Chapter 30. Multithreading & Parallel Programming

Objectives:

To develop task classes by implementing the Runnable interface (§30.3).
To execute tasks in a thread pool (§30.6).
To use synchronized methods or blocks to synchronize threads to avoid race conditions (§30.7)

Problem A

Task:

Implement a simple program that simulates a traffic light with three colors: red, yellow, and green. The traffic light should change colors every few seconds.

Instructions:

Create a TrafficLight class that implements the Runnable interface. This class should have a run() method that implements the logic for changing the traffic light colors.

In the run() method, use Thread.sleep() to pause the program for a few seconds between each color change.

Use a while loop to continuously run the traffic light simulation.

In the main() method, create a new Thread object that takes an instance of the TrafficLight class as a parameter. Start the thread and observe the traffic light simulation.

Requirements:

The traffic light should change colors in the following order: green, yellow, red, green, yellow, red, and so on.

The green light should be displayed for 10 seconds, the yellow light for 2 seconds, and the red light for 5 seconds.

The program should run indefinitely until the user manually stops it.

The program should output the current traffic light color to the console.

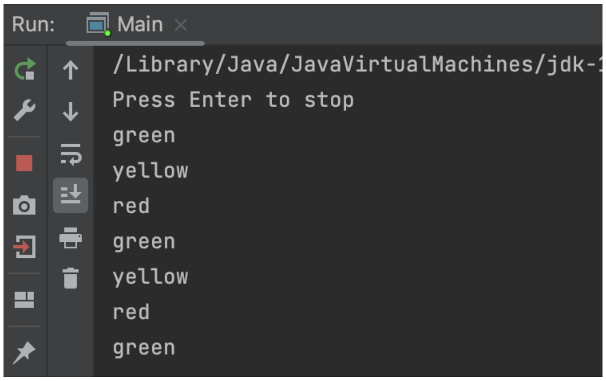
Hints:

Use an integer variable to keep track of the current color index.

Use a switch statement to determine which color to display based on the current index.

Use System.out.println() to output the current color to the console.

Sample output:



```
/Library/Java/JavaVirtualMachines/jdk-
Press Enter to stop
green
yellow
red
green
yellow
red
green
```

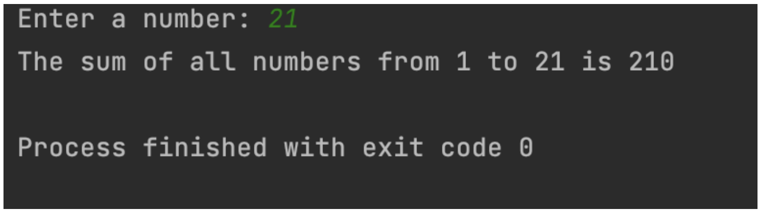
Problem B

Write a Java program that calculates the sum of all numbers from 1 to a given integer using a thread pool.
Create a class called "SumCalculator". A method called "calculateSum" calculates the sum of all numbers from 1 to "number" using a thread pool. The method should do the following:

- Create a thread pool with a fixed size of 5 threads using the Executors.newFixedThreadPool method.
- Divide the range of numbers from 1 to "number" into 5 equal parts.
- Create 5 Runnable objects, each of which calculates the sum of its assigned part of the range.
- Submit the Runnable objects to the thread pool using the execute() method.
- Wait for all the threads to finish using the shutdown() method.
- Add up the individual sums calculated by each thread to get the total sum.

Note: You can use the Thread.sleep() method to simulate a time-consuming calculation in the Runnable objects.

Example output:



```
Enter a number: 21
The sum of all numbers from 1 to 21 is 210

Process finished with exit code 0
```

Java Code:

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class SumCalculator {
   private final int number;
   private int sum;

   public SumCalculator(int number) {
      this.number = number;
   }

   public void calculateSum() {
      //your code
System.out.println("The sum of all numbers from 1 to " + number + " is " + sum);
   }
```

```
    private class SumWorker implements Runnable {
        private final int start;
        private final int end;

        public SumWorker(int start, int end) {
            this.start = start;
            this.end = end;
        }

        @Override
        public void run() {
            //your code
            }
        }
    }
}
```

SumCalculatorApp.java

```
import java.util.Scanner;

public class SumCalculatorApp {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();
        scanner.close();

        SumCalculator calculator = new SumCalculator(number);
        calculator.calculateSum();
    }
}
```

Problem C

You are given a *BankAccount* class that represents a simple bank account. The BankAccount class has two methods: *deposit()* and *withdraw()*, which can be called by multiple threads concurrently. Your task is to use synchronized methods or blocks to synchronize access to the deposit() and withdraw() methods to avoid race conditions and ensure that the account balance is always accurate.

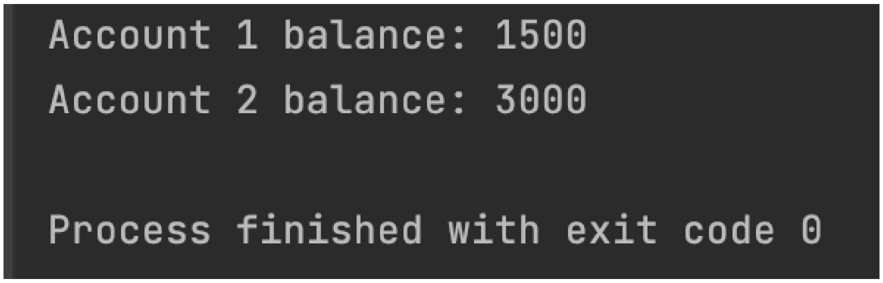
```
public class BankAccount {
    private int balance;

    public BankAccount(int balance) {
        this.balance = balance;
    }

    public int getBalance() {
        return balance;
    }

    public void deposit(int amount) {
        // TODO: synchronize access to this method
        balance += amount;
    }

    public void withdraw(int amount) {
        // TODO: synchronize access to this method
        balance -= amount;
    }
}

public class BankAccountThread extends Thread {
    private final BankAccount account;
    private final boolean isDeposit;
    private final int amount;

    public BankAccountThread(BankAccount account, boolean isDeposit, int
amount) {
        this.account = account;
        this.isDeposit = isDeposit;
        this.amount = amount;
    }

    @Override
    public void run() {
        //your code
}
```

Requirements:

- Use synchronized methods or blocks to synchronize access to the deposit() and withdraw() methods to avoid race conditions.
- Test your program by using class Main:

```
public class Main {
    public static void main(String[] args) throws InterruptedException {
        BankAccount account1 = new BankAccount(1000);
        BankAccount account2 = new BankAccount(2000);

        List<Thread> threads = new ArrayList<>();

        for (int i = 0; i < 10; i++) {
            threads.add(new BankAccountThread(account1, true, 100));
            threads.add(new BankAccountThread(account2, true, 200));
            threads.add(new BankAccountThread(account1, false, 50));
            threads.add(new BankAccountThread(account2, false, 100));
        }

        for (Thread thread : threads) {
            thread.start();
        }

        for (Thread thread : threads) {
            thread.join();
        }

        System.out.println("Account 1 balance: " + account1.getBalance());
        System.out.println("Account 2 balance: " + account2.getBalance());
    }
}
```

Output:



```
Account 1 balance: 1500
Account 2 balance: 3000


Process finished with exit code 0
```