**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**

**MUQADAS ZAHRA  
416557**

**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.

class NumberPrinter extends Thread {

    @Override

    public void run() {

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

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

            try {

                Thread.sleep(100); // Sleep for 100ms to simulate concurrent execution

            } catch (InterruptedException e) {

                e.printStackTrace();

            }

        }

    }

}

class SquarePrinter extends Thread {

    @Override

    public void run() {

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

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

            try {

                Thread.sleep(100); // Sleep for 100ms to simulate concurrent execution

            } catch (InterruptedException e) {

                e.printStackTrace();

            }

        }

    }

}

public class MultithreadingExample {

    public static void main(String[] args) {

        NumberPrinter numberPrinter = new NumberPrinter();

        SquarePrinter squarePrinter = new SquarePrinter();

        numberPrinter.start();

        squarePrinter.start();

    }

}

#### **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.
3. class Counter {
4. private int count = 0;
6. public synchronized void increment() {
7. count++;
8. }
10. public int getCount() {
11. return count;
12. }
13. }
14. class CounterIncrementer extends Thread {
15. private Counter counter;
17. public CounterIncrementer(Counter counter) {
18. this.counter = counter;
19. }
21. @Override
22. public void run() {
23. for (int i = 0; i < 100; i++) {
24. counter.increment();
25. }
26. }
27. }
28. public class SynchronizationExample {
29. public static void main(String[] args) throws InterruptedException {
30. Counter counter = new Counter();
32. Thread t1 = new CounterIncrementer(counter);
33. Thread t2 = new CounterIncrementer(counter);
34. Thread t3 = new CounterIncrementer(counter);
36. t1.start();
37. t2.start();
38. t3.start();
40. t1.join();
41. t2.join();
42. t3.join();
44. System.out.println("Final Counter Value: " + counter.getCount());
45. }
46. }

#### **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.
3. import java.util.concurrent.CopyOnWriteArrayList;
4. class ListUpdater extends Thread {
5. private CopyOnWriteArrayList<Integer> list;
7. public ListUpdater(CopyOnWriteArrayList<Integer> list) {
8. this.list = list;
9. }
11. @Override
12. public void run() {
13. for (int i = 0; i < 10; i++) {
14. list.add(i);
15. System.out.println("Added: " + i);
16. try {
17. Thread.sleep(50);
18. } catch (InterruptedException e) {
19. e.printStackTrace();
20. }
21. }
22. }
23. }
24. public class ConcurrentListExample {
25. public static void main(String[] args) throws InterruptedException {
26. CopyOnWriteArrayList<Integer> list = new CopyOnWriteArrayList<>();
28. Thread t1 = new ListUpdater(list);
29. Thread t2 = new ListUpdater(list);
31. t1.start();
32. t2.start();
34. t1.join();
35. t2.join();
37. System.out.println("Final List: " + list);
38. }
39. }

#### **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.
3. import java.util.concurrent.atomic.AtomicInteger;
4. class BankAccount {
5. private AtomicInteger balance = new AtomicInteger(0);
7. public void deposit(int amount) {
8. balance.addAndGet(amount);
9. }
11. public void withdraw(int amount) {
12. balance.addAndGet(-amount);
13. }
15. public int getBalance() {
16. return balance.get();
17. }
18. }
19. class BankClient extends Thread {
20. private BankAccount account;
22. public BankClient(BankAccount account) {
23. this.account = account;
24. }
26. @Override
27. public void run() {
28. for (int i = 0; i < 5; i++) {
29. int amount = (int) (Math.random() \* 100);
30. if (Math.random() > 0.5) {
31. account.deposit(amount);
32. System.out.println(Thread.currentThread().getName() + " deposited: " + amount);
33. } else {
34. account.withdraw(amount);
35. System.out.println(Thread.currentThread().getName() + " withdrew: " + amount);
36. }
37. try {
38. Thread.sleep(200);
39. } catch (InterruptedException e) {
40. e.printStackTrace();
41. }
42. }
43. }
44. }
45. public class BankTransactionExample {
46. public static void main(String[] args) throws InterruptedException {
47. BankAccount account = new BankAccount();
49. Thread client1 = new BankClient(account);
50. Thread client2 = new BankClient(account);
51. Thread client3 = new BankClient(account);
53. client1.start();
54. client2.start();
55. client3.start();
57. client1.join();
58. client2.join();
59. client3.join();
61. System.out.println("Final Account Balance: " + account.getBalance());
62. }
63. }

**- Paste Code in word document**

**- Push Your Code on GitHub  
- Add Git Link in Document.**

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

**Solution**

### 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.)