Day 11 - 18th June 2025

|  |  |
| --- | --- |
| Advance concepts | Collections Framework intro, Streams, File I/O, Multithreading overview |
| Troubleshooting | Debugging Tools, Error Messages and Stack Traces, Breakpoints and Code Stepping, Logging for Debugging, Common Bug Patterns, Debugging Strategies, Hands-on Debugging Practice |

Multi-Threading:

Task 1:

What is a thread and a Process?

**Thread:** A lightweight unit of execution within a process that shares memory and resources with other threads in the same process. Multiple threads can run concurrently within a single process.

**Process:** An independent program in execution with its own memory space, resources, and system state. Processes are isolated from each other and communicate through inter-process communication mechanisms.

Key difference: Threads share memory within a process, while processes have separate memory spaces and are more resource-intensive to create and manage.

Task 2:

Understand the below code and run it to see the output.. Need to explain…

class RunnableDemo implements Runnable {

private Thread t;

private String threadName;

RunnableDemo( String name){

threadName = name;

System.out.println("Creating " + threadName );

}

public void run() {

System.out.println("Running " + threadName );

try {

for(int i = 4; i > 0; i--) {

System.out.println("Thread: " + threadName + ", " + i);

// Let the thread sleep for a while.

Thread.sleep(50);

}

} catch (InterruptedException e) {

System.out.println("Thread " + threadName + " interrupted.");

}

System.out.println("Thread " + threadName + " exiting.");

}

public void start ()

{

System.out.println("Starting " + threadName );

if (t == null)

{

t = new Thread (this, threadName);

t.start ();

}

}

}

public class TestThread {

public static void main(String args[]) {

RunnableDemo R1 = new RunnableDemo( "Thread-1");

R1.start();

RunnableDemo R2 = new RunnableDemo( "Thread-2");

R2.start();

}

}

Task 3

In the above code … try extending Thread class… and observe the output..

Understand:

ex:

Public Class1 extends Thread{   }

Or

4public Class1 extends  Class2 implements Runnable (     )

class ThreadDemo extends Thread {

private String threadName;

ThreadDemo(String name) {

threadName = name;

System.out.println("Creating " + threadName);

}

public void run() {

System.out.println("Running " + threadName);

try {

for(int i = 4; i > 0; i--) {

System.out.println("Thread: " + threadName + ", " + i);

Thread.sleep(50);

}

} catch (InterruptedException e) {

System.out.println("Thread " + threadName + " interrupted.");

}

System.out.println("Thread " + threadName + " exiting.");

}

}

public class TestThread {

public static void main(String args[]) {

ThreadDemo T1 = new ThreadDemo("Thread-1");

T1.start(); // Direct call to start()

ThreadDemo T2 = new ThreadDemo("Thread-2");

T2.start();

}

}

Task 4:

class Counter {

    private int count = 0;

    public void increment() {

        count++;

    }

    public int getCount() {

        return count;

    }

}

class ThreadDemo extends Thread {

    Counter counter;

    ThreadDemo(Counter counter) {

        this.counter = counter;

    }

    public void run() {

        for (int i = 0; i < 10; i++) {

            counter.increment();

        }

    }

}

public class Main {

    public static void main(String[] args) {

        Counter counter = new Counter();

        ThreadDemo t1 = new ThreadDemo(counter);

        ThreadDemo t2 = new ThreadDemo(counter);

        t1.start();

        t2.start();

        try {

            t1.join();

            t2.join();

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

        System.out.println("Final count: " + counter.getCount());

    }

}

Task 5:

Use synchronized method:

Hint:

1. Synchronized Method:  
Synchronize the entire method to ensure only one thread can execute it at a time.

class Counter {  
    private int count = 0;  
  
    public synchronized void increment() {  
        count++;  
    }  
  
    public int getCount() {  
        return count;  
    }  
}

Task 6:

Using Sync Block

Hint:

2. Synchronized Block:  
Synchronize a block of code instead of the entire method, providing more control and efficiency.

class Counter {  
    private int count = 0;  
  
    public void increment() {  
        synchronized (this) {  
            count++;  
        }  
    }  
  
    public int getCount() {  
        return count;  
    }  
}

Task 7:

Using Static Sync

Hint:

3. Static Synchronization:  
Synchronize static methods to ensure only one thread can execute them for the class, not the instance.

class Counter {  
    private static int count = 0;  
  
    public static synchronized void increment() {  
        count++;  
    }  
  
    public static int getCount() {  
        return count;  
    }  
}

Task 8:

Using Locks

Hint:

4. Locks:  
Use `java.util.concurrent.locks.Lock` for more sophisticated thread synchronization.

import java.util.concurrent.locks.Lock;  
import java.util.concurrent.locks.ReentrantLock;  
  
class Counter {  
    private int count = 0;  
    private final Lock lock = new ReentrantLock();  
  
    public void increment() {  
        lock.lock();  
        try {  
            count++;  
        } finally {  
            lock.unlock();  
        }  
    }  
  
    public int getCount() {  
        return count;  
    }  
}

Task 10:

Dead Lock 👍

Example of Deadlock

class Resource {  
    synchronized void method1(Resource r) {  
        System.out.println(Thread.currentThread().getName() + " is executing method1");  
        try { Thread.sleep(100); } catch (InterruptedException e) {}  
        r.method2(this);  
    }  
  
    synchronized void method2(Resource r) {  
        System.out.println(Thread.currentThread().getName() + " is executing method2");  
        try { Thread.sleep(100); } catch (InterruptedException e) {}  
        r.method1(this);  
    }  
}  
  
public class DeadlockExample {  
    public static void main(String[] args) {  
        final Resource r1 = new Resource();  
        final Resource r2 = new Resource();  
  
        Thread t1 = new Thread(() -> r1.method1(r2), "Thread-1");  
        Thread t2 = new Thread(() -> r2.method1(r1), "Thread-2");  
  
        t1.start();  
        t2.start();  
    }  
}

Task 11:

Inter- thread communication…

Example of Inter-thread Communication

class SharedResource {  
    private boolean ready = false;  
  
    synchronized void produce() {  
        try {  
            while (ready) {  
                wait();  
            }  
            System.out.println("Producing...");  
            ready = true;  
            notify();  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
    }  
  
    synchronized void consume() {  
        try {  
            while (!ready) {  
                wait();  
            }  
            System.out.println("Consuming...");  
            ready = false;  
            notify();  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
    }  
}  
  
public class InterThreadCommunicationExample {  
    public static void main(String[] args) {  
        SharedResource resource = new SharedResource();  
  
        Thread producer = new Thread(resource::produce);  
        Thread consumer = new Thread(resource::consume);  
  
        producer.start();  
        consumer.start();  
    }  
}

Task 12:

Interrupting a thread

Example of Interrupting a Thread

class InterruptibleThread extends Thread {  
    public void run() {  
        try {  
            while (!Thread.currentThread().isInterrupted()) {  
                System.out.println("Thread is running");  
                Thread.sleep(100);  
            }  
        } catch (InterruptedException e) {  
            System.out.println("Thread was interrupted");  
        }  
    }  
}  
  
public class InterruptExample {  
    public static void main(String[] args) {  
        InterruptibleThread thread = new InterruptibleThread();  
        thread.start();  
  
        try {  
            Thread.sleep(500);  
            thread.interrupt();  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
    }  
}

Task 14:

What is Thread pool?

**Thread Pool:** A collection of pre-created threads that are reused to execute multiple tasks, rather than creating new threads for each task.

**Benefits:** Reduces overhead of thread creation/destruction, controls resource usage by limiting concurrent threads, and improves performance through thread reuse.

**How it works:** Tasks are submitted to a queue, and available threads from the pool pick up tasks to execute. Once a task completes, the thread returns to the pool for the next task.

========================================================================

File handling:

========================================================================

Task 15:

Run the below code

public class WriteByte

{

public static void main(String args[])

{

File f1=new File(“FileName01.txt”); \\ to create new file FileOutputStream outfile = null;

byte Text[] = {'I',’ ‘,’'L','O','V','E',’ ‘,'I','N','D','I’,’A'};

try

{

outfile = new FileOutputStream(f1);

outfile.write(Text);

}

catch(IOException e)

{

System.out.println(e);

System.exit(-1);

}

System.out.println("Write Byte");

System.out.println("Thank You...!!!");

}

}

Task 16:

Try this code to see the output …

**Write a program which reads byte from file.**

import java.io.\*;

public class ReadingByte

{

public static void main(String args[])

{

FileInputStream infile = null;

int b;

try

{

infile = new FileInputStream("FileName01.txt");

while((b = infile.read()) != -1)

{

System.out.println((char)b);

}

infile.close();

}

catch(IOException e)

{

System.out.println("Sorry..!! File Not Found...!!!");

}

}

}

Task 17:

Create  a file and see the output…

import java.io.\*;

import java.util.\*;

public class WriteByte\_1

{

public static void main(String args[]) {

FileOutputStream outfile = null;

//String s=args[0]; // to input string from command line Scanner sc=new Scanner(System.in);

String s=sc.nextLine();

byte b1[] = s.getBytes();

try

{

outfile = new FileOutputStream("FileName02.txt");

outfile.write(b1);

}

catch(IOException e)

{

System.out.println(e);

System.exit(-1);

}

System.out.println("Write Byte");

System.out.println("Thank You...!!!");

}

}