PECANWOOD COLLEGE

Prepared for Life

Preliminary Examination
September 2020

INFORMATION TECHNOLOGY: PAPER II

Time: 3 hours 120 marks

Student Name	

EXAMINERS: C Kader, C Lewis, R Viljoen **MODERATORS**: M Walker, M Ounaceur, E van Aarde

PLEASE READ THE FOLLOWING INSTRUCTIONS CAREFULLY

1. This question paper consists of 17 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.

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 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:
    DeliveryDB.mdb
    SQLAnswerSheet.doc $\}\leftarrow *$

    Section B:
    Data.txt

Cluster 480

PRELIMINARY EXAMINATION

Grade 12

2020

Information Technology – P2

Practical

EXAMINERS: C Kader, C Lewis **DURATION:** 3 Hours (+ Printing)

MODERATORS: R Viljoen, M Ounaceur, E van Aarde **MAXIMUM MARK:** 120

This paper consists of 7 questions printed on 14 pages including the cover page

Name: ____________________ UserName: ____________

Question	1	2	3	4	5	6	7	Total
Max	40							120
Mark								

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.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.

SECTION A **STRUCTURED QUERY LANGUAGE**

SCENARIO:

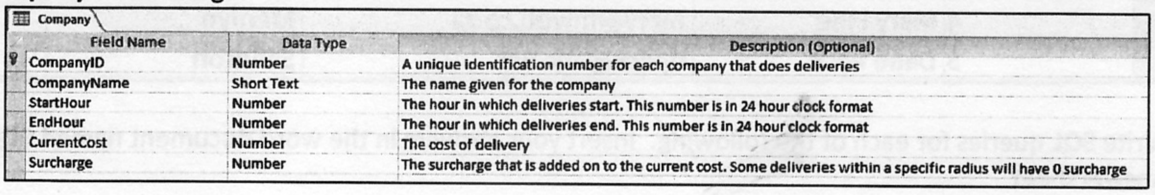
Due to the Covid-19 Lockdown, people have decided to rather stay home and be safe. Many therefore subscribe to delivery services for their food, be it groceries and/or fast foods.

QUESTION 1

The database **DeliveryDB** is supplied. Three tables are used to store data related to customers, delivery companies and orders. The fields in the database are described in the following tables together with some sample data. The first 5 rows of data are shown for each table, but the tables do contain more data.

The Company table contains details of each company that provides a delivery service.

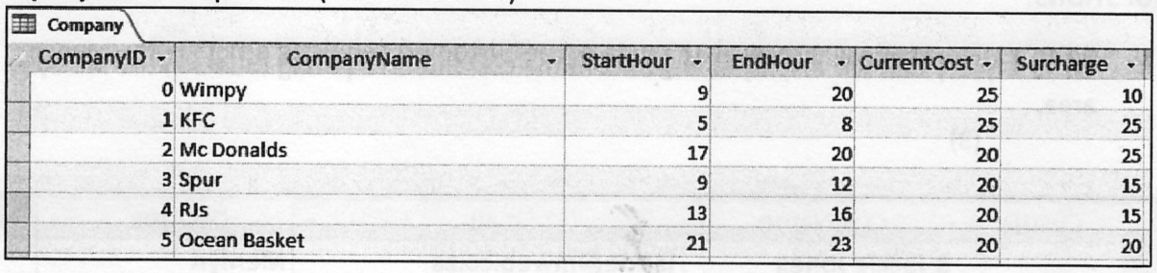
Company table design:



Company

Field Name	Data Type	Description (Optional)
CompanyID	Number	A unique identification number for each company that does deliveries
CompanyName	Short Text	The name given for the company
StartHour	Number	The hour in which deliveries start. This number is in 24 hour clock format
EndHour	Number	The hour in which deliveries end. This number is in 24 hour clock format
CurrentCost	Number	The cost of delivery
Surcharge	Number	The surcharge that is added on to the current cost. Some deliveries within a specific radius will have 0 surcharge

Company table sample data (first 10 records):

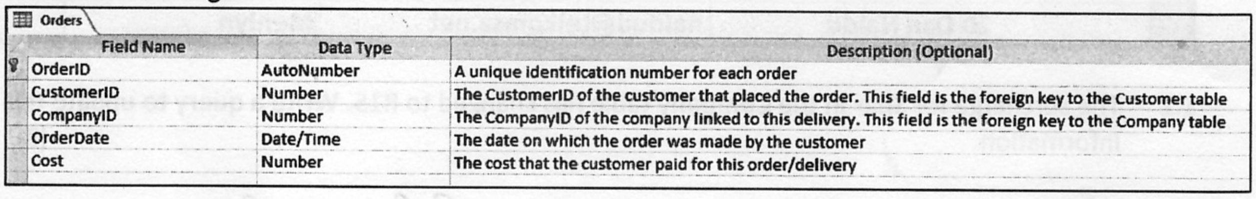


Company

CompanyID	CompanyName	StartHour	EndHour	CurrentCost	Surcharge
0	Wimpy	9	20	25	10
1	KFC	5	8	25	25
2	Mc Donalds	17	20	20	25
3	Spur	9	12	20	15
4	RJs	13	16	20	15
5	Ocean Basket	21	23	20	20

The **Orders** table contains the basic order details of the order.

