Task 1:

What is a Process?

A process is simply a program that is currently running on a computer. It’s like an active task that the computer executing.

* Opening an app
* Printing a document
* Downloading a file

Each process has its own memory space, multiple processes can run at same time, Each process is independent of others. When you close an application, its process ends.

Task 2:

What is a Thread?

A thread is a smaller unit of a process that can execute tasks independently. Think of it as a way to split a process into smaller tasks that can run at the same time.

For example, MS Word

* One thread checks your spelling
* Another thread auto-saves your document
* A third updates the screen as you type

Threads share the same memory space within a process. They are lighter than processes. Multiple threads can run parallel. Threads within the same process can communicate easily

Task 3:

public class task03 {

    static 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);

                    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 static void main(String args[]) {

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

        R1.start();

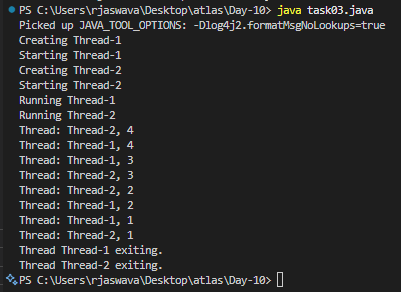
        RunnableDemo R2 = new RunnableDemo("Thread-2");

        R2.start();

    }

}

Output



Task 4

Extending Thread class

public class task04 {

    static class ThreadDemo extends Thread {

        private Thread t;

        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 void start() {

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

            if (t == null) {

                t = new Thread(this, threadName);

                t.start();

            }

        }

    }

    public static void main(String args[]) {

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

        T1.start();

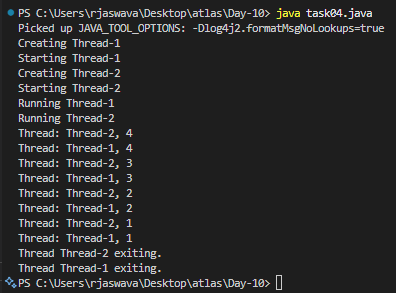
        ThreadDemo T2 = new ThreadDemo("Thread-2");

        T2.start();

    }

}

Output



Task 05:

public class task05 {

    static class Counter {

        private int count = 0;

        public void increment() {

            count++;

        }

        public int getCount() {

            return count;

        }

    }

    static 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 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();

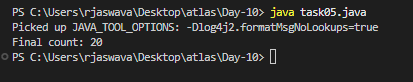
        }

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

    }

}

Output



Task 06:

public class task06 {

    static class Counter {

        private int count = 0;

        public synchronized void increment() {

            count++;

            System.out.println(Thread.currentThread().getName() + " incremented count to: " + count);

        }

        public int getCount() {

            return count;

        }

    }

    static class ThreadDemo extends Thread {

        Counter counter;

        ThreadDemo(Counter counter, String name) {

            super(name);

            this.counter = counter;

        }

        public void run() {

            for (int i = 0; i < 5; i++) {

                counter.increment();

                try {

                    Thread.sleep(100);

                } catch (InterruptedException e) {

                    e.printStackTrace();

                }

            }

        }

    }

    public static void main(String[] args) {

        Counter counter = new Counter();

        ThreadDemo t1 = new ThreadDemo(counter, "Thread-1");

        ThreadDemo t2 = new ThreadDemo(counter, "Thread-2");

        t1.start();

        t2.start();

        try {

            t1.join();

            t2.join();

        } catch (InterruptedException e) {

            e.printStackTrace();

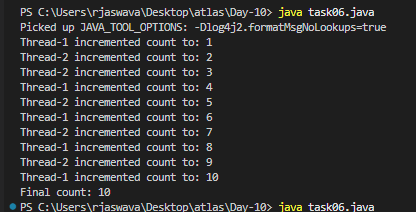
        }

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

    }

}

Output



Task 07:

public class task07 {

    static class Counter {

        private int count = 0;

        public void increment() {

            synchronized (this) {

                count++;

                System.out.println(Thread.currentThread().getName() +

                    " is incrementing. Count is now: " + count);

                try {

                    Thread.sleep(100);

                } catch (InterruptedException e) {

                    e.printStackTrace();

                }

            }

            System.out.println(Thread.currentThread().getName() +

                " finished processing increment operation.");

        }

        public int getCount() {

            return count;

        }

    }

    static class ThreadDemo extends Thread {

        Counter counter;

        ThreadDemo(Counter counter, String name) {

            super(name);

            this.counter = counter;

        }

        public void run() {

            System.out.println(Thread.currentThread().getName() + " started");

            for (int i = 0; i < 5; i++) {

                counter.increment();

            }

            System.out.println(Thread.currentThread().getName() + " finished");

        }

    }

    public static void main(String[] args) {

        Counter counter = new Counter();

        ThreadDemo t1 = new ThreadDemo(counter, "Thread-1");

        ThreadDemo t2 = new ThreadDemo(counter, "Thread-2");

        t1.start();

        t2.start();

        try {

            t1.join();

            t2.join();

