

**Programming II**

Year 1 (2021/22), Semester 2

## SCHOOL OF INFOCOMM TECHNOLOGY

Diploma in Cybersecurity & Forensics

Diploma in Data Science

Diploma in Information Technology

**COMMON TEST**

Date : 13 December 2021 (Monday)

Time : 11.00 AM – 12.30 PM

INSTRUCTIONS TO CANDIDATES:

1. This paper consists of 12 pages including this cover page.

Check carefully to make sure your set is complete.

1. There are 3 questions. Answer ALL questions.
2. This is an open book assessment.
3. You are allowed to refer to materials in the computer laptop.
4. Language translators are NOT allowed.

|  |  |
| --- | --- |
| **GRADE** |  |

**QUESTION 1** (30 marks)

A simple application is to be developed to find the intersection point of 2 straight lines. The straight line is expressed in an equation . The Class Element Diagram used for the application is shown in Figure 1 below.

|  |
| --- |
| **Line** |
| -a:int  -b:int  -c:int |
| +Line()  +Line(int,int,int)  +FindIntersectionX(Line):double  +FindIntersectionY(Line):double  +ToString():string |

Figure 1: Class Element Diagram - Line

Note

* The attribute **a** denotes the coefficient of .
* The attribute **b** denotes the coefficient of .
* The attribute **c** denotes the constant.
* The formula to determine the intersection point of 2 straight lines

and

is as follows:

|  |  |
| --- | --- |
|  | Figure 1-1: Intersection point of 2 straight lines |

Figure 1-1 shows the intersection point of 2 straight lines.

QUESTION 1 (cont.)

1. Based on the Class Element Diagram in Figure 1,
2. Write the **parameterized constructor**.

(5 marks)

|  |
| --- |
| public Line(int a, int b, int c)  {  A = a;  B = b;  C = c;  } |

1. Write the **FindIntersectionX()** method to determine the value of the intersection point.

(10 marks)

|  |
| --- |
| public double FindIntersectionX(line one)  {  return (one.B \* C - B \* one.C) / (one.A \* B - A \* one.B);  } |

QUESTION 1 (cont.)

1. Write code to find the intersection point based on user input of 2 straight line equations as shown in Figure 1(b).

Assume 2 straight lines will intersect with each other. Input validation is not required.

|  |
| --- |
| Enter the values of a,b,c of line 1 equation ax+by=c: **2,3,5**  Enter the values of a,b,c of line 2 equation ax+by=c: **6,6,3**  The intersection point (x, y): (2.5, -3) |

Figure 1(b): Sample output

Note: values underlined depict the user input.

(15 marks)

|  |
| --- |
| Console.Write("Enter the values of a,b,c of line 1 equation ax+by=c: ");  string line1 = Console.ReadLine();  string[] split1 = line1.Split(",");  line l1 = new line(Convert.ToInt32(split1[0]), Convert.ToInt32(split1[1]), Convert.ToInt32(split1[2]));  Console.Write("Enter the values of a,b,c of line 2 equation ax+by=c: ");  string line2 = Console.ReadLine();  string[] split2 = line2.Split(",");  line l2 = new line(Convert.ToInt32(split2[0]), Convert.ToInt32(split2[1]), Convert.ToInt32(split2[2]));  Console.WriteLine("The intersection point (x, y): ({0},{1})", l1.FindIntersectionX(l2), l1.FindIntersectionY(l2)); |

**QUESTION 2** (30 marks)

A delivery service offers two options. The Class Diagram used for calculating delivery costs is shown in Figure 2 below.

|  |
| --- |
| **StandardDelivery** |
| -name:string  -address:string  -distance:int |
| +StandardDelivery()  +StandardDelivery(string,string,int)  +CalculateCost():double  +ToString():string |

|  |
| --- |
| **InstantDelivery** |
| -baseFee:double |
| +InstantDelivery()  +InstantDelivery(string,string,int,double)  +CalculateCost():double  +ToString():string |

Figure 2: Class Diagram

Note

* The attributes **name** and **address** store the name and address of the recipient.
* The attribute **distance** stores the distance of the trip.
* The method **CalculateCost()** calculates the delivery cost.

