

Institute of Information Technology (IIT)

Noakhali Science and Technology University

Bachelor of Science in Software Engineering

Lab Class Test -2025 (Year-1, Term-2)

Course Title: Object Oriented Concepts I Lab

Total Time: 1 Hour

Course Code: CSE 12141

Full Marks: 25

1	<p>Create a class Employee with the following private fields:</p> <ul style="list-style-type: none">• int id• String name• double baseSalary <p>1) Constructors Implement two constructors:</p> <p>Employee(int id, String name)</p> <ul style="list-style-type: none">◦ Initializes baseSalary to a sensible default (e.g., 0.0). <p>Employee(int id, String name, double baseSalary)</p> <ul style="list-style-type: none">◦ Must validate salary (see Encapsulation rules). <p>2) Encapsulation & Validation</p> <ul style="list-style-type: none">• Provide getters for all fields.• Provide a setter for baseSalary:<ul style="list-style-type: none">◦ Reject negative values◦ Accept zero and positive values. <p>Note: Do not expose public setters for id and name (read-only after construction).</p> <p>3) Method Overloading — pay Overload a pay method with these exact signatures:</p> <ol style="list-style-type: none">1. double pay()<ul style="list-style-type: none">◦ Returns the baseSalary.2. double pay(double bonus)<ul style="list-style-type: none">◦ Returns baseSalary + bonus. (No side effects; do not mutate fields.)3. double pay(double bonus, double taxRate)<ul style="list-style-type: none">◦ Returns the net amount after applying tax to (baseSalary + bonus).◦ Assume taxRate is in [0.0, 1.0] (e.g., 0.25 = 25% tax).◦ Validate taxRate range; throw IllegalArgumentException if invalid.	10
---	--	----

2.	<p>Class Specification</p> <p>Create a class <code>CartItem</code> with the following private fields:</p> <ul style="list-style-type: none"> • <code>String productId</code> • <code>String name</code> • <code>double unitPrice</code> • <code>int quantity</code> <p>1) Constructors</p> <p>Implement two constructors:</p> <ol style="list-style-type: none"> 1. Full-args: <code>CartItem(String productId, String name, double unitPrice, int quantity)</code> <ul style="list-style-type: none"> ○ Must validate all inputs (see Encapsulation rules). 2. Overloaded (defaults quantity to 1): <code>CartItem(String productId, String name, double unitPrice)</code> <ul style="list-style-type: none"> ○ Sets <code>quantity = 1</code> after validating <code>productId</code>, <code>name</code>, and <code>unitPrice</code>. <p>2) Encapsulation & Validation</p> <ul style="list-style-type: none"> • Provide getters for all fields. • Provide setters only for <code>unitPrice</code> and <code>quantity</code>, with validation: <ul style="list-style-type: none"> ○ <code>unitPrice >= 0</code> (reject negative; throw <code>IllegalArgumentException</code> with a clear message). ○ <code>quantity >= 1</code> (reject 0 or negative in the standard setter). • <code>productId</code> and <code>name</code> are immutable after construction (no public setters). • Validate non-empty <code>productId</code> and <code>name</code> in constructors (reject null/blank). <p>3) Method Overloading</p> <p>Implement the following overloaded methods:</p> <p>a) Quantity Updates</p>	15

	<ol style="list-style-type: none"> 1. void updateQuantity(int quantity) <ul style="list-style-type: none"> ○ Sets quantity to the given value; must enforce quantity >= 1. ○ If invalid, throw IllegalArgumentException. 2. void updateQuantity(int quantity, boolean allowZero) <ul style="list-style-type: none"> ○ If allowZero == true, permit quantity == 0 to indicate “remove item” (you may set internal quantity to 0, or alternatively keep a boolean removed flag—pick one and document it). ○ If allowZero == false, behave like the single-arg version (require quantity >= 1). <p>Note: If you choose to allow quantity == 0, ensure other methods (e.g., total()) behave consistently (e.g., return 0.0 when quantity is 0).</p> <p>b) Totals</p> <ol style="list-style-type: none"> 1. double total() <ul style="list-style-type: none"> ○ Returns unitPrice * quantity. 2. double total(double discountPercent) <ul style="list-style-type: none"> ○ Applies a percentage discount to the unit price before multiplying by quantity. ○ discountPercent is in [0.0, 100.0]. ○ Validate range; throw IllegalArgumentException if outside range. ○ Compute as: effectiveUnit = unitPrice * (1.0 - discountPercent / 100.0) return effectiveUnit * quantity 	
--	---	--

--	--	--