**CONCURRENT PROGRAMMING PARADIGM  
THREAD CLASSES AND METHODS**

**Tutorial assignment week-6**

1. **Write a java program that implements a multi-thread application that has three threads. First thread generates random integer every second and if the value is even, second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of cube of the number.**

import java.util.Random;

public class Main {

    public static void main(String[] args) {

        RandomNumberGenerator generator = new RandomNumberGenerator();

        SquareThread squareThread = new SquareThread(generator);

        CubeThread cubeThread = new CubeThread(generator);

        Thread t1 = new Thread(generator);

        Thread t2 = new Thread(squareThread);

        Thread t3 = new Thread(cubeThread);

        t1.start();

        t2.start();

        t3.start();

        // Sleep for a while to demonstrate the threads running

        try {

            Thread.sleep(10000); // Sleep for 10 seconds

        } catch (InterruptedException e) {

            Thread.currentThread().interrupt();

        }

        // Stop the generator and threads gracefully

        generator.stop();

        squareThread.stopThread();

        cubeThread.stopThread();

    }

}

class RandomNumberGenerator implements Runnable {

    private volatile int number;

    private volatile boolean running = true;

    private final Object lock = new Object();

    @Override

    public void run() {

        Random random = new Random();

        while (running) {

            int newNumber = random.nextInt(100); // Generate a random integer between 0 and 99

            System.out.println("Generated: " + newNumber);

            synchronized (lock) {

                number = newNumber; // Update the number variable atomically

            }

            try {

                Thread.sleep(1000); // Sleep for 1 second

            } catch (InterruptedException e) {

                Thread.currentThread().interrupt();

            }

        }

    }

    public void stop() {

        running = false;

    }

    public int getNumber() {

        return number;

    }

}

class SquareThread implements Runnable {

    private RandomNumberGenerator generator;

    private volatile boolean running = true;

    public SquareThread(RandomNumberGenerator generator) {

        this.generator = generator;

    }

    @Override

    public void run() {

        while (running) {

            int num = generator.getNumber();

            int square = num \* num;

            System.out.println("Squared: " + square);

            try {

                Thread.sleep(1000); // Sleep for 1 second

            } catch (InterruptedException e) {

                Thread.currentThread().interrupt();

            }

        }

    }

    public void stopThread() {

        running = false;

    }

}

class CubeThread implements Runnable {

    private RandomNumberGenerator generator;

    private volatile boolean running = true;

    public CubeThread(RandomNumberGenerator generator) {

        this.generator = generator;

    }

    @Override

    public void run() {

        while (running) {

            int num = generator.getNumber();

            int cube = num \* num \* num;

            System.out.println("Cubed: " + cube);

            try {

                Thread.sleep(1000); // Sleep for 1 second

            } catch (InterruptedException e) {

                Thread.currentThread().interrupt();

            }

        }

    }

    public void stopThread() {

        running = false;

