CHRIST
(DEEMED TO BE UNIVERSITY)
BENGALURU · INDIA

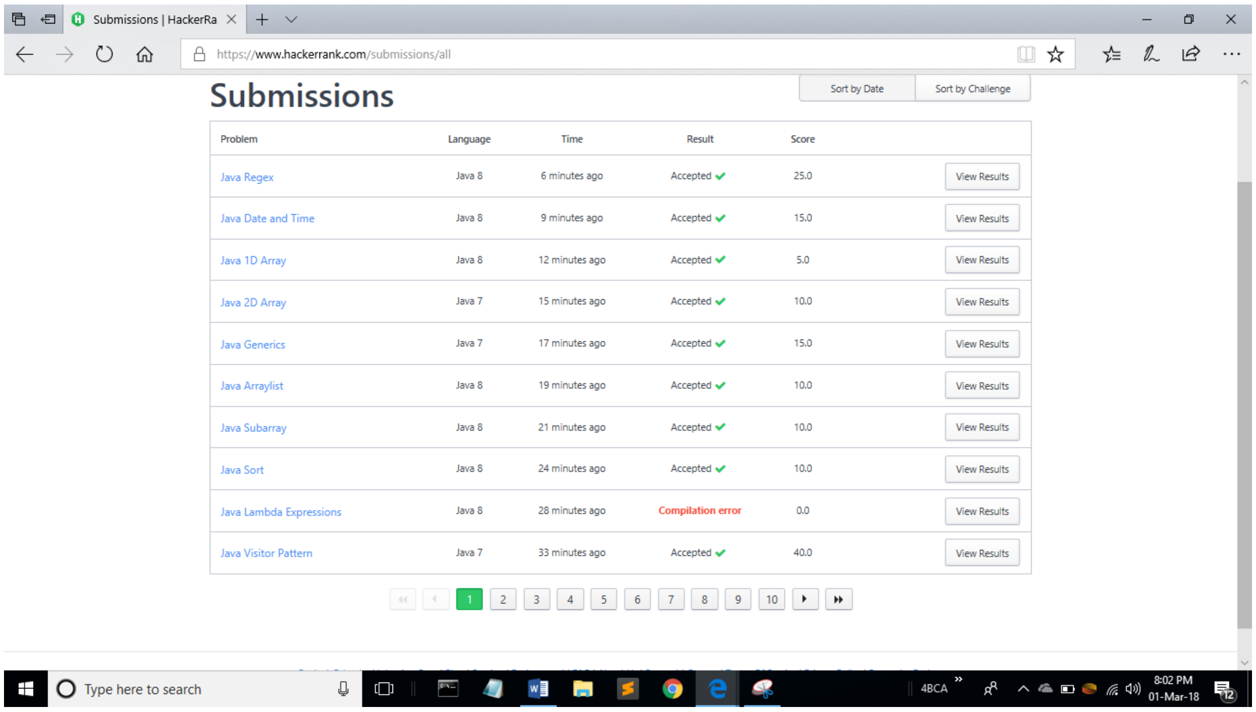
PROGRAMMING IN JAVA

CIA 3

(WWW.HACKERRANK.COM)

TOTAL QUESTIONS ATTEMPTED **63**

Name:	**Yerraballi Suresh Kumar Reddy**
Reg. No.:	**1641065**
Class:	**BCA 'A'**
Date:	**28/02/2017**



Submissions | HackerRa

https://www.hackerrank.com/submissions/all

Submissions

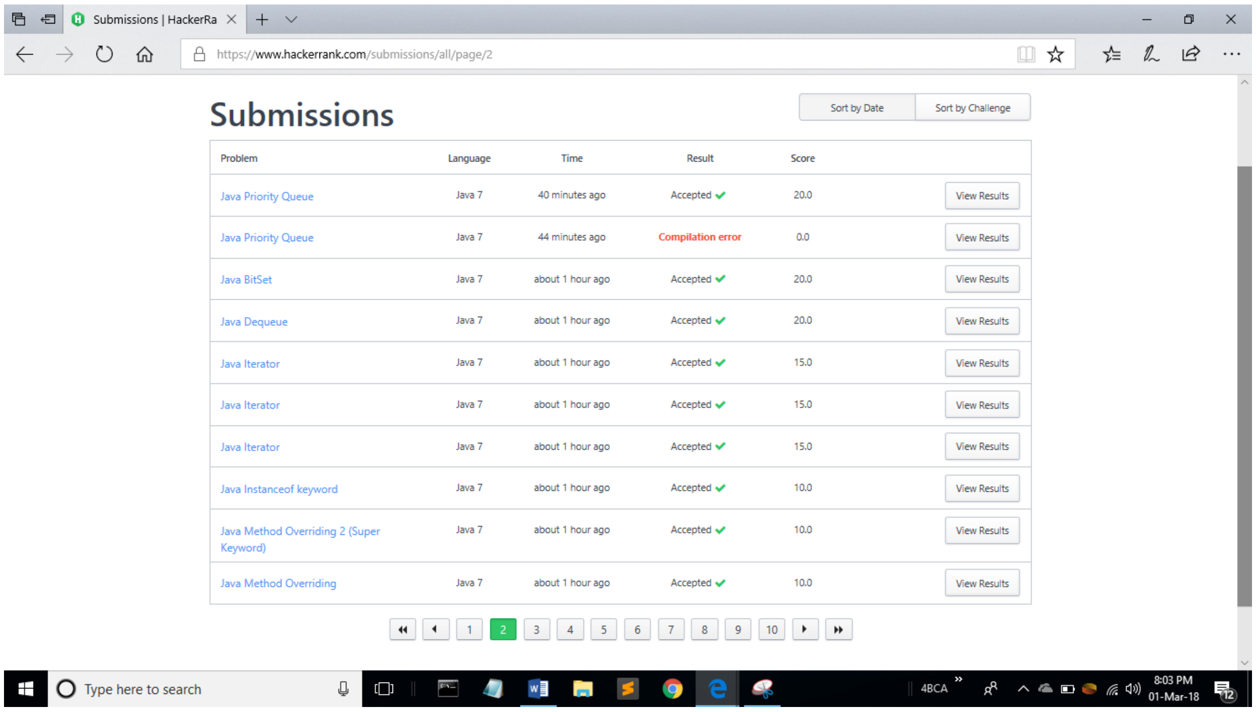
Sort by Date | Sort by Challenge

Problem	Language	Time	Result	Score	
Java Regex	Java 8	6 minutes ago	Accepted ✓	25.0	View Results
Java Date and Time	Java 8	9 minutes ago	Accepted ✓	15.0	View Results
Java 1D Array	Java 8	12 minutes ago	Accepted ✓	5.0	View Results
Java 2D Array	Java 7	15 minutes ago	Accepted ✓	10.0	View Results
Java Generics	Java 7	17 minutes ago	Accepted ✓	15.0	View Results
Java Arraylist	Java 8	19 minutes ago	Accepted ✓	10.0	View Results
Java Subarray	Java 8	21 minutes ago	Accepted ✓	10.0	View Results
Java Sort	Java 8	24 minutes ago	Accepted ✓	10.0	View Results
Java Lambda Expressions	Java 8	28 minutes ago	Compilation error	0.0	View Results
Java Visitor Pattern	Java 7	33 minutes ago	Accepted ✓	40.0	View Results

1 2 3 4 5 6 7 8 9 10

Type here to search

4BCA 8:02 PM 01-Mar-18



Submissions | HackerRa

https://www.hackerrank.com/submissions/all/page/2

Submissions

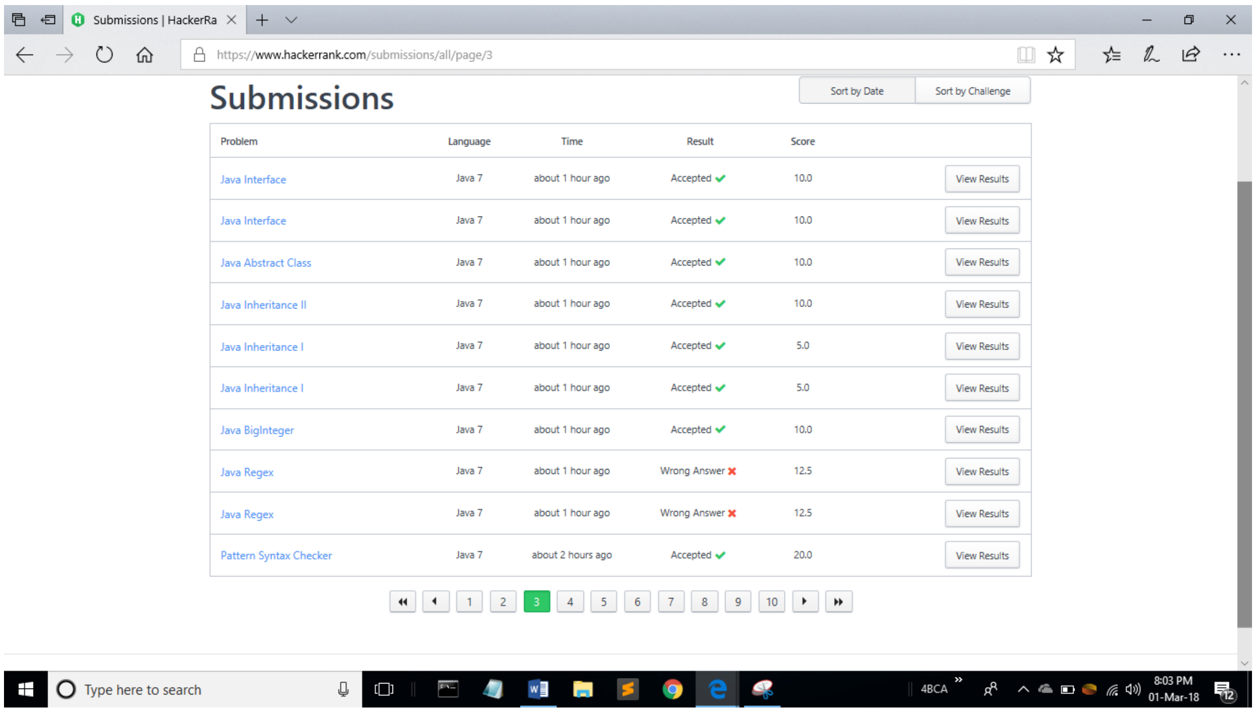
Sort by Date | Sort by Challenge

Problem	Language	Time	Result	Score	
Java Priority Queue	Java 7	40 minutes ago	Accepted ✓	20.0	View Results
Java Priority Queue	Java 7	44 minutes ago	Compilation error	0.0	View Results
Java BitSet	Java 7	about 1 hour ago	Accepted ✓	20.0	View Results
Java Dequeue	Java 7	about 1 hour ago	Accepted ✓	20.0	View Results
Java Iterator	Java 7	about 1 hour ago	Accepted ✓	15.0	View Results
Java Iterator	Java 7	about 1 hour ago	Accepted ✓	15.0	View Results
Java Iterator	Java 7	about 1 hour ago	Accepted ✓	15.0	View Results
Java Instanceof keyword	Java 7	about 1 hour ago	Accepted ✓	10.0	View Results
Java Method Overriding 2 (Super Keyword)	Java 7	about 1 hour ago	Accepted ✓	10.0	View Results
Java Method Overriding	Java 7	about 1 hour ago	Accepted ✓	10.0	View Results

1 2 3 4 5 6 7 8 9 10

Type here to search

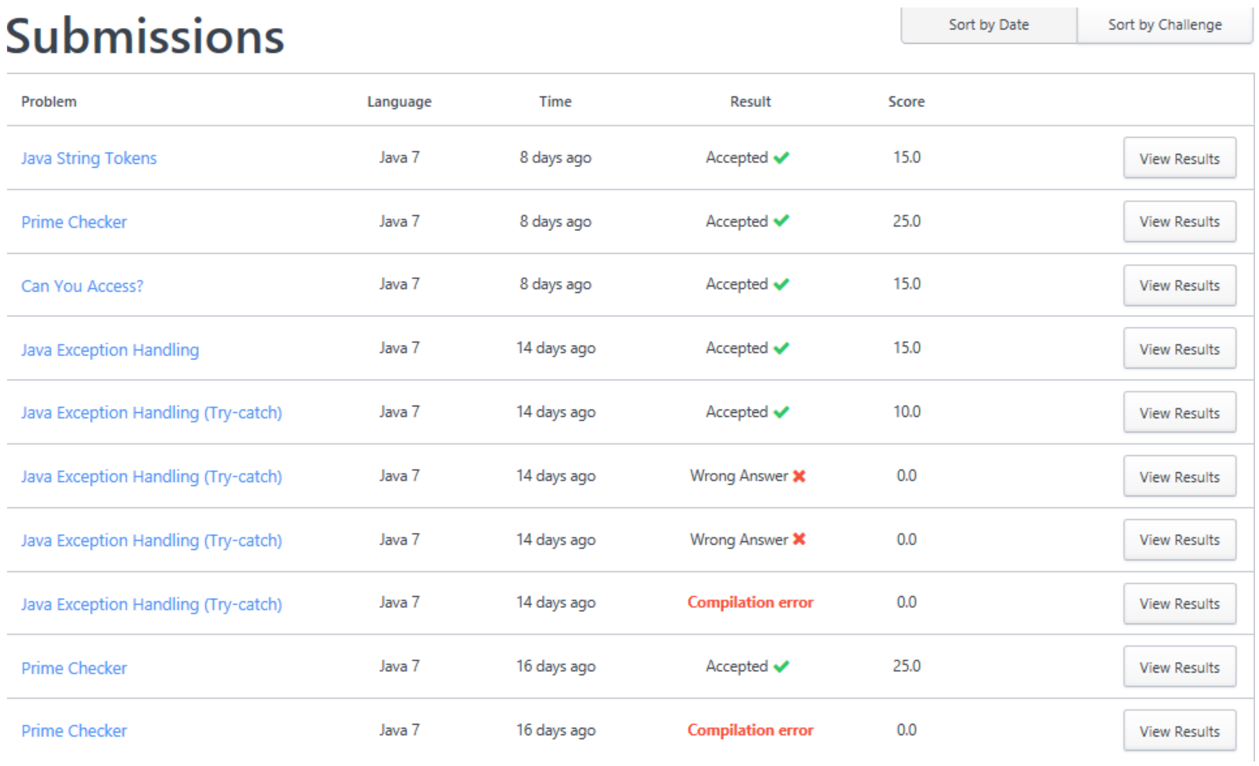
4BCA 8:03 PM 01-Mar-18



Submissions

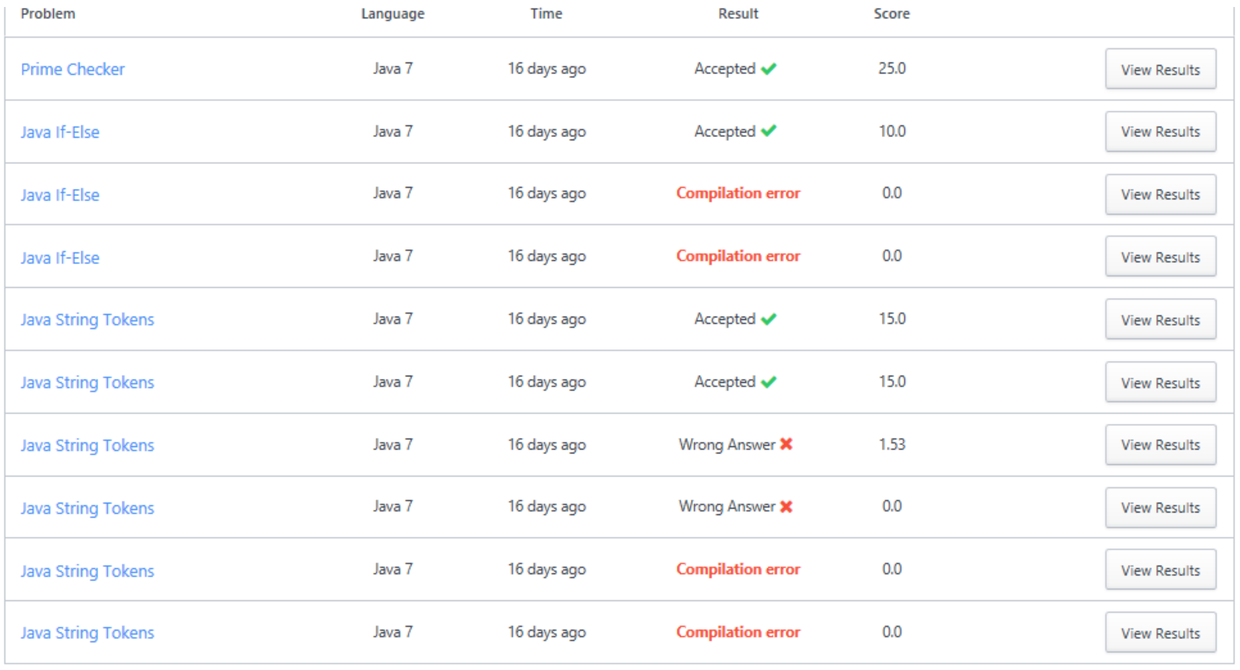
Problem	Language	Time	Result	Score	
Java Interface	Java 7	about 1 hour ago	Accepted ✓	10.0	View Results
Java Interface	Java 7	about 1 hour ago	Accepted ✓	10.0	View Results
Java Abstract Class	Java 7	about 1 hour ago	Accepted ✓	10.0	View Results
Java Inheritance II	Java 7	about 1 hour ago	Accepted ✓	10.0	View Results
Java Inheritance I	Java 7	about 1 hour ago	Accepted ✓	5.0	View Results
Java Inheritance I	Java 7	about 1 hour ago	Accepted ✓	5.0	View Results
Java BigInteger	Java 7	about 1 hour ago	Accepted ✓	10.0	View Results
Java Regex	Java 7	about 1 hour ago	Wrong Answer ✗	12.5	View Results
Java Regex	Java 7	about 1 hour ago	Wrong Answer ✗	12.5	View Results
Pattern Syntax Checker	Java 7	about 2 hours ago	Accepted ✓	20.0	View Results

1 2 3 4 5 6 7 8 9 10

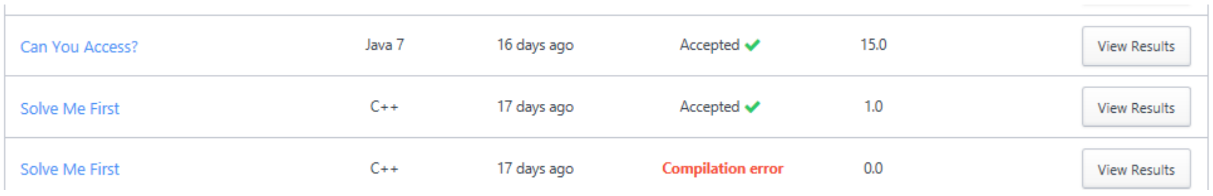


Submissions

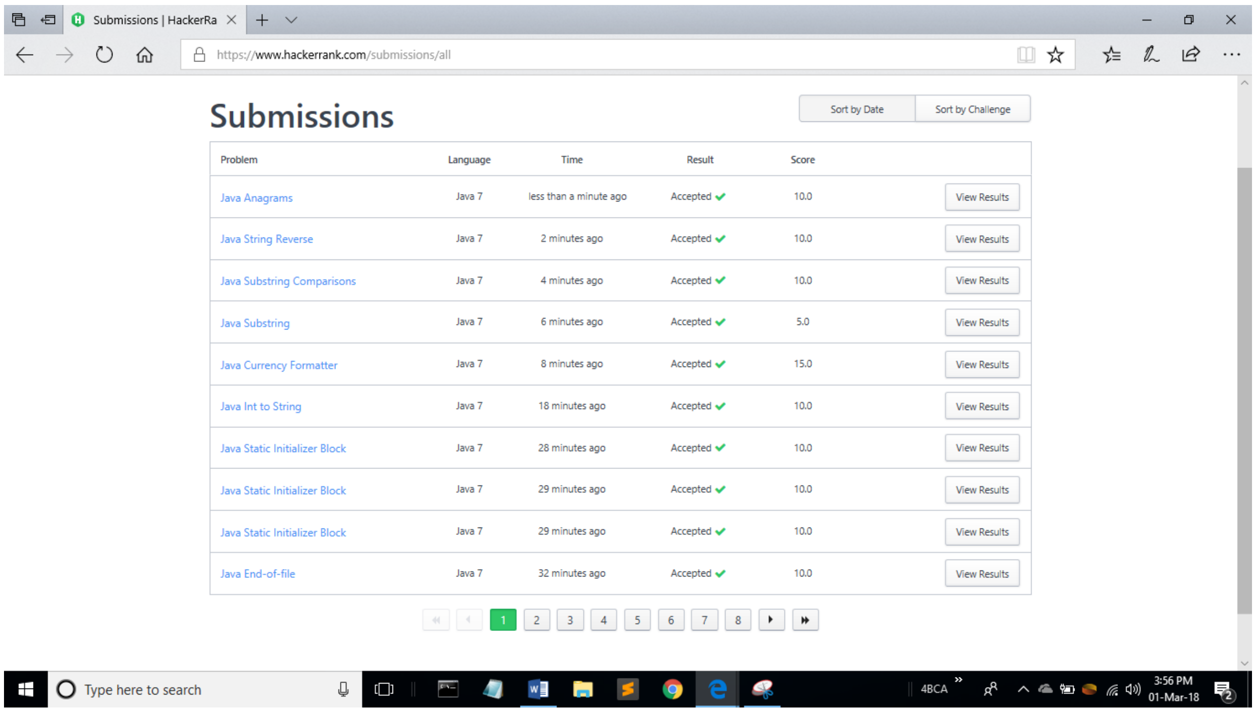
Problem	Language	Time	Result	Score	
Java String Tokens	Java 7	8 days ago	Accepted ✓	15.0	View Results
Prime Checker	Java 7	8 days ago	Accepted ✓	25.0	View Results
Can You Access?	Java 7	8 days ago	Accepted ✓	15.0	View Results
Java Exception Handling	Java 7	14 days ago	Accepted ✓	15.0	View Results
Java Exception Handling (Try-catch)	Java 7	14 days ago	Accepted ✓	10.0	View Results
Java Exception Handling (Try-catch)	Java 7	14 days ago	Wrong Answer ✗	0.0	View Results
Java Exception Handling (Try-catch)	Java 7	14 days ago	Wrong Answer ✗	0.0	View Results
Java Exception Handling (Try-catch)	Java 7	14 days ago	Compilation error	0.0	View Results
Prime Checker	Java 7	16 days ago	Accepted ✓	25.0	View Results
Prime Checker	Java 7	16 days ago	Compilation error	0.0	View Results



Problem	Language	Time	Result	Score	
Prime Checker	Java 7	16 days ago	Accepted ✓	25.0	View Results
Java If-Else	Java 7	16 days ago	Accepted ✓	10.0	View Results
Java If-Else	Java 7	16 days ago	Compilation error	0.0	View Results
Java If-Else	Java 7	16 days ago	Compilation error	0.0	View Results
Java String Tokens	Java 7	16 days ago	Accepted ✓	15.0	View Results
Java String Tokens	Java 7	16 days ago	Accepted ✓	15.0	View Results
Java String Tokens	Java 7	16 days ago	Wrong Answer ✗	1.53	View Results
Java String Tokens	Java 7	16 days ago	Wrong Answer ✗	0.0	View Results
Java String Tokens	Java 7	16 days ago	Compilation error	0.0	View Results
Java String Tokens	Java 7	16 days ago	Compilation error	0.0	View Results



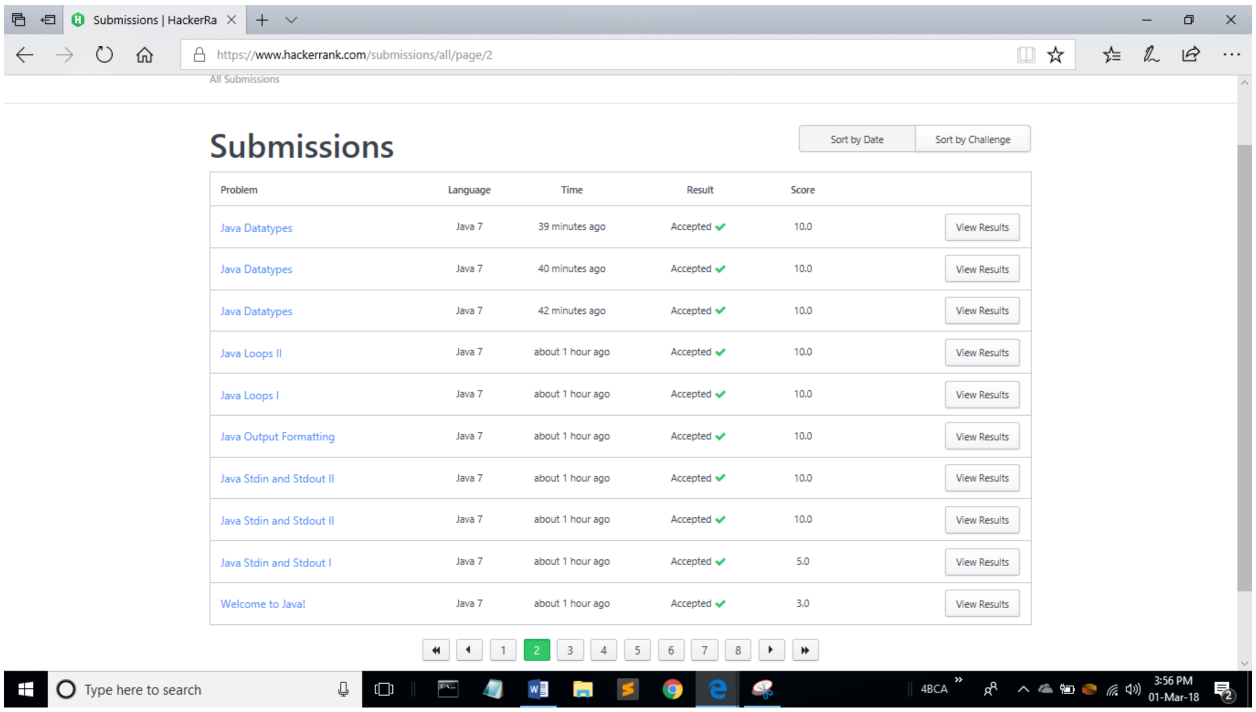
Can You Access?	Java 7	16 days ago	Accepted ✓	15.0	View Results
Solve Me First	C++	17 days ago	Accepted ✓	1.0	View Results
Solve Me First	C++	17 days ago	Compilation error	0.0	View Results



https://www.hackerrank.com/submissions/all

Submissions

Problem	Language	Time	Result	Score	
Java Anagrams	Java 7	less than a minute ago	Accepted ✓	10.0	View Results
Java String Reverse	Java 7	2 minutes ago	Accepted ✓	10.0	View Results
Java Substring Comparisons	Java 7	4 minutes ago	Accepted ✓	10.0	View Results
Java Substring	Java 7	6 minutes ago	Accepted ✓	5.0	View Results
Java Currency Formatter	Java 7	8 minutes ago	Accepted ✓	15.0	View Results
Java Int to String	Java 7	18 minutes ago	Accepted ✓	10.0	View Results
Java Static Initializer Block	Java 7	28 minutes ago	Accepted ✓	10.0	View Results
Java Static Initializer Block	Java 7	29 minutes ago	Accepted ✓	10.0	View Results
Java Static Initializer Block	Java 7	29 minutes ago	Accepted ✓	10.0	View Results
Java End-of-file	Java 7	32 minutes ago	Accepted ✓	10.0	View Results


Submissions | HackerRa

https://www.hackerrank.com/submissions/all/page/2

All Submissions

Submissions

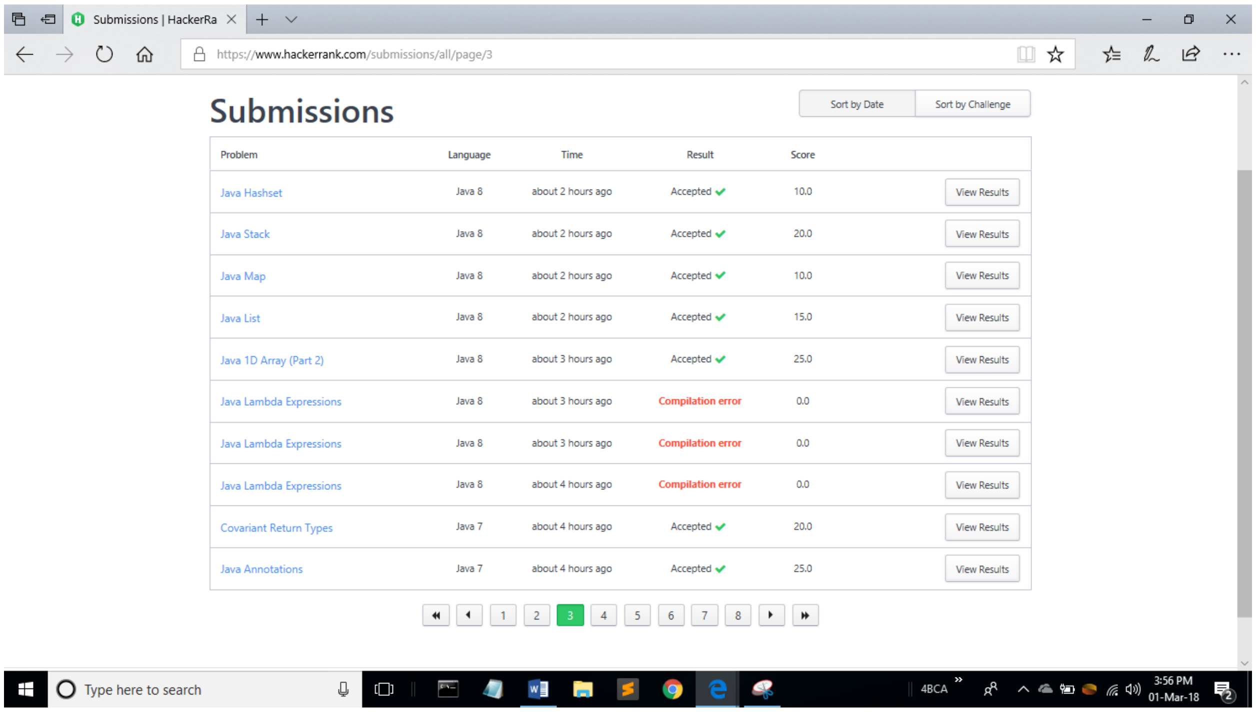
Sort by Date | Sort by Challenge

Problem	Language	Time	Result	Score	
Java Datatypes	Java 7	39 minutes ago	Accepted ✓	10.0	View Results
Java Datatypes	Java 7	40 minutes ago	Accepted ✓	10.0	View Results
Java Datatypes	Java 7	42 minutes ago	Accepted ✓	10.0	View Results
Java Loops II	Java 7	about 1 hour ago	Accepted ✓	10.0	View Results
Java Loops I	Java 7	about 1 hour ago	Accepted ✓	10.0	View Results
Java Output Formatting	Java 7	about 1 hour ago	Accepted ✓	10.0	View Results
Java Stdin and Stdout II	Java 7	about 1 hour ago	Accepted ✓	10.0	View Results
Java Stdin and Stdout II	Java 7	about 1 hour ago	Accepted ✓	10.0	View Results
Java Stdin and Stdout I	Java 7	about 1 hour ago	Accepted ✓	5.0	View Results
Welcome to Java!	Java 7	about 1 hour ago	Accepted ✓	3.0	View Results

1 2 3 4 5 6 7 8

Type here to search

4BCA 3:56 PM 01-Mar-18

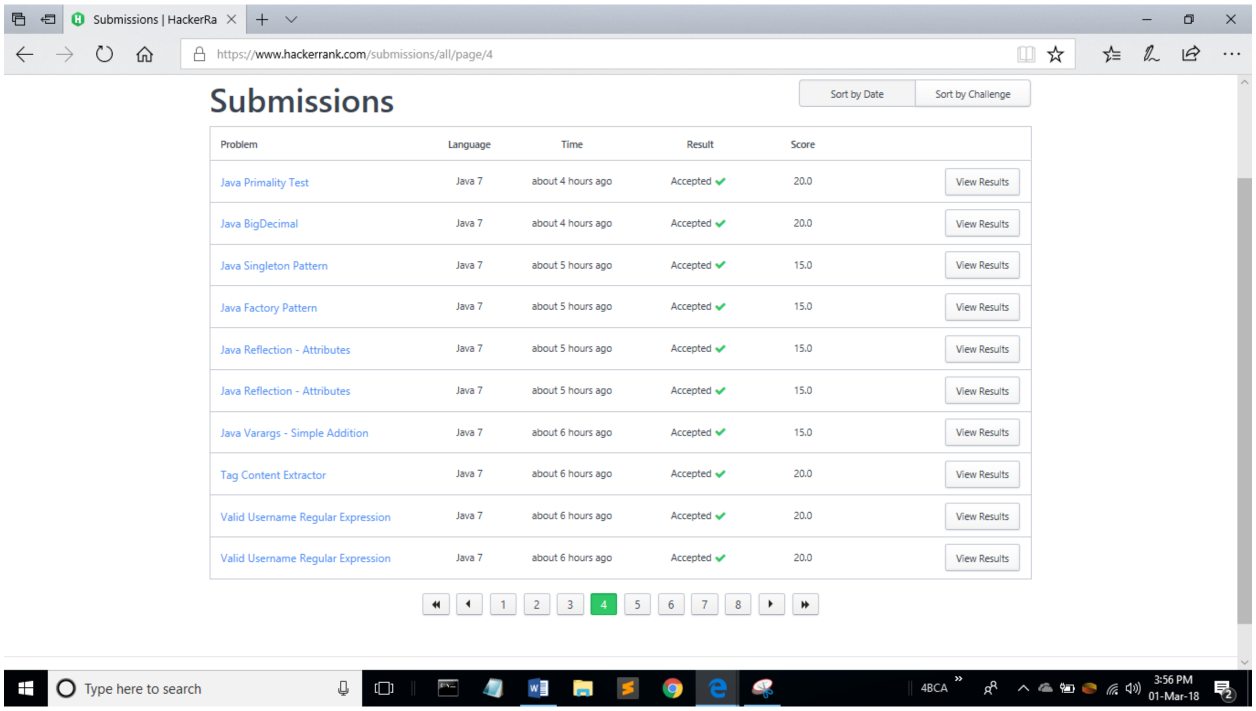


https://www.hackerrank.com/submissions/all/page/3

Submissions

Problem	Language	Time	Result	Score	
Java Hashset	Java 8	about 2 hours ago	Accepted ✓	10.0	View Results
Java Stack	Java 8	about 2 hours ago	Accepted ✓	20.0	View Results
Java Map	Java 8	about 2 hours ago	Accepted ✓	10.0	View Results
Java List	Java 8	about 2 hours ago	Accepted ✓	15.0	View Results
Java 1D Array (Part 2)	Java 8	about 3 hours ago	Accepted ✓	25.0	View Results
Java Lambda Expressions	Java 8	about 3 hours ago	Compilation error	0.0	View Results
Java Lambda Expressions	Java 8	about 3 hours ago	Compilation error	0.0	View Results
Java Lambda Expressions	Java 8	about 4 hours ago	Compilation error	0.0	View Results
Covariant Return Types	Java 7	about 4 hours ago	Accepted ✓	20.0	View Results
Java Annotations	Java 7	about 4 hours ago	Accepted ✓	25.0	View Results

3:56 PM 01-Mar-18

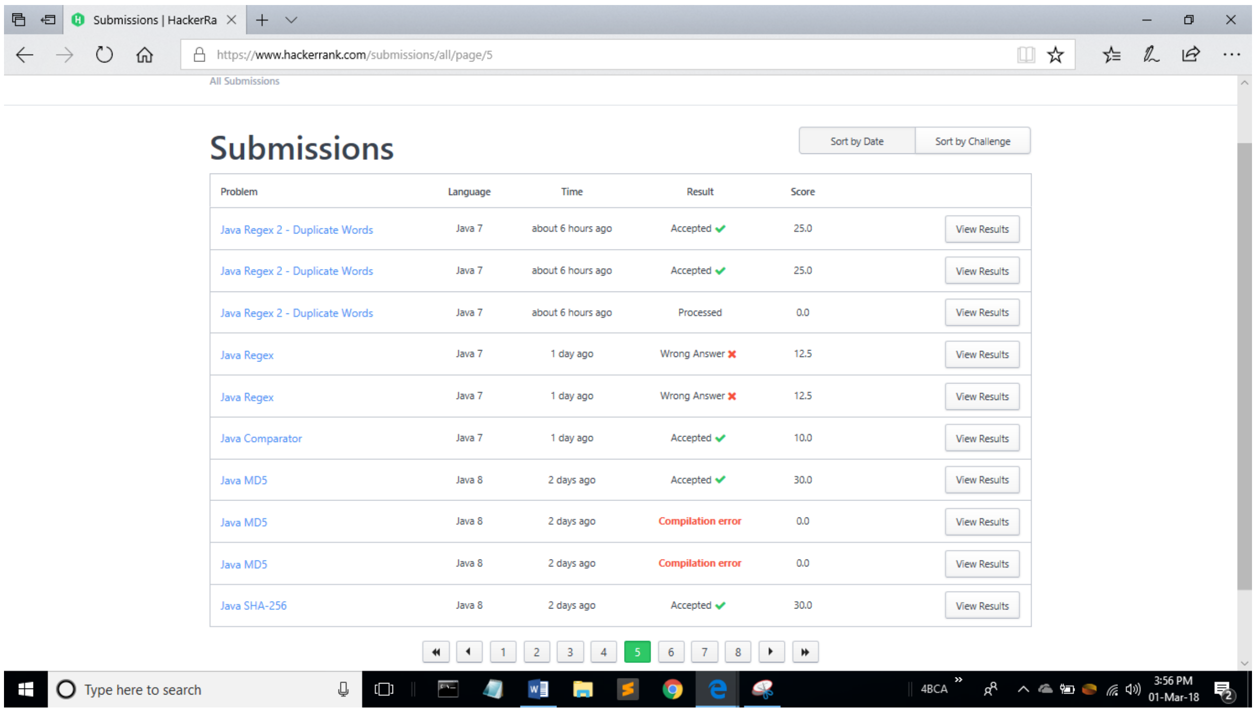


https://www.hackerrank.com/submissions/all/page/4

Submissions

Problem	Language	Time	Result	Score	
Java Primality Test	Java 7	about 4 hours ago	Accepted ✓	20.0	View Results
Java BigDecimal	Java 7	about 4 hours ago	Accepted ✓	20.0	View Results
Java Singleton Pattern	Java 7	about 5 hours ago	Accepted ✓	15.0	View Results
Java Factory Pattern	Java 7	about 5 hours ago	Accepted ✓	15.0	View Results
Java Reflection - Attributes	Java 7	about 5 hours ago	Accepted ✓	15.0	View Results
Java Reflection - Attributes	Java 7	about 5 hours ago	Accepted ✓	15.0	View Results
Java Varargs - Simple Addition	Java 7	about 6 hours ago	Accepted ✓	15.0	View Results
Tag Content Extractor	Java 7	about 6 hours ago	Accepted ✓	20.0	View Results
Valid Username Regular Expression	Java 7	about 6 hours ago	Accepted ✓	20.0	View Results
Valid Username Regular Expression	Java 7	about 6 hours ago	Accepted ✓	20.0	View Results

3:56 PM 01-Mar-18


Submissions | HackerRa

https://www.hackerrank.com/submissions/all/page/5

All Submissions

Submissions

Sort by Date | Sort by Challenge

Problem	Language	Time	Result	Score	
Java Regex 2 - Duplicate Words	Java 7	about 6 hours ago	Accepted ✓	25.0	View Results
Java Regex 2 - Duplicate Words	Java 7	about 6 hours ago	Accepted ✓	25.0	View Results
Java Regex 2 - Duplicate Words	Java 7	about 6 hours ago	Processed	0.0	View Results
Java Regex	Java 7	1 day ago	Wrong Answer ✗	12.5	View Results
Java Regex	Java 7	1 day ago	Wrong Answer ✗	12.5	View Results
Java Comparator	Java 7	1 day ago	Accepted ✓	10.0	View Results
Java MD5	Java 8	2 days ago	Accepted ✓	30.0	View Results
Java MD5	Java 8	2 days ago	Compilation error	0.0	View Results
Java MD5	Java 8	2 days ago	Compilation error	0.0	View Results
Java SHA-256	Java 8	2 days ago	Accepted ✓	30.0	View Results

1 2 3 4 5 6 7 8

Type here to search

4BCA 3:56 PM 01-Mar-18

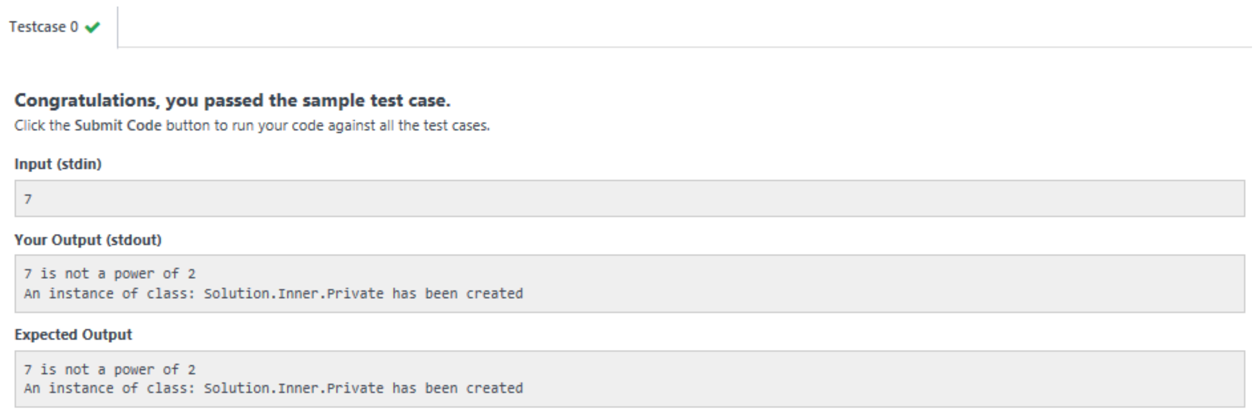
Q1. You are given a class *Solution* and an inner class *Inner.Private*. The main method of class *Solution* takes an integer num as input. The *powerof2* in class *Inner.Private* checks whether a number is a power of 2 . You have to call the method *powerof2* of the class *Inner.Private* from the *main* method of the class *Solution*.

Program:

```
import java.io.*;

import java.lang.reflect.*;

import java.util.*;

import java.util.regex.*;

import java.security.*;



public class Solution {
	public static void main(String[] args) throws Exception {
		DoNotTerminate.forbidExit();

		try{
			BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
			int num = Integer.parseInt(br.readLine().trim());
			Object o;// Must be used to hold the reference of the instance of the class
Solution.Inner.Private

//code
o = new Inner().new Private();
System.out.println(num + " is " + ((Solution.Inner.Private)o).powerof2(num));
System.out.println("An instance of class: " + o.getClass().getCanonicalName() + " has been created");

		}//end of try

		catch (DoNotTerminate.ExitTrappedException e) {
			System.out.println("Unsuccessful Termination!!");
		}
	}//end of main
	static class Inner{
		private class Private{
			private String powerof2(int num){
				return ((num&num-1)==0)?"power of 2":"not a power of 2";
			}
		}
	}//end of Inner

}//end of Solution
```

```
class DoNotTerminate { //This class prevents exit(0)

  public static class ExitTrappedException extends SecurityException {

                private static final long serialVersionUID = 1L;
  }

  public static void forbidExit() {
    final SecurityManager securityManager = new SecurityManager() {
      @Override
      public void checkPermission(Permission permission) {
        if (permission.getName().contains("exitVM")) {
          throw new ExitTrappedException();
        }
      }
    };
    System.setSecurityManager(securityManager);
  }
}
```

OUTPUT:



Testcase 0 ✓

Congratulations, you passed the sample test case.
Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

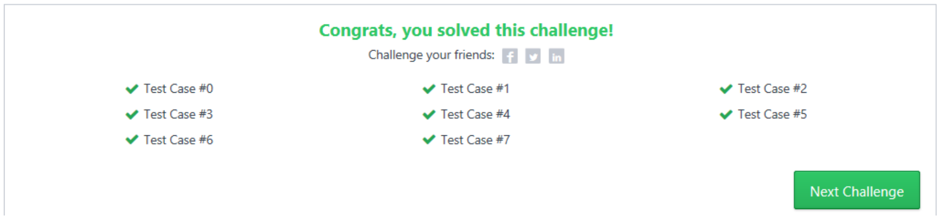
```
7
```

Your Output (stdout)

```
7 is not a power of 2
An instance of class: Solution.Inner.Private has been created
```

Expected Output

```
7 is not a power of 2
An instance of class: Solution.Inner.Private has been created
```



Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0	✓ Test Case #1	✓ Test Case #2
✓ Test Case #3	✓ Test Case #4	✓ Test Case #5
✓ Test Case #6	✓ Test Case #7	

Q2. You are given a class *Solution* and its *main* method in the editor. Your task is to create a class *Prime*. The class *Prime* should contain a single method *checkPrime*.
The locked code in the editor will call the *checkPrime* method with one or more integer arguments. You should write the *checkPrime* method in such a way that the code prints only the prime numbers.

