INTERNATIONAL SECONDARY EXAMINATION
NOVEMBER 2024

COMPUTER SCIENCE: PAPER I

MARKING GUIDELINES

Time: 3 hours 150 marks

These marking guidelines are prepared for use by examiners and sub-examiners, all of whom are required to attend a standardisation meeting to ensure that the guidelines are consistently interpreted and applied in the marking of candidates' scripts.

The IEB will not enter into any discussions or correspondence about any marking guidelines. It is acknowledged that there may be different views about some matters of emphasis or detail in the guidelines. It is also recognised that, without the benefit of attendance at a standardisation meeting, there may be different interpretations of the application of the marking guidelines.

SECTION A SQL

QUESTION 1

1.1

```
SELECT *
FROM tblSuppliers
WHERE Rating > 3
```

1.2

```
SELECT IceCreamName, Description, Size, Discount
FROM tblIceCreams
WHERE Description LIKE "*berry*"
AND Discount = Yes
```

MySQL and JavaDB:

```
SELECT IceCreamName, Description, Size, Discount
FROM tblIceCreams
WHERE Description LIKE '%berry%'
AND Discount = True
```

1.3

```
SELECT OrderID, IceCreamID, Qty, Price
FROM tblItemsOrdered
WHERE DeliveryDate BETWEEN #2024/09/13# AND #2024/09/20#
AND OrderID MOD 2 <> 0
```

Alternative:

```
SELECT OrderID, IceCreamID, Qty, Price
FROM tblItemsOrdered
WHERE DeliveryDate >= #2024/09/13#
AND DeliveryDate <= #2024/09/20#
AND OrderID MOD 2 <> 0
```

MySQL:

```
SELECT OrderID, IceCreamID, Qty, Price
FROM tblItemsOrdered
WHERE DeliveryDate BETWEEN BETWEEN '2024-09-13' AND '2024-
09-20'
AND MOD(OrderID, 2) <> 0
```

JavaDB

```
SELECT TBLORDERS.ORDERID, ICECREAMID, QTY, PRICE
FROM TBLITEMSORDERED, TBLORDERS
WHERE TBLITEMSORDERED.ORDERID = TBLORDERS.ORDERID
AND ORDERDATE BETWEEN '2024-09-13' AND '2024-09-20'
AND NOT MOD(TBLORDERS.ORDERID,2) = 0;
```


1.4

```
SELECT EmployeeName, COUNT(*) AS OrdersMade
FROM tblOrders
WHERE SupplierID = 10
GROUP BY EmployeeName
ORDER BY COUNT(*) DESC
```

Alternative:

```
SELECT EmployeeName, COUNT(*) AS OrdersMade
FROM tblOrders
WHERE SupplierID = 10
GROUP BY EmployeeName
ORDER BY 2 DESC
```

MySQL AND JavaDB:

```
SELECT EmployeeName, COUNT(*) AS OrdersMade
FROM tblOrders
WHERE SupplierID = 10
GROUP BY EmployeeName
ORDER BY OrdersMade DESC
```

1.5

```
SELECT IceCreamID, SUM (Qty)
FROM tblItemsOrdered
GROUP BY IceCreamID
HAVING SUM(Qty) > 250
```


1.6

```
SELECT IceCreamName, tblOrders.OrderID, ROUND (Price * (1 +
MarkUp), 2) AS SellingPrice
FROM tblIceCreams, tblItemsOrdered, tblOrders
WHERE tblIceCreams.IceCreamID = tblItemsOrdered.IceCreamID
AND tblOrders.OrderID = tblItemsOrdered.OrderID
AND Price * (1 + MarkUp) > 16
AND LEN (IceCreamName) <= 20
ORDER BY IceCreamName, tblOrders.OrderID
```

Alternative:

```
SELECT IceCreamName, tblOrders.OrderID, ROUND (Price * (1 +
MarkUp), 2) AS SellingPrice
FROM tblIceCreams, tblItemsOrdered, tblOrders
WHERE tblIceCreams.IceCreamID = tblItemsOrdered.IceCreamID
AND tblOrders.OrderID = tblItemsOrdered.OrderID
AND Price * (1 + MarkUp) > 16
AND LEN (IceCreamName) <= 20
ORDER BY 1, 2
```