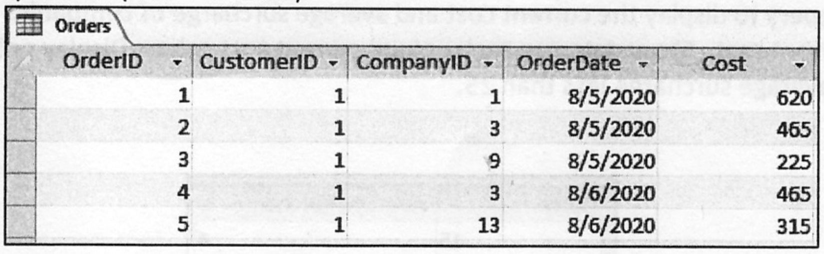
Orders table design:



Orders

Field Name	Data Type	Description (Optional)
OrderID	AutoNumber	A unique identification number for each order
CustomerID	Number	The CustomerID of the customer that placed the order. This field is the foreign key to the Customer table
CompanyID	Number	The CompanyID of the company linked to this delivery. This field is the foreign key to the Company table
OrderDate	Date/Time	The date on which the order was made by the customer
Cost	Number	The cost that the customer paid for this order/delivery

Orders table sample data (first 5 records):

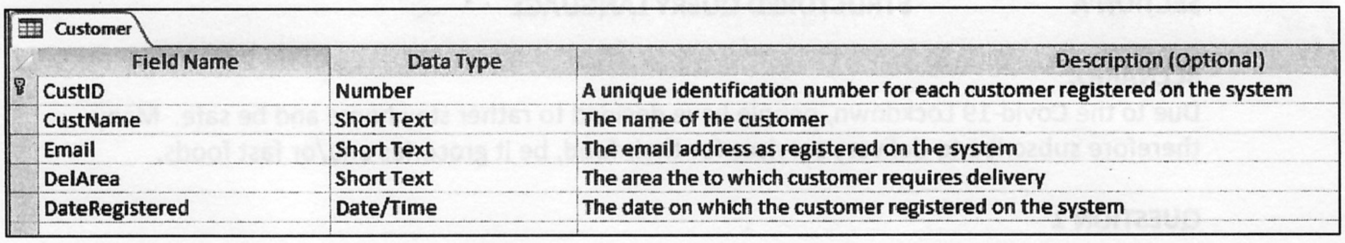


Orders

OrderID	CustomerID	CompanyID	OrderDate	Cost
1	1	1	8/5/2020	620
2	1	3	8/5/2020	465
3	1	9	8/5/2020	225
4	1	3	8/6/2020	465
5	1	13	8/6/2020	315

The **Customer** table contains details about each customer.

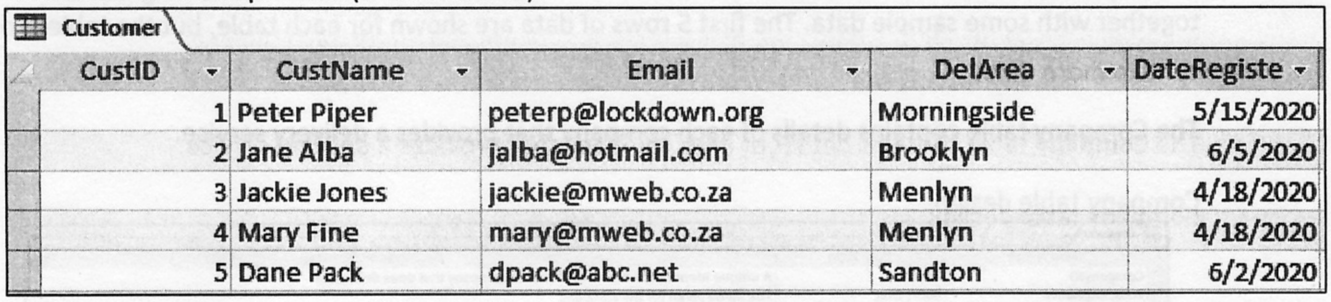
Customer table design:



Customer

Field Name	Data Type	Description (Optional)
CustID	Number	A unique identification number for each customer registered on the system
CustName	Short Text	The name of the customer
Email	Short Text	The email address as registered on the system
DelArea	Short Text	The area the to which customer requires delivery
DateRegistered	Date/Time	The date on which the customer registered on the system

Customer table sample data (first 5 records):



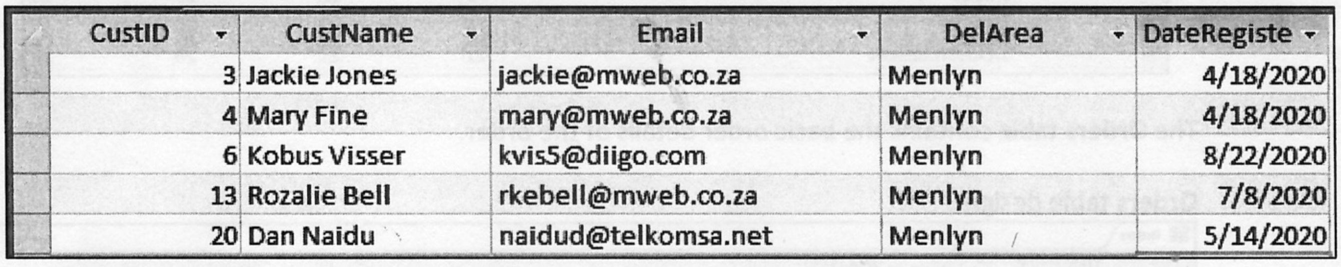
Customer

CustID	CustName	Email	DelArea	DateRegiste
1	Peter Piper	peterp@lockdown.org	Morningside	5/15/2020
2	Jane Alba	jalba@hotmail.com	Brooklyn	6/5/2020
3	Jackie Jones	jackie@mweb.co.za	Menlyn	4/18/2020
4	Mary Fine	mary@mweb.co.za	Menlyn	4/18/2020
5	Dane Pack	dpack@abc.net	Sandton	6/2/2020