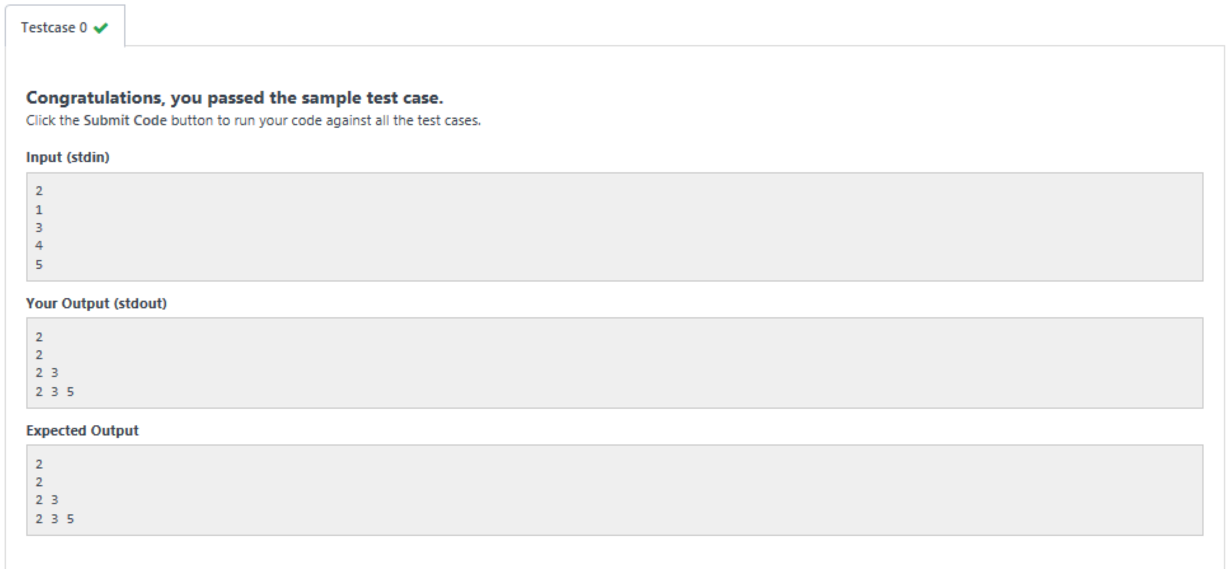
Please read the code given in the editor carefully. Also please do not use method overloading!

Program:

```
import java.io.*;

import java.util.*;

import java.text.*;

import java.math.*;

import java.util.regex.*;

import java.lang.reflect.*; import static java.lang.System.in;

class Prime {


    public void checkPrime(int... vars){

        for(int num : vars){

            boolean flag = true;

            if(num<2) flag=false;

            for(int i=2 ; i<num ; i++){

                if( num%i == 0)

                    flag=false;

            }

            System.out.print(flag==true?num+" ":"");

        }
        System.out.println();
    }
}public class Solution {

        public static void main(String[] args) {
                try{
                BufferedReader br=new BufferedReader(new InputStreamReader(in));
                int n1=Integer.parseInt(br.readLine());
                int n2=Integer.parseInt(br.readLine());
                int n3=Integer.parseInt(br.readLine());
                int n4=Integer.parseInt(br.readLine());
                int n5=Integer.parseInt(br.readLine());
                Prime ob=new Prime();
                ob.checkPrime(n1);
                ob.checkPrime(n1,n2);
                ob.checkPrime(n1,n2,n3);
                ob.checkPrime(n1,n2,n3,n4,n5);
                Method[] methods=Prime.class.getDeclaredMethods();
                Set<String> set=new HashSet<>();
                boolean overload=false;
                for(int i=0;i<methods.length;i++)
                {
                        if(set.contains(methods[i].getName()))
                        {
                                overload=true;
                                break;
                        }
                        set.add(methods[i].getName());

                }
```

```
                        if(overload)
                        {
                                throw new Exception("Overloading not allowed");
                        }
                        }
                        catch(Exception e)
                        {
                                System.out.println(e);
                        }
                }

        }
```

OUTPUT:



Testcase 0 ✓

Congratulations, you passed the sample test case.
Click the Submit Code button to run your code against all the test cases.

Input (stdin)

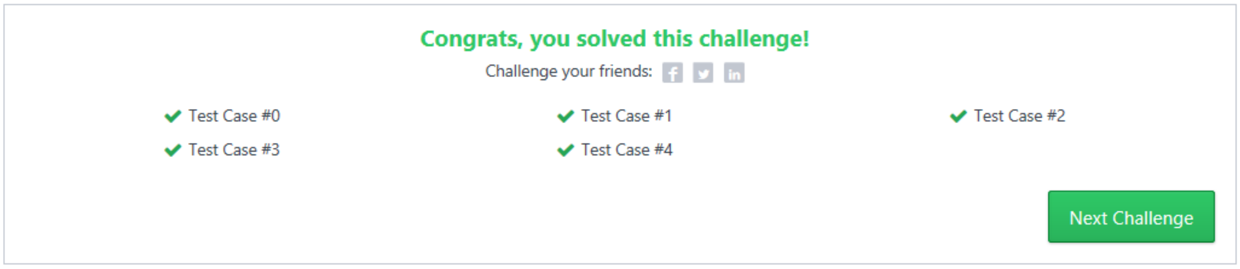
```
2
1
3
4
5
```

Your Output (stdout)

```
2
2
2 3
2 3 5
```

Expected Output

```
2
2
2 3
2 3 5
```

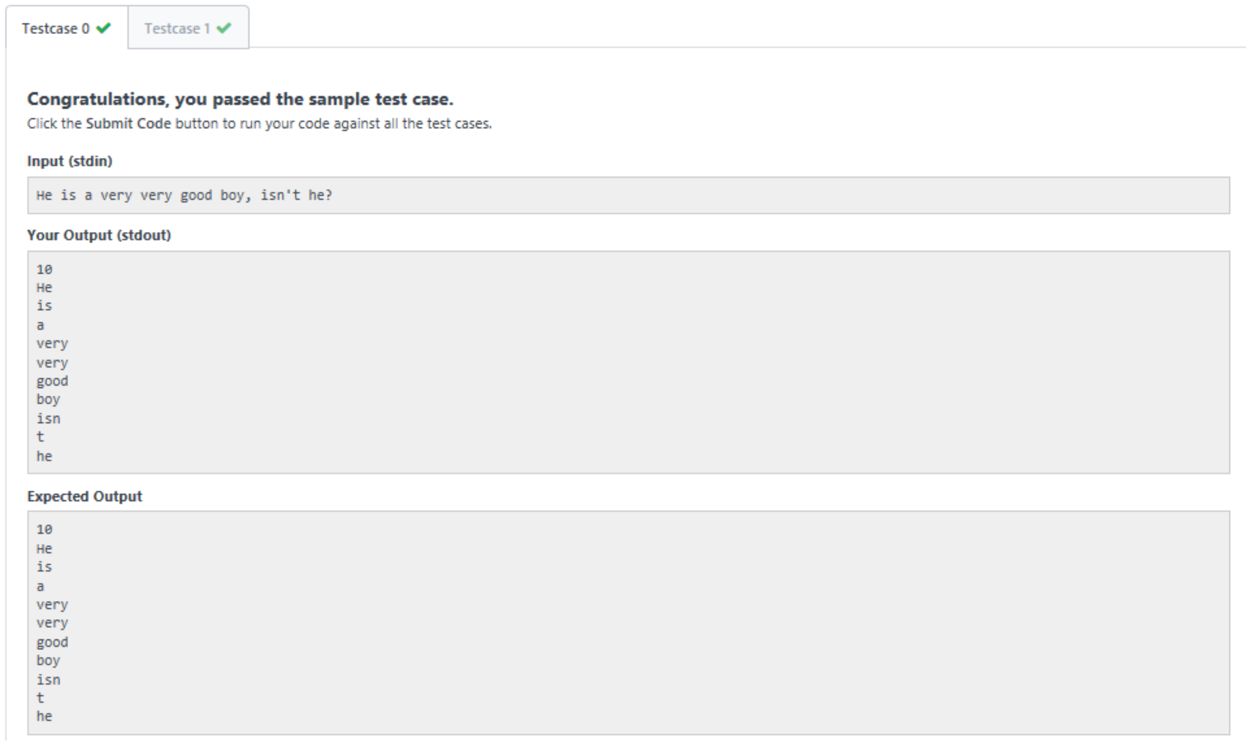

Congrats, you solved this challenge!

Q3. Given a string, s matching the regular expression [A-Za-z !,?._'@]+, **split the string into *tokens*. We define a token to be one or more consecutive English alphabetic letters. Then, print the number of tokens, followed by each token on a new line.**

Program:

```
import java.io.*;

import java.util.*;


public class Solution {


    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        String s = scan.nextLine();


    String[] items = s.trim().split("[ !,?.\\_'@]+");


    if (s == null || s.equals("") || s.trim().equals("")){

       System.out.println("0");

    }

    else if(s.length() > 400000){

        return;

    }

    else{
        System.out.println(items.length);
      }

      for(String item: items){
        System.out.println(item);
      }

      scan.close();
    }
  }
```

OUTPUT:



Testcase 0 ✓ Testcase 1 ✓

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

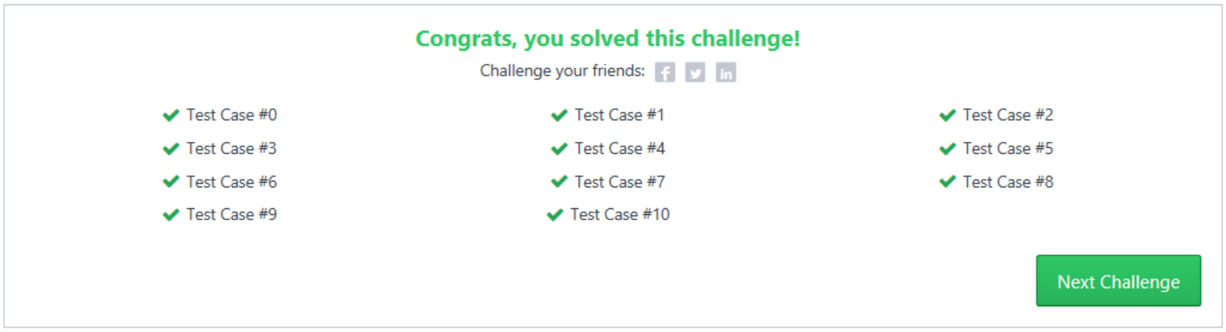
```
He is a very very good boy, isn't he?
```

Your Output (stdout)

```
10
He
is
a
very
very
good
boy
isn
t
he
```

Expected Output

```
10
He
is
a
very
very
good
boy
isn
t
he
```


Congrats, you solved this challenge!

Challenge your friends:

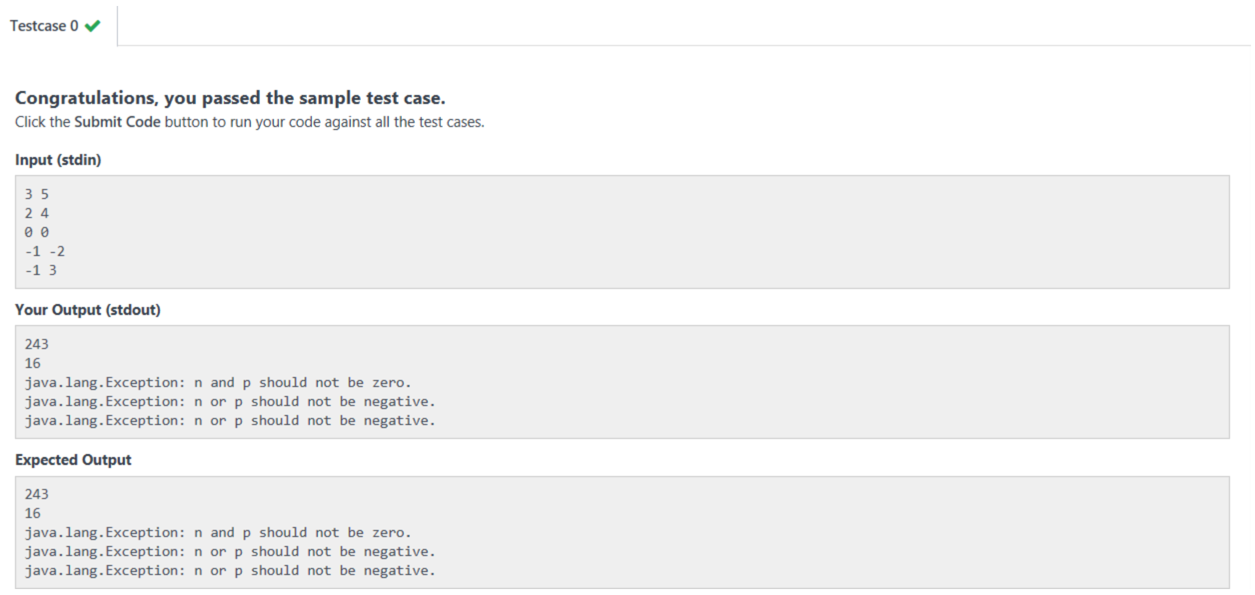
✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2
✓ Test Case #3 ✓ Test Case #4 ✓ Test Case #5
✓ Test Case #6 ✓ Test Case #7 ✓ Test Case #8
✓ Test Case #9 ✓ Test Case #10

Q4 You are required to compute the power of a number by implementing a calculator. Create a class *MyCalculator* which consists of a single method `long power(int, int)`. This method takes two integers, n and p, as parameters and finds npowerp . If either n or p is negative, then the method must throw an exception which says "n or p should not be negative". Also, if both n and p are zero, then the method must throw an exception which says "n and p should not be zero" For example, *-4* and *-5* would result in .java.lang.Exception: n or p should not be negative
Complete the function `power` in class *MyCalculator* and return the appropriate result after the power operation or an appropriate exception as detailed above.

Program:

```
import java.util.Scanner;
class MyCalculator {
    long power(int n, int p)
        throws Exception {
      if (n < 0 || p < 0) {
         throw new Exception("n or p should not be negative.");
      } else if (n == 0 && p == 0) {
         throw new Exception("n and p should not be zero.");
      } else {
         return (long) Math.pow(n, p);
      }
    }
}
```

OUTPUT:



Testcase 0 ✓

Congratulations, you passed the sample test case.
Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

```
3 5
2 4
0 0
-1 -2
-1 3
```

Your Output (stdout)

```
243
16
java.lang.Exception: n and p should not be zero.
java.lang.Exception: n or p should not be negative.
java.lang.Exception: n or p should not be negative.
```

Expected Output

```
243
16
java.lang.Exception: n and p should not be zero.
java.lang.Exception: n or p should not be negative.
java.lang.Exception: n or p should not be negative.
```

Q5. Exception handling is the process of responding to the occurrence, during computation, of exceptions – anomalous or exceptional conditions requiring special processing – often changing the normal flow of program execution. (Wikipedia)

Java has built-in mechanism to handle exceptions. Using the *try* statement we can test a block of code for errors. The *catch* block contains the code that says what to do if exception occurs.

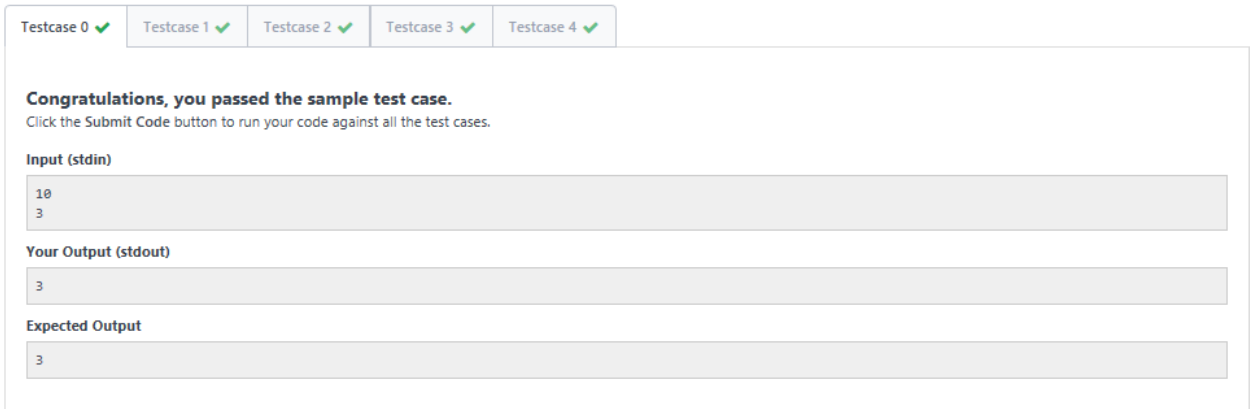
This problem will test your knowledge on try-catch block.

You will be given two integers x and y as input, you have to compute x/y. If x and y are not 32 bit signed integers or if is zero, exception will occur and you have to report it. Read sample Input/Output to know what to report in case of exceptions.

Program:

```
import java.io.*;

import java.util.*;

import java.text.*;

import java.math.*;

import java.util.regex.*;


public class Solution {


    public static void main(String[] args) {

try{

            Scanner sc = new Scanner(System.in);

            int x = sc.nextInt();

            int y = sc.nextInt();

            if(y==0)

                throw  new ArithmeticException("/ by zero");

            else

                System.out.println(x/y);

        }

        catch(InputMismatchException e){

            System.out.println(e.getClass().getName());

        }


        catch(ArithmeticException e){

            System.out.println(e);

        }

          /* Enter your code here. Read input from STDIN. Print output to STDOUT. Your class should be
named Solution. */

    }

}
```

OUTPUT:



Testcase 0 ✓ Testcase 1 ✓ Testcase 2 ✓ Testcase 3 ✓ Testcase 4 ✓

Congratulations, you passed the sample test case.
Click the Submit Code button to run your code against all the test cases.

Input (stdin)

```
10
3
```

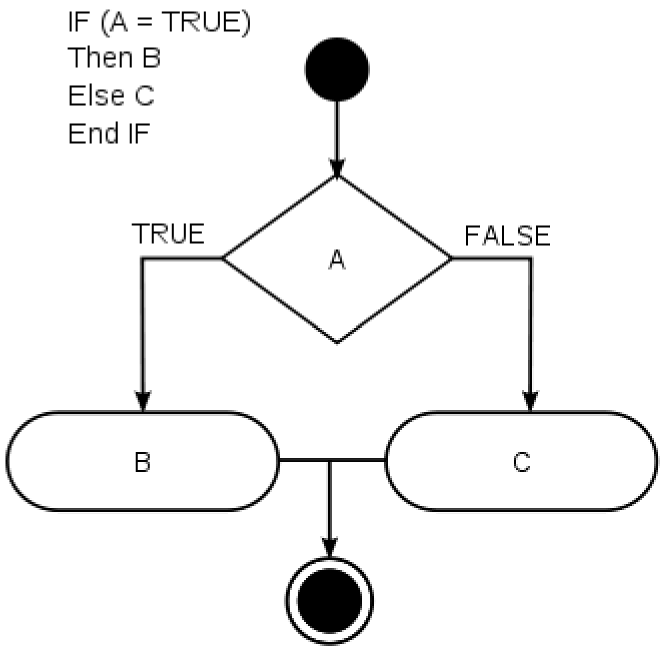
Your Output (stdout)

```
3
```

Expected Output

```
3
```

Q6. In this challenge, we test your knowledge of using *if-else* conditional statements to automate decision-making processes. An if-else statement has the following logical flow:



Task

Given an integer,n , perform the following conditional actions:

- If is n odd, print `Weird`

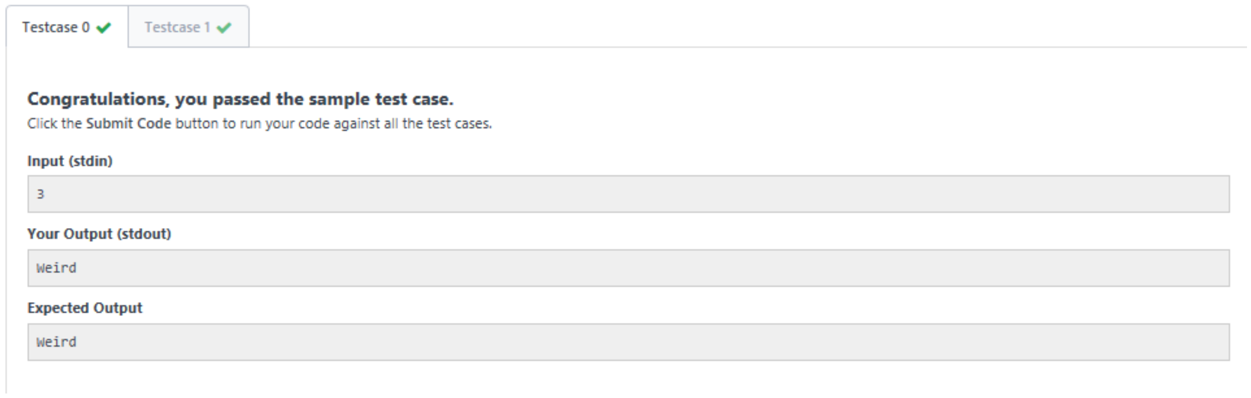
- If is n even and in the inclusive range of 2 to 5, print `Not Weird`
- If is n even and in the inclusive range of 6 to 20 , print `Weird`
- If is n even and greater than 20, print `Not Weird`

Complete the stub code provided in your editor to print whether or not n is weird.

Program:

```
import java.io.*;

import java.util.*;

import java.text.*;

import java.math.*;

import java.util.regex.*;


public class Solution {


    public static void main(String[] args) {


        Scanner sc=new Scanner(System.in);

        int n=sc.nextInt();

        String ans="";

        if(n%2==1){

          ans = "Weird";

        }

        else


          if(n%2==0&&(n>=2&&n<=5))

          {

             ans = "Not Weird"; //Complete the code

          }

            else

               if(n%2==0&&(n>=6&&n<=20))
            {
              ans = "Weird"; //Complete the code
            }
          else
            {
              ans = "Not Weird"; //Complete the code
            }
          System.out.println(ans);


          }


        }
```

OUTPUT:


Testcase 0 ✓ Testcase 1 ✓

Congratulations, you passed the sample test case.
Click the Submit Code button to run your code against all the test cases.

Input (stdin)

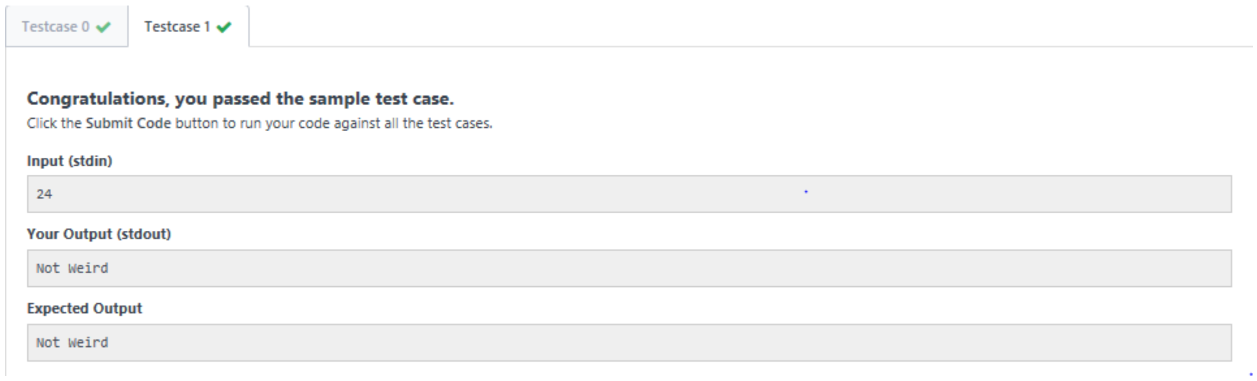
```
3
```

Your Output (stdout)

```
Weird
```

Expected Output

```
Weird
```


Testcase 0 ✓ Testcase 1 ✓

Congratulations, you passed the sample test case.
Click the Submit Code button to run your code against all the test cases.

Input (stdin)

```
24
```

Your Output (stdout)

```
Not Weird
```

Expected Output

```
Not Weird
```

Q7 **Complete the function solveMeFirst to compute the sum of two integers.**

Function prototype:

int solveMeFirst(int x, int y);

where,

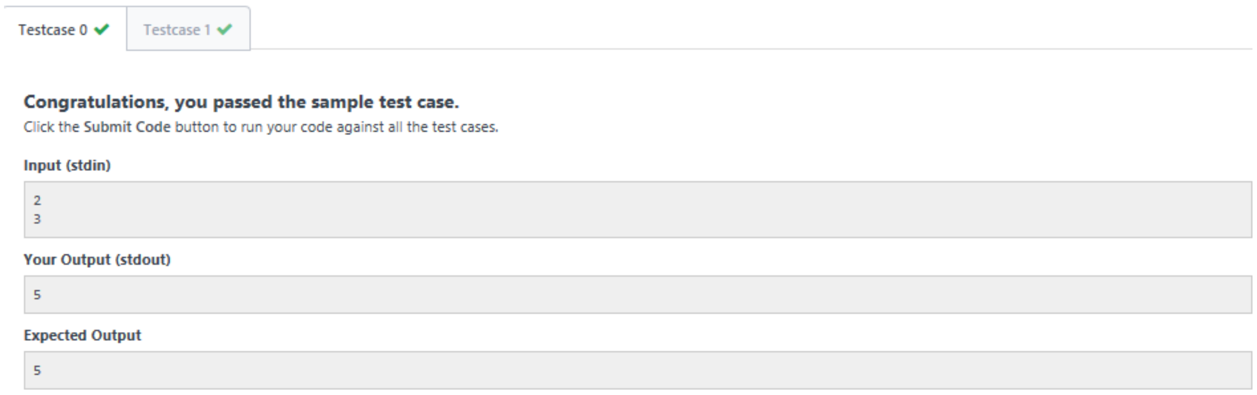
- **x is the first integer input.**
- **y is the second integer input**

Return values

- **sum of the above two integers**

```
import java.io.*;

import java.util.*;

import java.text.*;

import java.math.*;

import java.util.regex.*;


public class Solution {



   static int solveMeFirst(int a, int b) {

     return (a+b);


  }



 public static void main(String[] args) {

        Scanner in = new Scanner(System.in);

        int a;

        a = in.nextInt();
        int b;

        b = in.nextInt();

        int sum;

        sum = solveMeFirst(a, b);

        System.out.println(sum);

   }

}
```

OUTPUT:



Congratulations, you passed the sample test case.
Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

```
2
3
```

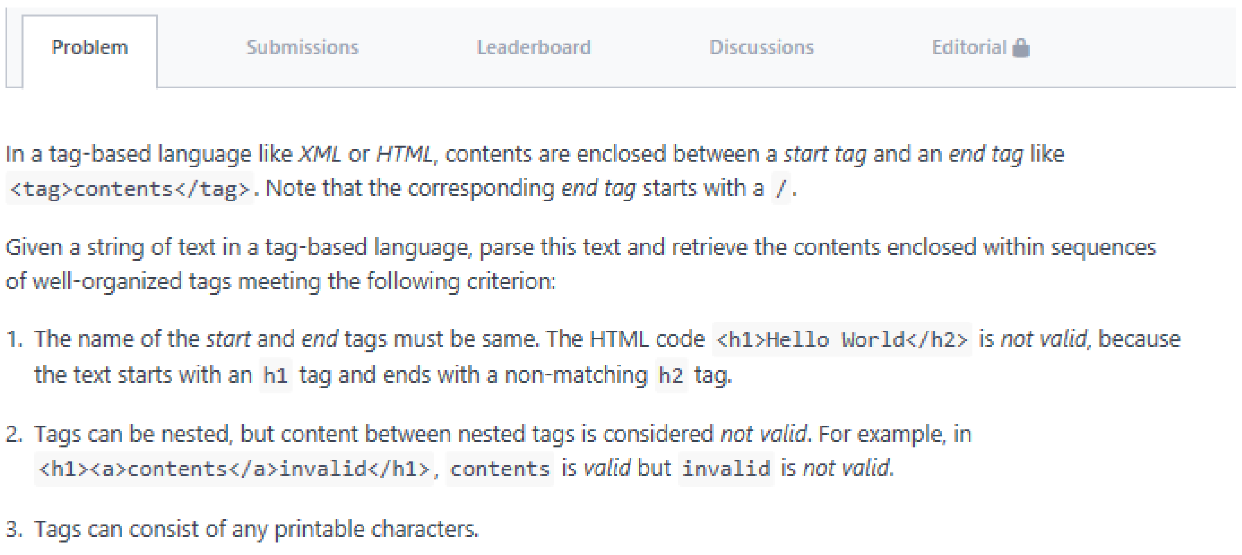
Your Output (stdout)

```
5
```

Expected Output

```
5
```

Q8



In a tag-based language like *XML* or *HTML*, contents are enclosed between a *start tag* and an *end tag* like `<tag>contents</tag>`. Note that the corresponding *end tag* starts with a `/`.

Given a string of text in a tag-based language, parse this text and retrieve the contents enclosed within sequences of well-organized tags meeting the following criterion:

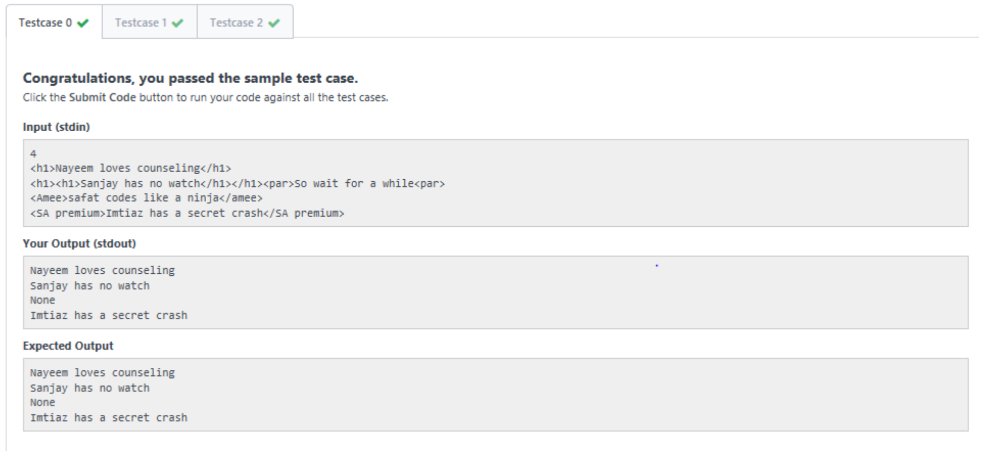
1. The name of the *start* and *end* tags must be same. The HTML code `<h1>Hello World</h2>` is *not valid*, because the text starts with an `h1` tag and ends with a non-matching `h2` tag.
2. Tags can be nested, but content between nested tags is considered *not valid*. For example, in `<h1><a>contents</a>invalid</h1>`, `contents` is *valid* but `invalid` is *not valid*.
3. Tags can consist of any printable characters.

Program:

```
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

public class Solution{
  public static void main(String[] args){

    Scanner in = new Scanner(System.in);
    int testCases = Integer.parseInt(in.nextLine());
    while(testCases>0){
      String line = in.nextLine();

        int count=0;
      Pattern r = Pattern.compile("<(.+?)>([^<>]+)</\\1>");
      Matcher m = r.matcher(line);
```

```
        while(m.find()) {
          if (m.group(2).length() !=0) {
            System.out.println(m.group(2));
          count++;
          }
        }
        if (count == 0) System.out.println("None");  //Write your code here

        testCases--;
      }
    }
  }
```

OUTPUT:



Testcase 0 ✓ Testcase 1 ✓ Testcase 2 ✓

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

```
4
<h1>Nayeem loves counseling</h1>
<h1><h1>Sanjay has no watch</h1></h1><par>So wait for a while<par>
<Amee>safat codes like a ninja</amee>
<SA premium>Imtiaz has a secret crash</SA premium>
```

Your Output (stdout)

```
Nayeem loves counseling
Sanjay has no watch
None
Imtiaz has a secret crash
```

Expected Output

```
Nayeem loves counseling
Sanjay has no watch
None
Imtiaz has a secret crash
```

Q9

This Java 8 challenge tests your knowledge of Lambda expressions!

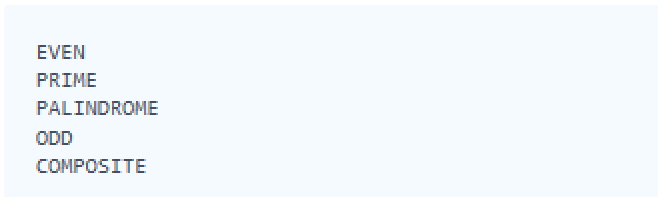
Write the following methods that *return a lambda expression* performing a specified action:

1. PerformOperation isOdd(): The lambda expression must return $true$ if a number is odd or $false$ if it is even.
2. PerformOperation isPrime(): The lambda expression must return $true$ if a number is prime or $false$ if it is composite.
3. PerformOperation isPalindrome(): The lambda expression must return $true$ if a number is a palindrome or $false$ if it is not.

Program:

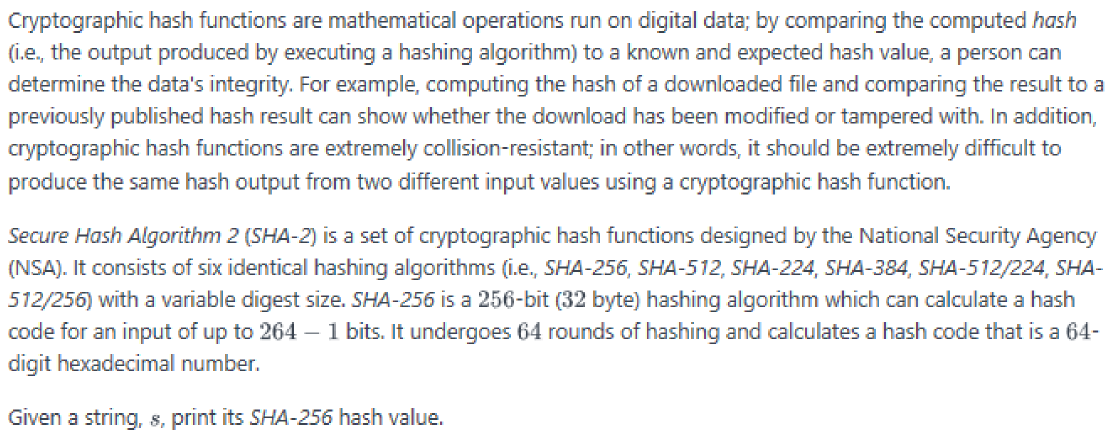
```
import java.io.*;
import java.util.*;
interface PerformOperation {
 boolean check(int a);
}
class MyMath {
 public static boolean checker(PerformOperation p, int num) {
  return p.check(num);
 }
public static PerformOperation is_odd(){
        return (int a) -> a % 2 != 0;
    }

    public static PerformOperation is_prime(){
        return (int a) -> java.math.BigInteger.valueOf(a).isProbablePrime(1);
    }

    public static PerformOperation is_palindrome(){
        return (int a) ->  Integer.toString(a).equals( new
StringBuilder(Integer.toString(a)).reverse().toString() );
    }public class Solution {

 public static void main(String[] args) throws IOException {
  MyMath ob = new MyMath();
  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
  int T = Integer.parseInt(br.readLine());
  PerformOperation op;
  boolean ret = false;
  String ans = null;
  while (T--> 0) {
   String s = br.readLine().trim();
   StringTokenizer st = new StringTokenizer(s);
   int ch = Integer.parseInt(st.nextToken());
   int num = Integer.parseInt(st.nextToken());
   if (ch == 1) {
    op = ob.isOdd();
    ret = ob.checker(op, num);
    ans = (ret) ? "ODD" : "EVEN";
   } else if (ch == 2) {
    op = ob.isPrime();
    ret = ob.checker(op, num);
    ans = (ret) ? "PRIME" : "COMPOSITE";
   } else if (ch == 3) {
    op = ob.isPalindrome();
    ret = ob.checker(op, num);
    ans = (ret) ? "PALINDROME" : "NOT PALINDROME";

   }
   System.out.println(ans);
  }
 }
}
```

OUTPUT:



```
EVEN
PRIME
PALINDROME
ODD
COMPOSITE
```

Q10



Cryptographic hash functions are mathematical operations run on digital data; by comparing the computed *hash* (i.e., the output produced by executing a hashing algorithm) to a known and expected hash value, a person can determine the data's integrity. For example, computing the hash of a downloaded file and comparing the result to a previously published hash result can show whether the download has been modified or tampered with. In addition, cryptographic hash functions are extremely collision-resistant; in other words, it should be extremely difficult to produce the same hash output from two different input values using a cryptographic hash function.

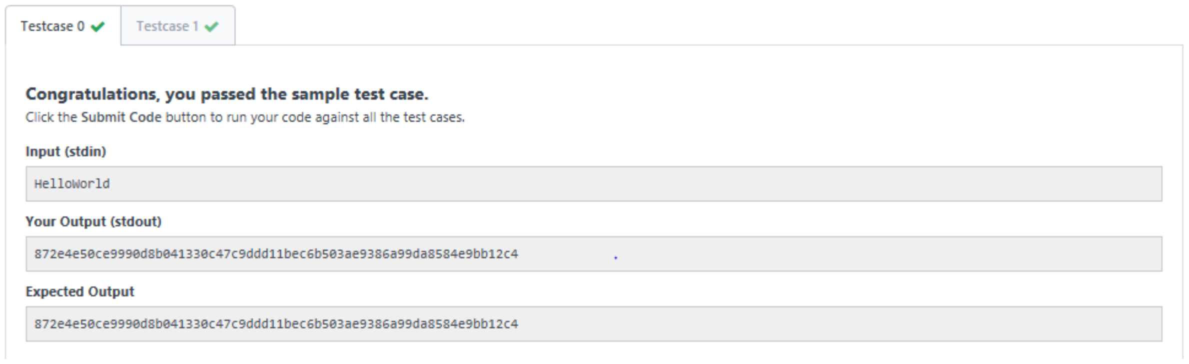
Secure Hash Algorithm 2 (*SHA-2*) is a set of cryptographic hash functions designed by the National Security Agency (NSA). It consists of six identical hashing algorithms (i.e., *SHA-256*, *SHA-512*, *SHA-224*, *SHA-384*, *SHA-512/224*, *SHA-512/256*) with a variable digest size. *SHA-256* is a 256-bit (32 byte) hashing algorithm which can calculate a hash code for an input of up to $264 - 1$ bits. It undergoes 64 rounds of hashing and calculates a hash code that is a 64-digit hexadecimal number.

Given a string, s, print its *SHA-256* hash value.

Program:

```
 import java.util.*;
import java.security.*;

public class Solution {

    public static void main(String[] args) throws NoSuchAlgorithmException {
        Scanner input = new Scanner(System.in);
        MessageDigest m = MessageDigest.getInstance("SHA-256");
        m.reset();
        m.update(input.nextLine().getBytes());
        for (byte i : m.digest()) {
            System.out.print(String.format("%02x", i));
        }
        System.out.println();
    }
}
```

OUTPUT:



Testcase 0 ✓ Testcase 1 ✓

Congratulations, you passed the sample test case.
Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

```
HelloWorld
```

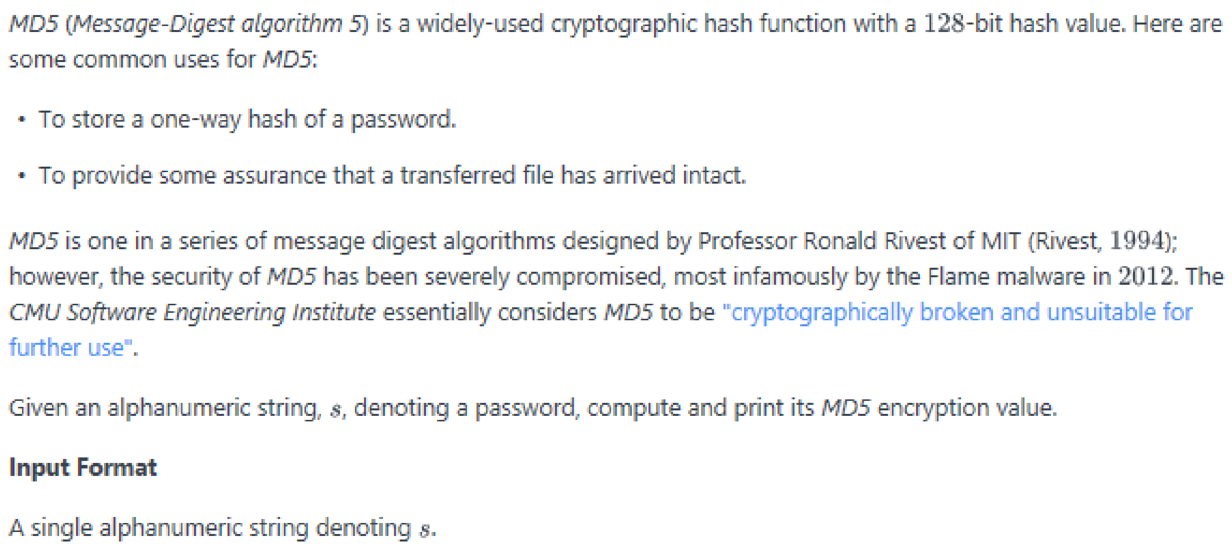
Your Output (stdout)

```
872e4e50ce9990d8b041330c47c9ddd11bec6b503ae9386a99da8584e9bb12c4
```

Expected Output

```
872e4e50ce9990d8b041330c47c9ddd11bec6b503ae9386a99da8584e9bb12c4
```

Q11



MD5 (*Message-Digest algorithm 5*) is a widely-used cryptographic hash function with a 128-bit hash value. Here are some common uses for *MD5*:

- To store a one-way hash of a password.
- To provide some assurance that a transferred file has arrived intact.

MD5 is one in a series of message digest algorithms designed by Professor Ronald Rivest of MIT (Rivest, 1994); however, the security of *MD5* has been severely compromised, most infamously by the Flame malware in 2012. The *CMU Software Engineering Institute* essentially considers *MD5* to be "cryptographically broken and unsuitable for further use".

Given an alphanumeric string, s, denoting a password, compute and print its *MD5* encryption value.

Input Format

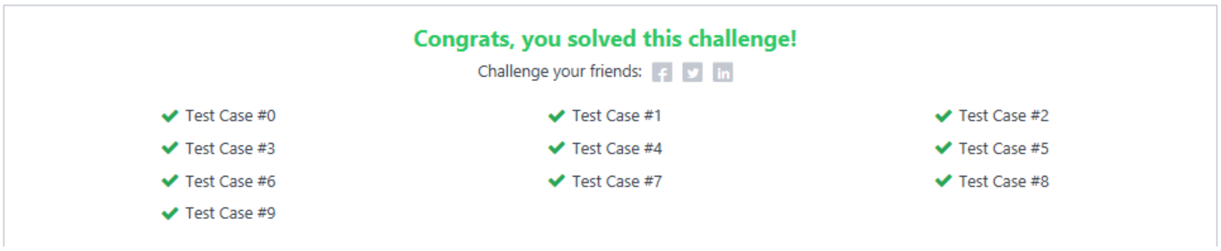
A single alphanumeric string denoting s.

Program:

```
import java.io.*;
import java.util.*;
import java.security.MessageDigest;
import java.util.Scanner;
```

```
public class Solution {
    public static void main(String[] argh) {
        Scanner sc = new Scanner(System.in);
        String str = sc.next();
        sc.close();
        try {
            MessageDigest md = MessageDigest.getInstance("MD5");
            md.update(str.getBytes());
            // return bytesToHex(md.digest
            byte[] digest = md.digest();
            for (byte b : digest) {
                System.out.printf("%02x", b);
            }
        } catch (Exception ex) {
            throw new RuntimeException(ex);
        }
    }
}
```

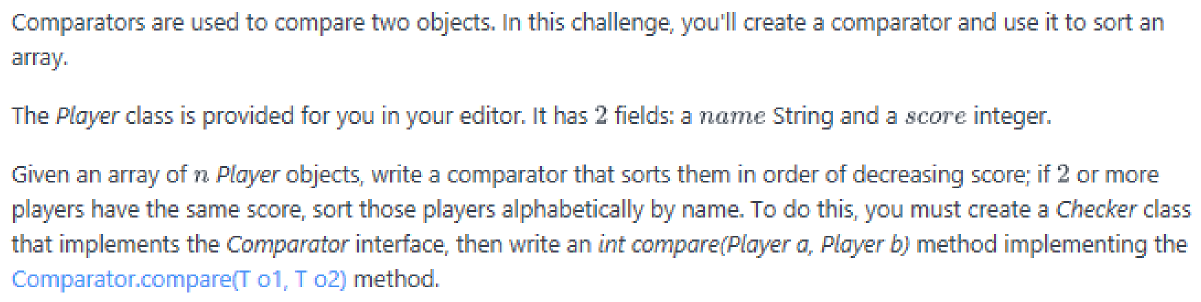
OUTPUT:


Congrats, you solved this challenge!

