Thread Control and Deadlocks

1. **Thread Interruption**:

Thread interruption provides a way to signal a thread to stop its execution gracefully.

- Threads can be interrupted using the `interrupt()` method.
- A thread checks for interruptions using `isInterrupted()` or catches `InterruptedException`.

Example:

```
class InterruptExample implements Runnable {
    @Override
    public void run() {
        try {
            while (!Thread.currentThread().isInterrupted()) {
                System.out.println("Thread running...");
                Thread.sleep(1000); // Sleep simulates blocking
            }
        } catch (InterruptedException e) {
            System.out.println("Thread was interrupted");
        }
    }
```

```
public static void main(String[] args) {
        Thread thread = new Thread(new InterruptExample());
        thread.start();

        try {
            Thread.sleep(3000); // Let thread run for 3 seconds
            thread.interrupt(); // Interrupt after 3 seconds
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Fork/Join Framework

Fork/Join is a framework for parallelizing tasks that can be broken into smaller subtasks.

- The `ForkJoinPool` executes tasks that are broken down into smaller sub-tasks, which are processed concurrently.
- Used for CPU-bound tasks that can benefit from parallel execution.

Code Example

java

```
import java.util.concurrent.RecursiveTask;
import java.util.concurrent.ForkJoinPool;

class ForkJoinSumTask extends RecursiveTask<Long> {
    private long start, end;
    private static final long THRESHOLD = 10_000;

    public ForkJoinSumTask(long start, long end) {
        this.start = start;
        this.end = end;
    }

    @Override
    protected Long compute() {
        if (end - start <= THRESHOLD) {
            long sum = 0;
            for (long i = start; i <= end; i++) {
                sum += i;
            }
            return sum;
        } else {
            long middle = (start + end) / 2;
            ForkJoinSumTask task1 = new ForkJoinSumTask(start, middle);
            ForkJoinSumTask task2 = new ForkJoinSumTask(middle + 1, end);
            task1.fork();
            long task2Result = task2.compute();
```

```java
            long task1Result = task1.join();
            return task1Result + task2Result;
        }
    }
```

```java
    public static void main(String[] args) {
        ForkJoinPool pool = new ForkJoinPool();
        ForkJoinSumTask task = new ForkJoinSumTask(0, 1_000_000);
        long result = pool.invoke(task);
        System.out.println("Sum: " + result);
    }
}
```

3. **Deadlock Scenarios**:
Deadlock occurs when two or more threads are waiting on each other to release resources.

- Deadlocks happen when thread locking occurs in a circular chain, where each thread holds a resource that the other needs.

Example:

```java
class DeadlockExample {
    private final Object lock1 = new Object();
    private final Object lock2 = new Object();

    public void method1() {
        synchronized (lock1) {
            System.out.println("Thread 1: Holding lock1...");
            try { Thread.sleep(100); } catch (InterruptedException e) {}

            synchronized (lock2) {
                System.out.println("Thread 1: Holding lock2...");
            }
        }
    }
    public void method2() {
        synchronized (lock2) {
            System.out.println("Thread 2: Holding lock2...");
            try { Thread.sleep(100); } catch (InterruptedException e) {}

            synchronized (lock1) {
                System.out.println("Thread 2: Holding lock1...");
            }
        }
    }

    public static void main(String[] args) {
        DeadlockExample example = new DeadlockExample();
        new Thread(example::method1).start();
        new Thread(example::method2).start();
    }
}
```

4. **Deadlock Prevention**:

Deadlocks can be avoided by locking resources in a consistent order.

- By ensuring threads always lock objects in the same order, circular dependencies can be avoided.

Example:

```
class DeadlockPrevention {
    private final Object lock1 = new Object();
    private final Object lock2 = new Object();

    public void method1() {
        synchronized (lock1) {
            System.out.println("Thread 1: Holding lock1...");
            synchronized (lock2) {
                System.out.println("Thread 1: Holding lock2...");
            }
        }
    }

    public void method2() {
        synchronized (lock1) {  // Lock in the same order to prevent deadlock
            System.out.println("Thread 2: Holding lock1...");
            synchronized (lock2) {
                System.out.println("Thread 2: Holding lock2...");
            }
        }
    }

    public static void main(String[] args) {
        DeadlockPrevention example = new DeadlockPrevention();
        new Thread(example::method1).start();
        new Thread(example::method2).start();
    }
}
```

Summary:

-Thread Interruption allows a thread to stop gracefully using the `interrupt()` method.
- The Fork/Join Framework is essential for parallel processing, breaking tasks into smaller pieces.

- Deadlocks occur when two threads hold locks in circular dependency, but can be avoided with consistent locking orders.