Rewarding Learning

ADVANCED SUBSIDIARY (AS)
General Certificate of Education
2014

Software Systems Development

Unit AS1:

Introduction to Object Oriented Development

[A1S11]

TUESDAY 20 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 Classes describe the **type** of objects, while objects are usable instances of classes. The act of creating an object is called **instantiation**. Inheritance, together with **encapsulation** and polymorphism, is one of the three primary **characteristics** (or pillars) of object-oriented programming. Inheritance enables you to create a new class that **reuses**, extends, and modifies the **behaviour** that is defined in another class.
Encapsulation means that a group of **related** properties, methods, and other members are treated as a **single** unit or object. Polymorphism means that you can have **multiple** classes that can be used interchangeably, even though each class **implements** the same properties or methods in different ways.

([1] × 10) [10] 10

2 **(a)** Explanation to include:
(basic definition of structure)

Name:
name of method or visibility [1]

Parameters:
pass parameters by value,
value of – min, max, row, col [1]
reference value for object- String [1]

Return type:
return statement for enter_No_Of_Items – value returning [1]
(description of void) [1]

Expansion on:
passing parameters by reference or by value) or visibility [1]

(b) **Sample answer (code in java or C#):**

Data declaration (for any 1) [1]

```
String str
int num;
boolean okFlag;
```

```
loop
    okFlag = true          (control of loop)                    [1]
    prompt /enter number of items to str                        [1]
    if( change str type to int is ok  )                        [1]
            if( range check invalid)                            [2]
                    output error message of range
                    set okFlag = false
    else                                          either error message [1]
            output error message of data type
            set okFlag = false
while(okFlag is false)
clear error message
return num                                                      [1]
```

(c) Sample answer java

		AVAILABLE MARKS
data declaration or correct return type		
boolean okFlag = true;	[1]	
int first =0, length = telNo.length();		
use string methods for telephone number:		
if(length < 9)	[1]	
okFlag = false;		
check digits		
if(telNo.charAt[0] == '+')	[1]	
first = 1;		
for(int x = first; x< length; x++)	[1]	
if(! Character.isDigit(telNo.charAt[x]))	[1]	
okFlag = false;	[1]	
return okFlag	[1]	
(student may combine both string methods and checks.)		
(d) noOfItems = enter_No_Of_Items(min, max);	[1]	
validTelNo = validTelephoneNo(telNo);	[1]	23

AVAILABLE MARKS

3 **(a)** class Hamper{

```
private        double      weight;
private        char        nextDayDelivery;
public Hamper( )
{
      weight = 0;
      nextDayDelivery = ' ';
}
```

C# sample

A **constructor** method with fields

```
public Hamper( double  weight,  char nextDay )                    [1]

{
      Weight = weight;                                            [1]
      NextDayDelivery = nextDay;                                  [1]
}
```

GET and **SET** (Properties / Methods)

C# Sample

```
public  double Weight()                                           [1]
{
      get { return weight; }                                      [1]
      set { weight = value; }                                     [1]
}
```

Java Sample

```
public char getNextDayDelivery()
{
      return nextDayDelivery;
}                                                                 [1]
public void setNextDayDelivery( char nextDay)
{                                                                 [1]
      this.nextDayDelivery = nextDay;
}                                                                 [1]
```

A **Method** to determine delivery cost (**standard delivery charge + next day delivery**)

```
public double  deliveryCost()                                     [1]
{
      double cost=0;                                              [1]
      if ( weight > 15 )
                  cost = 16;                                      [1]
      else if (weight >= 10)
                  cost = 10;                                      [1]
      else if (weight >= 5)
                  cost = 4.25;                                    [1]
      else
                  cost = 2.50;                                    [1]

      if( nextDayDelivery = = 'Y')
                  cost += cost /100 *30;                          [1]
                  return cost;                                    [1]
      }
```

AVAILABLE MARKS

(b) instantiate object

Sample answer C#

Hamper hamper = new Hamper(hamperWeight , nextDayDelivery) [2]

1 mark instantiation

1 mark passing/setting values

output cost of delivery for the hamper:

Console.WriteLine("\n Cost of delivery = " + hamper.deliveryCost()) [2]

1 mark write statement

1 mark method call

21

4 (a) Fixed length array must set space required for all expected data items on creation.

Array items of same type – example int, double, object

Array has name

Items can be accessed by an index numbered from 0 to array length -1

New array area must be allocated if more items required and array copied to new area

([1] × any 3) [3]

Sample

```
int [ ] myNums = new int[ 10];
```

or

```
String [ ] myNames  = { "Ann", "Bill", "John", "Mary", "Peter"};
```

([1] × 1) [1]

myNums[0] [1]

(b) Sample C#

```
int totalRainfall = 0,  noBelowAverage = 0;
```

double average; (1 mark each any 2) [2]

```
for( int x = 0; x < rainfall.Length; x++)
{
      totalRainfall += rainfall[x];
}
```

[1]

```
average = (double) totalRainfall / rainfall.Length;
```

[1]

```
for( int x = 0; x < rainfall.Length; x++)
{
      if( rainfall [x] < average)
                  noBelowAverage++;
}
```

[1]

for either loop [1]

```
Console.WriteLine("\n\t Number of days with below average rainfall
of   " + average + " is " + noBelowAverage);
```

[1]

(c) (i) Check if current number is greater than target number [1]

Exit loop. On average this will greatly enhance efficiency. [1]

AVAILABLE MARKS

(ii) Discuss why this search routine might not be effective for a large array of sorted numbers and why the **binary search** method would improve the efficiency.

Number of sequential comparisons is significant as position of target is towards end of structure.
Sorted array would allow for search from either end.
Average number of comparisons is half of array length.
Number of comparisons is maximum if number greater than last number in array.
Could half search area until found or not present
or any other valid reason

([1] × any 3) [3]

(iii)

(1) Target number to be searched for = 30

4	6	10	17	21	30	42

start = 0

mid = (i 3)

last = (ii 6) [2]

4	6	10	17	21	30	42

start = (iii 4)

mid = (iv 5)

last = (v 6) [3]

(2) Terminate search if target found or **last** < **start**

(target not in array) [1]

23

			AVAILABLE MARKS
5	**Encapsulation:** Modifiers – private, protected, public ([1] × any 2)	[2]	
	Overloading: can use the same method name multiple times the passed arguments changed (return type does not apply) (within a class or hierarchy of classes)	[1] [1]	
	Overriding: derived class can modify functionality of an inherited method (c# explicit - use of 'new' / 'override' word in derived class method of same name) (java implicit – uses override method , can invoke base method using super) overriding method has visibility. ([1] × 2)	[2]	
	Advantages: reusability of code modification of functionality required only in derived class testing required for derived class only greater reliability of structure or any other valid reason ([1] × any 3)	[3]	9
6	**(a)** Class definition; C# public class Sculpture : ArtWork Java public class Sculpture extends ArtWork	[1]	
	Fields: private, type, name	[1]	
	Constructors: Empty constructor Field constructor Use of : / **super** to pass base parameters (1 mark super; 1 mark child fields)	[1] [2]	
	Properties / GetSet: (1 mark any method name; 1 mark any get; 1 mark any set)	[3]	
	Override toString() method Call of super.toString()	[2]	
	(b) **(i)** ArtWork [ ] artWorksArray = new ArtWork[50]; (array class 1 mark; instantiation 1 mark)	[2]	
	(ii) artWorksArray[0] = new Sculpture(…….) 1 mark class type …… base data followed by sculpture data	[1] [1]	14
		Total	**100**