Challenge your friends:

```
✓ Test Case #0        ✓ Test Case #1        ✓ Test Case #2
✓ Test Case #3        ✓ Test Case #4        ✓ Test Case #5
✓ Test Case #6        ✓ Test Case #7        ✓ Test Case #8
✓ Test Case #9
```

Q12



Comparators are used to compare two objects. In this challenge, you'll create a comparator and use it to sort an array.

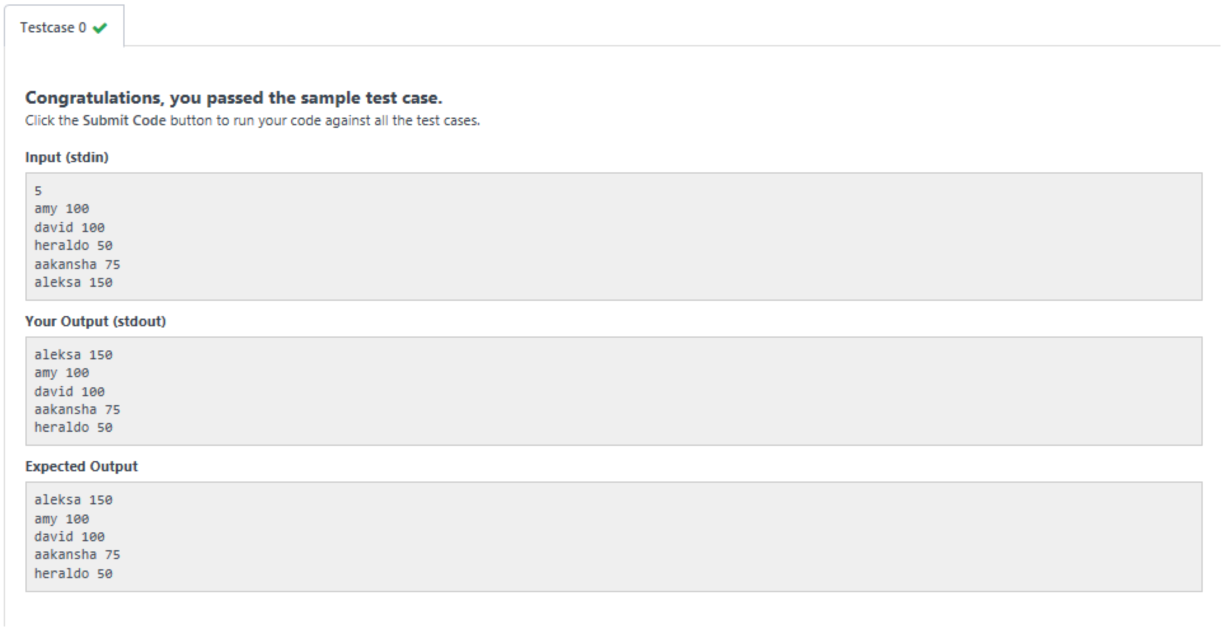
The *Player* class is provided for you in your editor. It has 2 fields: a $name$ String and a $score$ integer.

Given an array of n *Player* objects, write a comparator that sorts them in order of decreasing score; if 2 or more players have the same score, sort those players alphabetically by name. To do this, you must create a *Checker* class that implements the *Comparator* interface, then write an *int compare(Player a, Player b)* method implementing the Comparator.compare(T o1, T o2) method.

Program:

```
import java.util.*;
// Write your Checker class here
class Checker implements Comparator<Player> {
  @Override
  public int compare(Player p1, Player p2) {
    if (p2.score == p1.score) {
      return p1.name.compareTo(p2.name);
    } else {
      return p1.score > p2.score ? -1 : 1;
    }
  }
}class Player{
  String name;
  int score;

  Player(String name, int score){
    this.name = name;
    this.score = score;
  }
}

class Solution {
  public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    int n = scan.nextInt();

    Player[] player = new Player[n];
    Checker checker = new Checker();

    for(int i = 0; i < n; i++){
      player[i] = new Player(scan.next(), scan.nextInt());
    }
    scan.close();

    Arrays.sort(player, checker);
    for(int i = 0; i < player.length; i++){
      System.out.printf("%s %s\n", player[i].name, player[i].score);
    }
  }
}
```

OUTPUT:


Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

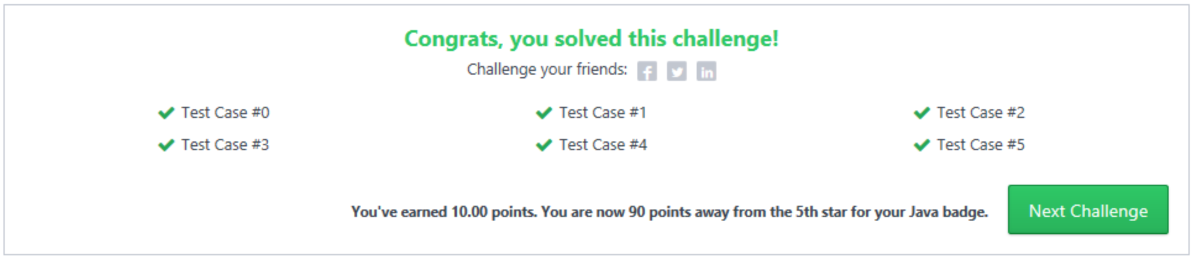
```
5
amy 100
david 100
heraldo 50
aakansha 75
aleksa 150
```

Your Output (stdout)

```
aleksa 150
amy 100
david 100
aakansha 75
heraldo 50
```

Expected Output

```
aleksa 150
amy 100
david 100
aakansha 75
heraldo 50
```


Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2

✓ Test Case #3 ✓ Test Case #4 ✓ Test Case #5

You've earned 10.00 points. You are now 90 points away from the 5th star for your Java badge.

Q13

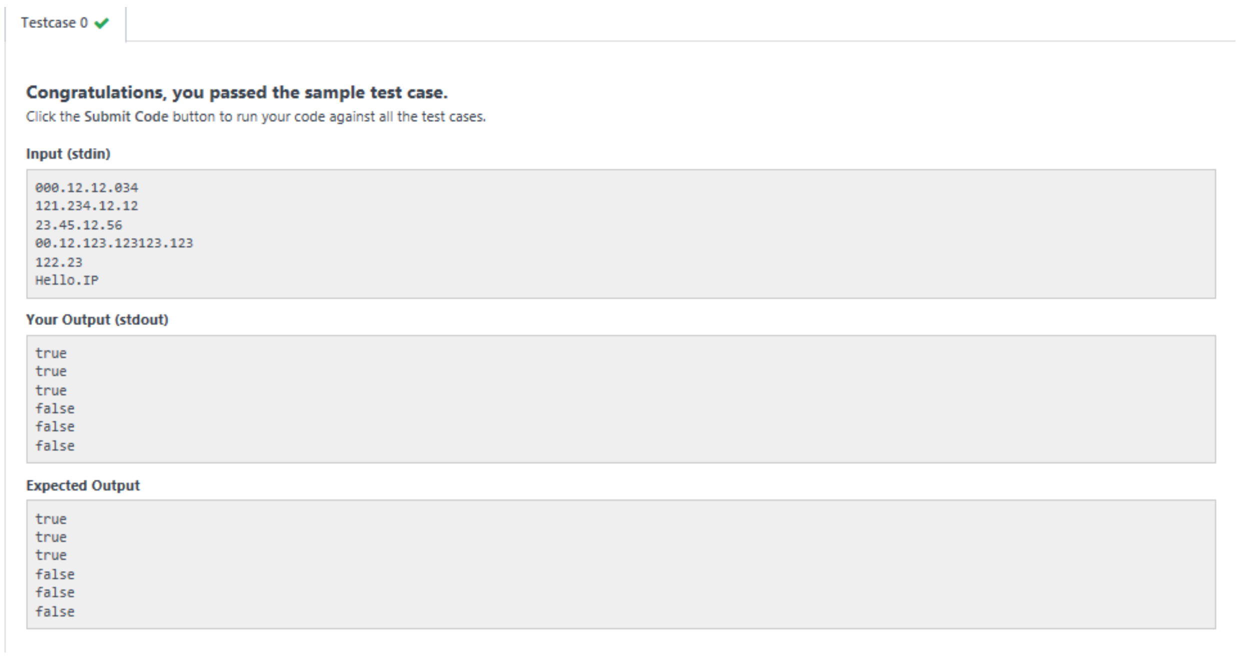
Write a class called *MyRegex* which will contain a string pattern. You need to write a regular expression and assign it to the pattern such that it can be used to validate an IP address. Use the following definition of an IP address:

```
IP address is a string in the form "A.B.C.D", where the value of A, B, C, and D may range from 0 to 2
55. Leading zeros are allowed. The length of A, B, C, or D can't be greater than 3.
```

Program:

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.Scanner;

class Solution{

    public static void main(String[] args){
        Scanner in = new Scanner(System.in);
        while(in.hasNext()){
            String IP = in.next();
            System.out.println(IP.matches(new MyRegex().pattern));
        }

    }
}class MyRegex{
    String Zip="([\\d]{1,2}|(0|1)[\\d]{1,2}|(2)[0-5]{1,2})";
    String pattern=Zip + "\\."+Zip+"\\."+Zip+"\\."+Zip;
}
```

OUTPUT:


Testcase 0 ✓

Congratulations, you passed the sample test case.
Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

```
000.12.12.034
121.234.12.12
23.45.12.56
00.12.123.123123.123
122.23
Hello.IP
```

Your Output (stdout)

```
true
true
true
false
false
false
```

Expected Output

```
true
true
true
false
false
false
```

Q14

In this challenge, we use regular expressions (RegEx) to remove instances of words that are repeated more than once, but retain the *first occurrence* of any case-insensitive repeated word. For example, the words `love` and `to` are repeated in the sentence `I love Love to To tO code`. Can you complete the code in the editor so it will turn `I love Love to To tO code` into `I love to code`?

To solve this challenge, complete the following three lines:

1. Write a RegEx that will match any repeated word.
2. Complete the second *compile* argument so that the compiled RegEx is case-insensitive.
3. Write the two necessary arguments for *replaceAll* such that each repeated word is replaced with the very first instance the word found in the sentence. It must be the *exact* first occurrence of the word, as the expected output is case-sensitive.

Program:

import java.util.Scanner;

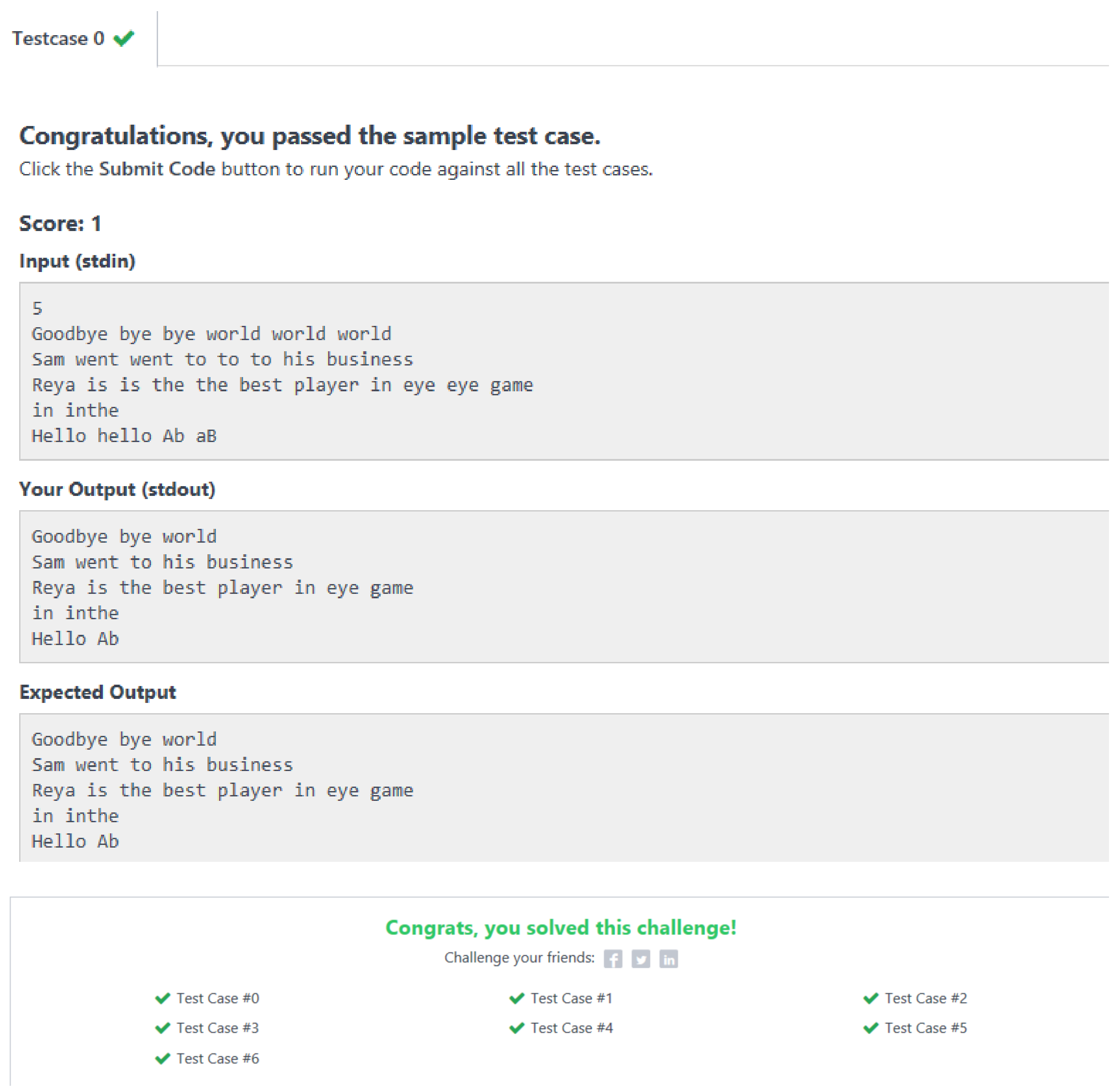
import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class DuplicateWords {

```
    public static void main(String[] args) {

        String regex = "\\b(\\w+)(\\W+\\1\\b)+";
        Pattern p = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);

        Scanner in = new Scanner(System.in);
        int numSentences = Integer.parseInt(in.nextLine());

        while (numSentences-- > 0) {
            String input = in.nextLine();

            Matcher m = p.matcher(input);

            // Check for subsequences of input that match the compiled pattern
            while (m.find()) {
                input = input.replaceAll(m.group(), m.group(1));
            }

            // Prints the modified sentence.
            System.out.println(input);
        }

        in.close();
    }
}
```

OUTPUT:



Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Score: 1

Input (stdin)

```
5
Goodbye bye bye world world world
Sam went went to to to his business
Reya is is the the best player in eye eye game
in inthe
Hello hello Ab aB
```

Your Output (stdout)

```
Goodbye bye world
Sam went to his business
Reya is the best player in eye game
in inthe
Hello Ab
```

Expected Output

```
Goodbye bye world
Sam went to his business
Reya is the best player in eye game
in inthe
Hello Ab
```

Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2
✓ Test Case #3 ✓ Test Case #4 ✓ Test Case #5
✓ Test Case #6

Q15

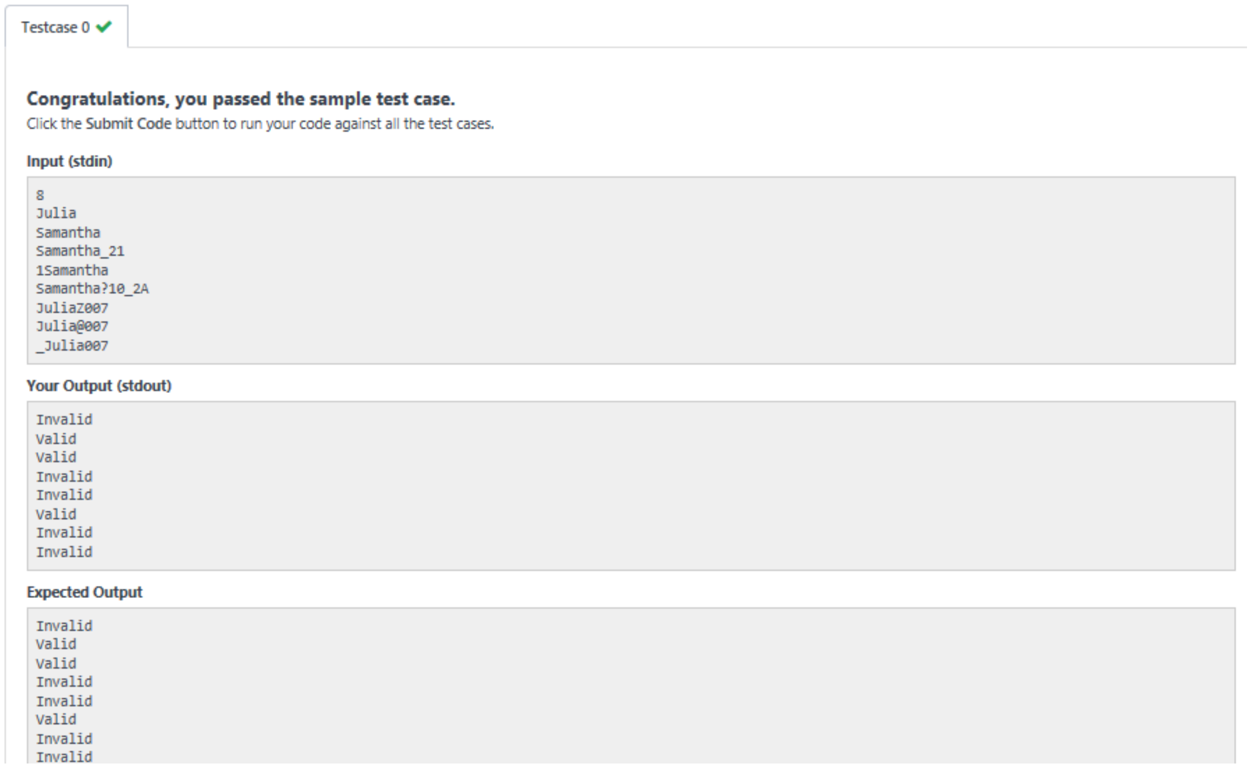
You are updating the username policy on your company's internal networking platform. According to the policy, a username is considered valid if all the following constraints are satisfied:

- The username consists of 8 to 30 characters inclusive. If the username consists of less than 8 or greater than 30 characters, then it is an invalid username.
- The username can only contain alphanumeric characters and underscores (_). Alphanumeric characters describe the character set consisting of *lowercase* characters $[a-z]$, *uppercase* characters $[A-Z]$, and digits $[0-9]$.
- The *first* character of the username must be an *alphabetic* character, i.e., either *lowercase* character $[a-z]$ or *uppercase* character $[A-Z]$.

Program:

```
import java.util.Scanner;


class UsernameValidator {
  String pattern = "^[A-ZA-z]\\w{7,29}$";
   public static final String regularExpression = "\\b[A-Z|a-z]\\w{7,29}$";
}


public class Solution {
   private static final Scanner scan = new Scanner(System.in);


   public static void main(String[] args) {
      int n = Integer.parseInt(scan.nextLine());
      while (n-- != 0) {
         String userName = scan.nextLine();


         if (userName.matches(UsernameValidator.regularExpression)) {
            System.out.println("Valid");
         } else {
            System.out.println("Invalid");
         }
      }
   }
}
```

OUTPUT:



Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

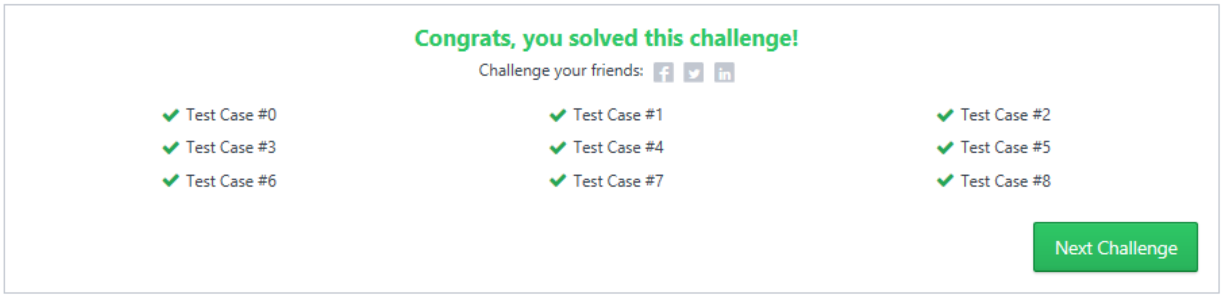
```
8
Julia
Samantha
Samantha_21
1Samantha
Samantha?10_2A
JuliaZ007
Julia@007
_Julia007
```

Your Output (stdout)

```
Invalid
Valid
Valid
Invalid
Invalid
Valid
Invalid
Invalid
```

Expected Output

```
Invalid
Valid
Valid
Invalid
Invalid
Valid
Invalid
Invalid
```



Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0	✓ Test Case #1	✓ Test Case #2
✓ Test Case #3	✓ Test Case #4	✓ Test Case #5
✓ Test Case #6	✓ Test Case #7	✓ Test Case #8

Q16

In a tag-based language like *XML* or *HTML*, contents are enclosed between a *start tag* and an *end tag* like `<tag>contents</tag>`. Note that the corresponding *end tag* starts with a `/`.

Given a string of text in a tag-based language, parse this text and retrieve the contents enclosed within sequences of well-organized tags meeting the following criterion:

1. The name of the *start* and *end* tags must be same. The HTML code `<h1>Hello World</h2>` is *not valid*, because the text starts with an `h1` tag and ends with a non-matching `h2` tag.

2. Tags can be nested, but content between nested tags is considered *not valid*. For example, in `<h1><a>contents</a>invalid</h1>`, `contents` is *valid* but `invalid` is *not valid*.

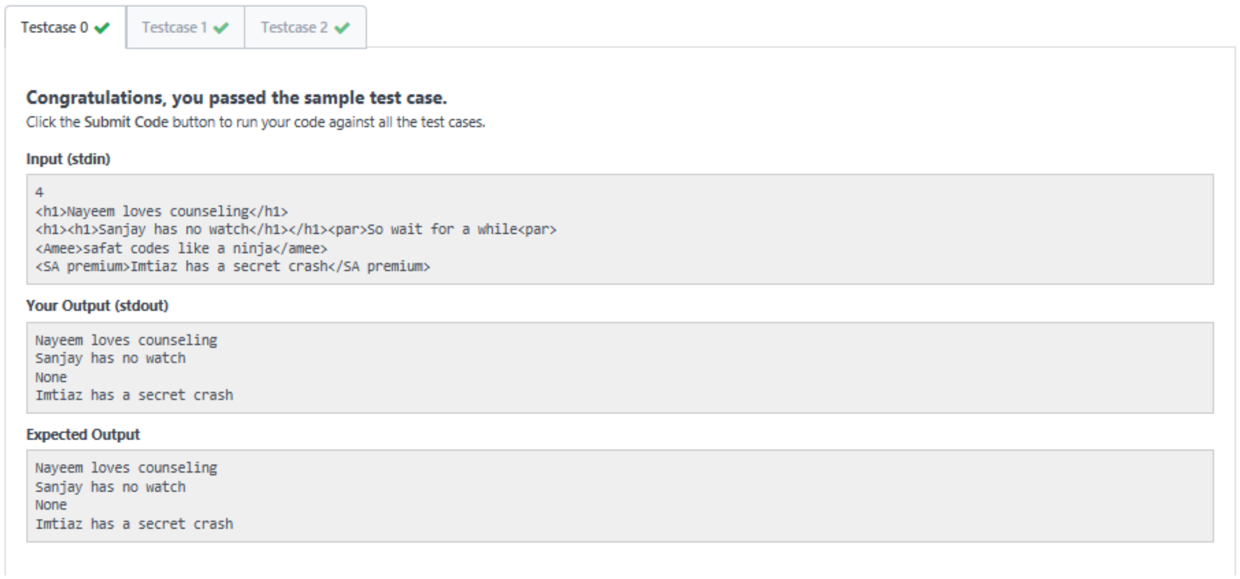
3. Tags can consist of any printable characters.

Program:

```
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

public class Solution{
  public static void main(String[] args){

    Scanner in = new Scanner(System.in);
    int testCases = Integer.parseInt(in.nextLine());
    while(testCases>0){
      String line = in.nextLine();



       int count=0;
      Pattern r = Pattern.compile("<(.+?)>([^<>]+)</\\1>");
      Matcher m = r.matcher(line);
      while(m.find()) {
         if (m.group(2).length() !=0) {
            System.out.println(m.group(2));
         count++;
         }
      }
      if (count == 0) System.out.println("None");
```

```
        testCases--;
      }
    }
  }
```

OUTPUT:



Testcase 0 ✓ | Testcase 1 ✓ | Testcase 2 ✓

Congratulations, you passed the sample test case.
Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

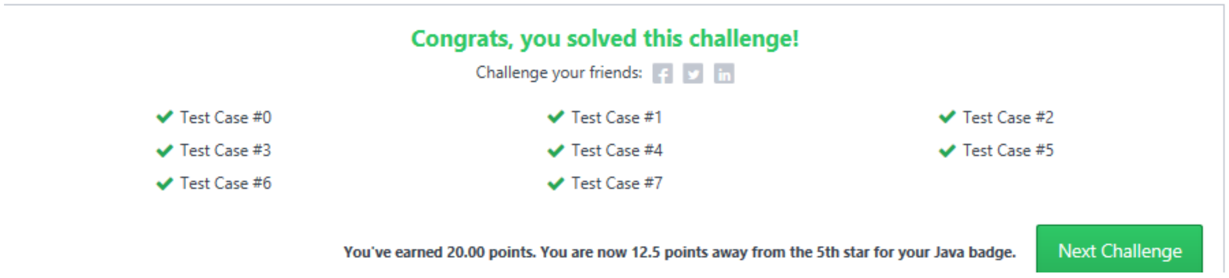
```
4
<h1>Nayeem loves counseling</h1>
<h1><h1>Sanjay has no watch</h1></h1><par>So wait for a while<par>
<Amee>safat codes like a ninja</amee>
<SA premium>Imtiaz has a secret crash</SA premium>
```

Your Output (stdout)

```
Nayeem loves counseling
Sanjay has no watch
None
Imtiaz has a secret crash
```

Expected Output

```
Nayeem loves counseling
Sanjay has no watch
None
Imtiaz has a secret crash
```



Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2
✓ Test Case #3 ✓ Test Case #4 ✓ Test Case #5
✓ Test Case #6 ✓ Test Case #7

You've earned 20.00 points. You are now 12.5 points away from the 5th star for your Java badge.

Q17

You are given a class *Solution* and its *main* method in the editor.
Your task is to create the class *Add* and the required methods so that the code prints the *sum of the numbers* passed to the function *add*.

Program:

import java.io.*;

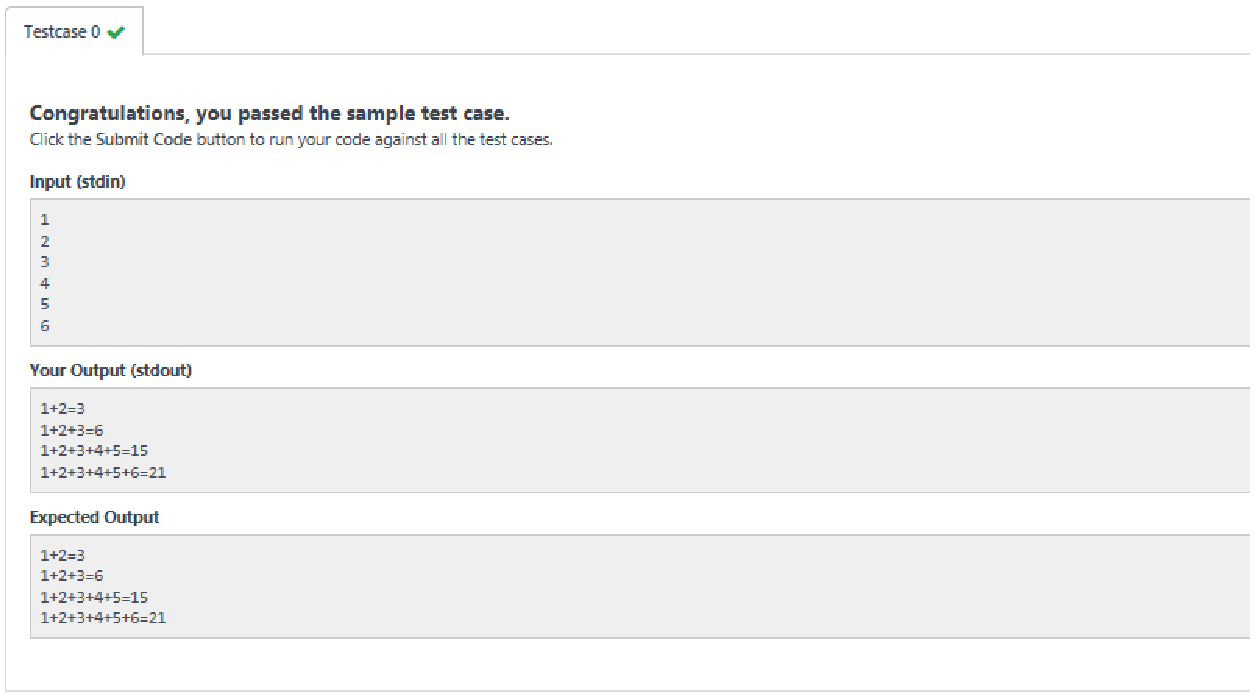
import java.lang.reflect.*;

import java.util.*;

```
import java.text.*;
import java.math.*;
import java.util.regex.*;
class Add {
        public void add(int... intArgs) {
                int sum = 0;
                String separator = "";
                for (int i : intArgs) {
                        sum += i;
                        System.out.print(separator + i);
                        separator = "+";
                }
                System.out.println("=" + sum);
        }
}

public class Solution {

   public static void main(String[] args) {
     try{
                        BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
                        int n1=Integer.parseInt(br.readLine());
                        int n2=Integer.parseInt(br.readLine());
                        int n3=Integer.parseInt(br.readLine());
                        int n4=Integer.parseInt(br.readLine());
                        int n5=Integer.parseInt(br.readLine());
                        int n6=Integer.parseInt(br.readLine());
                        Add ob=new Add();
                        ob.add(n1,n2);
                        ob.add(n1,n2,n3);
                        ob.add(n1,n2,n3,n4,n5);
                        ob.add(n1,n2,n3,n4,n5,n6);
                        Method[] methods=Add.class.getDeclaredMethods();
                        Set<String> set=new HashSet<>();
                        boolean overload=false;
                        for(int i=0;i<methods.length;i++)
                        {
                            if(set.contains(methods[i].getName()))
                            {
                                overload=true;
                                break;
                            }
                            set.add(methods[i].getName());

                        }
                        if(overload)
                        {
                            throw new Exception("Overloading not allowed");
                        }
                        }
                        catch(Exception e)
                        {
                            e.printStackTrace();
                        }
                }
```

}

OUTPUT:


Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the Submit Code button to run your code against all the test cases.

Input (stdin)

```
1
2
3
4
5
6
```

Your Output (stdout)

```
1+2=3
1+2+3=6
1+2+3+4+5=15
1+2+3+4+5+6=21
```

Expected Output

```
1+2=3
1+2+3=6
1+2+3+4+5=15
1+2+3+4+5+6=21
```

Q18

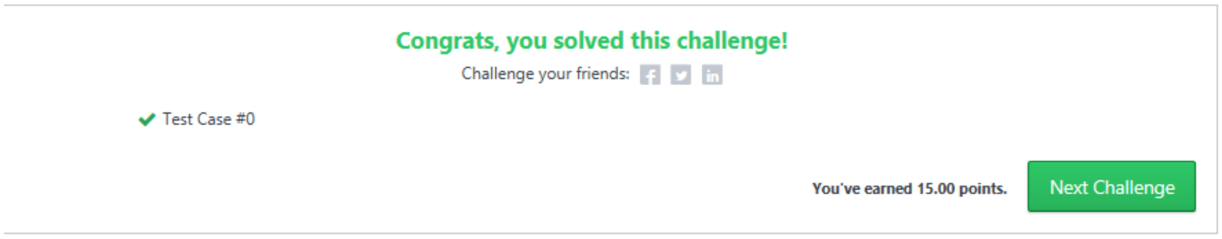
JAVA reflection is a very powerful tool to inspect the attributes of a class in runtime. For example, we can retrieve the list of public fields of a class using *getDeclaredMethods()*.

In this problem, you will be given a class *Solution* in the editor. You have to fill in the incompleted lines so that it prints all the methods of another class called *Student* in alphabetical order. We will append your code with the *Student* class before running it. The *Student* class looks like this:

Program:

```
public class Solution {



    public static void main(String[] args){
        Class student = Student.class;
        Method[] methods = student.getDeclaredMethods();
        ArrayList<String> methodList = new ArrayList<>();
        for(Method m : methods){
          methodList.add(m.getName());
        }
        Collections.sort(methodList);
        for(String name: methodList){
          System.out.println(name);
        }
      }


      }
```

OUTPUT:


Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0

You've earned 15.00 points.

Q19

According to Wikipedia, a factory is simply an object that returns another object from some other method call, which is assumed to be "new".

In this problem, you are given an interface *Food*. There are two classes *Pizza* and *Cake* which implement the *Food* interface, and they both contain a method *getType*().

The main function in the *Main* class creates an instance of the *FoodFactory* class. The *FoodFactory* class contains a method *getFood(String)* that returns a new instance of *Pizza* or *Cake* according to its parameter.

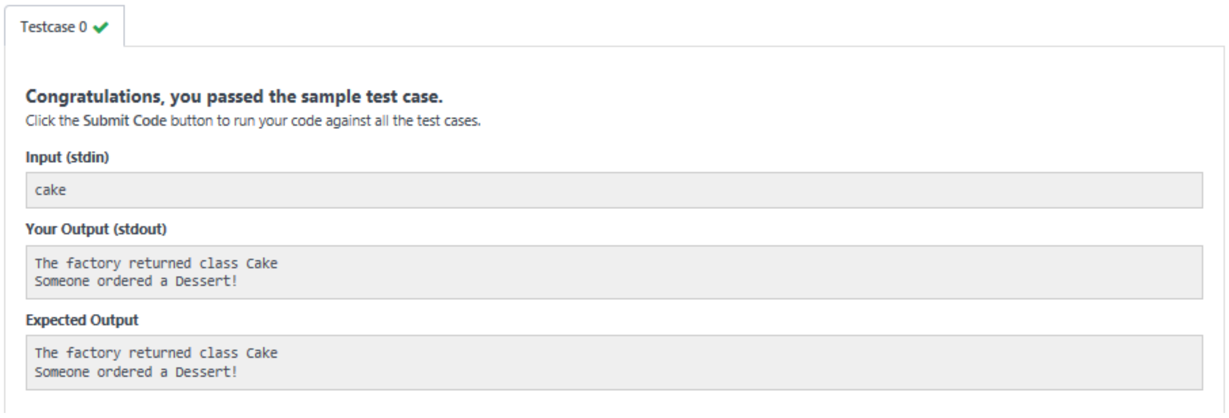
You are given the partially completed code in the editor. Please complete the *FoodFactory* class.

Program:

```
import java.util.*;
import java.security.*;
interface Food {
             public String getType();
            }
            class Pizza implements Food {
             public String getType() {
             return "Someone ordered a Fast Food!";
             }
            }

            class Cake implements Food {

             public String getType() {
             return "Someone ordered a Dessert!";
             }
            }
            class FoodFactory {
                        public Food getFood(String order) {



 try{
       return (Food)FoodFactory.class.getClassLoader().loadClass(camelCase(order)).newInstance();
    } catch ( Exception e ) {
       return null;
    }
}
private String camelCase(String name){
    return name.substring(0, 1).toUpperCase() + name.substring(1).toLowerCase();}//End of getFood
method


          }//End of factory class


          public class Solution {


           public static void main(String args[]){
                          Do_Not_Terminate.forbidExit();


                    try{


                          Scanner sc=new Scanner(System.in);
                          //creating the factory
                          FoodFactory foodFactory = new FoodFactory();


                          //factory instantiates an object
                          Food food = foodFactory.getFood(sc.nextLine());



                          System.out.println("The factory returned "+food.getClass());
                          System.out.println(food.getType());
                    }
                    catch (Do_Not_Terminate.ExitTrappedException e) {
                          System.out.println("Unsuccessful Termination!!");
                    }
           }


          }
          class Do_Not_Terminate {
```

```
    public static class ExitTrappedException extends SecurityException {

            private static final long serialVersionUID = 1L;
    }

    public static void forbidExit() {
        final SecurityManager securityManager = new SecurityManager() {
            @Override
            public void checkPermission(Permission permission) {
                if (permission.getName().contains("exitVM")) {
                    throw new ExitTrappedException();
                }
            }
        };
        System.setSecurityManager(securityManager);
    }
}
```

OUTPUT:


Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

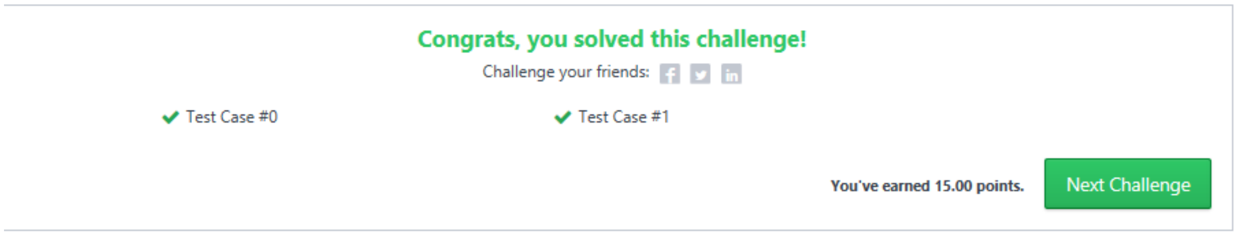
```
cake
```

Your Output (stdout)

```
The factory returned class Cake
Someone ordered a Dessert!
```

Expected Output

```
The factory returned class Cake
Someone ordered a Dessert!
```

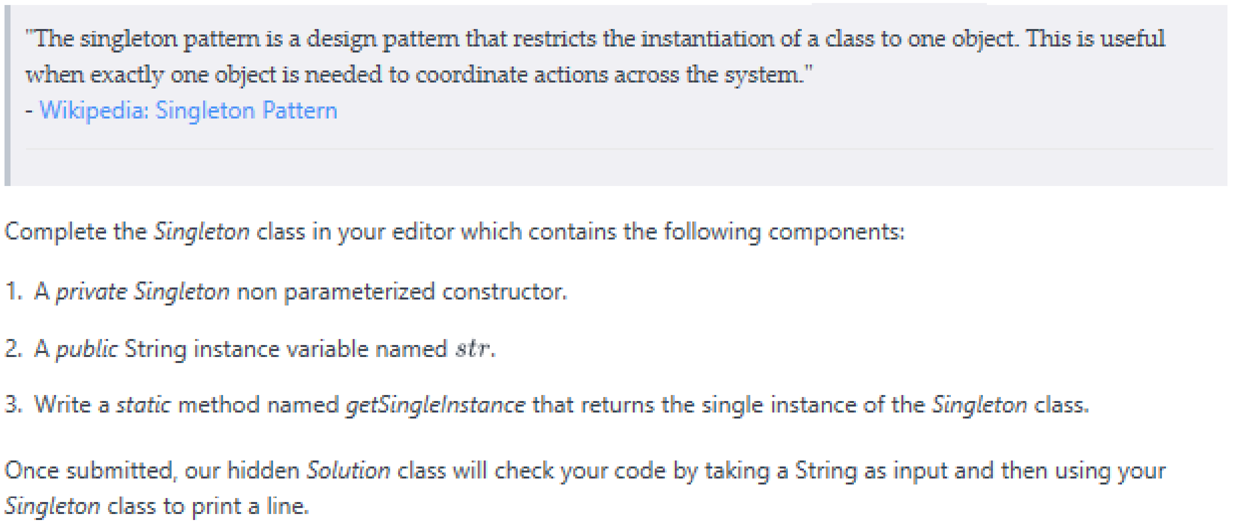

Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1

You've earned 15.00 points.

Q20


"The singleton pattern is a design pattern that restricts the instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system."
- Wikipedia: Singleton Pattern

Complete the *Singleton* class in your editor which contains the following components:

1. A *private Singleton* non parameterized constructor.
2. A *public* String instance variable named str.
3. Write a *static* method named *getSingleInstance* that returns the single instance of the *Singleton* class.

Once submitted, our hidden *Solution* class will check your code by taking a String as input and then using your *Singleton* class to print a line.

Program:

import java.io.*;

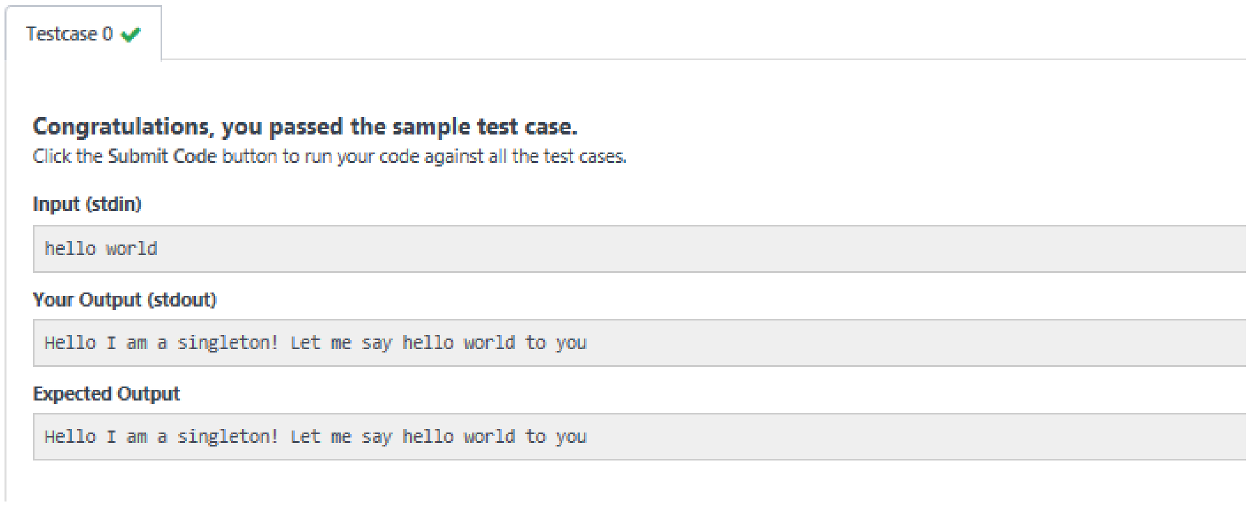
import java.util.*;

import java.text.*;

import java.math.*;

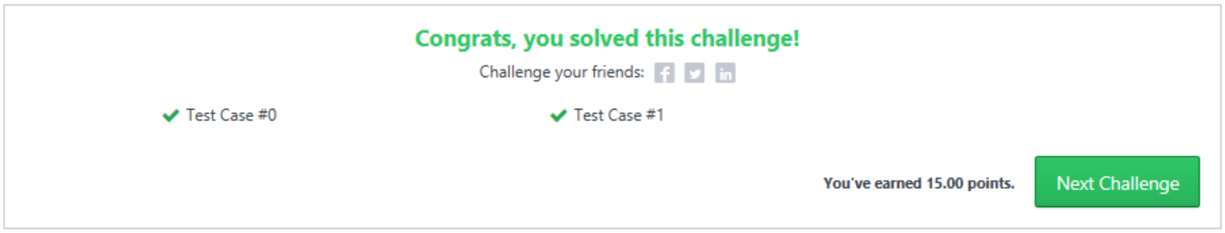
```
import java.util.regex.*;
import java.lang.reflect.*;


class Singleton {
    private volatile static Singleton instance;
    public static String str;
    private Singleton() {}

    static Singleton getSingleInstance() {
        if (instance == null) {
            synchronized (Singleton.class) {
                if (instance == null) {
                    instance = new Singleton();
                }
            }
        }
        return instance;
    }
}
```

OUTPUT:


Testcase 0 ✓

Congratulations, you passed the sample test case.
Click the Submit Code button to run your code against all the test cases.

Input (stdin)
```
hello world
```
Your Output (stdout)
```
Hello I am a singleton! Let me say hello world to you
```
Expected Output
```
Hello I am a singleton! Let me say hello world to you
```


Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1

You've earned 15.00 points.

Q21

Java's BigDecimal class can handle arbitrary-precision signed decimal numbers. Let's test your knowledge of them!

Given an array, s, of n real number strings, sort them in descending order — but wait, there's more! Each number must be printed in the exact same format as it was read from stdin, meaning that $.1$ is printed as $.1$, and 0.1 is printed as 0.1. If two numbers represent numerically equivalent values (e.g., $.1 \equiv 0.1$), then they must be listed in the same order as they were received as input).

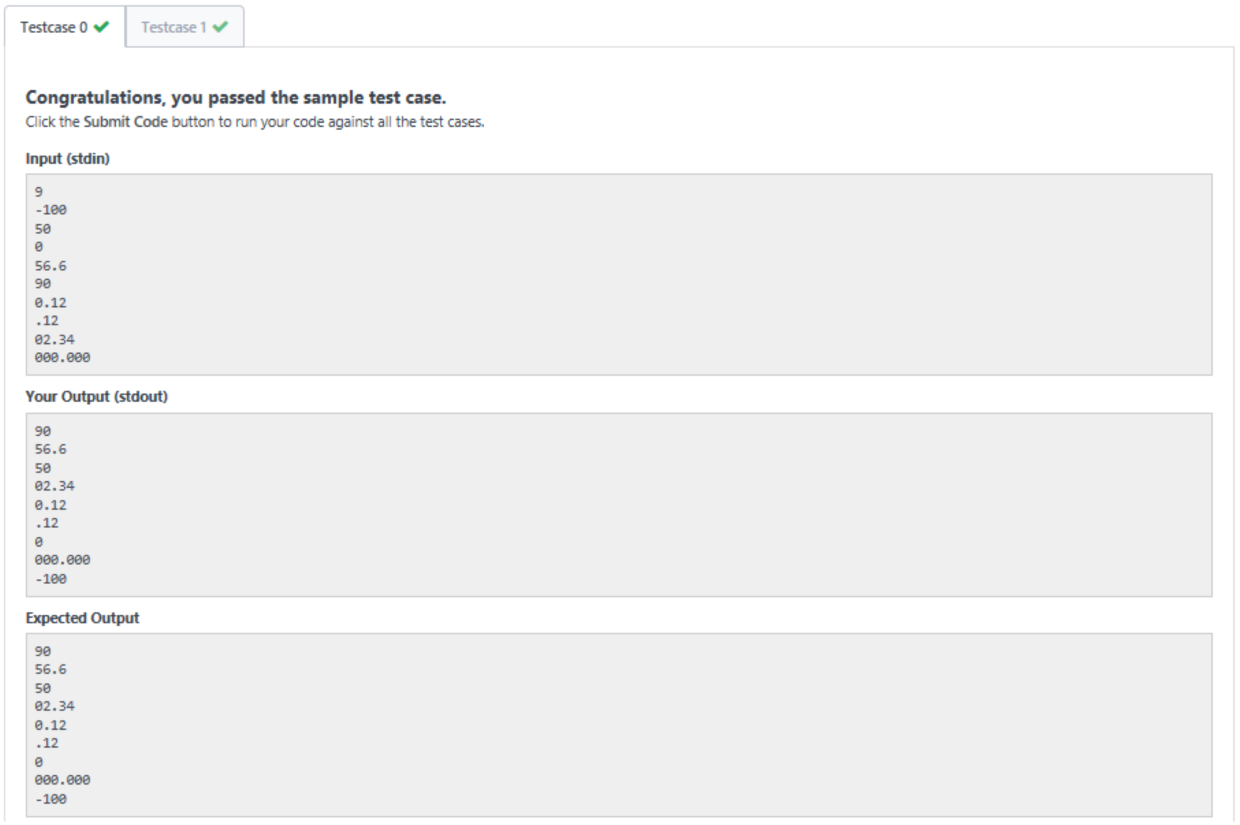
Complete the code in the unlocked section of the editor below. You must rearrange array s's elements according to the instructions above.

Program:

```
import java.math.BigDecimal;

import java.util.*;

class Solution{


  public static void main(String []args){

    //Input
```

```
        Scanner sc= new Scanner(System.in);
        int n=sc.nextInt();
        String []s=new String[n+2];
        for(int i=0;i<n;i++){
            s[i]=sc.next();
        }
             sc.close();
         Arrays.sort(s, 0, n, new Comparator<String>(){
        public int compare(String s1, String s2) {
            BigDecimal b1 = new BigDecimal(s1);
            BigDecimal b2 = new BigDecimal(s2);
            return -1 * b1.compareTo(b2);
        }
        });          //Output
        for(int i=0;i<n;i++)
        {
            System.out.println(s[i]);
        }
    }

}
```

OUTPUT:


Testcase 0 ✓ Testcase 1 ✓

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

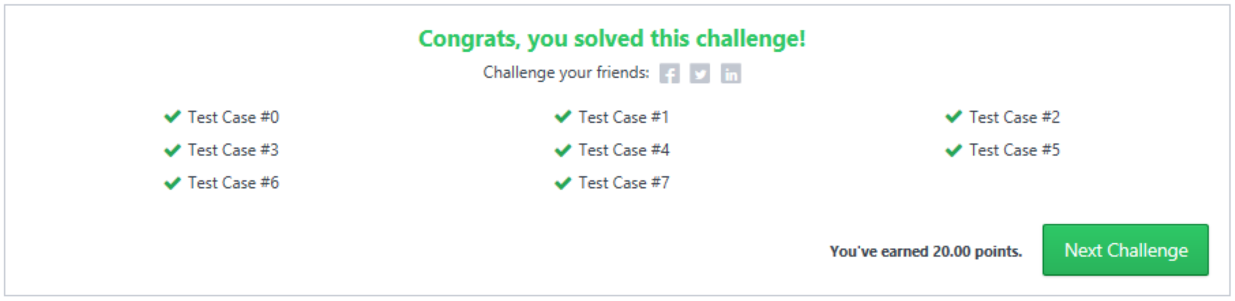
```
9
-100
50
0
56.6
90
0.12
.12
02.34
000.000
```

Your Output (stdout)

```
90
56.6
50
02.34
0.12
.12
0
000.000
-100
```

Expected Output

```
90
56.6
50
02.34
0.12
.12
0
000.000
-100
```

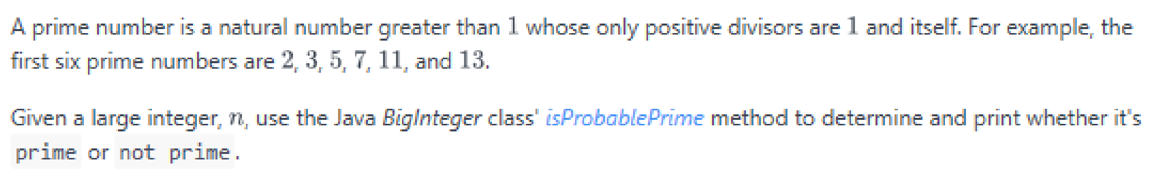

Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2
✓ Test Case #3 ✓ Test Case #4 ✓ Test Case #5
✓ Test Case #6 ✓ Test Case #7

You've earned 20.00 points. Next Challenge

Q22


A prime number is a natural number greater than 1 whose only positive divisors are 1 and itself. For example, the first six prime numbers are 2, 3, 5, 7, 11, and 13.

Given a large integer, n, use the Java *BigInteger* class' *isProbablePrime* method to determine and print whether it's `prime` or `not prime`.

Program:

import java.io.*;

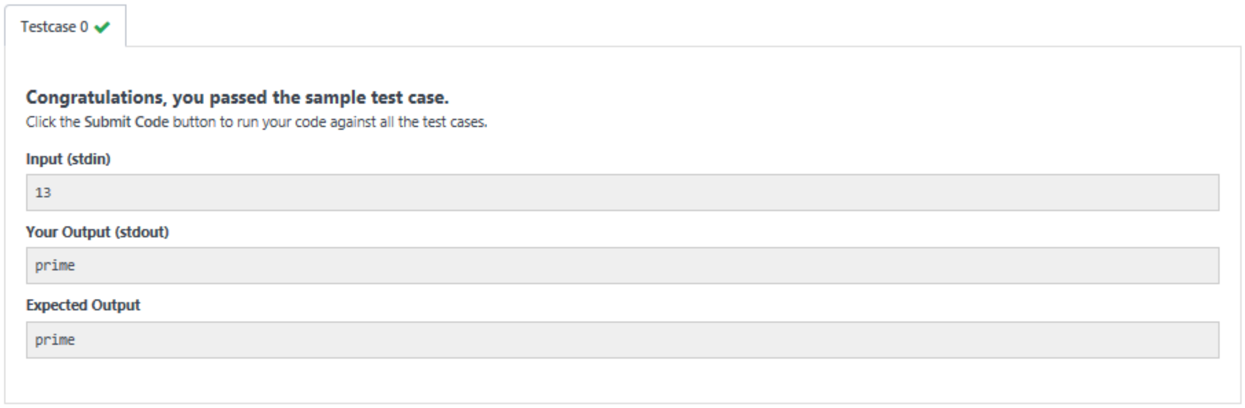
import java.util.*;

import java.text.*;

import java.math.*;

```
import java.util.regex.*;

public class Solution

{

    public static void main(String[] args)

    {

        try (Scanner scanner = new Scanner(System.in);)

        {

            System.out.println(scanner.nextBigInteger().isProbablePrime(100) ? "prime" : "not
prime");

        }

    }

}
```

OUTPUT:



Testcase 0 ✓

Congratulations, you passed the sample test case.
Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

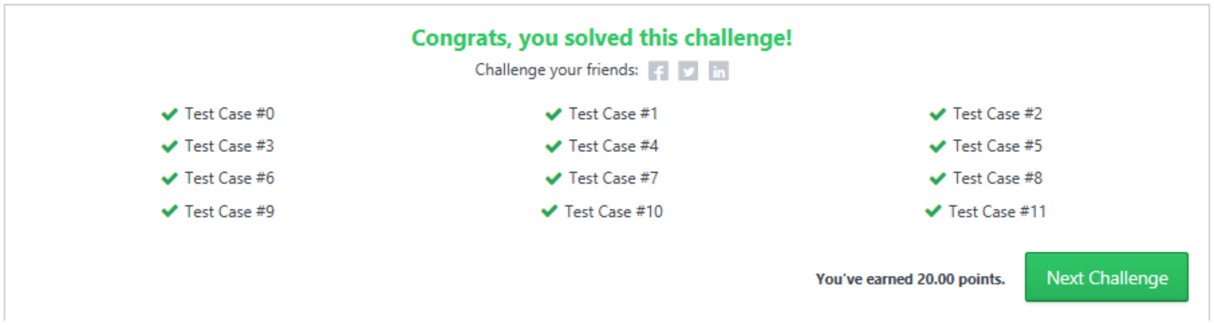
```
13
```

Your Output (stdout)

```
prime
```

Expected Output

```
prime
```



Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0	✓ Test Case #1	✓ Test Case #2
✓ Test Case #3	✓ Test Case #4	✓ Test Case #5
✓ Test Case #6	✓ Test Case #7	✓ Test Case #8
✓ Test Case #9	✓ Test Case #10	✓ Test Case #11

You've earned 20.00 points.

Q23

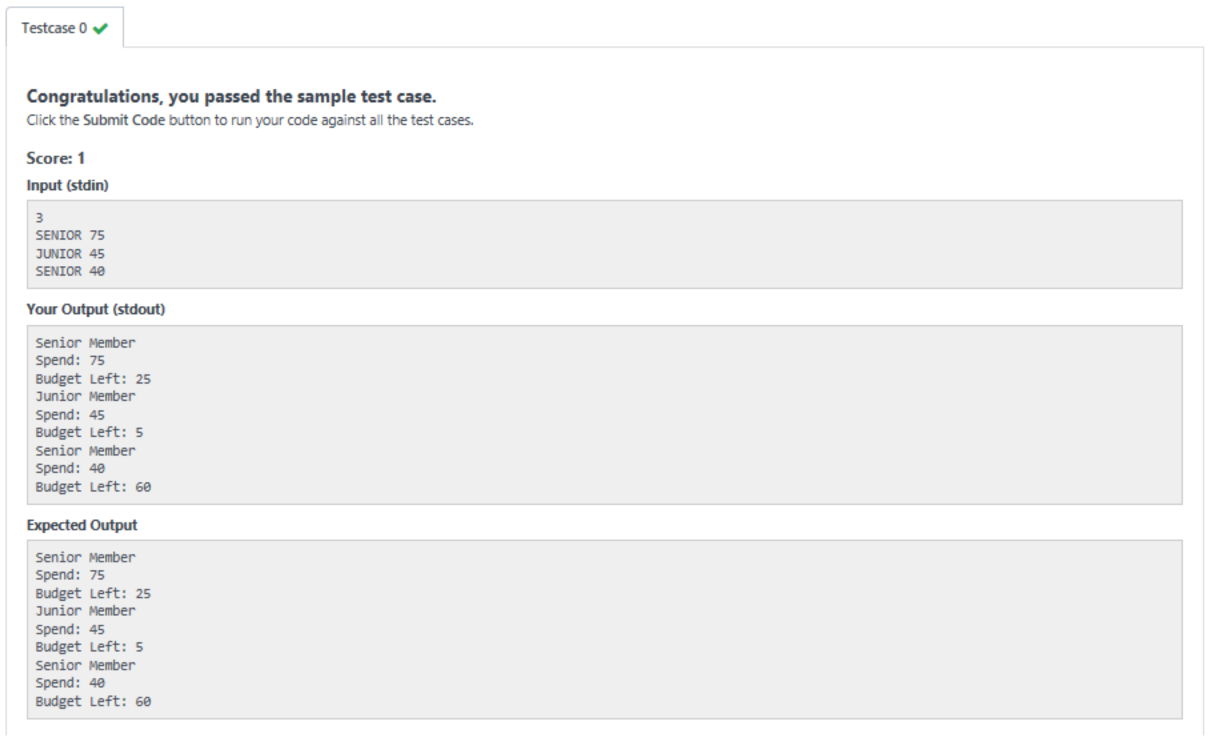
Java annotation can be used to define the metadata of a Java class or class element. We can use Java annotation at the compile time to instruct the compiler about the build process. Annotation is also used at runtime to get insight into the properties of class elements.

Program:

```
import java.lang.annotation.*;
import java.lang.reflect.*;
import java.util.*;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@interface FamilyBudget {
  String userRole() default "GUEST";
  int budgetLimit() default 0;


}

class FamilyMember {
  @FamilyBudget(userRole = "SENIOR", budgetLimit = 100)
  public void seniorMember(int budget, int moneySpend) {
    System.out.println("Senior Member");
    System.out.println("Spend: " + moneySpend);
    System.out.println("Budget Left: " + (budget - moneySpend));
  }

  @FamilyBudget(userRole = "JUNIOR", budgetLimit = 50)
  public void juniorUser(int budget, int moneySpend) {
    System.out.println("Junior Member");
    System.out.println("Spend: " + moneySpend);
    System.out.println("Budget Left: " + (budget - moneySpend));
  }
```

```
}

public class Solution {
  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    int testCases = Integer.parseInt(in.nextLine());
    while (testCases > 0) {
      String role = in.next();
      int spend = in.nextInt();
      try {
        Class annotatedClass = FamilyMember.class;
        Method[] methods = annotatedClass.getMethods();
        for (Method method : methods) {
          if (method.isAnnotationPresent(FamilyBudget.class)) {
            FamilyBudget family = method
                .getAnnotation(FamilyBudget.class);
            String userRole = family.userRole();
            int budgetLimit = family.budgetLimit();
            if (userRole.equals(role)) {
              if(budgetLimit >= spend){
                method.invoke(FamilyMember.class.newInstance(),
                    budgetLimit, spend);
              }else{
                System.out.println("Budget Limit Over");
              }
            }
          }
        }
      } catch (Exception e) {
        e.printStackTrace();
      }
      testCases--;
    }
  }
}
```

Output:



Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Score: 1

Input (stdin)

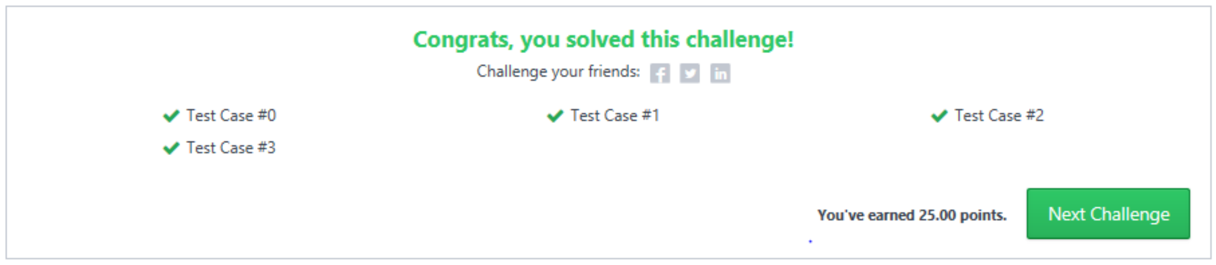
```
3
SENIOR 75
JUNIOR 45
SENIOR 40
```

Your Output (stdout)

```
Senior Member
Spend: 75
Budget Left: 25
Junior Member
Spend: 45
Budget Left: 5
Senior Member
Spend: 40
Budget Left: 60
```

Expected Output

```
Senior Member
Spend: 75
Budget Left: 25
Junior Member
Spend: 45
Budget Left: 5
Senior Member
Spend: 40
Budget Left: 60
```



Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2
✓ Test Case #3

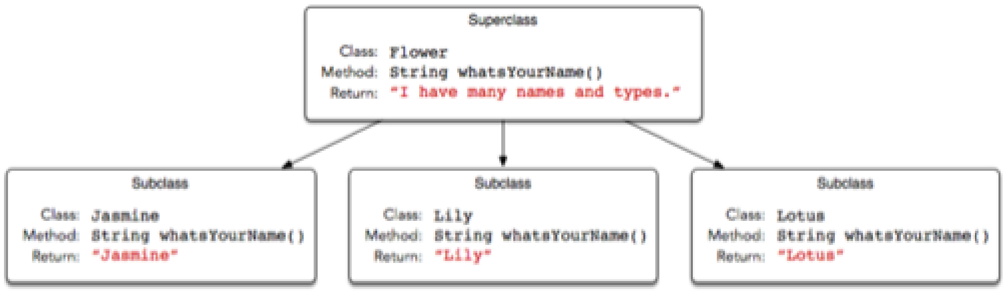
You've earned 25.00 points.

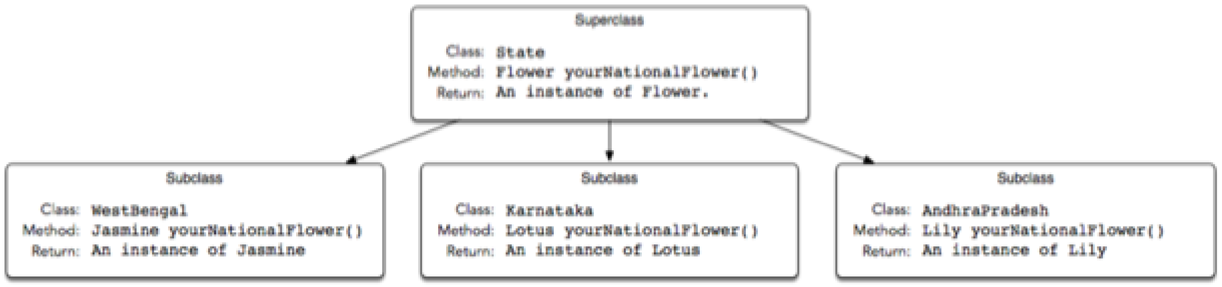
Q24

Java allows for Covariant Return Types, which means you can vary your return type as long you are returning a subclass of your specified return type.

Method Overriding allows a subclass to *override* the behavior of an existing superclass method and specify a return type that is some subclass of the original return type. It is best practice to use the `@Override` annotation when overriding a superclass method.

Implement the classes and methods detailed in the diagram below:


Superclass
Class: Flower
Method: String whatsYourName()
Return: "I have many names and types."
Subclass
Class: Jasmine
Method: String whatsYourName()
Return: "Jasmine"
Subclass
Class: Lily
Method: String whatsYourName()
Return: "Lily"
Subclass
Class: Lotus
Method: String whatsYourName()
Return: "Lotus"


Superclass
Class: State
Method: Flower yourNationalFlower()
Return: An instance of Flower.
Subclass
Class: WestBengal
Method: Jasmine yourNationalFlower()
Return: An instance of Jasmine
Subclass
Class: Karnataka
Method: Lotus yourNationalFlower()
Return: An instance of Lotus
Subclass
Class: AndhraPradesh
Method: Lily yourNationalFlower()
Return: An instance of Lily

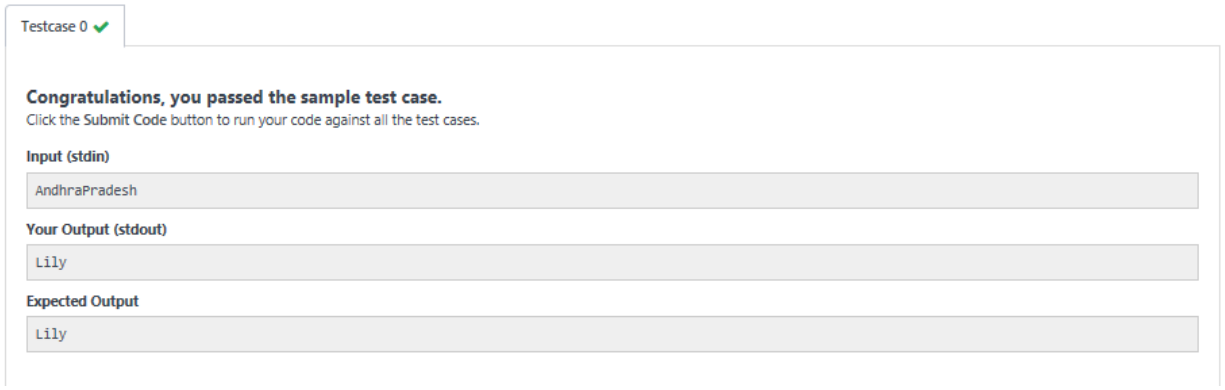
Your implementation is checked by a hidden *Solution* class, where the *main* method takes the name of a state and prints the national flower of that state using the classes and methods written by you.

Program:

```
class Flower {
   String whatsYourName() {
      return "I have many names and types.";
   }
}


class Jasmine extends Flower {
   @Override
   String whatsYourName() {
      return "Jasmine";
   }
```

```
}

class Lily extends Flower {
    @Override
    String whatsYourName() {
        return "Lily";
    }
}

class Lotus extends Flower {
    @Override
    String whatsYourName() {
        return "Lotus";
    }
}

class State {
    Flower yourNationalFlower() {
        return new Flower();
    }
}

class WestBengal extends State {
    @Override
    Jasmine yourNationalFlower() {
        return new Jasmine();
    }
}

class Karnataka extends State {
    @Override
    Lotus yourNationalFlower() {
        return new Lotus();
    }
}

class AndhraPradesh extends State {
    @Override
    Lily yourNationalFlower() {
        return new Lily();
    }
}
```

OUTPUT:



Testcase 0 ✓

Congratulations, you passed the sample test case.
Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

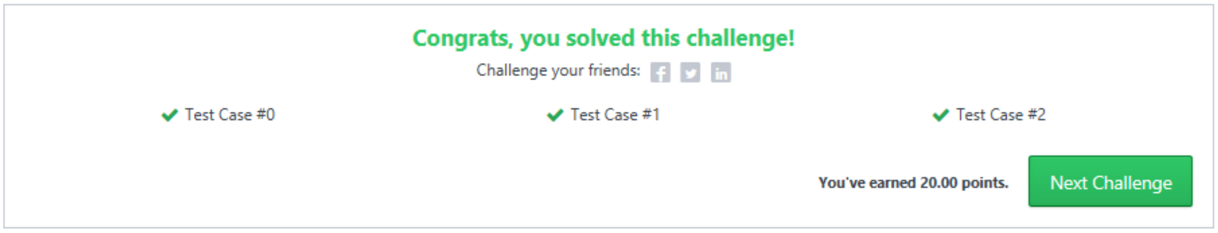
```
AndhraPradesh
```

Your Output (stdout)

```
Lily
```

Expected Output

```
Lily
```



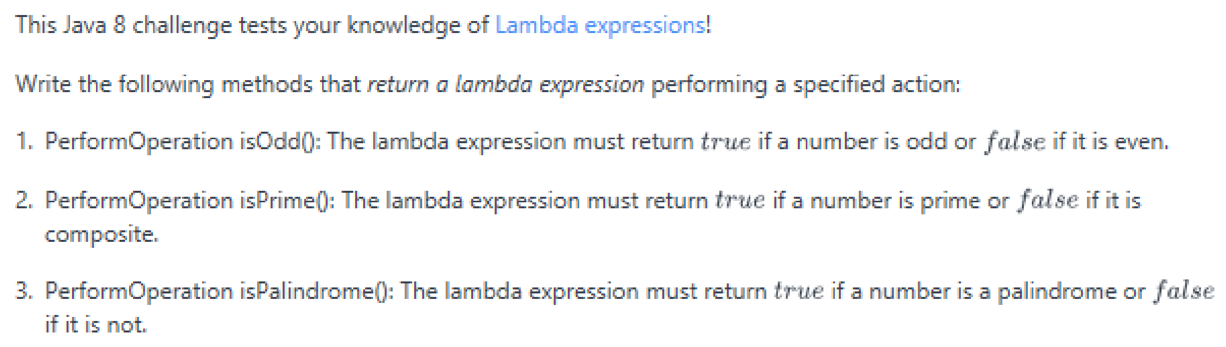
Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2

You've earned 20.00 points.

Q25


This Java 8 challenge tests your knowledge of Lambda expressions!

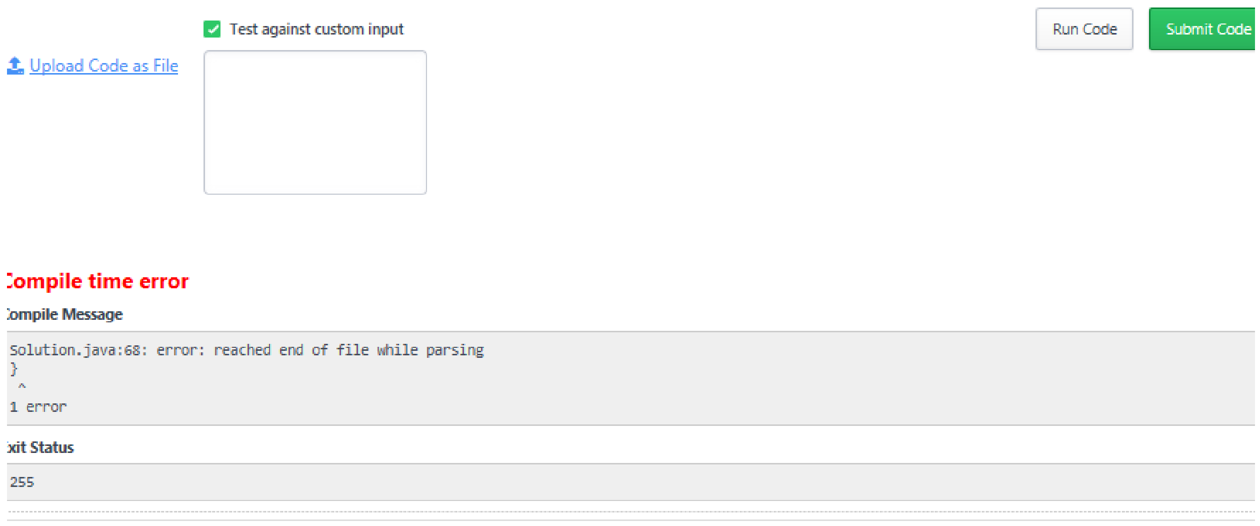
Write the following methods that *return a lambda expression* performing a specified action:

1. PerformOperation isOdd(): The lambda expression must return $true$ if a number is odd or $false$ if it is even.
2. PerformOperation isPrime(): The lambda expression must return $true$ if a number is prime or $false$ if it is composite.
3. PerformOperation isPalindrome(): The lambda expression must return $true$ if a number is a palindrome or $false$ if it is not.

Program:

```
import java.io.*;
import java.util.*;
interface PerformOperation {
 boolean check(int a);
}
class MyMath {
 public static boolean checker(PerformOperation p, int num) {
  return p.check(num);
 }PerformOperation is_odd(){
    return a->a%2==1?true:false;
}

PerformOperation is_prime(){
    return a->{
        for (int i=2; i<a/2; i++){
            if (a%i==0){
                return false;
            }
        }
        return true;
    };
```

```
}

PerformOperation is_palindrome(){
    return a->{
        String str = Integer.toString(a);
        int n = str.length();
        for (int i=0; i<n/2; i++){
            if (str.charAt(i)!=str.charAt(n-i-1)){
                return false;
            }
        }
        return true;
    };
}
public class Solution {

 public static void main(String[] args) throws IOException {
  MyMath ob = new MyMath();
  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
  int T = Integer.parseInt(br.readLine());
  PerformOperation op;
  boolean ret = false;
  String ans = null;
  while (T--> 0) {
   String s = br.readLine().trim();
   StringTokenizer st = new StringTokenizer(s);
   int ch = Integer.parseInt(st.nextToken());
   int num = Integer.parseInt(st.nextToken());
   if (ch == 1) {
    op = ob.isOdd();
    ret = ob.checker(op, num);
    ans = (ret) ? "ODD" : "EVEN";
   } else if (ch == 2) {
    op = ob.isPrime();
    ret = ob.checker(op, num);
    ans = (ret) ? "PRIME" : "COMPOSITE";
   } else if (ch == 3) {
    op = ob.isPalindrome();
    ret = ob.checker(op, num);
    ans = (ret) ? "PALINDROME" : "NOT PALINDROME";

   }
   System.out.println(ans);
  }
 }
}
```



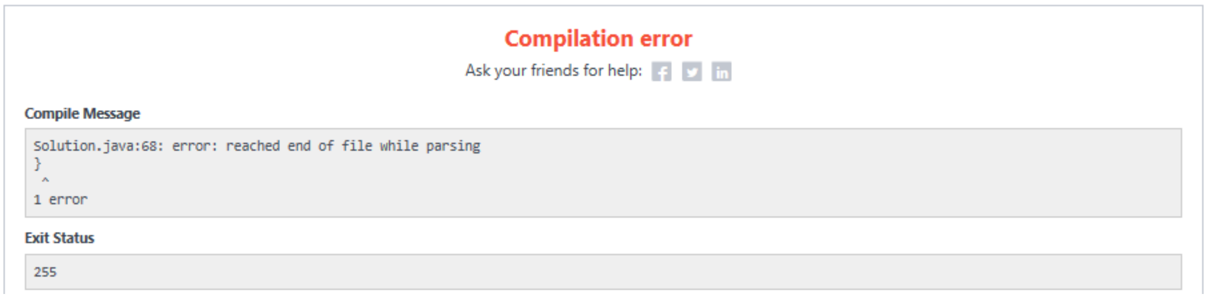
ompile time error

ompile Message

```
Solution.java:68: error: reached end of file while parsing
}
 ^
1 error
```

xit Status

```
255
```


Compilation error

Ask your friends for help:

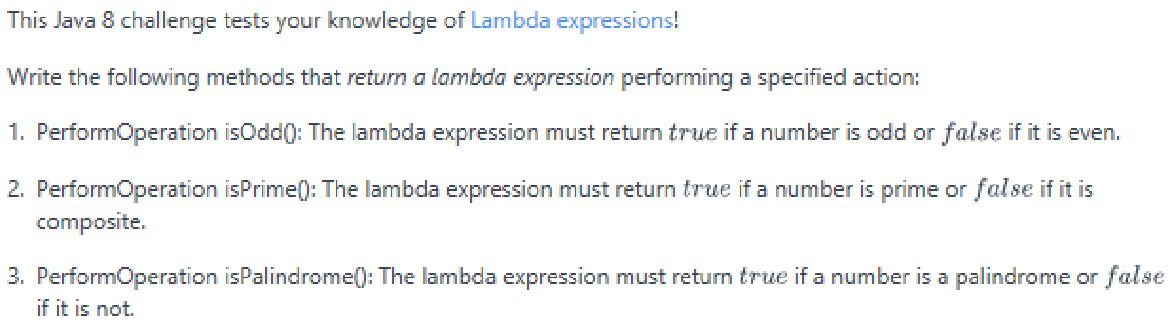
Compile Message

```
Solution.java:68: error: reached end of file while parsing
}
 ^
1 error
```

Exit Status

```
255
```

Q26


This Java 8 challenge tests your knowledge of Lambda expressions!

Write the following methods that *return a lambda expression* performing a specified action:

1. PerformOperation isOdd(): The lambda expression must return $true$ if a number is odd or $false$ if it is even.
2. PerformOperation isPrime(): The lambda expression must return $true$ if a number is prime or $false$ if it is composite.
3. PerformOperation isPalindrome(): The lambda expression must return $true$ if a number is a palindrome or $false$ if it is not.

Program:

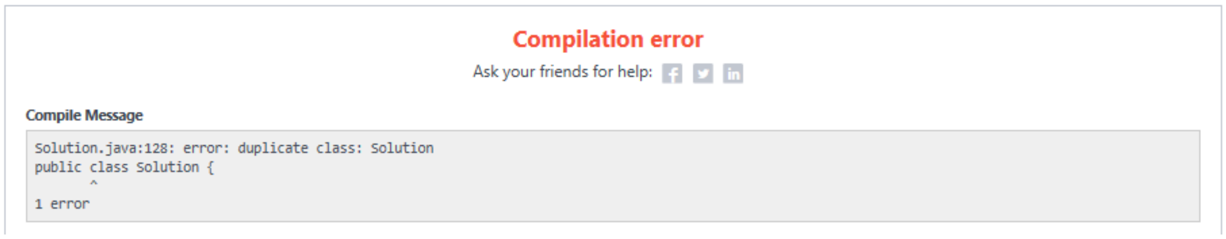
```
import java.io.*;

import java.util.*;

interface PerformOperation {

 boolean check(int a);

}

class MyMath {

 public static boolean checker(PerformOperation p, int num) {

  return p.check(num);

 }



    public static PerformOperation is_odd() {


        return n -> (n & 1) == 1;
```

```
}

public static PerformOperation is_prime() {
    // O(n^(1/2)) runtime
    return n -> {
        if (n < 2) {
            return false;
        }
        int sqrt = (int) Math.sqrt(n);
        for (int i = 2; i <= sqrt; i++) {
            if (n % i == 0) {
                return false;
            }
        }
        return true;
        };
    }

    public static PerformOperation is_palindrome() {
        return n -> {
            String original = Integer.toString(n);
            String reversed = new StringBuilder(Integer.toString(n)).reverse().toString();
            return original.equals(reversed);
        };
    }
}

public class Solution {
    public static void main(String[] args) throws IOException {
        MyMath ob = new MyMath();
```

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int T = Integer.parseInt(br.readLine());

PerformOperation op;

boolean ret = false;

String ans = null;

while (T--> 0) {

   String s = br.readLine().trim();

   StringTokenizer st = new StringTokenizer(s);

   int ch = Integer.parseInt(st.nextToken());

   int num = Integer.parseInt(st.nextToken());

   if (ch == 1) {

      op = ob.is_odd();

      ret = ob.checker(op, num);

      ans = (ret) ? "ODD" : "EVEN";

   } else if (ch == 2) {
```

```
                op = ob.is_prime();

                ret = ob.checker(op, num);

                ans = (ret) ? "PRIME" : "COMPOSITE";

            } else if (ch == 3) {

                op = ob.is_palindrome();

                ret = ob.checker(op, num);

                ans = (ret) ? "PALINDROME" : "NOT PALINDROME";

            }

            System.out.println(ans);

        }

    }
}
```

OUTPUT:



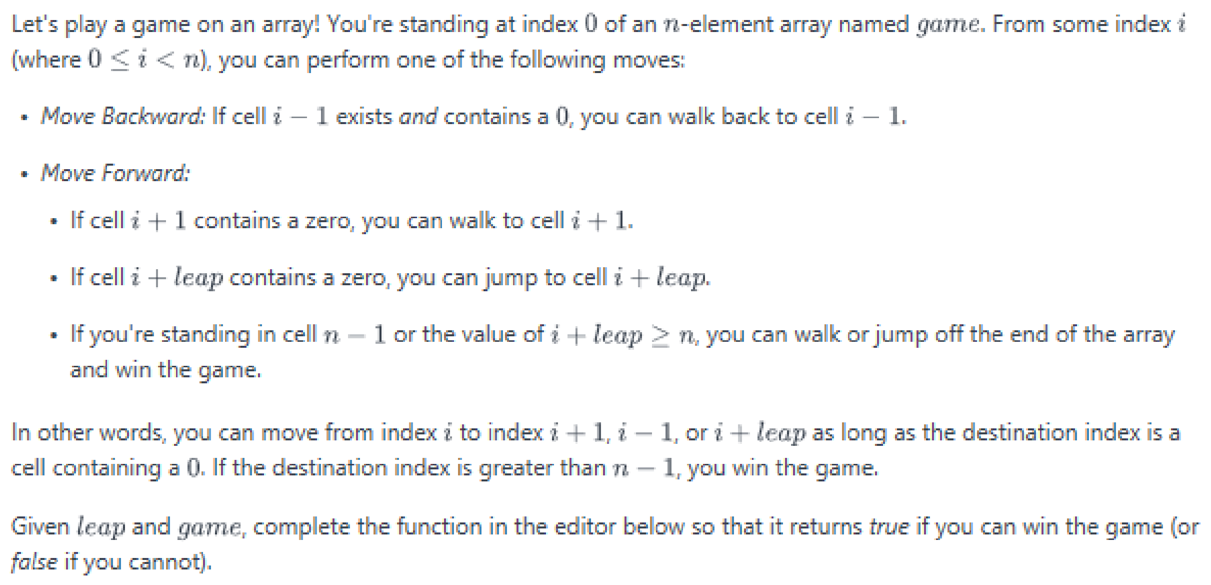
Compilation error

Ask your friends for help:

Compile Message

```
Solution.java:128: error: duplicate class: Solution
public class Solution {
       ^
1 error
```

Q27


Let's play a game on an array! You're standing at index 0 of an n-element array named $game$. From some index i (where $0 \leq i < n$), you can perform one of the following moves:

- *Move Backward:* If cell $i - 1$ exists *and* contains a 0, you can walk back to cell $i - 1$.
- *Move Forward:*
  - If cell $i + 1$ contains a zero, you can walk to cell $i + 1$.
  - If cell $i + leap$ contains a zero, you can jump to cell $i + leap$.
  - If you're standing in cell $n - 1$ or the value of $i + leap \geq n$, you can walk or jump off the end of the array and win the game.

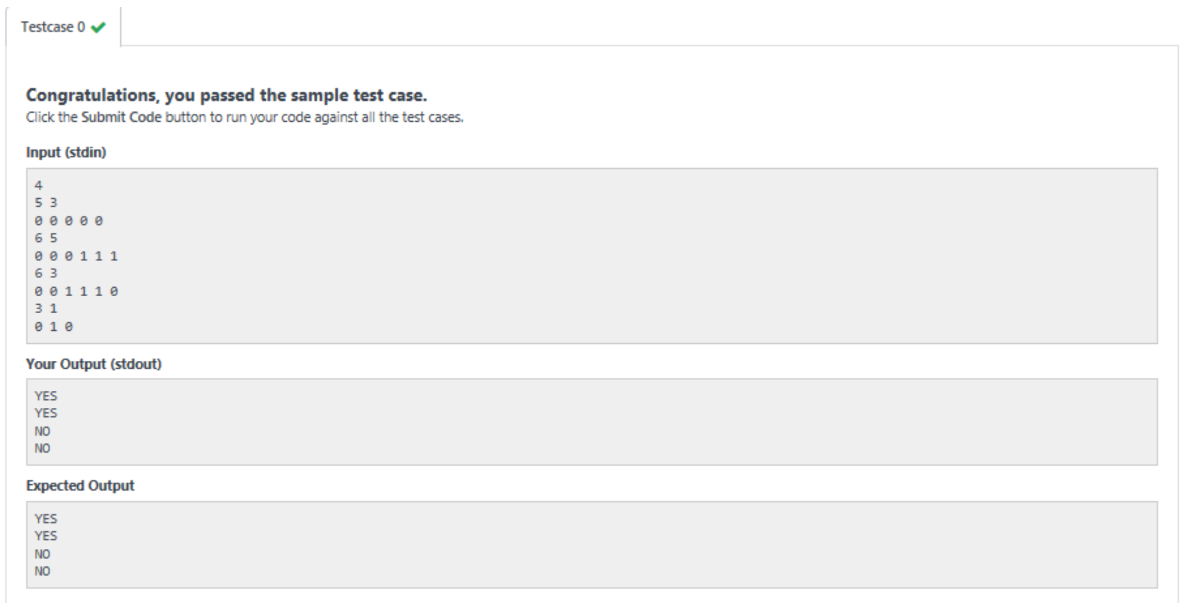
In other words, you can move from index i to index $i + 1$, $i - 1$, or $i + leap$ as long as the destination index is a cell containing a 0. If the destination index is greater than $n - 1$, you win the game.

Given $leap$ and $game$, complete the function in the editor below so that it returns *true* if you can win the game (or *false* if you cannot).

Program:

```
import java.util.Scanner;



public class Solution {

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        int T = scan.nextInt();

        while (T-- > 0) {

            int n = scan.nextInt();
            int m = scan.nextInt();

            int [] array = new int[n];

            for (int i = 0; i < n; i++) {

                array[i] = scan.nextInt();

            }

            System.out.println(isSolvable(array, m, 0) ? "YES" : "NO");

        }

        scan.close();

    }



    /* Basically a depth-first search (DFS).

       Marks each array value as 1 when visiting (Side-effect: alters array) */

    private static boolean isSolvable(int[] array, int m, int i) {

        /* Base Cases */

        if (i < 0 || array[i] == 1) {
            return false;

        } else if (i + 1 >= array.length || i + m >= array.length) {

            return true;

        }



        array[i] = 1; // marks as visited



        /* Recursive Cases (Tries +m first to try to finish game quickly) */

        return isSolvable(array, m, i + m) ||

              isSolvable(array, m, i + 1) ||

              isSolvable(array, m, i - 1);

    }

}
```

OUTPUT:


Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the Submit Code button to run your code against all the test cases.

Input (stdin)

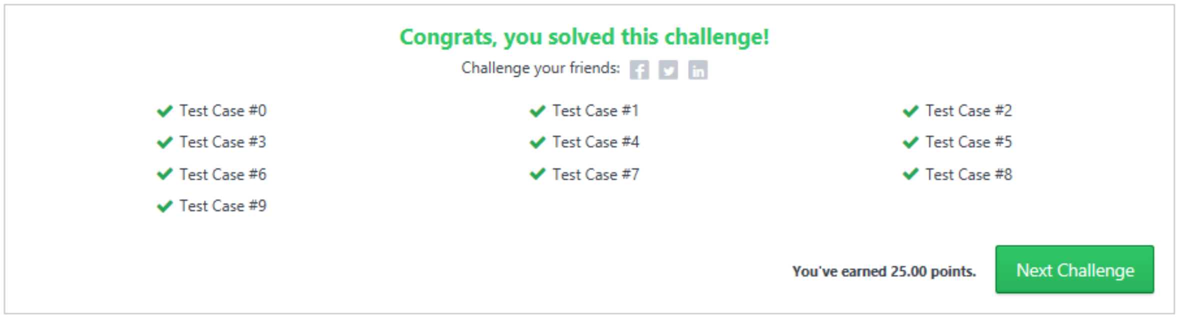
```
4
5 3
0 0 0 0 0
6 5
0 0 0 1 1 1
6 3
0 0 1 1 1 0
3 1
0 1 0
```

Your Output (stdout)

```
YES
YES
NO
NO
```

Expected Output

```
YES
YES
NO
NO
```

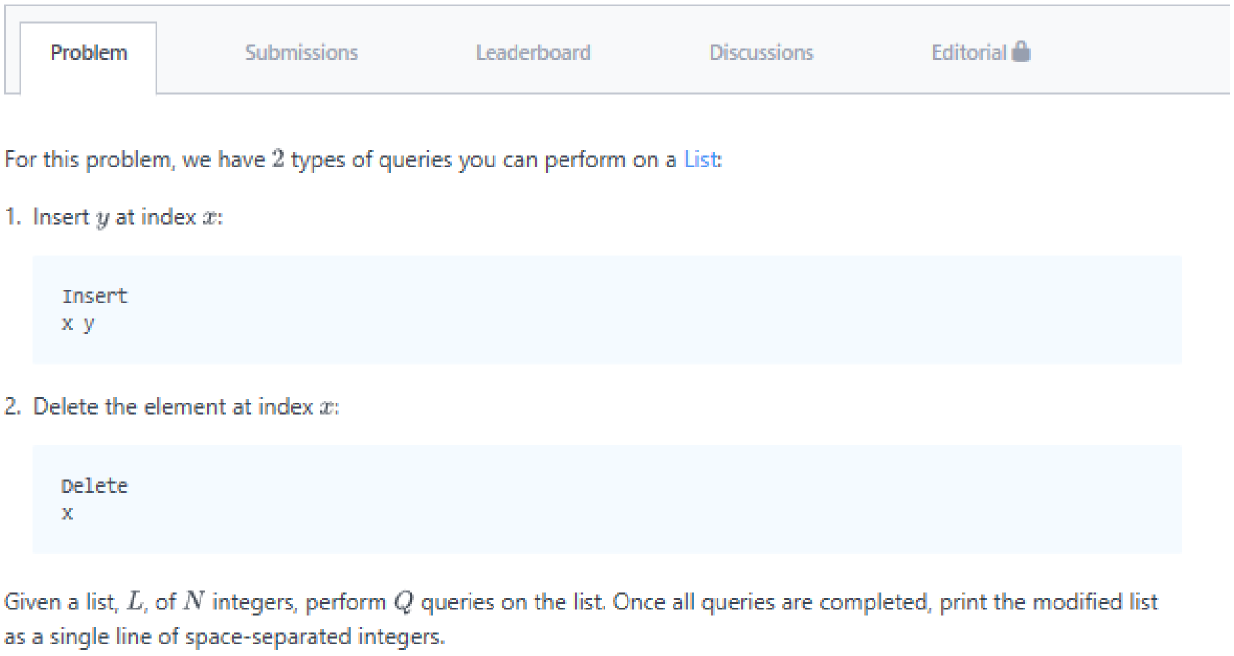

Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2
✓ Test Case #3 ✓ Test Case #4 ✓ Test Case #5
✓ Test Case #6 ✓ Test Case #7 ✓ Test Case #8
✓ Test Case #9

You've earned 25.00 points.

