NATIONAL SENIOR CERTIFICATE EXAMINATION
NOVEMBER 2018

INFORMATION TECHNOLOGY: PAPER II

Time: 3 hours 120 marks

PLEASE READ THE FOLLOWING INSTRUCTIONS CAREFULLY

1. This question paper consists of 16 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. 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 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 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

In order to attract tourists to the City of Idayimani, the City Tourism Board has created a number of facilities.

These are:

- A centralised booking system for excursions from the city centre to various attraction points near the city.
- A bus service operating on various routes between attraction points in the city.

SECTION A SQL

QUESTION 1

One of the facilities allows tourists to book various daily excursions. Any daily excursion that is available can be booked by any tourist registered on the system.

Three tables are used to store data related to the bookings. The fields in the database are described in the following tables together with some sample data. The first ten rows of data are shown, but the tables do contain more data.

TOURIST table contains the details of each tourist:

FIELD	DATA TYPE	DESCRIPTION
TouristID	INTEGER	A unique identification number for each tourist registered on the system
TouristName	TEXT	The name of the tourist
Email	TEXT	The email address as registered on the system
Hotel	TEXT	The name of the hotel the tourist is currently staying at
DateRegistered	DATE	The date on which the tourist registered on the system

TOURIST table – sample data (first 10 records)

TouristID	TouristName	Email	Hotel	DateRegistered
1	Andris Nuth	anuth0@dyndns.org	Peninsula Hotel	2015-05-15
2	Flor Byres	fbyres1@cargocollective.com	President Hotel	2018-02-05
3	Dwight Crisell	dwight@crisell.com	Village Lodge	2018-04-18
4	Mary Crisell	mary@crisell.com	Village Lodge	2018-04-18
5	Darryl Poleykett	dpoleykett4@bbb.org	Mount Grace Hotel	2017-06-02
6	Kort McAndie	kmcandie5@diigo.com	Village Lodge	2015-08-22
7	Arron Haney	ahaney6@blogspot.com	Peninsula Hotel	2012-05-17
8	Jorrie Potten	jpotten@seattletimes.com	Mount Grace Hotel	2016-11-03
9	Mic MacArd	mmacard8@weebly.com	Peninsula Hotel	2018-08-02
10	Johnath Nixon	jnixon@seattletimes.com	Peninsula Hotel	2016-05-13

EXCURSION

This table contains the details of all daily excursions available for booking.

FIELD	DATA TYPE	DESCRIPTION
ExcursionID	INTEGER	A unique identification number for each excursion
ExcursionName	TEXT	The name given to the excursion
StartHour	INTEGER	The excursion commences at this hour. This number is in 24 hour format, i.e. 0–11 is considered morning, 12–16 afternoon and 17–23 evening.
EndHour	INTEGER	The excursion ends at this hour. It is also in 24 hour format with values from 0 to 23.
CurrentCost	DOUBLE	The current cost of this excursion
Surcharge	DOUBLE	The conservation surcharge that is added to the current cost. Some excursions are not related to conservation and will have a 0 surcharge.

EXCURSION table – sample data (first 10 records)
Data may be formatted differently on your computer depending on its settings.

ExcursionID	ExcursionName	StartHour	EndHour	CurrentCost	Surcharge
1	Sunrise Breakfast River Cruise	5	8	550	25
2	Sunset River Cruise	17	20	700	25
3	Morning Safari	9	12	450	15
4	Afternoon Safari	13	16	450	15
5	Night Safari	21	23	450	20
6	Township Excursion 1	8	10	250	0
7	Township Excursion 2	10	12	250	0
8	Township Excursion 3	12	14	250	0
9	Township Excursion 4	14	16	250	0
10	Township Excursion 5	16	18	250	0

BOOKING

This table contains tourists' excursion booking details.

FIELD	DATA TYPE	DESCRIPTION
BookingID	AUTONUMBER	A unique identification number for each booking.
TouristID	INTEGER	The TouristID of the tourist that booked this excursion. This field is the foreign key to the **TOURIST** table.
ExcursionID	INTEGER	The ExcursionID of the excursion linked to this booking. This field is the foreign key to the **EXCURSION** table.
ExcursionDate	DATE	The date for which the excursion was booked by the tourist.
CostCharged	DOUBLE	The cost of this excursion to the tourist.

