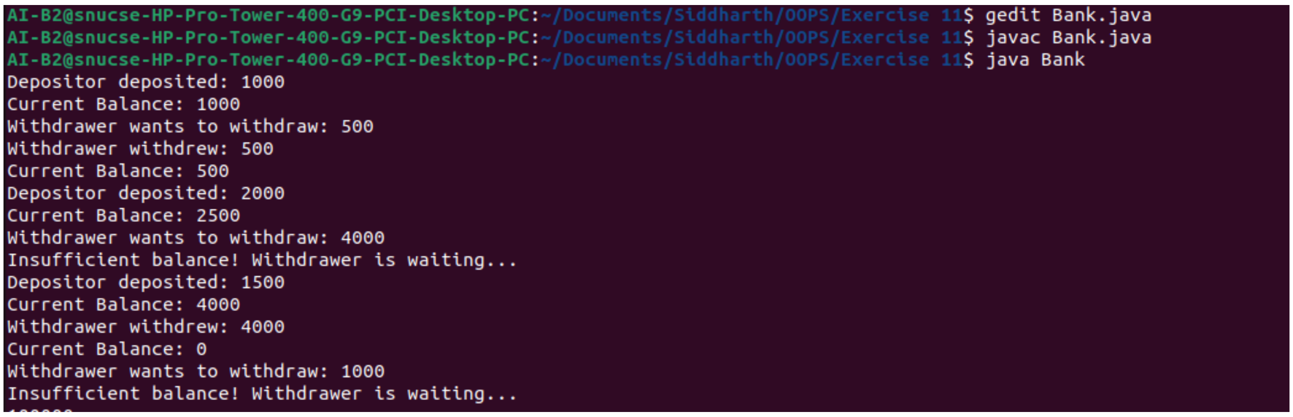
```java
// A simple Bank Account program using Inter-thread Communication
class BankAccount {
    private int balance = 0;

    // Deposit money into the account
    synchronized void deposit(int amount) {
        balance += amount;
        System.out.println(Thread.currentThread().getName() + " deposited: " + amount);
        System.out.println("Current Balance: " + balance);
        notify(); // Notify waiting thread (if any)
    }

    // Withdraw money from the account
    synchronized void withdraw(int amount) {
        System.out.println(Thread.currentThread().getName() + " wants to withdraw: " +
amount);

        // Wait if there isn't enough balance
        while (balance < amount) {
            System.out.println("Insufficient balance! " + Thread.currentThread().getName()
+ " is waiting...");
            try {
                wait(); // Wait until someone deposits money
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }

        // Perform withdrawal
        balance -= amount;
        System.out.println(Thread.currentThread().getName() + " withdrew: " + amount);
        System.out.println("Current Balance: " + balance);
    }
}

class DepositThread extends Thread {
    private BankAccount account;

    DepositThread(BankAccount account) {
        this.account = account;
    }

    public void run() {
        int[] deposits = {1000, 2000, 1500};
        for (int amount : deposits) {
            account.deposit(amount);
            try {
                Thread.sleep(500); // Simulate time between deposits
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }
}

class WithdrawThread extends Thread {
    private BankAccount account;

    WithdrawThread(BankAccount account) {
        this.account = account;
    }

    public void run() {
        int[] withdrawals = {500, 4000, 1000};
        for (int amount : withdrawals) {
            account.withdraw(amount);
            try {
                Thread.sleep(800); // Simulate time between withdrawals
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }
}

public class Bank{
    public static void main(String[] args) {
        BankAccount account = new BankAccount();

        DepositThread depositor = new DepositThread(account);
        WithdrawThread withdrawer = new WithdrawThread(account);

        depositor.setName("Depositor");
        withdrawer.setName("Withdrawer");

        depositor.start();
        withdrawer.start();
    }
}
```

```java
// Class representing the shared resource (Stock)
class Stock {
    private int stock = 0;   // current stock value
    private final int MAX = 5; // maximum stock limit

    // Method for producer to add stock
    synchronized void addStock(int value) {
        // If stock is full, producer must wait
        while (stock >= MAX) {
            System.out.println("Stock is full. Producer is waiting...");
            try {
                wait(); // wait until consumer consumes stock
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }

        stock += value;
        System.out.println("Producer added: " + value + " | Current Stock: " + stock);
        notify(); // Notify consumer that stock is available
    }

    // Method for consumer to get stock
    synchronized void getStock(int value) {
        // If stock is empty, consumer must wait
        while (stock < value) {
            System.out.println("Stock is empty. Consumer is waiting...");
            try {
                wait(); // wait until producer adds stock
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }

        stock -= value;
        System.out.println("Consumer bought: " + value + " | Remaining Stock: " + stock);
        notify(); // Notify producer that space is available
    }
}

// Producer class
class Producer extends Thread {
    private Stock stock;

    Producer(Stock stock) {
        this.stock = stock;
    }

    public void run() {
        for (int i = 1; i <= 6; i++) {
            stock.addStock(1);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }
}

// Consumer class
class Consumer extends Thread {
    private Stock stock;

    Consumer(Stock stock) {
        this.stock = stock;
    }

    public void run() {
        for (int i = 1; i <= 6; i++) {
            stock.getStock(1);
            try {
                Thread.sleep(800);
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }
}

// Main class
public class ProducerConsumer {
    public static void main(String[] args) {
        Stock stock = new Stock();

        Producer producer = new Producer(stock);
        Consumer consumer = new Consumer(stock);

        producer.start();
        consumer.start();
    }
}
```

