



## Internal Assessment Test 2 – January 2026

Sub:	Object Oriented Programming with JAVA			Sub Code:	BCS306A	Branch:	AIML/CSE AIML		
Date:	/01/26	Duration:	90 min	Max Marks:	50	Sem/Sec:	III A, B & C		
<b><u>Answer any FIVE FULL Questions</u></b>						<b>MAR KS</b>	<b>CO</b>	<b>RB T</b>	
1a	<p>Explain <b>method overriding</b> and <b>dynamic method dispatch</b> with example.</p> <p><b>1. Method Overriding(3 marks)</b> Method overriding occurs when a <b>subclass provides a specific implementation of a method that is already defined in its superclass</b>.</p> <p><b>Rules for Method Overriding:</b></p> <ul style="list-style-type: none"> <li>Method name and parameter list must be <b>exactly the same</b>.</li> <li>The method must be <b>inherited</b> from the superclass.</li> <li>Access level cannot be more restrictive than the superclass method.</li> <li>Overriding happens only in <b>runtime polymorphism</b>.</li> </ul> <p><b>2. Dynamic Method Dispatch(3 marks)</b> Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at runtime, based on the object being referenced, not the reference type.</p> <p>In Java, this is achieved using a superclass reference pointing to a subclass object.</p>						6	CO3	L2
1b	<p>Explain <b>abstract classes and interfaces</b>. Compare both.</p> <p><b>1. Abstract Class (2 marks)</b> An <b>abstract class</b> is a class that <b>cannot be instantiated</b> and may contain <b>abstract methods (without body)</b> as well as <b>concrete methods (with body)</b>. It is used when classes share <b>common behavior and state</b>.</p> <p><b>Key Features</b></p> <ul style="list-style-type: none"> <li>Declared using the <b>abstract</b> keyword</li> <li>Can have <b>abstract and non-abstract methods</b></li> <li>Can contain <b>instance variables</b></li> <li>Can have <b>constructors</b></li> <li>Supports <b>single inheritance</b> (extends one class)</li> </ul> <p><b>2. Interface (2 marks)</b> An <b>interface</b> is a collection of <b>abstract methods and constants</b> that represents a <b>contract</b>. A class implementing an interface must provide implementations for all its methods.</p> <p><b>Key Features</b></p> <ul style="list-style-type: none"> <li>Declared using the <b>interface</b> keyword</li> <li>All methods are <b>public and abstract by default</b></li> <li>Variables are <b>public, static, and final</b></li> <li>No constructors</li> <li>Supports <b>multiple inheritance</b></li> </ul>						4	CO3	L3
2a	<p>Explain <b>multilevel inheritance</b> with a Java program.</p> <p><b>Multilevel Inheritance in Java(3 marks)</b> Multilevel inheritance is a type of inheritance where a class is derived from another class, which itself is derived from another class. In simple terms:</p> <p><b>Class A → Class B → Class C</b> Here, Class C inherits the properties and methods of <b>both</b> Class A and Class B.</p> <p><b>Example(3 marks)</b></p>						6	CO3	L2
2b	<p>Explain <b>final keyword</b> with respect to class, method, and variable.</p> <p><b>1. final Variable(1)</b> A <b>final variable</b> is treated as a <b>constant</b>. Its value <b>cannot be changed once assigned</b>.</p> <p><b>Key Points</b></p> <ul style="list-style-type: none"> <li>Must be initialized only once</li> <li>Can be initialized at declaration, in constructor, or in initializer block</li> <li>Prevents reassignment</li> </ul> <p><b>2. final Method(1.5)</b> A <b>final method</b> <b>cannot</b> be <b>overridden</b> by <b>subclasses</b>.</p>						4	CO3	L3

