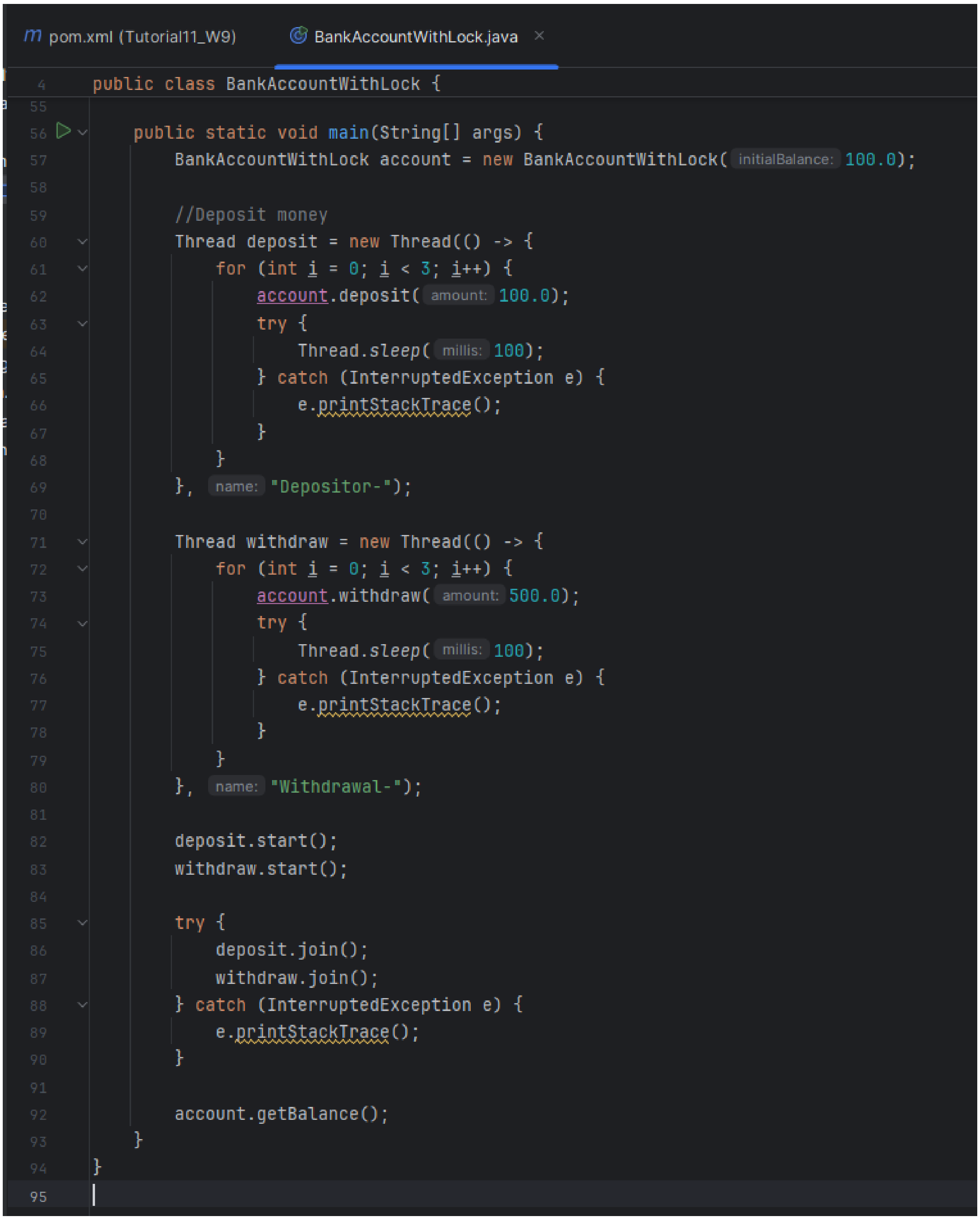
STIWK3014 REAL TIME PROGRAMMING
Tutorial / Exercise 11: ReentrantReadWriteLock () Total 15 Marks

```
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

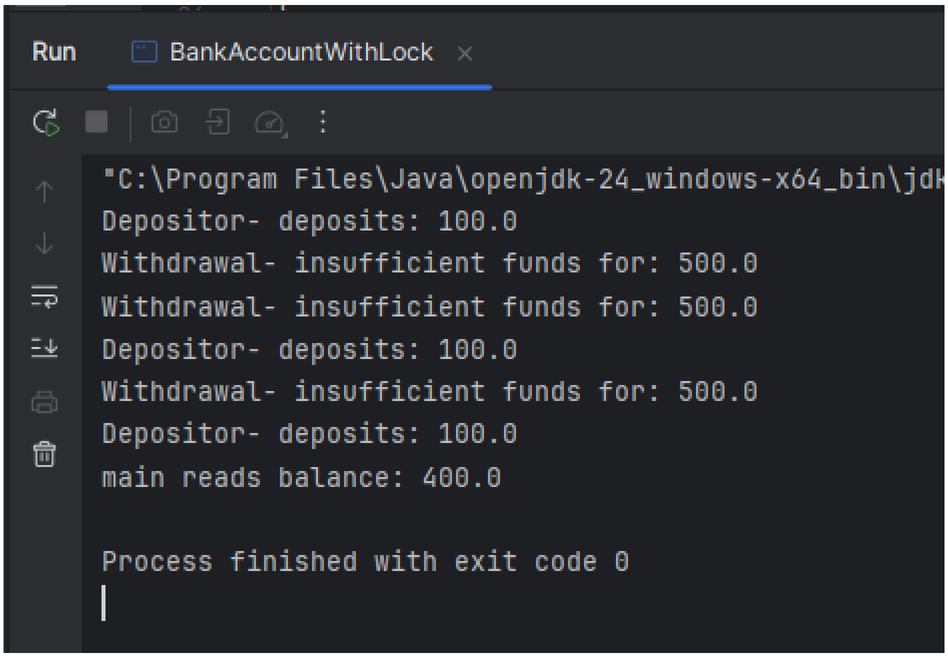
```
public static void main(String[] args) {
    BankAccountWithLock account = new BankAccountWithLock(100.0);