BOOKING table – sample data – first 10 records
Data may be formatted differently on your computer depending on its settings.

BookingID	TouristID	ExcursionID	ExcursionDate	CostCharged
1	1	1	2018-10-05	620
2	1	3	2018-10-05	465
3	1	9	2018-10-05	225
4	1	3	2018-10-06	465
5	1	13	2018-10-06	315
6	1	16	2018-10-07	720
7	2	6	2018-10-07	225
8	2	12	2018-10-07	315
9	2	2	2018-10-07	785
10	3	4	2018-10-07	475

1.1 Write a query that will display the information of all the tourists whose email address ends with "seattletimes.com". A sample of the correct output is shown below. (Note that date format may differ on your system).

Tourist ID	Tourist Name	Email	Hotel	Date Registered
8	Jorrie Potten	jpotten@seattletimes.com	Mount Grace Hotel	2016-11-03
10	Johnath Nixon	jnixon@seattletimes.com	Peninsula Hotel	2016-05-13
11	Davis Eginton	degintona@seattletimes.com	Peninsula Hotel	2018-07-30

(4)

1.2 All guests currently staying at the "Lunar Hotel" have been moved to the "Three Seasons Hotel". Write a query to update this information. (3)

1.3 Write a query that will display the name and duration of morning excursions that are at most 3 hours. Recall that morning excursions start on the hour of 0 to 11 inclusive. Name the column displaying the calculated hours "Duration". A sample of the correct output is shown below.

ExcursionName	Duration
Sunrise Breakfast River Cruise	3
Township Excursion 1	2
National Art Museum Excursion 1	3

(6)

1.4 Write a query that will display the names of the hotels with at least 3 tourists staying there. Also display the number of tourists staying at each of these hotels.

Hotel	NumberOfTourist
Mount Grace Hotel	5
Peninsula Hotel	6
President Hotel	3
Village Lodge	5

(6)

1.5 Write a query that will list the names of the tourists who have not yet booked for any excursion. Display the tourists' names alphabetically.

TouristName (first ten rows)
Arron Haney
Cornall Prout
Darryl Poleykett
Davis Eginton
Eba Gillison
Eleen Yeomans
Irina Gouny
Johnath Nixon
Jorrie Potten
Kort McAndie
...

(6)

1.6 Excursion codes are created by combining the first three characters of the excursion name with a random number between 10 and 99 inclusive. Write a query to generate excursion codes for each excursion.

ExcursionName	**ExcursionCode** *The last two digits will be different in your case because they are randomly generated.*
Sunrise Breakfast River Cruise	Sun96
Sunset River Cruise	Sun56
Morning Safari	Mor32
Afternoon Safari	Aft31
Night Safari	Nig25
Township Excursion 1	Tow36
Township Excursion 2	Tow84
Township Excursion 3	Tow82
Township Excursion 4	Tow18
Township Excursion 5	Tow25
…	…

(6)

1.7 All of the tourists staying at the "President Hotel" are going on all the following excursions today.

- Sunrise Breakfast River Cruise (ExcursionID: 1)
- Township Excursion 2 (ExcursionID: 7)
- National Art Museum Excursion 3 (ExcursionID: 13)
- Night Safari (ExcursionID: 5)

Write a query that will sign them up for these excursions by inserting appropriate entries into the **BOOKING** table.

Note that:

- the cost charged can be calculated by adding the surcharge to the current cost.
- today's date should not be hard coded (use an appropriate function to determine today's date).
- the excursion IDs 1, 7, 13 and 5 as well as the name of the hotel may be hard coded. (These values may form part of the SQL statement). (9)

40 marks

SECTION B OBJECT-ORIENTED PROGRAMMING

SCENARIO

The City of Idayimani runs an open-top bus service that takes tourists to several points of interest, called **stops**, in the city. There are several bus **routes**, each visiting a number of **stops**. Some of these stops are shared among multiple routes.

