Day 18:

Task 1: Creating and Managing Threads

Write a program that starts two threads, where each thread prints numbers from 1 to 10 with a 1-second delay between each number

```java
package Assignment18;
public class PrintNumber implements Runnable {
    @Override
    public void run() {
    try {
        for (int i = 1; i <= 10; i++) {
            System.out.println(Thread.currentThread().getName() + ":" + 1);
            Thread.sleep( millis: 1000); // 1 second delay
        }
        } catch (InterruptedException e) {

        System.out.println(Thread.currentThread().getName() + "interrupted.");
    }

    }

    public static void main(String[] args) {
        Runnable task = new PrintNumber();

        Thread thread1 = new Thread(task,  name: "Thread-1");
        Thread thread2 = new Thread(task,  name: "Thread-2");

        thread1.start();
        thread2.start();

        try {
            thread1.join();
            thread2.join();
```

```
        try {
            thread1.join();
            thread2.join();

        } catch (InterruptedException e){

        System.out.println("Main thread interrupted.");

    } System.out.println("Both threads have finished.");

}
}
```

Output:

```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.
Thread-2:1
Thread-1:1
Thread-2:1
Thread-1:1
Thread-2:1
Thread-1:1
Thread-2:1
Thread-1:1
Thread-2:1
Thread-1:1
Thread-2:1
Thread-1:1
Thread-2:1
Thread-1:1
Thread-2:1
Thread-1:1
Thread-2:1
Thread-1:1
Thread-1:1
Thread-2:1
Both threads have finished.

Process finished with exit code 0
```

Task 2: States and Transitions

Create a Java class that simulates a thread going through different lifecycle states: NEW, RUNNABLE, WAITING, TIMED_WAITING, BLOCKED, and TERMINATED. Use methods like sleep(), wait(), notify(), and join() to demonstrate these states.

```java
package Assignment18;

class ThreadExample implements Runnable {  2 usages
    @Override
    public void run() {         try {
        Thread.sleep( millis: 1500);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

        System.out.println(
                "State of thread1 while it called join() method on thread2 - " + LifeCycle.thread1.getState()
        );
        try {
            Thread.sleep( millis: 200);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    } }
public class LifeCycle implements Runnable {
    public static Thread thread1;       5 usages
    public static LifeCycle obj;     2 usages
    public static void main(String[] args) {
    obj = new LifeCycle();
    thread1 = new Thread(obj);
    System.out.println("State of thread1 after creating it - " + thread1.getState());
    thread1.start();
    System.out.println("State of thread1 after calling start() method on it - " + thread1.getState());
}     @Override     public void run() {
    ThreadExample myThread = new ThreadExample();
```

```
    Thread thread2 = new Thread(myThread);
    System.out.println("State of thread2 after creating it - " + thread2.getState());
    thread2.start();
    System.out.println("State of thread2 after calling start() method on it - " + thread2.getState());
    try {
        Thread.sleep( millis: 200);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("State of thread2 after calling sleep() method on it - " + thread2.getState());
    try {
        thread2.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("State of thread2 after it finished execution - " + thread2.getState());
}
}
```

Output:

```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition
State of thread1 after creating it - NEW
State of thread1 after calling start() method on it - RUNNABLE
State of thread2 after creating it - NEW
State of thread2 after calling start() method on it - RUNNABLE
State of thread2 after calling sleep() method on it - TIMED_WAITING
State of thread1 while it called join() method on thread2 - WAITING
State of thread2 after it finished execution - TERMINATED

Process finished with exit code 0
```