Q28



For this problem, we have 2 types of queries you can perform on a List:

1. Insert y at index x:

```
Insert
x y
```

2. Delete the element at index x:

```
Delete
x
```

Given a list, L, of N integers, perform Q queries on the list. Once all queries are completed, print the modified list as a single line of space-separated integers.

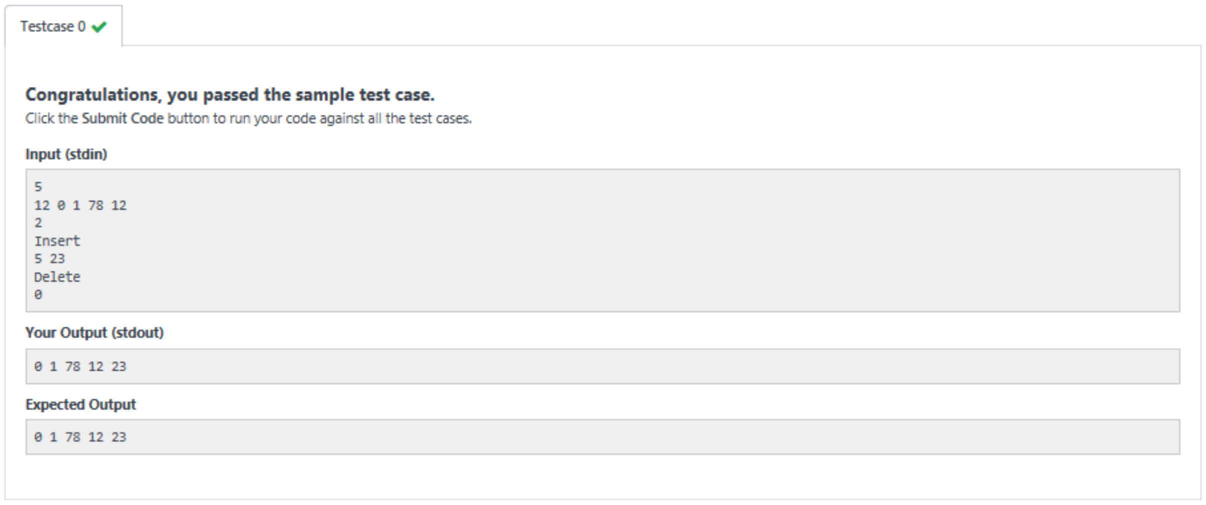
Program:

```
import java.util.Scanner;

import java.util.LinkedList;



public class Solution {

    public static void main(String[] args) {

        /* Create and fill Linked List of Integers */

        Scanner scan = new Scanner(System.in);

        int N = scan.nextInt();

        LinkedList<Integer> list = new LinkedList<>();

        for (int i = 0; i < N; i++) {

            int value = scan.nextInt();

            list.add(value);

        }



        /* Perform queries on Linked List */
```

```
int Q = scan.nextInt();

for (int i = 0; i < Q; i++) {

    String action = scan.next();

    if (action.equals("Insert")) {

        int index = scan.nextInt();

        int value = scan.nextInt();

        list.add(index, value);

    } else { // "Delete"

        int index = scan.nextInt();

        list.remove(index);

    }

}

scan.close();



/* Print our updated Linked List */
```

```
        for (Integer num : list) {

            System.out.print(num + " ");

        }

    }

}
```

OUTPUT:



Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

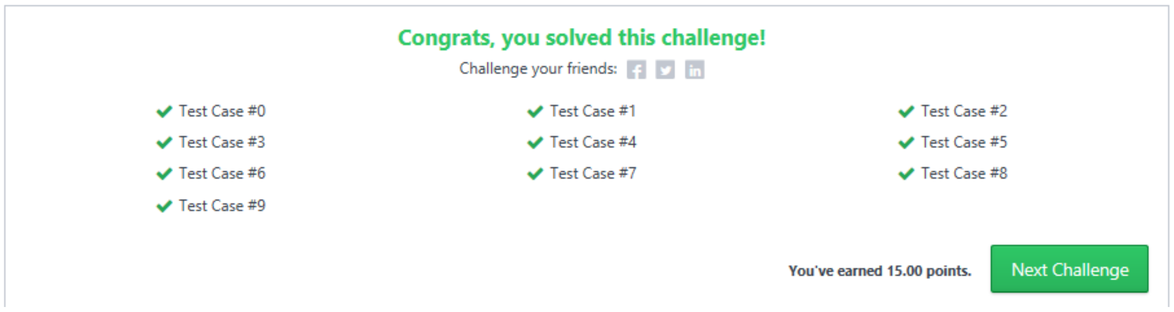
```
5
12 0 1 78 12
2
Insert
5 23
Delete
0
```

Your Output (stdout)

```
0 1 78 12 23
```

Expected Output

```
0 1 78 12 23
```



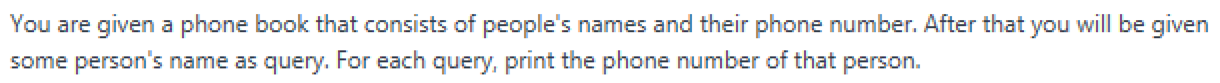
Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0	✓ Test Case #1	✓ Test Case #2
✓ Test Case #3	✓ Test Case #4	✓ Test Case #5
✓ Test Case #6	✓ Test Case #7	✓ Test Case #8
✓ Test Case #9		

You've earned 15.00 points.

Q29


You are given a phone book that consists of people's names and their phone number. After that you will be given some person's name as query. For each query, print the phone number of that person.

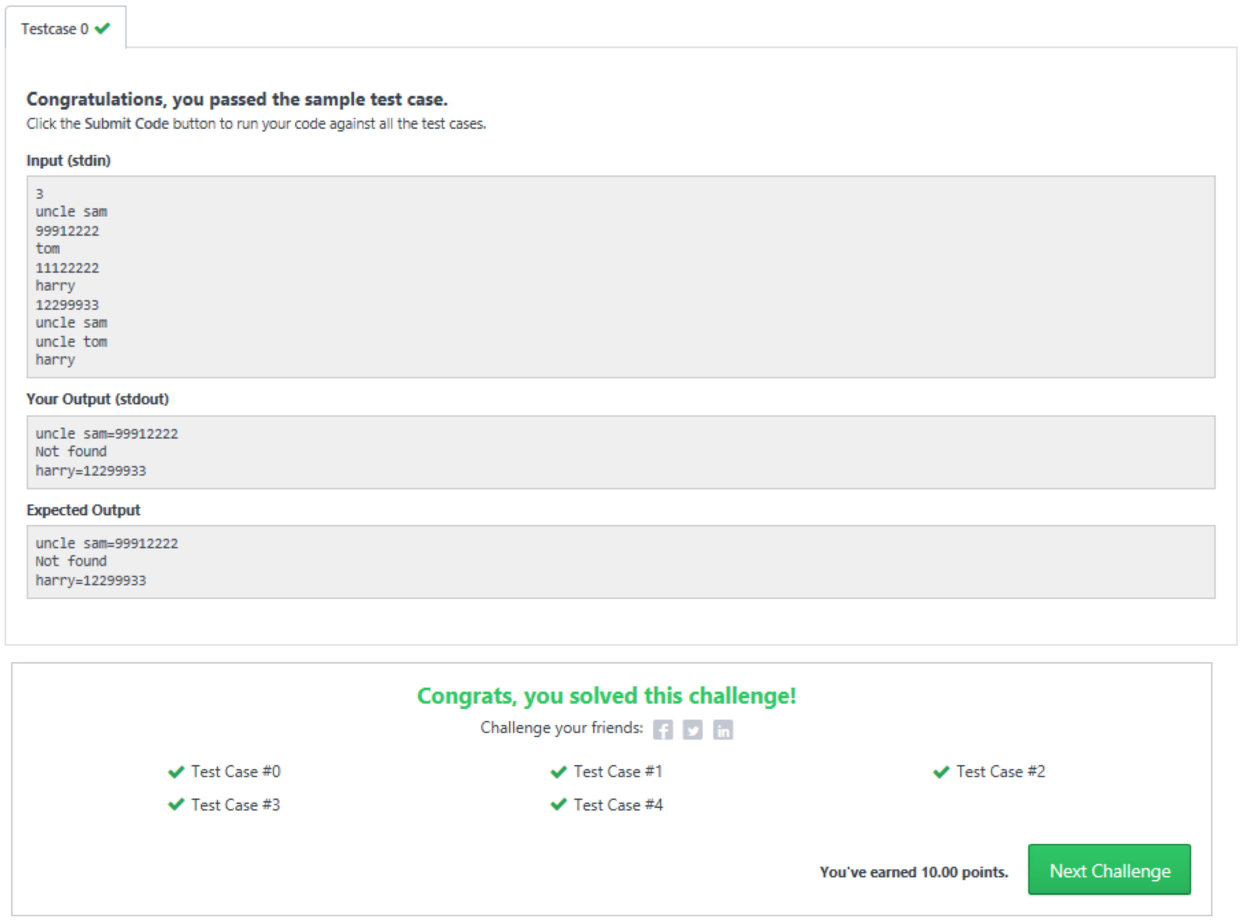
Program:

```
import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.IOException;

import java.util.HashMap;



/* BufferedReader is used instead of Scanner since it's faster and

   won't time out on HackerRank test cases when using Java8 */

class Solution {

    public static void main(String[] args) throws IOException {

        /* Save input as entries in a HashMap */

        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        int n = Integer.parseInt(br.readLine());

        HashMap<String, Integer> map = new HashMap<>();
```

```
while (n-- > 0) {

    String name = br.readLine();

    int phone   = Integer.parseInt(br.readLine());

    map.put(name, phone);

}



/* Read each query and check if its in our HashMap */

String s;

while((s = br.readLine()) != null) {

    if (map.containsKey(s)) {

        System.out.println(s + "=" + map.get(s));

    } else {

        System.out.println("Not found");

    }

}
```

```
        br.close();

    }

}
```

OUTPUT:


Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

```
3
uncle sam
99912222
tom
11122222
harry
12299933
uncle sam
uncle tom
harry
```

Your Output (stdout)

```
uncle sam=99912222
Not found
harry=12299933
```

Expected Output

```
uncle sam=99912222
Not found
harry=12299933
```

Congrats, you solved this challenge!

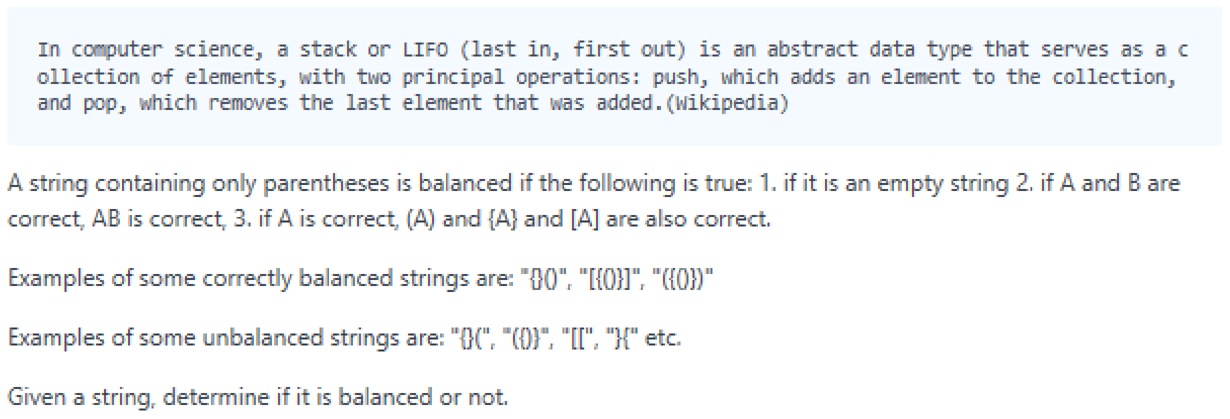
Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2

✓ Test Case #3 ✓ Test Case #4

You've earned 10.00 points.

Q30



```
In computer science, a stack or LIFO (last in, first out) is an abstract data type that serves as a c
ollection of elements, with two principal operations: push, which adds an element to the collection,
and pop, which removes the last element that was added.(Wikipedia)
```

A string containing only parentheses is balanced if the following is true: 1. if it is an empty string 2. if A and B are correct, AB is correct, 3. if A is correct, (A) and {A} and [A] are also correct.

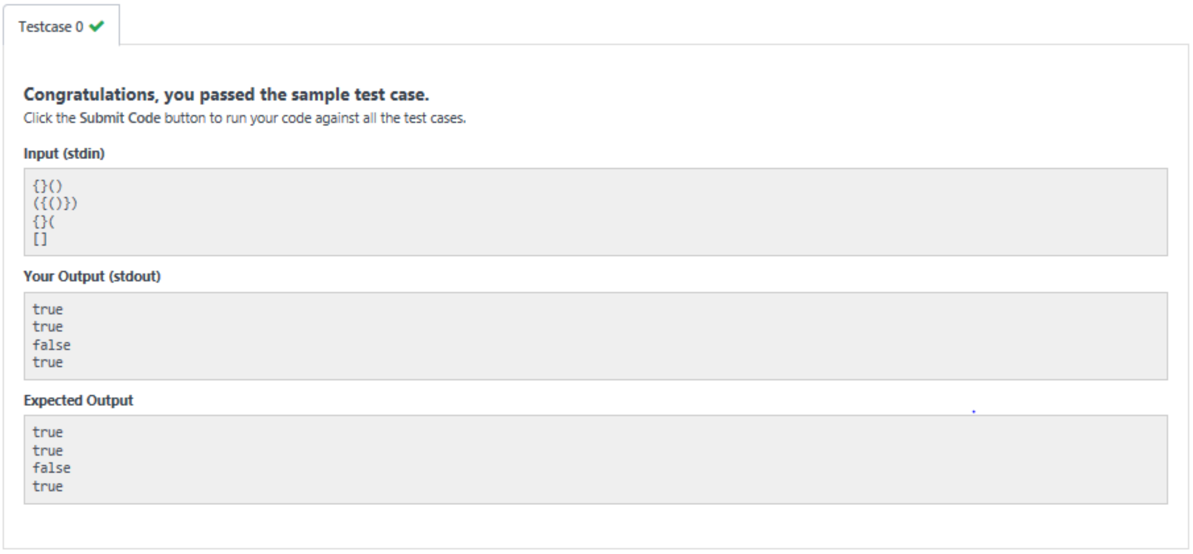
Examples of some correctly balanced strings are: "{}()", "[{()}]", "({()})"

Examples of some unbalanced strings are: "{}(", "({)}", "[[", "}{" etc.

Given a string, determine if it is balanced or not.

```
import java.util.Scanner;

import java.util.HashMap;

import java.util.ArrayDeque;



// ArrayDeque is "likely to be faster than Stack when used as a stack" - Java documentation



class Solution {

  public static void main(String[] args) {

    /* Create HashMap to match opening brackets with closing brackets */

    HashMap<Character, Character> map = new HashMap<>();

    map.put('(', ')');
    map.put('[', ']');

    map.put('{', '}');



    /* Test each expression for validity */

    Scanner scan = new Scanner(System.in);

    while (scan.hasNext()) {

        String expression = scan.next();

        System.out.println(isBalanced(expression, map) ? "true" : "false" );

    }

    scan.close();

}



private static boolean isBalanced(String expression, HashMap<Character, Character> map) {

    if ((expression.length() % 2) != 0) {

        return false; // odd length Strings are not balanced
        }

        ArrayDeque<Character> deque = new ArrayDeque<>(); // use deque as a stack

        for (int i = 0; i < expression.length(); i++) {

            Character ch = expression.charAt(i);

            if (map.containsKey(ch)) {

                deque.push(ch);

            } else if (deque.isEmpty() || ch != map.get(deque.pop())) {

                return false;

            }

        }

        return deque.isEmpty();

    }

}
```

OUTPUT:



Congratulations, you passed the sample test case.
Click the Submit Code button to run your code against all the test cases.

Input (stdin)

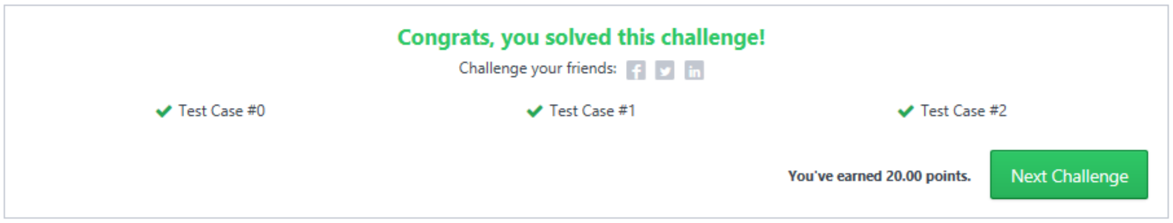
```
{}()
({()})
{}(
[]
```

Your Output (stdout)

```
true
true
false
true
```

Expected Output

```
true
true
false
true
```



Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2

You've earned 20.00 points.

Q31

In computer science, a set is an abstract data type that can store certain values, without any particular order, and no repeated values(Wikipedia). $\{1, 2, 3\}$ is an example of a set, but $\{1, 2, 2\}$ is not a set. Today you will learn how to use sets in java by solving this problem.

You are given n pairs of strings. Two pairs (a, b) and (c, d) are identical if $a = c$ and $b = d$. That also implies (a, b) is *not* same as (b, a). After taking each pair as input, you need to print number of unique pairs you currently have.

Complete the code in the editor to solve this problem.

Program

import java.io.*;

import java.util.*;

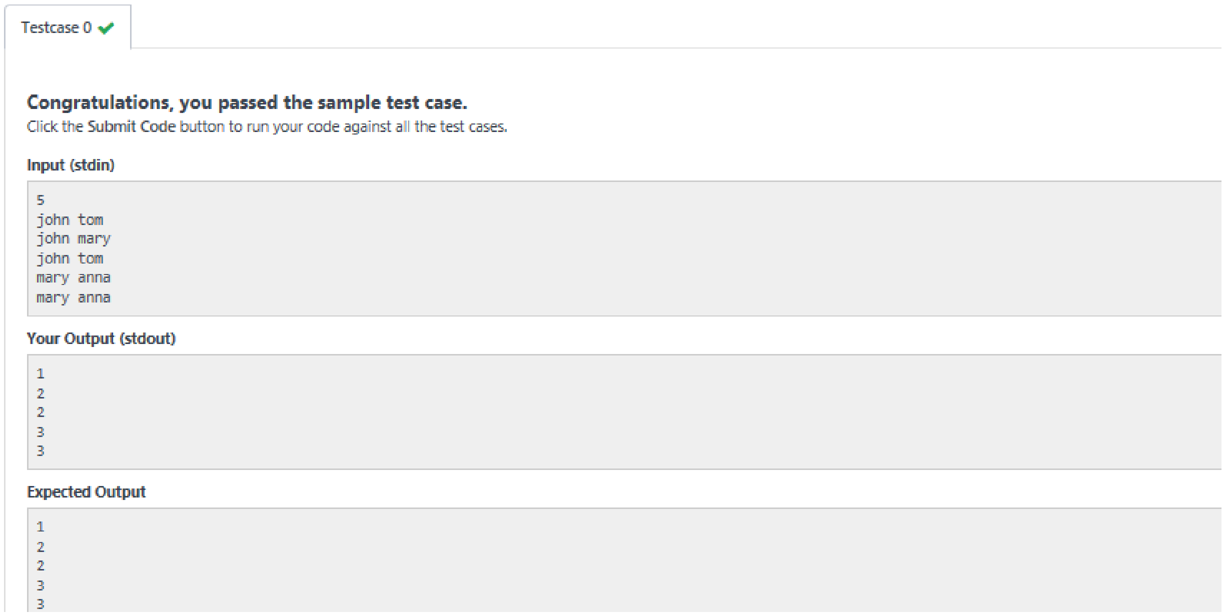
import java.text.*;

import java.math.*;

import java.util.regex.*;

```
public class Solution {

 public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int t = s.nextInt();
        String [] pair_left = new String[t];
        String [] pair_right = new String[t];

        for (int i = 0; i < t; i++) {
            pair_left[i] = s.next();
            pair_right[i] = s.next();
        }
        s.close();



        HashSet<String> set = new HashSet<>(t);

        for (int i = 0; i < t; i++) {

            set.add(pair_left[i] + " " + pair_right[i]);

            System.out.println(set.size());

        }
}}
```

OUTPUT:


Testcase 0 ✓

Congratulations, you passed the sample test case.
Click the Submit Code button to run your code against all the test cases.

Input (stdin)

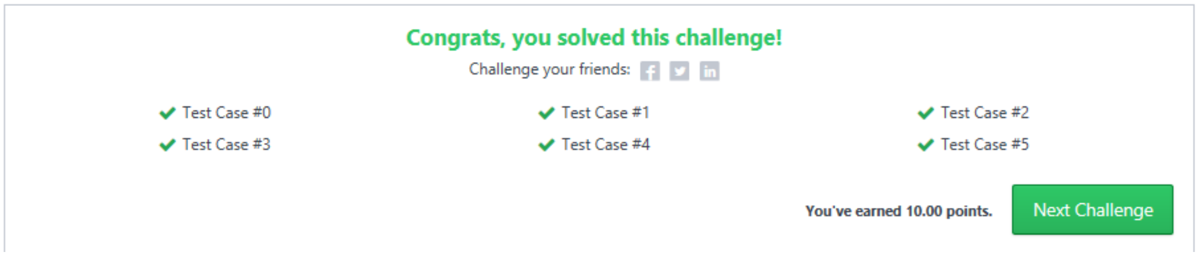
```
5
john tom
john mary
john tom
mary anna
mary anna
```

Your Output (stdout)

```
1
2
2
3
3
```

Expected Output

```
1
2
2
3
3
```


Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2
✓ Test Case #3 ✓ Test Case #4 ✓ Test Case #5

You've earned 10.00 points.

Q32

Welcome to the world of Java! In this challenge, we practice printing to stdout.

The code stubs in your editor declare a *Solution* class and a *main* method. Complete the *main* method by copying the two lines of code below and pasting them inside the body of your *main* method.

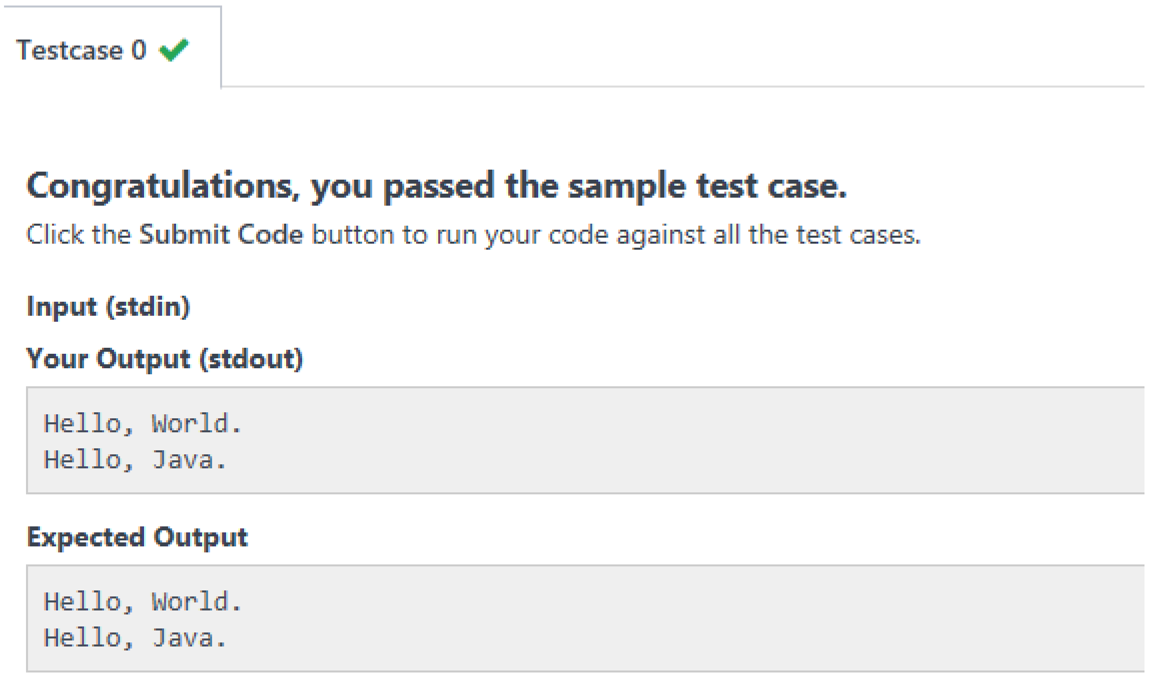
```
System.out.println("Hello, World.");
System.out.println("Hello, Java.");
```

Program

```
public class Solution {


    public static void main(String[] args) {
      System.out.println("Hello, World.");
System.out.println("Hello, Java.");
    }
```

}

Output:



Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

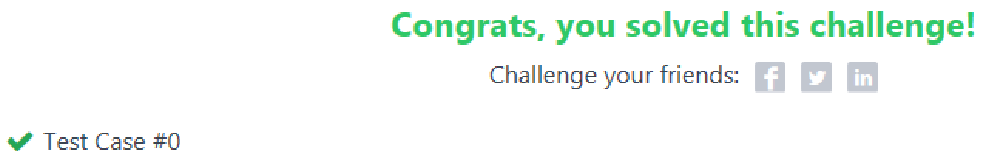
Input (stdin)

Your Output (stdout)

```
Hello, World.
Hello, Java.
```

Expected Output

```
Hello, World.
Hello, Java.
```

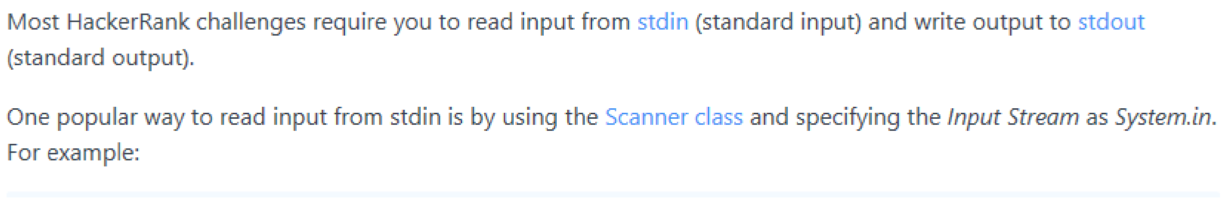


Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0

Q33



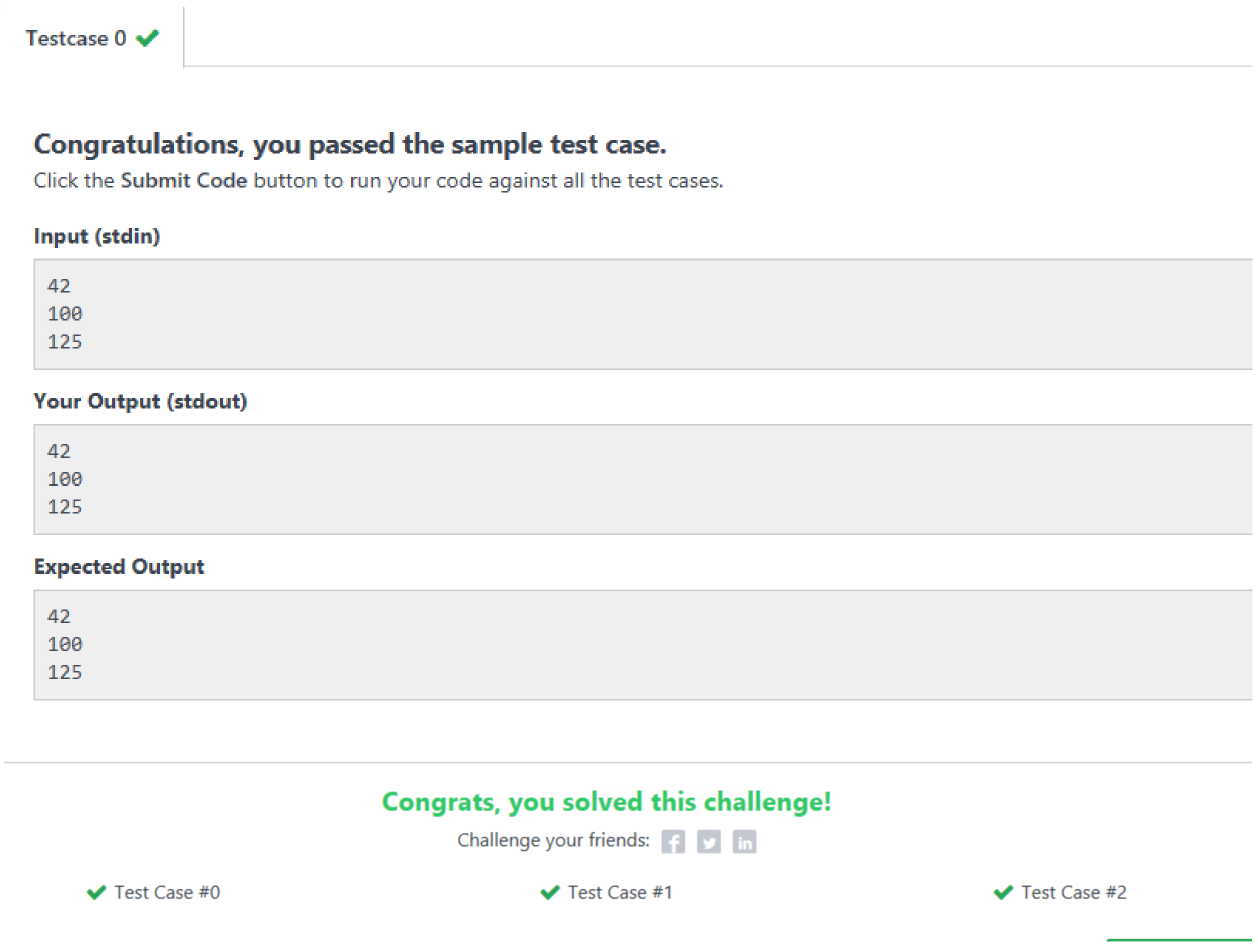
Most HackerRank challenges require you to read input from stdin (standard input) and write output to stdout (standard output).

One popular way to read input from stdin is by using the Scanner class and specifying the *Input Stream* as *System.in*. For example:

Program

```
import java.util.*;


public class Solution {


  public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int a = scan.nextInt();
        int b = scan.nextInt();
        int c = scan.nextInt();

        System.out.println(a+"\n"+b+"\n"+c);
    }
}
```

Output:



Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

```
42
100
125
```

Your Output (stdout)

```
42
100
125
```

Expected Output

```
42
100
125
```

Congrats, you solved this challenge!

Challenge your friends:

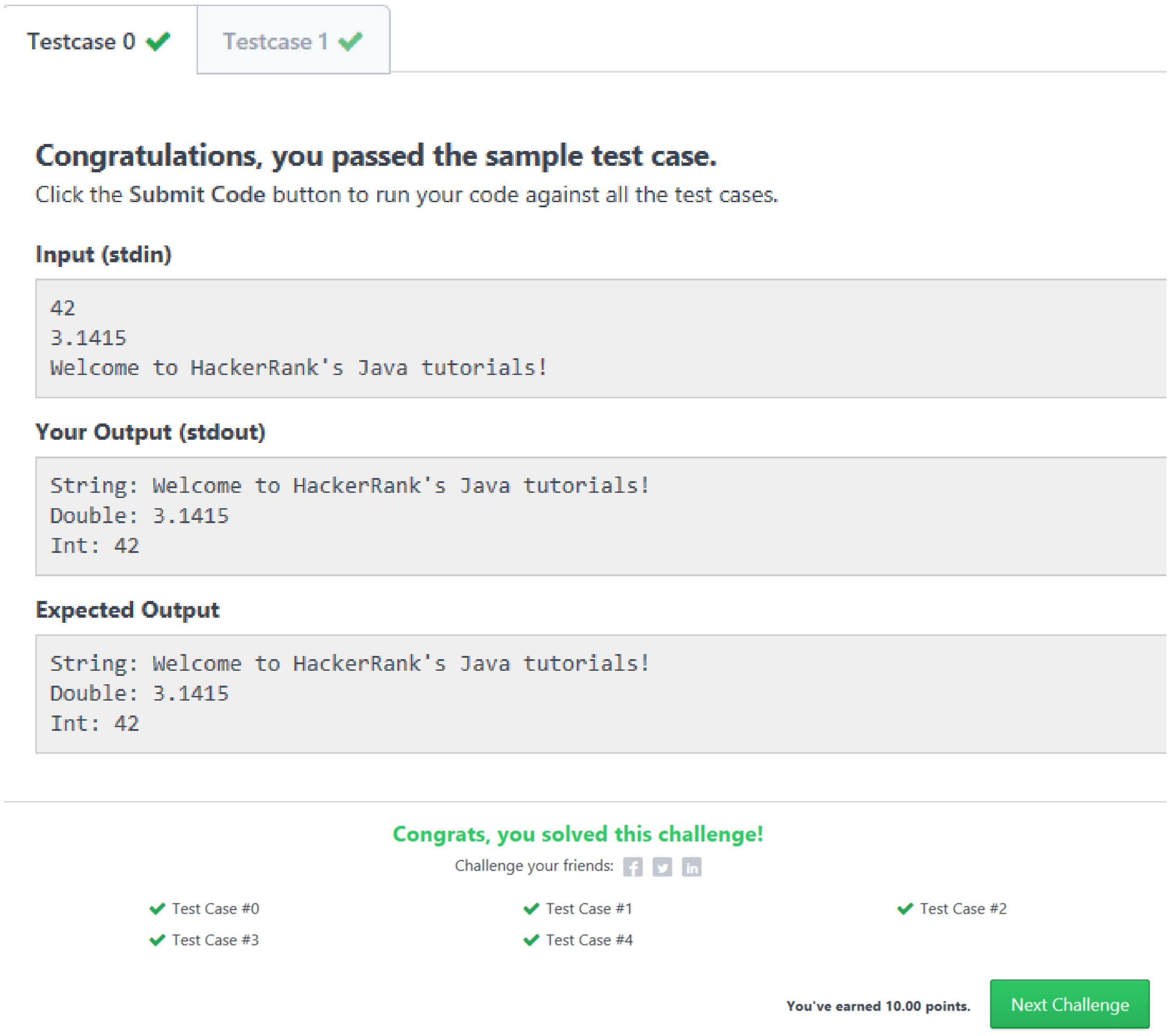
✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2

Q34

In this challenge, you must read an *integer*, a *double*, and a *String* from stdin, then print the values according to the instructions in the *Output Format* section below. To make the problem a little easier, a portion of the code is provided for you in the editor.

```
import java.util.Scanner;


public class Solution {


    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        int i=sc.nextInt();
        double d = sc.nextDouble();
        sc.nextLine();
        String s = sc.nextLine();


        System.out.println("String: " + s);
        System.out.println("Double: " + d);
        System.out.println("Int: " + i);
    }
}
```

OUTPUT:



Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

```
42
3.1415
Welcome to HackerRank's Java tutorials!
```

Your Output (stdout)

```
String: Welcome to HackerRank's Java tutorials!
Double: 3.1415
Int: 42
```

Expected Output

```
String: Welcome to HackerRank's Java tutorials!
Double: 3.1415
Int: 42
```

Congrats, you solved this challenge!

Q35

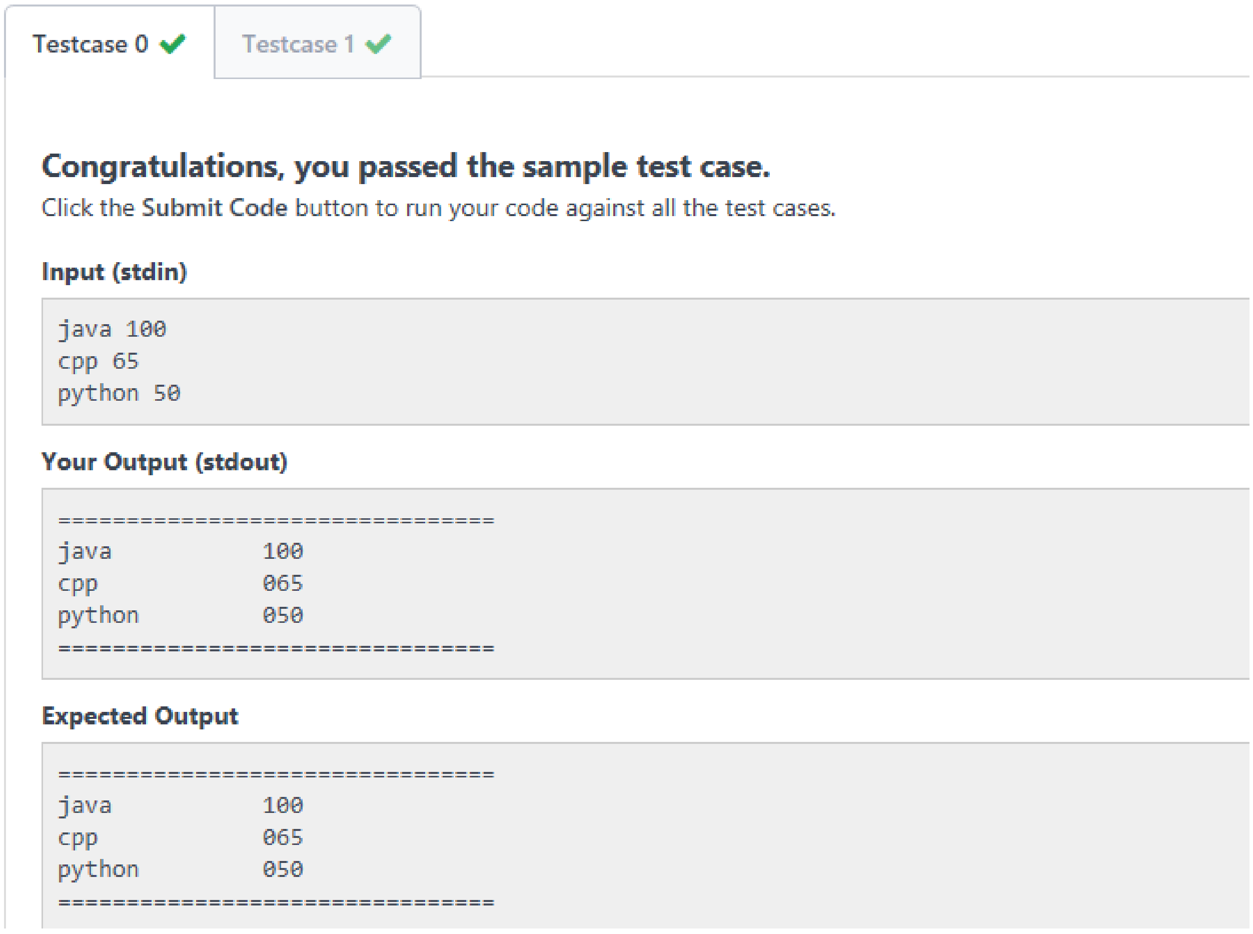
Java's *System.out.printf* function can be used to print formatted output. The purpose of this exercise is to test your understanding of formatting output using *printf*.

To get you started, a portion of the solution is provided for you in the editor; you must format and print the input to complete the solution.

Program

```
import java.util.Scanner;


public class Solution {
    public static void main(String[] args) {
            Scanner sc=new Scanner(System.in);
            System.out.println("================================");
            for(int i=0;i<3;i++)
        {
            String s1=sc.next();
            int x=sc.nextInt();
            System.out.printf("%-15s%03d%n", s1, x);
        }
System.out.println("================================");

    }
}
```

Output


Testcase 0 ✓ Testcase 1 ✓

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

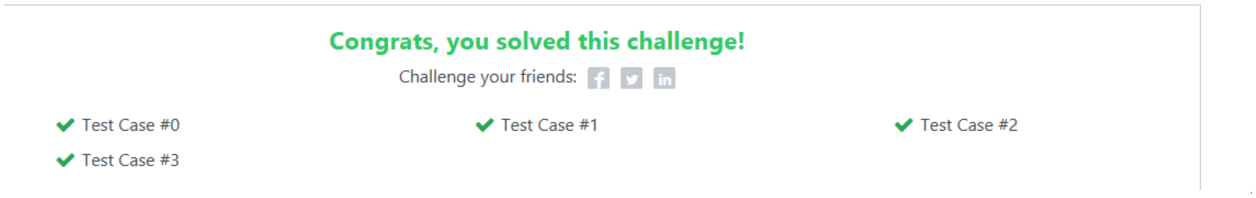
```
java 100
cpp 65
python 50
```

Your Output (stdout)

```
================================
java           100
cpp            065
python         050
================================
```

Expected Output

```
================================
java           100
cpp            065
python         050
================================
```

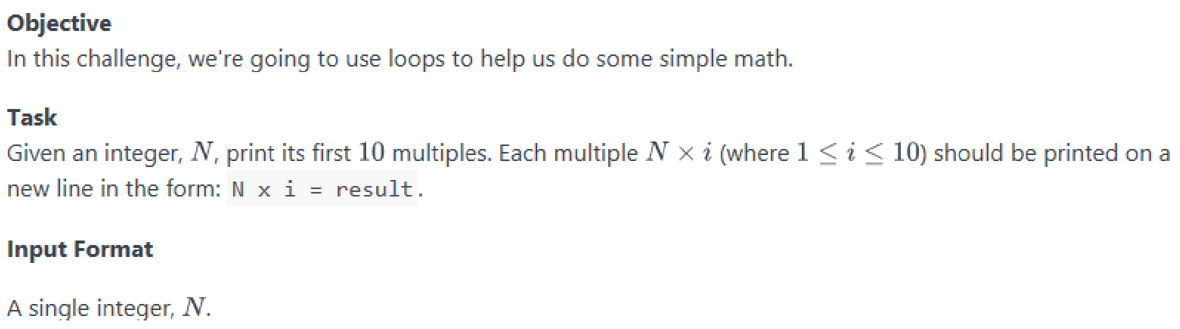

Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2

✓ Test Case #3

Q36


Objective

In this challenge, we're going to use loops to help us do some simple math.

Task

Given an integer, N, print its first 10 multiples. Each multiple $N \times i$ (where $1 \leq i \leq 10$) should be printed on a new line in the form: `N x i = result`.

