Day 10 - 20th June 2025

| Advance concepts | Collections Framework intro, Streams, File I/O, Multithreading overview |
| --- | --- |
| Trobuleshooting | 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 Process?

A **process** is an instance of a program that is **being executed**. It includes the program code and its current activity

Processes are basically the programs that are dispatched from the ready state and are scheduled in the CPU for execution. PCB ( Process Control Block ) holds the context of process. A process can create other processes which are known as Child Processes. The process takes more time to terminate, and it is isolated means it does not share the memory with any other process. The process can have the following states new, ready, running, waiting, terminated and suspended.

### **Example:**

When you open a text editor like Notepad or run a Python script, the operating system creates a **process** for that program to execute.

What is a Thread?

A **thread** is the **smallest unit of execution** within a process.

Lightweight unit within a process and fast

Shares memory with other threads

Example Multiple tabs in Chrome

Task 3:

class task003 implements Runnable {

private Thread t;

private String threadName;

task003( 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*(5000);

}

} 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[]) {

task003 R1 = new task003( "Thread-1");

R1.start();

task003 R2 = new task003( "Thread-2");

R2.start();

}

}

Output



class Customer extends ParentClass ,Thread ====> not possible

class Customer extends ParentClass implement Runnable ====> possible

## Difference between Thread and Runnable

The following table provides differences between Thread and Runnable:

| **Sr. No.** | **Thread** | **Runnable** |
| --- | --- | --- |
| 1 | Thread is a class. It is used to create a thread | Runnable is a functional interface which is used to create a thread |
| 2 | It has multiple methods including start() and run() | It has only abstract method run() |
| 3 | Each thread creates a unique object and gets associated with it | Multiple threads share the same objects. |
| 4 | More memory required | Less memory required |
| 5 | Multiple Inheritance is not allowed in java hence after a class extends Thread class, it can not extend any other class | If a class is implementing the runnable interface then your class can extend another class. |

Task 4

//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 task013 {

public static void main(String[] args) {

InterruptibleThread thread = new InterruptibleThread();

thread.start();

try {

Thread.*sleep*(500);

thread.interrupt();

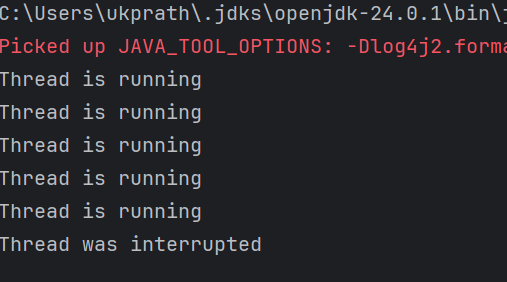
} catch (InterruptedException e) {

e.printStackTrace();

}

}

}



Task 5

class Counter {

private int count = 0;

public void increment() {

count++;

}

public int getCount() {

return count;

}

}

class ThreadDemo2 extends Thread {

Counter counter;

ThreadDemo2(Counter counter) {

this.counter = counter;

}

public void run() {

for (int i = 0; i < 20; i++) {

counter.increment();

System.*out*.println(i);

}

}

}

public class task005 {

public static void main(String[] args) {

Counter counter = new Counter();

ThreadDemo2 t1 = new ThreadDemo2(counter);

ThreadDemo2 t2 = new ThreadDemo2(counter);

t1.start();

t2.start();

try {

t1.join();

t2.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

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

}

}

Task 6

class Counter1 {

private int count = 0;

public synchronized void increment() {

count++;

}

public int getCount() {

return count;

}

}

class ThreadDemo3 extends Thread {

Counter1 counter;

ThreadDemo3(Counter1 counter) {

this.counter = counter;

}

public void run() {

for (int i = 0; i <20; i++) {

counter.increment();

}

}

}

public class task006 {

public static void main(String[] args) {

Counter1 counter = new Counter1();

ThreadDemo3 t1 = new ThreadDemo3(counter);

ThreadDemo3 t2 = new ThreadDemo3(counter);

t1.start();

t2.start();

try {

t1.join();

t2.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

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

}

}

Task 7

class Counter2 {

private int count = 0;

public void increment() {

synchronized (this) {

count++;

}

}

public int getCount() {

return count;

}

}

class ThreadDemo4 extends Thread {

Counter2 counter;