    }

}

*Output:*

Generated: 16

Squared: 256

Cubed: 4096

Generated: 33

Squared: 1089

Cubed: 35937

Generated: 31

Squared: 961

Cubed: 29791

Generated: 87

Squared: 7569

Cubed: 658503

Generated: 47

Squared: 2209

Cubed: 103823

Generated: 74

Squared: 5476

Cubed: 405224

Generated: 63

Squared: 3969

Cubed: 250047

Generated: 88

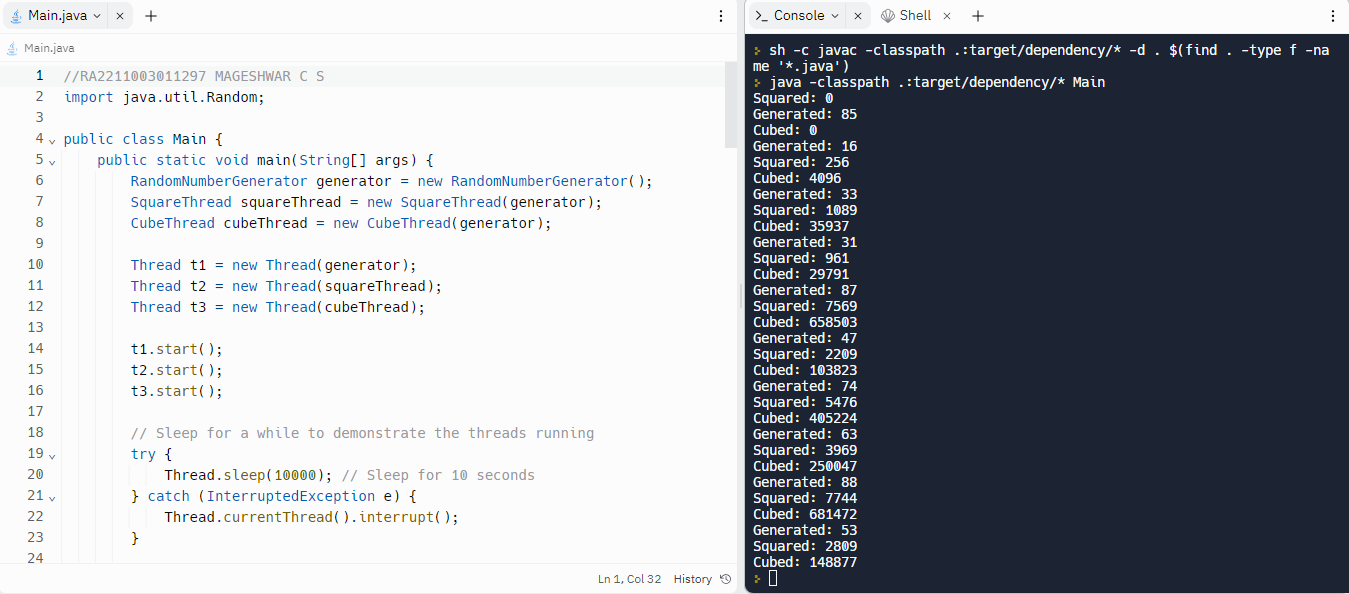
Squared: 7744

Cubed: 681472

Generated: 53

Squared: 2809

Cubed: 148877



1. **Write a java program for to solve producer consumer problem in which a producer produce a value and consumer consume the value before producer generate the next value.**

import java.util.concurrent.ArrayBlockingQueue;

import java.util.concurrent.BlockingQueue;

public class ProducerConsumerExample {

    public static void main(String[] args) {

        BlockingQueue<Integer> buffer = new ArrayBlockingQueue<>(1); // Capacity of 1

        Thread producerThread = new Thread(new Producer(buffer));

        Thread consumerThread = new Thread(new Consumer(buffer));

        producerThread.start();

        consumerThread.start();

    }

}

class Producer implements Runnable {

    private final BlockingQueue<Integer> buffer;

    public Producer(BlockingQueue<Integer> buffer) {

        this.buffer = buffer;

    }

    @Override

    public void run() {

        try {

            for (int i = 1; i <= 10; i++) {

                Integer value = i;

                System.out.println("Producing: " + value);

                buffer.put(value); // Put the value into the buffer

                value++; // Generate the next value

            }

        } catch (InterruptedException e) {

            Thread.currentThread().interrupt();

        }

    }

}

class Consumer implements Runnable {

    private final BlockingQueue<Integer> buffer;

    public Consumer(BlockingQueue<Integer> buffer) {

        this.buffer = buffer;

    }

    @Override

    public void run() {

        try {

            for (int i = 1; i <= 10; i++) {

                Integer value = buffer.take(); // Consume the value

                System.out.println("Consuming: " + value);

                Thread.sleep(1500); // Simulate some time taken to consume

            }

        } catch (InterruptedException e) {

            Thread.currentThread().interrupt();

