Problem 1

Intuition

Given a string columnTitle that represents the column title as appears in an Excel sheet, return *its corresponding column number*.

For example:

```
A -> 1
B -> 2
C -> 3
...
Z -> 26
AA -> 27
AB -> 28
...
```

Example 1:

Input: columnTitle = "A"
Output: 1

Example 2:

Input: columnTitle = "AB"
Output: 28

Example 3:

Input: columnTitle = "ZY"
Output: 701

Constraints:

- 1 <= columnTitle.length <= 7
- columnTitle consists only of uppercase English letters.
- columnTitle is in the range ["A", "FXSHRXW"].

Solution

```java
class Solution {
    public int titleToNumber(String columnTitle) {
        int res = 0;
        for(int i = 0; i < columnTitle.length(); i++){
            int value = columnTitle.charAt(i) - 'A' + 1;
            res = res * 26 + value;
        }
        return res;
    }
}
```

Problem 2

Intuition

Given an integer n, return *the number of trailing zeroes in* n!.

Note that n! = n * (n - 1) * (n - 2) * ... * 3 * 2 * 1.

Example 1:

Input: n = 3
Output: 0
Explanation: 3! = 6, no trailing zero.

Example 2:

Input: n = 5
Output: 1
Explanation: 5! = 120, one trailing zero.

Example 3:

Input: n = 0
Output: 0

Constraints:

- 0 <= n <= 10^4

Solution

```
class Solution {
    public int trailingZeroes(int n) {
        int count = 0;
        while( n >= 5 ){
            count += n/5;
            n = n/5;
        }
        return count;
    }
}
```

Problem 3

Intuition

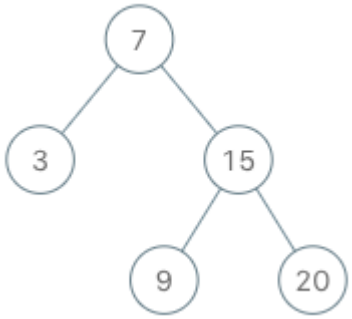
Implement the BSTIterator class that represents an iterator over the **in-order traversal** of a binary search tree (BST):

- BSTIterator(TreeNode root) Initializes an object of the BSTIterator class. The root of the BST is given as part of the constructor. The pointer should be initialized to a non-existent number smaller than any element in the BST.
- boolean hasNext() Returns true if there exists a number in the traversal to the right of the pointer, otherwise returns false.
- int next() Moves the pointer to the right, then returns the number at the pointer.

Notice that by initializing the pointer to a non-existent smallest number, the first call to next() will return the smallest element in the BST.

You may assume that next() calls will always be valid. That is, there will be at least a next number in the in-order traversal when next() is called.

Example 1:



Input
["BSTIterator", "next", "next", "hasNext", "next", "hasNext", "next", "hasNext", "next", "hasNext"]
[[[7, 3, 15, null, null, 9, 20]], [], [], [], [], [], [], [], [], []]
Output
[null, 3, 7, true, 9, true, 15, true, 20, false]

Explanation
BSTIterator bSTIterator = new BSTIterator([7, 3, 15, null, null, 9, 20]);
bSTIterator.next();   // return 3
bSTIterator.next();   // return 7
bSTIterator.hasNext(); // return True
bSTIterator.next();   // return 9
bSTIterator.hasNext(); // return True
bSTIterator.next();   // return 15
bSTIterator.hasNext(); // return True
bSTIterator.next();   // return 20
bSTIterator.hasNext(); // return False

Constraints:

- The number of nodes in the tree is in the range [1, 10^5].

- 0 <= Node.val <= 10
- At most 10 calls will be made to hasNext, and next.

Solution

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class BSTIterator {
    private List<TreeNode> stack;
    public BSTIterator(TreeNode root) {
        stack = new ArrayList<>();
        putLeft(root);
    }

    public int next() {
        TreeNode curr = stack.remove(stack.size() - 1);
        putLeft(curr.right);
        return curr.val;
    }

    public boolean hasNext() {
        return !stack.isEmpty();
    }
    private void putLeft(TreeNode curr){
        while(curr != null){
            stack.add(curr);
            curr = curr.left;
        }
    }
}

/**
 * Your BSTIterator object will be instantiated and called as such:
 * BSTIterator obj = new BSTIterator(root);
 * int param_1 = obj.next();
 * boolean param_2 = obj.hasNext();
 */
```

Problem 4

Intuition

Table: Employee

```
+-------------+------+
| Column Name | Type |
+-------------+------+
| id          | int  |
| salary      | int  |
+-------------+------+
```

id is the primary key (column with unique values) for this table.
Each row of this table contains information about the salary of an employee.

Write a solution to find the second highest **distinct** salary from the Employee table. If there is no second highest salary, return null (return None in Pandas).

The result format is in the following example.

Example 1:

Input:
Employee table:

```
+----+--------+
| id | salary |
+----+--------+
| 1  | 100    |
| 2  | 200    |
| 3  | 300    |
+----+--------+
```

Output:

```
+---------------------+
| SecondHighestSalary |
+---------------------+
| 200                 |
+---------------------+
```

Example 2:

Input:
Employee table:

```
+----+--------+
| id | salary |
+----+--------+
| 1  | 100    |
+----+--------+
```

Output:

```
+---------------------+
| SecondHighestSalary |
+---------------------+
| null                |
+---------------------+
```

Solution

```sql
select (select distinct salary
from Employee
order by salary desc
limit 1 offset 1) as SecondHighestSalary;
```

Problem 5

Intuition

The Java *instanceof* operator is used to test if the object or instance is an instanceof the specified type.

In this problem we have given you three classes in the editor:

- Student class
- Rockstar class
- Hacker class

In the main method, we populated an *ArrayList* with several instances of these classes. *count* method calculates how many instances of each type is present in the ArrayList. The code prints three integers, the number of instance of Student class, the number of instance of Rockstar class, the number of instance of Hacker class.

But some lines of the code are missing, and you have to fix it by modifying only lines! Don't add, delete or modify any extra line.

To restore the original code in the editor, click on the top left icon in the editor and create a new buffer.

Sample Input

5
Student
Student
Rockstar
Student
Hacker

Sample Output

3 1 1

Solution

```
import java.util.*;


class Student{}
class Rockstar{   }
class Hacker{}


public class InstanceOFTutorial{

   static String count(ArrayList mylist){
      int a = 0,b = 0,c = 0;
      for(int i = 0; i < mylist.size(); i++){
         Object element=mylist.get(i);
         if(element instanceof Student)
            a++;
         if(element instanceof Rockstar)
```

```
            b++;
         if(element instanceof Hacker)
            c++;
      }
      String ret = Integer.toString(a)+" "+ Integer.toString(b)+" "+
Integer.toString(c);
      return ret;
   }

   public static void main(String []args){
      ArrayList mylist = new ArrayList();
      Scanner sc = new Scanner(System.in);
      int t = sc.nextInt();
      for(int i=0; i<t; i++){
         String s=sc.next();
         if(s.equals("Student"))mylist.add(new Student());
         if(s.equals("Rockstar"))mylist.add(new Rockstar());
         if(s.equals("Hacker"))mylist.add(new Hacker());
      }
      System.out.println(count(mylist));
   }
}
```