**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** | MUHAMMAD SAAD UMER |
| **CLASS** | BESE 13A |
| **CMS** | 408485 |

**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.

|  |
| --- |
| **SOLUTION WITH OUTPUTS** |
| **public class MultithreadingExample {**  **public static void main(String[] args) {**  **// Create the first thread using the Thread class**  **Thread printNumbersThread = new Thread(new Runnable() {**  **@Override**  **public void run() {**  **for (int i = 1; i <= 10; i++) {**  **System.out.println("Number: " + i);**  **try {**  **Thread.sleep(500); // Delay for better observation**  **} catch (InterruptedException e) {**  **System.out.println("Thread interrupted: " + e.getMessage());**  **}**  **}**  **}**  **});**  **// Create the second thread using the Runnable interface**  **Thread printSquaresThread = new Thread(new Runnable() {**  **@Override**  **public void run() {**  **for (int i = 1; i <= 10; i++) {**  **System.out.println("Square of " + i + ": " + (i \* i));**  **try {**  **Thread.sleep(500); // Delay for better observation**  **} catch (InterruptedException e) {**  **System.out.println("Thread interrupted: " + e.getMessage());**  **}**  **}**  **}**  **});**  **// Start both threads**  **printNumbersThread.start();**  **printSquaresThread.start();**  **// Join threads to ensure they finish before the main thread exits (optional)**  **try {**  **printNumbersThread.join();**  **printSquaresThread.join();**  **} catch (InterruptedException e) {**  **System.out.println("Main thread interrupted: " + e.getMessage());**  **}**  **System.out.println("Both threads have finished execution.");**  **}**  **}** |

#### **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.

|  |
| --- |
| **SOLUTION WITH OUTPUTS** |
| **public class ThreadSynchronizationExample {**  **public static void main(String[] args) {**  **// Shared Counter object**  **Counter counter = new Counter();**  **// Create three threads**  **Thread thread1 = new Thread(new IncrementTask(counter));**  **Thread thread2 = new Thread(new IncrementTask(counter));**  **Thread thread3 = new Thread(new IncrementTask(counter));**  **// Start all threads**  **thread1.start();**  **thread2.start();**  **thread3.start();**  **// Wait for all threads to finish**  **try {**  **thread1.join();**  **thread2.join();**  **thread3.join();**  **} catch (InterruptedException e) {**  **System.out.println("Thread interrupted: " + e.getMessage());**  **}**  **// Print the final counter value**  **System.out.println("Final Counter Value: " + counter.getValue());**  **}**  **}**  **// Shared Counter class**  **class Counter {**  **private int count = 0;**  **// Synchronized method to increment the counter**  **public synchronized void increment() {**  **count++;**  **}**  **// Method to get the counter value**  **public int getValue() {**  **return count;**  **}**  **}**  **// Task to increment the counter**  **class IncrementTask implements Runnable {**  **private final Counter counter;**  **public IncrementTask(Counter counter) {**  **this.counter = counter;**  **}**  **@Override**  **public void run() {**  **for (int i = 0; i < 100; i++) {**  **counter.increment();**  **try {**  **Thread.sleep(1); // Slight delay to make race conditions more apparent without synchronization**  **} catch (InterruptedException e) {**  **System.out.println("Thread interrupted: " + e.getMessage());**  **}**  **}**  **}**  **}** |

