UNIVERSITY OF RUHUNA

Faculty of Engineering

End-Semester 2 Examination in Engineering: March 2026

Module Number: EE2202 **Module Name: Object Oriented Programming**
C-23 (Proper)

Part-B

[Part-A + Part-B = 2 hours]

Instructions for candidates

- Answer all questions
- Write the index number very clearly in each answer sheet.
- For Q1 answer in the seperate sheet Provided.
- For Q2 and Q3 answer in the booklet provided.

Q1. Complete the following C# programs by filling in the blanks. Choose the correct term for each blank from the word list given for the correspoding question. Each correct blank is 0.5 marks. Note that there are more options than blanks.)

a) The code defines a class with a parameterized constructor, a restricted property, and a method, and then creates an object in the `Main` method.

Brand	brand	class	create	Device
Device()	GetDeviceInfo	GetDeviceInfo()	new	private
protected	public	string	void	

```
public class Device{
    public string Brand { get; ..... (1) ..... set; }

    public .....(2).....(string brand){
        Brand = .....(3).....;
    }
    public .....(4)..... GetDeviceInfo(){
        return $"Device Brand: {Brand}";
    }
}

public class Program{
```

```
    public static void Main(string[] args){
        Device myDevice = ..... Device("Tablet");
        string currentBrand = myDevice.....;
        string info = myDevice. .....();

        Console.WriteLine(info);
    }
}
```

b) The following code demonstrates inheritance, polymorphic object creation, and method overriding.

base	Car	extends	new	overload
override	private	protected	public	StartEngine
StartEngine()	static	super	Vehicle	virtual

```
public class Vehicle
{
    // A field accessible only within this class and derived
   classes
    ..... int topSpeed;

    // Method that allows derived classes to provide their own
   implementation
    public ..... void StartEngine()
    {
        Console.WriteLine("Vehicle engine starting...");
    }
}

// Derived class inheriting from the base class

public class Car : .....
{
    // Modifying the inherited method
    public ..... void StartEngine()
    {
        // Calling the original method from the parent class
        ..... .StartEngine();
        topSpeed = 120;
        Console.WriteLine("Car is ready to drive.");
    }
}

public class Program
{
    public static void Main(string[] args)
```

```
    {
        // Polymorphic object creation
        Vehicle myCar = ..... Car();

        // Executing the method
        myCar. ..... ();
    }
}
```

c) The following code demonstrates different ways to pass parameters to methods and return data.

in	internal	int	out	private
public	ref	rem	return	val
value	void			

```
// Class accessible only within the same assembly
..... class Calculator
{
    // Method that modifies the original variable passed to it
    public void DoubleValue( ..... int number)
    {
        number = number * 2;
    }

    // Method returning a calculated value and extracting a
   secondary value
    public ..... GetDivisionStats(int a, int b, ..... int
   remainder)
    {
        remainder = a % b;
        ..... a / b;
    }
}

public class Program{
    public static void Main(string[] args)
    {
        Calculator calc = new Calculator();

        int val = 5;
        // Passing the variable so it can be modified by the method
        calc.DoubleValue( ..... val);
```

```
        int rem;
        // Passing the variable to extract the secondary return
    value

        int quotient = calc.GetDivisionStats(10, 3, ..... rem);

        // This should print 10 to the console

        Console.WriteLine(.....);
    }
}
```

(21) (22)

d) The following code demonstrates a C# Windows Forms application that handles a button click event to draw and fill shapes on the form.

Brush	Content	CreateGraphics	DrawBox
DrawRectangle	Event	EventArgs	FillBox
FillRectangle	Form	GetGraphics	Graphic
Graphics	Paint	Pen	Text
Value	Window		

```
// The main window class inheriting from the base Windows Forms
   class

public partial class MainForm : .....
{
    public MainForm()
    {
        InitializeComponent();
    }

    // Event handler for a button click

    private void btnDraw_Click(object sender, ..... e)
    {
        // Obtain the object required to draw on the form
        ..... g = this.....();

        // Create tools for drawing outlines and filling shapes
        ..... myPen = new Pen(Color.Black, 2);
        SolidBrush myBrush = new SolidBrush(Color.Blue);

        // Draw the outline of a rectangle
        g.....(myPen, 10, 10, 100, 50);

        // Draw a solid rectangle filled with color
        g.....(myBrush, 10, 70, 100, 50);
```

(23) (24) (25) (26) (27) (28) (29)

```
            // Update the label on the button
            btnDraw.Text = "Shapes Drawn!";
        }
    }
```

Q2. Design a C# class named `EVBattery` to manage an electric vehicle's charge state. Follow the requirements below to demonstrate encapsulation and basic class structure:

a) Write the class definition for `EVBattery` with the following requirements:

- Create a private integer field named `_currentLevel`.
- Create a public read-only integer property named `MaxCapacity`.
- Implement a public constructor that takes one integer parameter to initialize the `MaxCapacity`.

[1.5 marks]

b) Implement a property to securely encapsulate the battery's charge level with the following logic:

- Create a public integer property named `CurrentLevel` that wraps the `_currentLevel` field.
- In its `set` accessor, write logic ensuring the assigned value cannot drop below 0 or exceed the `MaxCapacity`.

[2 marks]

c) Write a method to simulate charging the battery based on the following rules:

- Create a public `void` method named `Charge` that accepts an integer parameter for `minutes`.
- Inside the method, increase the `CurrentLevel` property by 8 units for every minute charged.

[2 marks]

d) Write a C# `Main` method to demonstrate the usage of your class by performing the following actions in order:

- Instantiate an `EVBattery` object with a capacity of 100.
- Intentionally assign 150 directly to its `CurrentLevel` property.
- Call the `Charge` method for 5 minutes.
- Print the final `CurrentLevel` to the console.

[2 marks]

Q3. In automated manufacturing, various sensors collect data, but each type processes information differently. Object-oriented polymorphism allows a central controller system to interact with all sensors uniformly. Design a C# class hierarchy demonstrating these concepts:

a) Write an `abstract` class named `Sensor` with the following requirements:

- Create a `protected` string field named `_location`.
- Implement a public constructor that takes a string parameter to initialize the `_location`.
- Declare a public `abstract` method named `ReadData` that returns a `double`.

[1.5 marks]

b) Write a class named `PressureSensor` that inherits from `Sensor` with the following requirements:

- Create a private `double` field named `_basePressure`.
- Implement a public constructor taking a string for the location and a double for the base pressure.
- Pass the location parameter to the base class constructor using the appropriate keyword, and initialize the `_basePressure` field.

[2 marks]

c) Implement the specific data reading logic for the pressure sensor:

- `override` the `ReadData` method inherited from the base class.
- Inside the method, simulate a reading by returning the `_basePressure` added to a fixed simulated fluctuation value of `2.5`.

[2 marks]

d) Write a C# `Main` method to demonstrate late binding using a collection:

- Create a `List<Sensor>` to hold multiple sensor objects.
- Instantiate two `PressureSensor` objects with different locations and base pressures, and add them to the list.
- Use a `foreach` loop to iterate through the list.
- Inside the loop, call `ReadData()` on the base `Sensor` reference and print the result to the console.

[2 marks]