USN | | | | | | | | | | |

Internal Assessment Test III – Mar-2024

Sub:	OOPS WITH JAVA					Sub Code:	BCS306A	Branch:	CSE	
Date:	05/03/24	Duration:	90 mins	Max Marks:	50	Sem/Sec:	III A,B,C		OBE	

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1.	**a)** How to write custom exceptions in Java? Explain with example.	[5]	CO4	L3
	b) What are different states in life cycle of a thread? Explain with neat diagram.	[5]	CO5	L2
2.	a) How can we create a Thread in java? Write a program to create a thread using any one way.	[5]	CO5	L2
	b) Define the following terms in Exception: 1. try 2. catch 3. finally 4. throw 5. throws	[5]	CO4	L1
3.	a) What is synchronization in java? Write the difference between method synchronization and block synchronization.	[5]	CO5	L2
	b) Write details about the below methods used in multithreading. 1. sleep() , 2. yield(), 3. join() 4. suspend() 5. resume() 6. stop() 7. getName() 8. isAlive() 9. start() 10. run()	[5]	CO5	L1
4.	a) Explain how autoboxing and unboxing work for boolean and character values in Java.	[5]	CO5	L3
	b) What is Inter Thread communication in java? Define the working of wait(), notify() and notifyAll() methods with example.	[5]	CO5	L3
5.	a) What is Enum in java? Define the below methods with example. 1. values() 2. valueOf() 3. ordinal()	[5]	CO5	L1
	b) What are Wrapper classes in java? Explain Autoboxing and Unboxing concept with example.	[5]	CO5	L2

6.	a) `class MyThread1 implements Runnable {`
	`// Complete the code print "thread-1" 5 times`
	`}`
	`class MyThread2 implements Runnable {`
	`// Complete the code print "thread-2" 5 times`
	`}`
	`public class GFG {`
	`public static void main(String[] args)`
	`{`
	`// (create both threads and start the threads to print the output) Complete the code`
	`}`
		[5]	CO5	L3

	b) What do you understand about Thread Priority? Write a program to set 7 priorities for main Thread	[5]	CO5	L3

CI **CCI** **HOD**

USN | | | | | | | | | | |

Internal Assessment Test III – Mar 2024

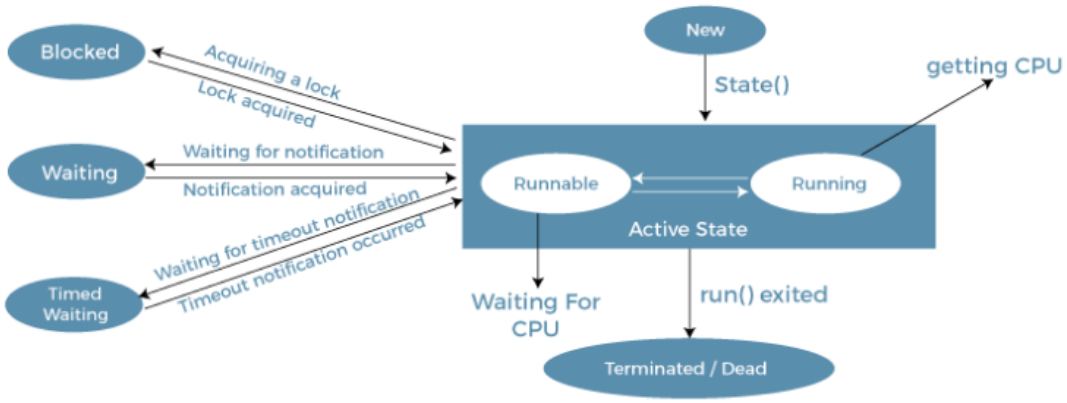
Sub:	OOPS WITH JAVA					Sub Code:	BCS306A	Branch:	CSE
Date:	05/2/24	Duration:	90 mins	Max Marks:	50	Sem/Sec:	III A,B,C		OBE