MySQL:

```
SELECT IceCreamName, tblOrders.OrderID, ROUND (Price * (1 +
MarkUp) ,2) AS SellingPrice
FROM tblIceCreams, tblItemsOrdered, tblOrders
WHERE tblIceCreams.IceCreamID = tblItemsOrdered.IceCreamID
AND tblOrders.OrderID = tblItemsOrdered.OrderID
AND Price * (1 + MarkUp) > 16
AND LENGTH(IceCreamName) <= 20
ORDER BY IceCreamName, tblOrders.OrderID
```

JavaDB:

```
SELECT ICECREAMNAME, ORDERID,
FLOOR((PRICE*(1+MARKUP))*100+0.5)/100 AS SELLINGPRICE
FROM TBLICECREAMS, TBLITEMSORDERED
WHERE TBLICECREAMS.ICECREAMID = TBLITEMSORDERED.ICECREAMID
AND PRICE*(1+MARKUP)>16
AND LENGTH(ICECREAMNAME)<=20
ORDER BY IceCreamName, tblOrders.OrderID;
```


1.7

```
INSERT INTO tblItemsOrdered (OrderID, IceCreamID, Price,
Qty)
SELECT 20, IceCreamID, Price, INT (RND(IceCreamID) * 51 +
125)          for IceCreamID and Price fields
FROM tblItemsOrdered
WHERE OrderID = 12
```

MySQL:

```
INSERT INTO tblItemsOrdered (OrderID, IceCreamID, Price,
Qty)
SELECT 20, IceCreamID, Price, FLOOR(RAND()* 51 + 125)
FROM tblItemsOrdered
WHERE OrderID = 12
```

JavaDB:

```
INSERT INTO TBLITEMSORDERED (ORDERID,ICECREAMID,PRICE,QTY)
(SELECT 20,ICECREAMID, PRICE, INT(RANDOM()*51+125)
FROM TBLITEMSORDERED
WHERE ORDERID=12);
```

1.8

```
DELETE
FROM tblIceCreams
WHERE IceCreamID IN (1, 2, 6, 10)
```

Alternative:

```
DELETE
FROM tblIceCreams
WHERE IceCreamID = 1
OR IceCreamID = 2
OR IceCreamID = 6
OR IceCreamID = 10
```


SECTION B OBJECT ORIENTATED PROGRAMMING

QUESTION 2, QUESTION 6.1

```
//Question 2.1
//class header
//all fields private
//all fields correctly typed with correct names
public class IceCream
{
    private String name;
    private int categoryNum;
    private double markUp;
    private int qty;
    private LocalDate expiry;

    //Question 2.2
    //correct method name
    //correct parameter names and types
    //fields set to parameters, deduct 1 per mistake, max of 2
      deductions
    public IceCream (String inName, int inCategoryNum,
                double inMarkUp, int inQty, LocalDate inExpiry)
    {
        name = inName;
        categoryNum = inCategoryNum;
        markUp = inMarkUp;
        qty = inQty;
        expiry = inExpiry;
    }

    //Question 2.3
    //correct headers for accessors
    //correct returns for accessors
     public String getName()
    {
        return name;
    }

    public LocalDate getExpiry()
    {
        return expiry;
    }
```

```
    //Question 2.4
    //correct header
    //correct cost per category
    //correct calculation
    //return calculation
    public double getSellingPrice()
    {
        double catCost = 10.15;
        switch (categoryNum)
        {
            case 2 -> catCost = 11.09;
            case 3 -> catCost = 11.26;
            case 4 -> catCost = 11.65;
        }
        return (catCost + catCost * markUp / 100);
    }

    //Question 2.5
    //correct header
    //correct text for fields
    //contains all fields
    //use of getSellingPrice()
    //return formatted string with new lines
     public String toString()
    {
        return "Ice cream Name: " + name + "\n" +
               "Quantity: " + qty + "\n" +
               "Expiry Date: " + expiry + "\n" +
               "Selling Price: " + getSellingPrice() + "\n";
    }


    //Question 6.1
    //method header correct
    //compare ice cream's expiry date with current date
    //determine right boolean value
    //return boolean value
    public boolean checkExpiry()
    {
        if(expiry.isBefore(LocalDate.parse("2024-10-18")))
      //if(expiry.isBefore(LocalDate.now()))
        {
            return true;
        }
        return false;
    }
}
```

