Rewarding Learning

ADVANCED
General Certificate of Education
2022

Software Systems Development

Unit AS 1

Introduction to Object Oriented Development

[SDV11]

WEDNESDAY 25 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.

COVID-19 Context

Given the unprecedented circumstances presented by the COVID-19 public health crisis, senior examiners, under the instruction of CCEA awarding organisation, are required to train assistant examiners to apply the mark scheme in case of disrupted learning and lost teaching time. The interpretation and intended application of the mark scheme for this examination series will be communicated through the standardising meeting by the Chief or Principal Examiner and will be monitored through the supervision period. This paragraph will apply to examination series in 2021–2022 only.

		AVAILABLE MARKS
1 **(a)**	A Return statement is reached and a value is returned to the calling program. The value must conform to the return type given in the method header. An anticipated illegal or invalid operation is attempted and an Exception is thrown. An unexpected illegal operation is performed and the program crashes. (allow) [1 mark] for the basic condition from any one of the above. [1 mark] for detail. [2]	
(b) **(i)**	Common process required many times by many other classes. One occurrence only/only one copy at Type level Referenced explicitly by the class name Results Referenced implicitly if the class Results is imported. [1 mark] each for any two [2]	
(ii)	sample: ans = Results.GradeLevel(totalMark); [1 mark] [1] Note: if **Results**. is not given the class import must be shown or be the reason given in **(i)**	
(iii)	[1 mark] An object of type Order can call the method Cost() Many instances of the method can exist. Can be overridden [1 mark] for either or other valid difference [2]	
(iv)	sample answer Order myOrder = new Order(); Double cost = myOrder.Cost(); [2]	9

2 Sample answer C#.

```
public Boolean ValidCode( String itemCode){
    boolean result = true;
    if(String.IsNullOrEmpty( itemCode ) OR itemCode.Length != 5)
        return false;
    elseif( itemCode[0] != 'O' AND itemCode[0] != 'M'  OR
        itemCode[1] != '-'      OR
        itemCode[4] != 'N' AND itemCode[4] != 'S'  itemCode[4] != 'W' )
            return false;
    else{
        try{
            int num = Int32.Parse( itemCode.Substring( 2,2) )
            if (num <10 OR num >55)
                return false;
            return result;
        }catch (FormatException ef){
            return false;
```

[1 mark] header
[1 mark] check null string
[1 mark] check length is 5 characters
[1 mark] check character[0] is O or M
[1 mark] check character[1] is –
[3 – (1 mark each)] check character[4] is N, S or W
[1 mark] convert correct characters to integer
[1 mark] catch non-integer
[1 mark] check range
[1 mark] correct logic of return false
[1 mark] correct logic of return true [13] 13

3 A type that is defined as a class is a **reference** type. When a class variable of this type is declared at **run** time, the variable contains the value **null** until the creation of an **instance** of the class by using the **new** operator, or it is assigned to an existing object of a **compatible** type.
When the object is created, enough **memory** is allocated for that specific **object**, and the variable holds only a **reference** to the **location** of the object.
[1 mark each word in bold] [10] 10

AVAILABLE MARKS

		AVAILABLE MARKS
4 **(a)** **(i)**	[1 mark] Exception Handing	
(ii)	[1 mark] Property/Set method [2]	
(b)	public Item (String itemID, String StyleID, styleID, string patternID, char material, double price)	
	{ ItemID = itemID; StyleID = styleID; PatternID = patternID; Material = material; Price = price; }	
	[1 mark] constructor class name [1 mark] fields passed as parameters [1 mark] use of Property/Set method [3]	

(c)	Sample answer C#
	public double Price{
	get { return price; }
	set { if (ValidPrice(value))
	price = value;
	else
	throw new ItemException (“ Price must be in range (10–900) ”);
	}
	}	

	[1 mark] type [1 mark] set [1 mark] call validation method [1 mark] check validity [1 mark] throw customised exception(with meaningful message) [1 mark] new [6]	
(d)	reference to: [1 mark] try [1 mark] catch [1 mark] finally [1 mark] correct placement of data entry code or use of finally [1 mark] handling of exception [5]	16

5 (a)

AVAILABLE MARKS

field	class	Reason
Address1	Order	Required for postage of items or confirmation of client/customised patterns sent for agreement
startDate	OrderCourse	Starting date of course allowing scheduling /end date to be determined *Allow Order but complicated by agreement date check or duplication??*
agreementDate	OrderCustom	Date coloured design pattern is agreed by client/printing can begin.

