NOVEMBER 2014

INFORMATION TECHNOLOGY: PAPER II

Time: 3 hours 120 marks

PLEASE READ THE FOLLOWING INSTRUCTIONS CAREFULLY

1. This question paper consists of 12 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).

5. Make sure 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.

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 of the file, 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 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 which 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.

14. If there is a technical interruption that prevents you from writing your examination, e.g. a power failure, when you resume writing your examinations, you will only be given the time that was remaining when the interruption began. 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 hard copy that you hand in.

16. Print a code listing of all the programs/classes 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.

SCENARIO

LuxAir is a domestic airline that specialises in flying high profile people to domestic locations. They have a small number of customers who fly with them on a regular basis. They require various IT solutions to manage baggage, ticket and flight details.

SECTION A STRUCTURED QUERY LANGUAGE

When a passenger checks in for a flight they can check in one or more items of baggage which are loaded onto the plane. Each item of baggage is weighed and the type of baggage together with the insured value of its contents are recorded. A passenger can have multiple items of baggage for a flight. Alternatively passengers can choose to take small items of luggage on the plane as hand luggage. Hand luggage items are not recorded. When an item of baggage is damaged, stolen or lost it is reported by the passenger and recorded by the airline. For simplicity of this examination, we will assume that each passenger only has **one flight**.

The database contains three tables. The first table **Passengers** contains the details of each passenger including their name, flight number and destination. The second table **Baggage** contains information about each item of baggage that is checked in. There may be multiple items of baggage for each passenger. The third table **Claims** contains details of damaged, lost or stolen baggage. You are required to extract some useful information from the database that stores information of passengers' baggage.

The fields in the database are discussed below. Below each description is a screen-shot of the first 10 rows of data for your convenience. The tables do contain more data:

Passengers

PassengerID	This integer field assigns each passenger a unique ID number.
FullName	This text field contains the names of the passengers.
Flight	This text field contains the flight that the passenger is on. Note that each passenger is only linked to one flight.
Destination	This text field contains the destination the passenger is flying to.

PassengerID	Fullname	Flight	Destination
1	Emma Glenn	FL101	Cape Town
2	Veda Bennett	FL203	Durban
3	Burton Wiley	FL405	Johannesburg
4	Zoe Rose	FL101	Cape Town
5	Alma Berg	FL405	Johannesburg
6	Calvin Rodgers	FL405	Johannesburg
7	Evelyn Wagner	FL203	Durban
8	Macy Mullins	FL405	Johannesburg
9	Nero Mercado	FL405	Johannesburg
10	Kimberly Lawrence	FL101	Cape Town

Baggage

BaggageID	This integer field contains a unique ID for each item of baggage.
PassengerID	This integer field contains the ID of the passenger that the baggage belongs to. This field is a foreign key to the Passengers table.
Weight	This real/double field contains the weight of the item of baggage in kilograms.
CheckInCounter	This integer field contains the number of the check-in counter where the item of baggage was weighed.
BaggageType	This text field contains the type of baggage. Possible values are 'Suitcase', 'Sporting Equipment', 'Parcel', 'Other'.
InsuredValue	This real/double field contains the value of the contents of the item of baggage.
Fragile	This Boolean field indicates whether the item of baggage is considered to contain fragile contents or not.

BaggageID	PassengerID	Weight	CheckInCounter	BaggageType	InsuredValue	Fragile
105640	61	28.93	1	Sporting Equipment	1648.1	True
106310	53	7.14	1	Parcel	1249.2	True
108210	50	1.57	2	Other	2054.1	False
112896	13	15.74	1	Sporting Equipment	710.8	False
114047	26	2.0	1	Suitcase	1729.7	True
117502	65	21.45	4	Other	693.3	False
119910	43	23.66	3	Other	2420.7	True
120622	30	20.24	2	Sporting Equipment	563.5	False
123568	33	24.69	3	Parcel	1515.4	True
130349	73	21.64	4	Parcel	148.7	True

Claims

ClaimID	This integer field contains the unique ID of the Claim. This field is NOT an autonumber.
BaggageID	This integer field contains the ID of the Baggage item that the claim relates to. This field is a foreign key to the Baggage table.
Description	This text field contains a description of how the baggage loss occurred. Possible values are 'Theft', 'Breakage', 'Lost in Transit' and 'Other'.
Reference	This text field contains a reference the airline gives to passengers so that they can track their claims. The contents have intentionally been left blank.