	<p>It is used to prevent modification of method behavior in child classes.</p> <p><b>Key Points</b></p> <ul style="list-style-type: none"> <li>• Method implementation remains unchanged in subclasses</li> <li>• Supports method overloading but not overriding</li> </ul> <p><b>3. final Class(1.5)</b></p> <p>A <b>final class</b> <b>cannot</b> <b>be</b> <b>inherited.</b></p> <p>It is used when a class should not be extended for security or design reasons.</p> <p><b>Key Points</b></p> <ul style="list-style-type: none"> <li>• Prevents inheritance</li> <li>• All methods are implicitly final</li> </ul>		
3a	<p>Explain <b>exception handling in Java</b> with suitable examples.</p> <p><b>Exception Handling in Java(3 marks)</b></p> <p><b>Exception handling</b> in Java is a mechanism to <b>handle runtime errors</b> so that the normal flow of the program can be maintained.</p> <p>An <b>exception</b> is an event that occurs during program execution and disrupts normal execution.</p> <p><b>1. Types of Exceptions</b></p> <p><b>a) Checked Exceptions</b></p> <ul style="list-style-type: none"> <li>• Checked at <b>compile time</b></li> <li>• Must be handled using try-catch or throws</li> <li>• Example: IOException, SQLException</li> </ul> <p><b>b) Unchecked Exceptions</b></p> <ul style="list-style-type: none"> <li>• Occur at <b>runtime</b></li> <li>• Subclasses of RuntimeException</li> <li>• Example: ArithmeticException, NullPointerException</li> </ul> <p><b>2. Exception Handling Keywords</b></p> <p><b>Keyword Purpose</b></p> <p>try Contains code that may cause an exception</p> <p>catch Handles the exception</p> <p>finally Executes whether exception occurs or not</p> <p>throw Explicitly throws an exception</p> <p>throws Declares exceptions in method signature</p> <p>Example(2 marks)</p>	5	CO4 L3
3b	<p>Explain <b>user-defined exceptions</b> with a program.</p> <p><b>User-Defined Exceptions in Java(3 marks)</b></p> <p>A <b>user-defined (custom) exception</b> is an exception created by the programmer to handle <b>application-specific error conditions</b>.</p> <p>It is created by <b>extending the Exception class</b> (for checked exceptions) or <b>RuntimeException class</b> (for unchecked exceptions).</p> <p><b>Steps to Create a User-Defined Exception</b></p> <ol style="list-style-type: none"> <li>1. Create a class that extends Exception or RuntimeException</li> <li>2. Provide a constructor to pass the error message</li> <li>3. Use throw to explicitly throw the exception</li> <li>4. Handle it using try-catch</li> </ol> <p>Example (2 marks)</p>	5	CO4 L3
4a	<p>Explain <b>throw and throws</b> keywords with examples.</p> <p><b>1. throw Keyword(2.5 marks)</b></p> <p>The <b>throw keyword</b> is used to <b>explicitly throw a single exception</b> from a method or block of code.</p> <p><b>Key Points</b></p> <ul style="list-style-type: none"> <li>• Used inside a method or block</li> <li>• Can throw <b>one exception at a time</b></li> <li>• Used with <b>exception objects</b></li> <li>• Transfers control to the nearest catch block</li> </ul> <p><b>2. throws Keyword(2.5 marks)</b></p> <p>The <b>throws keyword</b> is used to <b>declare exceptions</b> that a method may pass to the calling method instead of handling them.</p> <p><b>Key Points</b></p> <ul style="list-style-type: none"> <li>• Used in <b>method signature</b></li> <li>• Can declare <b>multiple exceptions</b></li> <li>• Mainly used for <b>checked exceptions</b></li> <li>• Responsibility of handling exception is passed to caller</li> </ul>	5	CO4 L2

