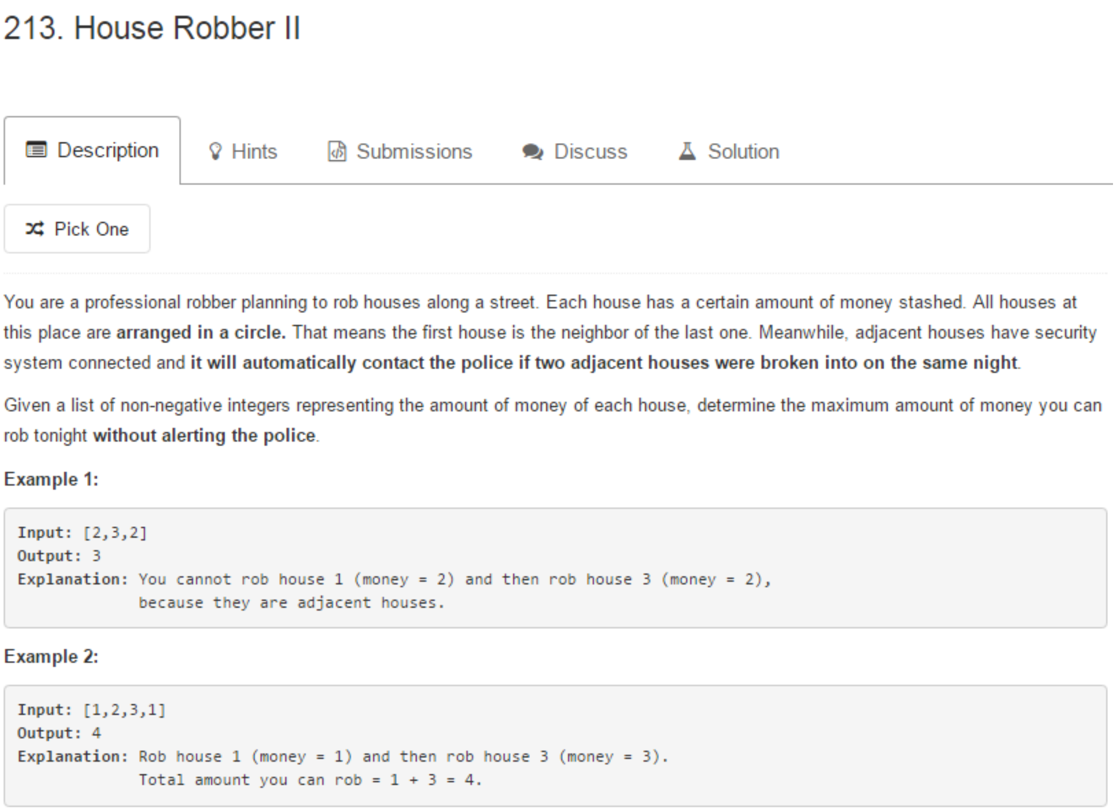


213. House Robber II

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed. All houses at this place are **arranged in a circle.** That means the first house is the neighbor of the last one. Meanwhile, adjacent houses have security system connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