Answer any FIVE FULL Questions

		MARKS	CO	RBT

1	**a)** How to write custom exceptions in Java? Explain with example.
	Explanation - In Java, you can create custom exceptions by extending the **Exception** class or one of its subclasses.
	Program Example:
	`public class CustomException extends Exception {`
	`  public CustomException(String message) {`
	`  super(message);`
	`  }`
	`}`
	`public class Example {`
	`  public static void main(String[] args) {`
	`  try {`
	`  // Some code that may throw your custom exception`
	`  validateInput(5);`
	`  } catch (CustomException e) {`
	`  System.out.println("Caught CustomException: " + e.getMessage());`
	`  }`
	`  }`
	
	`  private static void validateInput(int value) throws CustomException {`
	`  if (value < 10) {`
	`  throw new CustomException("Input value must be greater than or equal to 10");`
	`  }`
	`  // Continue with the rest of the code if validation passes`
	`  System.out.println("Input value is valid");`
	`  }`
	`}`
	`try {`
	`  // Some code that may throw your custom exception`
	`  validateInput(5);`
	`} catch (CustomException e) {`
	`  System.out.println("Caught CustomException: " + e.getMessage());`
	`  // Additional error-handling logic if needed`
	`}`
		[5]	CO4	L3

	b) What are different states in life cycle of a thread? Explain with neat diagram. **Explanation:** The life cycle of a thread in Java refers to the various states a thread goes through during its execution. There are six different states in the life cycle of a thread: **New (or Born):** The thread is in this state when an instance of the Thread class is created, but the start() method is not yet called. At this point, the thread is considered to be born, but it is not yet eligible to run. **Runnable (or Ready to Run):** After calling the start() method, the thread moves to the runnable state.	[5]	CO5	L2

	In this state, the thread is ready to run, but the scheduler has not yet selected it to be the running thread. The thread scheduler determines which runnable thread will execute next. **Blocked (or Waiting):** A thread can transition to the blocked state for several reasons, such as waiting for I/O operations or waiting for a lock. In the blocked state, the thread cannot continue its execution until the condition causing it to be blocked is resolved. **Timed Waiting:** This state is similar to the blocked state but with a specified time duration. A thread enters the timed waiting state when it calls methods like sleep() or join() with a specified timeout. **Waiting:** A thread enters the waiting state when it is waiting for another thread to perform a particular action. Threads in the waiting state can be awakened by other threads using methods like notify() and notifyAll(). **Terminated (or Dead):** A thread enters the terminated state when its run() method completes or when the stop() method is called. Once in this state, a thread cannot be started again. **Diagram:**  Life Cycle of a Thread			

2	a) How can we create a Thread in java? Write a program to create a thread using any one way.
	Explanation: There are two ways to create a thread:
	1. By extending Thread class
	2. By implementing Runnable interface.
	
	Program:
	class Multi extends Thread{
	public void run(){
	System.out.println("thread is running...");
	}
	public static void main(String args[]){
	Multi t1=new Multi();
	t1.start();
		[5]	CO5	L2

	} }			
	b) Define the following terms in Exception: 1. try 2. catch 3. finally 4. throw 5. throws **Explanation:** **1. try:** The try block encloses a section of code where an exception might occur. It is used to define a block of statements that need to be monitored for exceptions. If an exception occurs within the try block, the control is transferred to the corresponding catch block. **2. catch:** • The catch block follows a try block and is used to catch and handle exceptions that might occur in the associated try block. • The catch block specifies the type of exception it can catch and provides code to handle the exception. **3. finally:** • The finally block is used to define a block of code that will be executed whether an exception is thrown or not. • It is typically used to perform cleanup operations, such as closing resources, regardless of whether an exception occurs or not. **4. throw:** • The throw keyword is used to explicitly throw an exception in Java. • It is followed by an instance of an exception or a throwable expression. • It is often used within conditional statements or methods to indicate that an exceptional condition has occurred **5. throws:** • The throws keyword is used in the method signature to declare that the method might throw one or more types of exceptions. • It indicates that the responsibility of handling these exceptions lies with the calling method or the caller. • Multiple exception types can be declared using a comma-separated list.	[5]	CO4	L1
3	a) What is synchronization in java? Write the difference between method synchronization and block synchronization. **Synchronization:** In Java, synchronization is a mechanism that helps control the access of multiple threads to shared resources. When multiple threads operate concurrently and share data, synchronization ensures that only one thread can access a shared resource at a time. This helps prevent race conditions and ensures the consistency of shared data. **Difference:** **Method Synchronization:** • In method synchronization, the synchronized keyword is used in the method	[5]	CO5	L2