        }

    }

}

*Output:*

Producing: 1

Producing: 2

Consuming: 1

Producing: 3

Consuming: 2

Producing: 4

Consuming: 3

Producing: 5

Consuming: 4

Producing: 6

Consuming: 5

Producing: 7

Consuming: 6

Producing: 8

Consuming: 7

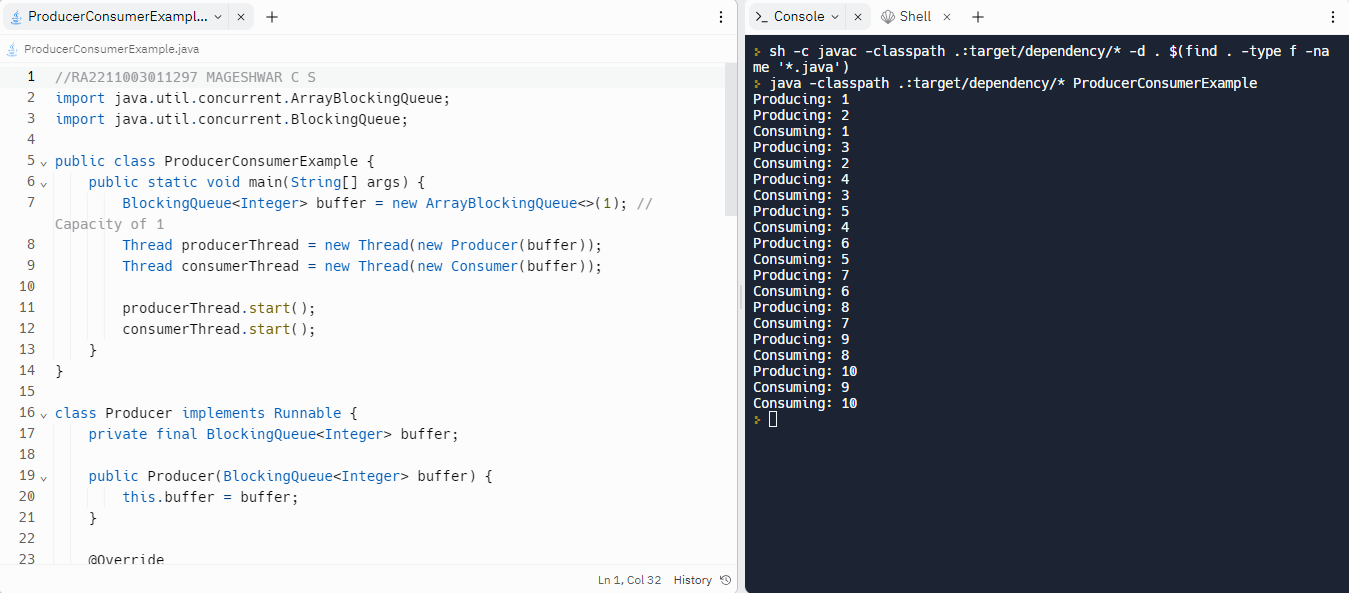
Producing: 9

Consuming: 8

Producing: 10

Consuming: 9

Consuming: 10



1. **Write a java program in which thread sleep for 5 sec and change the name of thread.**

public class SleepAndRenameThread {

    public static void main(String[] args) {

        Thread customThread = new Thread(new CustomRunnable());

        customThread.start();

    }

}

class CustomRunnable implements Runnable {

    @Override

    public void run() {

        try {

            // Sleep for 5 seconds

            Thread.sleep(5000);

        } catch (InterruptedException e) {

            Thread.currentThread().interrupt();

        }

        // Change the name of the thread

        Thread currentThread = Thread.currentThread();

        currentThread.setName("NewThreadName");

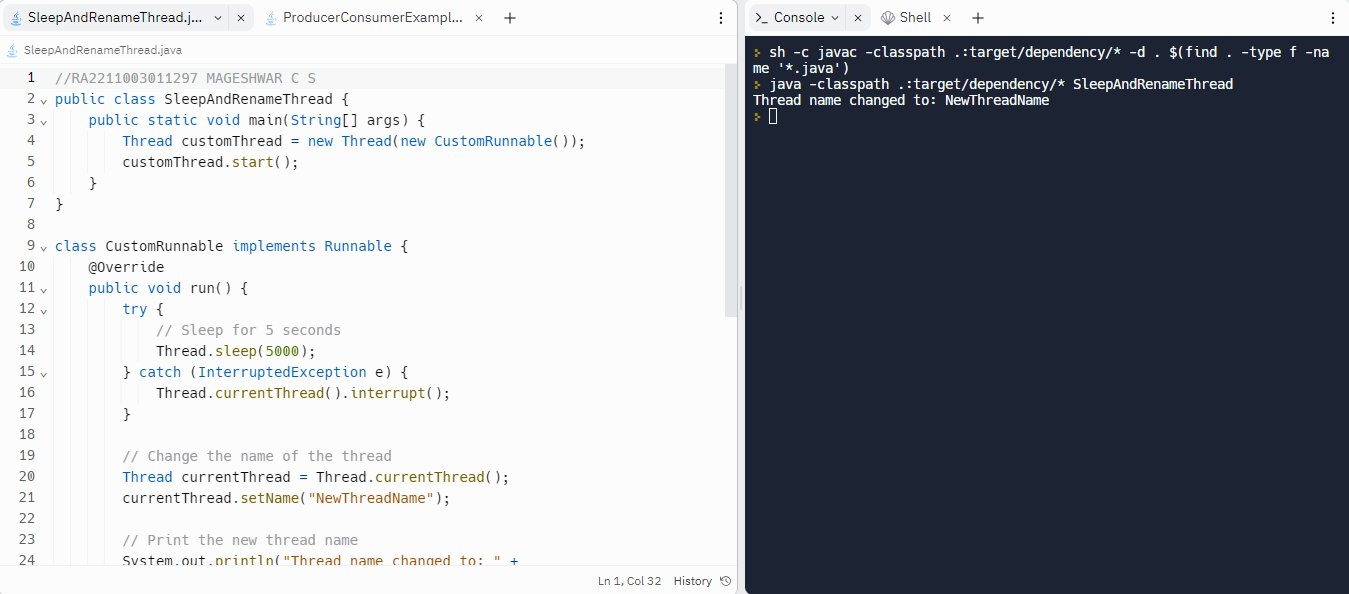
        // Print the new thread name

        System.out.println("Thread name changed to: " + currentThread.getName());