ClaimID	BaggageID	Description	Reference
1	978472	Other	
2	418006	Other	
3	119910	Breakage	
4	879591	Lost in Transit	
5	900899	Breakage	
6	117502	Theft	
7	555325	Theft	
8	130349	Other	
9	895701	Breakage	
10	596430	Other	

QUESTION 1

1.1 Write a query that will list all the details of all **Passengers** travelling to Cape Town. (3)

1.2 Write a query that will display each type of baggage from the **Baggage** table. Your query should display each **BaggageType** only once. An example of the output is given below: (3)

BaggageType
Other
Parcel
Sporting Equipment
Suitcase

1.3 Francis Porter's suitcase has been stolen (BaggageID 240402). Write a SQL statement which will add an entry to the **Claims** table using 16 as the **ClaimID** for your new entry. (4)

1.4 Shana Woodward (**PassengerID** 14) has decided that she would rather take her baggage on the plane as hand luggage and not check it in. Write a query that will remove all entries about her baggage from the **Baggage** table. (2)

1.5 The **Reference** field in the **Claims** table has been left blank. The airline would like to generate a reference number for each claim using the first two letters of the **Description** and the last four digits of the **BaggageID** in the **Claims** table. Write a query that will update the **Reference** field for all claims. After running your query the contents of the **Claims** table should appear as follows: (5)

ClaimID	BaggageID	Description	Reference
1	978472	Other	Ot8472
2	418006	Other	Ot8006
3	119910	Breakage	Br9910
4	879591	Lost in Transit	Lo9591
5	900899	Breakage	Br0899
6	117502	Theft	Th7502
7	555325	Theft	Th5325
8	130349	Other	Ot0349
9	895701	Breakage	Br5701
10	596430	Other	Ot6430
11	108210	Breakage	Br8210
12	883064	Breakage	Br3064
13	180426	Breakage	Br0426
14	585953	Theft	Th5953
15	892992	Breakage	Br2992

1.6 Passengers have booked in more than one item of baggage. Write a query that will show the total weight of each passenger's baggage items. Call the total weight of each person's baggage **TotalWeight**. Order the results in descending order of total weight. The airline needs to determine the total weight booked by a passenger. (6)

1.7 Write a query which will display the **InsuredValue** and **Weight** of each item of baggage classified as 'Sporting Equipment' or 'Other' as well as the name of the passenger that owns the baggage. An example of the first 5 rows of output is shown below: (5)

Fullname	InsuredValue	Weight
Cheryl Booth	1648.1	28.93
Amery Harvey	2054.1	1.57
Tyrone Holt	710.8	15.74
Suki Baldwin	693.3	21.45
Barry Chaney	2420.7	23.66

1.8 Write a query which will list the **Fullname** of any passenger that does not have checked in baggage (i.e. all passengers whose **PassengerID** does not appear in the **Baggage** table). An example of the output is given below: **(Note: Shana Woodward will only appear in the result of this query if you successfully executed the query in Question 1.4.)** (5)

Fullname
Alma Berg
Shana Woodward
Georgia Macias
Hop Hull
Clarke May
Elvis Stewart
Benjamin Shannon
Colleen Mann
Nayda Rose
Davis William
Aimee Osborne
Asher Graves

1.9 Write a query which will display all the details of all fragile **Baggage** that was checked in at **CheckInCounter** 6 which has an **InsuredValue** greater than the average **InsuredValue** of all **Baggage**. An example of the output is given below: (7)

BaggageID	PassengerID	Weight	CheckInCounter	BaggageType	InsuredValue	Fragile
157491	2	12.11	6	Sporting Equipment	1784.2	True
240402	11	19.57	6	Suitcase	2326.2	True
335841	38	17.95	6	Suitcase	2255.6	True
596430	6	8.72	6	Sporting Equipment	1746.8	True
615505	47	23.01	6	Sporting Equipment	2207.8	True
723694	38	12.35	6	Other	1649.9	True
989365	37	10.35	6	Suitcase	2014.6	True

40 marks

SECTION B OBJECT ORIENTED PROGRAMMING

LuxAir needs to be able to keep track of all passengers' tickets for their domestic flights. For each journey a passenger is provided with a single ticket. A ticket consists of a unique ticket number and the passenger's name. In addition each ticket contains information on the departing flight and the returning flight. The structure is therefore as follows:

Each **Ticket** contains the following information:

- TicketID
- Full name of the passenger
- **Departing Flight** – the flight to the passenger's destination
- **Returning Flight** – the return flight from the passenger's destination

Each **Flight** (whether it is a **Departing Flight** or **Returning Flight**) contains the following information:

- **Code –** the flight number of the flight.
- **Origin –** the location that the flight leaves from (as a three letter code).
- **Destination –** the destination the flight will arrive at (as a three letter code).
- **Departure Time –** the date and time that the flight leaves the origin.
- **Arrival Time –** the date and time that the flight arrives at its destination.
- **Cost –** the total cost of that flight.