	declaration. • When a thread invokes a synchronized method, it acquires a lock for the object on which the method is invoked. • Other threads trying to invoke synchronized methods on the same object will be blocked until the lock is released. **public class SynchronizedExample {** **// Synchronized method** **public synchronized void synchronizedMethod() {** **// Code that needs to be synchronized** **}** **}** **Block Synchronization:** • In block synchronization, the synchronized keyword is used within a block of code. • It allows more fine-grained control over synchronization, as it enables synchronization on a specific object rather than the entire method. • Multiple threads can execute non-synchronized parts of the code concurrently. **public class SynchronizedExample {** **private Object lock = new Object();** **// Synchronized block** **public void synchronizedBlock() {** **synchronized (lock) {** **// Code that needs to be synchronized** **}** **}** **}**			
	b) Write details about the below methods used in multithreading. 1. sleep() , 2. yield(), 3. join() 4. suspend() 5. resume() 6. stop() 7. getName() 8. isAlive() 9. start() 10. run() **Explanation :** **1. sleep():** Pauses the execution of the current thread for a specified amount of time, allowing other threads to execute. **2. yield():** Suggests to the scheduler that the current thread is willing to yield its current use of a processor, allowing other threads to run. **3. join():** Waits for the specified thread to finish its execution before the current thread continues. **4. suspend():** Deprecated method that temporarily halts the execution of a thread. Should be avoided due to potential deadlock issues. **5. resume():** Deprecated method that resumes the execution of a suspended thread. Should be avoided due to potential deadlock issues. **6. stop():** Deprecated method that abruptly stops the execution of a thread. Should be avoided due to unsafe termination. **7. getName():** Returns the name of the thread.	[5]	CO5	L1

	8. isAlive(): Checks if the thread is still alive (has been started and not terminated). **9. start():** Initiates the execution of a thread by invoking its run() method. **10. run():** Contains the code to be executed by the thread when started using the start() method.			

4	a) Explain how autoboxing and unboxing work for boolean and character values in Java.
	Explanation:
	Autoboxing and unboxing are mechanisms in Java that allow automatic conversion between primitive data types and their corresponding wrapper classes. Autoboxing is the process of converting a primitive type to its corresponding wrapper class, while unboxing is the process of extracting the primitive value from the wrapper class. This process simplifies code and enhances readability.
	For boolean and character values:
	1. Autoboxing (Primitive to Wrapper):
	
	When a boolean or char primitive is assigned to an object of the corresponding wrapper class (Boolean or Character), autoboxing automatically occurs.
	Example:
	boolean primitiveBoolean = true;
	Boolean wrapperBoolean = primitiveBoolean; // Autoboxing
	2. Unboxing (Wrapper to Primitive):
	When a Boolean or Character object is assigned to a boolean or char primitive, unboxing automatically occurs.
	Example:
	Boolean wrapperBooleanObj = true;
	boolean primitiveBoolean = wrapperBooleanObj; // Unboxing
		[5]	CO5	L3

	b) What is Inter Thread communication in java? Define the working of wait(), notify() and notifyAll() methods with example. **Explanation:** Inter Thread Communication in Java refers to the communication between two or more threads to synchronize their actions and coordinate their execution. This is achieved using methods like wait(), notify(), and notifyAll() provided by the Object class. **1. wait():** • The wait() method is used by a thread to release the lock it holds and wait until another thread invokes notify() or notifyAll() on the same object. • It should be called within a synchronized block or method to avoid illegal monitor state exception. **2. notify():** • The notify() method is used to wake up one of the threads that are currently waiting on the same object. • It is important to note that notify() does not release the lock immediately; the lock is released only when the synchronized block or method is exited. **3. notifyAll():** • The notifyAll() method is used to wake up all threads that are currently waiting on the same object. • Like notify(), notifyAll() does not release the lock immediately.	[5]	CO5	L3
5	a) What is Enum in java? Define the below methods with example. 1. values() 2. valueOf()	[5]	CO5	L1

3. ordinal()

Explanation of Enum: In Java, an enum (enumeration) is a special data type that represents a set of predefined constants. Enumerations are typically used to define a fixed set of values that represent distinct elements within a program. The enum type was introduced in Java 5 to provide a more structured way to represent sets of constant values.

