1. FizzBuzz with Multithreading You are required to implement a class named FizzBuzz that facilitates the printing of a series based on specific divisibility rules in a multi-threaded environment. This class will be used in conjunction with four separate threads, each responsible for printing a different type of output according to the rules outlined below. Requirements: 1. Class Definition: o FizzBuzz(int n): The constructor initializes the FizzBuzz object with an integer n, which represents the total number of elements in the sequence that should be printed (from 1 to n). 2. Output Functions: o void fizz(Runnable printFizz): This method should be called by a thread to print the word "fizz". o void buzz(Runnable printBuzz): This method should be called by a thread to print the word "buzz". o void fizzbuzz(Runnable printFizzBuzz): This method should be called by a thread to print the word "fizzbuzz". o void number(Runnable printNumber): This method should be called by a thread to print the current integer. 3. Output Rules: For each integer iii (1-indexed) in the range from 1 to n: o Print "fizzbuzz" if iii is divisible by both 3 and 5. o Print "fizz" if iii is divisible by 3 but not by 5. o Print "buzz" if iii is divisible by 5 but not by 3. o Print iii itself if it is not divisible by either 3 or 5. 4. Thread Behavior: o You will have four threads: ♣ Thread A: Calls fizz(). ♣ Thread B: Calls buzz(). ♣ Thread C: Calls fizzbuzz(). ♣ Thread D: Calls number(). o These threads should operate in a synchronized manner to ensure the correct output sequence is maintained. Implementation Details: • Each thread should wait for its turn to print its respective output based on the defined rules. • You need to manage the coordination between the threads to ensure that they output in the correct order according to the rules above. • Use appropriate synchronization techniques (like wait() and notify()) to achieve this

PROGRAM:

A screenshot of a computer

Description automatically generated

OUTPUT:

A screenshot of a computer

Description automatically generated

2) Bank Account with Synchronized Methods Implement a simple banking system where multiple threads can deposit and withdraw money from a shared bank account. Description: • Create a BankAccount class with synchronized methods deposit() and withdraw(). • Use these methods to ensure that money is not double-withdrawn when two threads try to withdraw simultaneously. • Simulate multiple threads attempting to deposit and withdraw money concurrently. Key Concepts: • Use of synchronized keyword to ensure thread safety. • Demonstrate thread safety by observing the balance before and after concurrent operations.

PROGRAM:

A screenshot of a computer

Description automatically generated

OUTPUT:

A screenshot of a computer

Description automatically generated

3) Synchronization Using Locks Build a banking application where multiple threads represent different bank accounts accessing a shared resource (the total balance). • Implementation: o Create a BankAccount class with a method for withdrawing and depositing money. o Use ReentrantLock to synchronize access to the account balance to prevent race conditions. o Demonstrate a scenario where multiple threads try to withdraw funds simultaneously and show how locks ensure thread safety. Key Concepts: • Use of Locks and synchronization. • Avoiding race conditions using ReentrantLock.

PROGRAM:

A screenshot of a computer

Description automatically generated

OUTPUT:

A screenshot of a computer

Description automatically generated