Write SQL queries for each of the following. Insert your answers in the word document named **SQL Answersheet**

QUESTIONS:

1.1 Write a query that will display all the data of the customers requiring orders in the Menlyn area. (3)

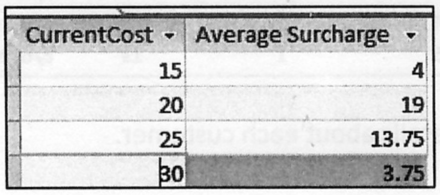


CustID	CustName	Email	DelArea	DateRegiste
3	Jackie Jones	jackie@mweb.co.za	Menlyn	4/18/2020
4	Mary Fine	mary@mweb.co.za	Menlyn	4/18/2020
6	Kobus Visser	kvis5@diigo.com	Menlyn	8/22/2020
13	Rozalie Bell	rkebell@mweb.co.za	Menlyn	7/8/2020
20	Dan Naidu	naidud@telkomsa.net	Menlyn	5/14/2020

1.2 The surcharge for Spar in the Company table has changed to R15. Write a query to update this information. (3)

GROUP BY

1.3 Write a query to display the current cost and average surcharge of companies having the same current cost. Show data per each unique current cost value. Display only those records with an average surcharge less than 25. (5)



CurrentCost	Average Surcharge
15	4
20	19
25	13.75
30	3.75

1.4 Write a query to display the company name, the customer name, order date and the cost for all orders placed in the month of July. Name the output of the cost field, "Cost of Order". Display the data in descending order of cost. (9)

Note: The format of the date may differ depending on your computer settings.

CompanyName	CustName	OrderDate	Cost of Order
Burger King	Peter Piper	2019/07/12	865
Mc Donalds	Jackie Jones	2020/07/12	785
Spur	Jane Alba	2020/07/06	475
Woolworths	Jane Alba	2020/07/07	315
Spar	Mary Fine	2020/07/15	225

1.5 Write a query to display the customer name, delivery area and a code for each customer under the heading **CustCode**, made up as follows:

- The first 2 letters of the customer name
- The last 3 letters of the delivery area
- A random number between 10 and 99 (including these values) that is different for every customer. (7)

Note: The numbers will be different in your display since they are randomly generated.

CustName	DelArea	CustCode
Peter Piper	Morningside	Peide34
Jane Alba	Brooklyn	Jalyn66
Jackie Jones	Menlyn	Jalyn48
Mary Fine	Menlyn	Malyn18
Dane Pack	Sandton	Daton60

1.6 Write a query to delete all orders placed in 2018 and 2019 from the Order table. (3)

1.7 Customers who subscribe to Telkom, Vodacom and Mweb, receive a discount of 10 % on their surcharge. Write a query to display the customer name and email address of all customers who subscribe to Mweb. (2)

CustName	EMail
Jackie Jones	jackie@mweb.co.za
Mary Fine	mary@mweb.co.za
Millie Monroe	mmonroe8@mweb.co.za
Sharlene Bend	sbend@mweb.co.za
Rozalie Bell	rkebell@mweb.co.za

1.8 Display the details of all customers who are not in any of the following delivery areas: Brooklyn, Menlyn and Sandton. (2)

NOTE: Do not use multiple OR statements in the query.

CustID	CustName	Email	DelArea	DateRegiste
1	Peter Piper	peterp@lockdown.org	Morningside	5/15/2020
7	Abby Holden	abbyhol6@telkomsa.net	Morningside	5/17/2020
9	Millie Monroe	mmonroe8@mweb.co.za	Morningside	8/2/2020
10	Jack Nixon	jnixon@gmail.com	Morningside	5/13/2020
11	Dolly Eginton	degintona@gmail.com	Morningside	7/30/2020
18	Jay Singh	jaysingh@telkomsa.net	Morningside	5/21/2020
19	Ndu Sbu	ndusbu@opera.org	Midrand	6/7/2020

1.9 The **Uber Eats** needs to be inserted in the Company table. Except for the EndHour field, which will be **23**, all other details are the same as KFC. Write a query to insert the Uber Eats details into the Company table, selecting the applicable KFC data from the database. (6)

SECTION A STRUCTURED QUERY LANGUAGE

SCENARIO:

40 marks

SECTION B OBJECT ORIENTED PROGRAMMING

SCENARIO

Souper-Eats has been taking orders for delivery during the national lockdown. Users can order from several different vendors and the delivery service will collect your order and deliver to your address.

All orders are recorded in a text file. Each line refers to one food item. Each order may consist of more than one food item. You will calculate the total value of each order and keep track of the number of items ordered.

A text file *data.txt* was generated. Here are the first three lines of this text file:

```
1;SSR;Chicken burger with chips;1;79.90
1;SSR;Cheese burger with chips;2;86.9
2;OB;Fish and chips;1;75.0
```

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

```
OrderNo;Vendor;Description;Quantity;Price
```

QUESTION 2

The **Food** class was designed with the following class diagram. It indicates the properties and methods required. Note the method names and parameter names in the diagram. No additional *public* method or property may be created. You may create additional *private* properties and methods if you need to.