In addition the following information is relevant:

- All flights take off at their origin and land at their destination on the same day (i.e. there are no overnight flights).
- Every ticket is a return ticket. This means that the passenger will return to the same location that they departed from.
- Each passenger has booked multiple tickets.

An example for Julian Vincent of a departure and return flight between Durban (DUR) and Port Elizabeth (PLZ) is as follows:

TICKETID	217085910	**NAME**	Jillian Vincent
Departing Flight		**Returning Flight**	
Code	YP794	**Code**	YP689
Origin	DUR	**Origin**	PLZ
Destination	PLZ	**Destination**	DUR
Departure Time	2014-12-08 09:55	**Departure Time**	2014-12-12 16:45
Arrival Time	2014-12-08 11:00	**Arrival Time**	2014-12-12 17:45
Cost	975.58	**Cost**	1036.54

You have been given a file called **tickets.txt** which contains the information for a number of tickets. A sample of the first 15 lines of the file is given below:

```
105255264#Jillian Vincent
YP794#DUR#PLZ#2014-12-08 09:55#2014-12-08 11:00#975.58
YP689#PLZ#DUR#2014-12-12 16:45#2014-12-12 17:45#1036.54
111864400#Barbara Knapp
AS976#PLZ#CPT#2015-01-24 10:05#2015-01-24 12:10#1036.47
DM137#CPT#PLZ#2015-01-25 19:00#2015-01-25 20:45#1128.36
113612810#Wanda Hutchinson
WJ455#JNB#PLZ#2015-04-19 11:45#2015-04-19 13:15#997.52
WJ377#PLZ#JNB#2015-04-28 20:15#2015-04-28 21:40#783.21
115194993#Brendan Montgomery
CA916#JNB#CPT#2015-03-21 07:05#2015-03-21 09:10#705.25
CA177#CPT#JNB#2015-03-30 18:00#2015-03-30 20:00#987.54
115907703#Teegan Lawrence
CA916#JNB#CPT#2015-03-10 07:05#2015-03-10 09:10#705.25
CA177#CPT#JNB#2015-03-11 18:00#2015-03-11 20:00#987.54
```

A single ticket's data is spread over exactly **THREE** lines in the file. In other words **ONE** ticket is represented by **THREE** consecutive lines in the text file. In the example above there are 5 separate tickets shown. Lines 1 – 3 contain the first ticket's information; lines 4 – 6 contain the second ticket's information; lines 7 – 9 contain the third ticket's information and so on. The ticket's data is structured as follows:

- Line 1 of each ticket – Ticket Information in the following format
  - `ticketid#passengername`
- Line 2 of each ticket – Departing Flight's information in the following format
  - `code#origin#destination#departureTime#arrivalTime#cost`
- Line 3 of each ticket – Returning Flight's information in the following format
  - `code#origin#destination#departureTime#arrivalTime#cost`

QUESTION 2

Use the class diagram below to create a new class called **Flight**. This class will be used to create objects that will store the details of an individual flight. Note that this object can be used to store either a departing or a returning flight. The class diagram below indicates the fields and methods that are required.

Flight
Fields: - String code - String origin - String destination - String departureTime - String arrivalTime - double cost
Methods: + Constructor(String cde, String orig, String dest, String dtime, String atime, double cst) + getCode() : String + getDepartureTime() : String + getCost() : double + toString() : String

2.1 Write code to create a new class called **Flight**. (1)

2.2 Write code to define the six fields for the **Flight** class as indicated in the above class diagram. (3)

2.3 Write code to create a constructor method that will assign values to the fields of the **Flight** class. (3)

2.4 Write code to create accessor methods for the **code**, **cost** and **departureTime** fields. (3)

2.5 Write code to create a **toString** method which will return a String comprised of the details of the flight in the following format:

```
code<tab>origin<space>departureTime<tab>destination<space>arrivalTime
```

for example:

```
YP794     DUR 2014-12-08 09:55     PLZ 2014-12-08 11:00
```

(4)

[14]

QUESTION 3

Use the class diagram below to create a new class called **Ticket**. This class will be used to store the details of a single ticket for a passenger. The first two fields will store the ticket number and the passenger name. The next two fields will be **Flight** objects (the class created in the previous question). The class diagram below indicates the fields and methods that are required.