    }

}

*Output:*

Thread name changed to: NewThreadName

1. **Write a java program in which thread sleep for 6 sec in the loop in reverse order from 5 to 1 and change the name of thread.**

public class SleepAndRenameThread2 {

    public static void main(String[] args) {

        Thread customThread = new Thread(new CustomRunnable());

        customThread.start();

    }

}

class CustomRunnable implements Runnable {

    @Override

    public void run() {

        for (int i = 5; i > 0; i--) {

            try {

                // Sleep for 6 seconds

                Thread.sleep(6000);

            } catch (InterruptedException e) {

                Thread.currentThread().interrupt();

            }

            // Change the name of the thread

            Thread currentThread = Thread.currentThread();

            currentThread.setName("NewThreadName " + i);

            // Print the new thread name

            System.out.println("Thread name changed to: " + currentThread.getName());

        }

    }

}

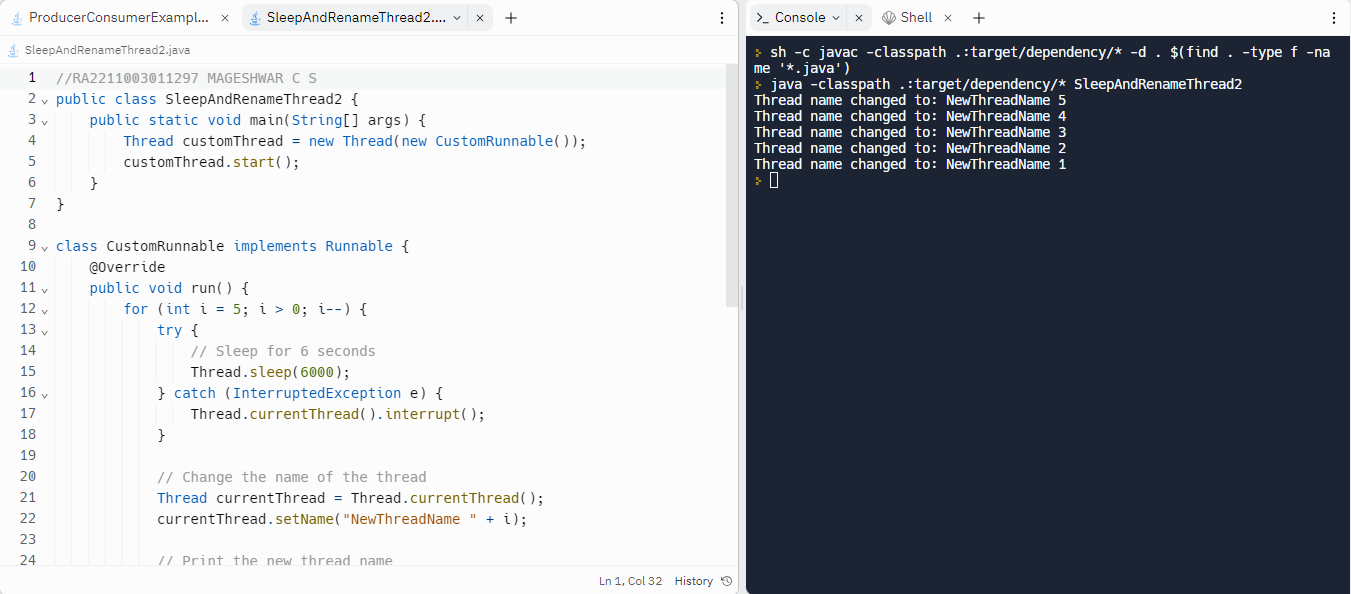
*Output:*

Thread name changed to: NewThreadName 5

Thread name changed to: NewThreadName 4

Thread name changed to: NewThreadName 3

Thread name changed to: NewThreadName 2

Thread name changed to: NewThreadName 1

1. **Write a java program for multithread in which user thread and thread started from main method invoked at a time each thread sleep for 1 sec.**

public class MultiThreadExample {

    public static void main(String[] args) {

        // Create and start a user thread

        Thread userThread = new Thread(new UserRunnable());

        userThread.start();

        // Start a thread from the main method

        Thread mainThread = new Thread(new MainRunnable());

        mainThread.start();

        // Sleep for a while to allow both threads to run

        try {

            Thread.sleep(2000); // Sleep for 2 seconds

        } catch (InterruptedException e) {

            Thread.currentThread().interrupt();

