**Faculty of Computing**

**SE-314: Software Construction**

**Class: BESE 13AB**

# Lab 14: Concurrency

**CLO-03:** Design and develop solutions based on Software Construction principles.  
**CLO-04:** Use modern tools such as Eclipse, NetBeans etc. for software construction.

**Date: 23rd Dec 2024**

**Time: 10:00 AM** **- 12:50 PM   
 02:30 PM – 04:50 PM**

**Name: Affan Rehman**

**CMS: 374064**

**Instructor: Dr. Mehvish Rashid  
Lab Engineer: Mr. Aftab Farooq**

**Introduction:**

# Lab 14: Concurrency

The objective of this lab manual is to provide hands on experience with concurrency concepts in

Java. Students will learn how to create and manage threads,synchronize access to shared resources, and understand the implications of concurrent programming.

## Lab Tasks

**Lab Task 1: Introduction to Multithreading**

Objective: To create and execute multiple threads in Java.

1. **Task Description:**
   * Write a Java program that creates two threads.
   * One thread prints numbers from 1 to 10.
   * The other thread prints the squares of numbers from 1 to 10.
   * Ensure both threads run concurrently and observe the output.
2. **Key Concepts:**
   * Thread creation using Thread class and Runnable interface.
   * Basic multithreading behavior.

#### **Lab Task 2: Thread Synchronization**

Objective: To implement thread synchronization to avoid race conditions.

1. **Task Description:**
   * Write a Java program that creates three threads.
   * All threads should access a shared counter variable and increment it 100 times each.
   * Implement synchronization to ensure that the final value of the counter is 300.
2. **Key Concepts:**
   * Race conditions.
   * Using the synchronized keyword to manage thread safety.
   * Shared resources.

#### **Lab Task 3: Concurrent Data Structures**

Objective: To implement and use thread-safe data structures.

1. **Task Description:**
   * Write a Java program that simulates concurrent access to a shared list by multiple threads.
   * Use a CopyOnWriteArrayList or ConcurrentHashMap to ensure thread-safe operations.
   * Test the program by creating multiple threads that read and write to the shared data structure concurrently.
2. **Key Concepts:**
   * Concurrent data structures (CopyOnWriteArrayList, ConcurrentHashMap).
   * Safe concurrent operations without explicit synchronization.

#### **Lab Task 4: Simulation of Bank Transaction System**

Objective: To simulate a simple bank transaction system where multiple threads perform deposits and withdrawals concurrently.

1. **Task Description:**
   * Write a Java program to simulate a bank account with multiple clients (threads).
   * Each client thread performs random deposit and withdrawal operations.
   * Ensure the account balance is thread-safe and accurate after all transactions.
2. **Key Concepts:**
   * Thread synchronization.
   * Atomic operations using AtomicInteger or synchronized methods.

### Deliverables:

Compile a single word document by filling in the solution part and submit this Word file on LMS.

In case of any problems with submissions on LMS, submit your Lab assignments by emailing it to [aftab.farooq@seecs.edu.pk.](mailto:aftab.farooq@seecs.edu.pk.)

**Solution**

**Task 1**

public class LabTask1 {

public LabTask1() {

// Default constructor

}

public static void main(String[] args) {

Thread numberThread = new Thread(new NumberPrinter());

Thread squareThread = new Thread(new SquarePrinter());

numberThread.start();

squareThread.start();

}

}

class NumberPrinter implements Runnable {

@Override

public void run() {

try {

System.out.println("Number Thread Started");

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

System.out.println("Number: " + i);

Thread.sleep(500);

}

} catch (InterruptedException e) {

System.out.println("Number Thread interrupted");

}

}

}

class SquarePrinter implements Runnable {

@Override

public void run() {

try {

System.out.println("Square Thread Started");

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

System.out.println("Square: " + (i \* i));

Thread.sleep(500);

