Rewarding Learning

ADVANCED SUBSIDIARY (AS)
General Certificate of Education
2024

Software Systems Development

Unit AS 1

Introduction to Object Oriented Development

[SDV11]

FRIDAY 24 MAY, MORNING

MARK SCHEME

1 The following statements relating to an object-oriented environment are either true or false. Write the word true or false beside each statement in the table below.

Statement	True/False
It is not possible to directly create an instance of an abstract class.	True [1]
Polymorphism facilitates the treatment of objects from diverse classes as instances of a shared parent class.	True [1]
An interface in object-oriented programming provide a complete implementation of methods.	False [1]
The primary purpose of encapsulation is to hide an object's internal state and shows only essential functionalities.	True [1]
Method overloading allows you to define multiple methods in the same class with the same name but different parameters	True [1]
Encapsulation allows the functionality and data in a class to be randomly accessed by code outside the class.	False [1]

[6] 6

2
```
public bool PlantCode(string plantName, string plantCode)
   {
      if (plantCode.Length != 6)
     {
        return false;
      }
     else if (!plantCode.StartsWith("#"))
     {
         return false;
      }
     else if (plantName.Substring(0, 3) != plantCode.Substring(1, 3)
     {
        return false;
     }
    else if (!int.TryParse(plantCode.Substring(4, 2), out int codeNumber) ||
     codeNumber < 11 || codeNumber > 99)
    {
       return false;
    }
    return true;
 }
```

Return type bool	[1]
Both parameters passed	[1]
Check for correct length	[1]
Check that code starts with #	[1]
Check that code includes the first three letters of plant name in the correct position	[1]
Check number is a number	[1]
Check number is within range	[1]
Correct use of the return statement	[1]

8

Accept alternative solution

AVAILABLE MARKS

AVAILABLE MARKS

3 **(a)** Sample solutions

C# Solution:	**Java Solution:**
int[,] ScreenSales = new int[6, 2];	int[ ][ ] ScreenSales = new int[6, 2];

Correct data type [1]
Correct use of brackets [1]
Correct numbers used to show length of rows and columns [1]

(b) Sample solutions

C# Solution:	**Java Solution:**

public void UpdateSales(int screen, int fpTicketsSold, int dpTicketsSold)
{
ScreenSales[screen−1,0] +=fpTicketsSold;
ScreenSales[screen−1,1] +=dpTicketsSold;
 	
Console.WriteLine(“Ticket sales updated for Screen:”+screen);
}
	public void updateSales(int screen, int fpTicketsSold, int dpTicketsSold)
	{
	ScreenSales[screen−1][0] +=fpTicketsSold;
	ScreenSales[screen−1][1] +=dpTicketsSold;
	
	System.out.print("Ticket sales updated for Screen:"+screen);
	}

Return type void [1]
All parameters passed [1]
Array updated correctly [1]
Suitable output message with screen number [1]

(c) Sample solutions

AVAILABLE MARKS

C# Solution:

```
   public void WeeklyFigures()
   {
         double totalRevenue=0;

       for (int screen = 0; screen
< ScreenSales.GetLength(0);
screen++)
       {
          int fpTicketsSold=
ScreenSales[screen, 0];
          int dpTicketsSold=
ScreenSales[screen, 1];

          double screenRevenue =
(fpTicketsSold* 12) + (dpTicketsSold
* 5.5);

          totalRevenue +=
screenRevenue;

          Console.WriteLine("Screen
{0} Weekly Figures:", screen + 1);
          Console.WriteLine("Full price
Tickets Sold:"+ fpTicketsSold);
          Console.
WriteLine("Discounted price Tickets
Sold:"+ dpTicketsSold)\n");
       }

       Console.WriteLine(Overall Total
Revenue: £"+totalRevenue);
   }
}
```

Java Solution:

```
public void weeklyFigures() {

       for (int screen = 0; screen <
ScreenSales.length; screen++)
{
         Int Screen = screen+1;
          int fpTicketsSold=
ScreenSales[screen][0];
          int dpTicketsSold =
ScreenSales[screen][1];
          double screenRevenue =
(fpTicketsSold* 12) + (dpTicketsSold *
5.5);

          totalRevenue +=
screenRevenue;

          System.out.println("Screen "+
Screen + " Weekly Figures:");
          System.out.println("Full price
Tickets Sold:" + fpTicketsSold);
          System.out.println("Discounted
price Tickets Sold:" +childTicketsSold)\
n");
       }

       System.out.println("Overall Total
Revenue: £" +totalRevenue);
   }
```

Return type void	[1]
Correct loop	[1]
Correct calculation of full price ticket	[1]
Correct calculation of discounted	[1]
Running total updated	[1]
Suitable details output for each screen	[1]
Suitable output for total revenue	[1]

Accept alternative solution

(d) Sample solutions

AVAILABLE MARKS

C# Solution:	**Java Solution:**

int[,] updatedScreenSales = new int[7, 2];
for (int screen = 0; screen < ScreenSales.GetLength(0); screen++)
{
  updatedScreenSales [screen, 0] = ScreenSales [screen, 0];
  updatedScreenSales [screen, 1] = ScreenSales [screen, 1];
}
updatedScreenSales [6, 0] = 0;
updatedScreenSales [6, 1] = 0;
ScreenSales = updatedScreenSales;
	int[][] updatedScreenSales = new int[7, 2];
	
	for (int screen = 0; screen < ScreenSales.length; screen++) {
	  updatedScreenSales [screen][0] = ScreenSales [screen][0];
	  updatedScreenSales [screen][1] = ScreenSales [screen][1];
	}
	updatedScreenSales [6, 0] = 0;
	updatedScreenSales [6, 1] = 0;
	ScreenSales = updatedScreenSales;

Creating new array	[1]	
Correct loop	[1]	
Populating new array	[1]	
Setting screen 7 values to zero	[1]	
Replacing ScreenSales array with new array	[1]	19

4 **(a)** Encapsulation [1]

AVAILABLE MARKS

(b) **C# Solution:**

```
public Order(string customerName, bool businessCustomer, char
    packageCode)
{
  CustomerName = customerName;
  BusinessCustomer = businessCustomer;
  PackageCode = packageCode;
  OrderDate = DateTime.Today();
}
```

Correct constructor and fields passed [1]
Correct assignment of passed fields [1]
Correct assignment for orderDate [1]

(c)

```
public double totalCost()
{
    double costExcludingVAT = 0;

    switch (packageCode)
    {
       case 'A':
          costExcludingVAT = 4500;
          break;
       case 'B':
          costExcludingVAT = 6900;
          break;
       case 'C':
          costExcludingVAT = 10000;
         break;
      default:
         return  0;
    }

    if (!businessCustomer)
    {

       return costExcludingVAT *1.2;
    }
    else
    {
       return costExcludingVAT;
   }
 }
```

Correct return type [1]
No parameters passed [1]
Declaration of variable [1]
packageCode checked [1]
Any correct assignment of cost [1]
Check if businessCustomer [1]
Correct calculation of tax [1]
Correct return [1]

Accept alternative solution

(d) Sample solutions

AVAILABLE MARKS

C# Solution:	**Java Solution:**

public DateTime InstallationDate()
{
return OrderDate.
AddMonths(4);
}
	public LocalDate InstallationDate()
	{
	return OrderDate.
	plusMonths(4);
	}

Modifier is public [1]
Return type [1]
Calculation to add 4 months [1]
Correct return [1]

(e)
```
public void checkPostcode (string clientsPostcode)
  {
    int spaceIndex = clientsPostcode.IndexOf(" ");
     bool isFound = false;

  foreach (string code in CurrentPostcodes)
  {
    if (code == clientsPostcode.Substring(0, spaceIndex)
    {
        isFound true;
      }
    }
       if (isFound)
       {
            Console.WriteLine(“The postcode is valid and panels can be
    supplied and installed in that location.");
     }
         else
    {
    Console.WriteLine(“This postcode is outside of the current area we
    supply and install solar panels.");
    }
  }
```

Correct heading with parameter passed [1]
Find index of space [1]
Declare variable needed in solution [1]
Loop [1]
Check if code is valid [1]
Use of substring [1]
Suitable message outputted [1] 23

Accept alternative solution

constructors	format	object	range
datatype	initialisation	overloading	reused
default	lower	parameters	test data
dependency	methods	presence	upper
failed	modifiers	properties	verification

AVAILABLE MARKS

5 **(a)** In a program, **constructors**, **properties** and **methods** should be tested to make sure they are properly defined. Methods used in a program can be tested and **reused**.
Validation of input data requires a number of different checks. Validation could include a **presence** check to ensure that data has been entered. If data must be between an **upper** and **lower** limit a **range** check should be used. Sometimes data must follow a set pattern and a **format** check should be used in this case.
A test plan should be constructed for a piece of software to ensure testing has been conducted properly and so that **failed** tests can be corrected.

1. constructors/properties/methods
2. properties/methods/constructors
3. methods/constructors/properties
4. reused
5. presence
6. upper
7. lower
8. range
9. format
10. failed

[1] for each **two** correct answers [5]

(b)

AVAILABLE MARKS

Test number	Test data	Reason for test	Expected outcome	Outcome (description – only if different from expected)	Further development
1	hrs = ten	To ensure an integer is passed	Error message appears on screen	✓	none
2	–1/0	To check that a number below minimum value is not accepted	Error message appears on screen	✓	none
3	hrs = 25	To check correct rate of pay is applied for valid hours up to and including 35	Correct value appears on screen	Incorrect value appeared on screen	Check and fix calculation in method
4	hrs = 40	To check correct rate of pay is applied for valid hours up to 35 and correct rate of pay for hours up to and including 42	Correct value appears on screen	✓	none
5	hrs = 44	To check correct rate of pay is applied for valid hours up to 35 and correct rate of pay for hours up to 42. And correct rate is applied for hours over 42	Correct value appears on screen	✓	none

[1] for each correct answer.

Accept alternative suitable correct answers [5] 10

6 (a) C# Solution:

AVAILABLE MARKS

C# Solution 1

```
public void DiscountRate()
  {
     if (numGuidedWalks == 2)
     {
        discountRate =0;
     }
     else if (numGuidedWalks == 4)
     {
        discountRate =0.05;
     }
     else
     {
        discountRate =0.10;
     }
  }
```

Correct header	[1]	
No parameters passed	[1]	
Correct Check	[1]	
Setting discountRate	[1]	[4]

C# Solution 2

```
public double DiscountRate()
  {
     if (numGuidedWalks == 2)
     {
        return 0;
     }
     else if (numGuidedWalks == 4)
     {
        return 0.05;
     }
     else
     {
        return 0.10;
     }
  }
```

Correct header	[1]	
No parameters passed	[1]	
Correct Check	[1]	
return discountRate	[1]	[4]

(b) C# Solution:

AVAILABLE MARKS

```
public class HillwalkingCourse : HillwalkingExperience
{
    private int numGuidedWalks;
    private double discountRate;

public HillwalkingCourse(string bookingName, string description,
string difficulty, DateTime bookingDate, bool pickupService, double
experiencePrice, int numGuidedWalks)
        : base(bookingName, description, difficulty, bookingDate,
pickupService, experiencePrice)
    {
        NumGuidedWalks = numGuidedWalks;
        discountRate=DiscountRate(); OR DiscountRate()  // based on
        solution from part (a)

    }

    public int NumGuidedWalks
    {
        get
        {
            return numGuidedWalks;
        }
        set
        {
            numGuidedWalks = value;
        }
}
```

: HillwalkingExperience or extends HillwalkingExperience	[1]
Field definitions	[1]
Fields passed for both HillwalkingCourse and HillwalkingExperience	[1]
No discount rate passed	[1]
Correct use of :base	[1]
Passing of HillwalkingExperience fields	[1]
Assignment of numGuidedWalks	[1]
Setting discount rate	[1]
Correct property/method heading	[1]
Correct get	[1]
Correct set	[1]

AVAILABLE MARKS

(c)
```
public virtual double CalculatePrice()
    {

      If(pickupService)
      {
      return experiencePrice+8;
      }
      Else
      {
      return experiencePrice;

      }

    }
```

Use of virtual [1]
Correct return type [1]
Check and calculation for additional service [1]
Correct Return [1]

(d) C# Solution:

```
public override double CalculatePrice()
   {
      return base.CalculatePrice() +numNights*35;
   }
```

Use of override [1]
Correct use of base.CalculatePrice() [1]
Calculations for number of nights price [1]
Correct Return [1]

(e) C# Solution:

```
   public void BookingType()
   {
      int hillWalkingExperience = 0;
      int hillWalkingCourse = 0;
      int mountainRetreat = 0;
      int totalExperiences = Experiences.Length;
      double totalCost = 0;

for (int x = 0; x < totalExperiences; x++)
   {
      if (Experiences[x].GetType() == typeof(HillwalkingExperience))
      {
            totalCost += ((HillwalkingExperience)Experiences [x]).
            CalculatePrice();
            hillWalkingExperience ++;
      }
      else if (Experiences[x].GetType() == typeof(HillwalkingCourse))
      {
            totalCost += ((HillwalkingCourse)Experiences [x]).CalculatePrice();
            hillWalkingCourse ++;
      }
      else
      {
            totalCost += ((MountainRetreat)Experiences [x]).CalculatePrice();
            mountainRetreat ++;
```

```
        }
    }
}

double hillWalkingExperiencePercentage = hillWalkingExperience /
totalExperiences*100);
double hillWalkingCoursePercentage = hillWalkingCourse /
totalExperiences*100);
double mountainRetreatPercentage = mountainRetreat /
totalExperiences*100);

Console.WriteLine("Total number of bookings for hillWalkingExperience was:
{0}, with an overall percentage of  all experiences:{1}%.",
 hillWalkingExperience, hillWalkingExperiencePercentage;

Console.WriteLine("Total number of bookings for intermediate walks:
{0}, with an overall percentage of all walks:{1}%.", hillWalkingCourse,
hillWalkingCoursePercentage;

Console.WriteLine("Total number of bookings for experienced walks:
{0}, with an overall percentage of all walks:{1}%.", mountainRetreat,
mountainRetreatPercentage;

Console.WriteLine("The total income generated from all experiences was:
{0}.", totalCost;

}
```

		AVAILABLE MARKS
Declaring variables	[1]	
Any correct loop	[1]	
Any correct Checking statement	[1]	
Use of experiences.length	[1]	
Any correct Checking object type	[1]	
Any correct Casting object type	[1]	
Call of CalculatePrice()	[1]	
Any Adding to count	[1]	
Any correct Calculation for percentage	[1]	
Correct formatted output statement for all Experiences	[1]	
Output for total income	[1]	34
	Total	**100**