ThreadDemo4(Counter2 counter) {

this.counter = counter;

}

public void run() {

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

counter.increment();

}

}

}

public class task007 {

public static void main(String[] args) {

Counter2 counter = new Counter2();

ThreadDemo4 t1 = new ThreadDemo4(counter);

ThreadDemo4 t2 = new ThreadDemo4(counter);

t1.start();

t2.start();

try {

t1.join();

t2.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

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

}

}

Task 8

class Counter3 {

private static int *count* = 0;

public static synchronized void increment() {

*count*++;

}

public static int getCount() {

return *count*;

}

}

class ThreadDemo5 extends Thread {

Counter3 counter;

ThreadDemo5(Counter3 counter) {

this.counter = counter;

}

public void run() {

for (int i = 0; i <60; i++) {

counter.*increment*();

}

}

}

public class task008 {

public static void main(String[] args) {

Counter3 counter = new Counter3();

ThreadDemo5 t1 = new ThreadDemo5(counter);

ThreadDemo5 t2 = new ThreadDemo5(counter);

t1.start();

t2.start();

try {

t1.join();

t2.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

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

}

}

Task 9

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

// Shared Counter class

class Counter4 {

private int count = 0;

private final Lock lock = new ReentrantLock();

public void increment() {

lock.lock(); // Acquire lock

try {

count++;

} finally {

lock.unlock(); // Always release lock

}

}

public int getCount() {

return count;

}

}

// Thread class that increments the counter

class ThreadDemo6 extends Thread {

private final Counter4 counter;

ThreadDemo6(Counter4 counter) {

this.counter = counter;

}

public void run() {

for (int i = 0; i < 20; i++) {

counter.increment();

}

}

}

// Main class

public class task009 {

public static void main(String[] args) {

Counter4 counter = new Counter4();

ThreadDemo6 t1 = new ThreadDemo6(counter);

ThreadDemo6 t2 = new ThreadDemo6(counter);

t1.start();

t2.start();

try {

t1.join(); // Wait for t1 to finish

t2.join(); // Wait for t2 to finish

} catch (InterruptedException e) {

e.printStackTrace();

}

// Expected output: 40

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

}

}

Task 10

**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);

System.*out*.println("End of method1");

}

synchronized void method2(Resource r) {

System.*out*.println(Thread.*currentThread*().getName() + " is executing method2");

try { Thread.*sleep*(100); } catch (InterruptedException e) {}

r.method1(this);

System.*out*.println("End of method2");

}

}

public class task010 {

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…

//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 task011 {

public static void main(String[] args) {

SharedResource resource = new SharedResource();

Thread producer = new Thread(resource::produce);

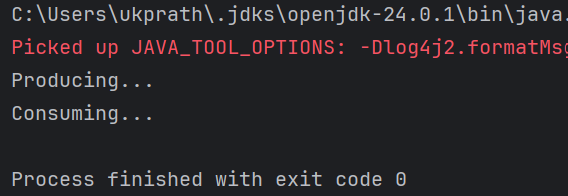
Thread consumer = new Thread(resource::consume);

producer.start();

consumer.start();

}

}



Task12

import java.util.stream.\*;

class task012 {

public static void main(String[] args) {

Stream<String> stream

= Stream.*of*("Hello", " My",

" name", " is",

" Pratheesh",

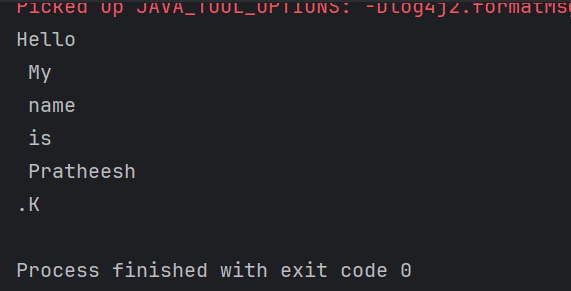
".K");

stream.forEach(System.*out*::println);

//System.out.println("println method");

}

}



Task 13

**Example of Interrupting a Thread**

//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 task013 {

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 are Daemon threads? Explain…

Daemon threa is a background thread that runs in the background to perform tasks like garbage collection,handling background I/O operations without blocking main program from exiting

Example : Auto-Save Feature in a Text Editor

=========================Trouble shooting ========================

Task 15:

What are the debugging tools in Java.. list down a few..

