USN | | | | | | | | | |

CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A++ GRADE BY NAAC

Internal Assessment Test 3 – December 2023

Sub:	**Object Oriented Programming with JAVA**				Sub Code:	**BCS306A**	Branch:	**AIML/CSE(AIML)/AIDS/CSE (AIML)**
Date:	**4/3/24**	Duration:	**90 minutes**	Max Marks:	**50**	Sem/Sec:	**III -A, B & C**	**OBE**

		Answer any FIVE FULL Questions	**MARKS**	**CO**	**RBT**
1	a	Define package. What are the steps involved in creating user defined packages with an example	[08]	4	L3

	b	**Predict the output**
		
		`public class A{`
		`    public static void main(String[] args){`
		`        int a[]=new int[2];`
		`        a[10]=5;`
		`        try{`
		`        int num =2/0;`
		`        }`
		`        catch(Exception ex){`
		`        System.out.println("Divide by Zero");`
		`        }`
		`        finally{`
		`        System.out.println("last block");`
		`        }`
		`        }`
		`        }`
			[02]	4	L3

2	a	What is an exception ? Explain the different types of Exception handling with an example.	[08]	4	L2
	b	Write 2 difference between ‘thow’ and ‘thows’ keyword	[02]	4	L3
3	a	What is thread? Explain the two ways to creating a thread in java with an example	[08]	5	L2
	b	What is enumeration? Write syntax and example	[02]	5	L3
4	a	What is the need of synchronization? Explain with an example how the synchronization is implemented in java.	[08]	5	L3
	b	Give an example for values() and valueOf() methods	[02]	5	L3
5	a	Describe thread priority.How to assign and get priority	[07]	5	L2
	b	Write a note TypeWrapper?	[03]	5	L2
6	a	Explain and write a syntax of try and catch block to handle multiple exception. Explain and write use of Alive() and join methods.	[08]	5	L3
	b	What is an Autoboxing and unboxing?	[02]	5	L3

CI **CCI** **HoD**

Internal Assessment Test 2 December 2024								
Sub:	OOP with Java				Sub Code:	BCS306A	Branch:	AIML
Date :	12 -12 -24	Duration:90 m	Max Marks:	50	Sem /Sec:	III, A/B/C		

	Answer any FIVE FULL Questions	Marks	CO	RBT
1 (a)	(see below)	8	CO3	L1, L4

Definition of Package:
A package in Java is a namespace that organizes classes and interfaces. It helps to avoid naming conflicts and provides access protection. Packages also make it easier to locate and use the classes in a project.

Steps to Create a User-Defined Package:

Create the Package:
Use the package keyword at the beginning of the Java file to define a package.
Example:
java
Copy code

```java
package myPackage;

public class MyClass {
    public void displayMessage() {
        System.out.println("This is a user-defined package.");
    }
}
```

1.

Compile the Class:
Use the javac command with the -d option to specify the location of the package.
Command:
shell
Copy code

```shell
javac -d . MyClass.java
```

2. This creates a directory named myPackage and places the compiled .class file inside it.

Import the Package in Another Program:
Use the import statement to include the package in another program.
Example:
java
Copy code

```java
import myPackage.MyClass;
```

```
public class TestPackage {
    public static void main(String[] args) {
        MyClass obj = new MyClass();
        obj.displayMessage();
    }
}
```

3.
4. Run the Program:
   Compile and run the program that imports the package. Ensure the package directory is in the classpath.

Complete Example:

Step 1: Create the Package (File: MyClass.java):

java
Copy code
```
package myPackage;

public class MyClass {
    public void displayMessage() {
        System.out.println("This is a user-defined package.");
    }
}
```

Step 2: Compile the Package:

shell
Copy code
```
javac -d . MyClass.java
```

Step 3: Import and Use the Package (File: TestPackage.java):

java
Copy code
```
import myPackage.MyClass;

public class TestPackage {
    public static void main(String[] args) {
        MyClass obj = new MyClass();
        obj.displayMessage();
    }
}
```

Step 4: Run the Program:

shell
Copy code
```
java TestPackage
```

	Output: kotlin Copy code This is a user-defined package.			
1(b)	Explanation: The program throws an ArrayIndexOutOfBoundsException at the line a[10] = 5;. Since this exception occurs before the try block, the program terminates, and the catch or finally blocks are not executed. Output: arduino Copy code Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10	2		

2. (a)	*Definition of Exception:*
	An exception in Java is an event that disrupts the normal flow of the program's execution. It occurs during runtime and can be caused by logical errors, invalid inputs, or hardware failures. For example, dividing a number by zero or accessing an array index out of bounds can result in exceptions.
	