Food

Properties

- orderNo : integer
- description : string
- vendor : string
- quantity : integer
- price : real
- totalCost : real
- noOfItems : integer

Methods

order # Description Vendor Quantity Price

+ constructor (inOrder : integer, inD : string, inV : string, inQ : integer, inP : real)
+ getOrderNo() : integer
+ getTotalCost() : real
+ toString() : string
+ getNoOfItems() : integer

2.1 Create a new class called **Food**. (1)

2.2 Create the six properties as indicated in the diagram. (3)

2.3 Create the static/class variable **noOfItems** to keep track of the total number of items ordered in this text file. (1)

2.4 Create a constructor method that will accept five parameters and use it to initialise the class properties. This method should also calculate the totalCost as quantity * price. In addition, increase the noOfItems by 1 to record the number of unique items. (5)

2.5 Create the accessor(get) methods for **orderNo** and **totalCost**. (2)

2.6 Create the static accessor **getNoOfItems** that returns the noOfItems. (1)

2.7 Create a **toString** method to return a single string to represent a Food item in the following format:
<orderNo><space><vendor><tab><description><tab><quantity><space>"R"<totalCost>

For example

```
1 SSR    Chicken burger with chips    1 R79.90
```

(2)

[15]

QUESTION 3

Use the class diagram below to create a new class called **Order**. This class will be used to store the details of an order and the food items included in this order. You may assume that one Order will include a maximum of 10 unique food items. The diagram below indicates the properties and methods that are required. Note the method names and parameter names in the diagram. No additional methods or properties may be created.

Order
Properties - orderNo : integer - foodArr : Food[ ] - orderTot : real - <u>SERVICE_FEE = 10 : integer</u>
Methods + constructor (inNo : integer, inFA : Food[ ]) + toString : string - calcOrderTot() : double

3.1 Create a new class called **Order** with the four properties as indicated in the class diagram. (4)

3.2 Create a constructor method that will assign the two values received as parameters to the class properties **orderNo** and **foodArr**. The constructor should also calculate the order total by calling the method **calcOrderTot**. (3)

3.3 Create a method called **calcOrderTot**. This method will calculate the total value of the order and return the result. The order total is calculated by adding up the total cost of each **Food** item in the array. The service fee is then applied using the class constant. A **SERVICE_FEE** of 10 indicates that a 10% service fee must be added to the total cost of the order. A R45 delivery fee is also added to each **Order**. Round the order total to two decimal places. (6)

3.4 Create the **toString** method to return the order details, showing the order number, the food items included (using the toString method from the Food class) and the order total. Format the output as shown in this example.

```
ORDER: 1
1 SSR      Chicken burger with chips   1 R79.90
1 SSR     Cheese burger with chips    2 R173.80
Order total: R324.07
```

(4)

[17]

QUESTION 4

4.1 Create a class called **OrderManager**. (1)

4.2 Create the following instance variables:

- An array that can store 20 Order objects.
- An array that can store 50 Food objects.
- Two counters to keep track of how many objects there are in each array. (3)

4.3 Create a constructor method that will read the contents of a text file containing the information of all the **Food** items ordered. The constructor must accept the file name as a string. Each line read in from the text file will result in ONE **Food** object being added to the Food array above.

Do the following in the constructor method:

- 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 create a **Food** object with the information present on each line.
  - Add the newly created **Food** object to the **Food array** and update the correct counter variable.

(7)

4.4 Create a new method called **listFoods**. It should return a single string with the details of all the Food objects in the array using the correct toString method. Each Food item should appear on a separate line. The screenshot below shows the possible output for this method.

```
1 SSR   Chicken burger with chips          1 R79.90
1 SSR   Cheese burger with chips           2 R173.80
```

(4)

4.5 Create a new method called **sort**. It should sort the items in the Food array in ascending order according to the order number. (5)

[20]

QUESTION 5

5.1 Create a simple user interface called **Interface** that will allow simple output. (1)

5.2 Declare and instantiate an **OrderManager** object. The name of the text file, data.txt should be sent as a parameter when instantiating the object. (2)

5.3 Write code to display all **Food** items. (1)

5.4 Write code to display the total number of **Food** items, with an appropriate message. Use the static method, getNoOfItems of the Food class to retrieve the value. (1)

5.5 Write code to sort the Food items using the method you wrote earlier. (1)

Sample of the output for the first two Food items:

```
1 SSR   Chicken burger with chips       1 R79.90
1 SSR   Cheese burger with chips        2 R173.80
```

[6]

QUESTION 6

Return to the **Food** class. Each Food item is recorded using a code for the vendor. The following list shows the code and the corresponding full name of the vendor.

Code	Vendor name
SSR	Spur Steak Ranch
PP	Pizza Perfect
OB	Ocean Basket
FA	FishAways
KFC	Kentucky Fried Chicken

Your task is to add code to this class to convert the code in the Food item to the vendor name. Note that this list may be expanded as vendors are added.