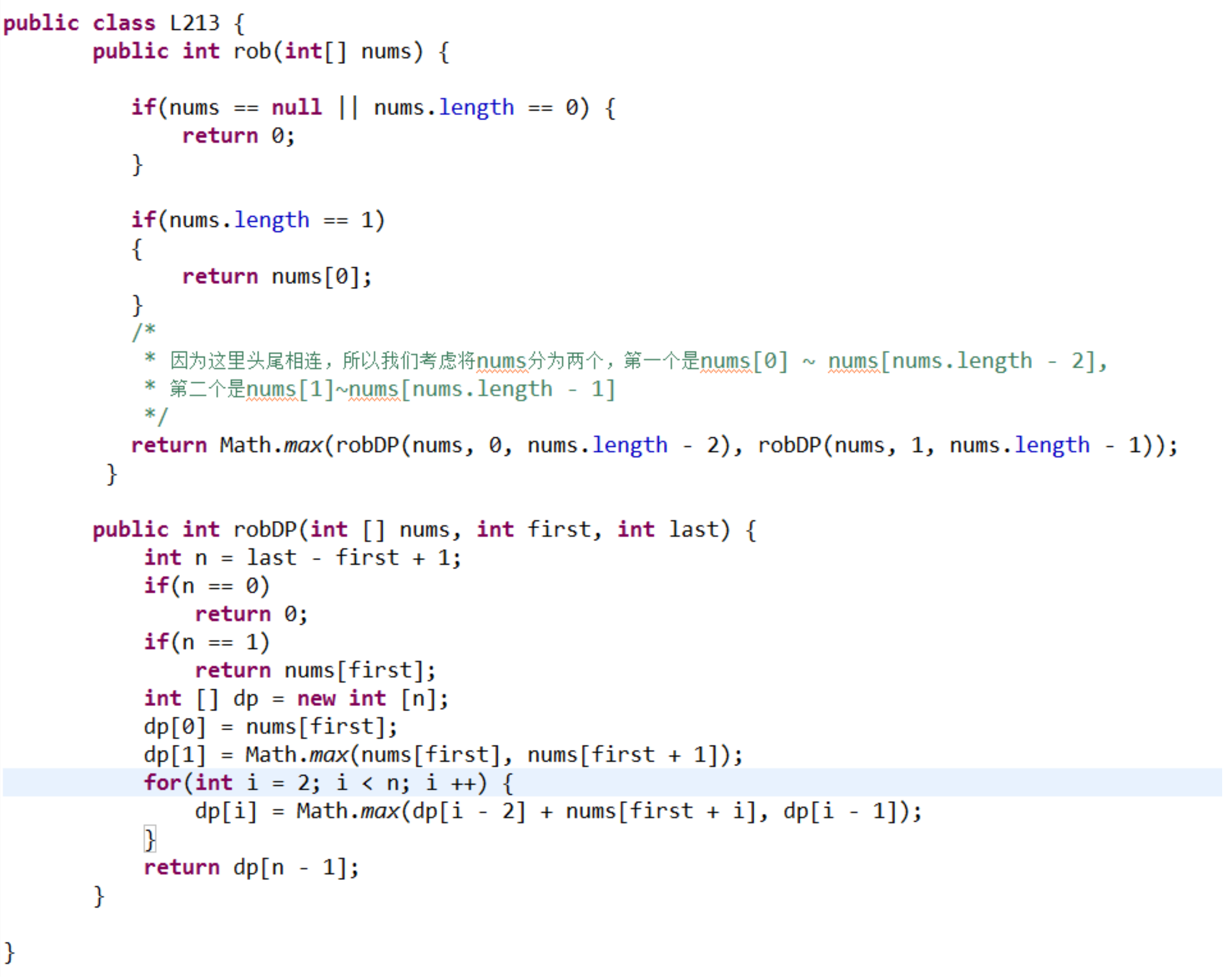
Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight **without alerting the police**.

Example 1:

```
Input: [2,3,2]
Output: 3
Explanation: You cannot rob house 1 (money = 2) and then rob house 3 (money = 2),
             because they are adjacent houses.
```

Example 2:

```
Input: [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
             Total amount you can rob = 1 + 3 = 4.
```



```java
public class L213 {
       public int rob(int[] nums) {

          if(nums == null || nums.length == 0) {
              return 0;
          }

          if(nums.length == 1)
          {
              return nums[0];
          }
          /*
           * 因为这里头尾相连，所以我们考虑将nums分为两个，第一个是nums[0] ~ nums[nums.length - 2],
           * 第二个是nums[1]~nums[nums.length - 1]
           */
          return Math.max(robDP(nums, 0, nums.length - 2), robDP(nums, 1, nums.length - 1));
        }

       public int robDP(int [] nums, int first, int last) {
           int n = last - first + 1;
           if(n == 0)
               return 0;
           if(n == 1)
               return nums[first];
           int [] dp = new int [n];
           dp[0] = nums[first];
           dp[1] = Math.max(nums[first], nums[first + 1]);
           for(int i = 2; i < n; i ++) {
               dp[i] = Math.max(dp[i - 2] + nums[first + i], dp[i - 1]);
           }
           return dp[n - 1];
       }

}
```