	--- 	
	
	Types of Exceptions:
	
	1. *Checked Exceptions:*
	These exceptions are checked at compile time. The program will not compile unless they are handled.
	Examples:
	○ *IOException: Occurs during input-output operations.\n\n Code Example:*
	
	java
	Copy code
	`import java.io.*;`
	
	`public class CheckedExceptionDemo {`
	`    public static void main(String[] args) {`
	`        try {`
	`            FileReader file = new FileReader("file.txt"); // File may not exist`
	`        } catch (FileNotFoundException e) {`
	`            System.out.println("File not found.");`
	`        }`
	`    }`
	`}`
	
	2.
	3. *Unchecked Exceptions:*
	These exceptions occur at runtime and are not checked at compile time. They usually represent programming bugs.
	Examples:
		8	CO3	L2

- ArithmeticException: Division by zero.\n\n
- ArrayIndexOutOfBoundsException: Accessing invalid array index.\n\n Code Example:

java
Copy code
```java
public class UncheckedExceptionDemo {
    public static void main(String[] args) {
        try {
            int num = 10 / 0; // Division by zero
        } catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero.");
        }
    }
}
```

4.
5. Errors:
   Errors are serious problems that cannot be handled by Java programs. They occur outside the application, such as OutOfMemoryError or StackOverflowError.

Exception Handling Mechanisms in Java:

try-catch Block:
Used to handle exceptions by catching them.\n\n Syntax:
java
Copy code
```java
try {
    // Code that might throw an exception
} catch (ExceptionType e) {
    // Code to handle the exception
}
```
Example:
java
Copy code
```java
public class ExceptionHandling {
    public static void main(String[] args) {
        try {
            int[] numbers = {1, 2, 3};
            System.out.println(numbers[5]); // ArrayIndexOutOfBoundsException
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Invalid array index accessed.");
        }
    }
}
```

1.

Multiple catch Blocks:
Used to handle different exceptions.\n\n Example:
java
Copy code
```java
public class MultipleCatchExample {
    public static void main(String[] args) {
        try {
            int result = 10 / 0;
            String text = null;
            text.length(); // NullPointerException
        } catch (ArithmeticException e) {
            System.out.println("Arithmetic Exception caught.");
        } catch (NullPointerException e) {
            System.out.println("Null Pointer Exception caught.");
        }
    }
}
```

2.

finally Block:
Always executes, whether an exception occurs or not.
Example:
java
Copy code

```
public class FinallyExample {
    public static void main(String[] args) {
        try {
            int result = 10 / 2;
            System.out.println(result);
        } catch (Exception e) {
            System.out.println("Exception caught.");
        } finally {
            System.out.println("Finally block executed.");
        }
    }
}
```

3.
4. *throw and throws:*

throw: Explicitly throw an exception.\n\n Example:
java
Copy code

```
public void validateAge(int age) {
    if (age < 18) {
        throw new IllegalArgumentException("Age must be 18 or above.");
    }
}
```

○

throws: Declare exceptions in the method signature.\n\n Example:
java
Copy code

```
public void readFile() throws IOException {
    FileReader file = new FileReader("file.txt");
}
```

○

2(b)

2

throw	throws
1. Used to explicitly throw an exception from a method or block of code.	1. Used to declare the exceptions that a method can throw.
2. Follows the syntax: throw new Exception();	2. Follows the syntax: void method() throws Exception {}

Examples:

throw Example:

java

Copy code

```
public void validateAge(int age) {
    if (age < 18) {
        throw new IllegalArgumentException("Age must be 18 or above.");
    }
}
```

throws Example:

java

Copy code

```
public void readFile() throws IOException {
    FileReader file = new FileReader("file.txt");
}
```

3(a)

What is a Thread?
A thread is a lightweight subprocess in Java that allows concurrent execution of tasks within a program. It runs independently but shares memory and resources with other threads. Threads are part of the java.lang.Thread class or the java.lang.Runnable interface.