### **Common Debugging Tools in Java**

1. **Java Debugger (JDB)** –  
    A command-line debugger provided with the JDK. It allows setting breakpoints, inspecting variables, and controlling program execution.
2. **Eclipse Debugger** –  
    Built into the Eclipse IDE. Provides graphical debugging features such as breakpoints, step execution, and variable inspection.
3. **IntelliJ IDEA Debugger** –  
    A powerful debugger integrated into IntelliJ IDEA. Supports conditional breakpoints, watches, and evaluating expressions at runtime.
4. **NetBeans Debugger** –  
    Part of NetBeans IDE. Provides similar debugging features like step-in, step-out, watches, and multi-thread debugging.
5. **VisualVM** –  
    A monitoring, profiling, and debugging tool included with the JDK. Useful for analyzing heap dumps, thread states, and memory leaks.
6. **JConsole** –  
    A monitoring and debugging tool for Java applications using JMX (Java Management Extensions). Good for analyzing performance issues.
7. **JProfiler** –  
    A commercial tool for profiling and debugging Java applications. Helps track memory leaks, thread problems, and CPU usage.
8. **YourKit** –  
    Another commercial profiler that provides advanced debugging, memory analysis, and performance monitoring.

Task 16:

Try to understand the error Messages.. What are they and when to you?

Few are given below.

Error Messages in Java

Compile time and run time

Compile time : grammatical mistakes … ;, {} , missing the code

Run time error or exceptions

Stack overflow error

Array index out of bounds

IO exception

Nulpointer exception

### **Error Messages in Java**

In Java, errors can broadly be categorized into **compile-time errors** and **run-time errors (exceptions)**.

#### **1. Compile-time Errors**

* **When they occur:** During the compilation of Java source code into bytecode (before the program runs).
* **Reason:** Violations of Java’s syntax or rules.
* **Examples:**
  + **Missing semicolon (;)**
  + **Unmatched brackets ({ }, ( ))**
  + **Using undeclared variables**
  + **Incorrect method signatures**

💡 *These must be fixed before the program can run.*

#### **2. Run-time Errors (Exceptions & Errors)**

* **When they occur:** While the program is running.
* **Reason:** Logical mistakes, invalid operations, or resource issues.
* **Examples:**

1. **StackOverflowError**
   * Happens when a method keeps calling itself recursively without a stopping condition.
   * Example: infinite recursion.
2. **ArrayIndexOutOfBoundsException**
   * When trying to access an array index that doesn’t exist.
   * Example: arr[10] when arr length is 5.
3. **IOException**
   * Input/Output related problems (e.g., file not found, network issues).
4. **NullPointerException**
   * When trying to access methods/variables on an object that is null.
   * Example: str.length() when str = null.

Task 17:

What is Stack trace.. What will it do?

Understand the below points..

Identify the error

Locate the code

Analyze the code

Solution also

### **What is a Stack Trace in Java?**

* A **stack trace** is a report generated when an **exception or error** occurs in Java.
* It shows the **sequence of method calls** (the call stack) that led to the error.
* It helps developers quickly **debug** by identifying *what went wrong, where it went wrong, and why*.

### **🔎 What does a Stack Trace do?**

1. **Identify the error**
   * The stack trace always begins with the **exception type and message**.

Example:  
  
 Exception in thread "main" java.lang.NullPointerException

* + 👉 This tells you the program tried to use a null reference.

1. **Locate the code**
   * The stack trace shows the **exact line number and class** where the error occurred.

Example:  
  
 at com.example.MainClass.main(MainClass.java:15)

* + 👉 Error happened in MainClass.java at line 15.

1. **Analyze the code**
   * By checking the method calls listed, you can **trace back** how the program reached the point of failure.
   * It helps you see if incorrect data, wrong logic, or unexpected input caused the issue.
2. **Solution**
   * Once you know the **type of error** and the **location**, you can fix it:  
     + For **NullPointerException** → ensure object is initialized before use.
     + For **ArrayIndexOutOfBoundsException** → check array length before accessing.
     + For **IOException** → ensure file path or resource exists.

✅ **In short:** A **stack trace = map of the error path**. It tells you *what exception happened, where it happened, and how the program got there*, helping you solve the problem faster.