Stops

A class will need to be created to represent a **stop**. A **stop** has the following information:

- **StopName** – the name of the **stop**
- **RouteCodes** – the route(s) on which this **stop** is listed. This string consists of **B**, **R**, **P**, **Y** and any combination thereof in any order. For example, a RouteCodes string of "BR" means that this **stop** is visited by both route B and route R. Some **stops** are not in use and therefore will have a blank route.
- **StopType** – each **stop** can be one of four types:

  1 – Cafe: The **stop** has a small shop that sells coffee and snacks to tourists.
  2 – Shelter: There is a small shelter with some seats. The bus will stop here and wait for at least three minutes.
  3 – Express: There is no seating area, only some standing room and no roof cover. The bus will not stop if there is no one wanting to get off or get on.
  4 – Other: The fallback type if the type is not yet known or is missing.

The class **Stop** also has four constant class (static) variables to represent the four different stop types mentioned above. Their values are 1 for *Cafe*, 2 for *Shelter*, 3 for *Express* and 4 for *Other*.

Text File

The text file *data.txt* contains information about all the **stops** shared amongst all of the **routes**. Each line contains information about one **stop**. Here are the first ten lines of this text file:

```
Waterfront,1,RB
St Monicas Cathedral,3,R
Conference Centre,2,R
CC,3,R
Market Square,2,RBY
Clock Tower,2,Y
Idayimani Museum,3,Y
Grande Hotel,5,BY
SA Heritage Museum,2,Y
Apartheid Museum,2,Y
```

Each line of the text file consists of the following information:

```
StopName,StopType,RouteCodes
```

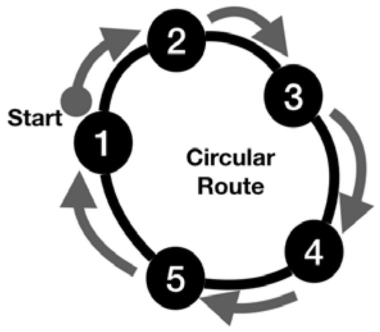
For example, the **stop** at "Waterfront" (given on the first line of the text file) is type 1 (which corresponds to *Cafe*) and is visited by routes R and B.

Routes

A class will have to be created to represent **routes**. A **route** is made up of multiple **stops**. There are two types of **routes**:

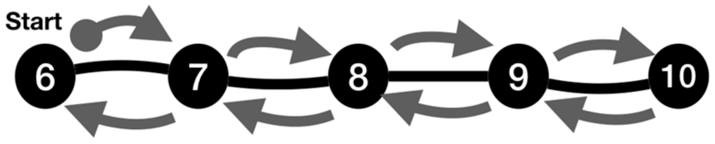
Circular route

Some of the **routes** are *circular*, meaning that the next **stop** after the last **stop** is the original departure **stop**. For example, a bus following a circular **route** with **stops** 1,2,3,4,5 will follow this pattern on its route: 1,2,3,4,5,1,2,3, ... and so on (see right).



Linear route

Some of the routes are *linear* meaning that once the last **stop** is reached, the bus returns along the same route and visits all of its **stops** in reverse. For example, a linear route with points 6,7,8,9,10 will follow this pattern on its route: 6,7,8,9,10,9,8,7,6,7,8, ... and so on (see right).



Linear route

Each **route** has the following information:

- **RouteCode** – a single character representing the **route**.
- **Whether the route is circular** – if a **route** is not *circular*, it is *linear*.
- **An array of stops** that form part of this **route**. They should be listed in the order they are visited, starting with the departure **stop**. Each **stop** should appear only once on the list. The array of stops only lists the stops relevant to this route, not all the stops listed in the text file *data.txt*.

The scenario given above must be used to answer QUESTION 2 TO QUESTION 7.

Note that each question starts on a new page.

QUESTION 2

The class **Stop** has the following UML class diagram. It indicates the properties and methods that are required. Take careful note of the method and parameter names in the diagram. **NO** additional *public* property or method should be created. However, you may create any additional *private* property or method as needed.