Two Ways to Create a Thread:

Extending the Thread Class:
A class extends Thread and overrides its run() method to define the task to be executed.
Example:
java
Copy code

```
class MyThread extends Thread {
   public void run() {
      System.out.println("Thread is running by extending Thread class.");
   }
}

public class ThreadExample1 {
   public static void main(String[] args) {
      MyThread t1 = new MyThread();
      t1.start(); // Start the thread
   }
}
```

1. Explanation:
   - The start() method is called to begin the execution of the thread.
   - The run() method contains the logic of the thread.

Implementing the Runnable Interface:
A class implements Runnable and provides the task in the run() method. A Thread object is created and passed the instance of the class.
Example:
java
Copy code

```
class MyRunnable implements Runnable {
   public void run() {
      System.out.println("Thread is running by implementing Runnable interface.");
   }
}
```

8

CO3

L3

```
public class ThreadExample2 {
    public static void main(String[] args) {
        MyRunnable myRunnable = new MyRunnable();
        Thread t2 = new Thread(myRunnable);
        t2.start(); // Start the thread
    }
}
```

2. Explanation:
    - Runnable allows multiple classes to implement threads, as it does not restrict inheritance like extending Thread.
    - A Thread object is required to invoke the start() method.

Comparison of the Two Methods:

Extending Thread	Implementing Runnable
Inherits Thread, so the class cannot extend other classes.	Implements Runnable, allowing mu inheritance.
Simpler for small programs.	More flexible and preferred in real- applications.

3(b) 2

What is Enumeration?
An enumeration (enum) in Java is a special data type used to define a set of predefined constants. Enums are used when a variable can take one of a limited set of values, such as days of the week, months, or directions.

Syntax:

java

Copy code

```
enum EnumName {

    CONSTANT1, CONSTANT2, CONSTANT3

}
```

Example:

java

Copy code

```java
enum Days { MONDAY, TUESDAY, WEDNESDAY }


public class EnumExample {
    public static void main(String[] args) {
        for (Days day : Days.values()) { // Loop through enum constants
            System.out.println(day);
        }


        // Access specific enum constant
        Days today = Days.MONDAY;
        System.out.println("Today is: " + today);
    }
}
```

Output:

csharp

Copy code

```
MONDAY
TUESDAY
WEDNESDAY
Today is: MONDAY
```

4a	Need for Synchronization:
	Synchronization in Java is necessary to control access to shared resources in a multithreaded environment. Without synchronization, multiple threads can simultaneously modify shared data, leading to data inconsistency or race conditions.
	
	Example Scenario:
	Consider a situation where two threads are incrementing a shared counter. If one thread reads the value of the counter while the other is updating it, the final value may be incorrect. Synchronization ensures that only one thread accesses the shared resource at a time.
	
	--- 	
	
	Implementation of Synchronization in Java:
	Java provides the synchronized keyword to achieve synchronization. It can be applied to methods or blocks of code to prevent multiple threads from accessing the resource simultaneously.
	
	Example Code:
	
	java
	
	Copy code
	
	`class Counter {`
	`    private int count = 0;`
	
	`    // Synchronized method`
	`    public synchronized void increment() {`
	`        count++;`
	`    }`
	
	`    public int getCount() {`
	`        return count;`
	`    }`
	`}`
		8	CO3	L1, L3

```
class MyThread extends Thread {
    Counter counter;


    MyThread(Counter counter) {
        this.counter = counter;
    }


    public void run() {
        for (int i = 0; i < 1000; i++) {
            counter.increment();
        }
    }
}


public class SyncDemo {
    public static void main(String[] args) throws InterruptedException {
        Counter counter = new Counter();


        MyThread t1 = new MyThread(counter);
        MyThread t2 = new MyThread(counter);


        t1.start();
        t2.start();


        t1.join();
        t2.join();
        System.out.println("Final Counter Value: " + counter.getCount());
    }
}
```

Explanation of the Code:

1. Shared Resource:
   The Counter object is shared between two threads (t1 and t2).
2. Synchronized Method:
   The increment() method is synchronized to ensure only one thread can execute it at a time.
3. Output:
   Without synchronization, the counter value may be less than expected. With synchronization, the final value will be correct (2000 in this case).

