USN		CMRIT

Internal Assessment Test 2 – Set 1 –Jan 2026

Sub:	Object Oriented Programming with Java					Sub Code:	BCS306A	Branch	ISE
Date:	8/1/26	Duration:	90 min's	Max Marks:	50	Sem/Sec:	III / A, B & C		OBE

Solution

		MARKS	CO	RBT
1	a)Compare and contrast between overloading and overriding in Java with example program for each	10	CO1	L1
	Any 4 differences -	4M	CO3	L2

Overloading and overriding are distinct concepts in Java:

Feature	**Overloading**	**Overriding**
Concept	Multiple methods in the same class with the same name but different parameters.	A subclass provides a specific implem for a method already defined in its sup
Binding	Compile-time polymorphism (static binding).	Run-time polymorphism (dynamic me dispatch).
Parameters	Must be different (number, type, or order).	Must be the same as in the superclass i
Return Type	Can be different.	Must be the same (or a covariant retur in the superclass method.

Example for Overloading (Compile-time Polymorphism): (3M)

```
class Adder {
   static int add(int a, int b) {
      return a + b;
   }
   static int add(int a, int b, int c) {
      return a + b + c;
   }
}
```

Example for Overriding (Run-time Polymorphism):

```
class Vehicle {
   void run() {
      System.out.println("Vehicle is running");
   }
}
class Bike extends Vehicle {
   @Override
   void run() {
      System.out.println("Bike is running safely");
   }
}
```

3M

		MARKS	CO	RBT
2	Build a Java program to create a class named 'Shape'. Create 3 sub classes namely circle, triangle and square; each class has 2 methods named draw() and erase(). Demonstrate polymorphism concepts by developing suitable methods and main program.	10M	CO3	L3

3M

```
// Parent class
class Shape {
   void draw() {
      System.out.println("Drawing a shape");
   }
```

	<pre> void erase() {
	        System.out.println("Erasing a shape");
	    }
	}
	
	// Child class Circle
	class Circle extends Shape {
	    void draw() {
	        System.out.println("Drawing Circle");
	    }
	
	    void erase() {
	        System.out.println("Erasing Circle");
	    }
	}
	
	// Child class Triangle
	class Triangle extends Shape {
	    void draw() {
	        System.out.println("Drawing Triangle");
	    }
	
	    void erase() {
	        System.out.println("Erasing Triangle");
	    }
	}
	
	// Child class Square
	class Square extends Shape {
	    void draw() {
	        System.out.println("Drawing Square");
	    }
	
	    void erase() {
	        System.out.println("Erasing Square");
	    }
	}
	
	// Main class
	public class ShapeDemo {
	    public static void main(String[] args) {
	
	        Shape s; // Parent class reference
	
	        s = new Circle(); // Circle object
	        s.draw();
	        s.erase();
	
	        s = new Triangle(); // Triangle object
	        s.draw();
	        s.erase();
	
	        s = new Square(); // Square object
	        s.draw();
	        s.erase();
	    }
	}</pre>**Output**
	Drawing Circle
	Erasing Circle
	Drawing Triangle
	Erasing Triangle
	Drawing Square
	Erasing Square
		3M
		
		4M			

3	Define package. Explain the steps involved in creating a user-defined package with this example Build a Java program to create a package "balance" containing Account Class with displayBalance( ) method and import this package in another program to access method of Account Class.	10M	CO4	L3
	Package: **Definition of Package** A **package** in Java is a **namespace** that groups related classes and interfaces together. **Advantages of Package** • Avoids **name conflicts** • Provides **access protection** • Improves **code organization** • Supports **reusability**	2M		

	Step 1: Create Account Class inside balance Package
	Account.java
	package balance;
	
	public class Account {
	
	double bal = 5000.75;
	
	public void displayBalance() {
	System.out.println("Account Balance = " + bal);
	}
	}
		2M		

	Step 2: Compile the Package javac -d . Account.java ✓ This creates a folder named **balance** with Account.class inside it.	2M		

