Rewarding Learning

ADVANCED SUBSIDIARY (AS)
General Certificate of Education
2025

Software Systems Development

Unit AS 1

Introduction to Object Oriented Development

[SDV11]

THURSDAY 29 MAY, MORNING

MARK SCHEME

1 Sample answer

AVAILABLE MARKS

Inheritance

- Inheritance allows a derived class to reuse code from a base class.
- The derived class inherits the fields, properties and methods of the base class.
- It creates an “is-a” relationship between a base class and a derived class.
- Common functionality is placed in the base class to avoid code duplication.
- Inheritance allows the derived class to use or override the base class’s members.
- The derived class can also define its own additional variables and methods.
- Inheritance supports polymorphism and improves code maintainability.
- Multiple inheritance not supported with classes, unless using interfaces.

[1] for any key point × 2

Overloading

- Method overloading allows multiple methods with the same name [1] but different parameter lists [1].
- The methods must differ by the number, type or order of parameters.
- Overloading is a form of compile-time polymorphism (early binding), where the method to be executed is determined at compile time based on the method signature being used.
- Overloading happens within the same class.
- It is resolved at compile-time (polymorphism).

[1] for any key point × 2

Overriding

- Method overriding allows a derived class to provide a new implementation of a method in the base class.
- The method name and parameter list must be exactly the same as in the base class.
- The method in the base class must be marked as virtual.
- The method in the derived class must use the override keyword.
- Overriding is an example of runtime polymorphism (late binding).
- It allows the derived class to provide specialist behaviour while keeping a common interface.

[1] for any key point × 2 [6] 6

2 (a) (i) Sample solutions

C# Solution:

```
public static bool validLicencePlate(string
plateNumber)
    {
    if (plateNumber.Length != 7)
    {
        return false;
    }
    if (!char.IsLetter(plateNumber[0]) ||
        !char.IsLetter(plateNumber[1]) ||
        !char.IsLetter(plateNumber[2]))
    {
        return false;
    }
    if (plateNumber[0] == plateNumber[1] ||
        plateNumber[0] == plateNumber[2] ||
        plateNumber[1] == plateNumber[2])
    {
        return false;
    }
    if (plateNumber[3] != '-')
    {
        return false;
    }
    if (!char.IsDigit(plateNumber[4]) ||
        !char.IsDigit(plateNumber[5]) ||
        !char.IsDigit(plateNumber[6]))
    {
        return false;
    }

    return true;
    }
```

Java Solution

```
public static boolean validLicencePlate(String
plateNumber)
{

if (plateNumber.length() != 7)
{
        return false;
}


)
      if (!Character.isLetter(plateNumber.charAt(0)) ||
         !Character.isLetter(plateNumber.charAt(1)) ||
         !Character.isLetter(plateNumber.charAt(2))) {
         return false;
      }

      if (plateNumber.charAt(0) == plateNumber.
charAt(1) ||
         plateNumber.charAt(0) == plateNumber.
charAt(2) ||
         plateNumber.charAt(1) == plateNumber.
charAt(2)) {
         return false;
      }

      if (plateNumber.charAt(3) != '-') {
         return false;
      }

      if (!Character.isDigit(plateNumber.charAt(4)) ||
         !Character.isDigit(plateNumber.charAt(5)) ||
         !Character.isDigit(plateNumber.charAt(6))) {
         return false;
      }
      return true;
   }
```

		AVAILABLE MARKS
Return type bool/Boolean	[1]	
String parameter passed	[1]	
Check if the length is 7 only	[1]	
Check that first three characters are letters	[1]	
Check that first three characters are not the same	[1]	
Check if the fourth character is a hyphen	[1]	
Check if the last three characters are numbers	[1]	
Correct return	[1]	
Use of checking method, ie isDigit, isLetter etc.	[1]	9

Allow alternative valid answers

(a) (ii) Sample solutions

AVAILABLE MARKS

C# Solution:	**Java Solution:**
string[] AllPlates = new string[1000];	String[] AllPlates = new String[1000];

Array type	[1]	
Array name	[1]	
Use of new	[1]	
Setting size to 1000	[1]	4

(b) Sample solutions

C# Solution:

```
   public void updateLicencePlate()
{
       for (int i = 0; i < AllPlates.Length; i++)
{
               string oldPlate = AllPlates[i];
               string updatedPlate = oldPlate.Replace('-', ' ');
               AllPlates [i] = updatedPlate;
               Console.WriteLine("Updated Licence Plate:
old licence plate:  " + oldPlate + ". New licence plate: " +
updatedPlate);
       }
   }
```