4b | 2 | CO3 | L1, L3

values() Method:
The values() method returns an array of all constants defined in an enum. It is used to iterate through the constants.

valueOf() Method:
The valueOf(String name) method returns the enum constant corresponding to the specified name. The name must match exactly, or it throws an IllegalArgumentException.

Example:

java
Copy code
```
enum Colors { RED, GREEN, BLUE }

public class EnumMethodsExample {
    public static void main(String[] args) {
        // Using values() to iterate through enum constants
        System.out.println("All colors:");
        for (Colors color : Colors.values()) {
            System.out.println(color);
        }

        // Using valueOf() to get a specific enum constant
        Colors selectedColor = Colors.valueOf("RED");
        System.out.println("Selected color: " + selectedColor);
    }
}
```

	Output: yaml Copy code All colors: RED GREEN BLUE Selected color: RED			

5a	Thread Priority
	
	Thread priority determines the order in which threads are executed relative to one another. In many systems, threads with higher priority are given more CPU time, and they can preempt lower-priority threads. Thread priorities are typically used in real-time and multitasking environments to ensure that critical tasks are given preference.
	
	Assigning Thread Priority In Java, thread priorities can be set using the Thread class. The priority of a thread can be assigned using the setPriority(int priority) method. Thread priorities are represented by integer values between Thread.MIN_PRIORITY (1) and Thread.MAX_PRIORITY (10), with the default being Thread.NORM_PRIORITY (5).
	
	Example:
	
	java
	Copy code
	`Thread thread = new Thread();`
	`thread.setPriority(Thread.MAX_PRIORITY); // Assign maximum priority`
	
	Getting Thread Priority To get the current priority of a thread, you can use the getPriority() method.
	
	Example:
	
	java
	Copy code
	`int priority = thread.getPriority(); // Get the current priority`
	
	Thread Priority Levels
	
	● Thread.MIN_PRIORITY: 1
	● Thread.NORM_PRIORITY: 5 (default)
	● Thread.MAX_PRIORITY: 10
	
	Important Notes:
	
	● Thread priority is relative and can be affected by the underlying operating system’s scheduling policies.
	● Not all operating systems respect thread priorities equally; some may not give higher priority threads more CPU time.
		7	CO3	L2

5a	Type Wrappers in Java
	
	Type wrappers, also known as wrapper classes, are used in Java to represent primitive data types as objects. Each primitive type has a corresponding wrapper class in the java.lang package. These classes allow primitive values to be treated as objects, which is useful when working with collections, or when you need to use methods that only accept objects.
	
	Wrapper Classes:
	
	● byte → Byte
	● short → Short
	● int → Integer
	● long → Long
	● float → Float
	● double → Double
	● char → Character
	● boolean → Boolean
	
	Key Features:
	
	Autoboxing and Unboxing: Java automatically converts between primitives and their corresponding wrapper classes. This feature is known as autoboxing (primitive to object) and unboxing (object to primitive).
	Example:
	java
	Copy code
	Integer intObj = 10; // Autoboxing
	int num = intObj; // Unboxing
	
	1.
	2. Methods: Wrapper classes provide utility methods like parseInt(), valueOf(), compareTo(), and more for converting between types or performing operations.
	
	Example:
	
	java
	Copy code
	int num = Integer.parseInt("123"); // Converts String to int
	
	Wrapper classes are essential for handling primitive types as objects, enabling their use in contexts that require objects (e.g., collections).
		3	CO3	L4

6a

Handling Multiple Exceptions in Java (try-catch block)

In Java, you can handle multiple exceptions using a single try-catch block by specifying multiple exception types in the catch clause. This helps manage various exception types that could arise from a particular block of code.

