NUR AZYYATI BINTI ABU BAKAR | 291560

STIWK3014 REAL TIME PROGRAMMING
Tutorial / Exercise 11: ReentrantReadWriteLock () Total 15 Marks

```
import java.util.concurrent.locks.ReentrantReadWriteLock;
import java.util.concurrent.locks.Lock;

public class BankAccountWithLock {
    private double balance;
    private final ReentrantReadWriteLock lock = new ReentrantReadWriteLock();
    private final Lock readLock = lock.readLock();
    private final Lock writeLock = lock.writeLock();

    public BankAccountWithLock(double initialBalance) {
        this.balance = initialBalance;
    }

    // Read balance (shared lock)
    public double getBalance() {
        readLock.lock();

        try {
    System.out.println(Thread.currentThread().getName() + " reads   balance: " +
    balance);
            return balance;
        } finally {
            readLock.unlock();
        }}

    // Deposit money (exclusive lock)
    public void deposit(double amount) {
        writeLock.lock();

        try {
    System.out.println(Thread.currentThread().getName() + " deposits:   " + amount);

         balance += amount;
        } finally {
            writeLock.unlock();
        } }

    // Withdraw money (exclusive lock)
    public void withdraw(double amount) {
        writeLock.lock();

        try {
            if (balance >= amount) {
    System.out.println(Thread.currentThread().getName() + "  withdraws: " + amount);

         balance -= amount;
            } else {
    System.out.println(Thread.currentThread().getName() + " insufficient funds for: "
    + amount);
            }

        } finally {
            writeLock.unlock();
        } }
}
```

The following questions are based on the above Java program.

a. Create the main class for this program

```
public class MainClass {
    public static void main(String[] args) {
        //create bankacc with RM1000 balance
        BankAccountWithLock myAccount = new BankAccountWithLock( initialBalance: 1000.0);

        //thread to read balance
        Thread readerA = new Thread(() -> {
            myAccount.getBalance();
        }, name: "Reader-A");

        //read balance
        Thread readerB = new Thread(() -> {
            myAccount.getBalance();
        }, name: "Reader-B");

        //Thread to deposit
        Thread depositor = new Thread(() -> {
            myAccount.deposit( amount: 300.0);
        }, name: "Depositor");

        //thread to withdraw
        Thread withdrawer = new Thread(() -> {
            myAccount.withdraw( amount: 200.0);
        }, name: "Withdrawer");
        //start
        readerA.start();
        readerB.start();
        depositor.start();
        withdrawer.start();

        //wait all threads to habis
        try {
            readerA.join();
            readerB.join();
            depositor.join();
            withdrawer.join();
        } catch (InterruptedException e) {
            System.out.println("Thread interrupted.");
        }
    }
}
```

(5 marks)

b. What is the output of this program?

```
"C:\Program Files\Java\jdk-16.0.1\bin\j
Reader-A reads  balance: 1000.0
Reader-B reads  balance: 1000.0
Withdrawer withdraws: 200.0
Depositor deposits: 300.0

Process finished with exit code 0
```

(3 marks)

c. What is the advantage of using `ReentrantReadWriteLock` over `synchronized` method in this program?

- the advantage of using ReentrantReadWriteLock over synchronized method in this program which is ReentrantReadWriteLock allows multiple threads to read the balance at the same time but for synchronized method is only one thread can access at a time
even for reading, so using ReentrantReadWriteLock makes the program faster and more efficient.

(2 marks)

d. Explain the difference between `readLock()` and `writeLock()`.

- the difference between readLock() and writeLock() is readlock used when we only want to read the data thus it can many thread use at the same time, but for writelock is used when we want to change the data like deposit and withdraw above and only one thread can use at a time.

(3 marks)

e. Why is `writeLock.unlock()` placed in a `finally` block?
- writeLock.unlock() placed in a finally block is to make sure the lock is always released even have problem. Like example if forget to unlock it other threads cannot continue and it will get stuck. That's why finally is like a safety step to always unlock the lock no matter what happens.

(2 marks)

f. Explain each line of the code and briefly describe the function of the thread involved in this sample code (10 marks)

1. ReentrantReadWriteLock lock = new ReentrantReadWriteLock();
   -This line creates a lock that can be used for both reading and writing. It helps manage how threads access shared data.

2. Lock readLock = lock.readLock();
   -This is the lock for reading. Many threads can use this at the same time to read data.

3. Lock writeLock = lock.writeLock();
   -This is the lock for writing. Only one thread can use this at a time to change data.

4. double balance = 0;
   -This is the variable to store money (balance). It starts from 0.

5. public void deposit(double amount)
   -This method is used to add money to the balance.

6. writeLock.lock();
   -Before adding money, we lock the data so only one thread can change it.

7. balance += amount;
   -Add the amount to the current balance.

8. writeLock.unlock();
   -After finishing, we unlock so other threads can use it. This is usually in a finally block to make sure it always unlocks.

9. public void withdraw(double amount)
   -This method is used to take out money from the balance.

10. balance -= amount;
    -To subtract the amount from the balance.

11. public double getBalance()
    -This method is to check the current balance.

12. readLock.lock(); and readLock.unlock();
    -We lock before reading and unlock after reading. Many threads can read at the same time.

(10 marks)

Total Marks: 25 marks

Plagiarism
No mark will be given for plagiarism activities

Submission:

Platform: 1. Online Learning - Sample Coding & Output in PdF form
2. GitHub – Upload the coding file to your GitHub account.

Date: 9 Jun 2025 (Monday, before 23.59 pm)