For standard delivery, the cost is calculated according to the following table:

|  |  |
| --- | --- |
| **Distance (km)** | **Cost ($)** |
| 0–10 | 7 |
| Above 10 | 13 |

For instant delivery, the cost is the sum of base fee and distance fee. The distance fee is calculated according to the following table:

|  |  |
| --- | --- |
| **Distance (km)** | **Rate ($ per km)** |
| First 5 | 1 |
| Above 5 | 0.70 |

Example: The cost of 15 km distance with a base fee of $5 is calculated as:

$5 + (5 km \* $1) + (10 km \* $0.70) = $17

QUESTION 2 (cont.)

(a) List ONE method that is ***overloaded*** in Figure 2. Give your reason.

(3 marks)

|  |
| --- |
| StandardDelivery(), this method is overloaded with 3 perimeters name, address and distance. |

(b) List ONE method that is ***overridden*** in Figure 2. Give your reason.

(3 marks)

|  |
| --- |
| CalculateCost(), calculate cost in standard delivery have a different rate than instantdelivery, it will not include the baseFee and use a flat rate while for instantdelivery the calculatecost() is then overridden to return the cost for instantdelivery with the addition of baseFee cost. |

(c) Write the **InstantDelivery** class.

(16 marks)

|  |
| --- |
| internal class InstantDelivery : StandardDelivery  {  public double BaseFee { get; set; }  public InstantDelivery() { }  public InstantDelivery(string n, string a, int d, double b) : base(n, a, d)  {  BaseFee = b;  }  public override double CalculateCost()  {  if (d <= 5)  {  return BaseFee + 1 \* d;  }  else  {  return BaseFee + 5 + (d - 5) \* 0.7;  }  }  public override string ToString()  {  return base.ToString() + "\tBase Fee: " + BaseFee;  }  } |

QUESTION 2 (cont.)

(d) Write the method, **CreateDelivery()**, in the Program class that asks the user to enter information to create an **InstantDelivery** object and then to calculate and display the delivery cost to 2 decimal places as shown in Figure 2(d).

|  |
| --- |
| Enter name: **John**  Enter address: **123 Clementi Road**  Enter distance (to nearest km): **15**  Enter base fee: **5**  The cost is $17.00 |

Figure 2(d): Sample output

Note: values underlined depict the user input.

(8 marks)

**public static void CreateDelivery()**

{

|  |
| --- |
| Console.Write("Enter name: ");  string name = Console.ReadLine();  Console.Write("Enter address: ");  string addr = Console.ReadLine();  Console.Write("Enter distance (to nearest km): ");  int dist = Convert.ToInt32(Console.ReadLine());  Console.Write("Enter base fee: ");  double bf = Convert.ToDouble(Console.ReadLine());  InstantDelivery delivery = new InstantDelivery(name, addr, dist, bf);  Console.WriteLine("The cost is ${0:0.00}", delivery.CalculateCost()); |

}

**QUESTION 3** (40 marks)

The class diagram for a customer management subsystem for a retail company is shown in Figure 3 below.

|  |
| --- |
| **Customer** |
| -custId:int  -name:string  -applicableDiscount:int |
| +Customer()  +Customer(int,string,int)  +ApplyDiscount(double):double  +ToString():string |

|  |
| --- |
| **Member** |
| -membershipNo:int  -points:int |
| +Member()  +Member(int,string,int,int,int)  +RedeemPoints(int):bool  +ToString():string |

Figure 3: Class Diagram for Customer Management Subsystem

Note

* All the **Customer** objects (which can be **Customer** or **Member**) are stored in a List named **customerList**

(a) Explain, if **polymorphism** can be exhibited in the above example.

(4 marks)

|  |
| --- |
| Yes. Eventhough the List accepts Customer data type, it also can store member object as customer is the superclass of member making it able to store any subclasses. |

QUESTION 3 (cont.)