    //Deposit money
    Thread deposit = new Thread(() -> {
      for (int i = 0; i < 3; i++) {
        account.deposit(100.0);
        try {
          Thread.sleep(100);
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }
    }, "Depositor-");

    Thread withdraw = new Thread(() -> {
      for (int i = 0; i < 3; i++) {
        account.withdraw(500.0);
        try {
          Thread.sleep(100);
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }
    }, "Withdrawal-");

    deposit.start();
    withdraw.start();

    try {
      deposit.join();
      withdraw.join();
    } catch (InterruptedException e) {
      e.printStackTrace();
    }

    account.getBalance();
  }
}
```


m pom.xml (Tutorial11_W9)   BankAccountWithLock.java ×

```java
public class BankAccountWithLock {

    public static void main(String[] args) {
        BankAccountWithLock account = new BankAccountWithLock( initialBalance: 100.0);

        //Deposit money
        Thread deposit = new Thread(() -> {
            for (int i = 0; i < 3; i++) {
                account.deposit( amount: 100.0);
                try {
                    Thread.sleep( millis: 100);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        },  name: "Depositor-");

        Thread withdraw = new Thread(() -> {
            for (int i = 0; i < 3; i++) {
                account.withdraw( amount: 500.0);
                try {
                    Thread.sleep( millis: 100);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        },  name: "Withdrawal-");

        deposit.start();
        withdraw.start();

        try {
            deposit.join();
            withdraw.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        account.getBalance();
    }
}
```

(5 marks)

b. What is the output of this program?



```
"C:\Program Files\Java\openjdk-24_windows-x64_bin\jdk
Depositor- deposits: 100.0
Withdrawal- insufficient funds for: 500.0
Withdrawal- insufficient funds for: 500.0
Depositor- deposits: 100.0
Withdrawal- insufficient funds for: 500.0
Depositor- deposits: 100.0
main reads balance: 400.0

Process finished with exit code 0
```

(3 marks)

c. What is the advantage of using `ReentrantReadWriteLock` over `synchronized` method in this program?
Using ReentrantReadWriteLock allows multiple thread to read the balance simultaneously without blocking each other. While Synchronized method allows only one thread at a time to access the method, causing blocking even during reads.

(2 marks)

d. Explain the difference between `readLock()` and `writeLock()`.
readLock() – allows multiple threads to acquire the lock concurrently as long as no thread are holding the write lock.
writeLock() – is exclusive, where it can only hold one thread at a time and preventing others from reading or writing while the lock is being held.

(3 marks)

e. Why is `writeLock.unlock()` placed in a `finally` block?
It is because finally block to guarantee that the lock is always released, even if an exception occurs during the locking section. This is to prevent the deadlock and ensures other threads can acquire the lock and proceed.

(2 marks)

Submission:

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

Date: 21 May 2025 (Wednesday, before 23.59 pm)