6.1 Choose a suitable data structure that will allow you to find the corresponding vendor name for a code. (2)

6.2 Create a private method called **getVendorName**. The method should receive a String representing the vendor code of the Food item. It should locate the corresponding vendor name and return that value. If the code is not found, return the value "Unknown". (5)

6.3 In the **toString** method of the Food class, call the **getVendorName** method to print the full name of the vendor instead of the code. Below is a sample of the output for two food items including the vendor name.

```
1 Spur Steak Ranch         Chicken burger with chips          1 R79.90
1 Spur Steak Ranch         Cheese burger with chips           2 R173.80
```

(2)

[9]

QUESTION 7

Return to the **OrderManager** class.

7.1 Create a new method called **collateOrders**. You have previously declared an array for **Order** items. The collateOrders method should collate (combine) all the **Food** items which are part of the same order number, into a single order. Ensure that the food array is sorted in ascending order according to the order number. Search through the **Food** items in the Food array to find all the items for a single order. When you have found all the items for a single order, create an **Order** object and add it to the appropriate array. Marks will be awarded for efficient code. (8)

7.2 Create a new method called **printOrders**. This method should return a String with the orders formatted for printing, using the **toString** method from the Order class. A sample of the output is shown below. Please note that it is not necessary to align the output in columns for all orders.

```
ORDER: 1
1 Spur Steak Ranch      Chicken burger with chips      1 R79.90
1 Spur Steak Ranch      Cheese burger with chips       2 R173.80
Order total: R324.07
```

(3)

Return to the **Interface** class.

7.3 Call the method to collate the Food items into orders. (1)

7.4 Add code to the interface to print all the orders. (1)

[13]

Complete set of sample output:

```
1 Spur Steak Ranch      Chicken burger with chips       1 R79.90
1 Spur Steak Ranch      Cheese burger with chips        2 R173.8
2 Ocean Basket  Fish and chips  1 R75.00
2 Ocean Basket  Prawn combo     2 R240.00
2 Pizza Perfect 4 Seasons Large 1 R110.00
2 KFC   15 pc bucket    1 R199.90
3 FishAways     Hake and calamari       2 R99.80
3 Pizza Perfect Regina medium   1 R65.90

Total number of items: 8

ORDER: 1
1 Spur Steak Ranch      Chicken burger with chips       1 R79.90
1 Spur Steak Ranch      Cheese burger with chips        2 R173.8
Order total: R324.07

ORDER: 2
2 Ocean Basket  Prawn combo     2 R240.00
2 Pizza Perfect 4 Seasons Large 1 R110.00
2 KFC   15 pc bucket    1 R199.90
Order total: R649.89

ORDER: 3
3 Pizza Perfect Regina medium   1 R65.90
Order total: R117.49
```

80 marks

Total: 120 marks

Bare Bones Approach

Notes from the 2020 Prelim practical exam

General

1. You MUST memorise how to read in a text file using the Scanner class. Write it down from memory before you start coding your exam (first 5 minutes)
2. In the second 5 minutes of the exam name your classes strictly according to the exam paper. Do not deviate. Start again if you must.
3. Read the class diagrams. Follow them EXACTLY.
4. Do not put the main method into the code needed for "question one". The main method goes into the UI class which can be question two or three or four or five. Read the exam paper.
5. Do not put methods **inside** your main method.
6. NetBeans will help you with "try catch" and the libraries you need to import.

Constructors

7. Constructors have the same name as their class.
8. You can have more that one constructor. They differ according to the **number** of parameters they accept.
9. The constructor initialises the newly created object i.e. gives the fields its initial values.
10. The constructor automatically runs code e.g. reading in the text file or loading a menu.
11. The constructor method heading does not have a return type e.g. public ~~void~~ Food ( ) public ~~String~~ Food (String n, int n)

The Interface - UI

12. Create the UI class first, regardless of the order in the exam paper.
13. The UI class has the main method.
14. The UI class instantiates an object from the manager class. It creates the manager object. e.g. FoodManager fm = new FoodManager( ); // no parameters in this example
15. The UI handles input, output and defensive coding (if asked for). The manager class has all the methods for processing.
16. The UI class calls the methods in the manager class.
17. The name of the class matches the file name EXACTLY.
18. Any other methods you need in the UI class besides the main method must be declared below and after the main method. These extra class methods must be "static".

Variables

19. Variables are either "local" to their method or "global" (declared outside of any method at the top of the class). Global variables are still generally private.
20. Global variables can be declared as final e.g. final double VAT_RATE = 0.15; // 15%

The Main Method in the UI class

21. We do not declare variables in the main method as private or public. They are local variables to main (the scope of the variable)
22. We do not declare variables inside **any** method as private or public. They are local to the method.

The Manager Class

24. Create the manager class second regardless of the order of the exam paper.
25. This has most of the methods that are called by the UI class e.g. **objectName**.methodName(parameter list).
26. The manager class may have "helper methods". These are private as they help the other methods in the manager class achieve their goals.

The Text File(s)

27. In NetBeans remember to save the text file in the project folder (not the scr folder)

The Object class – the class(es) from which you create (instantiate) objects. You will probably create an array of objects from your object class.