}

} catch (InterruptedException e) {

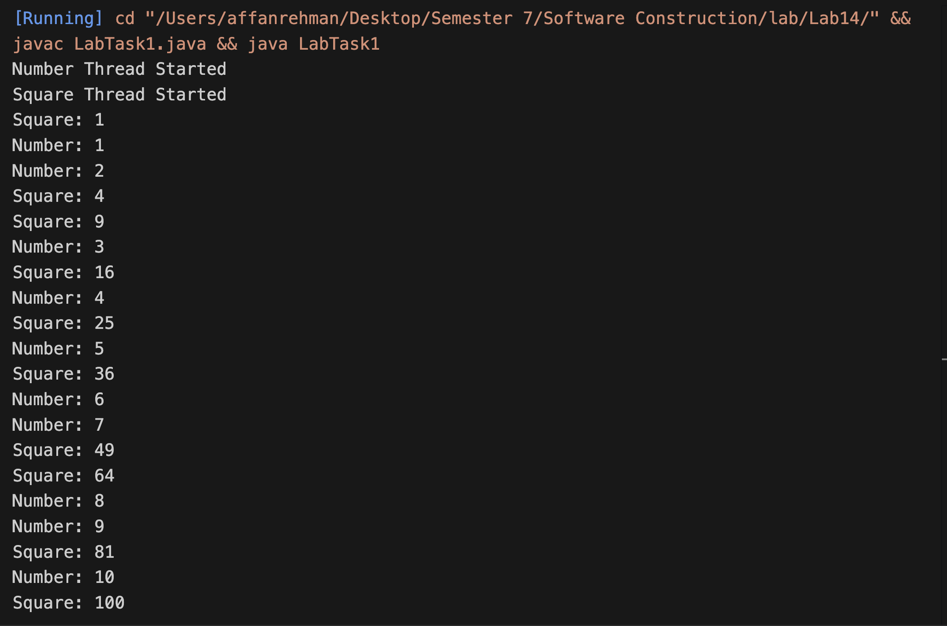
System.out.println("Square Thread interrupted");

}

}

}

**Output:**

****

**Task 2**

public class LabTask2 {

private static int counter = 0;

public static void main(String[] args) throws InterruptedException {

Thread t1 = new Thread(new CounterIncrementer(), "Thread-1");

Thread t2 = new Thread(new CounterIncrementer(), "Thread-2");

Thread t3 = new Thread(new CounterIncrementer(), "Thread-3");

t1.start();

t2.start();

t3.start();

t1.join();

t2.join();

t3.join();

System.out.println("Final Counter Value: " + counter);

}

static class CounterIncrementer implements Runnable {

@Override

public void run() {

for (int i = 0; i < 100; i++) {

incrementCounter();

}

}

private static synchronized void incrementCounter() {

counter++;

