指定区间链表翻转

```
public class ReverseLinkedListInRange {
    public ListNode reverseBetween(ListNode head, int m, int n) {
        if (head == null || m >= n) {
            return head;
        }

        ListNode dummy = new ListNode(-1);
        dummy.next = head;
        ListNode prev = dummy;

        // Move prev to the node just before the reversed segment
        for (int i = 1; i < m; i++) {
            prev = prev.next;
        }

        ListNode current = prev.next;
        ListNode nextNode = null;
        ListNode reversedSegmentTail = current;

        for (int i = m; i <= n; i++) {
            nextNode = current.next;
            current.next = nextNode.next;
            nextNode.next = prev.next;
            prev.next = nextNode;
        }

        reversedSegmentTail.next = current;

        return dummy.next;
    }
}
```

手写一个工厂模式

```
public class SendFactory {

  public Sender produce(String type) {
    if (type.equals("mail")) return new MailSender;
    else if (type.equals("sms")) return new SmsSender;
    else return new DefaultSender;
  }

}
```

手写一个单例模式

```
public class Singleton {

  private Singleton instance = null;

  public static Singleton getSingleton() {
    if (instance == null) {
      instance = new Singleton();
    }
    return instance;
  }

}
```

手写双Check实现单例模式

```
public class Singleton {
    private volatile static Singleton instance;  // 使用 volatile 关键字确保可见性和禁止指令重排

    private Singleton() {
        // 私有构造方法，防止外部实例化
    }

    public static Singleton getInstance() {
        if (instance == null) {  // 第一次检查，避免不必要的同步
            synchronized (Singleton.class) {  // 加锁确保线程安全
                if (instance == null) {  // 第二次检查，防止在同步块外的线程创建实例
                    instance = new Singleton();
                }
            }
        }
        return instance;
    }
}
```

手写一个读写锁

```
public class ReadWriteLock {
    private int readers; // 记录当前读取线程的数量
    private int writers; // 记录当前写入线程的数量
    private int writeRequests; // 记录等待写入的请求数量

    public synchronized void lockRead() throws InterruptedException {
        while (writers > 0 || writeRequests > 0) {
            wait();
        }
        readers++;
    }

    public synchronized void unlockRead() {
        readers--;
        notifyAll();
    }
```

```
    public synchronized void lockWrite() throws InterruptedException {
        writeRequests++;
        while (readers > 0 || writers > 0) {
            wait();
        }
        writeRequests--;
        writers++;
    }

    public synchronized void unlockWrite() {
        writers--;
        notifyAll();
    }
}
```

背包问题

```
public class Knapsack {
    public static int knapsack(int target, int[] wt, int[] val, int n) {
        int[][] dp = new int[n + 1][target + 1];
        for (int i = 0; i <= n; i++) {
            for (int w = 0; w <= target; w++) {
                if (i == 0 || w == 0)
                    dp[i][w] = 0;
                else if (wt[i - 1] <= w)
                    dp[i][w] = Math.max(val[i - 1] + dp[i - 1][w - wt[i - 1]], dp[i - 1]
[w]);
                else
                    dp[i][w] = dp[i - 1][w];
            }
        }
        return dp[n][target];
    }

    public static void main(String[] args) {
        int val[] = new int[] { 60, 100, 120 };
        int wt[] = new int[] { 10, 20, 30 };
        int W = 50;
        int n = val.length;
        System.out.println(knapsack(W, wt, val, n));
    }
}
```

手写一个promise

```
class Promise {
    constructer(executor) {
        this.status = 'PENDING'
        this.value = undefined
        this.reason = undefined

        let resolve = (value) => {
            if (this.status == 'PENDING') {
```

```
                this.status = 'FULFILLED'
                this.value = value
            }
        }

        let reject = (reason) => {
            if (this.status == 'PENDING') {
                this.status = 'RECJECTED'
                this.reason = reason
            }
        }

        try {
            executor(resolve, reject)
        } catch(err) {
            reject(err)
        }
    }

    then(onFulfilled, onRejected) {
        if (this.status == 'PENDING') {
            onFulfilled(this.value)
        }
        if (this.status == 'RECJECTED') {
            onRejected(this.reason)
        }
    }
}
```