28. Even if "question one" starts with the creation of an object class, do not do this first. Create the UI class first (may be a later question). Create your bare bones framework first i.e. UI, managerClass and then object class
29. The object class with have global private variables, the constructor method(s), the getter methods, the setter methods and the toString method.
30. The object class with have global private variables which you create OUTSIDE (at the top) of any methods the class may have. One of these will probably be an array (to create your array of objects)

The Golden Thread - regarding parameters

31. The while loop must match the fields in the text file. The parameters in the call must match the parameters in the constructor (or method heading), which must match the variables in the object class. (see java-teacher for "Golden Thread")

Defensive coding

32. Only code defensive coding if asked for in the exam.

Currency

33. Currency is always double.
34. Know how to round off to two decimal places using DecimalFormat.

SQL – If you don't know points 2, 3 and 4 you loose 25% of the marks in the SQL section.

a) Study practice, practice study. You do not know SQL the way you should.
b) When the fields are coming from more than one table . . . You MUST **explain the join** to SQL. You must show the primary key to foreign key relationship in your SQL code. This is done after the WHERE clause.
c) Study the **UPDATE, INSERT AND DELETE** structures. They are different to SELECT. They use "UPDATE SET WHERE", "DELETE FROM WHERE" and "INSERT INTO SELECT FROM WHERE"
d) Generating a **random number** in SQL e.g. generate a random number between 10 and 99
    a. INT (RND (custID) * 90 + 10). NOTE: The customer ID field is being used as the random number generation seed so that each random number is different for each customer. Because random numbers are doubles you need the INT function to get rid of the decimal fraction.

```
/*
 * BARE BONES: Create the framework structure first
 *
 * Prelim practical 2020. Pretoria cluster 480
 * Start with question 5.1 and 5.2, then 4.1 and 4.2, then the
 * the whole of question 2, then question 3.1 and 3.2.
 * Create the text file "data.txt" in the project folder (not scr folder)
 * Then read in the text file question 4.3
 * Allow NetBeans to insert "try catch" and your imports. Do this early
 * "Scanner scFile = new Scanner(new File(fileName));"
 *
 */
package prelimprac2020;

import javax.swing.JOptionPane;

public class InterfaceUI { ✓                                    12

    public static void main(String[] args) { ✓
        String fileName = JOptionPane.showInputDialog(null, "Enter the file
        name");
        OrderManager om = new OrderManager(fileName); ✓            14

        System.out.println("Hello World");
    } // end main

} // end class

============================================================
/*
 *  Prelim practical 2020. Pretoria cluster 480
 */
package prelimprac2020;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;

public class OrderManager { ✓

    private Order[] orders = new Order[20]; ✓
19  private Food[] foods = new Food[50]; ✓                   6
    private int size = 0; ✓
    private int foodSize = 0;
    private String fileName;

    public OrderManager(String file){ ✓  7
6       try {
            fileName = file;
            Scanner scFile = new Scanner(new File(fileName)); ✓
            while (scFile.hasNext()) { ✓
                String line = scFile.nextLine();
                Scanner scLine = new Scanner(line).useDelimiter(";"); ✓
                int id = scLine.nextInt();                       30
                String v = scLine.next();
      1         String d = scLine.next(); ✓
                int q = scLine.nextInt();
                double p = scLine.nextDouble(); ✓               30
             ✓  Food f = new Food(id, d, v, q, p); // single stand alone object
                foods[foodSize] = f; // copy stand alone to the array of objects
                foodSize++; ✓
            } // end while

        } catch (FileNotFoundException ex) {
            Logger.getLogger(OrderManager.class.getName()).log(Level.SEVERE,
            null, ex);
        }
```

This bare bones solution will ensure that you pass, or even do well.

Numbers are from the notes 2020 prelim prac.

```

          System.out.println("File read.");

      } // end constructor

  } // end class

  ============================================================

  /*
   * Prelim practical 2020. Pretoria cluster 480
   */
  package prelimprac2020;
                           ✓       28
  public class Food {
      private int orderNo = 0;
  19  private String description = "";
      private String vendor = "";
      private int quantity = 0;        ✓ ✓
  ✓   private double price = 0.0;
      private double totalCost = 0.0;
      private static int noOfItems = 0;
                                                         7          30        ✓
      public Food(int inOrder, String inD, String inV, int inQ, double inP){
          orderNo = inOrder;
          description = inD;     ✓ ✓
          vendor = inV;
          quantity = inQ;
          price = inP;
          totalCost = quantity * price;  ✓
      } // end constructor

      public int getOrderNo() {
          return orderNo;   ✓
      }

      public double getTotalCost() {  ✓
          return totalCost;
      }

      public String toString() {
          String foodItem = orderNo + description + vendor + quantity + price +
          "R" + totalCost;
          return foodItem;
      }

      public static int getNoOfItems(){  ✓
          return noOfItems;

      }

  } // end class
  ==============================================================

  /*
   * Prelim practical 2020. Pretoria cluster 480
   */
  package prelimprac2020;

  public class Order {  ✓
      private int orderNo = 0;          ✓
  19  private Food[] foodArr;  ✓
      private double orderTot = 0.0;
      private final int SERVICE_FEE = 10;  ✓

                                   7
      public Order(int inNo, Food[] inFA){  ✓


```

```
        } // end constructor

        public String toString(){
            String orderDetails = "";

            return orderDetails;
        } // end toString

        public double calcOrderTot(){

            return orderTot;
        }

    } // end class
    ==============================================================
```

Bare Bones $\frac{31}{80}$ ± 40%.

plus SQL $\frac{25}{40}$ ± 63%

$\frac{56}{120}$ 47% *

Bare Bones approach. No excuse for failing the practical paper. *

1. Create UI first
2. Create Manager class second
3. Create object classes and all constructors
4. <u>Read</u> the class diagrams given
5. Read in the text file
6. Follow the <u>Golden Thread</u>
7. Read exam

Learners wanting to do well.