System.out.println(

Thread.currentThread().getName() + " incremented counter to: " + counter

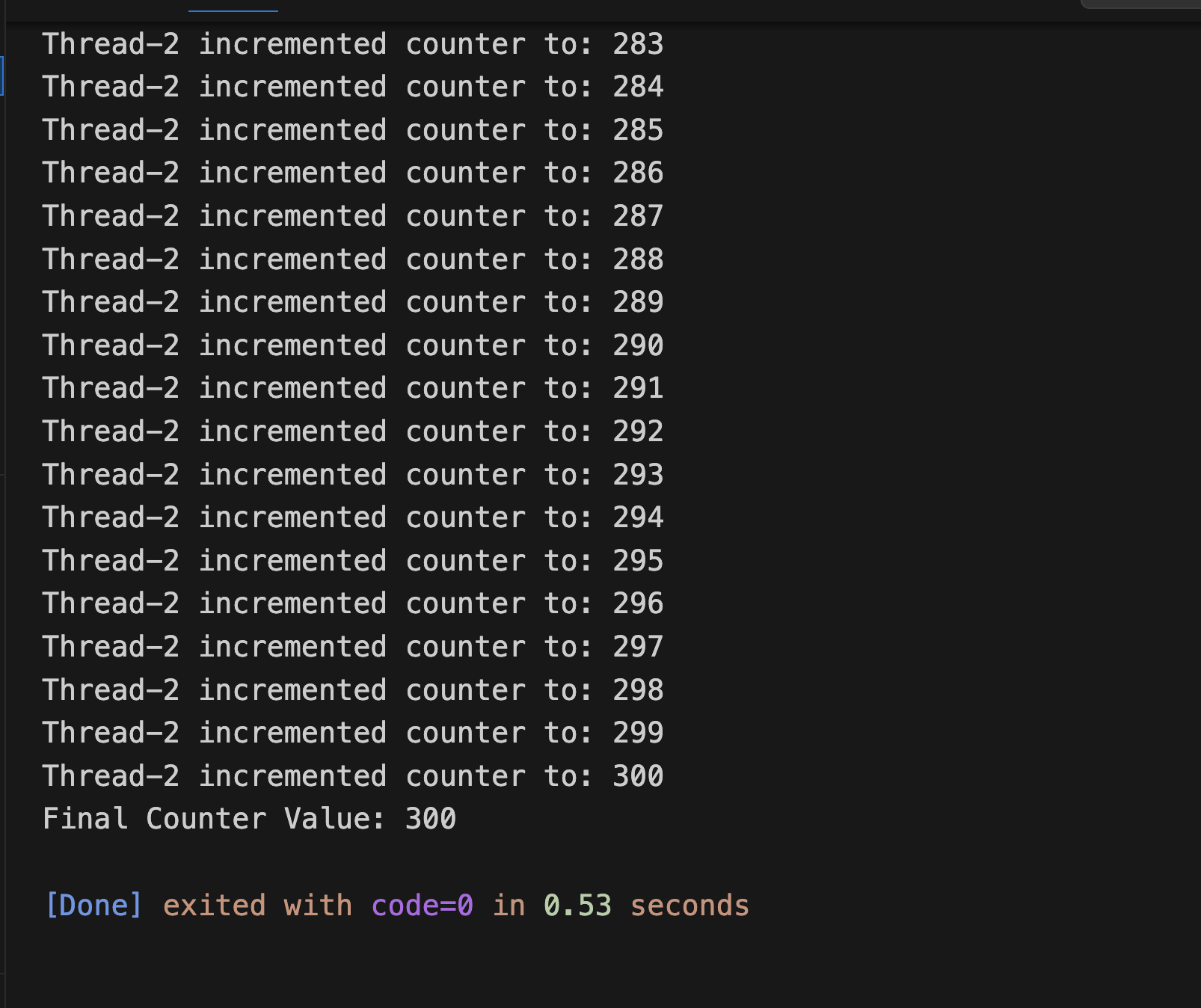
);

