

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			OBE
<u>Answer any FIVE FULL Questions</u>								MAR KS	CO	RB T
1a	<p>Explain method overriding and dynamic method dispatch with example.</p> <p>1. Method Overriding(3 marks) Method overriding occurs when a subclass provides a specific implementation of a method that is already defined in its superclass. Rules for Method Overriding:</p> <ul style="list-style-type: none"> • Method name and parameter list must be exactly the same. • The method must be inherited from the superclass. • Access level cannot be more restrictive than the superclass method. • Overriding happens only in runtime polymorphism. <p>2. Dynamic Method Dispatch(3 marks) 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 abstract classes and interfaces. Compare both.</p> <p>1. Abstract Class (2 marks) An abstract class is a class that cannot be instantiated and may contain abstract methods (without body) as well as concrete methods (with body). It is used when classes share common behavior and state. Key Features</p> <ul style="list-style-type: none"> • Declared using the abstract keyword • Can have abstract and non-abstract methods • Can contain instance variables • Can have constructors • Supports single inheritance (extends one class) <p>2. Interface (2 marks) An interface is a collection of abstract methods and constants that represents a contract. A class implementing an interface must provide implementations for all its methods. Key Features</p> <ul style="list-style-type: none"> • Declared using the interface keyword • All methods are public and abstract by default • Variables are public, static, and final • No constructors • Supports multiple inheritance 							4	CO3	L3
2a	<p>Explain multilevel inheritance with a Java program.</p> <p>Multilevel Inheritance in Java(3 marks) 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: Class A → Class B → Class C Here, Class C inherits the properties and methods of both Class A and Class B. Example(3 marks)</p>							6	CO3	L2
2b	<p>Explain final keyword with respect to class, method, and variable.</p> <p>1. final Variable(1) A final variable is treated as a constant. Its value cannot be changed once assigned. Key Points</p> <ul style="list-style-type: none"> • Must be initialized only once • Can be initialized at declaration, in constructor, or in initializer block • Prevents reassignment <p>2. final Method(1.5) A final method cannot be overridden by subclasses.</p>							4	CO3	L3

	<p>It is used to prevent modification of method behavior in child classes.</p> <p>Key Points</p> <ul style="list-style-type: none">• Method implementation remains unchanged in subclasses• Supports method overloading but not overriding <p>3. final Class(1.5)</p> <p>A final class cannot be inherited.</p> <p>It is used when a class should not be extended for security or design reasons.</p> <p>Key Points</p> <ul style="list-style-type: none">• Prevents inheritance• All methods are implicitly final												
3a	<p>Explain exception handling in Java with suitable examples.</p> <p>Exception Handling in Java(3 marks)</p> <p>Exception handling in Java is a mechanism to handle runtime errors so that the normal flow of the program can be maintained.</p> <p>An exception is an event that occurs during program execution and disrupts normal execution.</p>	5	CO4	L3									
	<p>1. Types of Exceptions</p> <p>a) Checked Exceptions</p> <ul style="list-style-type: none">• Checked at compile time• Must be handled using try-catch or throws• Example: IOException, SQLException <p>b) Unchecked Exceptions</p> <ul style="list-style-type: none">• Occur at runtime• Subclasses of RuntimeException• Example: ArithmeticException, NullPointerException												
	<p>2. Exception Handling Keywords</p> <table><thead><tr><th>Keyword</th><th>Purpose</th></tr></thead><tbody><tr><td>try</td><td>Contains code that may cause an exception</td></tr><tr><td>catch</td><td>Handles the exception</td></tr><tr><td>finally</td><td>Executes whether exception occurs or not</td></tr><tr><td>throw</td><td>Explicitly throws an exception</td></tr><tr><td>throws</td><td>Declares exceptions in method signature</td></tr></tbody></table> <p>Example(2 marks)</p>				Keyword	Purpose	try	Contains code that may cause an exception	catch	Handles the exception	finally	Executes whether exception occurs or not	throw
Keyword	Purpose												
try	Contains code that may cause an exception												
catch	Handles the exception												
finally	Executes whether exception occurs or not												
throw	Explicitly throws an exception												
throws	Declares exceptions in method signature												
3b	<p>Explain user-defined exceptions with a program.</p> <p>User-Defined Exceptions in Java(3 marks)</p> <p>A user-defined (custom) exception is an exception created by the programmer to handle application-specific error conditions.</p> <p>It is created by extending the Exception class (for checked exceptions) or RuntimeException class (for unchecked exceptions).</p>	5	CO4	L3									
	<p>Steps to Create a User-Defined Exception</p> <ol style="list-style-type: none">1. Create a class that extends Exception or RuntimeException2. Provide a constructor to pass the error message3. Use throw to explicitly throw the exception4. Handle it using try-catch <p>Example (2 marks)</p>												
4a	<p>Explain throw and throws keywords with examples.</p> <p>1. throw Keyword(2.5 marks)</p> <p>The throw keyword is used to explicitly throw a single exception from a method or block of code.</p> <p>Key Points</p> <ul style="list-style-type: none">• Used inside a method or block• Can throw one exception at a time• Used with exception objects• Transfers control to the nearest catch block <p>2. throws Keyword(2.5 marks)</p> <p>The throws keyword is used to declare exceptions that a method may pass to the calling method instead of handling them.</p> <p>Key Points</p> <ul style="list-style-type: none">• Used in method signature• Can declare multiple exceptions• Mainly used for checked exceptions• Responsibility of handling exception is passed to caller	5	CO4	L2									

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

	Enums improve type safety, readability, and maintainability compared to using constants like int or String.			
	<p>Defining an Enum</p> <pre>enum Day { MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY }</pre> <ul style="list-style-type: none"> Day is an enum Each identifier is a constant object of type Day <p>values() Method The values() method returns an array of all enum constants in the order they are declared.</p> <p>Example</p> <pre>class ValuesExample { public static void main(String[] args) { for (Day d : Day.values()) { System.out.println(d); } } }</pre> <p>valueOf() Method The valueOf() method returns the enum constant corresponding to the specified string name.</p> <p>Example</p> <pre>class ValueOfExample { public static void main(String[] args) { Day d = Day.valueOf("FRIDAY"); System.out.println(d); } }</pre>			
6b	<p>Explain Wrapper Classes and Autoboxing with examples.</p> <p>1. Wrapper Classes Wrapper classes in Java provide a way to convert primitive data types into objects. This is required because Java collections and many APIs work only with objects, 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>2. Autoboxing Autoboxing is the automatic conversion of a primitive type into its corresponding wrapper object 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