[1 mark each] placed in valid class
[1 mark each] valid reason for chosen class [6]

(b) [1 mark] No - circled
[1 mark] valid object can be created
[1 mark] for standard order (but not a course –if referenced in answer) [3]

(c) C#
header
[1 mark] OrderCourse :
[1 mark] Order
Field Definitions
[1 mark] only fields from OrderCourse
Field constructor
[2 mark] Relevant fields from Order [1] + all fields from OrderCourse [1]
[1 mark] except progress (and orderDate) from Order (initialised in class)
[1 mark] base
[1 mark] pass Order field values as parameters to base
[1 mark] move only OrderCourse values to class fields [9]

(d)
```
public override double Cost() {
      double total = 0.0;
      total  =  noClients * ( 20 * noOfDays  + base.Cost() * 0.5 );
      return total;
}
```

[1 mark] override/return
[1 mark] return type
[1 mark] no parameters
[1 mark] initialise (allow if implicit)
[1 mark] cost for noOfDays
[1 mark] cost for selected item(s)
[1 mark] total for one client
[1 mark] total for noClients [8]

(e) [1 mark] override [1]

27

6 **(a)**

OrderCourseResident	
Int	noSharing
Int	noSingleOccupancy
Boolean/char	catered

[1 mark] line drawn from OrderCourse to new class
[1 mark] appropriate name for class e.g.
[1 mark] noSharing – no sharing/no single occupancy
[1 mark] char catered or Boolean [4]

(b) number sharing must be even and not exceed 12
number single occupancy must not exceed 6
number single occupancy + no sharing must be equal to noClients
If char used for catered then ensure presence e.g. Y or N
(No mark for Boolean as it is automatically initialised)
[1 mark each for any two of the above] [2]

(c) Note: This sample answer will depend on the student's chosen fields
e.g. Using field noSharing from OrderCourseResidential and noClients from OrderCourse

```
private override double Cost() {
    double total = 0.0, roomCost = 50;
    // cost for 1 night for all clients
    total  =   (noClients – noSharing) * roomCost *0.70      +
               noSharing *  roomCost  /   2;
    if( catered ) {
               total += noClients *7.50;
    }
    total *= (noOfDays – 1);              //duration   allow number of days
    total  +=  base.Cost() ;              //add cost of course
    return total;
}
```

[1 mark] correct equation for sharing
[1 mark] correct equation for single occupancy
[1 mark] correct price per person sharing / correct price for single
[1 mark] correct for catered
[1 mark] cost for single plus double plus catered
[1 mark] cost for noOfDays (ignore previous wrong values)
[1 mark] addition of cost for course
[1 mark] return of total
[1 mark] correct reference to fields in OrderCourse [9]

AVAILABLE MARKS

15

			AVAILABLE MARKS
7	**(a)**	[1 mark] Polymorphism (accept late binding) [1 mark] Run time [2]	
	(b)	Private ItemSales [ ] ArrayItemSales = new ItemSales[56]; [1 mark] correct class and array [1 mark] new or correct size of array [2]	

	(c)	sample answer that holds position in array
		private void HighestSale(ItemSales[ ] arrayItemSales) {
		Int highestsale = 0;
		Double income =0.0;
		for(int x=1 ; x≤55; x++)
		{ if (arrayItemSales[x].totSold> arrayItemSales[highestsale].totSold)
		highestsale = x;
		}
		income = arrayItemSales[highestsale].itemS.price * arrayItemSales.
		[highestsale].totSold;
		
		Console.WriteLine(" Highest sale Item :- "+arrayItemSales[highestsale]
		.itemS.toString());
		Console.WriteLine(" Income generated :- " +String.Format("0:C", income));
		
		[1 mark] return type, parameter
		[1 mark] correct loop
		[1 mark] correct highest sale
		[1 mark] calculate income for highest sale item
		[1 mark] output toString() for highest sale Item
		[1 mark] output formatted income [6]
			10

		Total	**100**