Task18:

public class task18 extends Thread {

public void run(){

System.*out*.println("thread started.");

}

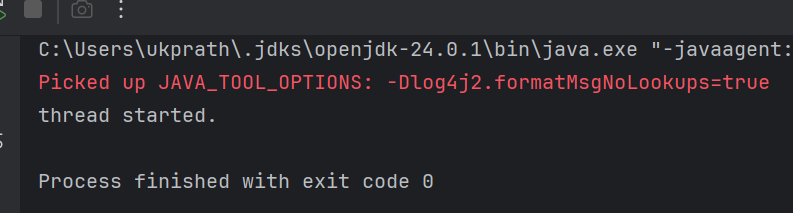
public static void main(String args[]){

task18 th1 = new task18();

th1.start();

}

}



Task 19:

class MyRunnable implements Runnable {

@Override

public void run() {

System.*out*.println("Code executed in a new thread via Runnable.");

}

}

class MyThread extends Thread {

@Override

public void run() {

System.*out*.println("Code executed in a new thread via Thread extension.");

}

}

class task19 {

public static void main(String[] args) {

MyRunnable runnableInstance = new MyRunnable();

MyThread threadInstance = new MyThread();

// Creating a Thread object using Runnable

Thread t1 = new Thread(runnableInstance);

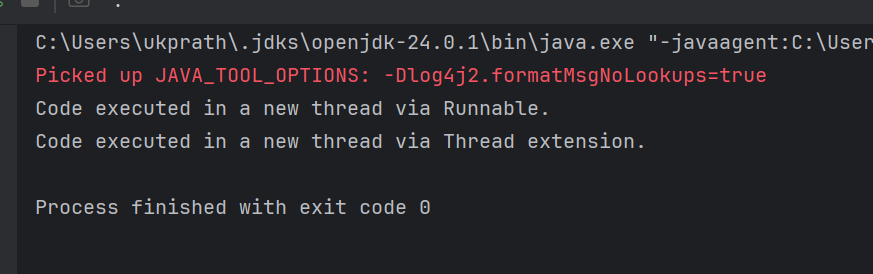
// Starting both threads

t1.start(); // Runs MyRunnable's run()

threadInstance.start(); // Runs MyThread's run()

}

}



***why not we directly call run() method? And why do we use start() to call the run internally… Hope this helps you Hemanth..***

The run() method is just an ordinary method (overridden by *you*). As with any other ordinary method and calling it directly will cause the *current thread* to execute run()

Inside start() method will cause the JVM to spawn a new thread and make the newly spawned thread execute run()..

Even if programmatically we are not creating any thread, For every application, O.S will create a default thread to execute its code with CPU.

Calling run method directly will make that run method execute in that main thread given by O.S.

But the intention of creating a thread class is to make sure that run method executes in a different thread. Unless thread manager of O.S / JVM creates a thread, your run method will not get executed in a separate thread. To request O.S/ JVM to create the separate thread you have to call start() method which will send a request to O.S / JVM to create a thread. Once O.S / JVM creates a thread, then O.S / JVM will automatically call run method of your thread class in that newly created thread context. And hence your purpose of creating a separate thread and executing your run method in a separate thread will be served.

If you call run method directly, then it is like O.S is not creating any thread for you, and default main thread will execute your run method. No point of creating a separate thread class for that!

Hope I am clear. Let me know if you need more explanation to answer your question.

JVM creates threads, internally JVM will have to send a request to thread manager driver of O.S layer to create a new thread in its thred pool.

So gist of the story is

If we want, we can call run() method, but if we call run method it will run as just a normal Java method. Whereas if we call strat() it JVM creates a new thread and run method will be executed on that thread. — where multi threading comes in to picture..

Task 20:

public class ThreadTraceExample {

public static void main(String[] args) {

*method1*();

}

public static void method1() {

*method2*();

}

public static void method2() {

*method3*();

}

public static void method3() {

StackTraceElement[] stackTrace = Thread.*currentThread*().getStackTrace();

System.*out*.println("Thread Stack Trace:");

for (StackTraceElement element : stackTrace) {

System.*out*.println(" Class: " + element.getClassName() +

", Method: " + element.getMethodName() +

", Line: " + element.getLineNumber());

}

}

}