Task 3: Synchronization and Inter-thread Communication
Implement a producer-consumer problem using wait() and notify() methods to handle the correct processing sequence between threads.

```java
package Assignment18;

class Common {  6 usages
    int num;  4 usages
    boolean available = false;  4 usages

    public synchronized int put(int num) {  1 usage
        synchronized (this) {
            if (available)
        try {
            wait();
        } catch (InterruptedException e) {
            // TODO: handle exception
             e.printStackTrace();
        }
        this.num = num;
        System.out.println("From Prod :" +
                this.num);
        try {
            Thread.sleep( millis: 1000);
        } catch (InterruptedException e) {
            // TODO: handle exception
             e.printStackTrace();
        }
        available = true;            notify();
    }
```

```
        return num;

    }

    public synchronized int get() {         if (!available)  1 usage
        try {
            wait();
        } catch (InterruptedException e) {

            e.printStackTrace();
        }
        System.out.println("From Consumer : " + this.num);

        try {
            Thread.sleep( millis: 1000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            //  e.printStackTrace();
        }
        available = false;          notify();       return num;

    }
```

```
}
class Producer extends Thread {  1 usage
    Common c;  2 usages
    public Producer(Common c) {     this.c = c;  1 usage
        new Thread( task: this,  name: "Producer :").start();
    }
    public void run() {         int x = 0, i = 0;       while (x <= 10) {           c.put(i++);
        x++;
    }
    }

}
class Consumer extends Thread {  1 usage
    Common c;  2 usages

    public Consumer(Common c) {         this.c = c;  1 usage
        new Thread( task: this,  name: "Consumer :").start();
    }

    public void run() {         int x = 0;
        while (x <= 10) {
            c.get();
            x++;
        }
    }
```

```
}
public class ProducerConsumer {
    public static void main(String[] args) {
    // TODO Auto-generated method stub
        Common c = new Common();
        new Producer(c);
        new Consumer(c);


}
}
```

Output:

```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\
From Prod :0
From Consumer : 0
From Prod :1
From Consumer : 1
From Prod :2
From Consumer : 2
From Prod :3
From Consumer : 3
From Prod :4
From Consumer : 4
From Prod :5
From Consumer : 5
From Prod :6
From Consumer : 6
From Prod :7
From Consumer : 7
From Prod :8
From Consumer : 8
From Prod :9
From Consumer : 9
From Prod :10
From Consumer : 10

Process finished with exit code 0
```

Task 4: Synchronized Blocks and Methods

Write a program that simulates a bank account being accessed by multiple threads to perform deposits and withdrawals using synchronized methods to prevent race conditions.

```java
package Assignment18;

public class BankAccountDemo {
    public static void main(String[] args) {    BankAccount account = new BankAccount();
            Thread depositThread1 = new Thread(new
                    DepositTask(account, amount: 100), name: "Deposit Thread1");
            Thread depositThread2 = new Thread(new
                    DepositTask(account, amount: 200), name: "Deposit Thread2");
            Thread withdrawThread1 = new Thread(new
                    WithdrawTask(account, amount: 150), name: "Withdraw Thread1");
            Thread withdrawThread2 = new Thread(new
                    WithdrawTask(account, amount: 50), name: "Withdraw Thread2");

            depositThread1.start();
            depositThread2.start();
            withdrawThread1.start();
            withdrawThread2.start();

            try {
                depositThread1.join();
                depositThread2.join();
                withdrawThread1.join();
                withdrawThread2.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

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

```
}
class BankAccount {     private int balance = 0;  6 usages

    public synchronized void deposit(int amount) {  1 usage
        balance += amount;          System.out.println(
                Thread.currentThread().getName() + " deposited amount " + amount + ", new balance: " + balance);
    }
    public synchronized void withdraw(int amount) {  1 usage
        if (balance >= amount) {        balance -= amount;          System.out.println(
                Thread.currentThread().getName() +
                        " withdrew amount " + amount + ", new balance: " + balance);
        } else {

            System.out.println(Thread.currentThread().getName()
                    + " attempted to withdraw " + amount + ", but insufficient funds. Balance: " + balance);
        }
    }
    public int getBalance() {       return balance;  1 usage
    }
}
class DepositTask implements Runnable {  2 usages
    private final BankAccount account;  2 usages