        } catch (InterruptedException e) {

            e.printStackTrace();

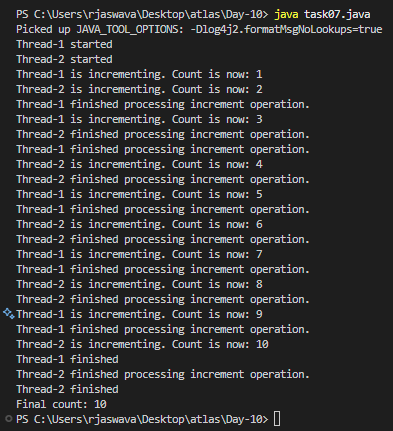
        }

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

    }

}

Output



Task 08:

public class task08 {

    static class Counter {

        private static int count = 0;

        public static synchronized void increment() {

            count++;

            System.out.println(Thread.currentThread().getName() +

                " incremented static count to: " + count);

            try {

                Thread.sleep(100);

            } catch (InterruptedException e) {

                e.printStackTrace();

            }

        }

        public static synchronized int getCount() {

            return count;

        }

    }

    static class ThreadDemo extends Thread {

        ThreadDemo(String name) {

            super(name);

        }

        public void run() {

            System.out.println(Thread.currentThread().getName() + " started");

            Counter c1 = new Counter();

            Counter c2 = new Counter();

            for (int i = 0; i < 5; i++) {

                if (i % 2 == 0) {

                    c1.increment();

                } else {

                    c2.increment();

                }

            }

            System.out.println(Thread.currentThread().getName() + " finished");

        }

    }

    public static void main(String[] args) {

        System.out.println("Starting main thread");

        ThreadDemo t1 = new ThreadDemo("Thread-1");

        ThreadDemo t2 = new ThreadDemo("Thread-2");

        t1.start();

        t2.start();

        try {

            t1.join();

            t2.join();

        } catch (InterruptedException e) {

            e.printStackTrace();

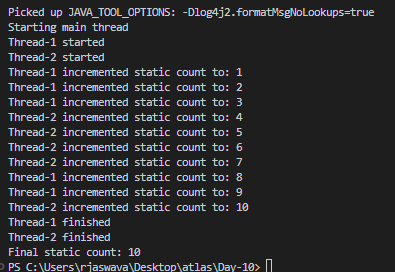
        }

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

    }

}

Output



Task 09:

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

public class task09 {

    static class Counter {

        private int count = 0;

        private final Lock lock = new ReentrantLock();

        public void increment() {

            lock.lock();

            try {

                count++;

                System.out.println(Thread.currentThread().getName() +

                    " incremented count to: " + count);

                Thread.sleep(100);

            } catch (InterruptedException e) {

                e.printStackTrace();

            } finally {

                lock.unlock();

                System.out.println(Thread.currentThread().getName() +

                    " released the lock");

            }

        }

        public int getCount() {

            return count;

        }

    }

    static class ThreadDemo extends Thread {

        Counter counter;

        ThreadDemo(Counter counter, String name) {

            super(name);

            this.counter = counter;

        }

        public void run() {

            System.out.println(Thread.currentThread().getName() + " started");

            for (int i = 0; i < 5; i++) {

                counter.increment();

                try {

                    Thread.sleep(50);

                } catch (InterruptedException e) {

                    e.printStackTrace();

                }

            }

            System.out.println(Thread.currentThread().getName() + " finished");

        }

    }

    public static void main(String[] args) {

        Counter counter = new Counter();

        ThreadDemo t1 = new ThreadDemo(counter, "Thread-1");

        ThreadDemo t2 = new ThreadDemo(counter, "Thread-2");

        t1.start();

        t2.start();

        try {

            t1.join();

            t2.join();

        } catch (InterruptedException e) {

            e.printStackTrace();

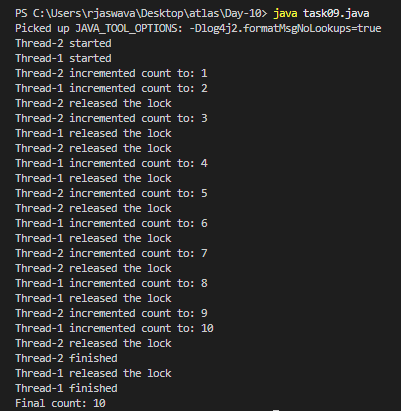
        }

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

    }

}

Output



Task 10:

public class task10 {

    static class Resource {

        private String name;

        public Resource(String name) {

            this.name = name;

        }

        synchronized void method1(Resource r) {

            System.out.println(Thread.currentThread().getName() +

                " acquired lock on " + this.name +

                " and trying to lock " + r.name);

            try {

                Thread.sleep(100);

            } catch (InterruptedException e) {

                e.printStackTrace();

            }

            System.out.println(Thread.currentThread().getName() +

                " waiting to lock " + r.name);

            r.method2(this);

        }

        synchronized void method2(Resource r) {

            System.out.println(Thread.currentThread().getName() +

                " acquired lock on " + this.name +

                " and trying to lock " + r.name);

            try {

                Thread.sleep(100);