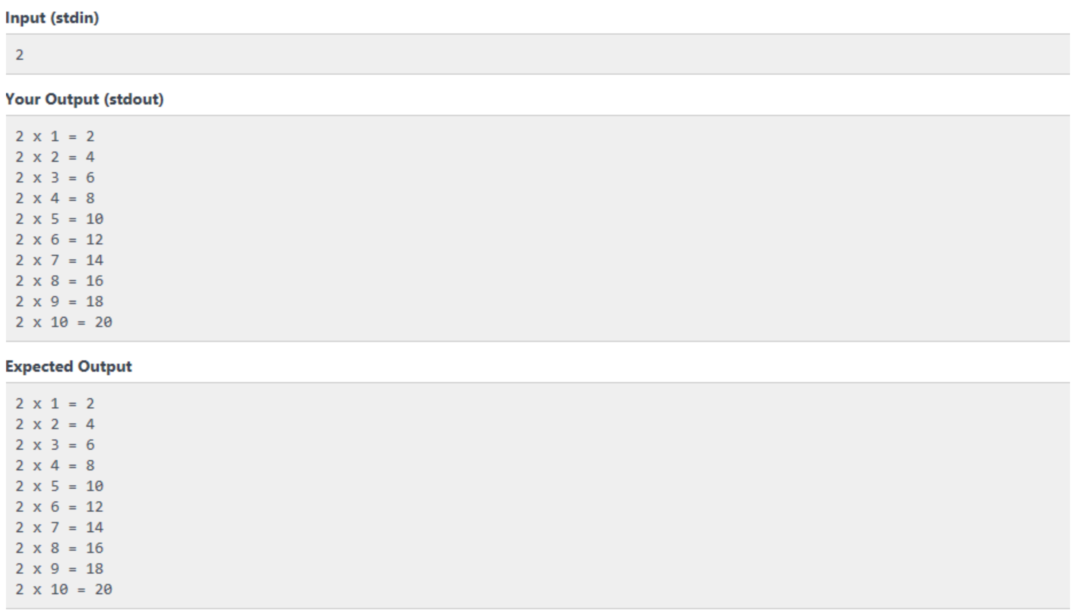
Input Format

A single integer, N.

import java.io.*;

```java
import java.util.*;

import java.text.*;

import java.math.*;

import java.util.regex.*;


public class Solution {


    public static void main(String[] args) {

        Scanner in = new Scanner(System.in);

        int N = in.nextInt();


        for(int i = 1; i <= 10; i++){


            System.out.printf("%d x %d = %d%n", N, i, N*i);

        }

    }

}
```

OUTPUT:



Input (stdin)

```
2
```

Your Output (stdout)

```
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20
```

Expected Output

```
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20
```

Q37

We use the integers a, b, and n to create the following series:

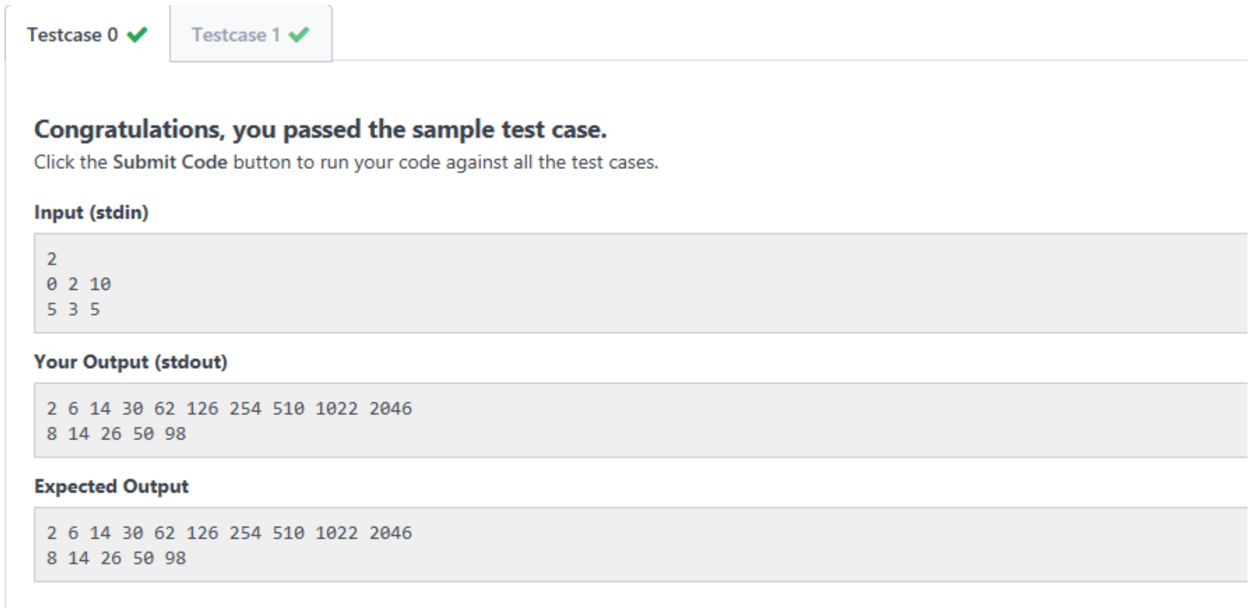
$$(a + 2^0 \cdot b), (a + 2^0 \cdot b + 2^1 \cdot b), \ldots, (a + 2^0 \cdot b + 2^1 \cdot b + \ldots + 2^{n-1} \cdot b)$$

You are given q queries in the form of a, b, and n. For each query, print the series corresponding to the given a, b, and n values as a single line of n space-separated integers.

Program

```
import java.util.*;

import java.io.*;


class Solution{

    public static void main(String []argh){

        Scanner in = new Scanner(System.in);

        int t=in.nextInt();

        for(int i=0;i<t;i++){

            int a = in.nextInt();

            int b = in.nextInt();

            int n = in.nextInt();


            int count = 0;

            int constant = 0;

            int sum = 0;


            while(count < n){

                if(count == 0){

                    constant = 1;

                    sum = a + (constant*b) + sum;

                    }else{

                    constant = constant * 2;

                    sum = (constant * b) + sum;
                }
                System.out.print (sum + " ");
                count += 1;
            }//end while
        System.out.println();





        }
        in.close();
    }
}
```

Output



Testcase 0 ✓ Testcase 1 ✓

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

```
2
0 2 10
5 3 5
```

Your Output (stdout)

```
2 6 14 30 62 126 254 510 1022 2046
8 14 26 50 98
```

Expected Output

```
2 6 14 30 62 126 254 510 1022 2046
8 14 26 50 98
```

Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2

✓ Test Case #3 ✓ Test Case #4

Q38

Java has 8 primitive data types; *char, boolean, byte, short, int, long, float, and double*. For this exercise, we'll work with the primitives used to hold integer values (*byte, short, int,* and *long*):

- A *byte* is an 8-bit signed integer.
- A *short* is a 16-bit signed integer.
- An *int* is a 32-bit signed integer.
- A *long* is a 64-bit signed integer.

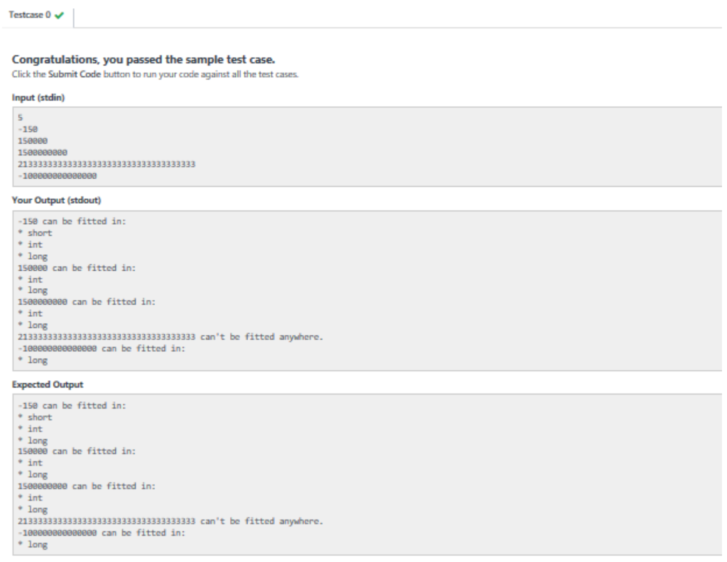
Given an input integer, you must determine which primitive data types are capable of properly storing that input.

To get you started, a portion of the solution is provided for you in the editor.

Program

```
import java.util.*;

import java.io.*;




class Solution{

    public static void main(String []argh)

    {




        Scanner sc = new Scanner(System.in);

        int t=sc.nextInt();



    for(int i=0;i<t;i++)

    {


        try

        {

            long x=sc.nextLong();
            System.out.println(x+" can be fitted in:");
            if(x>=-128 && x<=127)System.out.println("* byte");
            //Complete the code
            if(x >= -Math.pow(2, 15) && x <= Math.pow(2, 15) - 1)
               System.out.println("* short");
            if(x >= -Math.pow(2, 31) && x <= Math.pow(2, 31) - 1)
               System.out.println("* int");
            if(x >= -Math.pow(2, 63) && x <= Math.pow(2, 63) - 1)
               System.out.println("* long");
         }
         catch(Exception e)
         {
            System.out.println(sc.next()+" can't be fitted anywhere.");
         }

      }
      }}
```

Output



Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

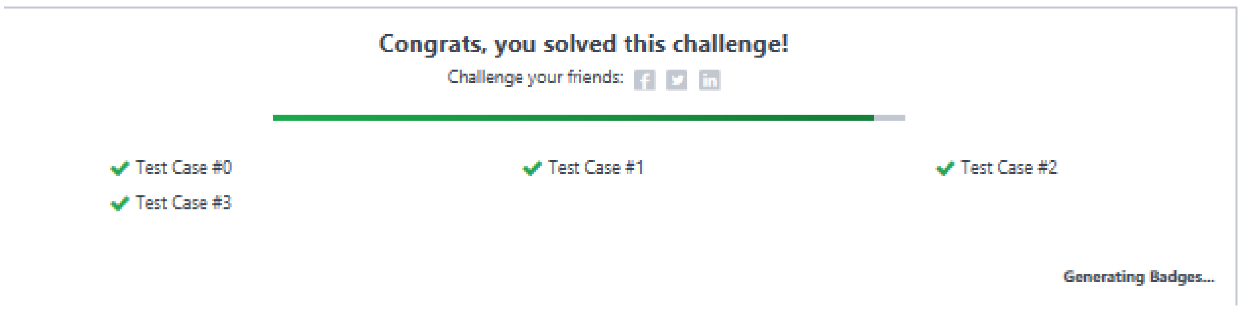
```
5
-150
150000
1500000000
213333333333333333333333333333333333
-100000000000000
```

Your Output (stdout)

```
-150 can be fitted in:
* short
* int
* long
150000 can be fitted in:
* int
* long
1500000000 can be fitted in:
* int
* long
213333333333333333333333333333333333 can't be fitted anywhere.
-100000000000000 can be fitted in:
* long
```

Expected Output

```
-150 can be fitted in:
* short
* int
* long
150000 can be fitted in:
* int
* long
1500000000 can be fitted in:
* int
* long
213333333333333333333333333333333333 can't be fitted anywhere.
-100000000000000 can be fitted in:
* long
```



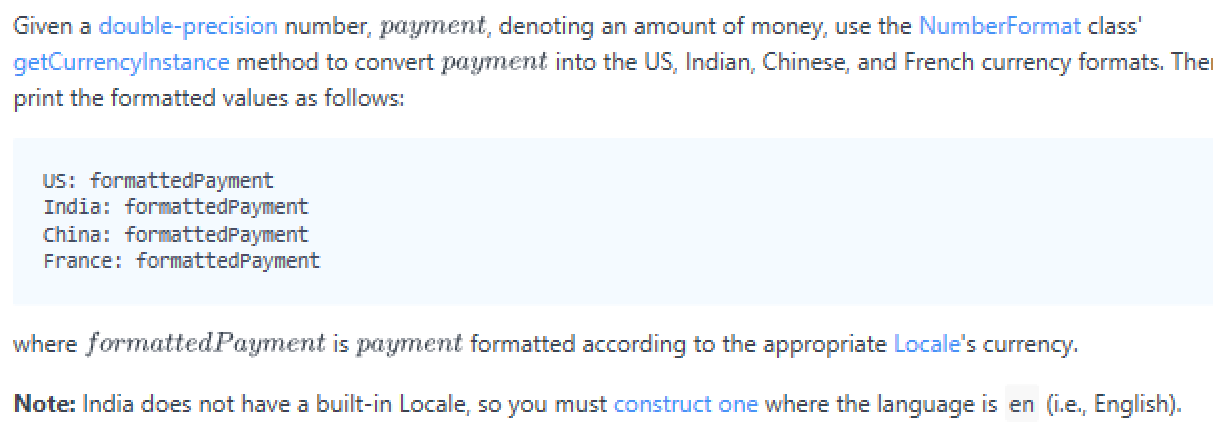
Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2
✓ Test Case #3

Generating Badges...

Q39



Given a double-precision number, $payment$, denoting an amount of money, use the NumberFormat class' getCurrencyInstance method to convert $payment$ into the US, Indian, Chinese, and French currency formats. Thei
print the formatted values as follows:

```
US: formattedPayment
India: formattedPayment
China: formattedPayment
France: formattedPayment
```

where $formattedPayment$ is $payment$ formatted according to the appropriate Locale's currency.

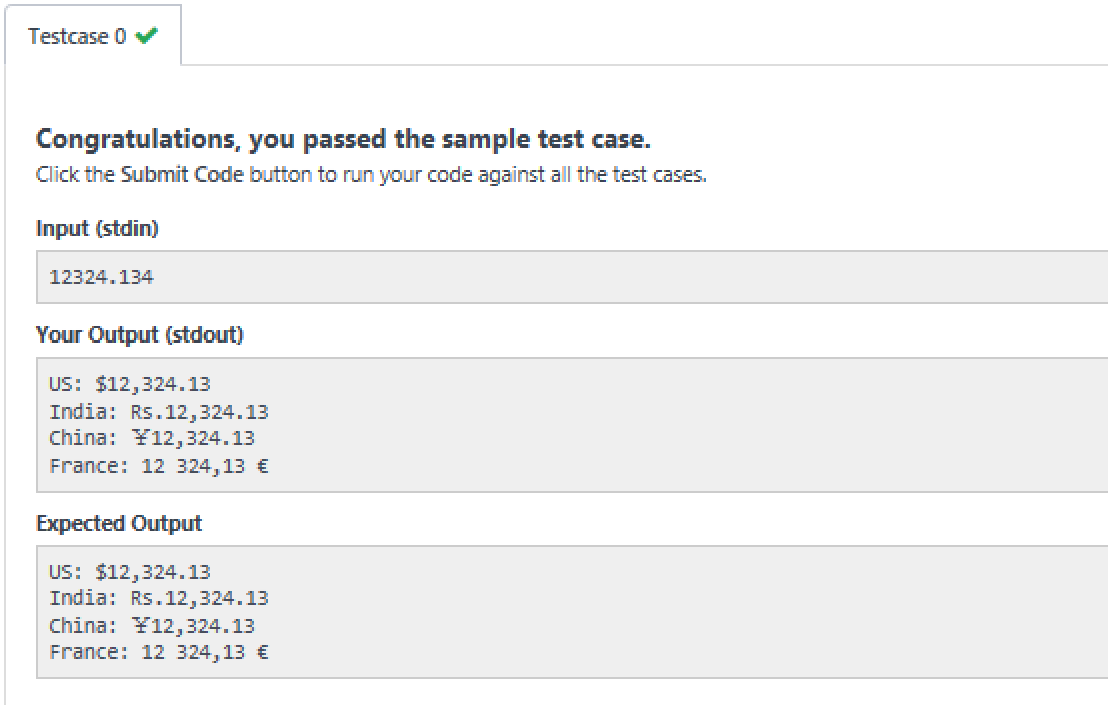
Note: India does not have a built-in Locale, so you must construct one where the language is `en` (i.e., English).

Program

```
    import java.util.*;
import java.text.*;
//import java.util.locale;

public class Solution {

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    double payment = scanner.nextDouble();
    scanner.close();

    Locale INDIA = new Locale("en","IN");
```

```
        NumberFormat us    = NumberFormat.getCurrencyInstance(Locale.US);
        NumberFormat india  = NumberFormat.getCurrencyInstance(INDIA);
        NumberFormat china  = NumberFormat.getCurrencyInstance(Locale.CHINA);
        NumberFormat france = NumberFormat.getCurrencyInstance(Locale.FRANCE);

        System.out.println("US: " + us.format(payment));
        System.out.println("India: " + india.format(payment));
        System.out.println("China: " + china.format(payment));
        System.out.println("France: " + france.format(payment));
    }
}
```

Output



Testcase 0 ✓

Congratulations, you passed the sample test case.
Click the Submit Code button to run your code against all the test cases.

Input (stdin)

```
12324.134
```

Your Output (stdout)

```
US: $12,324.13
India: Rs.12,324.13
China: ￥12,324.13
France: 12 324,13 €
```

Expected Output

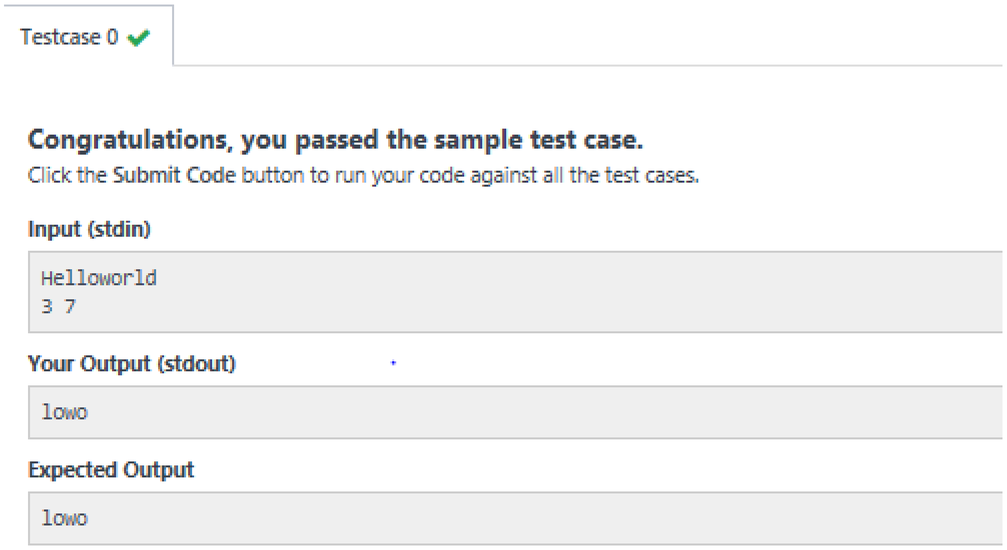
```
US: $12,324.13
India: Rs.12,324.13
China: ￥12,324.13
France: 12 324,13 €
```

Q40

Given a string, s, and two indices, $start$ and end, print a substring consisting of all characters in the inclusive range from $start$ to $end - 1$. You'll find the *String* class' substring method helpful in completing this challenge.

Program

```
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

public class Solution {

    public static void main(String[] args) {
 Scanner in = new Scanner(System.in);
 System.out.print(in.next().substring(in.nextInt(),in.nextInt()));
}
  }
```



Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the Submit Code button to run your code against all the test cases.

Input (stdin)

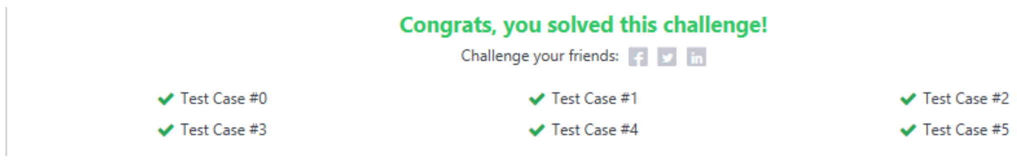
```
Helloworld
3 7
```

Your Output (stdout)

```
lowo
```

Expected Output

```
lowo
```

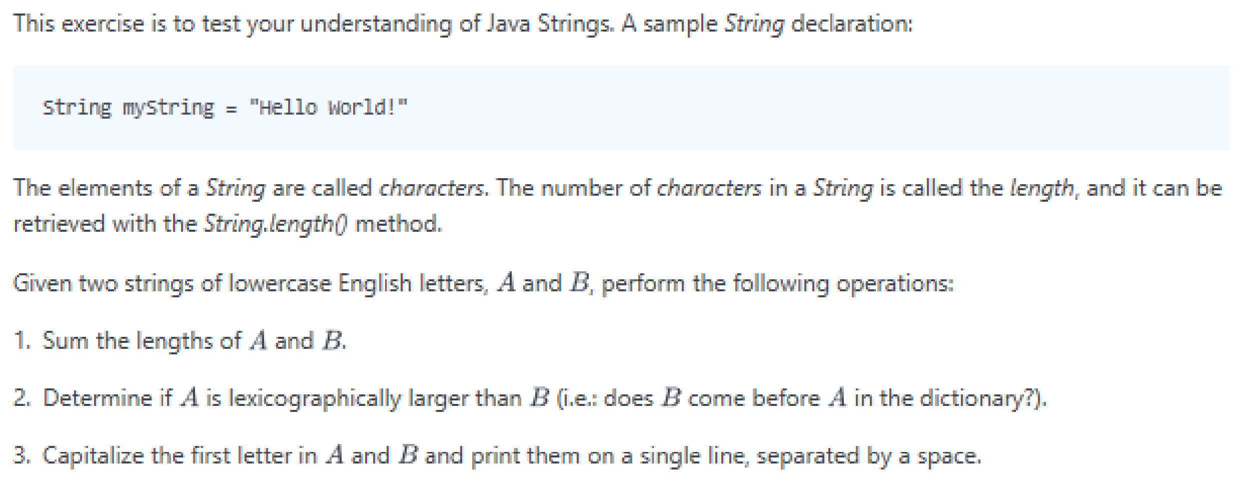


Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2
✓ Test Case #3 ✓ Test Case #4 ✓ Test Case #5

Q41


This exercise is to test your understanding of Java Strings. A sample *String* declaration:

```
String myString = "Hello World!"
```

The elements of a *String* are called *characters*. The number of *characters* in a *String* is called the *length*, and it can be retrieved with the *String.length()* method.

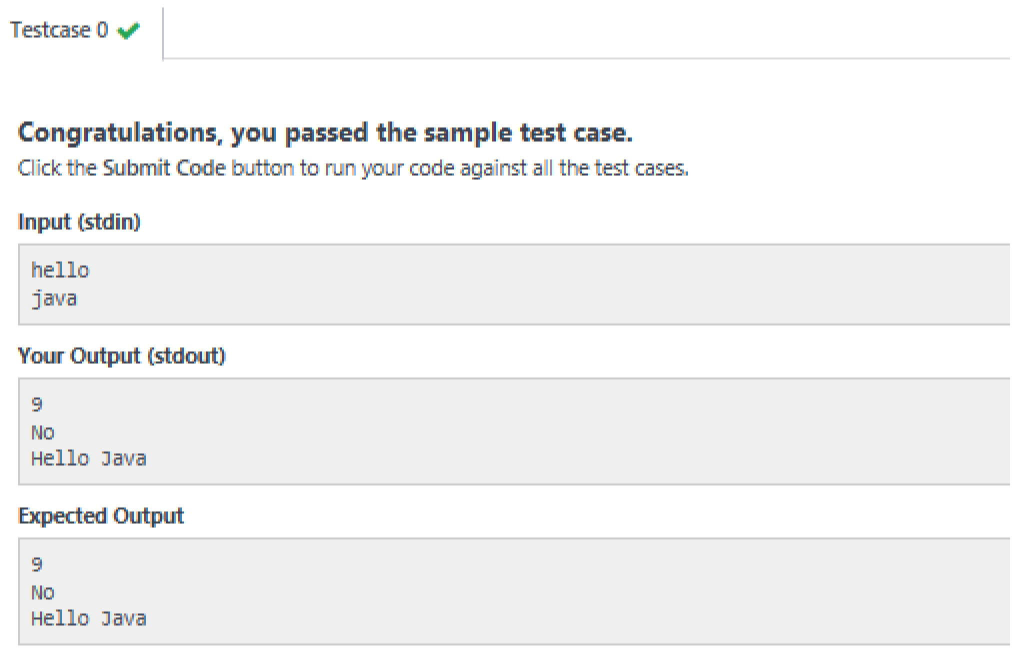
Given two strings of lowercase English letters, A and B, perform the following operations:

1. Sum the lengths of A and B.
2. Determine if A is lexicographically larger than B (i.e.: does B come before A in the dictionary?).
3. Capitalize the first letter in A and B and print them on a single line, separated by a space.

PROGRAM

```
import java.io.*;
import java.util.*;


public class Solution {


    public static void main(String[] args) {


        Scanner sc=new Scanner(System.in);
        String A=sc.next();
        String B=sc.next();
     System.out.println(A.length()+B.length());
System.out.println(A.compareTo(B)>0?"Yes":"No");
System.out.println(A.substring(0, 1).toUpperCase()+A.substring(1, A.length())+" "+B.substring(0,
1).toUpperCase()+B.substring(1, B.length()));
        /* Enter your code here. Print output to STDOUT. */


    }
}
```

Output:


Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the Submit Code button to run your code against all the test cases.

Input (stdin)

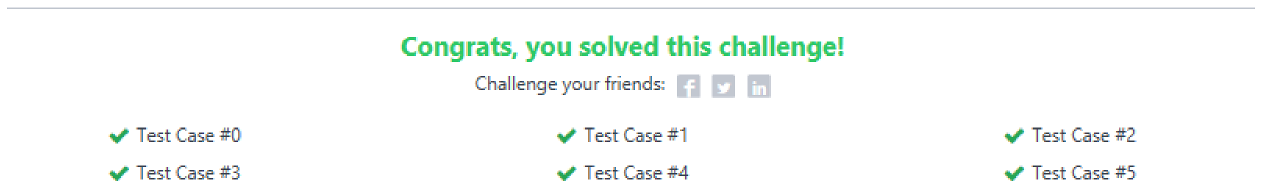
```
hello
java
```

Your Output (stdout)

```
9
No
Hello Java
```

Expected Output

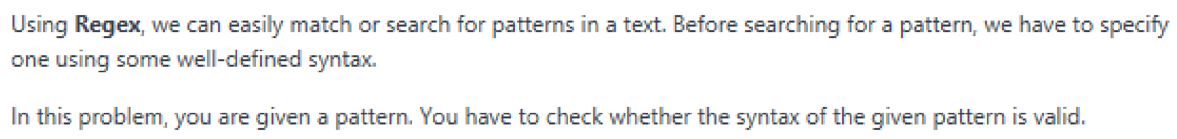
```
9
No
Hello Java
```


Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0	✓ Test Case #1	✓ Test Case #2
✓ Test Case #3	✓ Test Case #4	✓ Test Case #5

Q42


Using **Regex**, we can easily match or search for patterns in a text. Before searching for a pattern, we have to specify one using some well-defined syntax.

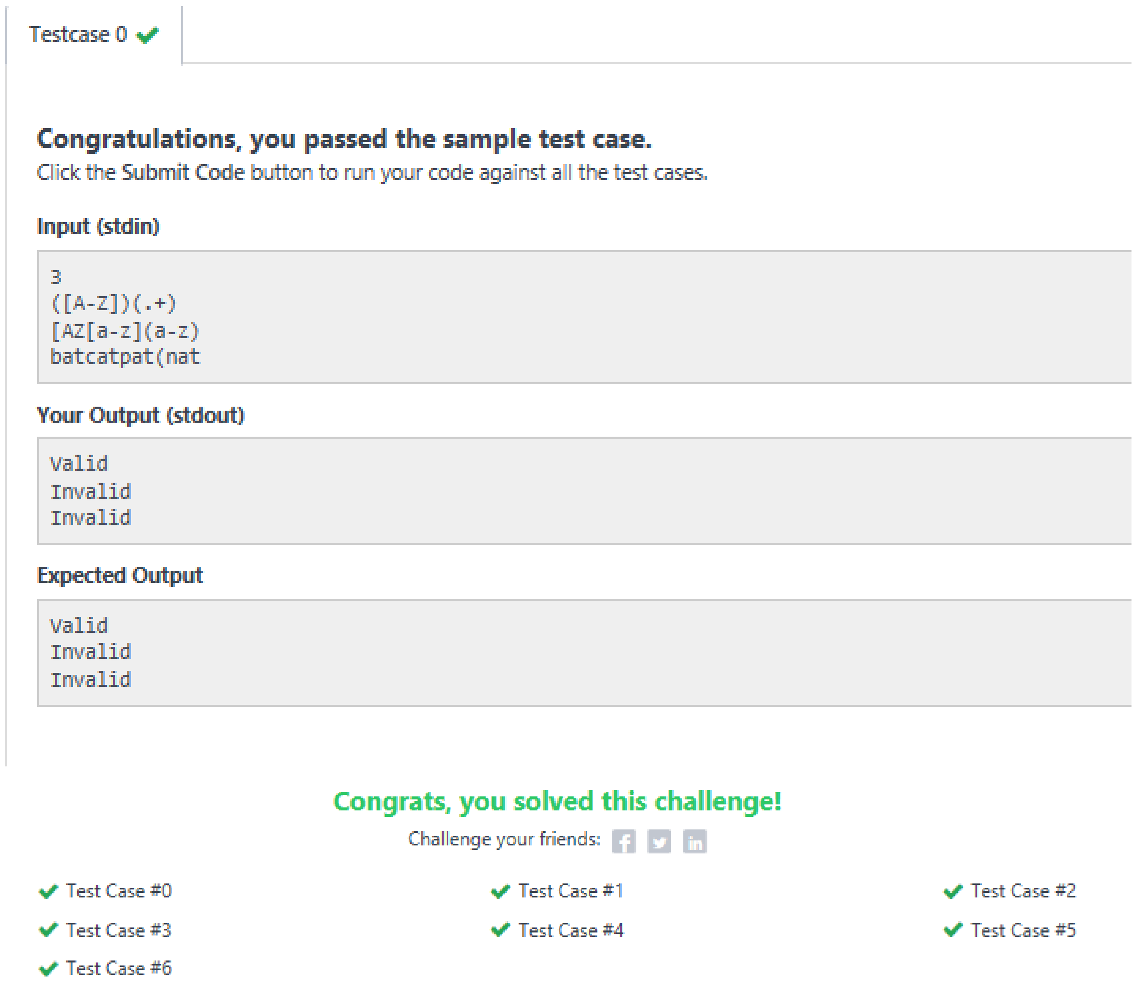
In this problem, you are given a pattern. You have to check whether the syntax of the given pattern is valid.

Program

```
import java.util.Scanner;
import java.util.regex.*;

public class Solution {
  public static void main(String[] args){
    Scanner in = new Scanner(System.in);
    int testCases = Integer.parseInt(in.nextLine());
    while(testCases > 0){
      String pattern = in.nextLine();
```

```
            try {
                Pattern.compile(pattern);
                System.out.println("Valid");
            } catch (PatternSyntaxException e) {
                System.out.println("Invalid");
            }
            testCases--;
        }
    }
}
```

Output:



Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the Submit Code button to run your code against all the test cases.

Input (stdin)

```
3
([A-Z])(.+)
[AZ[a-z](a-z)
batcatpat(nat
```

Your Output (stdout)

```
Valid
Invalid
Invalid
```

Expected Output

```
Valid
Invalid
Invalid
```

Congrats, you solved this challenge!

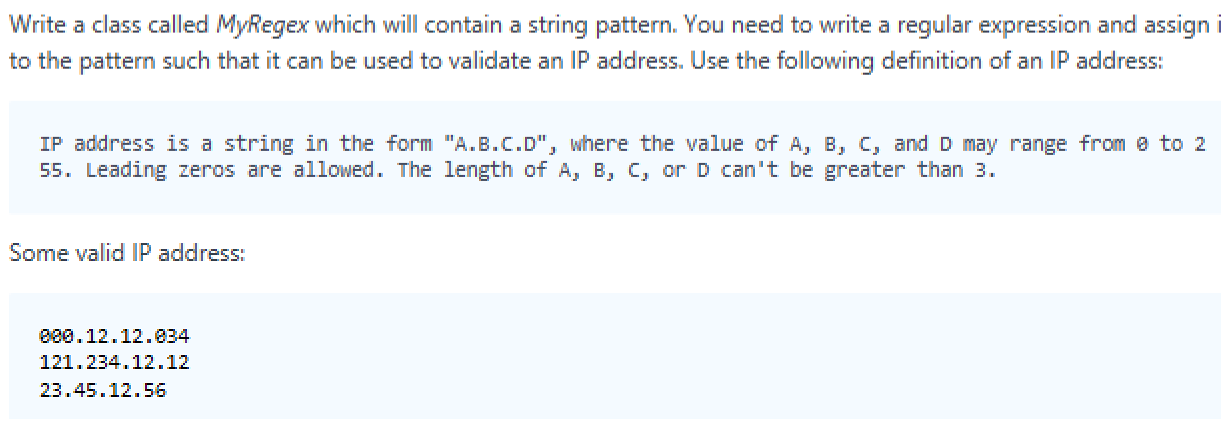
Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2

✓ Test Case #3 ✓ Test Case #4 ✓ Test Case #5

✓ Test Case #6

Q43


Write a class called *MyRegex* which will contain a string pattern. You need to write a regular expression and assign i
to the pattern such that it can be used to validate an IP address. Use the following definition of an IP address:

```
IP address is a string in the form "A.B.C.D", where the value of A, B, C, and D may range from 0 to 2
55. Leading zeros are allowed. The length of A, B, C, or D can't be greater than 3.
```

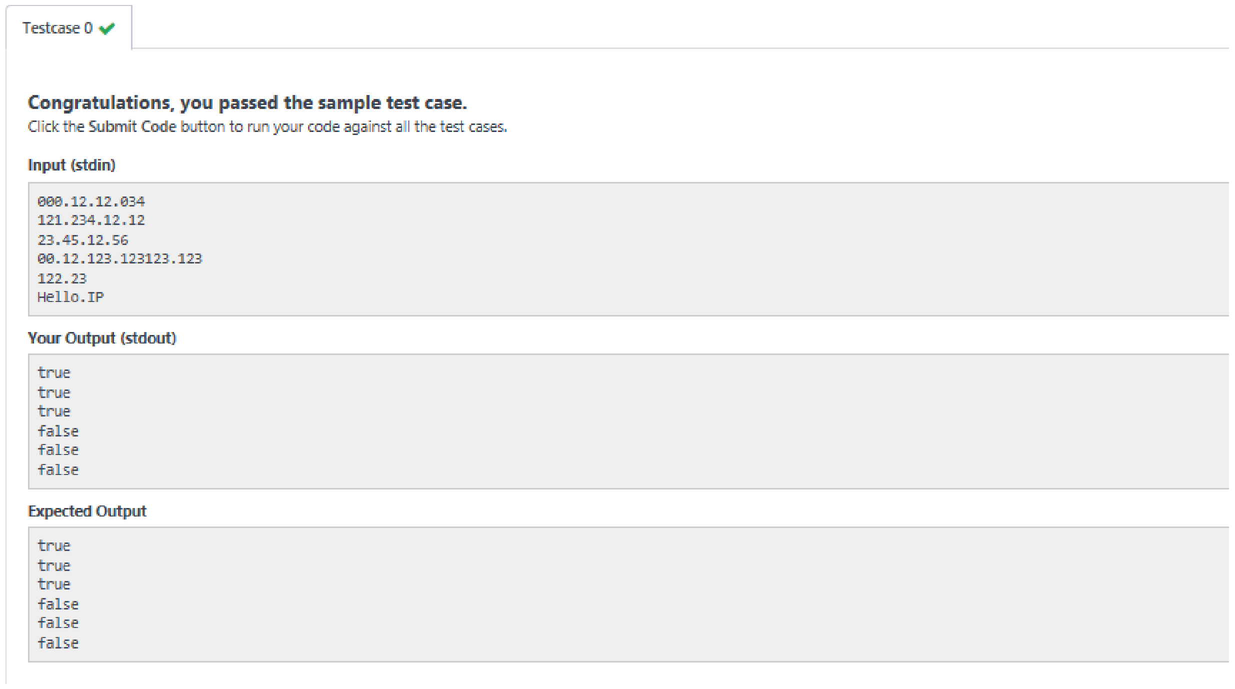
Some valid IP address:

```
000.12.12.034
121.234.12.12
23.45.12.56
```

Program

```
import java.util.regex.Matcher;

import java.util.regex.Pattern;

import java.util.Scanner;


class Solution{


    public static void main(String[] args){

        Scanner in = new Scanner(System.in);

        while(in.hasNext()){

            String IP = in.next();

            System.out.println(IP.matches(new MyRegex().pattern));

        }


    }

}class MyRegex{

    String Zip="([\\d]{1,2}|(0|1)[\\d]{1,2}|(2)[0-5]{1,2})";

    String pattern=Zip + "\\."+Zip+"\\."+Zip+"\\."+Zip;

}
```

Output



Testcase 0 ✓

Congratulations, you passed the sample test case.
Click the Submit Code button to run your code against all the test cases.

Input (stdin)

```
000.12.12.034
121.234.12.12
23.45.12.56
00.12.123.123123.123
122.23
Hello.IP
```

Your Output (stdout)

```
true
true
true
false
false
false
```

Expected Output

```
true
true
true
false
false
false
```

Ask your friends for help!

✓ Test Case #0 ✓ Test Case #1

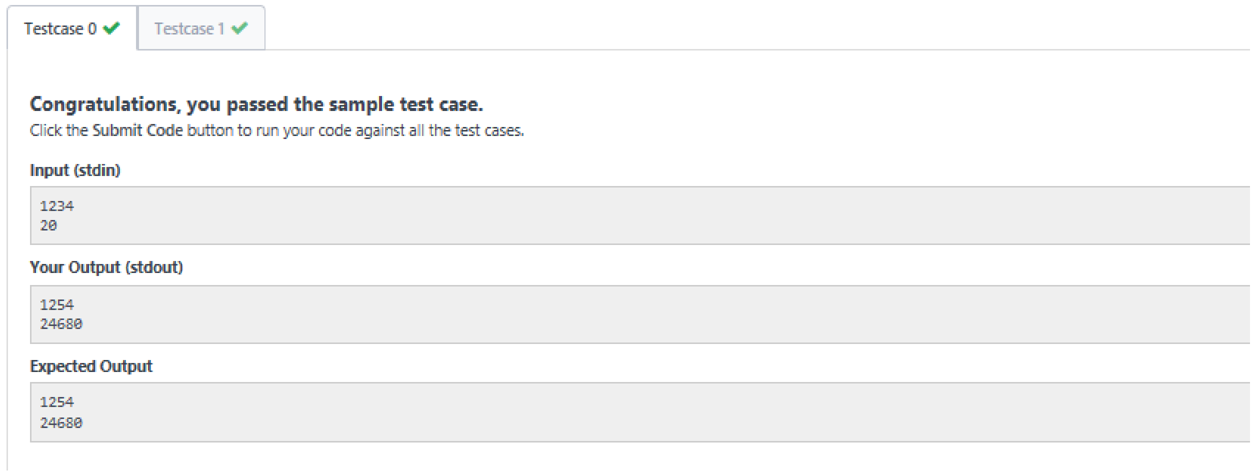
Q44

In this problem, you have to add and multiply huge numbers! These numbers are so big that you can't contain them in any ordinary data types like a long integer.

Use the power of Java's BigInteger class and solve this problem.

```
import java.io.*;

import java.util.*;

import java.text.*;

import java.math.*;

import java.util.regex.*;


public class Solution {


    public static void main(String[] args) {


     Scanner sc = new Scanner(System.in);
     BigInteger bi1 = new BigInteger(sc.next());
     BigInteger bi2 = new BigInteger(sc.next());


       BigInteger  bi3, bi4;
       bi3 = bi1.add(bi2);
       bi4 = bi1.multiply(bi2);
       System.out.println( bi3);
       System.out.println( bi4);



}
    }
```

Output



Testcase 0 ✓ Testcase 1 ✓

Congratulations, you passed the sample test case.
Click the Submit Code button to run your code against all the test cases.

Input (stdin)

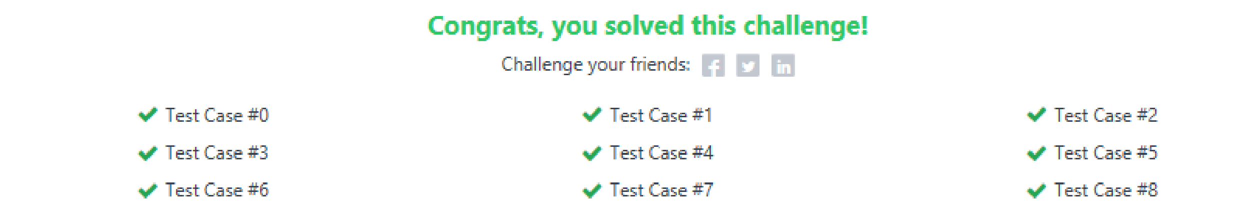
```
1234
20
```

Your Output (stdout)

```
1254
24680
```

Expected Output

```
1254
24680
```

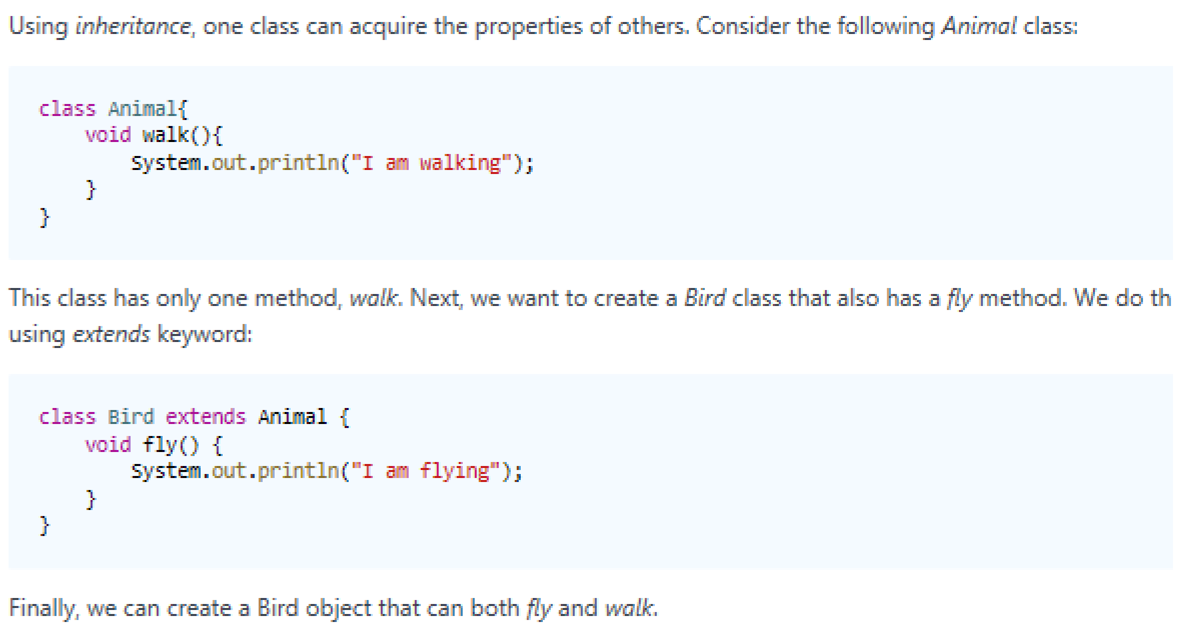


Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2
✓ Test Case #3 ✓ Test Case #4 ✓ Test Case #5
✓ Test Case #6 ✓ Test Case #7 ✓ Test Case #8

Q45



Using *inheritance*, one class can acquire the properties of others. Consider the following *Animal* class:

```
class Animal{
    void walk(){
        System.out.println("I am walking");
    }
}
```

This class has only one method, *walk*. Next, we want to create a *Bird* class that also has a *fly* method. We do th
using *extends* keyword:

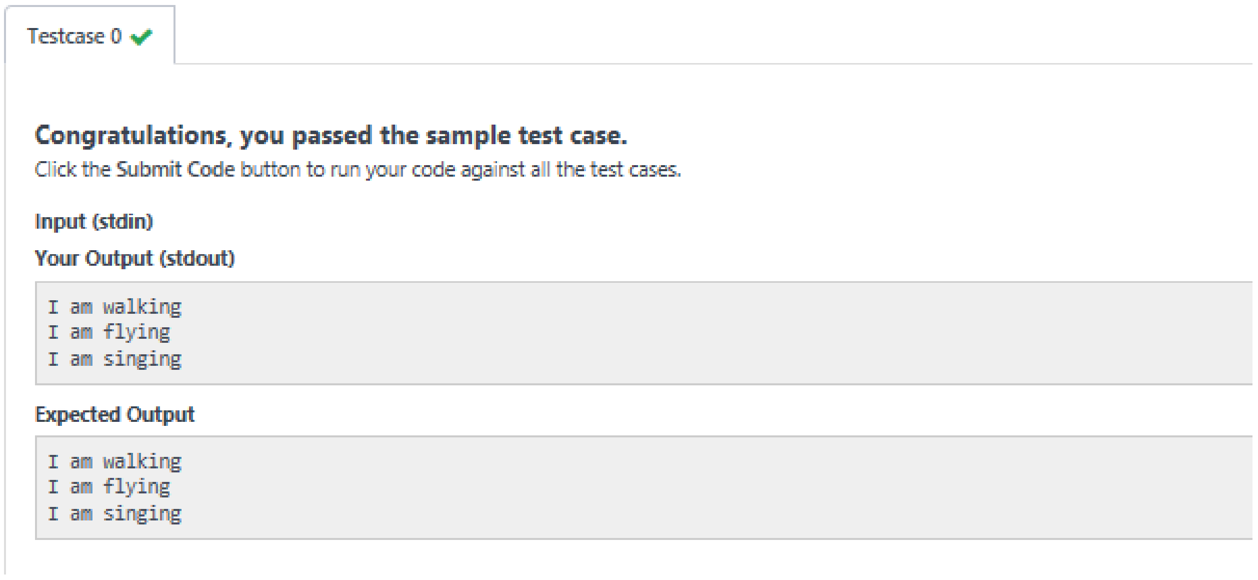
```
class Bird extends Animal {
    void fly() {
        System.out.println("I am flying");
    }
}
```

Finally, we can create a Bird object that can both *fly* and *walk*.

Program

```
import java.io.*;

import java.util.*;

import java.text.*;

import java.math.*;

import java.util.regex.*;


class Animal{

            void walk(){

                        System.out.println("I am walking");

            }

}



class Bird extends Animal{

            void fly(){

                        System.out.println("I am flying");

   }
```

```
          void sing()
    {
        System.out.println("I am singing");
    }

 public static void main(String[]args)
 {
     Bird b=new Bird();
     b.walk();
     b.fly();
     b.sing();
 }



}
public class Solution{

   public static void main(String args[]){


               Bird bird = new Bird();
               bird.walk();
               bird.fly();
      bird.sing();


   }
}
```


Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the Submit Code button to run your code against all the test cases.

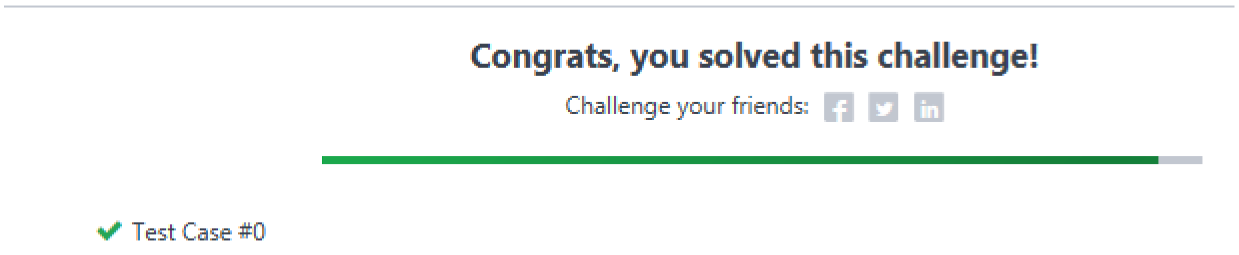
Input (stdin)

Your Output (stdout)

```
I am walking
I am flying
I am singing
```

Expected Output

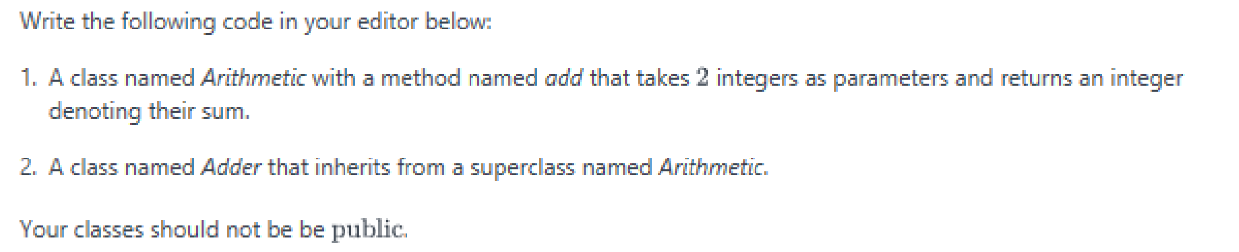
```
I am walking
I am flying
I am singing
```


Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0

Q46


Write the following code in your editor below:

1. A class named *Arithmetic* with a method named *add* that takes 2 integers as parameters and returns an integer denoting their sum.
2. A class named *Adder* that inherits from a superclass named *Arithmetic*.

Your classes should not be be public.

import java.io.*;

import java.util.*;

import java.text.*;

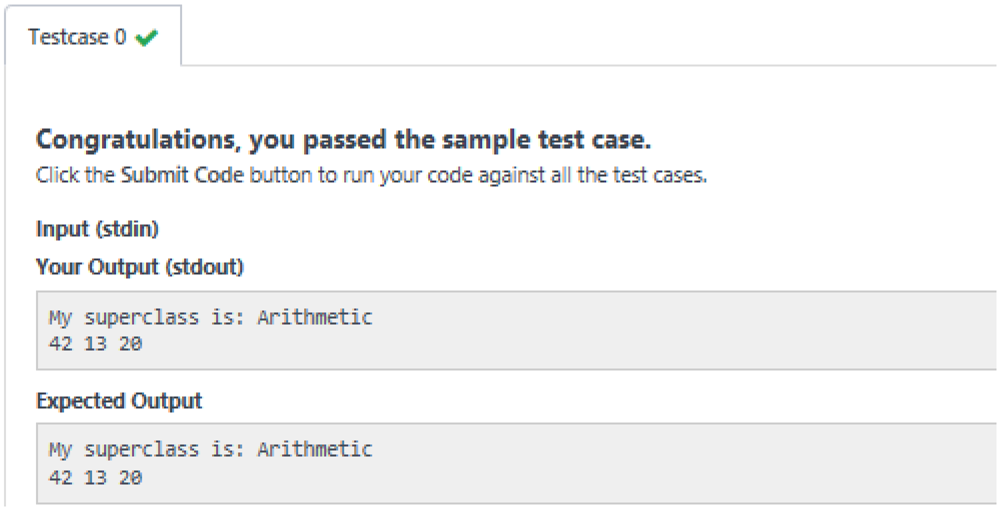
import java.math.*;

import java.util.regex.*;

class Arithmetic

```
{
    int a;
    int b;
    int add(int x,int y)
    {
        a=x;
        b=y;
        return (a+b);
    }
}
class Adder extends Arithmetic
{
        }
public class Solution{
    public static void main(String []args){
        // Create a new Adder object
        Adder a = new Adder();

        // Print the name of the superclass on a new line
        System.out.println("My superclass is: " + a.getClass().getSuperclass().getName());

        // Print the result of 3 calls to Adder's `add(int,int)` method as 3 space-separated integers:
        System.out.print(a.add(10,32) + " " + a.add(10,3) + " " + a.add(10,10) + "\n");
     }
}
```

OUTPUT:


Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the Submit Code button to run your code against all the test cases.

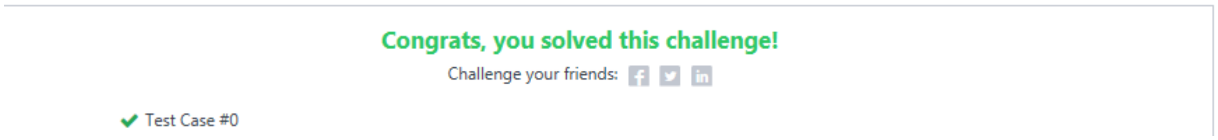
Input (stdin)

Your Output (stdout)

```
My superclass is: Arithmetic
42 13 20
```

Expected Output

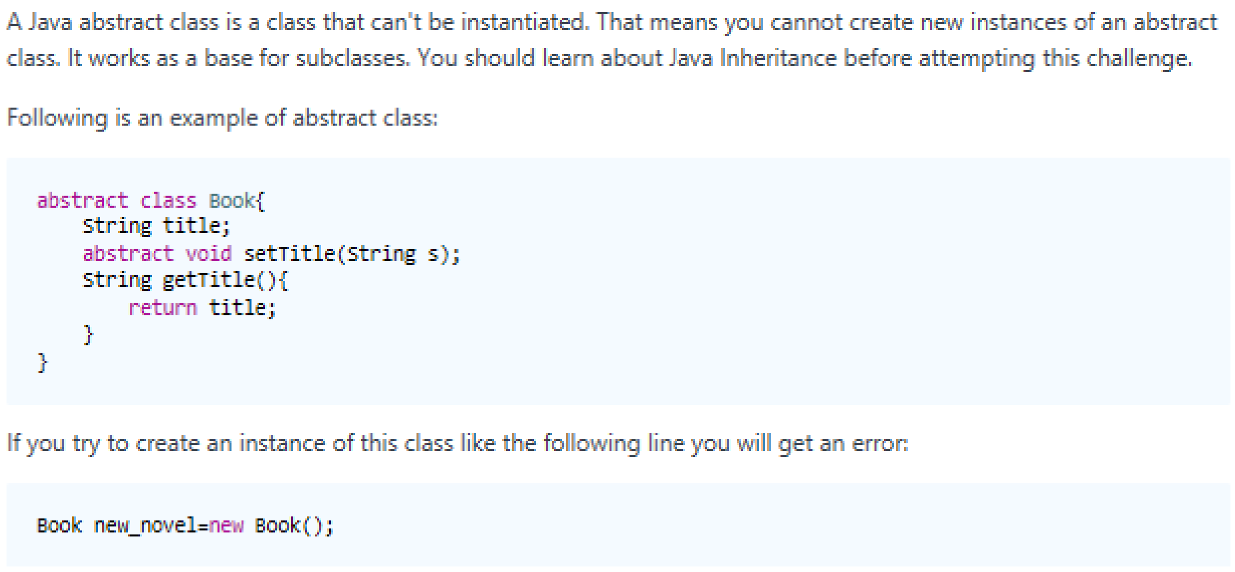
```
My superclass is: Arithmetic
42 13 20
```


Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0

Q47


A Java abstract class is a class that can't be instantiated. That means you cannot create new instances of an abstract class. It works as a base for subclasses. You should learn about Java Inheritance before attempting this challenge.

Following is an example of abstract class:

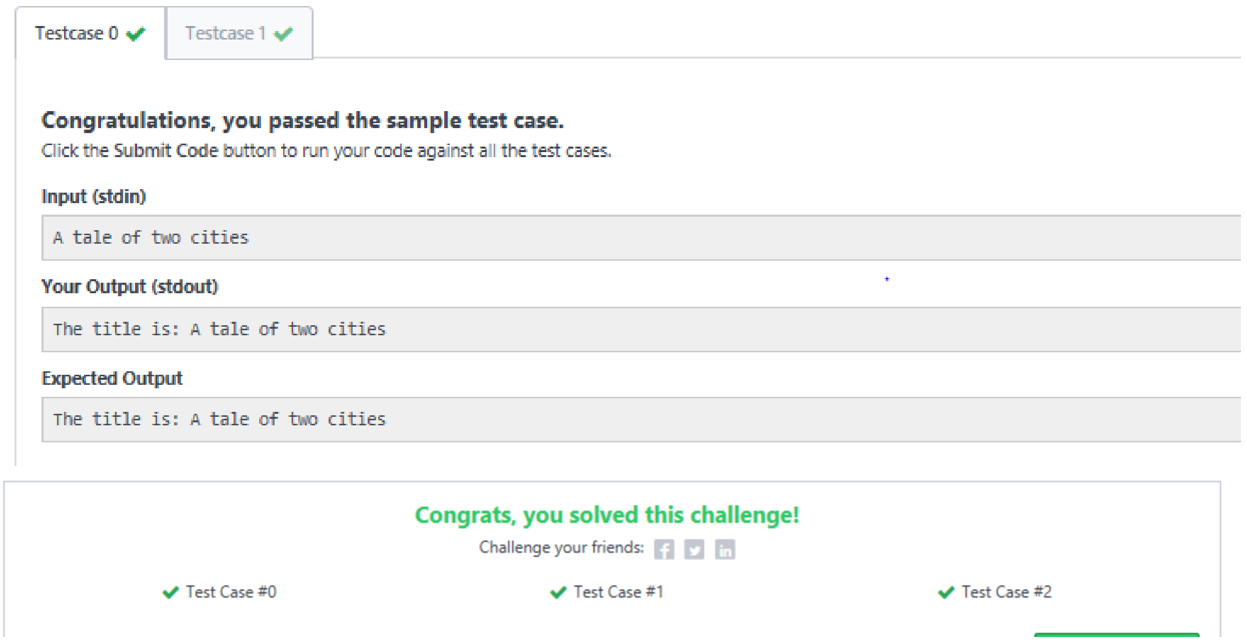
```
abstract class Book{
    String title;
    abstract void setTitle(String s);
    String getTitle(){
        return title;
    }
}
```

If you try to create an instance of this class like the following line you will get an error:

```
Book new_novel=new Book();
```

Program

```
import java.util.*;
abstract class Book{
        String title;
        abstract void setTitle(String s);
        String getTitle(){
                return title;
        }
}

class MyBook extends Book {
    @Override
    void setTitle(String s) {
        title = s;
    }
}public class Main{

        public static void main(String []args){
                //Book new_novel=new Book(); This line prHMain.java:25: error: Book is abstract;
cannot be instantiated
                Scanner sc=new Scanner(System.in);
                String title=sc.nextLine();
                MyBook new_novel=new MyBook();
                new_novel.setTitle(title);
                System.out.println("The title is: "+new_novel.getTitle());
        sc.close();

        }
}
```


Testcase 0 ✓ Testcase 1 ✓

Congratulations, you passed the sample test case.

Click the Submit Code button to run your code against all the test cases.

Input (stdin)

```
A tale of two cities
```

Your Output (stdout)

```
The title is: A tale of two cities
```

Expected Output

```
The title is: A tale of two cities
```

Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2

Q48

A Java interface can only contain method signatures and fields. The interface can be used to achieve polymorphism. In this problem, you will practice your knowledge on interfaces.

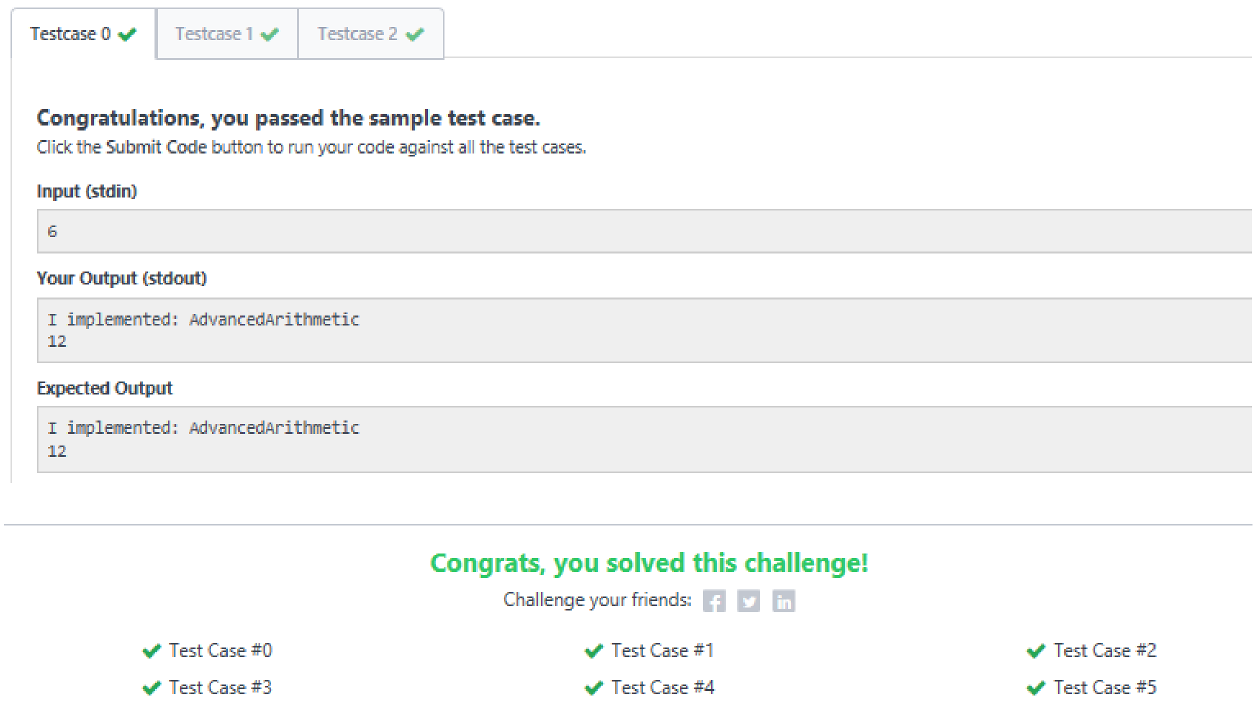
You are given an interface *AdvancedArithmetic* which contains a method signature *int divisor_sum(int n)*. You need to write a class called MyCalculator which implements the interface.

divisorSum function just takes an integer as input and return the sum of all its divisors. For example divisors of 6 are 1, 2, 3 and 6, so *divisor_sum* should return 12. The value of n will be at most 1000.

Read the partially completed code in the editor and complete it. You just need to write the MyCalculator class only. *Your class shouldn't be public.*

```
import java.util.*;
interface AdvancedArithmetic{
 int divisor_sum(int n);
}
class MyCalculator implements AdvancedArithmetic {
    int j = 0;
   public int divisor_sum(int n) {
        for(int i = 1; i <= n; i++) {
            if (n % i == 0) {
                j = j + i;
            }
        }
        return j;
    }


}class Solution{
    public static void main(String []args){
        MyCalculator my_calculator = new MyCalculator();
        System.out.print("I implemented: ");
        ImplementedInterfaceNames(my_calculator);
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        System.out.print(my_calculator.divisor_sum(n) + "\n");
             sc.close();
    }
    /*
     *  ImplementedInterfaceNames method takes an object and prints the name of the interfaces it
implemented
     */
    static void ImplementedInterfaceNames(Object o){
        Class[] theInterfaces = o.getClass().getInterfaces();
        for (int i = 0; i < theInterfaces.length; i++){
            String interfaceName = theInterfaces[i].getName();
            System.out.println(interfaceName);
        }
    }
}
```

Output


Testcase 0 ✓ | Testcase 1 ✓ | Testcase 2 ✓

Congratulations, you passed the sample test case.

Click the Submit Code button to run your code against all the test cases.

Input (stdin)

```
6
```

Your Output (stdout)

```
I implemented: AdvancedArithmetic
12
```

Expected Output

```
I implemented: AdvancedArithmetic
12
```

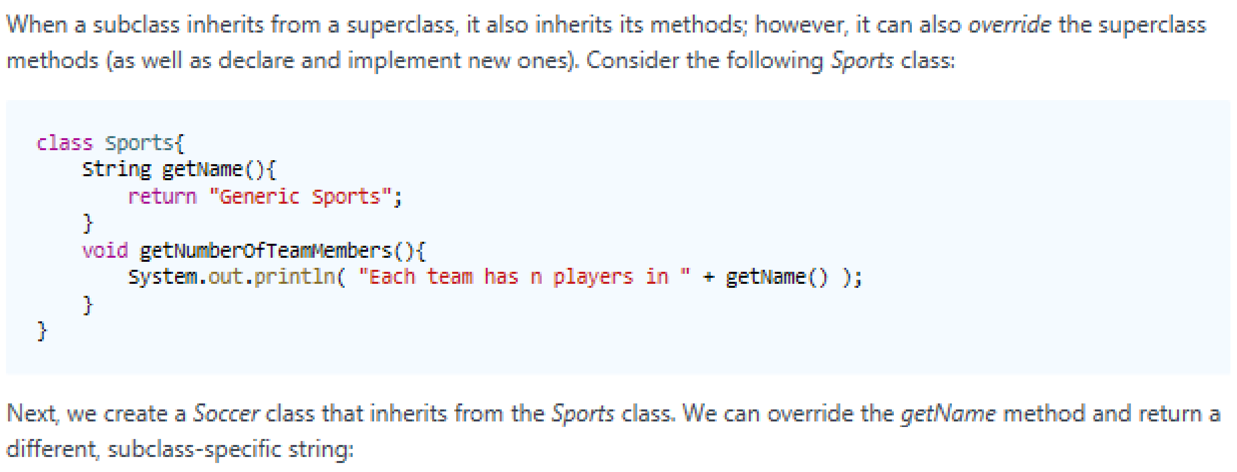
Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2

✓ Test Case #3 ✓ Test Case #4 ✓ Test Case #5

Q49


When a subclass inherits from a superclass, it also inherits its methods; however, it can also *override* the superclass methods (as well as declare and implement new ones). Consider the following *Sports* class:

```
class Sports{
    String getName(){
        return "Generic Sports";
    }
    void getNumberOfTeamMembers(){
        System.out.println( "Each team has n players in " + getName() );
    }
}
```

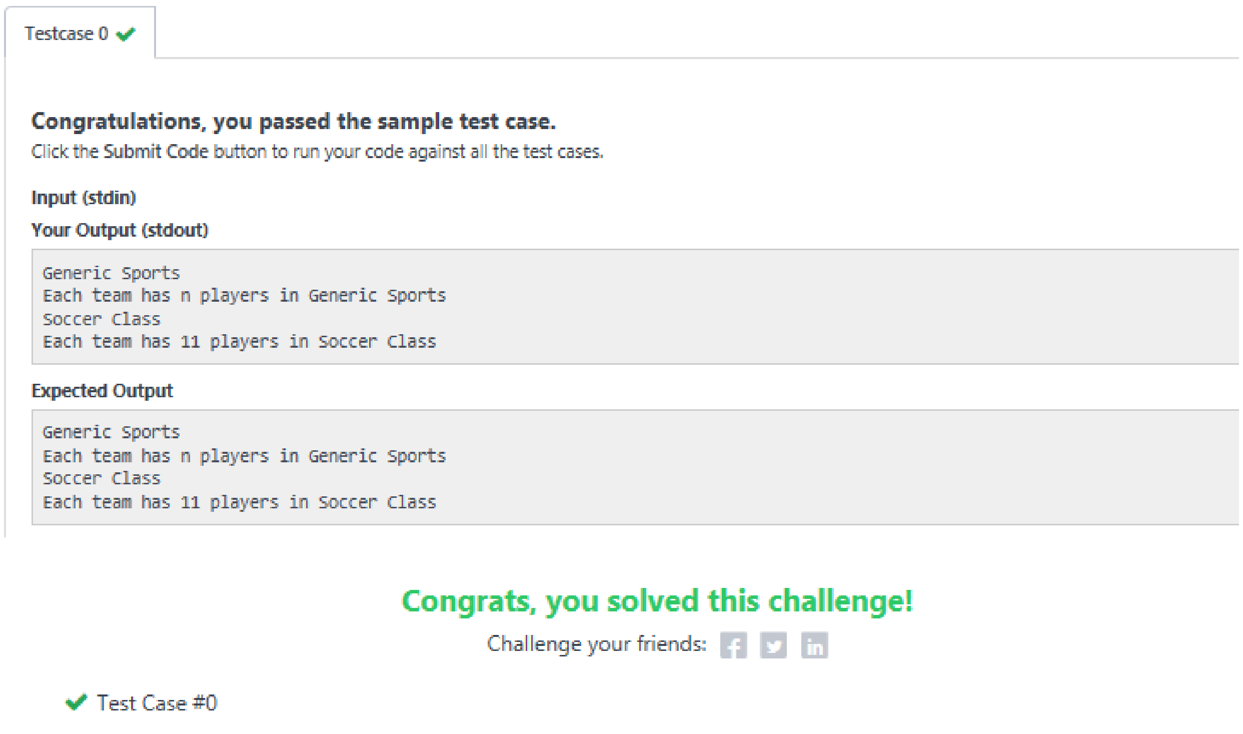
Next, we create a *Soccer* class that inherits from the *Sports* class. We can override the *getName* method and return a different, subclass-specific string:

Program

```
import java.util.*;
class Sports{

   String getName(){
      return "Generic Sports";
   }
```

```
    void getNumberOfTeamMembers(){
        System.out.println( "Each team has n players in " + getName() );
    }
}

class Soccer extends Sports{
    @Override
    String getName(){
        return "Soccer Class";
    }
    @Override
void getNumberOfTeamMembers()
{
        System.out.println( "Each team has 11 players in " + getName() );
}}

public class Solution{

    public static void main(String []args){
        Sports c1 = new Sports();
        Soccer c2 = new Soccer();
        System.out.println(c1.getName());
        c1.getNumberOfTeamMembers();
        System.out.println(c2.getName());
        c2.getNumberOfTeamMembers();
            }
}
```

Output



Congratulations, you passed the sample test case.

Click the Submit Code button to run your code against all the test cases.

Input (stdin)

Your Output (stdout)

```
Generic Sports
Each team has n players in Generic Sports
Soccer Class
Each team has 11 players in Soccer Class
```

Expected Output

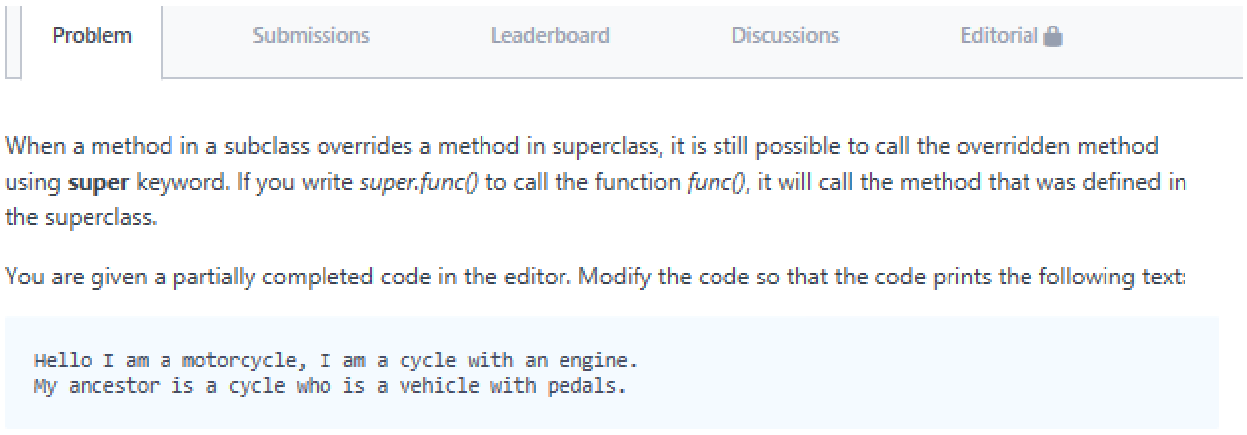
```
Generic Sports
Each team has n players in Generic Sports
Soccer Class
Each team has 11 players in Soccer Class
```

Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0

Q50



When a method in a subclass overrides a method in superclass, it is still possible to call the overridden method using **super** keyword. If you write *super.func()* to call the function *func()*, it will call the method that was defined in the superclass.

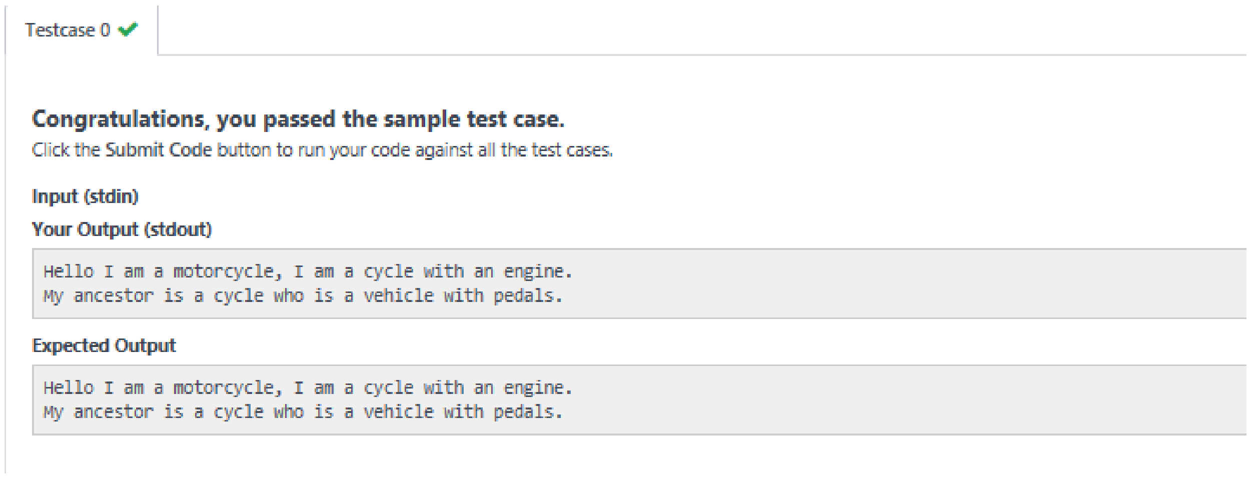
You are given a partially completed code in the editor. Modify the code so that the code prints the following text:

```
Hello I am a motorcycle, I am a cycle with an engine.
My ancestor is a cycle who is a vehicle with pedals.
```

```
import java.util.*;
import java.io.*;

class BiCycle{
        String define_me(){
                return "a vehicle with pedals.";
```

```
          }
}

class MotorCycle extends BiCycle{
          String define_me(){
                    return "a cycle with an engine.";
          }

          MotorCycle(){
                    System.out.println("Hello I am a motorcycle, I am "+ define_me());
String temp = super.define_me();                    System.out.println("My ancestor is a cycle who is "+
temp );
          }

}
class Solution{
          public static void main(String []args){
                    MotorCycle M=new MotorCycle();
          }
}
```

Output:



Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the Submit Code button to run your code against all the test cases.

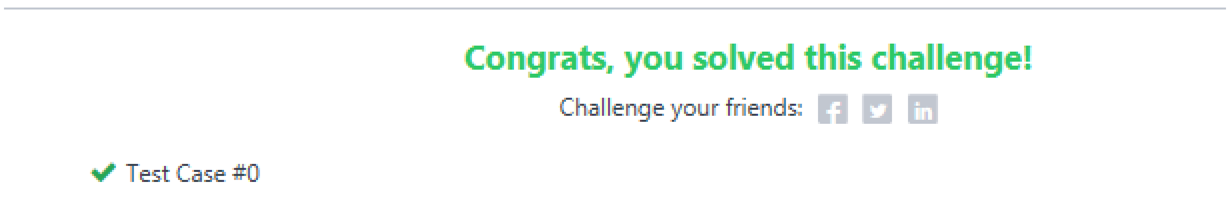
Input (stdin)

Your Output (stdout)

```
Hello I am a motorcycle, I am a cycle with an engine.
My ancestor is a cycle who is a vehicle with pedals.
```

Expected Output

```
Hello I am a motorcycle, I am a cycle with an engine.
My ancestor is a cycle who is a vehicle with pedals.
```



Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0

Q51

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 3 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.

Program

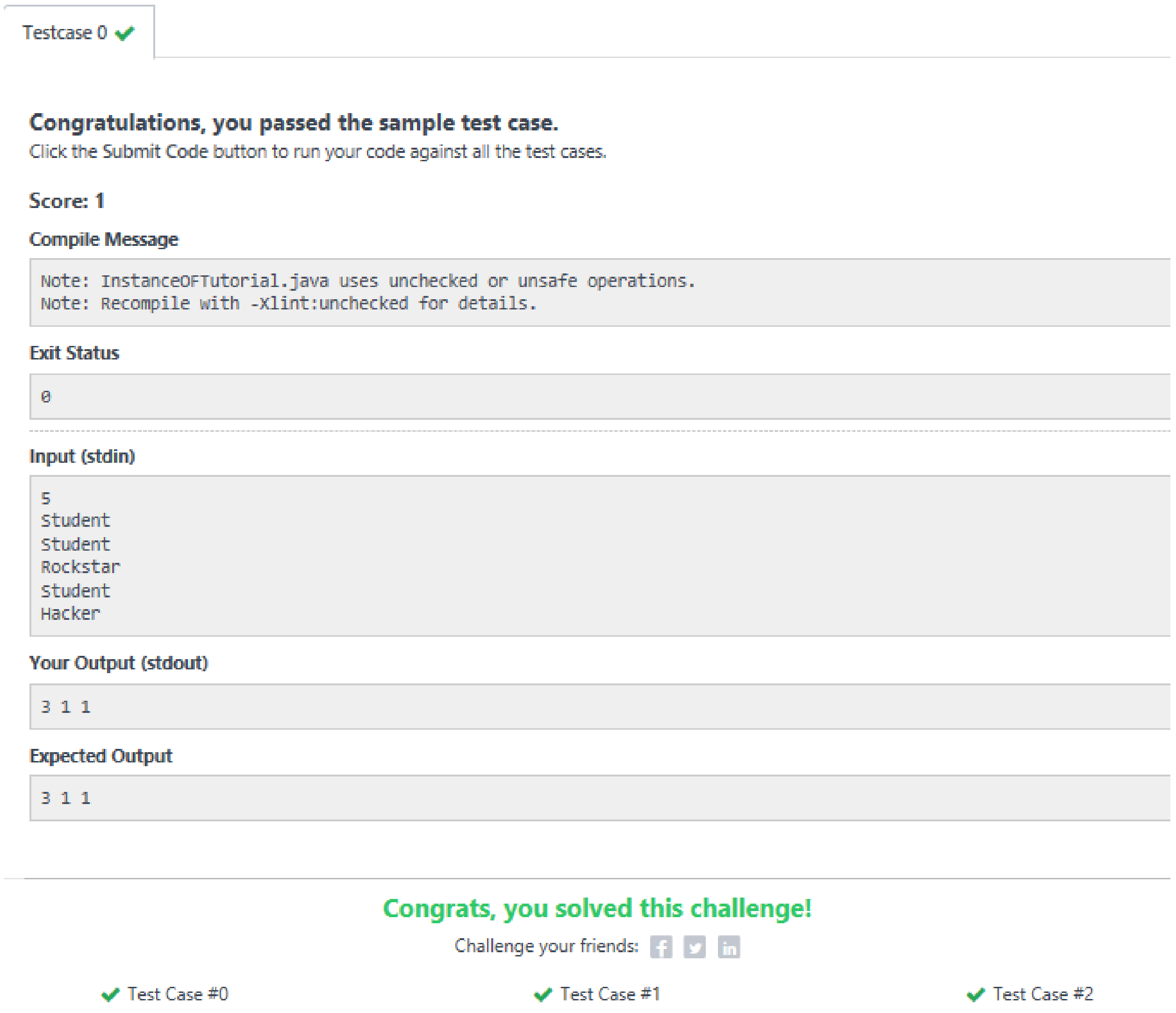
```
import java.util.*;




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));
```

```
    }


}
```

Output



Testcase 0 ✓

Congratulations, you passed the sample test case.
Click the Submit Code button to run your code against all the test cases.

Score: 1

Compile Message

```
Note: InstanceOFTutorial.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

Exit Status

```
0
```

Input (stdin)

```
5
Student
Student
Rockstar
Student
Hacker
```

Your Output (stdout)

```
3 1 1
```

Expected Output

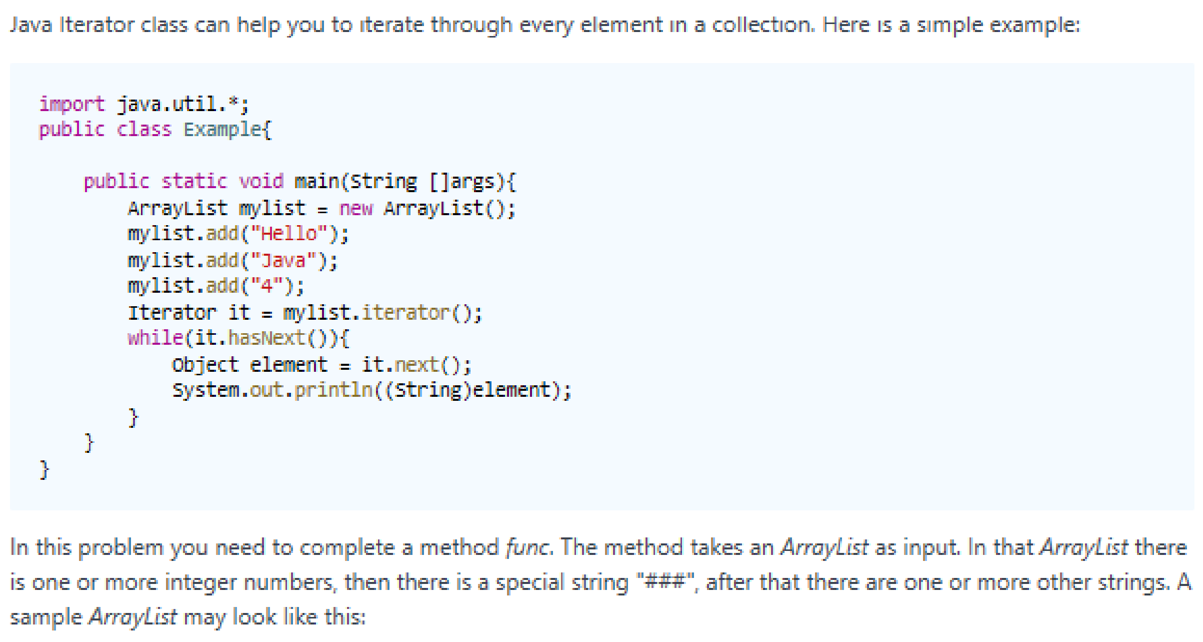
```
3 1 1
```

Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2

Q52



Java Iterator class can help you to iterate through every element in a collection. Here is a simple example:

```
import java.util.*;
public class Example{

    public static void main(String []args){
        ArrayList mylist = new ArrayList();
        mylist.add("Hello");
        mylist.add("Java");
        mylist.add("4");
        Iterator it = mylist.iterator();
        while(it.hasNext()){
            Object element = it.next();
            System.out.println((String)element);
        }
    }
}
```

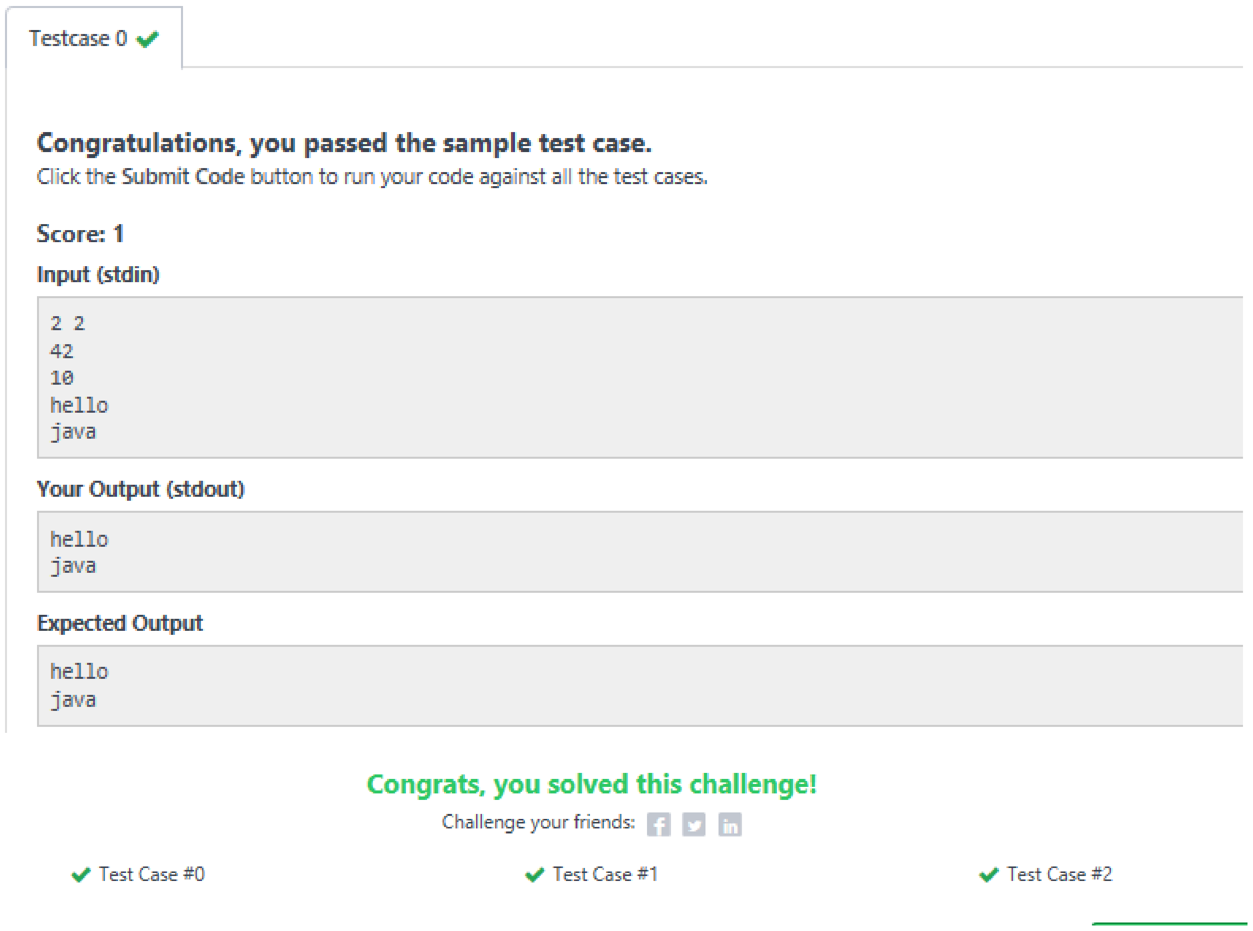
In this problem you need to complete a method *func*. The method takes an *ArrayList* as input. In that *ArrayList* there is one or more integer numbers, then there is a special string "###", after that there are one or more other strings. A sample *ArrayList* may look like this:

Program

```
import java.util.*;


public class Main{




  static Iterator func(ArrayList mylist){


    Iterator it=mylist.iterator();


    while(it.hasNext()){


      Object element = it.next();


      if (element.equals("###"))


        break;
    }

    return it;



}

@SuppressWarnings({ "unchecked" })

public static void main(String []args){

    ArrayList mylist = new ArrayList();

    Scanner sc = new Scanner(System.in);

    int n = sc.nextInt();

    int m = sc.nextInt();

    for(int i = 0;i<n;i++){

        mylist.add(sc.nextInt());

    }



    mylist.add("###");
```

```
  for(int i=0;i<m;i++){

    mylist.add(sc.next());

  }



  Iterator it=func(mylist);

  while(it.hasNext()){

    Object element = it.next();

    System.out.println((String)element);

  }

}
```

}



Testcase 0 ✓

Congratulations, you passed the sample test case.
Click the **Submit Code** button to run your code against all the test cases.

Score: 1

Input (stdin)

```
2 2
42
10
hello
java
```

Your Output (stdout)

```
hello
java
```

Expected Output

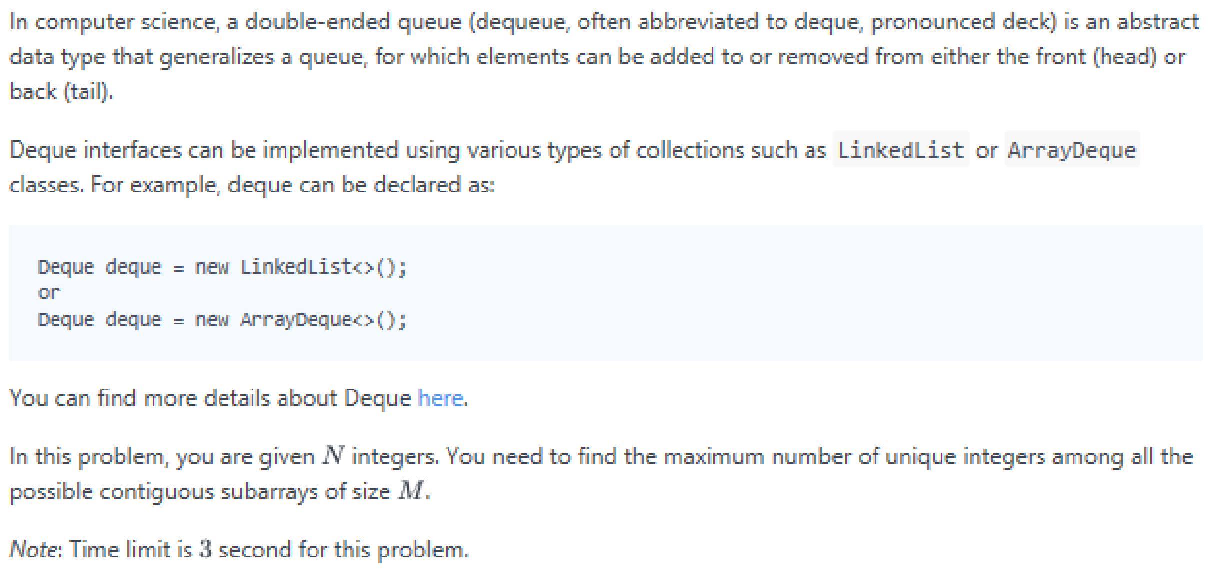
```
hello
java
```

Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2

Q53



In computer science, a double-ended queue (dequeue, often abbreviated to deque, pronounced deck) is an abstract data type that generalizes a queue, for which elements can be added to or removed from either the front (head) or back (tail).

Deque interfaces can be implemented using various types of collections such as `LinkedList` or `ArrayDeque` classes. For example, deque can be declared as:

```
Deque deque = new LinkedList<>();
or
Deque deque = new ArrayDeque<>();
```

You can find more details about Deque here.

In this problem, you are given N integers. You need to find the maximum number of unique integers among all the possible contiguous subarrays of size M.

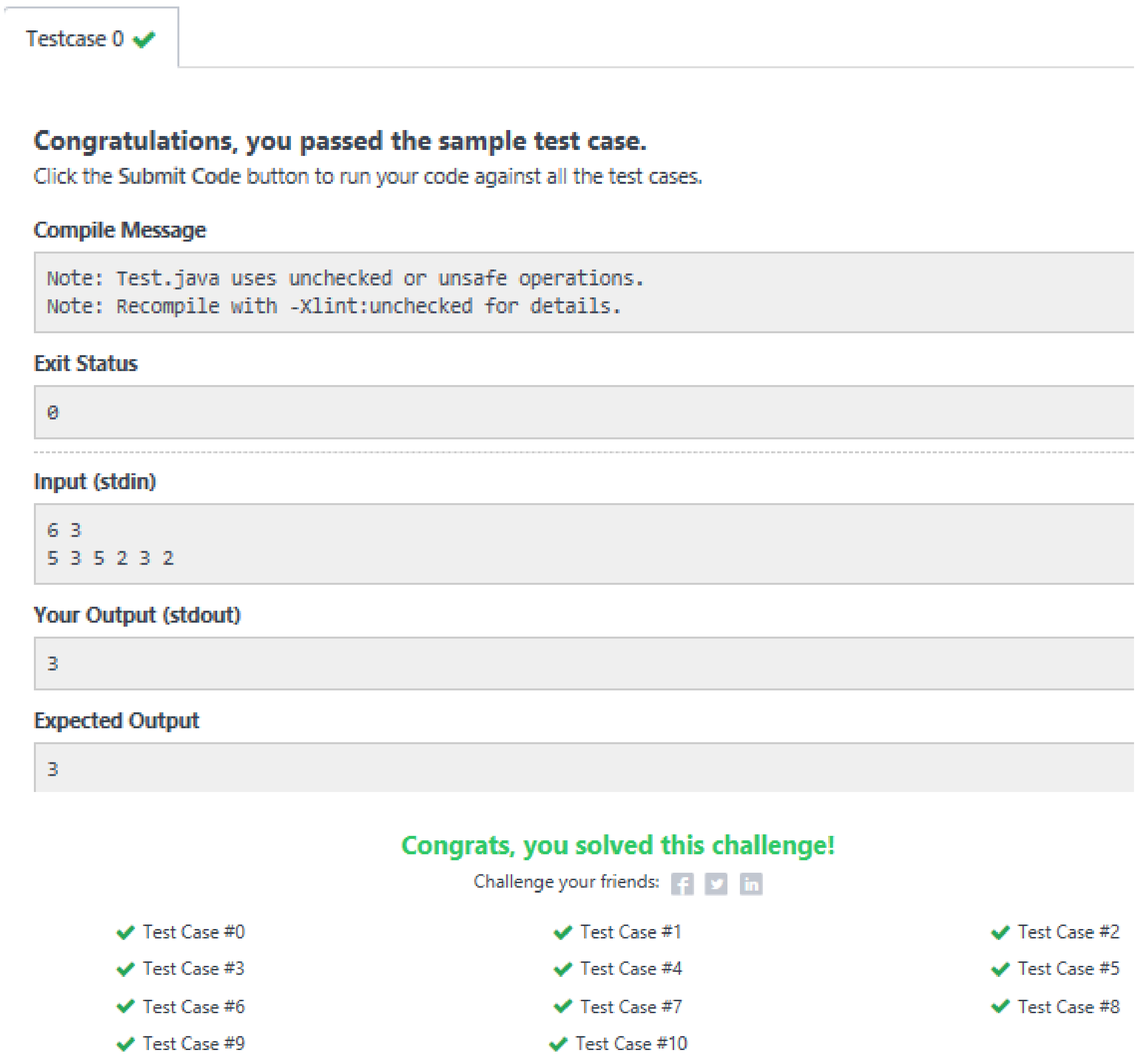
Note: Time limit is 3 second for this problem.

Program

```
import java.util.*;

public class Test {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        Deque deque = new ArrayDeque<>();
        int n = in.nextInt();
        int m = in.nextInt();
        int max = 0;
        HashSet hs = new HashSet();

        for (int i = 0; i < n; i++) {
            int num = in.nextInt();

            deque.add(num);
            hs.add(num);

            if (deque.size() == m + 1) {

                int q_out = (int) deque.remove();

                if (!deque.contains(q_out)) {
                    hs.remove(q_out);

                }

            }

            max = Math.max(hs.size(), max);
```

```
                }
                System.out.println(max);
        }
}
```

Output



Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the Submit Code button to run your code against all the test cases.

Compile Message

```
Note: Test.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

Exit Status

```
0
```

Input (stdin)

```
6 3
5 3 5 2 3 2
```

Your Output (stdout)

```
3
```

Expected Output

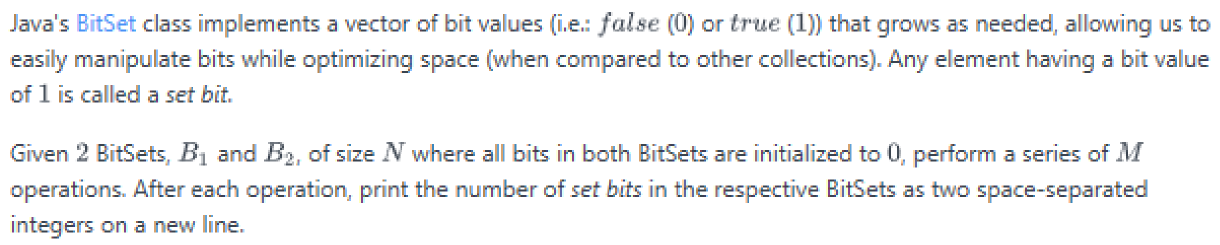
```
3
```

Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2
✓ Test Case #3 ✓ Test Case #4 ✓ Test Case #5
✓ Test Case #6 ✓ Test Case #7 ✓ Test Case #8
✓ Test Case #9 ✓ Test Case #10

Q54


Java's BitSet class implements a vector of bit values (i.e.: $false$ (0) or $true$ (1)) that grows as needed, allowing us to easily manipulate bits while optimizing space (when compared to other collections). Any element having a bit value of 1 is called a *set bit*.

