Rewarding Learning

ADVANCED SUBSIDIARY (AS)
General Certificate of Education
2017

Software Systems Development

Unit AS 1:

Introduction to Object Oriented Development

[A1S11]

MONDAY 15 MAY, AFTERNOON

MARK SCHEME

General Marking Instructions

Introduction
Mark schemes are published to assist teachers and students in their preparation for examinations. Through the mark schemes teachers and students will be able to see what examiners are looking for in response to questions and exactly where the marks have been awarded. The publishing of the mark schemes may help to show that examiners are not concerned about finding out what a student does not know but rather with rewarding students for what they do know.

The Purpose of Mark Schemes
Examination papers are set and revised by teams of examiners and revisers appointed by the Council. The teams of examiners and revisers include experienced teachers who are familiar with the level and standards expected of students in schools and colleges.

The job of the examiners is to set the questions and the mark schemes; and the job of the revisers is to review the questions and mark schemes commenting on a large range of issues about which they must be satisfied before the question papers and mark schemes are finalised.

The questions and the mark schemes are developed in association with each other so that the issues of differentiation and positive achievement can be addressed right from the start. Mark schemes, therefore, are regarded as part of an integral process which begins with the setting of questions and ends with the marking of the examination.

The main purpose of the mark scheme is to provide a uniform basis for the marking process so that all the markers are following exactly the same instructions and making the same judgements in so far as this is possible. Before marking begins a standardising meeting is held where all the markers are briefed using the mark scheme and samples of the students' work in the form of scripts. Consideration is also given at this stage to any comments on the operational papers received from teachers and their organisations. During this meeting, and up to and including the end of the marking, there is provision for amendments to be made to the mark scheme. What is published represents this final form of the mark scheme.

It is important to recognise that in some cases there may well be other correct responses which are equally acceptable to those published: the mark scheme can only cover those responses which emerged in the examination. There may also be instances where certain judgements may have to be left to the experience of the examiner, for example, where there is no absolute correct response – all teachers will be familiar with making such judgements.

AVAILABLE MARKS

1 (i) **Instantiation** is the creation of an object to the design of a class.

(ii) __ **Object** _____ is the base building block of an object-oriented system and all __ **classes** _______ are derived from it.

(iii) _ **overloading** ____ occurs where two or more methods have the same name but different ___ **signatures** ___, even if their return types differ.

(iv) Customising a super / base method within a derived / sub class is known as _ **overriding** _____.

(v) Multiple inheritance is implemented in C#/Java through the use of _ **interfaces** _____.

(vi) A primary concept of object-oriented programming is _ **polymorphism** _. It allows sub / derived class methods to be invoked through a super / base class reference during run-time. This is enabled through _ **late binding**.

([1] each) [9] 9

2 (a) // constant variable hourlyRate holding value 20.

```
C#  –        private const double hourlyrate = 20.0; allow int.
or  Java –   private static final double HOURLYRATE = 20.0;
```

([1] const/static final, [1] assignment) [2]

```
// GET and SET  - C#
public int  NoHours {                    [1] type/[1] if no brackets
      get { return noHours; }            [1]
      set { noHours = value; }           [1]
}
```

OR GET/SET java example

```
public int getNoHours()                             [1] alt
{
      return  noHours;                              [1] alt
}
public void setNoHours (double noHours)   {         [1] alt
      this. noHours = noHours;                      [1] alt
}
```

[4]

```
//    A method to determine the productionCost due on a job.
public double productionCost( ){
      return noHours * hourlyRate + materialCost + specialismCost
}
```

[1] return type, [1] no parameters, [1] correct return [3]

```
//    A method to determine the salePrice for a job.
public double salePrice( ){
      return productionCost() * 1.4
}
```

[1] return type, [1] no parameters,, [1] calculation, [1] return [4]

$\downarrow$

(only if method used with brackets)

AVAILABLE MARKS

(b) A program using the class Job has the following two lines of code:

```
Job [ ] jobArray = new Job[150];
```

(i) Instantiate array
size 150
References objects of type Job
Initialised to null – no references set
[1] each any two, with correct terms used [2]

(ii) Available lamps for sale - sample solution

```
int size =72;
double totalProductionCosts= 0.0;
for(int x=0; x<size; x++){
      if(jobArray[x].Category == 'A' &&  jobArray[x].SaleDate.Year ==
      0001)
      {      Console.WriteLine(jobArray[x].toString());
             totalProductionCosts+= jobArray[x].ProductionCost();
      }
      Console.WriteLine("\n\nTotal production cost   " +
      totalProductionCosts);
      }
```