Stop
Properties: - String stopName - String routeCodes - integer stopType + (*static/class constant*) integer STOPTYPE_CAFE = 1 + (*static/class constant*) integer STOPTYPE_SHELTER = 2 + (*static/class constant*) integer STOPTYPE_EXPRESS = 3 + (*static/class constant*) integer STOPTYPE_OTHER = 4
Methods: + Constructor (String inStopName, String inRouteCodes, integer inStopType) + getStopTypeName() : String + isPartOfRoute(char) : Boolean + toString() : String

2.1 Create a new class named **Stop**. (1)

2.2 Create three properties to store ***stopName***, ***routeCodes*** and ***stopType***. (3)

2.3 Create the four static/class constants `STOPTYPE_CAFE` with the value 1, `STOPTYPE_SHELTER` with the value 2, `STOPTYPE_EXPRESS` with the value 3 and `STOPTYPE_OTHER` with the value 4. (3)

2.4 Create a constructor method that accepts two strings and an integer as parameters. The first two string parameters represent the name of the **stop** and the code(s) of the routes that visit this **stop**. The integer represents the type of the **stop**.

The constructor should do the following:

- The two string parameters must be used to assign the values of the ***stopName***, and ***routeCodes*** properties.
- The integer should be used to represent the stop type and must be one of the values of the static constants. If the given integer parameter is not one of static constants, then the ***stopType*** property should have its value set to `STOPTYPE_OTHER`. (7)

2.5 Create a method named ***getStopTypeName()*** that returns the name of the stop type depending on the value of the ***stopType*** property:

- If the ***stopType*** property has the same value as STOPTYPE_CAFE the method should return "cafe".
- If the ***stopType*** property has the same value as STOPTYPE_SHELTER the method should return "shelter".
- If the ***stopType*** property has the same value as STOPTYPE_EXPRESS the method should return "express".
- For any other value for the ***stopType*** *property*, the method should return "other". (3)

2.6 Create the method ***isPartOfRoute(..)*** that takes in a character. The method should return a Boolean to indicate whether or not this **stop** is part of the route represented by the given character. (3)

2.7 Create a ***toString()*** method that can be used to output a string representation of the **Stop** instance that includes the ***stopName*** and the stop type name in this format:

```
<Stop Type Name><tab><Stop Name>
```

For example:

```
cafe    Waterfront
```

(3)

[23]

QUESTION 3

Use the class diagram below to create a new class called **Route**. This class will be used to store the details of a bus route. The diagram below indicates the properties and methods that are required. **NO** additional *public* attribute or method should be created. However, you may create any additional *private* attribute or method as needed.

Route
Properties: - char routeCode - boolean isCircular **Note**: This is *true* for circular route and *false* for linear route. - Stop[ ] stops
Methods: + Constructor(char routeCode, boolean isCircular) + setStops(Stop[ ] s) + getRouteCode() : char + getStopAt(int num) : Stop + toString() : String

3.1 Create a new class named **Route** with the three properties indicated above. (3)

3.2 Create a constructor method that accepts one char as a parameter representing the ***routeCode*** and a Boolean representing whether or not the **route** is circular. The constructor should use the given parameters to set the ***routeCode*** and ***isCircular*** properties of the **Route** instance. The property ***stops*** should NOT be set by the constructor as it will be set by ***setStops(..)*** method. (2)

3.3 Create a mutator (set) method for the ***stops*** property that accepts an array of Stops as a parameter. (2)

3.4 Create a get method for the property ***routeCode***. (1)

3.5 Create a method named ***getStopAt(..)***. The method will accept the position of a stop as an integer and return the corresponding **Stop** object in the array. The method should return *null* / *nil* if the position given is not valid. For example, if the array only has 5 stops, 8 would not be valid and a *null* object must be returned. (5)

The ***toString()*** method of the **Route** class will be completed later in Question 5.