Java Solution:

```
  public void updateLicencePlate() {
    for (int i = 0; i < AllPlates.length; i++) {
       String oldPlate = AllPlates[i];
       String updatedPlate = oldPlate.replace('-', ' ');
       AllPlates[i] = updatedPlate;
       System.out.println("Updated Licence Plate: old
licence plate: " + oldPlate + ". New licence plate: " +
updatedPlate);
    }
}
```

Loop of array	[1]	
Update plate	[1]	
Correct output of current plate	[1]	
Correct output of new plate	[1]	4

Allow alternative valid answers

			AVAILABLE MARKS
3	(a)	• Fields marked as private can only be accessed/modified within the class where they are declared • They are not accessible from outside the class in which they are declared • Private is an access modifier. Access modifiers control the visibility and accessibility of class fields • Declaring fields as private enforces encapsulation. This allows the class to control how its data is accessed and modified, typically through public getter and setter properties/ methods [1] mark for any key point × 2	2
	(b) (i)	Sample solutions	

C# Solution:	**Java Solution:**

public Book(string title, string author, string genre, int year, string isbn)
{
Title = title;
Author = author;
Genre = genre;
Year = year;
ISBN = isbn;
 	
}
	public Book(string title, string author, string genre, int year, string isbn)
	{
	setTitle(title);
	setAuthor(author);
	setGenre(genre);
	
	setYear(year);
	setISBN(isbn);
	}

Correct constructor name	[1]	
All fields passed	[1]	
Correct assignment of passed fields	[1]	3

(ii) Sample solutions

C# Solution:	**Java Solution:**

public string Title
{
get { return title; }
set{ title = value;}
}
	public string getTitle() {
	return title;
	}
	
	public void setTitle(string title) {
	this.title = title;
	}

Correct header	[1]	
Correct use of get	[1]	
Correct use of set	[1]	3

Allow alternative valid answers

(c) (i)

AVAILABLE MARKS

C# Solution	Java Solution

public static void authorSearch (string author)
{
bool found = false;
foreach (Book book in BookCollection)
{
if (book.author == author))
{
Console.WriteLine("Title: " + book.Title + ", Year: " + book.year +"ISBN: " + book.isbn);
 	
found = true;
}
}
if (!found)
{
Console.WriteLine("No books found for author: " + author);
} }
	public static void authorSearch (String author)
	{
	boolean found = false;
	for (Book book : BookCollection) {
	if (book.getAuthor() == author){
	System.out.println("("Title: " + book.Title + ", Year: " + book.year +"ISBN: " + book.isbn);
	}
	if (!found) {
	System.out.println("No books found for author: " + author);
	}
	}

Return type void	[1]	
Correct use of array	[1]	
Correct loop	[1]	
Check for author	[1]	
Book details outputted	[1]	
If not found, Suitable message outputted	[1]	6

Allow alternative valid answers

(ii) C# Solution:

AVAILABLE MARKS

```
public void genreCount()
{
    int fictionCount = 0;
    int nonFictionCount = 0;
    int totalBooks = 0;

    for (int i = 0; i < BookCollection.Length; i++)
    {
        if (BookCollection[i] != null)
        {
            totalBooks++;
            if (BookCollection[i].genre=="fiction")
            {
                fictionCount++;
            }
            else
            {
                nonFictionCount++;
            }
        }
    }

    Console.WriteLine("Total number of books: "
+ totalBooks);
    Console.WriteLine("Total number of fiction
books: " + fictionCount);
    Console.WriteLine("Total number of non-
fiction books: " + nonFictionCount);
}
```

Return type void	[1]	
Declare variables to hold counts (Min one needed)	[1]	
Check for null in array	[1]	
Check for genre ("fiction"/"non-fiction")	[1]	
Any running total	[1]	
Output of all three counts	[1]	6

4 When developing software, error handling techniques should be applied.

AVAILABLE MARKS

(a) Complete the following statements by inserting the appropriate words or phrases from the list given below.

ArithmeticException	errors	FormatException	robust
catch	Exception	functionality	runtime
enclose	exceptions	handling	throw
EndOfStreamException	fail	invalid	try
end	finally	IOException	unexpected

Effective exception handling is essential for **robust** software development, helping to maintain application stability. In programming **try** ___ blocks surround code where exceptions may occur, while **catch** blocks handle specific types of exceptions. The **finally** block, executes code after exception handling and is always executed.

Common exceptions in object-oriented programming include **Exception**, which handles all exceptional situations. **ArithmeticException** handles errors related to arithmetic operations, while **FormatException** handles errors related to data type conversion.

In addition to handling exceptions, developers can also explicitly **throw** exceptions to signal errors within the code. This process allows developers to indicate when **unexpected** or **invalid** conditions occur during program execution.

