



Daffodil International University
Faculty of Science & Information Technology
Department of Software Engineering
Midterm Examination, Spring 2026

Course Code: SE216; Course Title: Object-Oriented Programming.
Sections & Teachers: A-B (UKD), C-D (QFF), E-G (KM), H-J (SHN),
K-L (RJM), M-N (TM), O (AZ)

Time: 1 Hour 30 Mins

Marks: 25

Answer ALL Questions

[The figures in the right margin indicate the full marks and corresponding course outcomes. All portions of each question must be answered sequentially.]

1	a <i>Driver code:</i> <pre> public static void main(String[] args) { Brushstroke unknown = new Brushstroke(); Brushstroke veil = new Brushstroke("Lumière Veil", "Concealing"); Brushstroke seal = new Brushstroke("Gommage Seal"); unknown.printStroke(); veil.printStroke(); seal.printStroke(); unknown.updateInfo("Crimson Slash"); veil.updateInfo(5); seal.updateInfo("Monolith Bind", "Sealing"); unknown.printStroke(); seal.printStroke(); } </pre> <i>Expected output:</i> Unknown stroke is Mysterious. Lumière Veil stroke is Concealing. Gommage Seal stroke is Erasing. Lumière Veil stroke consumes 5 pigments. Crimson Slash stroke is Mysterious. Monolith Bind stroke is Sealing. Infer the minimum required constructors, fields, and methods for the <i>Brushstroke</i> class from the driver code and expected output, then implement the class so that running the program produces the exact output shown.	[Marks -7]	CLO-1 Level-2
b	Compute the output of the following code: <pre> class Referee { void whistle(int foul) { System.out.println("Ref foul " + foul); } void whistle(double t) { System.out.println("Ref added time: " + t); } } class VAR extends Referee { </pre>	[Marks -3]	

		<pre> @Override void whistle(int foul) { System.out.println("VAR foul " + (foul + 1)); } class Linesman extends Referee { @Override void whistle(int offside) { System.out.println("Offside player: " + offside); } } public class Main { public static void main(String[] args) { Referee r1 = new VAR(); Referee r2 = new Linesman(); r1.whistle(2); r2.whistle(3); r1.whistle(1.5); } } </pre>		
2	a	A university uses a Campus Shuttle App. Each user has a Profile that can be updated anytime. A profile has a <u>username</u>, a <u>phone number</u>, a <u>pickup point</u> and a <u>drop-off point</u>. • Username = "ABC" and you can only access but cannot modify it. • Phone number, pickup and drop-off points can be accessed and updated later, but the drop-off point cannot be the same as the pickup point. Translate the above scenario into a java class named <u>Profile</u>. Structure the class using the concept of Encapsulation and access modifiers. [Note: You have to write full Java code. No need to write main() method]	[Marks -5]	CLO-2 Level-2
	b	A drawing tool supports two drawable shapes: <u>Ring</u> and <u>Rectangle</u>. Shape: Every shape has attributes: <u>label</u> (string) and <u>border</u> (double), and supports a method <u>double inkNeeded()</u> which estimates how much ink is needed to draw the border. The default amount of ink needed is border * 1. Ring: • A ring has <u>outerRadius</u> (double) and <u>innerRadius</u> (double). • Its <u>inkNeeded()</u> returns: $2 * \pi * (\text{outerRadius} + \text{innerRadius}) * \text{borderWidth}$ [assume $\pi = 3.1416$]. • Sometimes the tool draws multiple identical rings, so it supports double inkNeeded(int copies) which returns copies * inkNeeded(). Rectangle: • A rectangle has <u>length</u> (double) and <u>width</u> (double). • Its <u>inkNeeded()</u> returns perimeter * border, where perimeter is $2 * (\text{length} + \text{width})$. Translate the above scenario into Java superclass Shape and subclasses Ring and Rectangle using inheritance, and method overriding and method overloading. Use appropriate constructors, and also identify where method overloading and method overriding are used. [Note: You have to write full Java code. No need to write main() method]	[Marks -8]	
	c	Differentiate between instance variables and static variables.	[Marks -2]	