    private final int amount;  2 usages


        public DepositTask(BankAccount account,int amount)  2 usages
        {
        this.account = account;
        this.amount = amount;
```

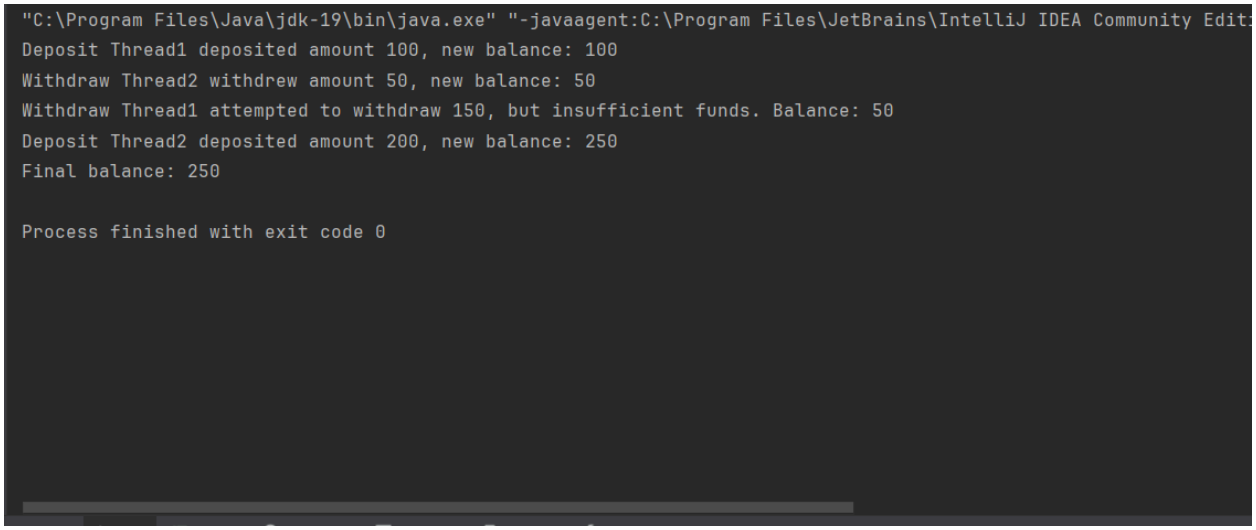
```
    }
    @Override
    public void run() {
        account.deposit(amount);

    }   }

class WithdrawTask implements Runnable {    private final BankAccount account;      private final int amount;  2 usages

    public WithdrawTask(BankAccount account, int amount)  2 usages
    {   this.account = account;     this.amount = amount; }
    @Override   public void run() {
        account.withdraw(amount);
    }
}
```

Output:



Task 5: Thread Pools and Concurrency Utilities
Create a fixed-size thread pool and submit multiple tasks that perform complex calculations or I/O operations and observe the execution.

```java
package Assignment18;

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

public class ThreadPoolDemo {

    public static void main(String[] args) {

        ExecutorService executor = Executors.newFixedThreadPool( nThreads: 4);

        for (int i = 0; i < 10; i++) {    executor.submit(new CalculationTask(i));
        }

        executor.shutdown();

        try {

            if (!executor.awaitTermination( timeout: 1, TimeUnit.HOURS)) {
                executor.shutdownNow();
            }
        } catch (InterruptedException e) {
            executor.shutdownNow();
        }
```

```java
public class ThreadPoolDemo {
    public static void main(String[] args) {

        System.out.println("All tasks have finished.");
    }
}

class CalculationTask implements Runnable {  1 usage
    private final int taskId;   private final Random random = new Random();  4 usages

    public CalculationTask(int taskId) {        this.taskId = taskId;  1 usage
    }

    @Override
    public void run() {
        System.out.println("Task " + taskId + " started.");

        long duration = random.nextInt( bound: 5) + 1;
        try {
            TimeUnit.SECONDS.sleep(duration);
        } catch (InterruptedException e) {
            System.out.println("Task " + taskId + " was interrupted.");
        }

        System.out.println("Task " + taskId + " finished after " + duration + " seconds.");
    }
}
```

Output:

```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community
Task 3 started.
Task 1 started.
Task 0 started.
Task 2 started.
Task 1 finished after 1 seconds.
Task 3 finished after 1 seconds.
Task 4 started.
Task 5 started.
Task 2 finished after 3 seconds.
Task 0 finished after 3 seconds.
Task 6 started.
Task 7 started.
Task 5 finished after 3 seconds.
Task 8 started.
Task 6 finished after 2 seconds.
Task 9 started.
Task 8 finished after 1 seconds.
Task 4 finished after 5 seconds.
Task 7 finished after 4 seconds.
Task 9 finished after 2 seconds.
All tasks have finished.