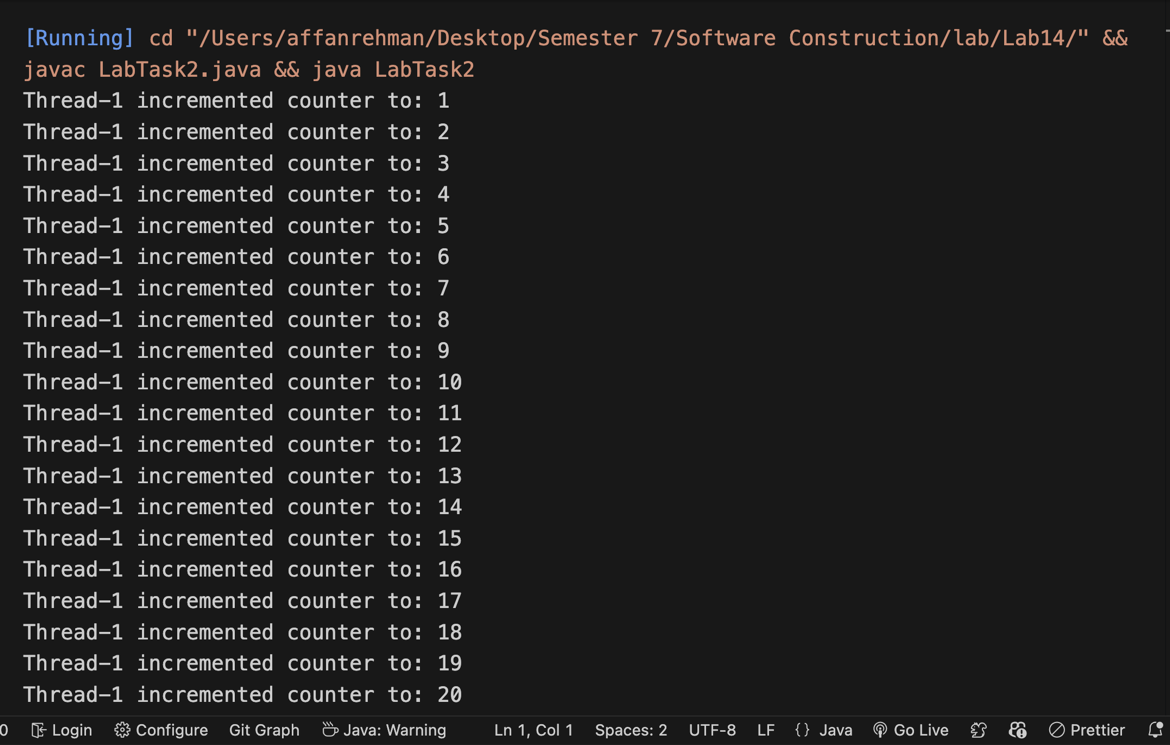
}

}

}

**Output:**

****

****

**Task 3**

import java.util.concurrent.CopyOnWriteArrayList;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.TimeUnit;

public class LabTask3 {

private static final CopyOnWriteArrayList<Integer> sharedList = new CopyOnWriteArrayList<>();

public static void main(String[] args) throws InterruptedException {

ExecutorService executorService = Executors.newFixedThreadPool(3);

for (int i = 0; i < 3; i++) {

executorService.submit(new ListWriter());

}

for (int i = 0; i < 2; i++) {

executorService.submit(new ListReader());

}

executorService.shutdown();

executorService.awaitTermination(10, TimeUnit.SECONDS);

System.out.println("Final list size: " + sharedList.size());

}

static class ListWriter implements Runnable {

@Override

public void run() {

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

int number = (int) (Math.random() \* 100);

sharedList.add(number);

System.out.println(

Thread.currentThread().getName() + " added number: " + number

);

try {

Thread.sleep(100);

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

}

static class ListReader implements Runnable {

@Override

public void run() {

while (!Thread.currentThread().isInterrupted()) {

for (Integer number : sharedList) {

System.out.println(

Thread.currentThread().getName() + " read number: " + number

);

}

try {

Thread.sleep(200);