        }

    }

}

class UserRunnable implements Runnable {

    @Override

    public void run() {

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

        try {

            Thread.sleep(1000); // Sleep for 1 second

        } catch (InterruptedException e) {

            Thread.currentThread().interrupt();

        }

        System.out.println("User thread completed.");

    }

}

class MainRunnable implements Runnable {

    @Override

    public void run() {

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

        try {

            Thread.sleep(1000); // Sleep for 1 second

        } catch (InterruptedException e) {

            Thread.currentThread().interrupt();

        }

        System.out.println("Main thread completed.");

    }

}

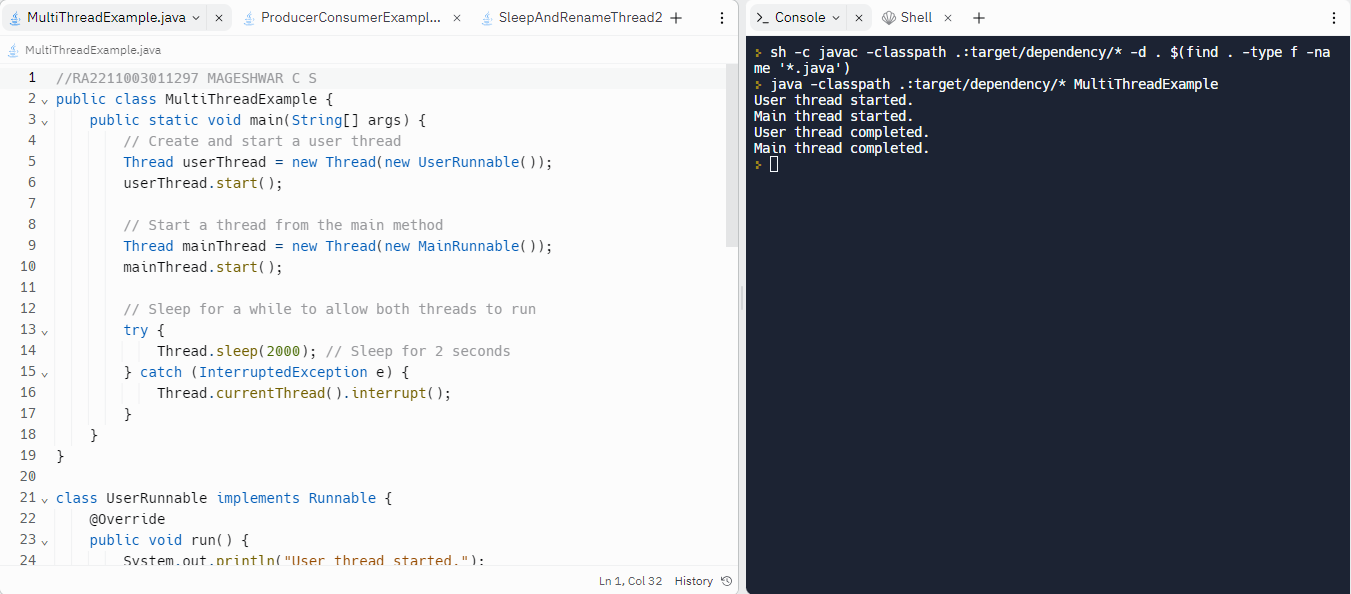
*Output:*

User thread started.

Main thread started.

User thread completed.

Main thread completed.



1. **Write a java program to solve printer synchronization problem in which all the jobs must be completed in order.**

import java.util.LinkedList;

import java.util.Queue;

public class PrinterSynchronizationDemo {

    private static Queue<String> jobs = new LinkedList<>();

    private static Object lock = new Object();

    public static void main(String[] args) {

        Thread t1 = new Thread(() -> {

            synchronized (lock) {

                jobs.add("Job 1");

                lock.notifyAll();

            }

        });

        Thread t2 = new Thread(() -> {

            synchronized (lock) {

                while (jobs.isEmpty() || !jobs.peek().equals("Job 1")) {

                    try {

                        lock.wait();

                    } catch (InterruptedException e) {

                        e.printStackTrace();

                    }

                }

                System.out.println("Printing " + jobs.poll());

                jobs.add("Job 2");

                lock.notifyAll();

            }

        });

        Thread t3 = new Thread(() -> {

            synchronized (lock) {

                while (jobs.isEmpty() || !jobs.peek().equals("Job 2")) {

                    try {

                        lock.wait();

                    } catch (InterruptedException e) {

                        e.printStackTrace();

                    }

                }

                System.out.println("Printing " + jobs.poll());

                jobs.add("Job 3");

                lock.notifyAll();

            }

        });

