SKILLWEKK-9 AOOP NAME:K.NIHARIKA
ID:2300033494

1)
```
public class FizzBuzz {
  private int n;
  private int current = 1; // Start from 1

  public FizzBuzz(int n) {
    this.n = n;
  }

  // Print "fizz" for numbers divisible by 3 but not 5
  public synchronized void fizz(Runnable printFizz) throws InterruptedException {
    while (current <= n) {
      if (current % 3 == 0 && current % 5 != 0) {
        printFizz.run();
        current++;
        notifyAll(); // Notify all waiting threads
      } else {
        wait(); // Wait until it's this thread's turn
      }
    }
  }

  // Print "buzz" for numbers divisible by 5 but not 3
  public synchronized void buzz(Runnable printBuzz) throws InterruptedException {
    while (current <= n) {
      if (current % 5 == 0 && current % 3 != 0) {
        printBuzz.run();
        current++;
        notifyAll(); // Notify all waiting threads
      } else {
        wait(); // Wait until it's this thread's turn
      }
```

```
    }
}

// Print "fizzbuzz" for numbers divisible by both 3 and 5
public synchronized void fizzbuzz(Runnable printFizzBuzz) throws InterruptedException {
    while (current <= n) {
        if (current % 15 == 0) { // Divisible by both 3 and 5
            printFizzBuzz.run();
            current++;
            notifyAll(); // Notify all waiting threads
        } else {
            wait(); // Wait until it's this thread's turn
        }
    }
}

// Print the current number for numbers not divisible by 3 or 5
public synchronized void number(Runnable printNumber) throws InterruptedException {
    while (current <= n) {
        if (current % 3 != 0 && current % 5 != 0) {
            printNumber.run();
            current++;
            notifyAll(); // Notify all waiting threads
        } else {
            wait(); // Wait until it's this thread's turn
        }
    }
}

public static void main(String[] args) {
    FizzBuzz fizzBuzz = new FizzBuzz(15); // Example: n = 15
```

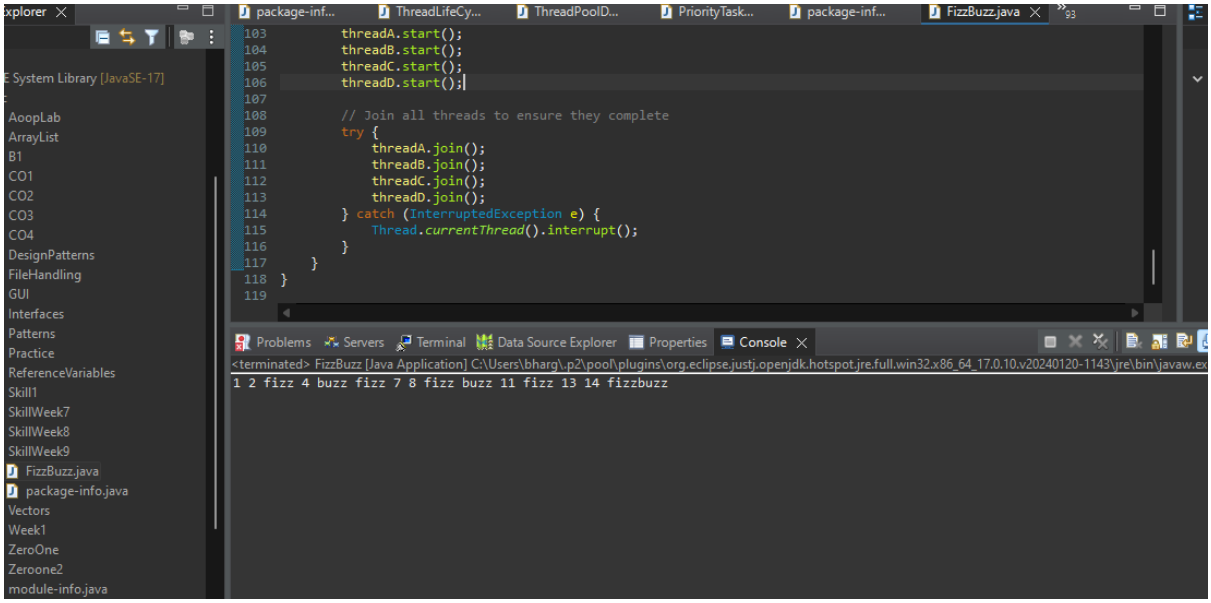
```
// Thread A: For fizz
Thread threadA = new Thread(() -> {
    try {
        fizzBuzz.fizz(() -> System.out.print("fizz "));
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
    }
});

// Thread B: For buzz
Thread threadB = new Thread(() -> {
    try {
        fizzBuzz.buzz(() -> System.out.print("buzz "));
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
    }
});

// Thread C: For fizzbuzz
Thread threadC = new Thread(() -> {
    try {
        fizzBuzz.fizzbuzz(() -> System.out.print("fizzbuzz "));
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
    }
});

// Thread D: For numbers
Thread threadD = new Thread(() -> {
    try {
```

```
            fizzBuzz.number(() -> System.out.print(fizzBuzz.current + " "));
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    });

    // Start all threads
    threadA.start();
    threadB.start();
    threadC.start();
    threadD.start();

    // Join all threads to ensure they complete
    try {
        threadA.join();
        threadB.join();
        threadC.join();
        threadD.join();
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
    }
  }
}
```

