Rewarding Learning

ADVANCED
General Certificate of Education
2018

Software Systems Development

Unit AS 1

Introduction to Object Oriented Development

[SDV11]

THURSDAY 24 MAY, AFTERNOON

MARK SCHEME

1

Term	Definition	Mark
Instantiation	The creation of an object.	[1]
Inheritance	The creation of a new class that reuses, extends, and modifies the behaviour that is defined in another class.	[1]
derived/sub/child	A class which **inherits** the **visible properties, methods and events** of the **super/base** class and can be customised with **additional properties, methods**, and extends.	[1] each – any 2 elements in bold
Polymorphism	A primary concept of object-oriented programming which allows **sub/ derived class methods** to be invoked through a **super/base class reference** during **run-time**. This is enabled through **late binding** and overriding (or meaning).	[1] each – any 3 elements in bold
Late Binding	The connection by a **polymorphic base object** to an **overriding method** during **runtime** when the **object type is known**. Used in polymorphism.	[1] each – any 3 elements in bold

[10]

AVAILABLE MARKS

10

AVAILABLE MARKS

2 **(a)** Client Field types:

(i) int
String
DateTime
String
String
[1] each [5]

(ii)
```
public Client(int clientNo, String companyName, DateTime dateJoined,
String telNo, String password)
{
    this.clientNo = clientNo;
    this. companyName = companyName;
    this.dateJoined = dateJoined;
    this.telNo = telNo;
    this.passWord = password;
    this.rating = 'A';
}
```

[1] parameter types, [1] five parameters, [1] no rating
[1] any correct assignment
[1] rating assignment as character A [5]

(iii) GET / SET - C#

```
public char Rating {          [1] type,  [1] no brackets
    get { return rating; }    [1]
    set { rating = value; }   [1]
}
```

OR GET / SET java example

```
public char getRating()                    [1] alt
{
    return  rating;                        [1] alt
}
public void setRating( char rating)  {     [1] alt
    this.rating = rating                   [1] alt
 }
```
[4]

(b) A method to determine the **MaximumCredit** due on a job.

```
public double MaximumCredit( ){

    double max=0.0;
    switch(rating){
        case 'A': max= 2500.0; break;
        case 'B': max= 7500.0; break;
        case 'C': max= 10000.0; break;
        case 'X': max= 0.0;  }
    return max;
}
```

[1] return type, [1] no parameters,
[1] correct initialisation
[1] switch / if
[1] comparison –: case / ==
[1] single quotes for character
[1] any correct assignment
[1] correct return [8]

22

AVAILABLE MARKS

3 **(a)** **(i)** Public – visibility, allows access by all classes [1]
Static – single instance – use class name to access [1] [2]

(ii) Example – e.g. Char.IsDigit(..), Math.PI() [1] [1]

(b)

```
public static Boolean validPassWord(String passWord){
   int countUpper = 0, countDigit = 0, countSpace=0;
   for(int x=0;x<passWord.Length;x++)
   {
       if (Char.IsDigit(passWord[x]))
              countDigit++;
       else
           if(Char.IsUpper(passWord [x]))
         //if ((Int16)passWord[x] >= 65 && (Int16)passWord[x]
  <= 90)
                   countUpper++;
            else
            if (Char.IsWhiteSpace(passWord[x]))
                     countSpace++;
   }
   if (countDigit > 1 && countUpper > 0
&& passWord.Length >= 7 && countSpace==0)
      return true;
   else
      return false;
    }
```

[1] return type, [1] parameter
[1] count initialisation
[1] loop
[1] correct check as digit
[1] correct check of Uppercase
[1] use of methods isDigit, isUpper / ascii code 40 or 65
[1] check length
[1] check all
[1] correct return value
[1] if space considered [11] 14

Allow alternative vaild answers.

AVAILABLE MARKS

4 **(a)** Sample c# answer
IOException
Trap any Input Output error such as unavailable file
[1] any Exception,
[1] explanation [2]

(b) **(i)** Exception – base class for Exception handling, handles *all* exceptions, all other Exception classes derived from it, hierarchial in catch system, placed last in catch sequence
([1] each for any two) [2]

(ii) Sample C#

```
{
    if (!validTelNo(value))
        throw new ClientException("Error – appropriate message
        for telephone numbers");
    else
        telNo = value;
 }
```

[1] if
[1] call of method, [1] parameter
[1] throw, [1] new, [1] Exception class – Client Exception or Exception,
[1] error message (not allowed with Exception)
[1] set of valid value [8]

(iii) use Property setting/implement check

[2] property setting/direct call method **or** [1] duplicate check [2]

(c) Any **five** from:
Sample answer
Loop around data entry
Use try -catch
try around set of the new telephone number
catch ClientException
output the returned exception message in the catch
by messageBox / errorProvider
set error flag
(5 x [1])

[1] correct placement [6] 20

AVAILABLE MARKS

5 **(a)** Override [1]

(b) **(i)** sample answer

```
public class Wedding:Party
{
```
Class Wedding [1], extends [1], Party [1],

```
private int[] noRooms= new int[2];
```
[1] noRooms array declaration, ([0] if any other field included)

```
public Wedding(int eventNo, String description, double
pricePerHead, int noBooked, DateTime, dateOfEvent, String
bandName, double bandCost, int tableType, int[]noRooms)
: base(eventNo, description,  pricePerHead,  noBooked,
dateOfEvent, bandName, bandCost, tableType)
{
     NoRooms= noRooms ;

}
```
[1] noRooms array declaration, [1] all fields
[1] base, [1] pass of fields
[1] assignment noRooms

```
public int[ ] NoRooms
{
  set { noRooms = value; }
        get { return noRooms; }
 }
```
[1] Get, [1] Set of noRooms array (ignore header) [11]

(ii) C# sample

```
public double costOfRooms( ){
double total =0.0;
double [] prices= {175.00, 130.00};
for( int x =0; x< noRooms.Length; x++)
total+= noRooms[x] * prices[x];
return total;
}
```
[1] header return type, [1] no parameters
[1] initialisation of total, [1] initialisation of prices
[1] loop, [1] calculation, [1] index
[1] return [8]
Allow two calculations/loop and 1 calculation

(iii) sample C# answer

```
public override double income ()
{
      return base.income() + costOfRooms(); [brackets necessary]
}
```
[1] return
[1] base call income(), [1] call costOfRooms() [3]

23

			AVAILABLE MARKS

6	(a)	double total = 0.0;
		for(int x =0; x< arrayEvent.Length; x++)
		{
		total += arrayEvent[x].income();
		}
		Console.Writeline(String.Format("{0:–30} {1:c}"," Income for all events is ", total));
		
		[1] declaration of total, [1] loop,
		[1] total addition, [1] array index, [1] method call
		[1] output
			[6]

	(b)	double totalWedding = 0.0;
		int countWedding =0;
		for(int x =0; x< arrayEvent.Length; x++)
		{
		if (arrayEvent[x].GetType() == typeof(Wedding))
		OR if (arrayEvent[x] is Wedding)
		{ totalWedding += ((Wedding)arrayEvent[x]).income();
		countWedding++;
		
		}
		}
		
		Console. Writeline (String.Format {0:–30} {1:c},"Income for Wedding", totalWedding)
		Console. Writeline (String.Format ({0:–30} {1}, "Number of Weddings", countWedding)
		[1] declaration of total and count,
		[1] placement of Wedding check in loop
		[1] check of Event type
		[1] total addition **or** [1] count increment
		[1] output wedding total **or** [1] number of weddings
		
		Note Java uses instanceOf for type comparison
			[5]	11

			Total	**100**