        Thread t4 = new Thread(() -> {

            synchronized (lock) {

                while (jobs.isEmpty() || !jobs.peek().equals("Job 3")) {

                    try {

                        lock.wait();

                    } catch (InterruptedException e) {

                        e.printStackTrace();

                    }

                }

                System.out.println("Printing " + jobs.poll());

                lock.notifyAll();

            }

        });

        t1.start();

        t2.start();

        t3.start();

        t4.start();

    }

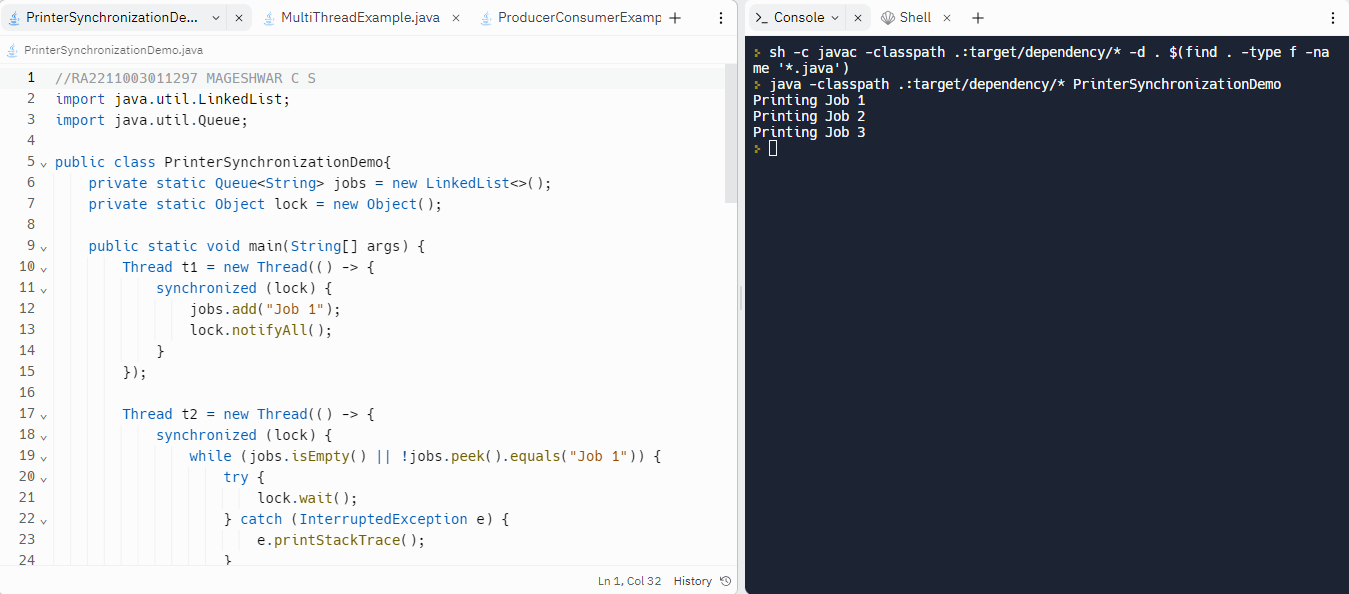
}

*Output:*

Printing Job 1

Printing Job 2

Printing Job 3



1. **Create a java program for the following**

**Use ThreadA to find number of digits present in the string k and store into variable dc, finally print the value of dc(output format: ThreadA:digitscount).**

**Use ThreadB to find number of alphabetic present in the string k and store into variable cc, finally print the value of cc(output format:ThreadB:digitscount).**

public class ThreadDemo {

    private static String k = "Hello World! 123";

    private static int dc = 0;

    private static int cc = 0;

    public static void main(String[] args) {

        Thread t1 = new Thread(() -> {

            for (int i = 0; i < k.length(); i++) {

                if (Character.isDigit(k.charAt(i))) {

                    dc++;

                }

            }

            System.out.println("ThreadA:" + dc);

        });

        Thread t2 = new Thread(() -> {

            for (int i = 0; i < k.length(); i++) {

                if (Character.isLetter(k.charAt(i))) {

                    cc++;

                }

            }

            System.out.println("ThreadB:" + cc);

        });

        t1.start();