OUTPUT:

SKILLWEKK-9 AOOP

NAME:K.NIHARIKA
ID:2300033494



2) class BankAccount {

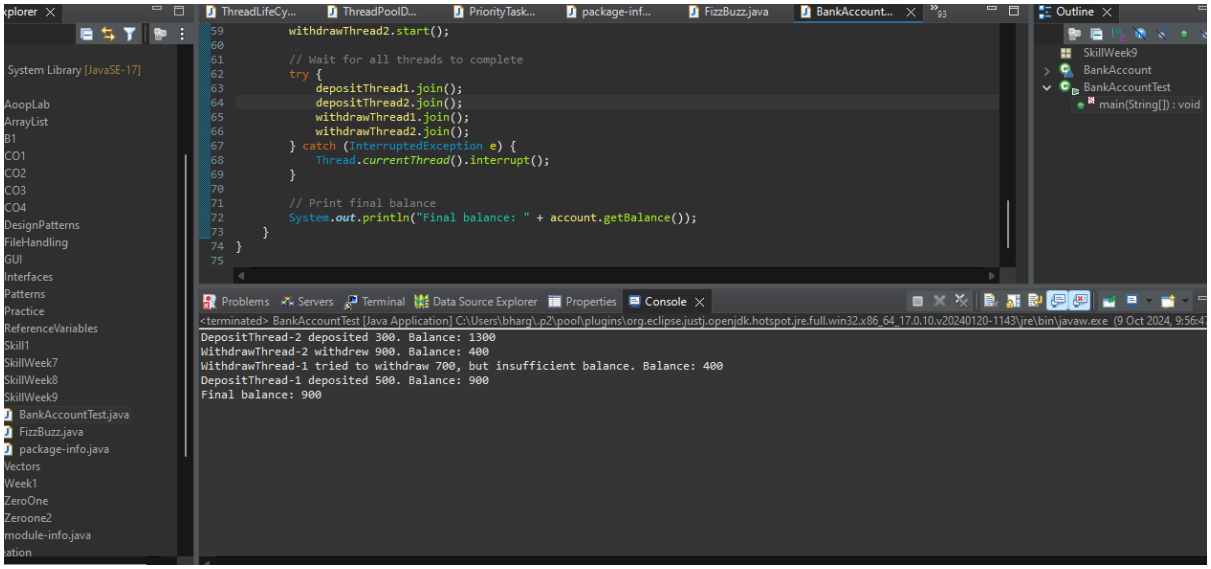
```
    private int balance;

    // Constructor to initialize the balance
    public BankAccount(int balance) {
        this.balance = balance;
    }

    // Synchronized method to deposit money
    public synchronized void deposit(int amount) {
        balance += amount;
        System.out.println(Thread.currentThread().getName() + " deposited " + amount + ". Balance: " + balance);
    }

    // Synchronized method to withdraw money
    public synchronized void withdraw(int amount) {
        if (balance >= amount) {
            balance -= amount;
```

```
        System.out.println(Thread.currentThread().getName() + " withdrew " + amount + ".
Balance: " + balance);
      } else {
        System.out.println(Thread.currentThread().getName() + " tried to withdraw " + amount + ",
but insufficient balance. Balance: " + balance);
      }
    }

    // Get current balance
    public synchronized int getBalance() {
      return balance;
    }
}

public class BankAccountTest {
    public static void main(String[] args) {
      BankAccount account = new BankAccount(1000); // Initial balance

      // Create two threads for depositing
      Thread depositThread1 = new Thread(() -> {
        account.deposit(500);
      }, "DepositThread-1");

      Thread depositThread2 = new Thread(() -> {
        account.deposit(300);
      }, "DepositThread-2");

      // Create two threads for withdrawing
      Thread withdrawThread1 = new Thread(() -> {
        account.withdraw(700);
      }, "WithdrawThread-1");
```

```
        Thread withdrawThread2 = new Thread(() -> {
            account.withdraw(900);
        }, "WithdrawThread-2");

        // Start all threads
        depositThread1.start();
        depositThread2.start();
        withdrawThread1.start();
        withdrawThread2.start();

        // Wait for all threads to complete
        try {
            depositThread1.join();
            depositThread2.join();
            withdrawThread1.join();
            withdrawThread2.join();
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }

        // Print final balance
        System.out.println("Final balance: " + account.getBalance());
    }
}
```