[1] initialisation of totalproductionCosts
[1] loop, [1] terminator (72 or variable)
[1] check of category (penalise if not == or correct quote marks),
(Do not penalise if full date comparison used but note CompareTo against it)
[1] check date,
[1] call of toString()
[1] running total, [1] ouput [8] 23

3 Sorting is a common activity conducted on data in information processing.

(a) (i) Selection, Bubble, Quicksort, Insertion or other
[1]

The **bubble** sort is a simple sorting algorithm not suitable for large sets of data due to O(n^2) performance. The bubble sort passes sequentially over a list, comparing each value to the adjacent following value. Their positions are swapped if the first value is greater. Each pass finds the maximum item and puts it at the end and so the *list* to be sorted can be reduced at each pass. A boolean variable is used to track if any swaps have been made in the current pass. The algorithm terminates when a pass completes without any swaps (boolean variable is false).

OR pseudocode as follows (assuming 0-based indexing):

```
repeat
    swap = false
    decrease listSize  by 1
     repeat index  x from 0 to listSize
        if (list[x]) > (list (x + 1))
            swap list items
            swap = true
until swap = false
```

[1] each - description of sequence, pass, swap method, termination [5]

(ii) Allow alternative to chosen sort in **(i)**
Sample illustration of bubble

	5	16	11	7	26	
1st pass	5	**11**	**7**	16	26	2 swaps(true)
2nd pass	5	**7**	11	16	26	1 swap (true)
3rd pass	5	7	11	16	26	0 swap (false) – terminate

[1] each for any two correct swaps
[1] any correct pass list involving swap
[1] correct termination
[1] all correct [5]

Reverse Bubble

Pass 1	5	16	11	7	26
	16	5			
	16	11	5		
	16	11	7	5	
	16	11	7	26	5
Pass 2	16	11	26	7	5
Pass 3	16	26	11	7	5
Pass 4	26	16	11	7	5

n – 1 passes

Interchange/Selection example
Interchange/Selection sort is a simple sort with O(n^2) performance. Each pass finds the minimum value, swaps it with the value in the first position of the current pass list. The pass list is reduced each time. It does no more than *n* swaps.

Sample illustration of Selection sort

	5	16	11	7	26	
1st pass	5	11	7	16	26	starting at position 1 – no swap
2nd pass	5	7	11	**16**	26	starting at position 2 swap 7 and 16
3rd pass	5	7	11	16	26	starting at position 3 no swap
4th pass	5	7	11	16	26	starting at position 4 no swap

QuickSort example

Popular sort which selects a pivot value and moves all lesser values to the left and all greater values to the right. Recursion is used to sort the left and right partitions until a single value remains. Generally has O(n log n) performance but worst can be O(n^2) .

Quicksort(array, startPosition, endPosition)

Sample illustration of Quicksort sort

```
                5   16   11    7   26
```

Pivot 11 – swap with number in first position

```
               11   16    5    **7**   26
```

Move *lo* to first number from left greater than pivot – 16

Move *hi* to first number from right less than pivot – 7 and swap

```
               11    7    5   16   26
```

```
repeat         11    7    5   16   26
```
hi passes *lo* –

to end partition swap hi with pivot

```
                5   16   11    7   26
```

(b) **(i)** C# - interface IComparable
Or Java - interface Comparable
[1] each [2]

(ii) Sample answer java

```
public  int compareTo(Object obj){
        Product  productObj = (Product)obj;
        if(this.category.compareTo(productObj.category) < 0)
                return -1;
        else if(this.category.compareTo(productObj.category) > 0)
                return 1;
        else
                if(this.price  < productObj.price)
                        return -1;
                else if( this.price  > productObj.price)
                        return 1;
                else
                        return 0;
}
```

[1] visibility, [1] return type, [1] add correct name CompareTo parameter

A {*[1] comparison of lesser category, [1] return*
[1] comparison of greater category, [1] return}

B {*[1] check lesser price on same category*
[1] check greater price on same category
[1] Return zero on same price}

[1] if class fields used – not the get/set [11] 23

Alternate marks

A	**B**
[1] use of CompareTo	[1] compareTo
[1] return	[1] return
[1] greater	[1] lesser/greater
[1] lesser	