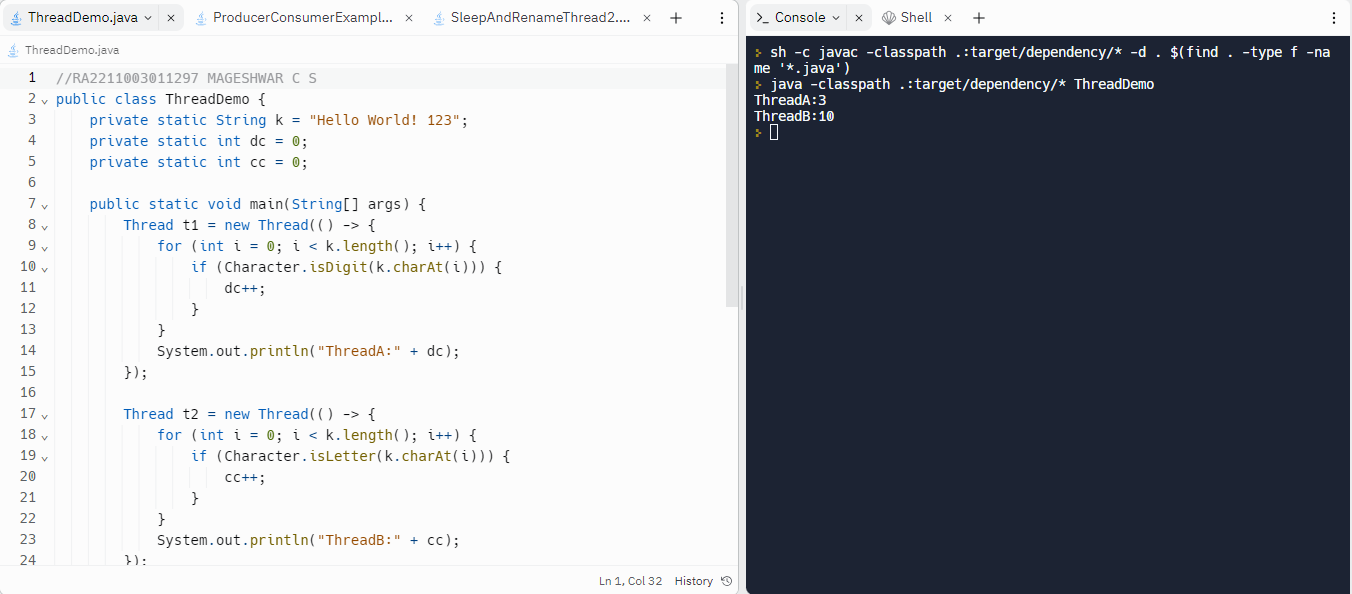
        t2.start();

    }

}

*Output:*

ThreadA:3

ThreadB:10

1. **Create two objects threadobj1 and threadobj2 for the class UserThreadPriority. Set the name of threadobj1 as “ThreadA” and threadobj2 as “ThreadB”. Get a String and a Character from the user and assign into UserThreadPriority class variable k and c respectively. Call the start() method for the thread objects threadobj1 and threadobj2.**

import java.util.Scanner;

public class UserThreadPriorityDemo {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a string: ");

        String k = scanner.nextLine();

        System.out.print("Enter a character: ");

        char c = scanner.next().charAt(0);

        scanner.close();

        UserThreadPriority userThreadPriority = new UserThreadPriority(k, c);

        Thread threadobj1 = new Thread(() -> {

            int dc = userThreadPriority.countDigits();

            System.out.println("ThreadA:" + dc);

        }, "ThreadA");

        Thread threadobj2 = new Thread(() -> {

            int cc = userThreadPriority.countChars();

            System.out.println("ThreadB:" + cc);

        }, "ThreadB");

        threadobj1.start();

        threadobj2.start();

    }

}

class UserThreadPriority {

    private String k;

    private char c;

    public UserThreadPriority(String k, char c) {

        this.k = k;

        this.c = c;

    }

    public int countDigits() {

        int dc = 0;

        for (int i = 0; i < k.length(); i++) {

            if (Character.isDigit(k.charAt(i))) {

                dc++;

            }

        }

        return dc;

    }

    public int countChars() {

        int cc = 0;

        for (int i = 0; i < k.length(); i++) {

            if (k.charAt(i) == c) {

                cc++;

            }

        }

        return cc;

    }

}

*Output:*

Enter a string: MyJava23

Enter a character: MJ

ThreadA:2

ThreadB:1

1. **Write java program using sleep() method of Thread class where the new class thread created from the previous class is implemented by using sleep method for 10,20,50,70,100 ns.**

public class SleepDemo {

    public static void main(String[] args) {

        Thread threadobj1 = new SleepThread("ThreadA", 10);

        Thread threadobj2 = new SleepThread("ThreadB", 20);

        Thread threadobj3 = new SleepThread("ThreadC", 50);

        Thread threadobj4 = new SleepThread("ThreadD", 70);

        Thread threadobj5 = new SleepThread("ThreadE", 100);

        threadobj1.start();

        threadobj2.start();

        threadobj3.start();

        threadobj4.start();

        threadobj5.start();