1. values():

- The values() method returns an array containing all the enum constants in the order they are declared.
- This method is automatically generated by the Java compiler when you create an enum.

Example:

```
enum Days {
   SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
}

public class EnumExample {
   public static void main(String[] args) {
      Days[] daysArray = Days.values();
      for (Days day : daysArray) {
         System.out.println(day);
      }
   }
}
```

2. valueOf():

- The valueOf(String name) method returns the enum constant with the specified name.
- It throws an IllegalArgumentException if the specified name is not a valid constant.

Example:

```
enum Colors {
   RED, GREEN, BLUE
}

public class EnumExample {
   public static void main(String[] args) {
      Colors color = Colors.valueOf("RED");
      System.out.println("Selected color: " + color);
   }
}
```

3. ordinal():

- The ordinal() method returns the position of an enum constant in its enum declaration, starting from zero.
- It can be useful for comparing the relative order of enum constants.

Example:

```
enum Months {
   JANUARY, FEBRUARY, MARCH, APRIL, MAY, JUNE, JULY, AUGUST, SEPTEMBER,
OCTOBER, NOVEMBER, DECEMBER
}

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

```
    Months month = Months.MARCH;
    System.out.println("Position of " + month + " is " + month.ordinal());
  }
}
```

b) What are Wrapper classes in java? Explain Autoboxing and Unboxing concept with example.

[5] CO5 L2

Explanation :

Wrapper classes in Java are a set of classes that provide a way to represent primitive data types as objects. In Java, primitive data types (such as int, char, boolean, etc.) are not objects, but sometimes it is necessary to treat them as objects. Wrapper classes encapsulate and wrap primitive data types in an object, allowing them to be used in situations where objects are required.
Here are the wrapper classes for the primitive data types:
Byte - for byte
Short - for short
Integer - for int
Long - for long
Float - for float
Double - for double
Character - for char
Boolean - for boolean

Autoboxing: It is the process of converting a primitive data type into its corresponding wrapper class object automatically by the Java compiler.
Unboxing: It is the process of converting a wrapper class object into its corresponding primitive data type automatically by the Java compiler.
Example:

```
public class WrapperExample {
  public static void main(String[] args) {
    // Autoboxing: converting primitive to wrapper
    Integer intValue = 42; // int to Integer
    Double doubleValue = 3.14; // double to Double
    Boolean boolValue = true; // boolean to Boolean

    // Autoboxing in collections
    java.util.List<Integer> integerList = new java.util.ArrayList<>();
    integerList.add(1); // int to Integer
    integerList.add(2); // int to Integer

    // Unboxing: converting wrapper to primitive
    int intPrimitive = intValue; // Integer to int
    double doublePrimitive = doubleValue; // Double to double
    boolean boolPrimitive = boolValue; // Boolean to boolean

    // Unboxing from collections
    int firstValue = integerList.get(0); // Integer to int

    System.out.println("Autoboxing and Unboxing Example:");
    System.out.println("Autoboxing - Integer: " + intValue);
```

```
        System.out.println("Autoboxing - Double: " + doubleValue);
        System.out.println("Autoboxing - Boolean: " + boolValue);
        System.out.println("Autoboxing in collections: " + integerList);
        System.out.println("Unboxing - int: " + intPrimitive);
        System.out.println("Unboxing - double: " + doublePrimitive);
        System.out.println("Unboxing - boolean: " + boolPrimitive);
        System.out.println("Unboxing from collections: " + firstValue);
    }
}
```

6	a)	[5]	CO5	L3

```
class MyThread1 implements Runnable {

// Complete the code print "thread-1" 5 times


}

class MyThread2 implements Runnable {
// Complete the code print "thread-2" 5 times
}

public class GFG {
    public static void main(String[] args)
    {
// (create both threads and start the threads to print the output)
Complete the code