QUESTION 3

```
//Question 3.1
//class header
//class inherits from IceCream class
//all fields private
//all fields correctly typed with correct names
public class ExtraIceCream extends IceCream
{

    private int extras;
    private String allergens;

    //Question 3.2
    //constant declared with final
    //constant and named and typed correctly with correct value
    public final double PRICEPEREXTRA = 12.00;

    //Question 3.3
    //correct method name
    //correct parameter names and types
    //super method called
    //correct parameters for super method
    //fields set to parameters
    public ExtraIceCream (String inName, int inCategoryNum,
                          double inMarkUp, int inQty,
                          LocalDate inExpiry, int inExtras,
                          String inAllergens)
    {
      super (inName, inCategoryNum, inMarkUp, inQty, inExpiry);
        extras = inExtras;
        allergens = inAllergens;
    }

    //Question 3.4
    //correct header & return statement
    public String getAllergens()
    {
        return allergens;
    }
```

```
    //Question 3.5
    //correct header to override getSellingPrice method,
      @Override compiler directive is optional
    //use of the getSellingPrice method from IceCream class
    //use of class constant
    //return of new price
    @Override
    public double getSellingPrice()
    {
      return super.getSellingPrice() + extras * PRICEPEREXTRA;
    }

    //Question 3.6
    //correct header to override toString method,
      @Override compiler directive is optional
    //use of the toString method from IceCream class
    //contains all fields
    //return formatted string
    @Override
    public String toString()
    {
        return super.toString() +
               "Extras: " + extras + "\n" +
               "Allergens: " + allergens + "\n";
    }

}
```