The "Bare Bones" gives you the correct foundation to build on.
If your foundation is faulty you will struggle to get the mark you have in mind.

Ensure that you fully understand the "Golden Thread" paradigm. See java-teacher.com for details.

2020 Practical Prelim MEMO

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package prelim4tmemo;

/**
 *
 * @author clewis
 */
//Q5.1
public class Interface ✓
{
    public static void main(String[] args)
    {
        //Q5.2
        OrderManager om = new OrderManager("data.txt"); ✓ ✓
        //Q5.3
        System.out.println(om.listFoods()); ✓
        //Q5.4
        System.out.println("Total number of items: " + Food.getNoOfItems()); ✓
        //Q5.5
        om.sort(); ✓
        //Q7.3
        om.collateOrders(); ✓
        //Q7.4
        System.out.println(om.printOrders()); ✓
    }
}
```

A		SQL	40
B	2	Food	15
	3	Order	17
	4	OrderManager	20
	5	Interface UI	6
	6	Food	9
	7	OrderManager	13

$\boxed{120}$

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package prelim4tmemo;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author clewis
 */
//Q4.1
public class OrderManager  ✓
{
    //Q4.2
    private Order[] orders = new Order[20];  ✓
    private int size = 0;
    private Food[] foods = new Food[50];  ✓
    private int foodSize = 0;  ✓  both counters

    //Q4.3
    OrderManager(String filename)  ✓
    {
        try
        {
            Scanner scFile = new Scanner(new File(filename));  ✓
            while (scFile.hasNext())  ✓
            {
                String line = scFile.nextLine();
                Scanner scLine = new Scanner(line).useDelimiter(";");  ✓
                int id = scLine.nextInt();
                String v = scLine.next();
                String d = scLine.next();  ✓
                int q = scLine.nextInt();
                double p = scLine.nextDouble();
                Food f = new Food(id, d, v, q, p); ✓
                foods[foodSize] = f; ✓
                foodSize++; ✓
            }
        } catch (FileNotFoundException ex)
        {
            Logger.getLogger(OrderManager.class.getName()).log(Level.SEVERE,
null, ex);
        }

    }

    //Q4.4
    public String listFoods() ✓
    {
        String out = "";
        for (int i = 0; i < foodSize; i++) ✓
        {
            out += foods[i].toString() + "\n"; ✓
        }
        return out; ✓
    }

    //Q4.5
    public void sort()
    {
        for (int x = 0; x < foodSize - 1; x++) ✓
        {
            for (int y = x + 1; y < foodSize; y++) ✓
            {
                if (foods[x].getOrderNo() > foods[y].getOrderNo()) ✓
                {
                    Food temp = foods[x];
                    foods[x] = foods[y]; ✓
                    foods[y] = temp;
                }
            }
        }
    }
    //Q7.1
    public void collateOrders()  ✓
    {
        Food[] temp = new Food[10];
        int tSize = 0;
        int lastID = foods[0].getOrderNo();
        for (int i = 0; i < foodSize; i++)  ✓  loop through the food items
        {
            if (foods[i].getOrderNo() == lastID)  ✓  check for change in order number
            {
                temp[tSize] = foods[i];  ✓  add to temp Food array
                tSize++;
            } else
            {
                // create order object
                Order order = new Order(lastID, temp);  ✓  create the order
                orders[size] = order;  ✓  add to array
          ✓     size++;  add to counter
                lastID = foods[i].getOrderNo();
                temp = new Food[10];
                temp[tSize] = foods[i];
                tSize=1;
            }
        }
        //add the last
        Order order = new Order(lastID, temp);  ✓  make sure last order is added
        orders[size] = order;
        size++;
    }

    //Q7.2
    public String printOrders()
    {
        String out = "";
        for (int i = 0; i < size; i++)  ✓
        {
            out += orders[i].toString() + "\n";  ✓
        }

        return out;  ✓
    }
```

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package prelim4tmemo;

import java.text.DecimalFormat;

/**
 *
 * @author clewis
 */
// Q2.1
public class Food ✓
{
    //Q2.2 correct modifier, data type and name ✓ ✓ ✓
    private int orderNo;
    private String description;
    private String vendor;
    private int quantity;
    private double price;

    private double totalCost;

    //Q2.3
    private static int noOfItems; ✓

    //Q6.1
    private String[] codes =          ✓
    {
        "SSR", "PP", "OB", "FA", "KFC"
    };
    private String[] names =
    {                                ✓
        "Spur Steak Ranch", "Pizza Perfect", "Ocean Basket", "FishAways",
"KFC"
    };

    //Q2.4
    Food(int inOrder, String inD, String inV, int inQ, double inP)   ✓
    {
        orderNo = inOrder;
        description = inD;
        vendor = inV;          ✓✓
        quantity = inQ;
        price = inP;

        totalCost = quantity * price;   ✓

        noOfItems++;  ✓
    }

    //Q2.5
    public int getOrderNo()
    {
        return orderNo;   ✓
    }

    public double getTotalCost()
    {
        return totalCost;    ✓
    }

    //Q2.6
    public static int getNoOfItems()
    {
        return noOfItems;    ✓
    }