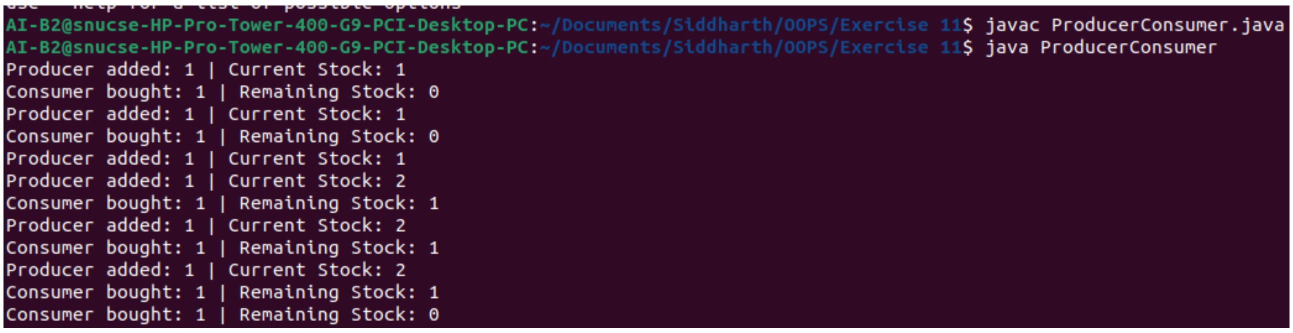
```java
// Program to illustrate Thread Priority
class MyThread extends Thread {
    public MyThread(String name) {
        super(name);
    }

    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(Thread.currentThread().getName() +
                                   " (Priority: " + Thread.currentThread().getPriority() + ")
- Count: " + i);
            try {
                Thread.sleep(500); // pause for clarity
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }
}

public class ThreadPriority{
    public static void main(String[] args) {
        // Create threads
        MyThread t1 = new MyThread("Low Priority Thread");
        MyThread t2 = new MyThread("Normal Priority Thread");
        MyThread t3 = new MyThread("High Priority Thread");

        // Set priorities
        t1.setPriority(Thread.MIN_PRIORITY);   // Priority 1
        t2.setPriority(Thread.NORM_PRIORITY);  // Priority 5 (default)
        t3.setPriority(Thread.MAX_PRIORITY);   // Priority 10

        // Start threads
        t1.start();
        t2.start();
        t3.start();
    }
}
```

1)