QUESTION 4, QUESTION 6.2, QUESTION 7.1

```
//Question 4.1
//class created correctly
public class IceCreamManager
{
    //Question 4.2
    //both properties private
    //IceCream array declared with correct name
    //array size set to 100
    //size property created
    private IceCream [] iArr = new IceCream[100];
    private  int size = 0;

    //Question 4.3
    //constructor header correct
    //try catch implemented correctly to handle exception
    //open file for reading
    //loop through all lines
    //read next line from file
    //split line into required parts
    //parse date into LocalDate
    //if line contains additional data for ExtraIceCream
    //extract additional data for ExtraIceCream
    //create ExtraIceCream object when necessary
    //create IceCream object when necessary
    //add created object to array
    //increment size
    public IceCreamManager()
    {
        try
        {
            Scanner is = new Scanner(
                        new FileInputStream("IceCreams.txt"));
            while (is.hasNextLine())
            {
                String l = is.nextLine();
                Scanner lr = new Scanner(l).useDelimiter(";");
                String name = lr.next();
                int categoryNum = lr.nextInt();
                double markUp = lr.nextDouble();
                int qty = lr.nextInt();
                LocalDate expiry = LocalDate.parse (lr.next(),
                      DateTimeFormatter.ofPattern("yyyy-MM-dd"));
                if (lr.hasNext())
                {
                    int extras = lr.nextInt();
                    String allergens = lr.next();
                    iArr[size] = new ExtraIceCream (name,
            categoryNum, markUp, qty, expiry, extras, allergens);
```



```
                }
                else
                {
                    iArr[size] = new IceCream(name, categoryNum,
                                 markUp, qty, expiry);
                }
                size++;
                lr.close();
            }
            is.close();
        }
        catch (FileNotFoundException e)
        {
            System.out.println("Error, file not found");
        }
    }

    //Question 4.4
    //method header correct
    //string initialised
    //loop through IceCream array
    //append to string
    //return string
    public String toString()
    {
        String details = "";
        for (int i = 0; i < size; i++)
        {
            details += iArr[i].toString() + "\n";
        }
        return details;
    }
```



```
//Question 4.5
//method header correct
//create and initialise a counter and a total variable
//loop through IceCream array
//if inside loop to check if object is a ExtraIceCream object
//typecast ExtraIceCream object
//call getAllergens method and check if not none
//add object's price to total and increment counter
//return correct calculation total / counter
public double getAverageExtra()
{
    int s = 0;
    double sumPrice = 0;
    for (int i = 0; i < size; i++)
    {
        if (iArr[i] instanceof ExtraIceCream)
        {
            if (!((ExtraIceCream)
             Arr[i]).getAllergens().equalsIgnoreCase("None"))
            {
                sumPrice += iArr[i].getSellingPrice();
                s++;
            }
        }
    }
    return Math.round(sumPrice / s);
}
```



```
//Question 6.2
//method header correct
//try catch implemented correctly to handle exception
//open file for writing
//loop through IceCream array
//if within loop
//checkExpiry method called on object in condition of if
//write details of ice cream to text file inside of if
//close connection to text file to save writing
public void expiredIceCreams()
{
    try
    {
        PrintWriter pw = new PrintWriter(new
        FileOutputStream("ExpiredIceCreams.txt"));
        for (int i = 0; i < size; i++)
        {
            if (iArr[i].checkExpiry())
            {
                pw.println(iArr[i].toString());
                pw.println();
            }
        }
        pw.close();
    } catch (Exception e)
    {
        System.out.println("Error writing to file");
    }

}
```



```
//Question 7.1

//method header correct

//METHOD 1 using temporary array:
//create a temporary array
//set a counter to 0
//loop through IceCream array
//if within loop with check not King Cone,
  using getName: 2 marks
//copy object to temporary array
//increment counter
//change size to counter
//equate array to temporary array: 2 marks

//METHOD 2 using only existing array:
//loop through IceCream array
//while within loop with check King Kone using getName,
  to ensure that shifted object is not King Kone: 2 marks
//inner loop starting at i going to size - 1: 2 marks
//objects copied 1 to the left: 2 marks
//clearing out last object: 2 marks
//decreasing size by 1


 public void removeIceCreams()
{

     //METHOD 1:
     IceCream [] iArrTemp = new IceCream[100];
     int j = 0;
     for (int i = 0; i < size; i++)
     {
        if(!iArr[i].getName().contains("King Kone"))
        {
           iArrTemp[j] = iArr[i];
           j++;
        }
     }
     size = j;
     iArr = iArrTemp;
```



```
        //METHOD 2:
        for (int i = 0; i < size; i++)
        {
            while(iArr[i].getName().contains("King Kone"))
            {
                for (int j = i; j < size - 1; j++)
                {
                  iArr[j] = iArr[j + 1];
                }
                iArr[size - 1] = null;
                size--;
            }
        }
    }

}
```

QUESTION 5, QUESTION 6.3, QUESTION 7.3

```
//Question 5.1
//class created with main method
public class IceCreamUI
{
    public static void main(String[] args)
    {
        //Question 5.2
        //IceCreamManager created in correct position
        IceCreamManager icm = new IceCreamManager();

        //Question 5.3
        //display IceCreamManager object
        System.out.println(icm.toString());
        System.out.println();


        //Question 5.4
        //display average price for Extra Ice Creams with
          allergens
        System.out.println("Extra Ice Creams Allergens) - Average
                            Price: " + icm.getAverageExtra());
        System.out.println();

        //Question 6.3
        //call of the expiredIceCreams method
        icm.expiredIceCreams();

        //Question 7.2
        //call of the removeIceCreams method
        //display IceCreamManager object
        icm.removeIceCreams();
        System.out.println(icm.toString());
    }
}
```

Total: 150 marks