}
```

Complete Program:

```
class MyThread1 implements Runnable {
    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println("thread-1");
        }
    }
}

class MyThread2 implements Runnable {
    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println("thread-2");
        }
    }
}

public class GFG {
    public static void main(String[] args) {
        // Creating instances of the custom threads
        MyThread1 myThread1 = new MyThread1();
        MyThread2 myThread2 = new MyThread2();

        // Creating Thread objects and passing the custom threads to them
        Thread thread1 = new Thread(myThread1);
```

```
        Thread thread2 = new Thread(myThread2);

        // Starting the threads to run concurrently
        thread1.start();
        thread2.start();
    }
}
```

b) What do you understand about Thread Priority? Write a program to set 7 priorities for main Thread

[5] | CO5 | L3

Thread Priority: In Java, thread priority is a way to indicate the importance or urgency of a thread to the scheduler. Threads with higher priority have a better chance of being executed before threads with lower priority. However, thread priority is just a hint to the scheduler, and it doesn't guarantee the exact order of execution.

Thread priority in Java is represented by an integer value ranging from Thread.MIN_PRIORITY (1) to Thread.MAX_PRIORITY (10), with Thread.NORM_PRIORITY (5) being the default.

Program :

```
public class MainThreadPriorityExample {
    public static void main(String[] args) {
        // Getting the current main thread
        Thread mainThread = Thread.currentThread();

        // Setting priority for the main thread
        mainThread.setPriority(7);

        // Displaying the priority of the main thread
        System.out.println("Main Thread Priority: " + mainThread.getPriority());

        // Rest of the main thread code
        for (int i = 1; i <= 5; i++) {
            System.out.println("Main Thread executing iteration: " + i);
        }
    }
}
```

CI **CCI** **HOD**

PO Mapping

CO-PO and CO-PSO Mapping																				
Course Outcomes		Blooms Level	Modules covered	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3	PSO4
CO1	Analyze the performance of the algorithms, state the efficiency using asymptotic notations, and analyze mathematically the complexity of the algorithm.	L2	M1	3	3	2	3	2	-	-	-	-	-	-	2	-	-	-	-
CO2	Apply divide and conquer approaches and decrease and conquer approaches in solving the problems and analyze the same	L3	M2	3	3	2	3	2	-	-	-	-	-	-	2	-	-	-	-
CO3	Apply the appropriate algorithmic design technique like the greedy method, transform and conquer approaches and compare the efficiency of algorithms to solve the given problem.	L3	M3	3	3	2	3	2	-	-	-	-	-	-	2	-	-	-	-
CO4	Apply and analyze dynamic programming approaches to solve some problems. and improve an algorithm's time efficiency by sacrificing space.	L3	M4	3	3	2	3	2	-	-	-	-	-	-	2	-	-	-	-
CO5	Apply and analyze backtracking, branch and bound methods to describe P, NP, and NP-complete problems.	L3	M5	3	2	2	3	2	-	-	-	-	-	-	2	-	-	-	-

COGNITIVE LEVEL	REVISED BLOOMS TAXONOMY KEYWORDS
L1	List, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name, who, when, where, etc.
L2	summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend
L3	Apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover.
L4	Analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain,

	infer.
L5	Assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize.

PROGRAM OUTCOMES (PO), PROGRAM SPECIFIC OUTCOMES (PSO)				CORRELATION LEVELS	
PO1	Engineering knowledge	PO7	Environment and sustainability	0	No Correlation
PO2	Problem analysis	PO8	Ethics	1	Slight/Low
PO3	Design/development of solutions	PO9	Individual and team work	2	Moderate/ Medium
PO4	Conduct investigations of complex problems	PO10	Communication	3	Substantial/ High
PO5	Modern tool usage	PO11	Project management and finance		
PO6	The Engineer and society	PO12	Life-long learning		
PSO1	Develop applications using different stacks of web and programming technologies				
PSO2	Design and develop secure, parallel, distributed, networked, and digital systems				
PSO3	Apply software engineering methods to design, develop, test and manage software systems.				
PSO4	Develop intelligent applications for business and industry				