4b	<p>Explain <b>chained exceptions</b> with example.</p> <p>A <b>chained exception</b> is an exception that is <b>caused by another exception</b>. Java provides this mechanism to <b>preserve the original cause</b> of an exception while throwing a new one.</p> <p>This helps in <b>better debugging and error tracing</b>.</p>	5	CO4	L2
	<p><b>Why Chained Exceptions Are Needed</b></p> <ul style="list-style-type: none"> <li>• To wrap a low-level exception with a higher-level exception</li> <li>• To maintain the <b>root cause</b> of the error</li> <li>• Common in layered applications (DAO → Service → Controller)</li> </ul>			
	<p><b>How Chaining Works in Java</b></p> <p>Java supports chained exceptions using:</p> <ul style="list-style-type: none"> <li>• <b>Constructor</b> that accepts a cause</li> <li>• <b>initCause()</b> method</li> <li>• <b>getCause()</b> method</li> </ul>			
5a	<p>What is thread? Explain the two ways to creating a thread in java with an example.</p> <p>A <b>thread</b> is the <b>smallest unit of execution</b> within a program. In Java, a thread represents a <b>separate path of execution</b>, allowing multiple tasks to run <b>concurrently</b> within a single program.</p> <p>Java supports <b>multithreading</b>, which improves:</p> <ul style="list-style-type: none"> <li>• Program responsiveness</li> <li>• CPU utilization</li> <li>• Performance in concurrent applications</li> </ul>	5	CO5	L3
	<p><b>Two Ways to Create a Thread in Java</b></p> <p>Java provides <b>two main approaches</b> to create a thread:</p> <p><b>1. By Extending the Thread Class</b></p> <p><b>Steps</b></p> <ol style="list-style-type: none"> <li>1. Create a class that extends Thread</li> <li>2. Override the run() method</li> <li>3. Create an object of the class</li> <li>4. Call the start() method</li> </ol> <p><b>2. By Implementing the Runnable Interface</b></p> <p><b>Steps</b></p> <ol style="list-style-type: none"> <li>1. Create a class that implements Runnable</li> <li>2. Override the run() method</li> <li>3. Pass the object to a Thread class constructor</li> <li>4. Call the start() method</li> </ol>			
5b	<p><b>Explain Thread Synchronization with suitable example.</b></p> <p><b>Thread Synchronization in Java</b></p> <p><b>Thread synchronization</b> is a mechanism used to <b>control access to shared resources</b> in a multithreaded environment. It ensures that <b>only one thread at a time</b> can access a critical section of code, thereby preventing <b>data inconsistency and race conditions</b>.</p>	5	CO5	L3
	<p><b>Why Synchronization Is Needed</b></p> <p>When multiple threads access a <b>shared object</b> simultaneously:</p> <ul style="list-style-type: none"> <li>• Data may become inconsistent</li> <li>• Output may be unpredictable</li> <li>• Race conditions can occur</li> </ul>			
	<p><b>Types of Synchronization in Java</b></p> <ol style="list-style-type: none"> <li>1. <b>Synchronized Method</b></li> <li>2. <b>Synchronized Block</b></li> <li>3. <b>Static Synchronization</b> (class-level lock)</li> </ol>			
6a	<p>Explain <b>Enumerations in Java</b> including: values(), valueOf()</p> <p><b>Enumerations (Enums) in Java</b></p> <p>An <b>Enumeration (enum)</b> in Java is a <b>special data type</b> used to define a <b>fixed set of named constants</b>.</p>	6	CO5	L3

	<p>Enums improve <b>type safety, readability, and maintainability</b> compared to using constants like int or String.</p> <p><b>Defining an Enum</b></p> <pre>enum Day {     MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY }</pre> <ul style="list-style-type: none"> <li>• Day is an enum</li> <li>• Each identifier is a <b>constant object</b> of type Day</li> </ul> <p><b>values() Method</b></p> <p>The <b>values()</b> method returns an <b>array of all enum constants</b> in the order they are declared.</p> <p><b>Example</b></p> <pre>class ValuesExample {     public static void main(String[] args) {         for (Day d : Day.values()) {             System.out.println(d);         }     } }</pre> <p><b>valueOf() Method</b></p> <p>The <b>valueOf()</b> method returns the <b>enum constant</b> corresponding to the specified <b>string name</b>.</p> <p><b>Example</b></p> <pre>class ValueOfExample {     public static void main(String[] args) {         Day d = Day.valueOf("FRIDAY");         System.out.println(d);     } }</pre>			
6b	<p>Explain <b>Wrapper Classes and Autoboxing</b> with examples.</p> <p><b>1. Wrapper Classes</b></p> <p><b>Wrapper classes</b> in Java provide a way to convert <b>primitive data types</b> into <b>objects</b>.</p> <p>This is required because Java collections and many APIs work only with <b>objects</b>, not primitives.</p> <pre>class WrapperDemo {     public static void main(String[] args) {         int a = 10;           // primitive         Integer obj = Integer.valueOf(a); // wrapping          System.out.println(obj);     } }</pre> <p><b>2. Autoboxing</b></p> <p><b>Autoboxing</b> is the <b>automatic conversion</b> of a <b>primitive type</b> into its <b>corresponding wrapper object</b> by the Java compiler.</p> <pre>class AutoboxingDemo {     public static void main(String[] args) {         int a = 20;         Integer obj = a; // autoboxing          System.out.println(obj);     } }</pre>	4	CO5	L3