1 mark for every correct answer. [10] marks total. 10

(b)

C# Sample Solution	Java Sample Solution

public double WithdrawFunds(double currentBalance, double amount)
{
try
{
if (amount > currentBalance)
{
throw new InsufficientFu ndsException("Insufficient funds: cannot withdraw more than available balance");
}
else if (amount < 1)
{
throw new Exception("Invalid withdrawal amount: amount cannot be less than £1");
}
else
{
currentBalance -= amount;
return currentBalance;
}
}
catch (InsufficientFundsException ex)
{
Console.WriteLine(ex. Message);
// Handle the exception without rethrowing
return currentBalance; // or another appropriate value
}
catch (Exception ex)
{
Console.WriteLine(ex. Message);
return currentBalance;
}
}
	public double withdrawFunds(double currentBalance, double amount)
	{
	try {
	
	if (amount > currentBalance)
	{
	throw new InsufficientFundsExceptio n("Insufficient funds: cannot withdraw more than available balance");
	}
	else if (amount < 1)
	{
	throw new Exception("Invalid withdrawal amount: amount cannot be less than £1");
	}
	else
	{
	currentBalance -= amount;
	return currentBalance;
	}
	}
	catch (InsufficientFundsException ex)
	{
	System.out.println(ex.Message);
	return currentBalance;
	}
	}
	
	catch (Exception ex)
	{
	System.out.println (ex.Message);
	return currentBalance;
	}
	}

In the context of try/catch

		AVAILABLE MARKS
Correct Structure using try/catch	[1]	
Check if amount is greater than currentBalance	[1]	
Throw new exception for Insufficient funds	[1]	
Throw new exception for invalid amount	[1]	
InsufficientFundsException class used	[1]	
currentBalance updated	[1]	
Correct return	[1]	7

Allow alternative valid answers

5 Candidates' answer should cover the main points below but are not expected to include everything.

AVAILABLE MARKS

(a) Explain the concept and purpose of object serialisation.

Concept [1]
- Transforming an object's state into a format by converting it into a stream of bytes.

Purpose [2]
- To store objects to be used later/perm storage
- To store objects to be used in other programs
- Commonly used to store an objects to a file

3

(b) Explain how object serialisation implemented.

How it is implemented [2]
Any **two** points explained from the list below
- Mark the Class as Serializable:
  - Use the [Serializable] attribute to mark the class you want to serialise. *C#*
  - Implements Serializable. *Java*

- Use a Formatter:
  - Use a formatter like BinaryFormatter to serialise and deserialise the object. *C#*
  - Serialise the Object: Use ObjectOutputStream to write the object to a file or other output stream. *Java*
  - Deserialise the Object: Use ObjectInputStream to read the object from a file or other input stream. *Java*
- Use of IO library
  A file is created

2

6 (a) (i) Sample solution

C# Solution

```
public class Lunch : SpaBooking
{
    private string lunchType;
    private string dietaryRestrictions;
    private string additionalNotes;

    public Lunch(string lunchType, string
dietaryRestrictions, string additionalNotes, int
bookingID, int guestID, DateTime checkinDate,
char chosenPackage)
        : base(bookingID, guestID, checkinDate,
chosenPackage)
    {
        LunchType = lunchType;
        DietaryRestrictions = dietaryRestrictions;
        AdditionalNotes = additionalNotes;
    }
    public string LunchType
    {
        get { return lunchType; }
        set
        {
            if (value == "Standard" || value ==
"Luxury")
            {
                lunchType = value;
            }
            else
            {
                throw new Exception("Lunch type
must be 'Standard' or 'Luxury'.");
            }
        }
    }
```

Java Solution

```
public class Lunch extends SpaBooking {
    private String lunchType;
    private String dietaryRestrictions;
    private String additionalNotes;

    public Lunch(String lunchType, String
dietaryRestrictions, String additionalNotes,
            int bookingID, int guestID, Date
checkinDate, char chosenPackage) {
        super(bookingID, guestID, checkinDate,
chosenPackage);
        setLunchType(lunchType);
        setDietaryRestrictions(dietaryRestrictions);
        setAdditionalNotes(additionalNotes);
    }

public String getLunchType() {
        return lunchType;
    }

    public void setLunchType(String lunchType) {
        if ("Standard".equals(lunchType) || "Luxury".
equals(lunchType)) {
            this.lunchType = lunchType;
        } else {
            throw new
IllegalArgumentException("Lunch type must be
'Standard' or 'Luxury'.");
        }
    }
```