            } catch (InterruptedException e) {

                e.printStackTrace();

            }

            System.out.println(Thread.currentThread().getName() +

                " waiting to lock " + r.name);

            r.method1(this);

        }

    }

    public static void main(String[] args) {

        final Resource r1 = new Resource("Resource-1");

        final Resource r2 = new Resource("Resource-2");

        Thread t1 = new Thread(() -> {

            r1.method1(r2);

        }, "Thread-1");

        Thread t2 = new Thread(() -> {

            r2.method1(r1);

        }, "Thread-2");

        t1.start();

        t2.start();

        try {

            Thread.sleep(2000);

            System.out.println("\nDeadlock Detection:");

            System.out.println("Thread-1 State: " + t1.getState());

            System.out.println("Thread-2 State: " + t2.getState());

        } catch (InterruptedException e) {

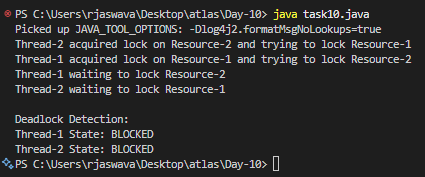
            e.printStackTrace();

        }

    }

}

Output



Task 11

public class task11 {

    static 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 static void main(String[] args) {

        SharedResource resource = new SharedResource();

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

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

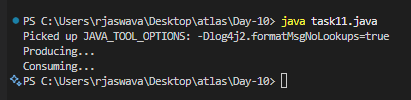
        producer.start();

        consumer.start();

    }

}

Output:



Task 12

import java.util.stream.\*;

public class task12 {

    public static void main(String[] args) {

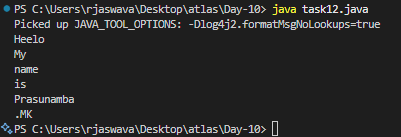
        Stream<String> stream = Stream.of("Heelo", "My", "name", "is", "Prasunamba", ".MK");

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

    }

}

Output:



Task 13

public class task13 {

    static 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 static void main(String[] args) {

        InterruptibleThread thread = new InterruptibleThread();

        thread.start();

        try {

            Thread.sleep(500);

            thread.interrupt();

        } catch (InterruptedException e) {

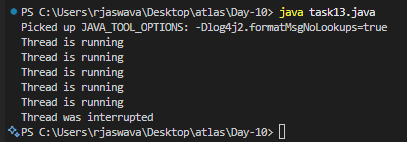
            e.printStackTrace();

        }

    }

}

Output



Task 14:

What are Daemon threads? Explain…

Daemon threads are background threads that provides support to main threads. They are low-priority threads that run in the background to perform tasks like garbage collection or providing services to main threads.

* Daemon threads are helpful background workers
* They stop when main program ends
* Shouldn’t rely on them for critical tasks

Task 15:

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

System.out.println()

- Simple print debugging

- Most basic debugging method

- Good for quick checks

Common Debugging Features:

1. Breakpoints

2. Step Over/Into/Out

3. Variable watching

4. Stack trace viewing

5. Memory monitoring

Task 16:

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

Compile time and run time

* Missing semicolon
* Missing brackets
* Undefined variables

Run time error or exceptions

* Stack overflow error

When recursive method calls to deep

* Array index out of bounds

Trying to access array index that doesn’t exist

* IO exception

Input and output operation fails

* Nulpointer exception

Trying to use an object that is 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

1. Identify the Error:

- First step is to understand what type of error occurred

- Look at the error message and exception type

2. Locate the Code:

- Find exactly where in your code the error occurred

- Stack trace shows file names and line numbers

3. Analyze the Code:

- Once located, examine what went wrong

- Check variable values

- Review logic

4. Solution:

- Fix the identified problem

- Add proper error handling

Task 18:

public class task18 {

    static class Test extends Thread {

        public void run() {

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

        }

    }

    public static void main(String args[]) {

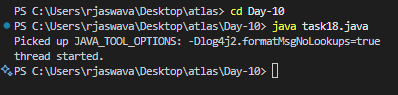
        Test t1 = new Test();

        t1.run();

    }

}

Output:



Task 19:

public class task19 {

    static class MyRunnable implements Runnable {

        @Override

        public void run() {

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

        }

    }

    static class MyThread extends Thread {

        @Override

        public void run() {

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

        }

    }

    public static void main(String[] args) {

        MyRunnable runnableInstance = new MyRunnable();

        MyThread threadInstance = new MyThread();

        Thread t1 = new Thread(runnableInstance);

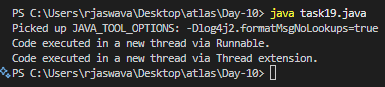
        t1.start(); // For MyRunnable

        threadInstance.start(); // For MyThread

    }

}

Output:



Task 20:

public class task20 {

    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:");

        // Iterate through the StackTraceElement array and print details

        for (StackTraceElement element : stackTrace) {

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

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

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

        }

    }

}

Output:

