UUM

Universiti Utara Malaysia

UNIVERSITI UTARA MALAYSIA

SECOND SEMESTER SESSION A242

STIWK 3014 REAL TIME PROGRAMMING

(GROUP A)

Assignment 2

Lecturer: Dr. Ruzita binti Ahmad

Name: Chong Mun Kei

Matric No: 298767

Github Link: https://github.com/Chong0508/RealTime.git

Source Code

```
package Assignment2;


import java.util.Scanner;
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();
        }
    }

    public static void main(String[] args) {
        BankAccountWithLock account = new
BankAccountWithLock(1000.00);
        Scanner scan = new Scanner(System.in);
        boolean operation = true;

        while (operation) {
            System.out.println("\n1. Read balance");
            System.out.println("2. Deposit");
            System.out.println("3. Withdraw");
            System.out.println("4. Exit");

            System.out.print("Enter your choice: ");
            if (scan.hasNextInt()) {
                int choice = scan.nextInt();

                switch (choice) {
                    case 1: {
                        Thread reader = new Thread(() -> {
                            account.getBalance();
                        }, "Reader-1");
                        reader.start();
                        try {
                            reader.join();
```

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

                    case 2: {
                        System.out.print("Enter your deposit
amount: ");
                        if (scan.hasNextDouble()) {
                            double depositAmount =
scan.nextDouble();
                            Thread depositor = new Thread(() -> {
                                account.deposit(depositAmount);
                            }, "Depositor-1");
                            depositor.start();
                            try {
                                depositor.join();
                            } catch (InterruptedException e) {
                                e.printStackTrace();
                            }
                        }
                        break;
                    }

                    case 3: {
                        System.out.print("Enter your withdraw
amount: ");
```

```
                        if (scan.hasNextDouble()) {
                            double withdrawAmount =
scan.nextDouble();
                            Thread withdrawer = new Thread(() ->
{
                                account.withdraw(withdrawAmount);
                            }, "Withdrawer-1");

                            withdrawer.start();
                            try {
                                withdrawer.join();
                            } catch (InterruptedException e) {
                                e.printStackTrace();
                            }
                        }
                        break;
                    }

                    case 4:
                        operation = false;
                        System.out.println("Exiting...");
                        break;

                    default:
                        System.out.println("Invalid choice.");
                }
            } else {
```

```
                System.out.println("Please enter a valid
number.");
                scan.next();
            }
        }
        scan.close();
    }
}
```

Question

a. **Create the main class for this program(5 marks)**

```
public static void main(String[] args) {
   BankAccountWithLock account = new
BankAccountWithLock(1000.00);
   Scanner scan = new Scanner(System.in);
   boolean operation = true;

   while (operation) {
       System.out.println("\n1. Read balance");
       System.out.println("2. Deposit");
       System.out.println("3. Withdraw");
       System.out.println("4. Exit");

       System.out.print("Enter your choice: ");
       if (scan.hasNextInt()) {
           int choice = scan.nextInt();

           switch (choice) {
               case 1: {
                   Thread reader = new Thread(() -> {
                       account.getBalance();
                   }, "Reader-1");
                   reader.start();
                   try {
                       reader.join();
                   } catch (InterruptedException e) {
                       e.printStackTrace();
```

```
                    }
                    break;
                }

                case 2: {
                    System.out.print("Enter your deposit
amount: ");
                    if (scan.hasNextDouble()) {
                        double depositAmount =
scan.nextDouble();
                        Thread depositor = new Thread(() ->
{
                            account.deposit(depositAmount);
                        }, "Depositor-1");
                        depositor.start();
                        try {
                            depositor.join();
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                    break;
                }

                case 3: {
                    System.out.print("Enter your withdraw
amount: ");
                    if (scan.hasNextDouble()) {
```

```
                        double withdrawAmount =
scan.nextDouble();
                        Thread withdrawer = new Thread(() ->
{

account.withdraw(withdrawAmount);
                        }, "Withdrawer-1");

                        withdrawer.start();
                        try {
                            withdrawer.join();
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                    break;
                }

                case 4:
                    operation = false;
                    System.out.println("Exiting...");
                    break;

                default:
                    System.out.println("Invalid choice.");
            }
        } else {
            System.out.println("Please enter a valid
number.");
```

```
            scan.next();
        }
    }
    scan.close();
}
```

b. What is the output of this program? (3 marks)

```
1. Read balance
2. Deposit
3. Withdraw
4. Exit
Enter your choice: 1
Reader-1 reads balance: 1000.0

1. Read balance
2. Deposit
3. Withdraw
4. Exit
Enter your choice: 2
Enter your deposit amount: 200
Depositor-1 deposits: 200.0

1. Read balance
2. Deposit
3. Withdraw
4. Exit
Enter your choice: 1
Reader-1 reads balance: 1200.0
```

```
1. Read balance
2. Deposit
3. Withdraw
4. Exit
Enter your choice: 3
Enter your withdraw amount: 300
Withdrawer-1 withdraws: 300.0

1. Read balance
2. Deposit
3. Withdraw
4. Exit
Enter your choice: 1
Reader-1 reads balance: 900.0
```

```
1. Read balance
2. Deposit
3. Withdraw
4. Exit
Enter your choice: 2
Enter your deposit amount: 300
Depositor-1 deposits: 300.0