    }

}

class SleepThread extends Thread {

    private int sleepDuration;

    public SleepThread(String name, int sleepDuration) {

        super(name);

        this.sleepDuration = sleepDuration;

    }

    @Override

    public void run() {

        try {

            System.out.println(getName() + " is sleeping for " + sleepDuration + " ns");

            Thread.sleep(sleepDuration);

            System.out.println(getName() + " has woken up");

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

    }

}

*Output:*

ThreadB is sleeping for 20 ns

ThreadA is sleeping for 10 ns

ThreadD is sleeping for 70 ns

ThreadC is sleeping for 50 ns

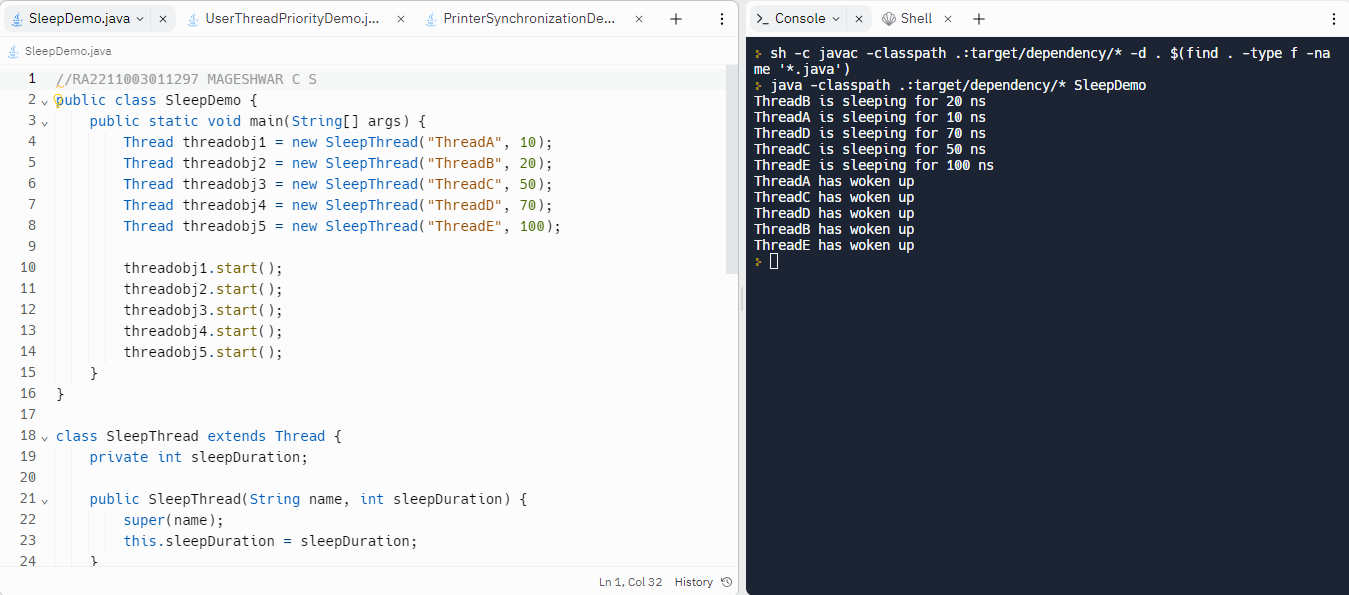
ThreadE is sleeping for 100 ns

ThreadA has woken up

ThreadC has woken up

ThreadD has woken up

ThreadB has woken up

ThreadE has woken up

1. **Write a java Thread program using Thread Priority for 5 processes and execute the priority order among the process.**

public class ThreadPriorityDemo {

    public static void main(String[] args) {

        Thread threadobj1 = new PriorityThread("ThreadA", Thread.MIN\_PRIORITY);

        Thread threadobj2 = new PriorityThread("ThreadB", 2);

        Thread threadobj3 = new PriorityThread("ThreadC", 5);

        Thread threadobj4 = new PriorityThread("ThreadD", 8);

        Thread threadobj5 = new PriorityThread("ThreadE", Thread.MAX\_PRIORITY);

        threadobj1.start();

        threadobj2.start();

        threadobj3.start();

        threadobj4.start();

        threadobj5.start();

    }

}

class PriorityThread extends Thread {

    public PriorityThread(String name, int priority) {

        super(name);

        setPriority(priority);

    }

    @Override

    public void run() {

        System.out.println(getName() + " is running with priority " + getPriority());

    }

}

*Output:*

ThreadA is running with priority 1

ThreadD is running with priority 8

ThreadB is running with priority 2

ThreadC is running with priority 5

ThreadE is running with priority 10