#### **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.

|  |
| --- |
| **SOLUTION WITH OUTPUTS** |
| **import java.util.concurrent.CopyOnWriteArrayList;**  **public class ThreadSafeDataStructureExample {**  **public static void main(String[] args) {**  **// Shared thread-safe list**  **CopyOnWriteArrayList<Integer> sharedList = new CopyOnWriteArrayList<>();**  **// Create threads to write to the shared list**  **Thread writer1 = new Thread(new ListWriter(sharedList, 1, 5));**  **Thread writer2 = new Thread(new ListWriter(sharedList, 6, 10));**  **Thread writer3 = new Thread(new ListWriter(sharedList, 11, 15));**  **// Create threads to read from the shared list**  **Thread reader1 = new Thread(new ListReader(sharedList));**  **Thread reader2 = new Thread(new ListReader(sharedList));**  **// Start all threads**  **writer1.start();**  **writer2.start();**  **writer3.start();**  **reader1.start();**  **reader2.start();**  **// Wait for all writer threads to finish**  **try {**  **writer1.join();**  **writer2.join();**  **writer3.join();**  **} catch (InterruptedException e) {**  **System.out.println("Thread interrupted: " + e.getMessage());**  **}**  **// Print the final state of the shared list**  **System.out.println("Final Shared List: " + sharedList);**  **}**  **}**  **// Task to write to the shared list**  **class ListWriter implements Runnable {**  **private final CopyOnWriteArrayList<Integer> list;**  **private final int start;**  **private final int end;**  **public ListWriter(CopyOnWriteArrayList<Integer> list, int start, int end) {**  **this.list = list;**  **this.start = start;**  **this.end = end;**  **}**  **@Override**  **public void run() {**  **for (int i = start; i <= end; i++) {**  **list.add(i);**  **System.out.println(Thread.currentThread().getName() + " added: " + i);**  **try {**  **Thread.sleep(100); // Simulate delay**  **} catch (InterruptedException e) {**  **System.out.println("Thread interrupted: " + e.getMessage());**  **}**  **}**  **}**  **}**  **// Task to read from the shared list**  **class ListReader implements Runnable {**  **private final CopyOnWriteArrayList<Integer> list;**  **public ListReader(CopyOnWriteArrayList<Integer> list) {**  **this.list = list;**  **}**  **@Override**  **public void run() {**  **while (true) {**  **System.out.println(Thread.currentThread().getName() + " reading: " + list);**  **try {**  **Thread.sleep(200); // Simulate delay**  **} catch (InterruptedException e) {**  **System.out.println("Thread interrupted: " + e.getMessage());**  **break;**  **}**  **}**  **}**  **}** |

#### **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.

|  |
| --- |
| **SOLUTION WITH OUTPUTS** |
| **import java.util.concurrent.atomic.AtomicInteger;**  **import java.util.Random;**  **public class BankTransactionSystem {**  **public static void main(String[] args) {**  **// Shared bank account**  **BankAccount account = new BankAccount();**  **// Create client threads**  **Thread client1 = new Thread(new Client(account), "Client 1");**  **Thread client2 = new Thread(new Client(account), "Client 2");**  **Thread client3 = new Thread(new Client(account), "Client 3");**  **// Start client threads**  **client1.start();**  **client2.start();**  **client3.start();**  **// Wait for all threads to finish**  **try {**  **client1.join();**  **client2.join();**  **client3.join();**  **} catch (InterruptedException e) {**  **System.out.println("Main thread interrupted: " + e.getMessage());**  **}**  **// Print final account balance**  **System.out.println("Final Account Balance: " + account.getBalance());**  **}**  **}**  **// Bank account class with thread-safe operations**  **class BankAccount {**  **private AtomicInteger balance = new AtomicInteger(0);**  **// Method to deposit money**  **public void deposit(int amount) {**  **balance.getAndAdd(amount);**  **System.out.println(Thread.currentThread().getName() + " deposited: " + amount + " | Balance: " + balance.get());**  **}**  **// Method to withdraw money**  **public void withdraw(int amount) {**  **if (balance.get() >= amount) {**  **balance.getAndAdd(-amount);**  **System.out.println(Thread.currentThread().getName() + " withdrew: " + amount + " | Balance: " + balance.get());**  **} else {**  **System.out.println(Thread.currentThread().getName() + " attempted to withdraw: " + amount + " | Insufficient funds! Balance: " + balance.get());**  **}**  **}**  **// Method to get the current balance**  **public int getBalance() {**  **return balance.get();**  **}**  **}**  **// Client class representing a thread performing transactions**  **class Client implements Runnable {**  **private final BankAccount account;**  **private final Random random = new Random();**  **public Client(BankAccount account) {**  **this.account = account;**  **}**  **@Override**  **public void run() {**  **for (int i = 0; i < 5; i++) { // Each client performs 5 transactions**  **int transactionType = random.nextInt(2); // 0 for deposit, 1 for withdrawal**  **int amount = random.nextInt(100) + 1; // Random amount between 1 and 100**  **if (transactionType == 0) {**  **account.deposit(amount);**  **} else {**  **account.withdraw(amount);**  **}**  **try {**  **Thread.sleep(100); // Simulate processing time**  **} catch (InterruptedException e) {**  **System.out.println(Thread.currentThread().getName() + " interrupted: " + e.getMessage());**  **}**  **}**  **}**  **}** |

### 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.)