1. Read balance
2. Deposit
3. Withdraw
4. Exit
Enter your choice: 1
Reader-1 reads balance: 1200.0
```

```
1. Read balance
2. Deposit
3. Withdraw
4. Exit
Enter your choice: 3
Enter your withdraw amount: 400
Withdrawer-1 withdraws: 400.0

1. Read balance
2. Deposit
3. Withdraw
4. Exit
Enter your choice: 1
Reader-1 reads balance: 800.0

1. Read balance
2. Deposit
3. Withdraw
4. Exit
Enter your choice: 4
Exiting...
```

c. What is the advantage of using ReentrantReadWriteLock over synchronized method in this program? (2 marks)

ReetrantReadWriteLock allows multiple threads to read concurrently (shared lock), which improves performance when there are frequent read operations. In contrast, synchronized blocks allow only one thread at a time whether read or write, reducing concurrency.

d. Explain the difference between readLock() and writeLock(). (3 marks)

- readLock() provides a shared lock, allowing multiple threads to read the data concurrently. writeLock() provides an exclusive lock, allowing only one thread to write and blocking all other read/write operations until it is released.
- readLock() is used for non-modifying operations such as reading data while writeLock() is used for modifying operations such as deposit and withdraw.
- readLock() improves performance by allowing concurrent reading. writeLock() ensures data consistency by blocking all other operations during write.

e. Why is writeLock.unlock() placed in a finally block? (2 marks)

Placing writeLock.unlock() in the finally block ensures that the lock is always released, even if an exception occurs in the try block. This prevents deadlock when other threads will be permanently blocked from accessing the method and ensures proper program flow.

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

```
import java.util.Scanner;

import java.util.concurrent.locks.ReentrantReadWriteLock;

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

Imports Scanner for reading user input. Imports ReentrantReadWriteeLock and Lock for thread-safe read/write operations.

```
private double balance;

private final ReentrantReadWriteLock lock = new
ReentrantReadWriteLock();

private final Lock readLock = lock.readLock();

private final Lock writeLock = lock.writeLock();
```

Variable balance is used to store the account balance.

It creates a read-write lock for thread-safe access. readLock allows multiple threads to read simultaneously. writeLock allows only one thread to write either deposit or withdraw at a time.

```
public BankAccountWithLock(double initialBalance) {

    this.balance = initialBalance;

}
```

Initializes the account with a starting balance.

```
// Read balance (shared lock)

public double getBalance() {

    readLock.lock();

    try {

        System.out.println(Thread.currentThread().getName()
+ " reads balance: " + balance);

        return balance;

    } finally {

        readLock.unlock();

    }

}
```

Acquires a shared lock so multiple reads can happen concurrently. Print and return the current balance. Ensures the lock is released even if an exception occurs.

```
// 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();
   }
}
```

Add money to the balance with thread safety.

```
// 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();
```

```
    }
}
```

Safety deducts money if the balance is sufficient.

```
BankAccountWithLock account = new
BankAccountWithLock(1000.00);

Scanner scan = new Scanner(System.in);

boolean operation = true;
```

Creates a bank account with an initial balance of 1000.00. Prepares for reading user input. Variable operation controls the menu loop.

```
while (operation) {

    System.out.println("\n1. Read balance");

    System.out.println("2. Deposit");

    System.out.println("3. Withdraw");

    System.out.println("4. Exit");
```

Keeps running until the user chooses to exit for a while loop. Display menu options for every loop.

```
if (scan.hasNextInt()) {

    int choice = scan.nextInt();
```

Checks if the input is a valid integer.

```
case 1: {
    Thread reader = new Thread(() -> {
        account.getBalance();
    }, "Reader-1");
    reader.start();
    try {
        reader.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    break;
}
```

Creates a new thread named “Reader-1” to read the balance. join() method ensures the main thread waits until the reader thread finishes.

```
case 2: {
    System.out.print("Enter your deposit amount: ");
    if (scan.hasNextDouble()) {
        double depositAmount = scan.nextDouble();
        Thread depositor = new Thread(() -> {
            account.deposit(depositAmount);
        }, "Depositor-1");
        depositor.start();
        try {
            depositor.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
```

```
    }
    break;
}
```

Creates a thread to perform deposit operations.

```
case 3: {
    System.out.print("Enter your withdraw amount: ");
    if (scan.hasNextDouble()) {
        double withdrawAmount = scan.nextDouble();
        Thread withdrawer = new Thread(() -> {
            account.withdraw(withdrawAmount);
        }, "Withdrawer-1");

        withdrawer.start();
        try {
            withdrawer.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    break;
}
```

Creates a thread to perform a withdrawal operation.

```
case 4:
    operation = false;
    System.out.println("Exiting...");
    break;
```

Exits the loop and ends the program.

```
        default:
            System.out.println("Invalid choice.");
    }
} else {
    System.out.println("Please enter a valid number.");
    scan.next();
}
```

Ensures the program does not crash with invalid input.

```
        scan.close();
    }
}
```

Closes the Scanner to avoid resource leaks.