Ticket
Fields: - String ticketID - String name - **Flight** departingFlight - **Flight** returningFlight
Methods: + Constructor(String tID, String nme, **Flight** dflight, **Flight** rflight) + getName() : String + getDepartingFlight() : **Flight** + getReturningFlight() : **Flight** + getTotalCost() : double + toString() : String

3.1 Write code to create a new class called **Ticket**. (1)

3.2 Write code to define four fields that will store the **ticketID**, **name**, **departingFlight** and **returningFlight** associated with a ticket. Choose appropriate data types for these fields but note that the **departingFlight** and **returningFlight** fields are objects of the class **Flight**. These fields should not be visible from outside the class. (3)

3.3 Write code to create a constructor method that will assign values to all the fields of the **Ticket** class. Note that in addition to the code and name parameters you are required to pass two **Flight** objects as parameters into the constructor. (3)

3.4 Write code to create accessor method for the **name** field. (1)

3.5 Write code to create accessor methods for the **departingFlight** and **returningFlight** fields. Both of these accessor methods should return the respective **Flight** objects. (2)

3.6 Write code to create a method called **getTotalCost**. This method should return a real number representing the total cost of the ticket. This can be calculated by adding the cost of the **departingFlight** to the cost of the **returningFlight**. (2)

3.7 Write code to create a **toString** method which will return a string comprised of the information for the ticket as well as the two flights contained in the ticket. The string returned should be in the following format:

```
ticketID<tab>name<tab>totalcost<newline>
departingFlightInformation<newline>
returningFlightInformation<newline>
```

for example:

```
105255264  Jillian Vincent   R 2012.12
YP794      DUR 2014-12-08 09:55    PLZ 2014-12-08 11:00
YP689      PLZ 2014-12-12 16:45    DUR 2014-12-12 17:45
```

(3)

[15]

QUESTION 4

4.1 Write code to create a new class called **FlightManager**. (1)

4.2 Write code to declare two instance variables in the class. The first being an array that can be used to store up to 500 **Ticket** objects. The second being an integer counter to keep track of how many **Tickets** are stored in the array. (3)

4.3 Write code to create a constructor method that will read the contents of the file **tickets.txt**. Each three lines that you read will result in one **Ticket** object being added to the array. Do the following:

- Check if the file exists. If it does not, display an error message.
- Open the file for reading.
- Loop through the file until there are no more lines. In each iteration of the loop:
  - Read the first line of each ticket and split the **Ticket** data contained in that line into the ticket number and ticket name.
  - Read the second line of the ticket and split the **Flight** data contained in that line into the flight code, origin, destination, departure time, arrival time and cost. This data represents the departing flight.
  - Read the third line of the ticket and split the **Flight** data contained in that line into the flight code, origin, destination, departure time and cost. This data represents the returning flight.
  - Create a **Flight** object representing the departing flight using the data you read from the second line.
  - Create a **Flight** object representing the returning flight using the data you read from the third line.
  - Create a **Ticket** object using the ticket data from the first line and the **TWO Flight** objects you created.
  - Add the **Ticket** object at the end of the array. (15)

4.4 Write code that will create a method called **allTickets**. This method should return a string that contains the information of all tickets. Use the object's **toString** methods that you created in the questions above. Each ticket's details should be separated by a blank line. (5)

4.5 Write code to create a method called **sort**. This method should sort the array of tickets based on the departure time of the departing flight of each ticket in ascending order. In other words the data should start with the ticket which has the earliest departing flight. (8)

[32]

QUESTION 5

5.1 Write code to create a simple user interface called **FlightUI** which will allow simple output. (1)

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

5.3 Write code that will display all the tickets in sorted order by calling the appropriate methods in the **FlightManager** class. You must call the methods in the following order:

Sort
All Tickets (2)
[4]

QUESTION 6

LuxAir wishes to find out which passenger flies the most. They want to determine this by calculating the total amount of time (in minutes) that each passenger spends flying. For any one particular ticket the duration of the departing flight is added to the duration of the returning flight. A passenger's total travelling time is calculated by summing the duration of all their tickets.

6.1 Write code to add a method in the appropriate class that will determine the duration of each flight. A flight's duration is the difference between the number of minutes of the departure time and the number of minutes of the arrival time of that flight. Your method must return the duration in minutes of a flight. (4)

6.2 Write code to add a method called **frequentFlyer** in the appropriate class which will return a string containing the name and total duration (in minutes) of the passenger with the most total flying time. This method should add up the flying duration for each passenger's tickets and determine which passenger has the most flying time. You may add any other helper methods in any of the classes as part of your solution. Marks will be awarded for efficiency of code. (10)

6.3 Add a call statement in the **FlightUI** interface which will allow the user to display the most frequent flyer with an appropriate heading. The results should be as follows: (1)

```
Most Frequent Flyer
-------------------
Wanda Hutchinson 1505
```

[15]

80 marks

Total: 120 marks