**Faculty of Computing**

**SE-314: Software Construction**

**Name: Muhammad Ahsan**

**CMS: 406267**

**Class: BESE 13 B**

# 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**

**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.

**Code:**

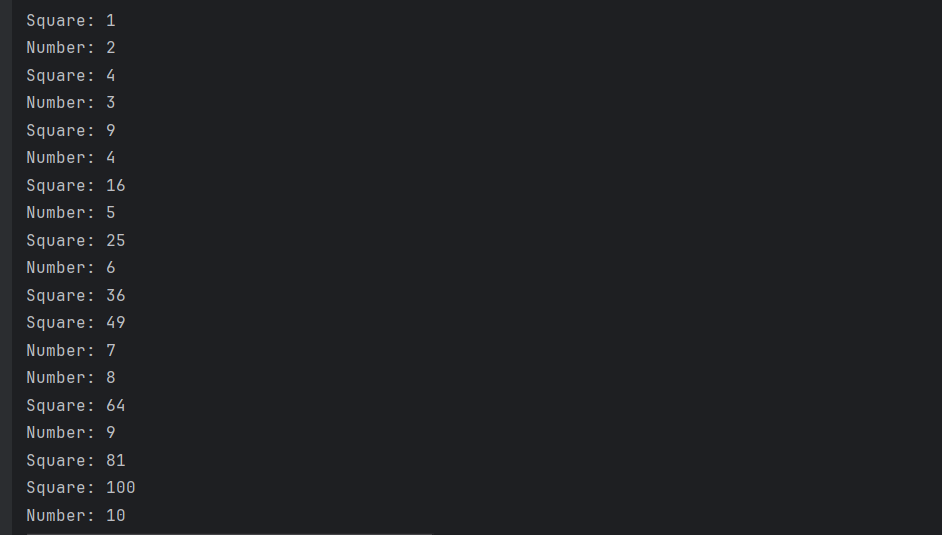
public class ThreadTask1 {  
  
 public static void main(String[] args) {  
   
 Thread numberThread = new Thread(() -> {  
 for (int i = 1; i <= 10; i++) {  
 System.*out*.println("Number: " + i);  
 try {  
 Thread.*sleep*(500);

} catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 });

Thread squareThread = new Thread(() -> {  
 for (int i = 1; i <= 10; i++) {  
 System.*out*.println("Square: " + (i \* i));  
 try {  
 Thread.*sleep*(500);

} catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 });  
  
 // Start both threads  
 numberThread.start();  
 squareThread.start();  
 }  
}

**Output:**

****

#### 

#### **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.

**Code:**

class SharedCounter {  
 private int counter = 0;  
  
 // Synchronized method to increment the counter  
 public synchronized void increment() {  
 counter++;  
 }  
  
 // Method to get the counter's value  
 public int getValue() {  
 return counter;  
 }  
}  
  
public class ThreadSynchronizationExample {  
  
 public static void main(String[] args) {  
 SharedCounter sharedCounter = new SharedCounter();  
  
 // Create three threads that increment the counter  
 Thread thread1 = new Thread(() -> {  
 for (int i = 0; i < 100; i++) {  
 sharedCounter.increment();  
 }  
 });  
  
 Thread thread2 = new Thread(() -> {  
 for (int i = 0; i < 100; i++) {  
 sharedCounter.increment();  
 }  
 });  
  
 Thread thread3 = new Thread(() -> {  
 for (int i = 0; i < 100; i++) {  
 sharedCounter.increment();  
 }  
 });  
  
 // 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) {  
 e.printStackTrace();  
 }  
  
 // Print the final value of the counter  
 System.*out*.println("Final Counter Value: " + sharedCounter.getValue());  
 }  
}

**Output:**

**A screenshot of a computer program

Description automatically generated**

#### **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.

**Code:**

import java.util.concurrent.CopyOnWriteArrayList;  
  
public class ConcurrentListExample {  
  
 public static void main(String[] args) {  
 // Shared CopyOnWriteArrayList  
 CopyOnWriteArrayList<Integer> sharedList = new CopyOnWriteArrayList<>();  
  
 // Writer thread: Adds numbers to the list  
 Thread writerThread = new Thread(() -> {  
 for (int i = 1; i <= 10; i++) {  
 sharedList.add(i);  
 System.*out*.println("Added: " + i);  
 try {  
 Thread.*sleep*(100); // Simulate delay  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 });  
  
 // Reader thread 1: Reads the list  
 Thread readerThread1 = new Thread(() -> {  
 for (int i = 0; i < 10; i++) {  
 System.*out*.println("Reader 1: " + sharedList);  
 try {  
 Thread.*sleep*(150); // Simulate delay  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 });  
  
