**LV Practical Exam July 2010 Memo**

**Section A: SQL**

1 SELECT LocationName, Suburb FROM tblLocations ORDER BY FilmCost; (1)

2 SELECT LocationName, Suburb, FilmCost FROM tblLocations WHERE Film = TRUE; (2)

3 SELECT \* FROM tblLocations WHERE Film = FALSE AND Photos = FALSE AND

Events = FALSE; (2)

4 SELECT LocationName, PhotosCost FROM tblLocations WHERE Photos = TRUE AND (PhotosCost BETWEEN 20000 AND 24000) AND Suburb = "Camps Bay"; (3)

5 SELECT Name, Email FROM tblPeople WHERE Company = "Ads R US"; (1)

6 SELECT \* FROM tblBookings WHERE MONTH(EndDate) = 5 OR MONTH(StartDate) = 5;

(3)

7 SELECT StartDate, EndDate FROM tblBookings WHERE MONTH(StartDate) = 8; (2)

8 SELECT MIN(EventsCost) AS CheapestEvent FROM tblLocations WHERE Events = TRUE;

(2)

9 SELECT \* FROM tblLocations WHERE FilmCost = (SELECT MAX(FilmCost) FROM tblLocations WHERE Suburb = "Camps Bay"); (3)

10 SELECT AVG(PhotosCost) AS KommAvg FROM tblLocations WHERE

Suburb = "Kommetjie" AND Photos = TRUE; (2)

11 INSERT INTO tblPeople ( Name, Email, Telephone, Company )

VALUES ("Joe Bloggs", "joe@joe.com", "033 2340011", "Private"); (4)

12 UPDATE tblLocations SET FilmCost = FilmCost \* 1.11 WHERE Suburb = "Cape Town"; (3)

13 DELETE \* FROM tblBookings WHERE MONTH(StartDate) = 6 AND LocationID = 17 OR LocationID = 32 OR LocationID = 35; (3)

14 SELECT LocationName, Name, Email, Company FROM tblLocations, tblPeople WHERE OwnerID = PersonID AND Suburb = "Stellenbosch"; (4)

15 SELECT LocationName, Company, StartDate, EndDate FROM tblLocations, tblPeople, tblBookings WHERE tblBookings.PersonID = tblPeople.PersonID AND tblBookings.LocationID = tblLocations.LocationID; (5)

|  |  |  |  |
| --- | --- | --- | --- |
| **QUESTION 2** |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Question 2 - Marking Grid** | | |  |  |
| **Question** | **Aspect** |  |  | **Max** | **Learner’s** |
|  |  | **Marks** | **Marks** |
| **2.1.1** | **Object class**  Declare attributes as private and methods (subtract marks for  errors, max 4) | | | **4** |  |
|  |
| **2.1.2** | **Constructor:** Parameters: correct order(1) correct  data type(1) initialise attributes(2) | | |  |  |
| **4** |  |
| **2.1.3** | **toString** method:Definition/heading returns typestring(1), return string(1), ItemCode with calculated spaces(any acceptable way) (1), item description, all the other values as strings | | | **3** |  |
| **2.1.4** | **getSellingPrice** method: Definition correct(1),  calculation: costPrice \* 1.10(1) , return sellingPrice (1) | | | **3** |  |
| **2.1.5** | **getStockCode** method: Definition correct and return  itemCode(1) | | | **1** |  |
| **2.1.6** | **getNumInStock** method return numInStock(1) | | | **1** |  |
| **2.1.7** | **setQuantity method**: Definition correct(1) receive one values(1),Decrease instance field of the object by the value received(2) | | | **3** |  |
|  | **SUBTOTAL** | | | **19** |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **2.2.1** | Array of **PlaneSpares** created(3), int count variable declared(2) | | | **5** |  |
|  |
| **2.2.2** | |  | | --- | |  | | | initialise size(1) , while not eof (1), copy fields(1), Instantiate object with parameters(2) Assign to array(1), inc size (1) Close File(1) | | |  | | | |  |  |
| **8** |  |
| **2.2.3** | **Display** method: loop(1),  System.out.println(arr[i]).toString() (2) | | | **3** |  |
| **2.2.4** | **Sort :** Declare variable temp of type PlaneSpares(1), OuterLoop(1), InnerLoop(1), call getSellingPrice method (2) swap (1) | | | **6** |  |
| **2.2.5** | **Purchase :** Input code and initialise variables(1),Loop condition(1), call getStockCode method (1), loop to test validity(1) input quantity purchased(1), test quantity(1), call setQuantity method(1), inc counter(1) if stock code not found output suitable message(1) | | | **9** |  |
| **2.3** | Call appropriate methods in menu options. N.B typed method for 2.3.5 must be called correctly | | | **10** |  |
|  | **SUBTOTAL** |  |  | **41** |  |
|  | **TOTAL** |  |  | **60** |  |

public class PlaneSpares

{

private String itemCode;√

private String itemDescription;√

private double costPrice;√

private int quantity;√

public PlaneSpares(String iC,String iD,double c,int q)

