



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

Course Code: SE-216 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: 2:00 Hrs

Marks: 40

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)	Illustrate the “is a” and “has a” relationships in OOP by constructing a short code example for each.	[Marks-3]	CLO-3 Level-3
	b)	You are managing the inventory for a warehouse that tracks boxes of fruit. You need to handle incoming shipments and damaged goods. Program a main() method that performs the following operations: 1. Create an ArrayList called fruitStock and insert: Banana, Orange, Lychee, Mango, Apple. 2. A new shipment arrives. From index 2 onwards, insert: Pear, Grape, Date. 3. The warehouse manager realizes the Apple box is expired. Replace “Apple” with “Rotten Apple”. 4. The first three fruits from fruitStock list are loaded onto a delivery truck. Extract these three fruits, <u>print them</u> and <u>then remove</u> them from fruitStock. 5. Check if the list contains “Mango”. If it does, print its index; otherwise, print “Out of Stock”. 6. Display the first and last item in the fruitStock. 7. Display the size of the fruitStock and then clear the fruitStock.	[Marks-7]	
	c)	A warehouse stores product names in a <u>string array</u> - <u>products</u> and their stock quantities in a <u>parallel integer array</u> - <u>quantities</u>. Build a Java program that attempts to look up a product at <u>index</u> and <u>check</u> its stock. Use <u>nested try-catch</u> blocks to handle the following cases: 1. If the index is out of bounds, catch <code>ArrayIndexOutOfBoundsException</code> and display: • “Error: Product slot does not exist.” • If the index is valid, print “Product found: <name>.” 2. Create a custom exception named <code>OutOfStockException</code>. Throw it if the quantity at that index is 0, displaying: • “Out of Stock: <name> is currently unavailable.” • If quantity is greater than 0, print: “In stock. Units available: <quantity>.” 3. Always print “=== Inventory check complete ===” at the end. All exceptions must be handled using a single pair of nested try-catch blocks – outer one for index out of bounds and inner one for product out of stock. Use the following hardcoded data –	[Marks-7]	

		<pre>int index = 2; String[] products = {"Keyboard", "Monifor", "Mouse", "Headset"}; int[] quantities = {15, 0, 8, 3};</pre>		
	d)	Program a Java package named "test" that contains a class Balance with a method show() to print a message. In a separate package "main", write another class Main that imports the "my" package, creates an object of Balance, and calls the show() method.	[Marks-3]	
2.	a)	Draw a UML class diagram for a hospital management system that manages doctors and patients, based on the following requirements: - The system contains an abstract class Person with the attributes name (string) and age (integer), and an abstract method void displayInfo() - There are two subclasses: Doctor and Patient, both derived from Person. - The Doctor class contains - an additional attribute specialization (string) and - maintains a list of assigned patients using an array Patient[]. - It also includes the methods void assignPatient(p: Patient) and int getPatientCount(). - The Patient class contains the additional attributes patientId (string) and diagnosis (string). Your UML class diagram should clearly show classes, attributes, methods, constructors, access modifiers, inheritance and association relationship.	[Marks-8]	CLO-4 Level-3
	b)	<pre> classDiagram class Transactionable { <<interface>> +deposit(amount: double): void +withdraw(amount: double): void } class ServiceChargeable { <<interface>> +deductCharge(): void } class Account { <<abstract>> -accNo: String -balance: double +Account(accNo: String, balance: double) +deposit(amount: double): void +withdraw(amount: double): void +{abstract} displayAccountType(): void } class CurrentAccount { -svgCharge: double +CurrentAccount(accNo: String, balance: double, svgCharge: double) +displayAccountType(): void } class SalaryAccount { -employerName: String +SalaryAccount(accNo: String, balance: double, employerName: String) +displayAccountType(): void } class Customer { -id: String -name: String +Customer(id: String, name: String) +getName(): String } Transactionable < .. Account ServiceChargeable < .. Account Account < -- CurrentAccount Account < -- SalaryAccount Customer "1" -- "1..*" Account </pre> Translate the following UML class diagram of a Bank Account System into Java code, implementing all classes, the interface, attributes, constructors, methods, and relationships shown.	[Marks-12]	