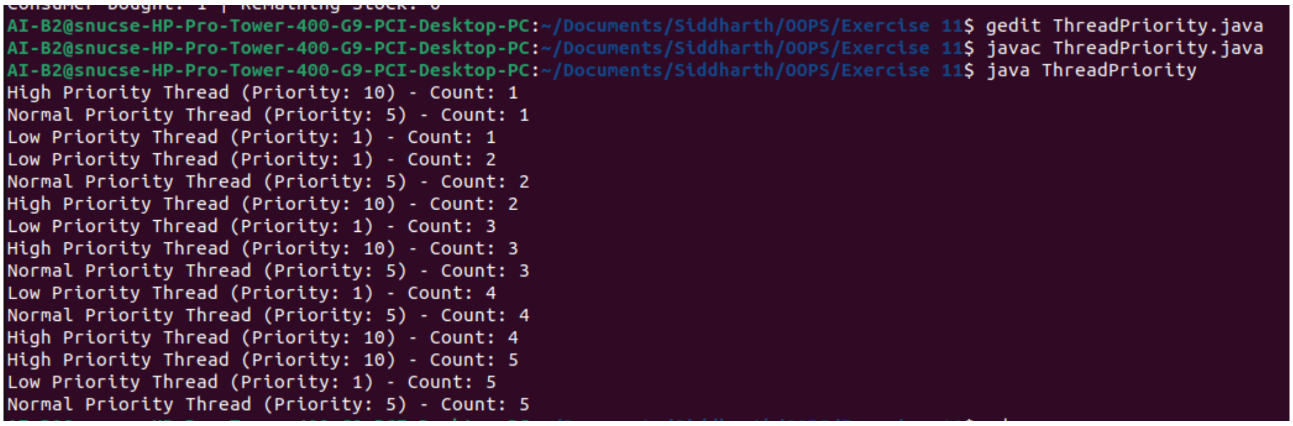
```
AI-B2@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/Documents/Siddharth/OOPS/Exercise 11$ gedit Bank.java
AI-B2@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/Documents/Siddharth/OOPS/Exercise 11$ javac Bank.java
AI-B2@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/Documents/Siddharth/OOPS/Exercise 11$ java Bank
Depositor deposited: 1000
Current Balance: 1000
Withdrawer wants to withdraw: 500
Withdrawer withdrew: 500
Current Balance: 500
Depositor deposited: 2000
Current Balance: 2500
Withdrawer wants to withdraw: 4000
Insufficient balance! Withdrawer is waiting...
Depositor deposited: 1500
Current Balance: 4000
Withdrawer withdrew: 4000
Current Balance: 0
Withdrawer wants to withdraw: 1000
Insufficient balance! Withdrawer is waiting...
```

2)



```
AI-B2@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/Documents/Siddharth/OOPS/Exercise 11$ javac ProducerConsumer.java
AI-B2@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/Documents/Siddharth/OOPS/Exercise 11$ java ProducerConsumer
Producer added: 1 | Current Stock: 1
Consumer bought: 1 | Remaining Stock: 0
Producer added: 1 | Current Stock: 1
Consumer bought: 1 | Remaining Stock: 0
Producer added: 1 | Current Stock: 1
Producer added: 1 | Current Stock: 2
Consumer bought: 1 | Remaining Stock: 1
Producer added: 1 | Current Stock: 2
Consumer bought: 1 | Remaining Stock: 1
Producer added: 1 | Current Stock: 2
Consumer bought: 1 | Remaining Stock: 1
Consumer bought: 1 | Remaining Stock: 0
```

3)



```
AI-B2@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/Documents/Siddharth/OOPS/Exercise 11$ gedit ThreadPriority.java
AI-B2@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/Documents/Siddharth/OOPS/Exercise 11$ javac ThreadPriority.java
AI-B2@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/Documents/Siddharth/OOPS/Exercise 11$ java ThreadPriority
High Priority Thread (Priority: 10) - Count: 1
Normal Priority Thread (Priority: 5) - Count: 1
Low Priority Thread (Priority: 1) - Count: 1
Low Priority Thread (Priority: 1) - Count: 2
Normal Priority Thread (Priority: 5) - Count: 2
High Priority Thread (Priority: 10) - Count: 2
Low Priority Thread (Priority: 1) - Count: 3
High Priority Thread (Priority: 10) - Count: 3
Normal Priority Thread (Priority: 5) - Count: 3
Low Priority Thread (Priority: 1) - Count: 4
Normal Priority Thread (Priority: 5) - Count: 4
High Priority Thread (Priority: 10) - Count: 4
High Priority Thread (Priority: 10) - Count: 5
Low Priority Thread (Priority: 1) - Count: 5
Normal Priority Thread (Priority: 5) - Count: 5
```