 // Reader thread 2: Reads the list  
 Thread readerThread2 = new Thread(() -> {  
 for (int i = 0; i < 10; i++) {  
 System.*out*.println("Reader 2: " + sharedList);  
 try {  
 Thread.*sleep*(200); // Simulate delay  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 });  
  
 // Start all threads  
 writerThread.start();  
 readerThread1.start();  
 readerThread2.start();  
  
 // Wait for threads to complete  
 try {  
 writerThread.join();  
 readerThread1.join();  
 readerThread2.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 System.*out*.println("Final List: " + sharedList);  
 }  
}

**Output:**

**A screen shot of a computer

Description automatically generated**

#### **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.

**Code:**

import java.util.Random;  
  
class BankAccount {  
 private int balance;  
  
 public BankAccount(int initialBalance) {  
 this.balance = initialBalance;  
 }  
  
 // Synchronized method to deposit money  
 public synchronized void deposit(int amount) {  
 balance += amount;  
 System.*out*.println(Thread.*currentThread*().getName() + " deposited: " + amount + ", New Balance: " + balance);  
 }  
  
 // Synchronized method to withdraw money  
 public synchronized void withdraw(int amount) {  
 if (balance >= amount) {  
 balance -= amount;  
 System.*out*.println(Thread.*currentThread*().getName() + " withdrew: " + amount + ", New Balance: " + balance);  
 } else {  
 System.*out*.println(Thread.*currentThread*().getName() + " attempted to withdraw: " + amount + " (Insufficient funds)");  
 }  
 }  
  
 // Get the current balance  
 public synchronized int getBalance() {  
 return balance;  
 }  
}  
  
public class BankSimulation {  
 public static void main(String[] args) {  
 // Create a shared bank account with an initial balance of 1000  
 BankAccount account = new BankAccount(1000);  
  
 // Runnable task for clients  
 Runnable clientTask = () -> {  
 Random random = new Random();  
 for (int i = 0; i < 10; i++) { // Each client performs 10 transactions  
 int amount = random.nextInt(500); // Random amount between 0 and 499  
 if (random.nextBoolean()) {  
 account.deposit(amount);  
 } else {  
 account.withdraw(amount);  
 }  
 try {  
 Thread.*sleep*(100); // Simulate transaction processing time  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 };  
  
 // Create and start multiple client threads  
 Thread client1 = new Thread(clientTask, "Client 1");  
 Thread client2 = new Thread(clientTask, "Client 2");  
 Thread client3 = new Thread(clientTask, "Client 3");  
  
 client1.start();  
 client2.start();  
 client3.start();  
  
 // Wait for all threads to finish  
 try {  
 client1.join();  
 client2.join();  
 client3.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 // Print the final account balance  
 System.*out*.println("Final Account Balance: " + account.getBalance());  
 }  
}

**Output:**

Client 1 deposited: 234, New Balance: 1234

Client 3 withdrew: 195, New Balance: 1039

Client 2 deposited: 240, New Balance: 1279

Client 1 deposited: 287, New Balance: 1566

Client 3 withdrew: 492, New Balance: 1074

Client 2 withdrew: 51, New Balance: 1023

Client 1 deposited: 71, New Balance: 1094

Client 3 deposited: 290, New Balance: 1384

Client 2 withdrew: 69, New Balance: 1315

Client 3 withdrew: 360, New Balance: 955

Client 1 deposited: 113, New Balance: 1068

Client 2 deposited: 336, New Balance: 1404

Client 3 withdrew: 217, New Balance: 1187

Client 1 deposited: 239, New Balance: 1426

Client 2 withdrew: 55, New Balance: 1371

Client 3 deposited: 190, New Balance: 1561

Client 1 deposited: 402, New Balance: 1963

Client 2 deposited: 432, New Balance: 2395

Client 3 deposited: 9, New Balance: 2404

Client 1 withdrew: 68, New Balance: 2336

Client 2 withdrew: 324, New Balance: 2012

Client 3 withdrew: 431, New Balance: 1581

Client 1 deposited: 227, New Balance: 1808

Client 2 deposited: 226, New Balance: 2034

Client 3 deposited: 432, New Balance: 2466

Client 1 withdrew: 90, New Balance: 2376

Client 2 withdrew: 336, New Balance: 2040

Client 3 deposited: 182, New Balance: 2222

Client 1 deposited: 267, New Balance: 2489

Client 2 deposited: 334, New Balance: 2823

Final Account Balance: 2823

**Github Repo:** [**https://github.com/ahsanirfan961/SC-Lab-14**](https://github.com/ahsanirfan961/SC-Lab-14)

### 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.)