{

itemCode = iC;√

itemDescription = iD;√

costPrice = c;√

quantity = q;√ }

public String toString()√

{

String temp; √

temp = itemCode + spaces(itemCode,20) + itemDescription +

spaces(itemDescription,32)√ + "R"+ costPrice +

spaces(""+costPrice,15)√ + quantity + spaces(""+quantity,4);√

return temp; √

}

public double √getSellingPrice()

{

double sellingPrice = costPrice √\* 1.1;√

return sellingPrice;√

}

public int√ getQuantity()

{

return quantity;√

}

public String √getStockCode()

{

return itemCode;√

}

√

public void √setQuantity(int q)

{

√ √

quantity = quantity - q;

}

public String spaces(String s,int n)

{

String spc = "";

for (int i = 0; i < n - s.length(); i++)

{

spc +=" ";

}

return spc;

}

}

import java.io.\*;

import java.util.Scanner;

public class TestPlaneSpares

{

public static void main(String [] args)√

{

PlaneSpares [] arrSpares = new PlaneSpares[10];√

int size = 0;√

try

{

Scanner inputStream = new Scanner(new

FileInputStream("Spares.txt"));√

System.out.println("Connection to Text File Established");

String line,iCode,iDesc;

double costP;

int quan;

while(inputStream.hasNext())√

{

line = inputStream.nextLine();√

Scanner sc = new Scanner(line)√.useDelimiter(";");√

iCode = sc.next();√

iDesc = sc.next();√

costP = sc.nextDouble();√

quan = sc.nextInt();√

arrSpares[size]√ = new PlaneSpares( iCode,iDesc,costP,quan);√

size++;√

}

inputStream.close();√ }

catch(FileNotFoundException f)

{

System.out.println("Error Reading From Text File");√

}

Scanner inKb = new Scanner (System.in);

char ch = ' ';

while (ch != 'Q')

{

System.out.println("\n\n");

System.out.println(" Menu");

System.out.println("");

System.out.println("A - Display spares ");

System.out.println("B - Sort ");

System.out.println("C - Purchase");

System.out.println("Q - QUIT");

System.out.println("");

System.out.print("Your Choice? :");

ch = inKb.next().toUpperCase().charAt(0);

switch (ch)

{

case 'A':

{ System.out.println("List of all Stock Items");

System.out.println("=======================");

System.out.println("Item Code\t\tItem Description\t\tCost

Price\tQuantity");√

for(int loop = 0; loop < size; loop++)√

{

System.out.println(arrSpares[loop]√.toString()√);

}

break;

}

case 'B':

{

PlaneSpares temp;√

for(int loop = 0; loop < size-1; loop++)√

{

for(int i = loop+1; i < size; i++)√

{

if(arrSpares[loop]√.getSellingPrice()√ >

arrSpares[i]√.getSellingPrice())√

{

temp = arrSpares[loop];√

arrSpares[loop] = arrSpares[i];√

arrSpares[i] = temp;√

}

}

} }

case 'C':

{

System.out.println("Enter the code of the item");

String code = inKb.next();√

int i = 0√,pos = -1;√

boolean found = false;√

int quant;

√ √

while(i < size && found == false)

{

if(arrSpares[i]√.getStockCode()√.equalsIgnoreCase(code))

{

found = true;√

System.out.println("Enter the number purchased");

do√

{

quant = inKb.nextInt();√

if (quant > arrSpares[i].getQuantity())√

System.out.println("Re-Enter the amount purchased");

}

while (quant > arrSpares[i].getQuantity());√

pos = i;√

arrSpares[i]√.setQuantity(quant);√

System.out.println("Item updated");

System.out.println(arrSpares[i].toString());

}

i++;√

}

if(pos == -1)√

{

System.out.println("The item was not located");

}

break;

}

default :

{

System.out.println("Incorrect Choice");

break;

}

case 'Q':

{

System.exit(0);

}

}

}

}

}