Syntax:

```
java
Copy code
try {
   // Code that may throw exceptions
} catch (ExceptionType1 e1) {
   // Handling ExceptionType1
} catch (ExceptionType2 e2) {
   // Handling ExceptionType2
} catch (ExceptionType3 e3) {
   // Handling ExceptionType3
} finally {
   // Optional: Code that always executes, whether an exception occurs or not
}
```

Example:

```
java
Copy code
try {
   int result = 10 / 0; // ArithmeticException
   String str = null;
   str.length(); // NullPointerException
} catch (ArithmeticException e) {
   System.out.println("Arithmetic Exception: " + e.getMessage());
} catch (NullPointerException e) {
   System.out.println("Null Pointer Exception: " + e.getMessage());
} finally {
   System.out.println("Finally block executed.");
}
```

Multi-catch Block (Java 7 and above):
In Java 7 and later, you can use a multi-catch block to handle different exceptions in a single catch block by separating them with a pipe (|).

Syntax:

```
java
Copy code
try {
   // Code that may throw exceptions
} catch (ExceptionType1 | ExceptionType2 | ExceptionType3 e) {
   // Handling multiple exceptions
}
```

8

Example:

java
Copy code

```
try {
   int result = 10 / 0;  // ArithmeticException
   String str = null;
   str.length(); // NullPointerException
} catch (ArithmeticException | NullPointerException e) {
   System.out.println("Exception occurred: " + e.getMessage());
}
```

isAlive() and join() Methods in Java

1. isAlive() Method:

The isAlive() method is used to check if a thread is still running or has finished executing.

Syntax:
java
Copy code

```
thread.isAlive();
```

-
- Return Type: boolean
    - true: The thread is still alive (running).
    - false: The thread has finished its execution or has not been started yet.

Example:

java
Copy code

```
Thread t = new Thread(() -> {
   System.out.println("Thread is running.");
});
t.start();
System.out.println("Is thread alive? " + t.isAlive()); // Returns true
```

2. join() Method:

The join() method allows one thread to wait for the completion of another thread. When join() is called on a thread, the current thread will pause execution until the thread on which join() was called has finished.

Syntax:
java
Copy code

```
thread.join();
```

-

Syntax with timeout:

java
Copy code
```
thread.join(long millis);
```

●

Example without timeout:
java
Copy code
```
Thread t = new Thread(() -> {
    System.out.println("Thread is running.");
});
t.start();
try {
    t.join();  // Main thread waits for t to finish
} catch (InterruptedException e) {
    e.printStackTrace();
}
System.out.println("Thread has finished.");
```

●

Example with timeout:
java
Copy code
```
Thread t = new Thread(() -> {
    try {
        Thread.sleep(1000);  // Sleep for 1 second
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
});
t.start();
try {
    t.join(500);  // Main thread waits for t to finish, with a timeout of 500 ms
} catch (InterruptedException e) {
    e.printStackTrace();
}
System.out.println("Thread either finished or timed out.");
```

●

Summary:

- The try-catch block handles multiple exceptions by either specifying each exception type in separate catch blocks or using multi-catch (Java 7+).
- The isAlive() method checks if a thread is still running.
- The join() method ensures that one thread waits for another to finish before continuing execution.

6. (b)	Autoboxing and Unboxing are features in Java that automatically convert between primitive types and their corresponding wrapper classes.
	
	Autoboxing: The automatic conversion of a primitive type to its corresponding wrapper class when required.
	Example:
	java
	Copy code
	`int num = 10;`
	`Integer intObj = num;  // Autoboxing: int to Integer`
	
	1.
	
	Unboxing: The automatic conversion of a wrapper class object to its corresponding primitive type when needed.
	Example:
	java
	Copy code
	`Integer intObj = 10;`
	`int num = intObj;  // Unboxing: Integer to int`
	
	2.
	
	Autoboxing and unboxing simplify working with collections (which require objects) while using primitives, as Java handles the conversion automatically.
		2	CO3	L2