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

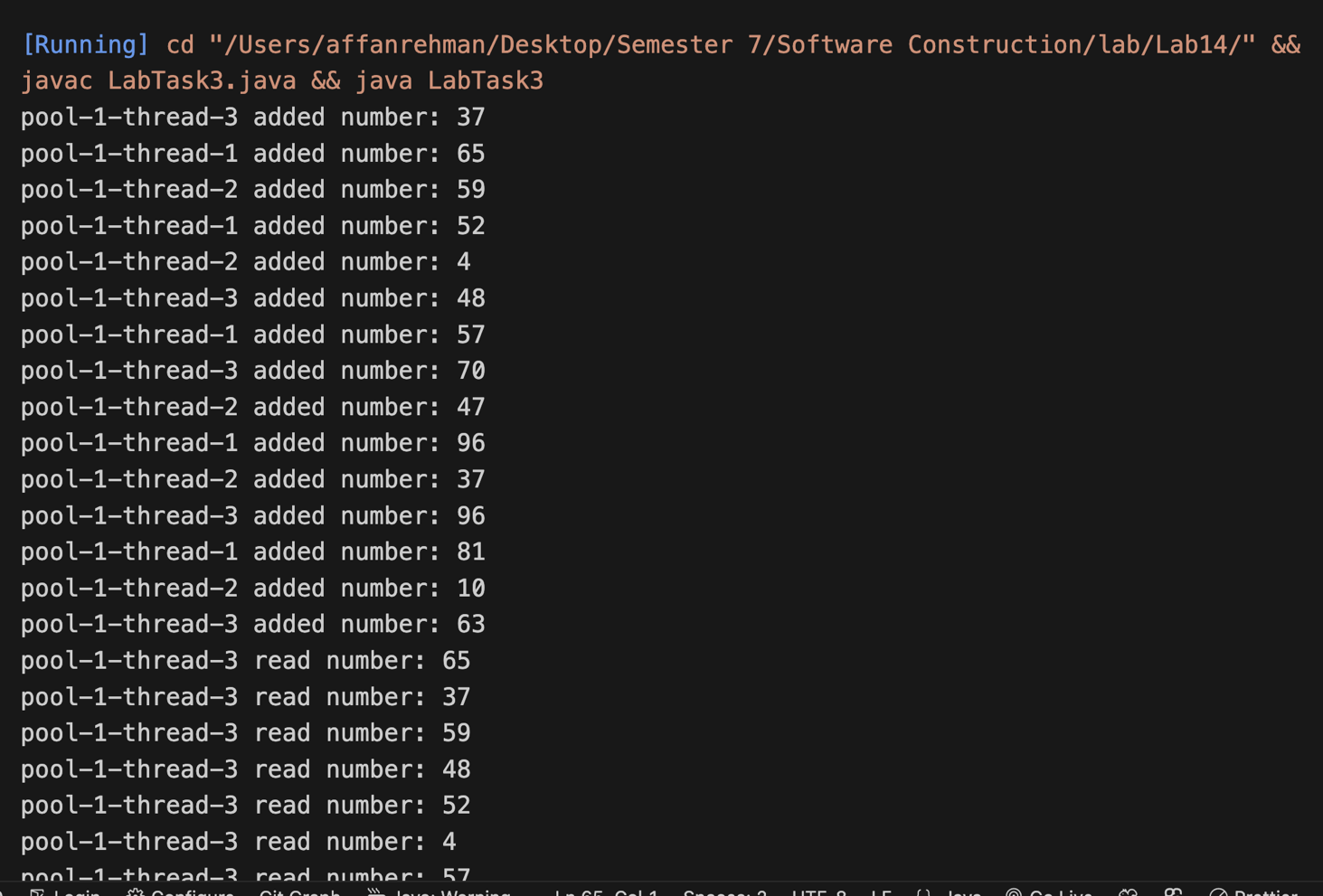
}

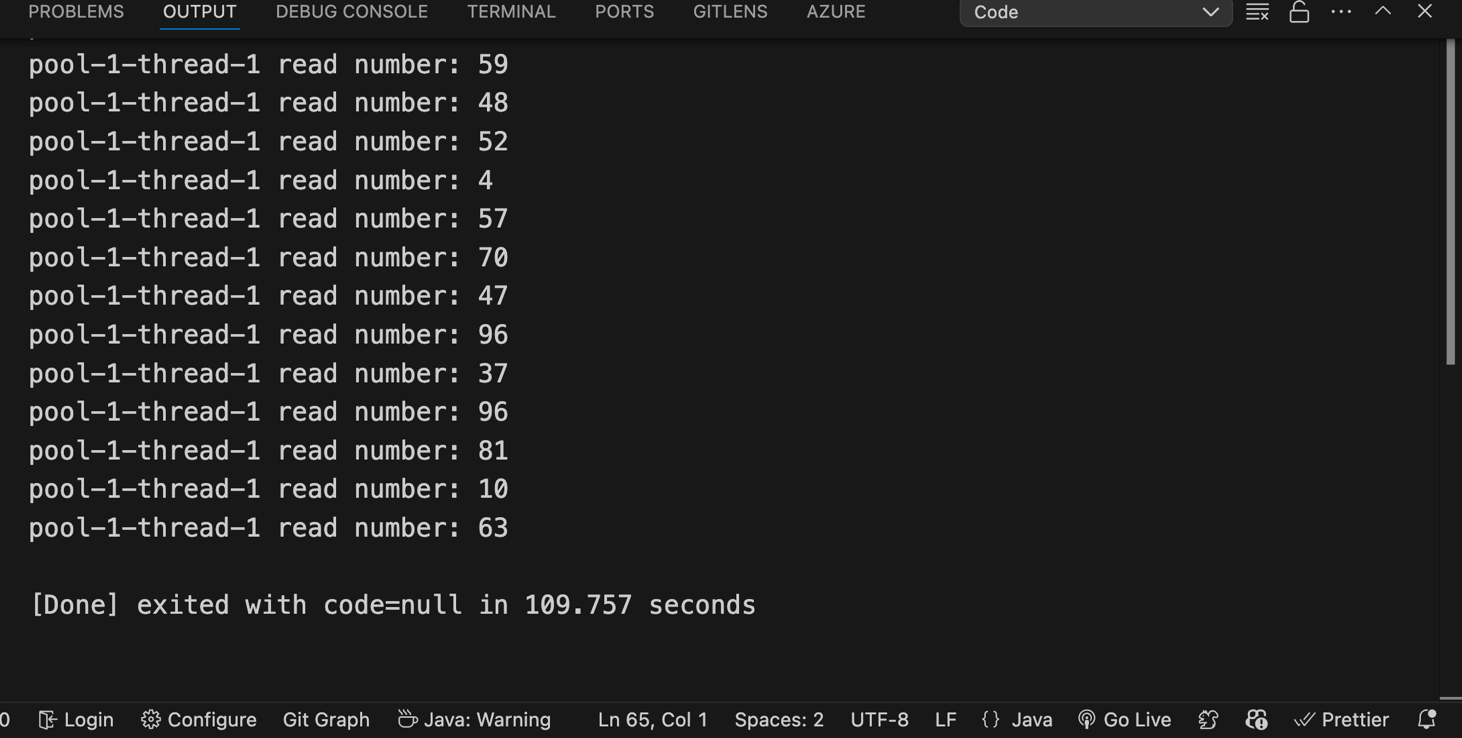
}

}

}

}

**Output:**

****

**Task 4**

import java.util.concurrent.ThreadLocalRandom;

import java.util.concurrent.atomic.AtomicInteger;

public class LabTask4 {

public static void main(String[] args) throws InterruptedException {

BankAccount account = new BankAccount(1000);

Thread[] clients = new Thread[5];

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

clients[i] = new Thread(new BankClient(account), "Client-" + (i + 1));

clients[i].start();

}

for (Thread client : clients) {

client.join();

}

System.out.println("Final Balance: $" + account.getBalance());

}

}

class BankAccount {

private AtomicInteger balance;

public BankAccount(int initialBalance) {

this.balance = new AtomicInteger(initialBalance);

}

public boolean withdraw(int amount) {

while (true) {

int currentBalance = balance.get();

if (currentBalance < amount) {

return false;

}

if (balance.compareAndSet(currentBalance, currentBalance - amount)) {

return true;

}

}

}

public void deposit(int amount) {

balance.addAndGet(amount);

}

public int getBalance() {

return balance.get();

}

}

class BankClient implements Runnable {

private final BankAccount account;

private final ThreadLocalRandom random = ThreadLocalRandom.current();

public BankClient(BankAccount account) {

this.account = account;

}

@Override

public void run() {

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

if (random.nextBoolean()) {

int amount = random.nextInt(100, 501);

account.deposit(amount);

System.out.println(

Thread.currentThread().getName() +

" deposited $" +

amount +

". New balance: $" +

account.getBalance()

);

} else {

int amount = random.nextInt(50, 251);

boolean success = account.withdraw(amount);

if (success) {

System.out.println(

Thread.currentThread().getName() +

" withdrew $" +

amount +

". New balance: $" +

account.getBalance()

);

} else {

System.out.println(

Thread.currentThread().getName() +

" failed to withdraw $" +

amount +

" (insufficient funds)"