OUTPUT:

SKILLWEKK-9 AOOP NAME:K.NIHARIKA
ID:2300033494



3) import java.util.concurrent.locks.ReentrantLock;

```
public class BankingApplication {

  // BankAccount class
  static class BankAccount {
    private double balance;
    private final ReentrantLock lock = new ReentrantLock(); // Lock for synchronization

    public BankAccount(double initialBalance) {
      this.balance = initialBalance;
    }

    // Method to deposit money
    public void deposit(double amount) {
      lock.lock(); // Acquire the lock
      try {
        balance += amount;
        System.out.println("Deposited: " + amount + ", New Balance: " + balance);
      } finally {
```

```
        lock.unlock(); // Release the lock
      }
    }

    // Method to withdraw money
    public void withdraw(double amount) {
      lock.lock(); // Acquire the lock
      try {
        if (amount <= balance) {
          balance -= amount;
          System.out.println("Withdrew: " + amount + ", New Balance: " + balance);
        } else {
          System.out.println("Insufficient funds for withdrawal of: " + amount);
        }
      } finally {
        lock.unlock(); // Release the lock
      }
    }

    public double getBalance() {
      return balance;
    }
  }

  // Runnable class for simulating bank user actions
  static class BankUser implements Runnable {
    private final BankAccount account;
    private final double amount;

    public BankUser(BankAccount account, double amount) {
      this.account = account;
```

```
        this.amount = amount;
    }

    @Override
    public void run() {
        // Simulate withdrawal
        account.withdraw(amount);
    }
}

public static void main(String[] args) {
    BankAccount account = new BankAccount(1000.0); // Initial balance

    // Create multiple threads trying to withdraw money
    Thread user1 = new Thread(new BankUser(account, 300.0), "User 1");
    Thread user2 = new Thread(new BankUser(account, 500.0), "User 2");
    Thread user3 = new Thread(new BankUser(account, 400.0), "User 3");
    Thread user4 = new Thread(new BankUser(account, 700.0), "User 4");

    // Start the threads
    user1.start();
    user2.start();
    user3.start();
    user4.start();

    // Wait for all threads to finish
    try {
        user1.join();
        user2.join();
        user3.join();
        user4.join();
```

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

        // Final balance
        System.out.println("Final Balance: " + account.getBalance());
    }
}
```

OUTPUT: