**QUESTION 1** (30 marks)

A simple application is to be developed to keep track of the modules taken by the students. The Class Element Diagram used for the application is shown in Figure 1 below.

|  |
| --- |
| **Module** |
| -code:string  -cw:double  -exam:double |
| +Module()  +Module(string,double,double)  +ComputeTotalMark():int  +ComputeGrade():char  +ToString():string |

Figure 1: Class Element Diagram - Module

Note

* The attribute ***code*** stores the code of the module.
* The attribute ***cw*** stores the coursework mark.
* The attribute ***exam*** stores the exam mark.
* The method ***ComputeTotalMark()*** computes (and return) the *total mark* for the module (rounded to nearest integer).
* The method ***ComputeGrade()*** computes (and return) the *grade* for the module based on the following criteria:

|  |  |
| --- | --- |
| **Total Mark** | **Grade** |
| total mark >= 80 | A |
| 70 <= total mark < 80 | B |
| 60 <= total mark < 70 | C |
| 50 <= total mark < 60 | D |
| total mark < 50 | F |

Note: The ComputeTotalMark() method returns an integer value.

QUESTION 1 (cont.)

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

(4 marks)

|  |
| --- |
| public Module(string c, double cw, double e)  {  Code = c;  Cw = cw;  Exam = e;  } |

1. Write the **ComputeGrade()** method to calculate and return the grade for the module.

(8 marks)

|  |
| --- |
| public char ComputeGrade()  {  int totalMark = ComputeTotalMark();  if (totalMark < 50)  {  return 'F';  }  else if (totalMark < 60)  {  return 'D';  }  else if (totalMark < 70)  {  return 'C';  }  else if (totalMark < 80)  {  return 'B';  }  else  {  return 'A';  }  } |

QUESTION 1 (cont.)

1. Assume a List, ModuleList, has been created in the application and is initialized with many modules (i.e. Module objects).

(i) Write the method, **ListModules()**, in the application class to display the details of all the modules in the ModuleList as shown in Figure 1(b)(i).

|  |
| --- |
| S/No Code CW Exam Total Grade  1 M01 75.0 69.0 73 B  2 M02 83.0 78.0 81 A  3 M03 78.0 83.0 80 B  4 M04 65.0 75.0 69 C  5 M05 75.0 65.0 71 B  . ... .... .... .. . |

Figure 1(b)(i): Sample output

(10 marks)

|  |
| --- |
| static void ListModules(List<Module> cpModule)  {  Console.WriteLine("{0,-10} {1,-15} {2, -10} {3,-10} {4,-10} {5,-10}", "S/No", "Code", "CW", "Exam", "Total", "Grade");  for (int i = 0; i < cpModule.Count; i++)  {  Console.WriteLine("{{0,-10} {1,-15} {2, -10:0.0} {3,-10:0.0} {4,-10} {5,-10}", (i+1), cpModule[i].Code, cpModule[i].Cw, cpModule[i].Exam, cpModule[i].computeTotalMark(), cpModule[i].ComputeGrade());  }  } |

QUESTION 1 (cont.)

(ii) Write the method, **AddModule()**, in the application class to add a new module to the ModuleList as shown in Figure 1(b)(ii).

|  |
| --- |
| Enter module code : **M06**  Enter cw mark : **77.5**  Enter exam mark : **68.5**  The module is added successfully. |

Figure 1(b)(ii): Sample output

Note: values underlined depict the user input.

(8 marks)

|  |
| --- |
| static void AddModule()  {  Console.Write("Enter module code : ");  string code = Console.ReadLine();  Console.Write("Enter cw mark : ");  double courswork = Convert.ToDouble(Console.ReadLine());  Console.Write("Enter exam mark : ");  double exam = Convert.ToDouble(Console.ReadLine());  moduleList.Add(new Module(code, courswork, exam));  Console.WriteLine("The module is added successfully.");  } |

**QUESTION 2** (30 marks)

An E-Commerce company is developing an application to keep track of its members. The Class Diagram used for the application is shown in Figure 2 below.

|  |
| --- |
| **Customer** |
| -id:int  -name:string  -amount:double |
| +Customer()  +Customer(int,string,double)  +ToString():string |

|  |
| --- |
| **Member** |
| -bonusPoints:int |
| +Member()  +Member(int,string,double,int)  +CalculateBonusPoints():int  +ToString():string |

Figure 2: Class Diagram

Note

* The attribute ***amount*** stores the amount spent by a customer.
* The attribute ***BonusPoints*** stores the bonus points accumulated by a member.
* The method ***CalculateBonusPoints()*** calculates the bonus points awarded to a member each time he/she make a purchase. The bonus points is calculated based to the following table:

|  |  |
| --- | --- |
| **Amount($)** | **Bonus Points** |
| amount < 1000 | 0 |
| 1000 <= amount <= 5000 | 100 |
| amount > 5000 | 200 |

QUESTION 2 (cont.)

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

(3 marks)

|  |
| --- |
| Customer() |

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

(3 marks)

|  |
| --- |
| ToString() |

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

(16 marks)

|  |
| --- |
| internal class Member : Customer  {  public int bonusPoints { get; set; }  public Member() { }  public Member(int id, string n, double a, int bp) : base(id,n,a)  {  bonusPoints = bp;  }  public int CalculateBonusPoints()  {  if (Amount < 1000)  {  return 0;  }  else if (Amount <= 5000)  {  return 100;  }  else  {  return 200;  }  }  public override string ToString()  {  return base.ToString() + "\tBonus Points: " + bonusPoints;  }  } |

QUESTION 2 (cont.)

(d) Write the method, **AwardBonusPoints()**, in the application class to award bonus points to a member when he/she make a purchase. The bonus points awarded is to be added to the accumulated bonus points by the member.