Process finished with exit code 0
```

Task 6: Executors, Concurrent Collections, CompletableFuture

Use an ExecutorService to parallelize a task that calculates prime numbers up to a given number and then use CompletableFuture to write the results to a file asynchronously.

```
        }

        writePrimesToFileAsync(allPrimes);

        System.out.println("Prime numbers written to file: " + FILE_NAME);
    }

    private static List<Future<List<Integer>>> calculatePrimes(int upperLimit) throws  1 usage
            Exception {
        ExecutorService executor = Executors.newFixedThreadPool(NUM_THREADS);
        List<Future<List<Integer>>> futures = new ArrayList<>();
        int chunkSize = upperLimit / NUM_THREADS;

        for (int i = 0; i < upperLimit; i += chunkSize) {
            int start = i + 1;
            int end = Math.min(i + chunkSize, upperLimit);
            futures.add(executor.submit(() -> findPrimesInRange(start, end)));
        }

        executor.shutdown();         executor.awaitTermination( timeout: 10, TimeUnit.SECONDS);

        return futures;
    }
```

```
private static List<Integer> findPrimesInRange(int start, int end) {  1 usage
    List<Integer> primes = new ArrayList<>();
    for (int num = start; num <= end; num++) {
    if (isPrime(num)) {
        primes.add(num);
    }
}
    return primes;
}

private static boolean isPrime(int num) {  1 usage
    if (num < 2) {             return false;
    }
    for (int i = 2; i <= Math.sqrt(num); i++) {             if (num % i == 0) {                 return false;
    }         }
    return true;
}

private static void writePrimesToFileAsync(List<Integer> primes) throws Exception {  1 usage
    CompletableFuture<Void> writeFuture = CompletableFuture.runAsync(() -> {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(FILE_NAME))) {
            for (int prime : primes) {
    writer.write( str: prime + "\n");
}
```

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

        writeFuture.get();
    }
}
```

```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ
Prime numbers written to file: prime_numbers.txt

Process finished with exit code 0
```

Task 7: Writing Thread-Safe Code, Immutable Objects

Design a thread-safe Counter class with increment and decrement methods. Then demonstrate its usage from multiple threads. Also, implement and use an immutable class to share data between threads.

```java
package Assignment18;

import java.util.concurrent.atomic.AtomicInteger;

class ThreadSafeCounter {  2 usages
    private final AtomicInteger count;  4 usages

    public ThreadSafeCounter() {  1 usage
        this.count = new AtomicInteger( initialValue: 0);
    }
    public void increment() {  1 usage
        count.incrementAndGet();
    }
    public void decrement() {  1 usage
        count.decrementAndGet();
    }
    public int get() {  1 usage
        return count.get();
    }
}
class ImmutableData {  2 usages
    private final String data;  2 usages

    public ImmutableData(String data) {  1 usage
        this.data = data;
    }
```

```
    public String getData() {         return data;  1 usage
    }
}
public class ThreadSafeDemo {
    public static void main(String[] args) {
        ThreadSafeCounter counter = new
                ThreadSafeCounter();
        ImmutableData data = new ImmutableData("Shared Data");
        int numThreads = 10;

        for (int i = 0; i < numThreads; i++) {
            Thread thread = new Thread(() -> {
                for (int j = 0; j < 1000; j++) {
                    if (Math.random() > 0.5) {
                        counter.increment();
        } else {
            counter.decrement();
        }
        }
            System.out.println("Thread " +
                    Thread.currentThread().getName() + " finished, Data: " + data.getData());             });
            thread.start();
        }
```

```
        for (int i = 0; i < numThreads; i++) {
            try {
                Thread.sleep( millis: 1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        System.out.println("Final counter value: " + counter.get());
    } }
```

```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community
Thread Thread-8 finished, Data: Shared Data
Thread Thread-9 finished, Data: Shared Data
Thread Thread-5 finished, Data: Shared Data
Thread Thread-6 finished, Data: Shared Data
Thread Thread-3 finished, Data: Shared Data
Thread Thread-0 finished, Data: Shared Data
Thread Thread-7 finished, Data: Shared Data
Thread Thread-4 finished, Data: Shared Data
Thread Thread-2 finished, Data: Shared Data
Thread Thread-1 finished, Data: Shared Data
Final counter value: 112

Process finished with exit code 0
```