[13]

QUESTION 4

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

4.2 In this class, declare the following array and counter variable as instance variables:

- An array that can be used to store up to 100 **Stop** objects.
- A counter variable to keep track of the number of **Stop** objects stored. (3)

4.3 Create a constructor method that will read the contents of a text file containing the information of **ALL** the stops. The constructor must take in a string representing the file name. Each line read in from the text file will result in ONE **Stop** object being added to the array above.

In the constructor method, do the following:

- Open the file for reading. You may assume that the file exists.
- Loop through the file until there are no more lines. In each iteration of the loop:
  - Read in each line and split the **Stop** data into separate items.
  - Create a **Stop** object with the information present on each line.
  - Add the newly created **Stop** object to the array and update the counter variable appropriately. (9)

4.4 Create a method called ***getRouteWithCode(..)*** that can take in a character representing a ***routeCode*** and a Boolean representing whether or not the route is circular. This method should return a new **Route** object with the corresponding attributes. The **Route** object's Stop array should contain ONLY the **stops** belonging to that **route**, in the order they appear in the text file.

Note that marks will be awarded for efficiency. For example, re-reading from the text file instead of working from the existing array is not efficient.

Also note that no new *public* properties or *public* methods are to be created in any other classes. You may, however, create additional *private* properties and/or *private* methods as you find necessary. (10)

[23]

QUESTION 5

You will now create the **toString**() method for the **Route** class that can be used to obtain a string representation of a **Route** instance in the following formats depending on whether the **route** is circular or linear. All routes with no stops should be treated as invalid routes.

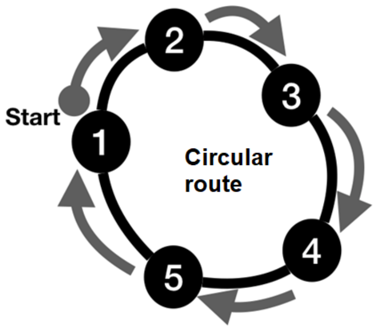
Note that marks will be awarded for efficiency. (9)

Format for invalid route

```
<Route Code> - Invalid
```

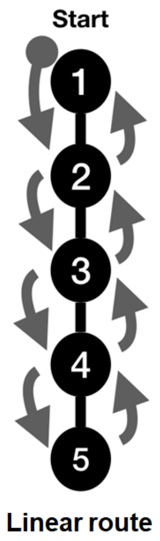
Format for circular route
(Example below has 5 stops)**:**

```
<Route Code> - Circular
-> 1 <Stop 1 in Stops' toString format>
-> 2 <Stop 2 in Stops' toString format>
-> 3 <Stop 3 in Stops' toString format>
-> 4 <Stop 4 in Stops' toString format>
-> 5 <Stop 5 in Stops' toString format>
-> 1 <Stop 1 in Stops' toString format>
```



Linear route
(Example below has 5 stops)**:**

```
<Route Code> - Linear
-> 1 <Stop 1 in Stops' toString format>
-> 2 <Stop 2 in Stops' toString format>
-> 3 <Stop 3 in Stops' toString format>
-> 4 <Stop 4 in Stops' toString format>
-> 5 <Stop 5 in Stops' toString format>
-> 4 <Stop 4 in Stops' toString format>
-> 3 <Stop 3 in Stops' toString format>
-> 2 <Stop 2 in Stops' toString format>
-> 1 <Stop 1 in Stops' toString format>
```



[9]

QUESTION 6

6.1 Create a simple user interface called **TourUI** that will allow simple output. (1)

6.2 Declare and instantiate a **TourManager** object at the appropriate place in the code. The text file *data.txt* should be used to instantiate the object. (1)

6.3 Write code to acquire two **Route** objects from the **TourManager** using the method ***getRouteWithCode(..)*** that was created in Question 4.4. The first **route** is a circular route with route code "R" and the second is a linear route with code "Y". (2)