	Step 3: Import Package in Another Program
	TestAccount.java
	import balance.Account;
	
	public class TestAccount {
	public static void main(String[] args) {
	
	Account acc = new Account();
	acc.displayBalance();
	}
	}
		4M		

	Step 4: Compile and Run javac TestAccount.java java TestAccount			
	Output Account Balance = 5000.75			
4	What is customized exception? Build a Java program for a banking application to throw an exception, where a person tries to withdraw the amount even though he/she has lesser than minimum balance (Create a custom exception).	10	CO4	L3

```
// Custom Exception class
class MinimumBalanceException extends Exception {

    MinimumBalanceException(String msg) {
        super(msg);
    }
}

// Bank Account class
class BankAccount {
    double balance = 5000;          // initial balance
    final double MIN_BAL = 1000;    // minimum balance

    void withdraw(double amount) throws MinimumBalanceException {

        if (balance - amount < MIN_BAL) {
            throw new MinimumBalanceException(
                "Withdrawal denied! Minimum balance must be maintained."
            );
        }

        balance = balance - amount;
        System.out.println("Withdrawal Successful!");
        System.out.println("Remaining Balance = " + balance);
    }
}

// Main class
public class BankDemo {
    public static void main(String[] args) {

        BankAccount acc = new BankAccount();

        try {
            acc.withdraw(4500);   // static input
        }
        catch (MinimumBalanceException e) {
            System.out.println("Exception Caught: " + e.getMessage());
        }
        finally {
            System.out.println("Transaction completed");
        }
    }
}
```

2M 3M 5M

Output

Exception Caught: Withdrawal denied! Minimum balance must be maintained.
Transaction completed

Q. No.	Question	Marks	CO	Level
5.	Define Thread. Explain the two ways in which Java threads can be instantiated. Support your explanation with a sample program.	10M	CO5	L2

two Ways to Create Threads in Java

Java provides **two primary ways** to create threads:

3M

xtending the Thread Class	mplementing the Runnable Interface

// Thread by extending Thread class
class MyThread extends Thread {
    public void run() {
        System.out.println("Thread running: " + Thread.currentThread().getName());
    }
}
	// Thread by implementing Runnable interface
	class MyRunnable implements Runnable {
	    public void run() {
	        System.out.println("Thre running: " +

		public class ThreadExample1 {
		public static void main(String[] args) {
		MyThread t1 = new MyThread();
		MyThread t2 = new MyThread();
		
		t1.start();
		t2.start();
		}
		
		Output (Order may vary)
		Thread running: Thread-0
		Thread running: Thread-1
			Thread.currentThread().getName());
			}
			}
			
			public class ThreadExample2 {
			public static void main(String[] args) {
			MyRunnable r1 = new MyRunnable();
			MyRunnable r2 = new MyRunnable();
			
			Thread t1 = new Thread(r1);
			Thread t2 = new Thread(r2);
			
			t1.start();
			t2.start();
			}
			
			Output (Order may vary)
			Thread running: Thread-0
			Thread running: Thread-1
				3M
				
				4M		

6	Explain about type wrappers. Discuss auto boxing. & unboxing with suitable example.			10M	CO5	L2

Type Wrappers in Java (2M)

Definition

- **Wrapper classes** provide a **way to use primitive data types as objects**.
- Each primitive type has a corresponding **wrapper class**:

Primitive	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

4M

```
public class BoxingUnboxingDemo {
    public static void main(String[] args) {

        // Autoboxing: primitive int → Integer object
        int num = 25;
        Integer obj = num;   // autoboxing
        System.out.println("Wrapper object (autoboxing): " + obj);

        // Unboxing: Integer object → primitive int
        Integer anotherObj = new Integer(50);
        int anotherNum = anotherObj;  // unboxing
        System.out.println("Primitive value (unboxing): " + anotherNum);

        // Using both in an expression
        int sum = obj + anotherObj;   // unboxing occurs automatically
        System.out.println("Sum of values: " + sum);
    }
```

4M

	} **Output** Wrapper object (autoboxing): 25 Primitive value (unboxing): 50 Sum of values: 75			