Name: K Srikanth Roll Number: 17ETCS002124

Laboratory 5

Title of the Laboratory Exercise: ATM application

1. Introduction and Purpose of Experiment

Aim and Objectives

Aim

- To develop programs to maintain state consistency

2. Experimental Procedure

i. Analyse the problem statement

ii. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code

iii. Implement the algorithm in Java language

iv. Compile the Java program

v. Test the implemented program

vi. Document the Results

vii. Analyse and discuss the outcomes of your experiment

3. Question

Create a multithreaded Java program (min 3 threads) with the following operations

- Balance Enquiry
- Deposit(int x)
- Withdrawal(int y)

4. Computations/Algorithms

1. Create an atomic integer called balance
2. Take an option from user to withdraw, deposit or enquire balance.
3. Create 3 inner classes inside Appclass.The inner classes extend fromThreadclass
    1. If user presses 1 then create a thread to call Enquire_balance.
    2. Balance_enquiry.start() starts execution of the thread.
    3. In Enquiry class use balance.get() to get current balance.
    4. public int get() - Gets the current value.
4. If user presses 2 then create a thread to call withdraw
    1. w.start() starts execution of the thread.
    2. public void set(int newValue) - Sets to the given value.
    3. In Withdraw class use balance.get() to get current balance.
    4. If the amount requested to be withdrawn is less than or equal to current balance the set the new balance by reducing the withdrawal amount from the current balance.
    5. If requested withdrawal amount more than current balance then alert the user.
1. If user presses 3 then create a thread to deposit
    1. public int addAndGet(int delta) - Atomically adds the given value to the current value.
    2. Print balance after deposit

5. Presentation of Results

Java Code

```java
import java.util.*;
import java.util.concurrent.atomic.AtomicInteger;

public class App {
    // AtomicInteger supports atomic operations on underlying int variable.
    static AtomicInteger balance = new AtomicInteger(200);
    Run | Debug
    public static void main(String[] args) throws Exception {
        while (true) {
            Scanner sc = new Scanner(System.in);    Resource leak: 'sc' is never
            System.out.println("1. Press 1 for balance enquiry"
            + "\n2. Press 2 for withdrawal\n3. Press 3 for deposit");
            int choice = sc.nextInt(); // take user input for the selected option

        if (choice == 1) {
            try {
                // create an object of class Equiry
                Enquiry balance_enquiry = new Enquiry();
                balance_enquiry.start();// the thread starts execution
                balance_enquiry.join();
            } catch (Exception ex) {
                System.out.println(ex);
            }
        } else if (choice == 2) {
            try {
                // create an object of class Withdraw
                Withdraw w = new Withdraw();
                w.start(); // the thread w starts execution
                w.join();
            } catch (Exception ex) {
                System.out.println(ex);
            }

```

Figure 1 Java Code for the given problem statement

```
33          } else if (choice == 3) {
34              try {
35                  // create an object of class Deposit
36                  Deposit d = new Deposit();
37                  d.start(); // the thread d starts execution
38                  d.join();
39              } catch (Exception ex) {
40                  System.out.println(ex);
41              }
42          } else {
43              System.out.println("Please press an option from the above list.");
44          }
45
46      }
47
48      private static class Enquiry extends Thread {
49          @Override
50          public void run() {
51              try {
52              // public int get() - Gets the current value.
53              System.out.println("Your current balance is: " + balance.get());
54              } catch (Exception ex) {
55                  System.out.println(ex);
56              }
57          }
58      }
59
```

Figure 2 Java Code for the given problem statement (Continued)

```
60      private static class Withdraw extends Thread {
61          @Override
62          public void run() {
63              Scanner sc = new Scanner(System.in);    Resource leak: 'sc' is never
64              try {
65                  System.out.println("Enter the amount to withdraw");
66                  int withdraw_amount = sc.nextInt();
67                  // public void set(int newValue) - Sets to the given value.
68                  // public int get() - Gets the current value.
69                  int currentBalance = balance.get(); // get the current balance
70                  // check if withdrawal amount is less than current balance
71                  if (currentBalance - withdraw_amount >= 0) {
72                      balance.set(currentBalance - withdraw_amount);
73                      System.out.println("Withdrawal Successful");
74                      System.out.println("Your balance is :Rs." + balance.get());
75                  } else
76                      System.out.println("You only have Rs." + currentBalance +
77                      " in your account.");
78              } catch (Exception ex) {
79                  System.out.println(ex);
80              }
81          }
82      }
82      }
83      private static class Deposit extends Thread {
84          @Override
85          public void run() {
86              Scanner sc = new Scanner(System.in);    Resource leak: 'sc' is ne
87              try {
88                  System.out.println("Enter the amount to deposit");
89                  int deposit_amount = sc.nextInt();
90                  // public int addAndGet(int delta) - Atomically adds the
91                  // given value to the current value.
92                  System.out.println(
93                          "Deposit Successful." +
94                          "\nYour current balance is: Rs."
95                          + balance.addAndGet(deposit_amount));
96              } catch (Exception ex) {
97                  System.out.println(ex);
98              }
99          }
100     }
101 }
```

Figure 3 Java Code for the given problem statement (Continued)

Java Result

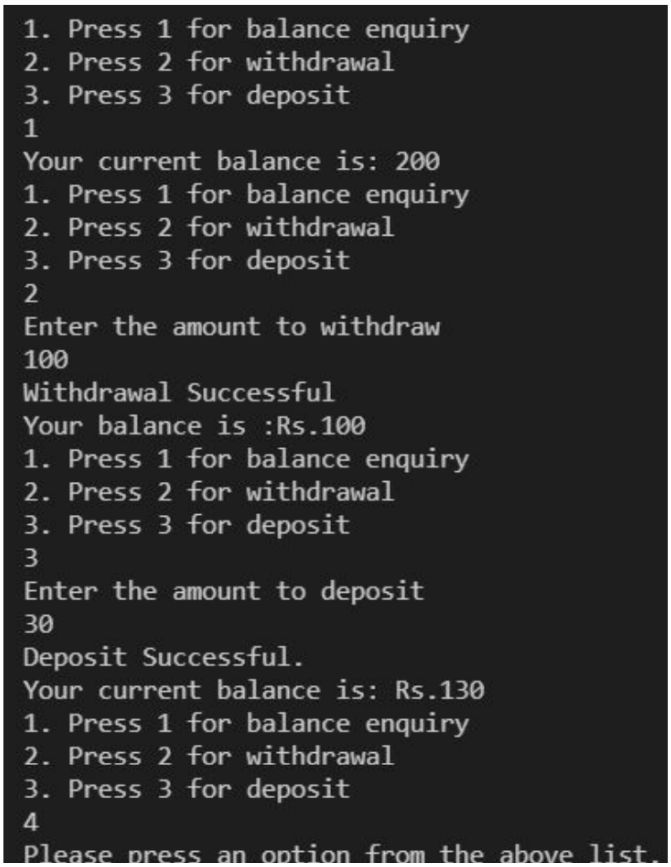

Figure 4 Java Program Output for the given problem statement

6. Analysis and Discussions

Atomic Integer is thread safe (in fact, all classes from java.util.concurrent.atomic package are thread safe), while normal integers are NOT thread-safe. You would require 'synchronized' & 'volatile' keywords, when you are using an 'Integer' variable in multi-threaded environment (to make it thread safe) whereas with atomic integers you don't need 'synchronized' & 'volatile' keywords as atomic integers take care of thread safety.

Name: K Srikanth Roll Number: 17ETCS002124