(b) Some changes are being made to the class diagram in red as shown in Figure 3(b) below.

|  |
| --- |
| **Customer** |
| -custId:int  -name:string  -applicableDiscount:int |
| +Customer()  +Customer(int,string,int)  +ApplyDiscount(double):double  +ToString():string |

|  |
| --- |
| **Member** |
| -membershipNo:int  -points:int |
| +Member()  +Member(int,string,int,int,int)  +ApplyDiscount(double):double  +RedeemPoints(int):bool  +ToString():string |

Figure 3(b): Changes made

Note

* The applicable discount, **applicableDiscount**, in **Customer** class is the discount (%) that can be applied for the customer’s purchases.
* **ApplyDiscount()** in **Customer** class, receives the purchase amount as parameter, calculates and returns the discount amount.
* **ApplyDiscount()** in **Member** class, receives the purchase amount as parameter, calculates and returns the discount amount. In the calculation, members get additional of 5% discount on top of the applicable discount.
* **RedeemPoints()** in **Member** class, receives the number of membership points to be redeemed, deducts it from the Member’s points and returns **true** if deduction is successful. The method will return **false** if there are not enough points for deduction.

QUESTION 3 (cont.)

Write a method, **ApplyDiscountToCustomers()** in Program class, that receives a purchase amount, then calculates and displays the discount amount of all the customers in the **customerList**.

Sample output is given in Figure 3(b)-1 for the case when the purchase amount is $1000.

|  |
| --- |
| Entitled Discount per $1000:  Cust Id Cust Name Discount amt ($)  1212 Gregory Peck 100.00  1213 Michelle Tan 145.00  1214 Tom 164.00 |

Figure 3(b)-1: Sample output

(12 marks)

**public static void ApplyDiscountToCustomers(List<Customer> customerList, int purchaseAmount)**

{

|  |
| --- |
| Console.WriteLine("Entitled Discount per $1000:");  Console.WriteLine("{0,-10} {1,-10} {2,-10}", "Cust Id", "Cust Name", "Discount amt ($)");  for (int i = 0; i < customerList.Count; i++)  {  Console.WriteLine("{0,-10} {1,-10} {2,-10:0.00}", customerList[i].CustId, customerList[i].Name, (Convert.ToDouble(purchaseAmount) - customerList[i].ApplyDiscount(Convert.ToDouble(purchaseAmount))));  } |

}

QUESTION 3 (cont.)

(c) During year end, all the members are supposed to use 100 points for administrative purposes. Write a method, **RedeemMemberFee()** in Program class, to deduct 100 points from all **members** in the **customerList**. It will display the members who did not have enough points. If there is no such member, the program will also print a message to inform as such.

Two sample outputs are given in Figure 3(c)-1 & Figure 3(c)-2.

|  |
| --- |
| Members with unsuccessful deduction of membership fee:  Cust Id Cust Name Points  1212 Gregory Peck 80  1214 Tom 50 |

Figure 3(c)-1: Sample output 1

|  |
| --- |
| There is no member with unsuccessful deduction of membership fee. |

Figure 3(c)-2: Sample output 2

(18 marks)

**public static void RedeemMemberFee(List<Customer> customerList)**

{

|  |
| --- |
| Console.WriteLine("Members with unsuccessful deduction of membership fee:");  Console.WriteLine("{0,-10} {1,-10} {2,-10}", "Cust Id", "Cust Name", "Points");  int fail = 0;  for (int i = 0;i < customerList.Count;i++)  {  if (customerList[i] is Member)  {  Member member = (Member)customerList[i];  if (member.RedeemPoints(100))  {  Console.WriteLine("{0,-10} {1,-10} {2,-10}", member.CustId, member.Name, member.Points);  }  else  {  fail++;  }  }  }  if (fail > 0)  {  Console.WriteLine("There is {0} member with unsuccessful deduction of membership fee", fail);  }  else  {  Console.WriteLine("There is no member with unsuccessful deduction of membership fee.");  } |

}

QUESTION 3 (cont.)

(d) An interface, **IPayable,** is defined as shown in Figure 3(d).

|  |
| --- |
| **<<interface>>**  **IPayable** |
|  |
| ***+makePayment()*** |

Figure 3(d): Interface

The **makePayment()** in **IPayable** interface makes payment via a specific mode.

Describe if changes need to be made to the **Customer** and **Member** class to use the **IPayable** interface.

All customers will make use of the interface to make payment.

You are not required to write code for the **IPayable** interface.

(6 marks)

|  |
| --- |
| IPayable interface will be added to the customer class as it is the superclass of the member class, making able to compare between a member and a customer for the appropriate payment method. Member class will not change as it cannot receive a customer data type. |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of paper \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*