Alternative answer to 3(b)(ii)

```
int comp = this.category.Compare to (product Obj.category);
if (comp = = 0)
//code as for price comparison

return comp
```

AVAILABLE MARKS

4 **(a)** Sample C# answer

```
Console.WriteLine(String.Format("\n {0,-8} {1,–15}  {2:D}  {3:C}   {4:C}",
stockNo, model, qtyInStock, price, (qtyInStock * price ));
```

Sample java formats %-8s %-15s %2d %8.2f %10.2f

[1] layout – minimum of 4 fields,
[1] for appropriate format any one of the numbers,
[1] correct stock value [3]

(b) C# [Serializable]
Or Java implements Serializable [1]

(c) C#/Java
Open and Read contents of file, stock.dat into a stream.
Set strm to reference the stream
Java [1] instantiation of type, [1] open file stock [1] declaration
C# [1] assign [1] Open Read Stockfile
C# Instantiate object of type BinaryFormatter to handle bit patterns
(Java instantiate object of type ObjectInputStream to handle stream as objects and primitive data)
C# *[1] instantiation of type, [1] handle patterns*   *JAVA [1] handle stream*
(create)   *[1] reference object strm*
[1] instantiation
Read a pattern from stream cast as one type Stock object
Into current position in an array of Stock objects.
C# Java *[1] read primitive data/object, [1] pattern of stock object*
C# only *[1] current position index, [1] array of stock objects* [8]

(d) IOException
SerializableException IndexOutOfRange not acceptable
FileNotFound
Exception
[1] each for any two [2] 14

AVAILABLE MARKS

5 (a) Class object cannot be instantiated *[1]*
More data required to describe valid event type, derived *[1]* [2]

(b) Sample answer

```
public double HostedIncome()
{
      return (ticketcharge -  venueChargePerPerson) * noAttended +
      donation;
}
```

[1] correct header, [1] calculation of ticket profit, [1] addition of donation,
[1] return [4]

(c) C# example

```
class FlagDay: Event                                [1]
{
      private int noOfBoxes;
      private double[ ] arrayBoxAmount;             [1]

      FlagDay(int eventNo, Date eventDate, String branchName, int
startTime, int endTime, int noOfBoxes)
      :base(eventNo, eventDate, branchName, startTime, endTime)
      {
            NoOfBoxes = noOfBoxes;
      }
```

[1] all relevant parameters, [1] base, [1] base fields
[2] call of Set Property /method ([1] if Set code repeated)

```
      public int NoOfBoxes
      {
            get {     return noOfBoxes;}
            set {     this.noOfBoxes = value;
                      arrayBoxAmount = new double[noOfBoxes];}
      }
```

[1] header, [1] assignment , [1] array instantiation in constructor or set,
[1] return

OR GET / SET java example

```
      public int getNoOfBoxes()                     [1] either correct header alt
      {
            return  noOfBoxes;                      [1] return alt
      }
      public void setNoOfBoxes(int noOfBoxes)   {
            this. noOfBoxes = noOfBoxes;                  [1] assignment alt
            arrayBoxAmount = new double[noOfBoxes];     [1] instantiation alt
      }
```

[11]

(d)

```
public double FlagDayIncome( ){
      double total =0.0;
      for(int x =0; x< arrayBoxAmount.Length; x++)
            total+= arrayBoxAmount[x];
      return total;
}
```

[1] header return type, [1] no parameters
[1] initialisation
[1] loop, [1] calculation, [1] index or if foreach used correctly
[1] return [7]

24

AVAILABLE MARKS

6 Sample C# answer

```
public double BranchIncome (Event [ ] arrayEvent,  String r
requiredBranch, int currentSize)
{
        double total = 0.0;
        for( int x =0; x< currentSize; x++)
        {
                if(arrayEvent[x].BranchName.CompareTo (requiredBranch) ==0)
                {
                        if (arrayEvent[x].GetType() == typeof( Hosted))
                        OR   if (arrayEvent[x] is Hosted)
                                total += ((Hosted)arrayEvent[x]).HostedIncome();
                        else
                                total += ((FlagDay)arrayEvent[x]).FlagDayIncome();
                }
        }
        return total;
}
```

[1] initialization total, [1] loop
[1] decision equality of branch name,
[1] check of Event type
[1] total addition, [1] either income method use of array index
[1] return

Note: Java uses instanceOf for type comparison [7] 7

Total **100**