Assignment 2

Due time: 02/17/2022, 11:59pm EST

Total credits: 100, 5 questions

Submission guide:

1. Create folder and name it with the format FirstName_LastName_Aassigment2 for example Chunyu_Yuan_Assignment1

2. Inside the folder, you should have 5 java files

3. compress your file to .zip format and submit it to the blackboard,

4. if you have any question, please send email to cyuan1@gradcenter.cuny.edu

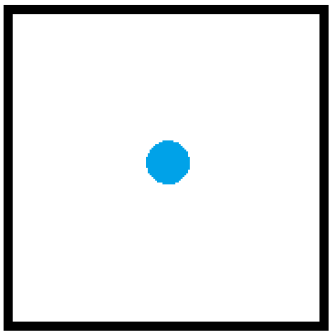
Below are examples about the folder contend (Square.java, TestSquare.java, Solution1.java, Solution2.java, Solution3.java):

1. continue design and implement Square Class (40 credits)

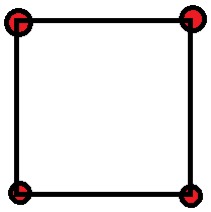
Requirements:

Add them to your assignment 1

- **One data field: central_point, type public integer array, initialize it to {0,0}**



- **Two constructors: one constructor with argu (central_point, side), one constructor with only argu (central_point)**
- **Four methods:**
  - **setCentral_Point, public, void**
  - **getCentral_Point, public, return integer array;**
  - **getDistance(other_point), public, return double distance, given a point, calculate the distance between central_point with that point**
  - **getSquare_Corners(), public, return int[][], (hint: based on the central_point and side to calculate the corners point(red points in the below picture), return a 2D array, the returned array's size will be int[4][2])**



2. continue to design and implement TestSquare Class (30 credits)

Add them to your assignment 1

Requirements:

- **declare one Square object square3 using one constructor with given central_point and side**
- **print out its central_point using variable reference (you can use array index to access the point element)**
- **using method setCentral_Point to change the central_point**
- **print out its central_point using method getCentral_Point (you can use array index to access the point element)**
- **print the calculated distance between the central_point with point{1,1} from getDistance(other_point)**
- **print the points from getSquare_Corners(), (use for loop the print)**

(make sure you can compile your program and get the results)

4. Question 1 (10 credits)

Given an array `nums` of size `n`, return *the majority element*.

The majority element is the element that appears more than `⌊n / 2⌋` times. You may assume that the majority element always exists in the array.

Example 1:

Input: nums = [1, 2, 1]

Output: 1

Example 2:

Input: nums = [3, 3, 1, 2, 0, 3, 3]

Output: 3

Constraints:

- `n == nums.length`
- `1 <= n <= 5 * 10^4`
- `-2^31 <= nums[i] <= 2^31 - 1`

Below is your start code, finish the method "public int answer(int[] nums)", don't need to write the main method inside the class Solution

```
class Solution {
    public int answer(int[] nums) {


    }
}
```

5. Question 2 **(10 credits)**

You are given an integer array `nums`. The unique elements of an array are the elements that appear **exactly once** in the array.

Return *the **sum** of all the unique elements of* `nums`.

Example 1:

Input: nums = [0,2,3,2]

Output: 3

Explanation: The unique elements are [0,3], and the sum is 3

Example 2:

Input: nums = [2,2,2,2,2,2,3,3]

Output: 0

Explanation: There are no unique elements, and the sum is 0

Example 3:

Input: nums = [1,2,3,4]

Output: 10

Explanation: The unique elements are [1,2,3,4], and the sum is 10

Below is your start code, finish the method "public int answer(int[] nums)", don't need to write the main method inside the class Solution

```
class Solution2 {
    public int answer(int[] nums) {


    }
}
```

6. Question 3 (10 credits)

Design a class to find the k^{th} sum in a stream. Note that it is the sum of k^{th} largest element and k^{th} smallest element in the sorted order.

Implement `KthSum` class:

- `KthSum(int k, int[] nums)` Initializes the object with the integer `k` and the stream of integers `nums`.
- `int add(int val)` Appends the integer `val` to the stream and returns the sum of k^{th} largest element and k^{th} smallest element in the sorted order

Example 1:

Input

```
["KthSum", "add", "add", "add"]
[[3, [4, 5, 8, 2]], [3], [5], [10]]
```

Output

```
[null, 8, 9, 9]
```

Explanation

```
KthSum kthSum = new KthSum (3, [4, 5, 8, 2]);
kthSum.add(3);   // return 8, 3th largest element is 4, 3th smallest element is 4, sum is 8
kthSum.add(5);   // return 9, 3th largest element is 5, 3th smallest element is 4, sum is 9
kthSum.add(10);  // return 9, 3th largest element is 5, 3th smallest element is 4, sum is 9
```

- **It is guaranteed that there will be at least `k` elements in the array when you search for the k^{th} element.**

Below is your start code,

```
class KthSum {

    public KthSum(int k, int[] nums) {

    }

    public int add(int val) {

    }
}

/**
 * Your KthSum object will be instantiated and called as such:
 * KthSum obj = new KthSum(k, nums);
 * int param_1 = obj.add(val);
 */
```