		AVAILABLE MARKS
Correct header	[1]	
Field definitions (for Lunch only)	[1]	
Fields passed for both Lunch and SpaBooking	[1]	
Correct use of :base	[1]	
Passing of SpaBooking fields	[1]	
Assignment of Lunch fields (only)	[1]	
Correct property/method heading	[1]	
Correct get	[1]	
Checking value is correct in set	[1]	
Setting correct value	[1]	10

Allow alternative valid answers

(ii)

C# Solution	Java Solution

private double
calculateLunchCost()
{
if (lunchType == "Standard")
{
return 16;
}
else
{
return 25;
}
}
	private double
	calculateLunchCost()
	{
	if ("Standard" == lunchType)
	{
	return 16;
	}
	else
	{
	return 25;
	}
	}

Correct header [1]
No parameters passed [1]
Correct check of lunchType [1]
Correct Return [1] 4

Allow alternative valid answers

(b) (i)

C# Solution	Java Solution

public virtual double totalPrice()
{
double totalPrice = 0;
switch (chosenPackage)
{
case 'A':
totalPrice += 150;
break;
case 'B':
totalPrice += 175;
break;
case 'C':
totalPrice += 250;
break;
case 'D':
totalPrice += 300;
break;
default: throw new
Exception("Invalid package
chosen.");
}
return totalPrice;
}
	public double totalPrice() {
	double totalPrice = 0;
	switch (chosenPackage) {
	case 'A':
	totalPrice += 150;
	break;
	case 'B':
	totalPrice += 175;
	break;
	case 'C':
	totalPrice += 250;
	break;
	case 'D':
	totalPrice += 300;
	break;
	default:
	throw new IllegalArgument
	Exception("Invalid package
	chosen.");
	}
	return totalPrice;
	}

Correct return type [1]
Correct use of Virtual [1]
Check of package (chosen package) [1]
Correct return [1] 4
Allow alternative valid answers

AVAILABLE MARKS

(ii)

C# Solution	Java Solution

public override double totalPrice()
{
double totalPrice= 85+base.
totalPrice();
 	
if (dinnerIncluded)
{
totalPrice += 65;
}
return totalPrice;
}
	@Override
	public double totalPrice() {
	double totalPrice = 85 + super.
	totalPrice();
	if (dinnerIncluded)
	{
	totalPrice += 65;
	}
	return totalPrice;
	}

AVAILABLE MARKS

Correct return type	[1]	
Correct use of override	[1]	
No parameters passed	[1]	
Check if dinner is included	[1]	
Call of base method	[1]	
Correct calculation	[1]	
Correct return	[1]	7

Allow alternative valid answers

7

C# Solution

```
public void increasedPrice()
{
    int standardCount = 0;
    int luxuryCount = 0;
    double totalCurrentRevenue;
    double totalIncreasedRevenue;

    foreach (Booking booking in PromotionalEvent)
    {
        if (booking.GetType() == typeof(Lunch))
        {
            if (booking.LunchType == "Standard")
            {
                standardCount++;
            }
            else
            {
                luxuryCount++;
            }
        }
    }
    totalCurrentRevenue = (standardCount * 16) +
(luxuryCount * 25);
    totalIncreasedRevenue = (standardCount * 20)
+ (luxuryCount * 30);
    Console.WriteLine("Total revenue from current
lunches: " + totalCurrentRevenue.ToString("C"));
    Console.WriteLine("Total revenue if lunch
prices are increased: " + totalIncreasedRevenue.
ToString("C"));
}
```

Java Solution

```
public void increasedPrice() {
    int standardCount = 0;
    int luxuryCount = 0;
    double totalCurrentRevenue;
    double totalIncreasedRevenue;

    for (Booking booking : PromotionalEvent) {
        if (booking instanceof Lunch) {
            Lunch lunchBooking = (Lunch)
booking;
            if (lunchBooking.getLunchType()==
"Standard") {
                standardCount++;
            } else {
                luxuryCount++;
            }
        }
    }

    totalCurrentRevenue = (standardCount *
16) + (luxuryCount * 25);
    totalIncreasedRevenue = (standardCount *
20) + (luxuryCount * 30);

    System.out.println("Total revenue from
current lunches: £" + String.format("%.2f",
totalCurrentRevenue));
    System.out.println("Total revenue if
lunch prices are increased: £" + String.
format("%.2f", totalIncreasedRevenue));
}
```

		AVAILABLE MARKS
Correct return type	[1]	
Variables declared	[1]	
Loop of array	[1]	
Check object type	[1]	
Casting of object	[1]	
Increase count based on type of lunch	[1]	
Calculation for current revenue	[1]	
Calculation for increased revenue	[1]	
Correct output	[1]	
Sting formatting use: Interpolated, Concatenation	[1]	10
Allow alternative valid answers		
	Total	**100**