AlternateAlphabet

```
AlphabetNumberAlternate alphabetNumberAlternate = new
AlphabetNumberAlternate();
Thread t1 = new
Thread(alphabetNumberAlternate::printNumber);
Thread t2 = new
Thread(alphabetNumberAlternate::printAlphabet);
t1.start();
t2.start();



public class AlphabetNumberAlternate {
    private int number = 1;
    private int alphabet = 1;
    private boolean isAlphabet=false;

    public synchronized void printNumber(){
        while(number<=26){
            while(isAlphabet){
                try {
                    wait();
                }
                catch (Exception e){
                    System.out.println("Exception while printing number "+e.getMessage());
                }
            }
            System.out.println(number);
            number++;
            isAlphabet=true;
            notify();
        }
    }

    public synchronized void printAlphabet(){
        while(alphabet<=26){
            while(!isAlphabet){
                try {
                    wait();
                }
                catch (Exception e){
                    System.out.println("Exception while printing alphabet "+e.getMessage());
                }
            }
            System.out.println(String.valueOf((char)(alphabet+64)));
            alphabet++;
            isAlphabet=false;
            notify();
        }
    }
}
```

ProducerConsumer - BlockingQueue

```
public class Main {
    public static void main(String[] args) {
        SharedQueue sharedQueue = new SharedQueue();

        Thread producer = new Thread(new Producer(sharedQueue));
        Thread consumer = new Thread(new Consumer(sharedQueue));

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

 // Shared resource (thread-safe queue)
 class SharedQueue {
    BlockingQueue<Integer> queue = new ArrayBlockingQueue<>(5); // size 5
 }


class Producer implements Runnable {
    private SharedQueue sharedQueue;
    public Producer(SharedQueue sharedQueue) {
        this.sharedQueue = sharedQueue;
    }
    public void run() {
        int value = 0;
        try {
            while (true) {
                sharedQueue.queue.put(value);  // waits if queue is full
                System.out.println("Produced: " + value);
                value++;
                Thread.sleep(500);  // simulate time
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}

class Consumer implements Runnable {
    private SharedQueue sharedQueue;
    public Consumer(SharedQueue sharedQueue) {
        this.sharedQueue = sharedQueue;
    }
    public void run() {
        try {
            while (true) {
                int value = sharedQueue.queue.take(); // waits if queue is empty
                System.out.println("Consumed: " + value);
                Thread.sleep(1000); // simulate time
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}
```

Reentrant Lock

```
ShareBankAccount shareBankAccount = new ShareBankAccount(500);

    Thread t1= new Thread(()->{
        for(int i=0;i<10;i++){
            shareBankAccount.subBalance(10);
            try {
                Thread.sleep(50);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    },"Thread 1");
    Thread t2= new Thread(()->{
        for(int i=0;i<10;i++){
            shareBankAccount.subBalance(20);
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    },"Thread 2");

    t1.start();t2.start();
    try {
        t1.join();t2.join();
    }
    catch (Exception e){
        System.out.println("Exception thrown");
    }
    System.out.println(shareBankAccount.getBalance());
}



public class ShareBankAccount {

    private int balance ;
    private final Lock lock = new ReentrantLock();

    ShareBankAccount(int balance){
        this.balance=balance;
    }

    public void subBalance(int amnt){
        lock.lock();

        try {

            if(balance-amnt>0){
                balance-=amnt;
                System.out.println(Thread.currentThread().getName() + " debited " + amnt + ". New balance: " + balance);
            }
        }finally {
            lock.unlock();
        }
    }

    public int getBalance(){
        return balance;
    }
}
```

CompletableFuture

```
public static void main(String[] args) {
    // First async task increments counter 1000 times
    CompletableFuture<Void> future1 = CompletableFuture.runAsync(() -> {
        for (int i = 0; i < 10; i++) {
            System.out.println("Hi from f1");
            counter.incrementAndGet();
        }
    });

    // Second async task increments counter 1000 times
    CompletableFuture<Void> future2 = CompletableFuture.runAsync(() -> {
        for (int i = 0; i < 10; i++) {
            System.out.println("Hi from f2");
            counter.incrementAndGet();
        }
    });

    // Wait for both tasks to finish
    CompletableFuture.allOf(future1, future2).join();

    // Print final counter value
    System.out.println("Final Counter Value: " + counter.get());
}
```

Threadpool Executor

```
import java.util.concurrent.*;

public class ThreadPoolExample {

    public static void main(String[] args) {
        // Thread pool banaya with 2 core threads, 4 max threads, 1 second idle timeout, and a queue of size 2
        ThreadPoolExecutor executor = new ThreadPoolExecutor(
                2,                  // core threads
                4,                  // maximum threads
                1,                  // idle thread timeout
                TimeUnit.SECONDS,
                new LinkedBlockingQueue<>(2) // task queue
        );

        // 6 dummy tasks submit kar rahe hai
        for (int i = 1; i <= 6; i++) {
            int taskId = i;
            executor.execute(() -> {
                System.out.println("Task " + taskId + " is running by " + Thread.currentThread().getName());
                try {
                    Thread.sleep(1000); // simulate work
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            });
        }

        // Shut down thread pool gracefully
        executor.shutdown();
    }
}
```