6.4 Write code to print out both routes in their ***toString()*** format. (1)
[5]

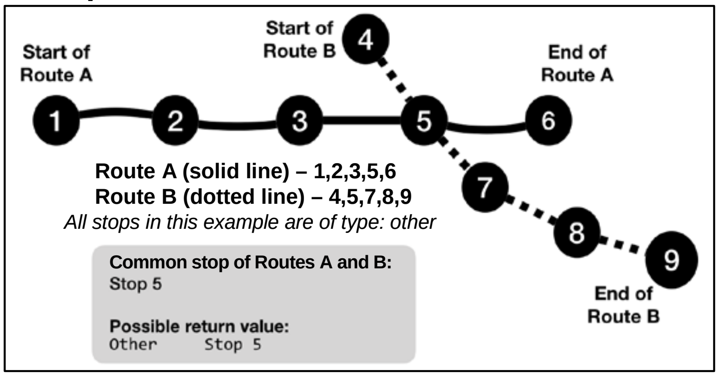
NB: PLEASE TURN OVER FOR QUESTION 7

QUESTION 7

7.1 In the **TourManager** class, create a suitable method that takes in two **Route** objects as parameters. The method will work out the **stops** that are common to both **routes**. The method should return a string listing the common **stops** on separate lines. Each common **stop** should be included ONCE in the **Stop** class' ***toString()*** format. (6)

Here are some examples:

Example 1: Two linear routes

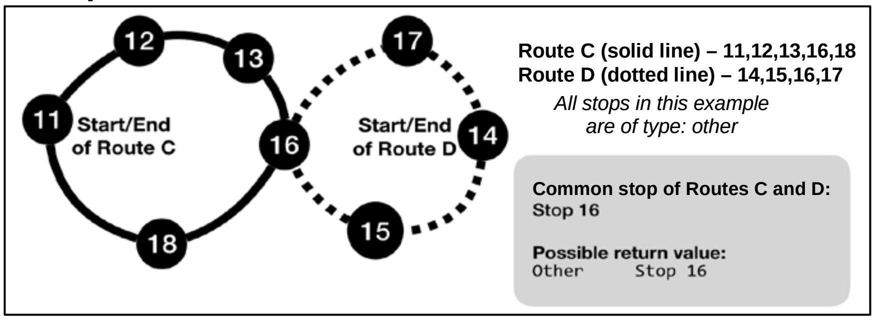


Route A (solid line) – 1,2,3,5,6
Route B (dotted line) – 4,5,7,8,9
All stops in this example are of type: other

Common stop of Routes A and B:
Stop 5

Possible return value:
```
Other     Stop 5
```

Example 2: Two circular routes

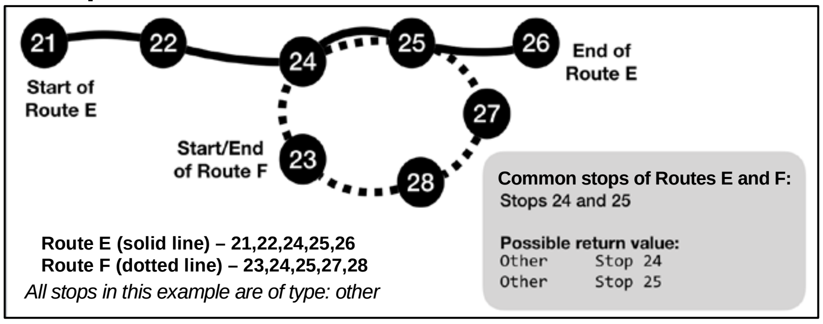


Route C (solid line) – 11,12,13,16,18
Route D (dotted line) – 14,15,16,17
All stops in this example are of type: other

Common stop of Routes C and D:
Stop 16

Possible return value:
```
Other     Stop 16
```

Example 3: A linear and a circular route



Route E (solid line) – 21,22,24,25,26
Route F (dotted line) – 23,24,25,27,28
All stops in this example are of type: other

Common stops of Routes E and F:
Stops 24 and 25

Possible return value:
```
Other     Stop 24
Other     Stop 25
```

7.2 In the **TourUI** class, use the method created in Question 7.1 to print out the common **stop**(s) shared by the two **Route** objects created in Question 6.3. (1)

[7]

80 marks

Total: 120 marks