# 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

1. Experimental Procedure
   * 1. Analyse the problem statement
     2. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
     3. Implement the algorithm in Java language
     4. Compile the Java program
     5. Test the implemented program
     6. Document the Results
     7. Analyse and discuss the outcomes of your experiment
2. Question

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

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

1. **Computations/Algorithms**
2. Create an atomic integer called balance
3. Take an option from user to withdraw, deposit or enquire balance.
4. 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.
5. 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 If the amount requested to be withdrawn is less than or equal to current balance the set the new balance by reducing the withdrawal amount from the current balance.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.
6. 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
7. **Presentation of Results**

**Java Code**

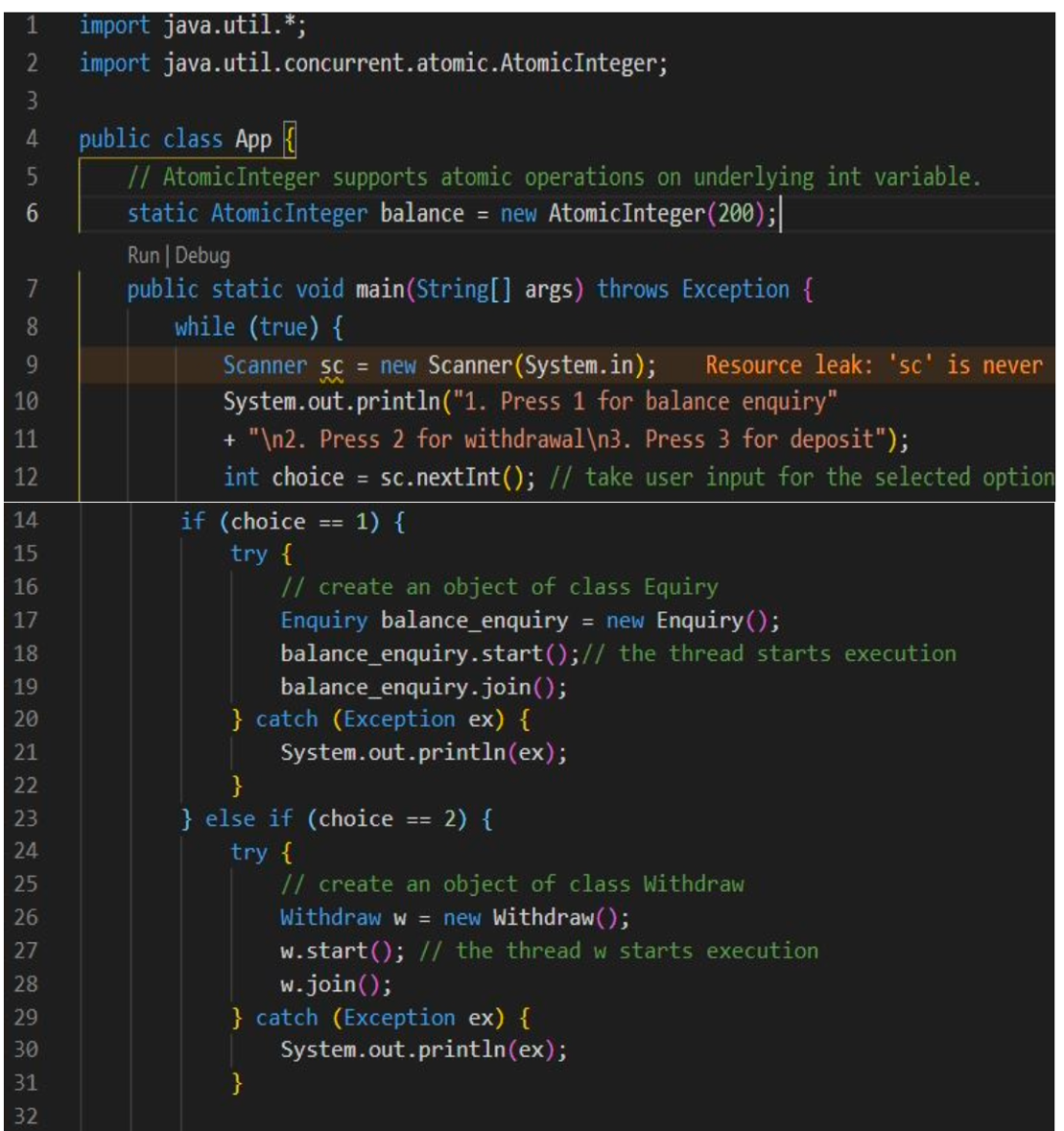
****

Figure 1 Java Code for the given problem statement

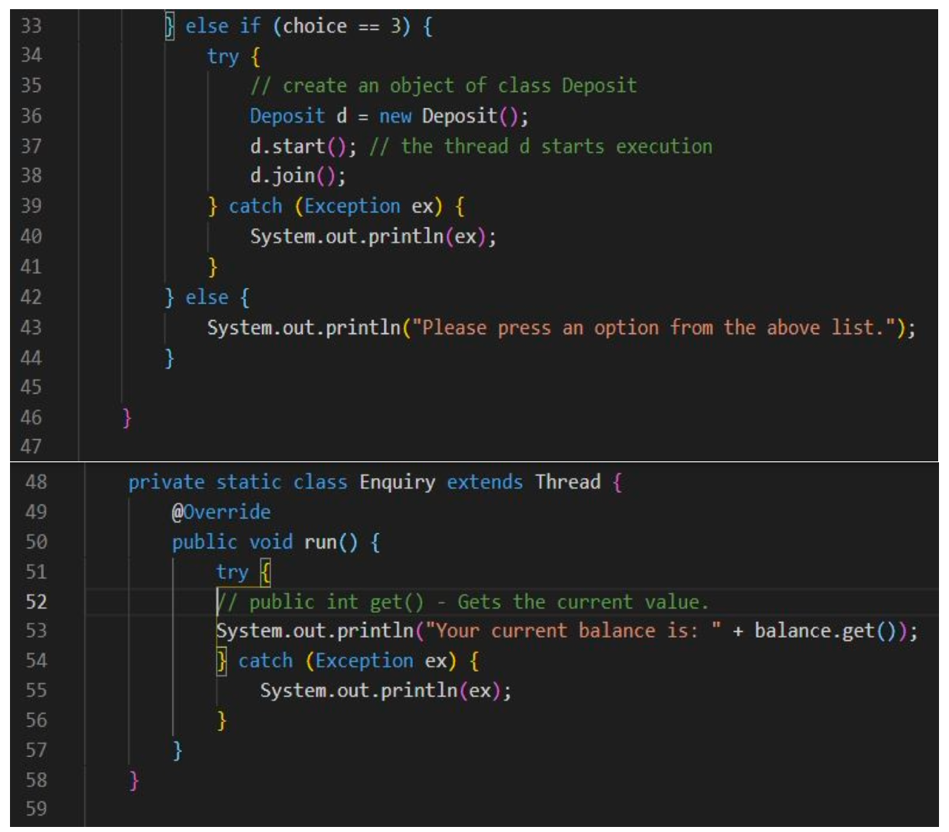
****

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

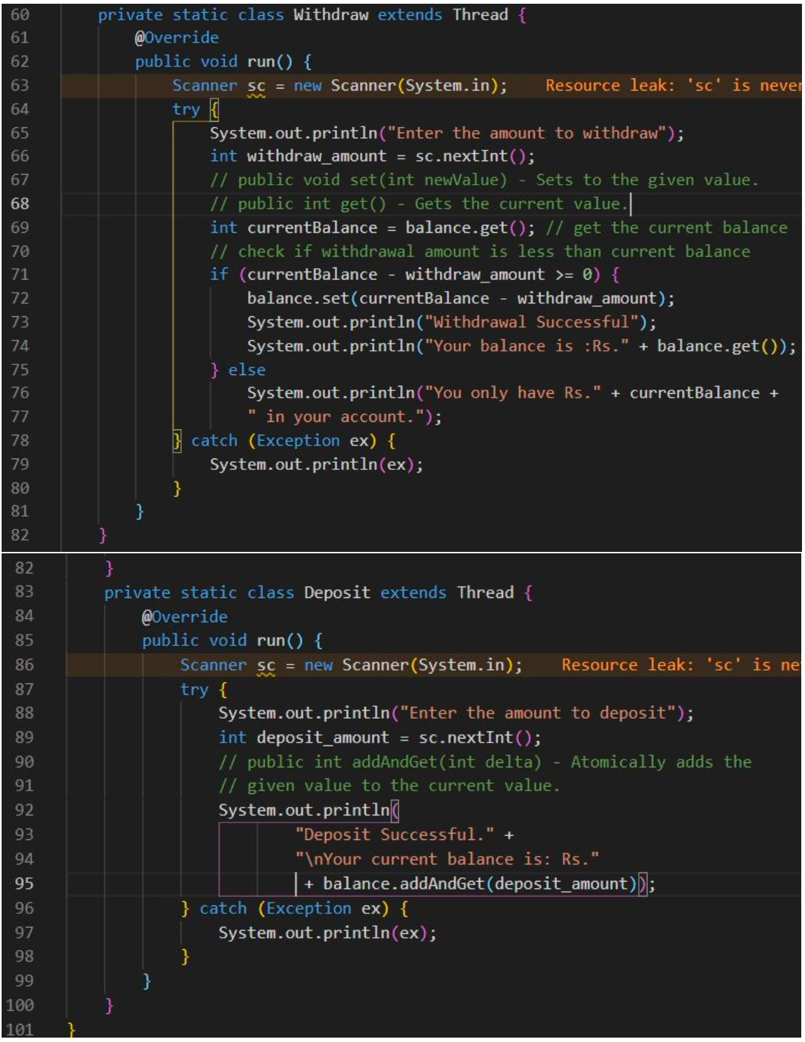
****

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

**Java Result**

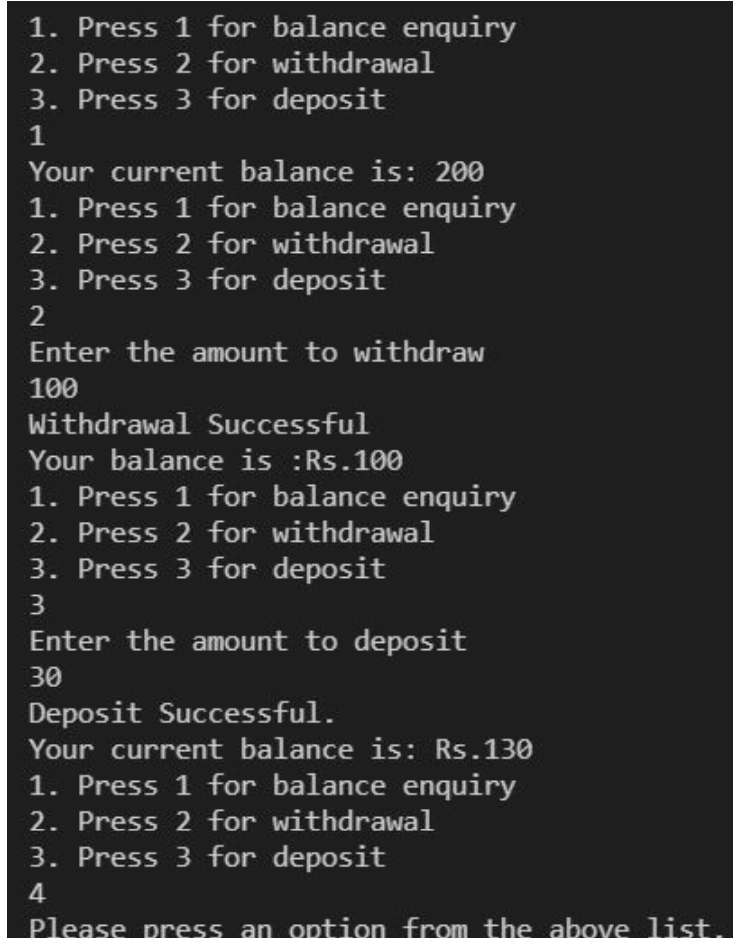
****

Figure Java Program Output for the given problem statement

1. **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.