**Day-11 Basics of Java**

**Problem Statement 1: Multithreading in Java**

**1.**

**package** Mphasis;

**class** Storage {

**private** **int** value;

**public** **synchronized** **void** setValue(**int** value) {

**this**.value = value;

   }

**public** **synchronized** **int** getValue() {

**return** value;

   }

}

**class** Counter **implements** Runnable {

**private** Storage storage;

**public** Counter(Storage storage) {

**this**.storage = storage;

   }

@Override

**public** **void** run() {

**int** count = 0;

**while** (**true**) {

storage.setValue(count);

count++;

**try** {

Thread.sleep(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

           }

       }

   }

}

**class** Printer **implements** Runnable {

**private** Storage storage;

**public** Printer(Storage storage) {

**this**.storage = storage;

   }

@Override

**public** **void** run() {

**while** (**true**) {

System.out.println("Current value: " + storage.getValue());

**try** {

Thread.sleep(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

           }

       }

   }

}

**public** **class** Main {

**public** **static** **void** main(String[] args){

Storage storage = **new** Storage();

Thread counterThread = **new** Thread(**new** Counter(storage));

Thread printerThread = **new** Thread(**new** Printer(storage));

counterThread.start();

printerThread.start();

}

}

2. **import** java.util.concurrent.locks.Condition;

**import** java.util.concurrent.locks.Lock;

**import** java.util.concurrent.locks.ReentrantLock;

**public** **class** SequentialPrinting{

**private** **static** **final** **int** ***LIMIT*** = 10;

**private** **int** number = 1;

**private** **int** threadIdToRun = 1;

**private** **final** Lock lock = **new** ReentrantLock();

**private** **final** Condition condition = lock.newCondition();

**public** **static** **void** main(String[] args) {

SequentialPrinting sp = **new** SequentialPrinting();

Thread t1 = **new** Thread(sp.**new** Printer(1));

Thread t2 = **new** Thread(sp.**new** Printer(2));

Thread t3 = **new** Thread(sp.**new** Printer(3));

t1.start();

t2.start();

t3.start();

}

**private** **class** Printer **implements** Runnable {

**private** **final** **int** threadId;

Printer(**int** threadId) {

**this**.threadId = threadId;

}

@Override

**public** **void** run() {

**while** (**true**) {

lock.lock();

ry {

**while** (threadId != threadIdToRun) {

condition.await();

}

**if** (number > LIMIT) {

condition.signalAll();

**break**; }

System.out.println("Thread " + threadId + " prints: " + number);

number++;

threadIdToRun = threadId % 3 + 1;

condition.signalAll();

} **catch** (InterruptedException e){

Thread.currentThread().interrupt();

}**finally**

lock.unlock();

}

}

}

}

**3.**

**import** java.io.BufferedReader;

**import** java.io.FileReader;

**import** java.io.IOException;

**import** java.util.concurrent.BlockingQueue;

**import** java.util.concurrent.LinkedBlockingQueue;

**class** NumberReader **implements** Runnable {

**private** String filename;

**private** BlockingQueue<Integer> queue;

**public** NumberReader(String filename, BlockingQueue<Integer> queue) {

**this**.filename = filename;

**this**.queue = queue;

}

@Override

**public** **void** run() {

**try** (BufferedReader reader = **new** BufferedReader(**new** FileReader(filename))) {

String line;

**while** ((line = reader.readLine()) != **null**) {

**int** number = Integer.parseInt(line.trim());

System.out.println("Read number: " + number + " from " + filename);

queue.put(number);

Thread.sleep((**int**) (Math.random() \* 1000));

}

queue.put(-1);

} **catch** (IOException | InterruptedException e) {

e.printStackTrace();

}

}

}

**class** FactorialCalculator **implements** Runnable {

**private** BlockingQueue<Integer> queue;

**private** **int** endSignals;

**public** FactorialCalculator(BlockingQueue<Integer> queue, **int** endSignals) {

**this**.queue = queue;

**this**.endSignals = endSignals;

}

@Override

**public** **void** run() {

**try** {

**int** endCount = 0;

**while** (**true**) {

**int** number = queue.take();

**if** (number == -1) {

endCount++;

**if** (endCount == endSignals) **break**;

} **else** {

**long** factorial = calculateFactorial(number);

System.out.println("Factorial of " + number + " is " + factorial);

Thread.sleep((**int**) (Math.random() \* 1000));

}

}

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

**private** **long** calculateFactorial(**int** number) {

**long** result = 1;

**for** (**int** i = 1; i <= number; i++) {

result \*= i;

}

**return** result;

}

}

**class** Main{

**public** **static** **void** main(String[] args) {

**if** (args.length < 2) {

System.out.println("Please provide two filenames as command line arguments.");

**return**;

}

String filename1 = args[0];

String filename2 = args[1];

BlockingQueue<Integer> queue = **new** LinkedBlockingQueue<>();

Thread readerThread1 = **new** Thread(**new** NumberReader(filename1, queue));

Thread readerThread2 = **new** Thread(**new** NumberReader(filename2, queue));

Thread factorialThread = **new** Thread(**new** FactorialCalculator(queue, 2));

readerThread1.start();

readerThread2.start();

factorialThread.start();

**try** {

readerThread1.join();

readerThread2.join();

factorialThread.join();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}