    //Q6.2
    private String getVendorName(String code)   ✓
    {
        String name = "Unknown";  ✓ return unknown if not found
        int i = 0;
        boolean found = false;
        while (!found && i < codes.length)  ✓  loop and stop when found
        {
            if (codes[i].equalsIgnoreCase(code))   ✓  check for the code
            {
                found = true;
                name = names[i];    ✓ return the correct name
            }
            i++;
        }
        return name;
    }

    public String toString()
    {
        DecimalFormat df = new DecimalFormat("R#.00");
        //Q6.3 using DecimalFormat          ✓  ✓ replace vendor with method call
        return orderNo + " " + getVendorName(vendor) + "\t" + description +
"\t" + quantity + " " + df.format(totalCost);

        // alternative to Q6.3 using String.format
        // return orderNo + " " + getVendorName(vendor) + "\t" + description +
"\t" + quantity + " " + String.format("R%.2f",totalCost);

        //Q2.7
        //return orderNo + " " + vendor + "\t" + description + "\t" + quantity
+ " " + df.format(totalCost);                         ✓ correct format
    }
}                                                     ✓ correct fields
```

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package prelim4tmemo;

/**
 *
 * @author clewis
 */
//Q3.1
public class Order ✓                          must match the class
{
    private int orderNo;
    private Food[] foodArr; ✓       array variable
    private double orderTot;          ✓ 2 other variables

    public static final int SERVICE_FEE = 10; ✓

    //Q3/2
    Order(int inNo, Food[] inFA) ✓
    {
        orderNo = inNo;
        foodArr=inFA; ✓
        orderTot = calcOrderTot(); ✓
    }

    //Q3.3
    private double calcOrderTot()
    {
        double total=0;
        for (int i = 0; i < foodArr.length; i++) ✓
        {
            if (foodArr[i] != null) ✓
            {
                total += foodArr[i].getTotalCost(); ✓
            }
        }
        total += (total * SERVICE_FEE/100.0) + 45; ✓
        total= total*100;
        total=Math.round(total);   ✓
        total=total/100;
        return total;   ✓
    }

    //Q3.4
    public String toString()
    {
        String out="\nORDER: " + orderNo + "\n";   ✓
        for (int i = 0; i < foodArr.length; i++)
        {
            if (foodArr[i] != null)
            out += foodArr[i].toString() + "\n";   ✓
        }
        out+= "Order total: " + String.format("R%.2f",orderTot);   ✓
        return out;   ✓

        //can also use DecimalFormat to format to 2 decimal places.
}
}
```

Prelim 2020
SQL MEMO (Total: 40)

Name:		
1.1	SELECT *✓	
	FROM Customer✓	
	WHERE DelArea = 'Menlyn'; ✓	3
1.2	UPDATE Company✓	
	SET Surcharge = 15✓	
	WHERE CompanyName = 'Spar'; ✓	3
	ALTERNATIVE:	
	WHERE CompanyID = 9	
1.3	SELECT CurrentCost, AVG(Surcharge) ✓ AS [Average Surcharge] ✓	
	FROM Company	
	GROUP BY CurrentCost ✓	5
	HAVING✓ AVG(Surcharge) < 25 ; ✓	
1.4	SELECT CompanyName, CustName, OrderDate, Cost AS [Cost of Order] ✓	
	FROM Company, Customer, Orders ✓✓ (one mark if only 2 tables)	
	WHERE Company.CompanyID = Orders.CompanyID✓	
	AND Customer.CustID = Orders.CustomerID✓	
	AND MONTH✓ (OrderDate) = 7✓	
	ORDER BY✓ Cost DESC; ✓	
	Alternative (INNER JOIN)	9
	SELECT CompanyName, CustName, OrderDate, Cost AS [Cost of Order] ✓	
	FROM (Company INNER JOIN ✓Orders ON Company.CompanyID = Orders.CompanyID) ✓ INNER JOIN Customer ✓ **ON** Customer.CustID = Orders.CustomerID✓	
	WHERE MONTH✓ (OrderDate) = 7✓	
	ORDER BY✓ Cost DESC; ✓	
1.5	SELECT CustName, DelArea, LEFT(CustName,2) ✓ & ✓ RIGHT(DelArea,3) ✓ & INT ✓ (RND✓ (CustID) ✓ *90 + 10✓) AS CustCode	
	FROM Customer;	7
	-1 if concatenation done with + instead of &	
1.6	DELETE *✓	
	FROM Orders	
	WHERE YEAR (OrderDate) = 2019✓	
	OR✓ YEAR(OrderDate) = 2018;	
	1st mark for DELETE, 2nd mark for BOTH conditions, 3rd mark for OR	3
	DELETE ✓	
	FROM Orders	
	WHERE YEAR(OrderDate) ✓ IN (2018,2019) ✓	
1.7	SELECT CustName, EMail	
	FROM Customer	2
	WHERE Email LIKE✓ '*mweb*'; ✓	

	Alternative:
	SELECT CustName, EMail
	FROM Customer
	WHERE Email LIKE✓ '*@mweb.co.za' ✓	

1.8	SELECT *
	FROM Customer
	WHERE DelArea NOT ✓ IN ✓ ("Brooklyn", "Menlyn", "Sandton")
	
	No marks for multiple OR
		2

1.9	INSERT INTO ✓Company
	(CompanyName, StartHour, EndHour, CurrentCost, Surcharge) ✓
	SELECT "Uber Eats"✓(hard code), StartHour, 23✓(hard code), CurrentCost, Surcharge✓(all other fields)
	FROM Company
	WHERE CompanyName = 'KFC'; ✓
		6

x/40