(8 marks)

**static void AwardBonusPoints(Member m)**

{

|  |
| --- |
| Console.WriteLine("Current bonus points: " + m.bonusPoints);  m.bonusPoints += m.CalculateBonusPoints();  Console.WriteLine("Update bonus points: " + m.bonusPoints); |

}

**QUESTION 3** (40 marks)

A Fruit Shop has decided to develop an online application to allow its customers to order items (i.e. fruits) online. The shop sells many types of fruits such as apples, grapes, oranges and watermelons. The Class Diagram used for the application is shown in Figure 3 below.

|  |
| --- |
| **Item** |
| -id:int  -name:string  -price:double |
| +Item()  +Item(int,string,double)  +CalculateAmount():double  +ToString():string |

|  |  |  |
| --- | --- | --- |
| **Apple** |  | **Grape** |
| -qty:int |  | -weight:double |
| +Apple()  +Apple(int,string,double,int)  +CalculateAmount():double  +ToString():string |  | +Grape()  +Grape(int,string,double,double)  +CalculateAmount():double  +ToString():string |

Figure 3 : Class Diagram

Note

* The attribute *name* stores the name of the fruit (e.g. “Fuji Apple”, “Purple Grape”).
* The attribute *price* stores the unit price of the fruit either by per piece or per kg.
* The attribute *qty* stores the number of pieces of the fruit.
* The attribute *weight* stores the weight of the fruit in kilogram (kg).
* The method *CalculateAmount()* calculates the amount of the item purchased based on the formula:

amount = price x qty (for apples)

amount = price x weight (for grapes)

QUESTION 3 (cont.)

(a) Analyze the three C# statements given below. State clearly with an explanation whether each of the statement causes any compilation error.

1. Item item1 = new Grape(201, “Purple Grape”, 6.00, 1.50);

(2 marks)

|  |
| --- |
| No. Item is the parent class of grape, it is able to store child class grape. |

1. Apple item2 = new Item(101, “Green Apple”, 0.80);

(2 marks)

|  |
| --- |
| Yes. Apple is the child class of item, parent class item is not able to be stored as an child class apple. |

(iii) Grape item3 = new Apple(102, “Fuji Apple”, 1.20, 5);

(2 marks)

|  |
| --- |
| Yes. Grape and Apple are both child class with no relation which causes a type mismatch. |

QUESTION 3 (cont.)

(b) (i) Write the **CalculateAmount()** method in the **Item** class.

(2 marks)

|  |
| --- |
| public virtual double CalculateAmount()  {  return 0;  } |

(ii) Write the **CalculateAmount()** method in the **Apple** class.

1. marks)

|  |
| --- |
| public override double CalculateAmount()  {  return Price \* Qty;  } |

(iii) Write the **CalculateAmount()** method in the **Grape** class.

(4 marks)

|  |
| --- |
| public override double CalculateAmount()  {  return Price \* Weight;  } |

QUESTION 3 (cont.)

(c) Assume that the application uses a List named **ItemList** to store all the items (i.e. Apple and Grape objects).

1. Write the method, **ListItems()**, in the application class to display the information of all the items. A sample output is shown below.

|  |
| --- |
| S/No Id Name Price($)  1 101 Fuji Apple 1.20  2 102 Green Apple 0.80  3 201 Fuji Grape 8.00  4 202 Purple Grape 6.00  . ... ............. .... |

Figure 3(c)(i) : Sample Output

(8 marks)

**public static void ListItems(List<Item> ItemList)**

{

|  |
| --- |
| Console.WriteLine("{0,-30} {1} {2,-57} {3,-65:0.00}", "S/No", "Id", "Price($)");  int count = 1;  foreach (Item s in ItemList)  {  Console.WriteLine("{0,-30} {1} {2,-57} {3,-65:0.00}", count, s.Id, s.Name, s.Price);  count++;  } |

}

QUESTION 3 (cont.)

1. Write the method, **OrderItem()**, in the application class to order fruits online. Two sample outputs are shown below.

|  |
| --- |
| S/No Id Name Price($)  1 101 Fuji Apple 1.20  2 102 Green Apple 0.80  3 201 Fuji Grape 8.00  4 202 Purple Grape 6.00  . ... ............. ....  Enter S/No of item : **2**  Enter quantity : **8**  Amount payable ($) : 6.40 |

Figure 3(b)(ii) : Sample Output 1

|  |
| --- |
| S/No Id Name Price($)  1 101 Fuji Apple 1.20  2 102 Green Apple 0.80  3 201 Fuji Grape 8.00  4 202 Purple Grape 6.00  . ... ............. ....  Enter S/No of item : **3**  Enter weight (kg) : **1.50**  Amount payable ($) : 12.00 |

Figure 3(c)(ii) : Sample Output 2

Note : values **underlined** depict the user input.

(16 marks)

**public static void OrderItem(List<Item> ItemList)**

{

|  |
| --- |
| Console.Write("Enter S/No of item : ");  int itemNo = Convert.ToInt32(Console.ReadLine());  Item Chosen = ItemList[itemNo - 1];  if(Chosen is Apple)  {  Console.Write("Enter Quantity : ");  double q = Convert.ToDouble(Console.ReadLine());  Apple a = (Apple)Chosen;  a.Qty = q;  Console.WriteLine("Amount payable ($) : {0:0.00}", a.CalculateAmount());  }  else if(Chosen is Grape)  {  Console.Write("Enter weight (kg) : ");  double w = Convert.ToDouble(Console.ReadLine());  Grape g = (Grape)Chosen;  g.Qty = w;  Console.WriteLine("Amount payable ($) : {0:0.00}", g.CalculateAmount());  } |

}

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