);

}

}

try {

Thread.sleep(random.nextInt(100, 300));

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

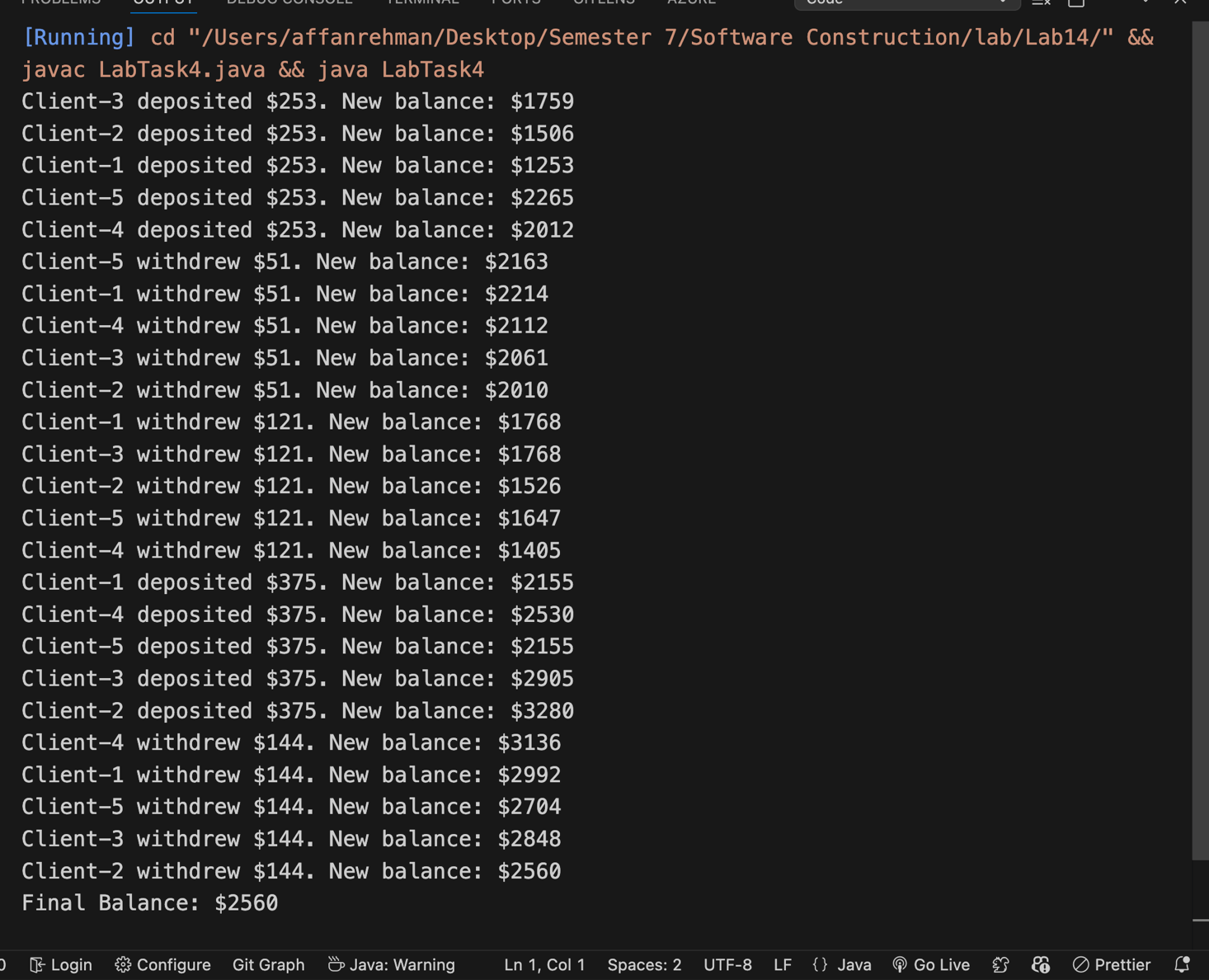
break;

}

}

}

}

**Output:**