Given 2 BitSets, B_1 and B_2, of size N where all bits in both BitSets are initialized to 0, perform a series of M operations. After each operation, print the number of *set bits* in the respective BitSets as two space-separated integers on a new line.

Program

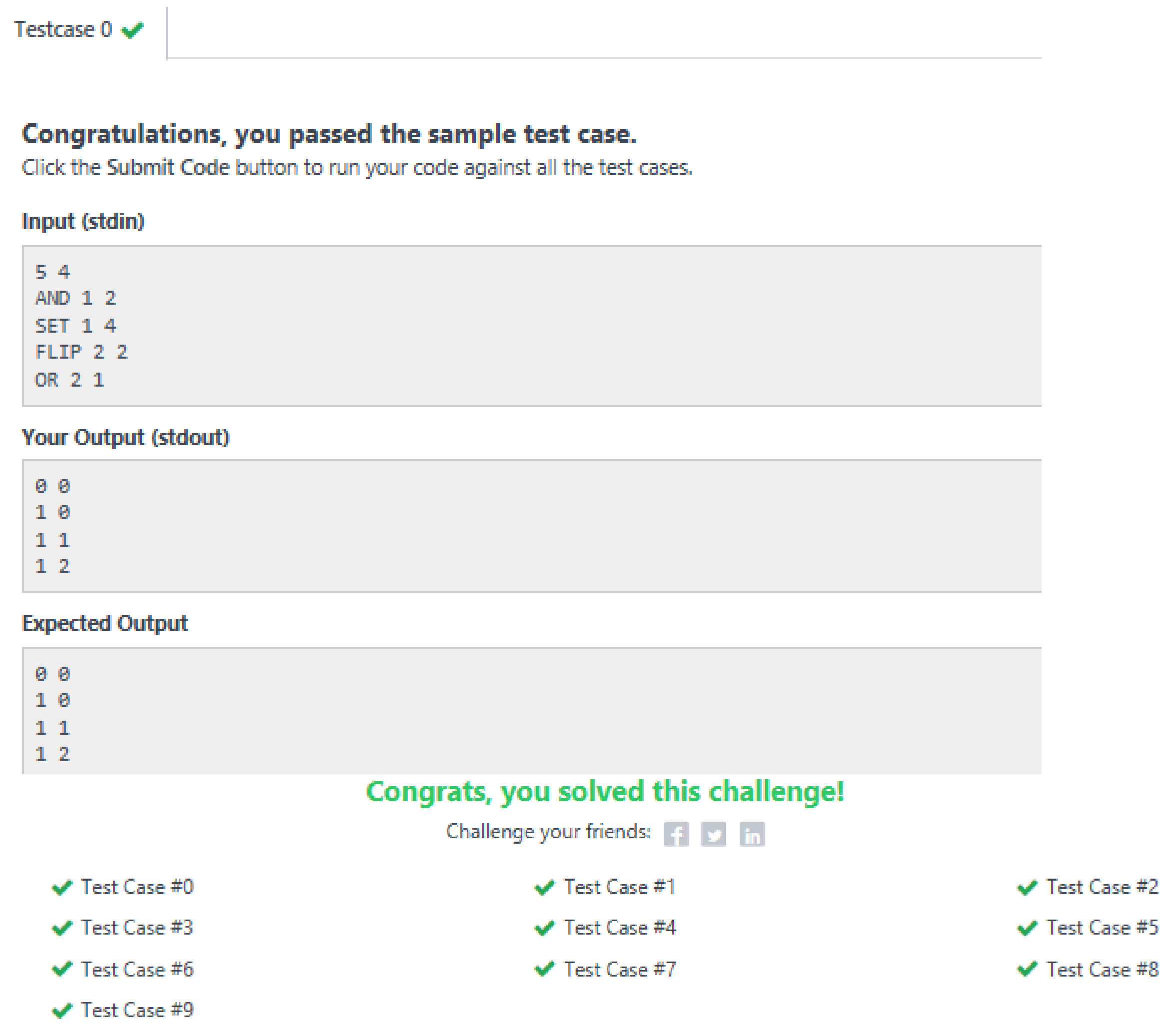
```
import java.util.Scanner;


import java.util.BitSet;




public class Solution {


    public static void main(String[] args) {


        Scanner scan = new Scanner(System.in);


        int N = scan.nextInt();


        int M = scan.nextInt();


        BitSet B1 = new BitSet(N);


        BitSet B2 = new BitSet(N);


        while (M-- > 0) {


            String str = scan.next();
```

```
int a      = scan.nextInt();

int b      = scan.nextInt();

switch (str) {

   case "AND":

      if (a == 1) {

         B1.and(B2);

      } else {

         B2.and(B1);

      }

      break;

   case "OR":

      if (a == 1) {

         B1.or(B2);

      } else {

         B2.or(B1);
```

```
    }
    break;
case "XOR":
    if (a == 1) {
        B1.xor(B2);
    } else {
        B2.xor(B1);
    }
    break;
case "FLIP":
    if (a == 1) {
        B1.flip(b);
    } else {
        B2.flip(b);
    }
```

```
                break;
            case "SET":
                if (a == 1) {
                    B1.set(b);
                } else {
                    B2.set(b);
                }
                break;
            default:
                break;
        }
        System.out.println(B1.cardinality() + " " + B2.cardinality());
    }
    scan.close();
}
```

}



Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

```
5 4
AND 1 2
SET 1 4
FLIP 2 2
OR 2 1
```

Your Output (stdout)

```
0 0
1 0
1 1
1 2
```

Expected Output

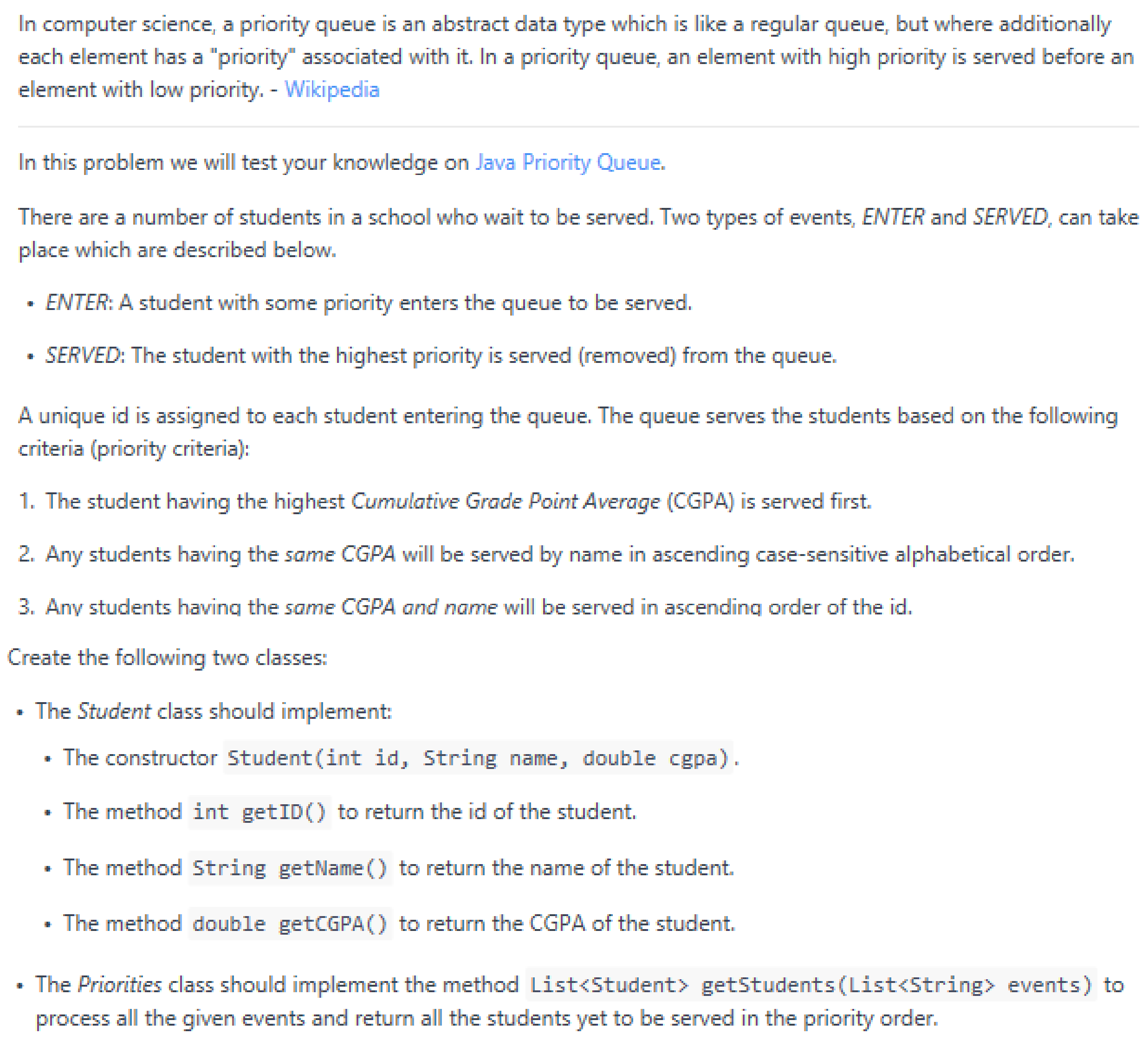
```
0 0
1 0
1 1
1 2
```

Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0	✓ Test Case #1	✓ Test Case #2
✓ Test Case #3	✓ Test Case #4	✓ Test Case #5
✓ Test Case #6	✓ Test Case #7	✓ Test Case #8
✓ Test Case #9		

Q55



In computer science, a priority queue is an abstract data type which is like a regular queue, but where additionally each element has a "priority" associated with it. In a priority queue, an element with high priority is served before an element with low priority. - Wikipedia

In this problem we will test your knowledge on Java Priority Queue.

There are a number of students in a school who wait to be served. Two types of events, *ENTER* and *SERVED*, can take place which are described below.

- *ENTER*: A student with some priority enters the queue to be served.
- *SERVED*: The student with the highest priority is served (removed) from the queue.

A unique id is assigned to each student entering the queue. The queue serves the students based on the following criteria (priority criteria):

1. The student having the highest *Cumulative Grade Point Average* (CGPA) is served first.
2. Any students having the *same CGPA* will be served by name in ascending case-sensitive alphabetical order.
3. Any students having the *same CGPA and name* will be served in ascending order of the id.

Create the following two classes:

- The *Student* class should implement:
  - The constructor `Student(int id, String name, double cgpa)`.
  - The method `int getID()` to return the id of the student.
  - The method `String getName()` to return the name of the student.
  - The method `double getCGPA()` to return the CGPA of the student.
- The *Priorities* class should implement the method `List<Student> getStudents(List<String> events)` to process all the given events and return all the students yet to be served in the priority order.

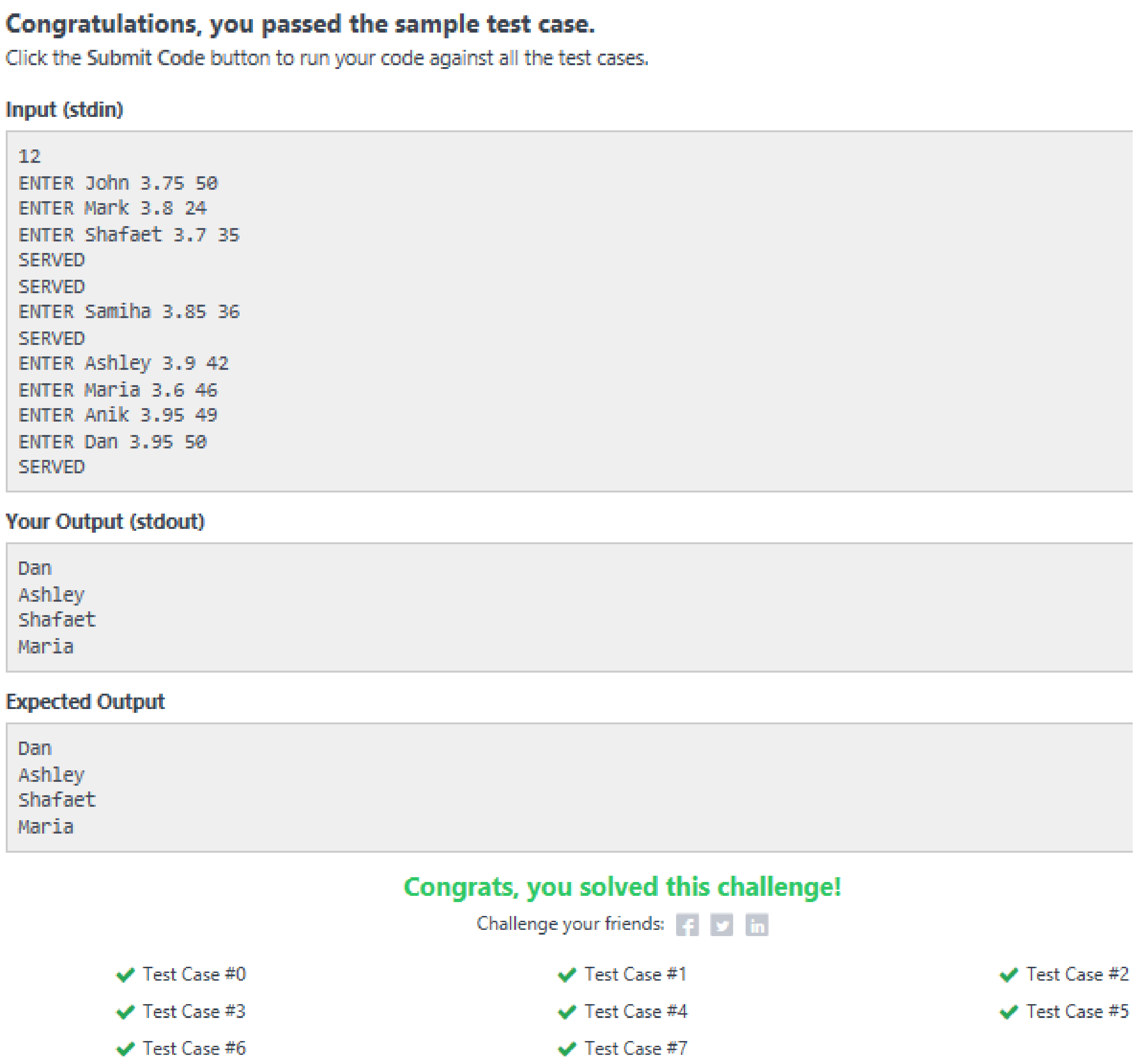
Program

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.util.*;
class Student implements Comparable<Student>{
   String name = new String();
   double cgpa;
   int id;
   public Student(String name,double cgpa,int id)
```

```
    {
        this.name = name;
        this.cgpa = cgpa;
        this.id = id;
    }
    public String getName(){
        return this.name;
    }
    public int compareTo(Student s)
    {
        if(cgpa == s.cgpa)
        {
            if(name.compareTo(s.name) == 0)
            {
                if(id == s.id)
                    return 0;
                else if (id > s.id)
                    return 1;
                else
                    return -1;
            }
            else
                return name.compareTo(s.name);
        }
        else if(cgpa > s.cgpa)
            return -1;
        else
            return 1;
    }
}
```

```
class Priorities{
    public ArrayList<Student> getStudents(List<String> events)
    {
        int n = events.size();
        PriorityQueue<Student> pq = new PriorityQueue<Student>();
        for(String i:events)
        {
            String[] s = new String[4];
            s = i.split("\\s");
            if(s.length>1)
            {
                pq.add(new Student(s[1],Double.valueOf(s[2]),Integer.valueOf(s[3])));
            }
            else
            {
                pq.poll();
            }
        }
        while(pq.size()>1)
        {
            System.out.println(pq.poll().name);
        }
        return new ArrayList<Student>(pq);
    }
}
public class Solution {
    private final static Scanner scan = new Scanner(System.in);
    private final static Priorities priorities = new Priorities();
    public static void main(String[] args) {
        int totalEvents = Integer.parseInt(scan.nextLine());
        List<String> events = new ArrayList<>();

        while (totalEvents-- != 0) {
            String event = scan.nextLine();
            events.add(event);
        }

        List<Student> students = priorities.getStudents(events);

        if (students.isEmpty()) {
            System.out.println("EMPTY");
        } else {
            for (Student st: students) {
                System.out.println(st.getName());
            }
        }
    }
}
```

Output


Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

```
12
ENTER John 3.75 50
ENTER Mark 3.8 24
ENTER Shafaet 3.7 35
SERVED
SERVED
ENTER Samiha 3.85 36
SERVED
ENTER Ashley 3.9 42
ENTER Maria 3.6 46
ENTER Anik 3.95 49
ENTER Dan 3.95 50
SERVED
```

Your Output (stdout)

```
Dan
Ashley
Shafaet
Maria
```

Expected Output

```
Dan
Ashley
Shafaet
Maria
```

Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2
✓ Test Case #3 ✓ Test Case #4 ✓ Test Case #5
✓ Test Case #6 ✓ Test Case #7

Q56

Note: In this problem you **must NOT** generate any output on your own. Any such solution will be considered as being against the rules and its author will be disqualified. The output of your solution must be generated by the uneditable code provided for you in the solution template.

An important concept in Object-Oriented Programming is the open/closed principle, which means writing code that is open to *extension* but closed to *modification*. In other words, new functionality should be added by writing an extension for the existing code rather than modifying it and potentially breaking other code that uses it. This challenge simulates a real-life problem where the open/closed principle can and should be applied.

A *Tree* class implementing a rooted tree is provided in the editor. It has the following publicly available methods:

- `getValue()`: Returns the *value* stored in the node.
- `getColor()`: Returns the *color* of the node.
- `getDepth()`: Returns the depth of the node. Recall that the depth of a node is the number of edges between the node and the tree's root, so the tree's root has depth 0 and each descendant node's depth is equal to the depth of its parent node $+1$.

In this challenge, we treat the internal implementation of the tree as being closed to modification, so we cannot directly modify it; however, as with real-world situations, the implementation is written in such a way that it allows external classes to extend and build upon its functionality. More specifically, it allows objects of the *TreeVis* class (a Visitor Design Pattern) to visit the tree and traverse the tree structure via the `accept` method.

There are two parts to this challenge.

Part I: Implement Three Different Visitors

Each class has three methods you must write implementations for:

1. `getResult()`: Return an integer denoting the $result$, which is different for each class:
   - The *SumInLeavesVisitor* implementation must return the sum of the values in the tree's *leaves* only.
   - The *ProductRedNodesVisitor* implementation must return the *product* of values stored in all *red* nodes, including leaves, computed modulo $10^9 + 7$. Note that the product of zero values is equal to 1.
   - The *FancyVisitor* implementation must return the absolute difference between the sum of values stored in the tree's *non-leaf nodes at even depth* and the sum of values stored in the tree's *green leaf nodes*. Recall that zero is an even number.
2. `visitNode(TreeNode node)`: Implement the logic responsible for visiting the tree's *non-leaf* nodes such that the *getResult* method returns the correct $result$ for the implementing class' visitor.
3. `visitLeaf(TreeLeaf leaf)`: Implement the logic responsible for visiting the tree's *leaf* nodes such that the *getResult* method returns the correct $result$ for the implementing class' visitor.

Part II: Read and Build the Tree

Read the n-node tree, where each node is numbered from 1 to n. The tree is given as a list of node values ($x_1, x_2, \ldots, x_n$), a list of node colors ($c_1, c_2, \ldots, c_n$), and a list of edges. Construct this tree as an instance of the *Tree* class. The tree is always rooted at node number 1.

Your implementations of the three visitor classes will be tested on the tree you built from the given input.

Program

```
import java.util.ArrayList;
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

import java.util.ArrayList;
import java.util.Scanner;

enum Color {
   RED, GREEN
}

abstract class Tree {

   private int value;
   private Color color;
   private int depth;

   public Tree(int value, Color color, int depth) {
      this.value = value;
      this.color = color;
      this.depth = depth;
   }

   public int getValue() {
      return value;
   }
```

```
    public Color getColor() {
        return color;
    }

    public int getDepth() {
        return depth;
    }

    public abstract void accept(TreeVis visitor);
}

class TreeNode extends Tree {

    private ArrayList<Tree> children = new ArrayList<>();

    public TreeNode(int value, Color color, int depth) {
        super(value, color, depth);
    }

    public void accept(TreeVis visitor) {
        visitor.visitNode(this);

        for (Tree child : children) {
            child.accept(visitor);
        }
    }

    public void addChild(Tree child) {
        children.add(child);
    }
}

class TreeLeaf extends Tree {

    public TreeLeaf(int value, Color color, int depth) {
        super(value, color, depth);
    }

    public void accept(TreeVis visitor) {
        visitor.visitLeaf(this);
    }
}

abstract class TreeVis
{
    public abstract int getResult();
    public abstract void visitNode(TreeNode node);
    public abstract void visitLeaf(TreeLeaf leaf);

}// Useful tutorial for Visitor Pattern:

// https://www.tutorialspoint.com/design_pattern/visitor_pattern.htm

//

// I recommend skipping this problem since it's more about creating a

// tree in an obscure format than it is about Visitor patterns.
```

```
class SumInLeavesVisitor extends TreeVis {

    private int result = 0;



    public int getResult() {

        return result;

    }



    public void visitNode(TreeNode node) {

        // do nothing

    }



    public void visitLeaf(TreeLeaf leaf) {

        result += leaf.getValue();

    }
```

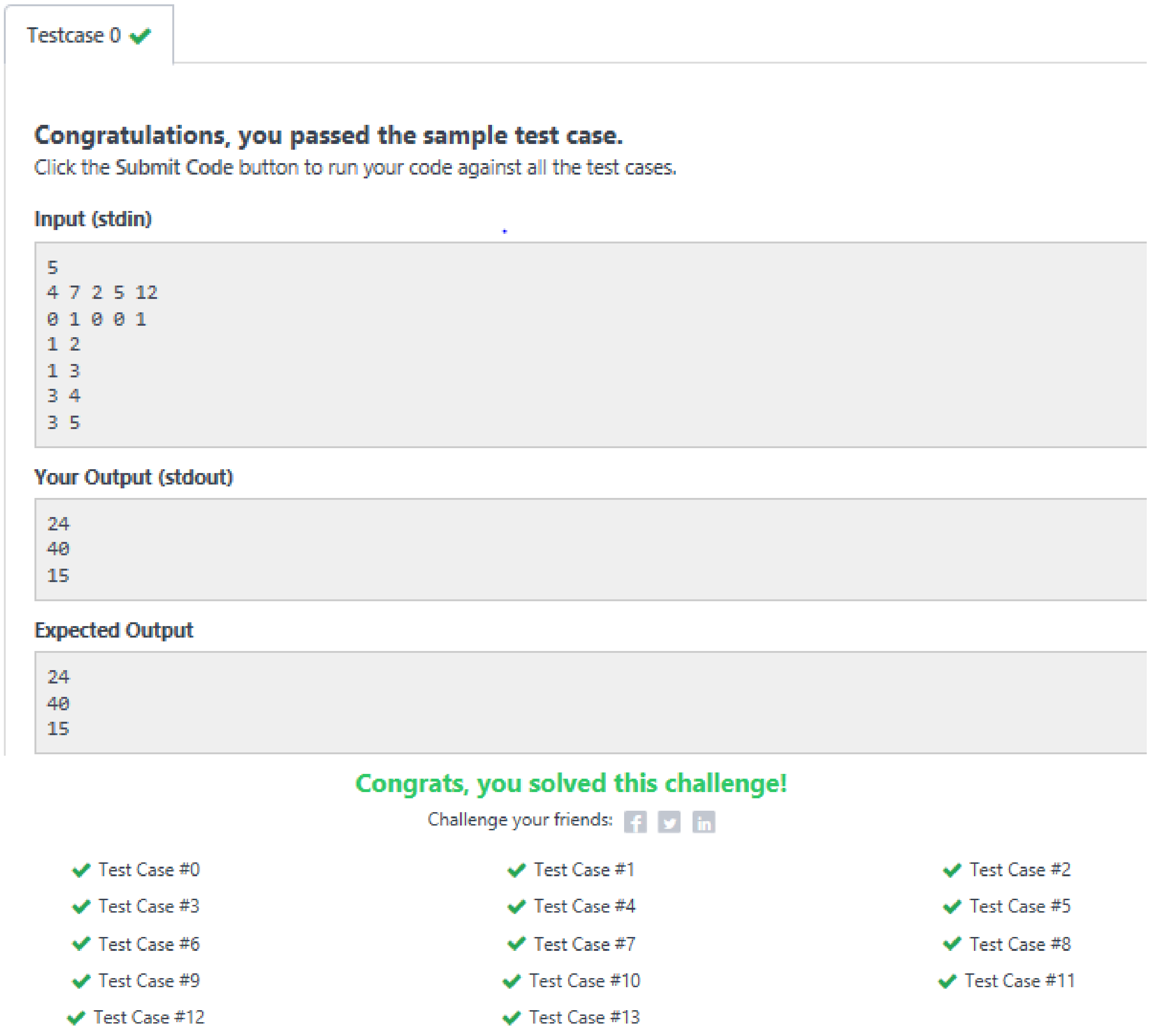
```
}


class ProductOfRedNodesVisitor extends TreeVis {

    private long result = 1;

    private final int M = 1000000007;


    public int getResult() {

        return (int) result;

    }


    public void visitNode(TreeNode node) {

        if (node.getColor() == Color.RED) {

            result = (result * node.getValue()) % M;

        }

    }
    public void visitLeaf(TreeLeaf leaf) {

        if (leaf.getColor() == Color.RED) {

            result = (result * leaf.getValue()) % M;

        }

    }

}



class FancyVisitor extends TreeVis {

    private int nonLeafEvenDepthSum = 0;

    private int greenLeavesSum = 0;



    public int getResult() {

        return Math.abs(nonLeafEvenDepthSum - greenLeavesSum);

    }
    public void visitNode(TreeNode node) {

        if (node.getDepth() % 2 == 0) {

            nonLeafEvenDepthSum += node.getValue();

        }

    }



    public void visitLeaf(TreeLeaf leaf) {

        if (leaf.getColor() == Color.GREEN) {

            greenLeavesSum += leaf.getValue();

        }

    }

}



public class Solution {
```

```
private static int [] values;

private static Color [] colors;

private static HashMap<Integer, HashSet<Integer>> map;



public static Tree solve() {

    Scanner scan = new Scanner(System.in);

    int numNodes = scan.nextInt();



    /* Save nodes and colors */

    values = new int[numNodes];

    colors = new Color[numNodes];

    map = new HashMap<>(numNodes);

    for (int i = 0; i < numNodes; i++) {

        values[i] = scan.nextInt();

    }
```

```
for (int i = 0; i < numNodes; i++) {

    colors[i] = scan.nextInt() == 0 ? Color.RED : Color.GREEN;

}



/* Save edges */

for (int i = 0; i < numNodes - 1; i++) {

    int u = scan.nextInt();

    int v = scan.nextInt();



    /* Edges are undirected: Add 1st direction */

    HashSet<Integer> uNeighbors = map.get(u);

    if (uNeighbors == null) {

        uNeighbors = new HashSet<>();

        map.put(u, uNeighbors);

    }
```

```
        uNeighbors.add(v);


        /* Edges are undirected: Add 2nd direction */
        HashSet<Integer> vNeighbors = map.get(v);
        if (vNeighbors == null) {
            vNeighbors = new HashSet<>();
            map.put(v, vNeighbors);
        }
        vNeighbors.add(u);
    }
    scan.close();


    /* Handle 1-node tree */
    if (numNodes == 1) {
        return new TreeLeaf(values[0], colors[0], 0);
        }



        /* Create Tree */

        TreeNode root = new TreeNode(values[0], colors[0], 0);

        addChildren(root, 1);

        return root;

    }



    /* Recursively adds children of a TreeNode */

    private static void addChildren(TreeNode parent, Integer parentNum) {

        /* Get HashSet of children and loop through them */

        for (Integer treeNum : map.get(parentNum)) {

            map.get(treeNum).remove(parentNum); // removes the opposite arrow direction



            /* Add child */
        HashSet<Integer> grandChildren = map.get(treeNum);

        boolean childHasChild = (grandChildren != null && !grandChildren.isEmpty());

        Tree tree;

        if (childHasChild) {

            tree = new TreeNode(values[treeNum - 1], colors[treeNum - 1], parent.getDepth() + 1);

        } else {

            tree = new TreeLeaf(values[treeNum - 1], colors[treeNum - 1], parent.getDepth() + 1);

        }

        parent.addChild(tree);



        /* Recurse if necessary */

        if (tree instanceof TreeNode) {

            addChildren((TreeNode) tree, treeNum);

        }

    }
    }


    public static void main(String[] args) {
        Tree root = solve();
                SumInLeavesVisitor vis1 = new SumInLeavesVisitor();
        ProductOfRedNodesVisitor vis2 = new ProductOfRedNodesVisitor();
        FancyVisitor vis3 = new FancyVisitor();

        root.accept(vis1);
        root.accept(vis2);
        root.accept(vis3);

        int res1 = vis1.getResult();
        int res2 = vis2.getResult();
        int res3 = vis3.getResult();

        System.out.println(res1);
        System.out.println(res2);
        System.out.println(res3);
        }
}
```

Output


Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

```
5
4 7 2 5 12
0 1 0 0 1
1 2
1 3
3 4
3 5
```

Your Output (stdout)

```
24
40
15
```

Expected Output

```
24
40
15
```

Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0	✓ Test Case #1	✓ Test Case #2
✓ Test Case #3	✓ Test Case #4	✓ Test Case #5
✓ Test Case #6	✓ Test Case #7	✓ Test Case #8
✓ Test Case #9	✓ Test Case #10	✓ Test Case #11
✓ Test Case #12	✓ Test Case #13	

Q57

You are given a list of student information: ID, FirstName, and CGPA. Your task is to rearrange them according to their CGPA in decreasing order. If two student have the same CGPA, then arrange them according to their first name in alphabetical order. If those two students also have the same first name, then order them according to their ID. No two students have the same ID.

Hint: You can use comparators to sort a list of objects. See the oracle docs to learn about comparators.

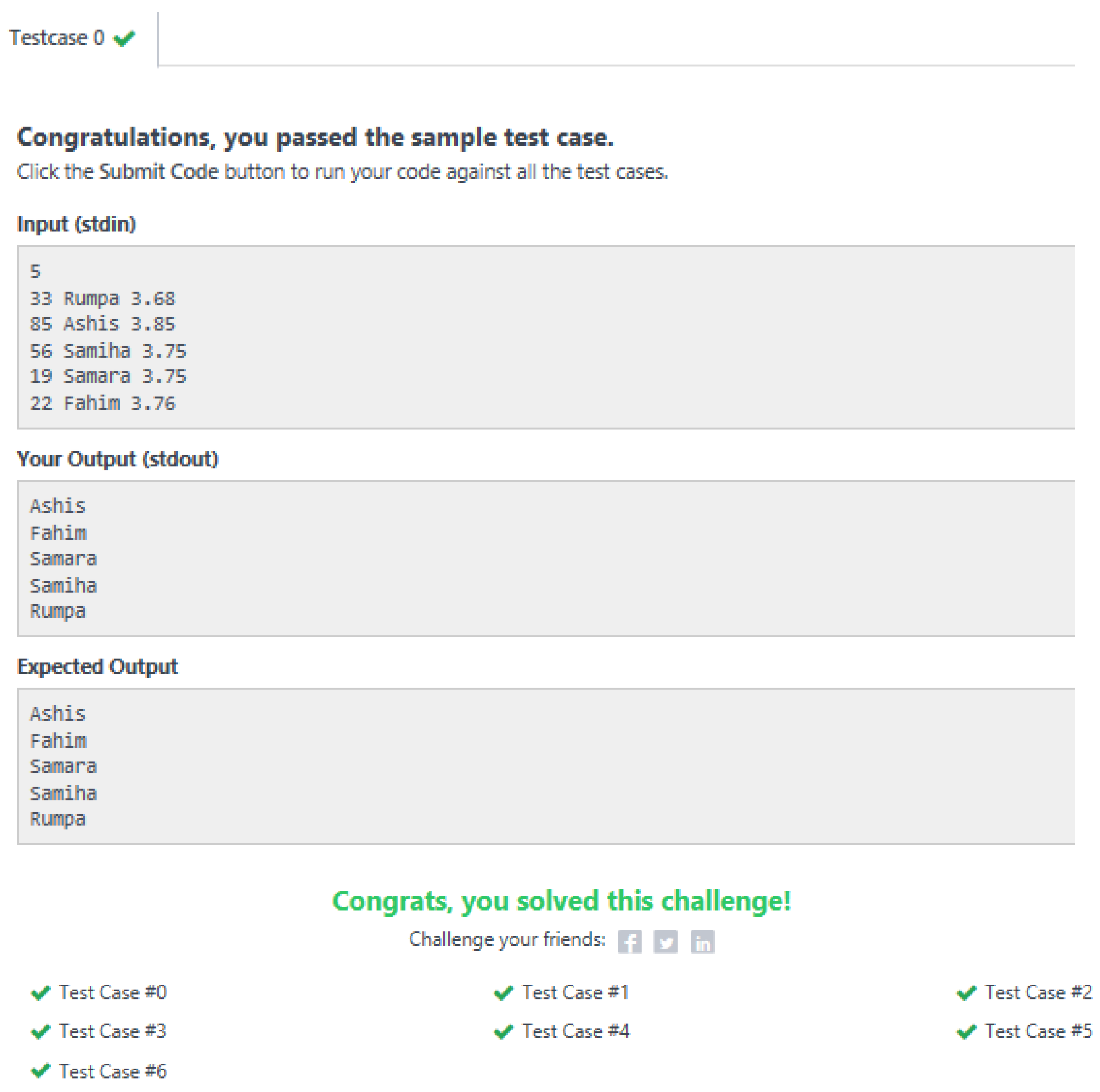
Program

import java.util.Scanner;

import java.util.List;

```
import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;



class Student {

   private int    id;

   private String fname;

   private double cgpa;

   public Student(int id, String fname, double cgpa) {

      super();

      this.id    = id;

      this.fname = fname;

      this.cgpa  = cgpa;

   }

   public int getId() {
        return id;

    }

    public String getFname() {

        return fname;

    }

    public double getCgpa() {

        return cgpa;

    }

}



public class Solution {

    public static void main(String[] args) {

        Scanner scan  = new Scanner(System.in);

        int testCases = Integer.parseInt(scan.nextLine());
        List<Student> studentList = new ArrayList<Student>();

        while (testCases-- > 0) {

            int id       = scan.nextInt();

            String fname = scan.next();

            double cgpa  = scan.nextDouble();

            Student st   = new Student(id, fname, cgpa);

            studentList.add(st);

        }

        scan.close();



        Collections.sort(studentList, new StudentComparator());

        for (Student st: studentList) {

            System.out.println(st.getFname());

        }

    }
```

```
}



class StudentComparator implements Comparator<Student> {

    double epsilon = 0.001; // since we shouldn't use "==" with doubles

    @Override

    public int compare(Student s1, Student s2) {

        if (Math.abs(s1.getCgpa() - s2.getCgpa()) > epsilon) {

            return s1.getCgpa() < s2.getCgpa() ? 1 : -1; // descending order

        } else if (!s1.getFname().equals(s2.getFname())) {

            return s1.getFname().compareTo(s2.getFname());

        } else {

            return s1.getId() - s2.getId();

        }

    }

}
```
Output



Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

```
5
33 Rumpa 3.68
85 Ashis 3.85
56 Samiha 3.75
19 Samara 3.75
22 Fahim 3.76
```

Your Output (stdout)

```
Ashis
Fahim
Samara
Samiha
Rumpa
```

Expected Output

```
Ashis
Fahim
Samara
Samiha
Rumpa
```

Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2
✓ Test Case #3 ✓ Test Case #4 ✓ Test Case #5
✓ Test Case #6

Q58

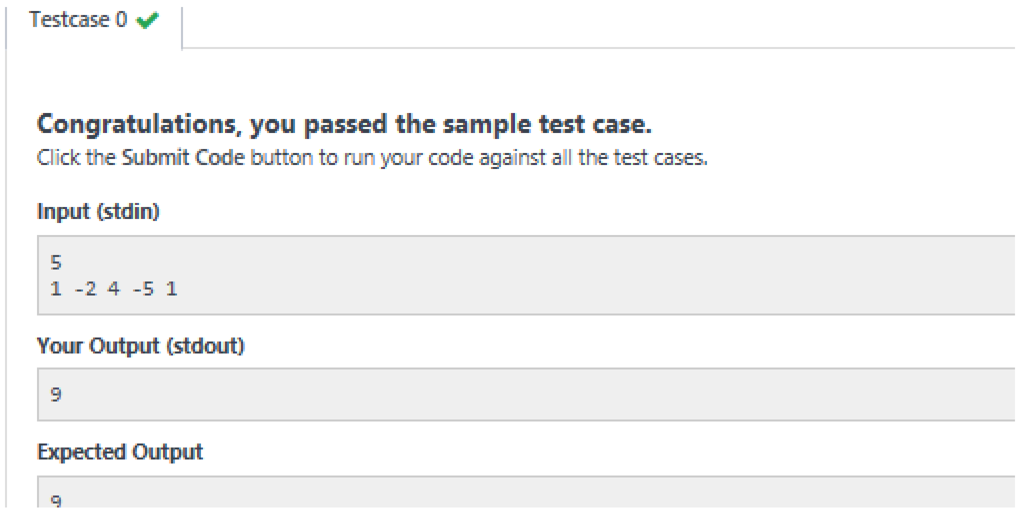
We define the following:

- A *subarray* of an n-element array is an array composed from a contiguous block of the original array's elements. For example, if $array = [1, 2, 3]$, then the subarrays are $[1]$, $[2]$, $[3]$, $[1, 2]$, $[2, 3]$, and $[1, 2, 3]$. Something like $[1, 3]$ would *not* be a subarray as it's not a contiguous subsection of the original array.
- The *sum* of an array is the total sum of its elements.
  - An array's sum is *negative* if the total sum of its elements is negative.
  - An array's sum is *positive* if the total sum of its elements is positive.

Given an array of n integers, find and print its number of *negative subarrays* on a new line.

Program

```
import java.util.Scanner;



// A subarray must be contiguous. There are O(n^2) contiguous subarrays.



//  Time Complexity: O(n^2)

// Space Complexity: O(1)

public class Solution {

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        int size     = scan.nextInt();

        int[] array = new int[size];

        for (int i = 0; i < size; i++) {

            array[i] = scan.nextInt();

        }

        scan.close();
        System.out.println(negativeSubarrays(array));

    }



    private static int negativeSubarrays(int[] array) {

        int count = 0;

        for (int i = 0; i < array.length; i++) {

            int sum = 0;

            for (int j = i; j < array.length; j++) {

                sum += array[j];

                if (sum < 0) {

                    count++;

                }

            }

        }
        return count;

    }

}
```



Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

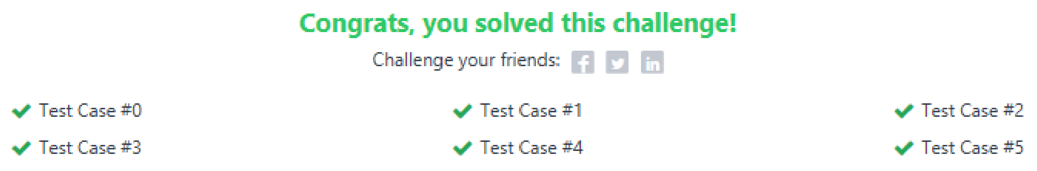
```
5
1 -2 4 -5 1
```

Your Output (stdout)

```
9
```

Expected Output

```
9
```



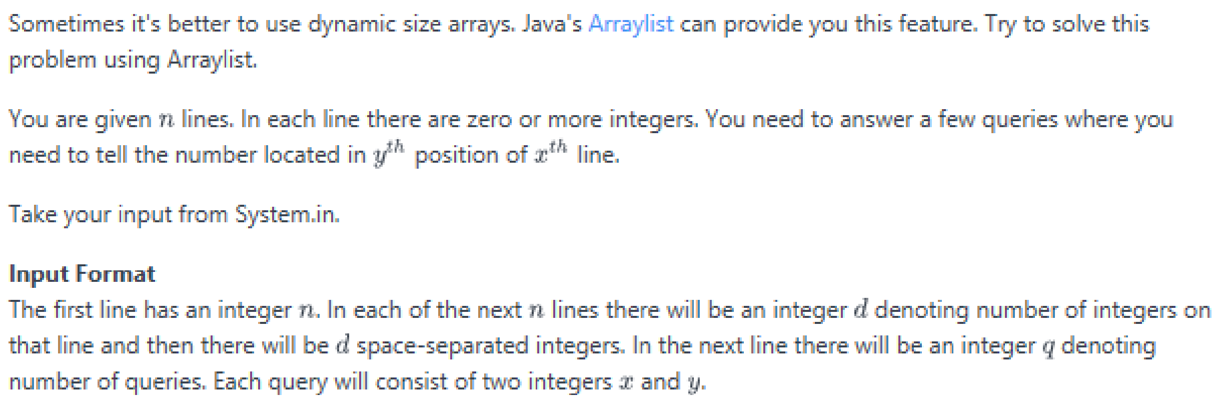
Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2

✓ Test Case #3 ✓ Test Case #4 ✓ Test Case #5

Q59



Sometimes it's better to use dynamic size arrays. Java's Arraylist can provide you this feature. Try to solve this problem using Arraylist.

You are given n lines. In each line there are zero or more integers. You need to answer a few queries where you need to tell the number located in y^{th} position of x^{th} line.

Take your input from System.in.

Input Format

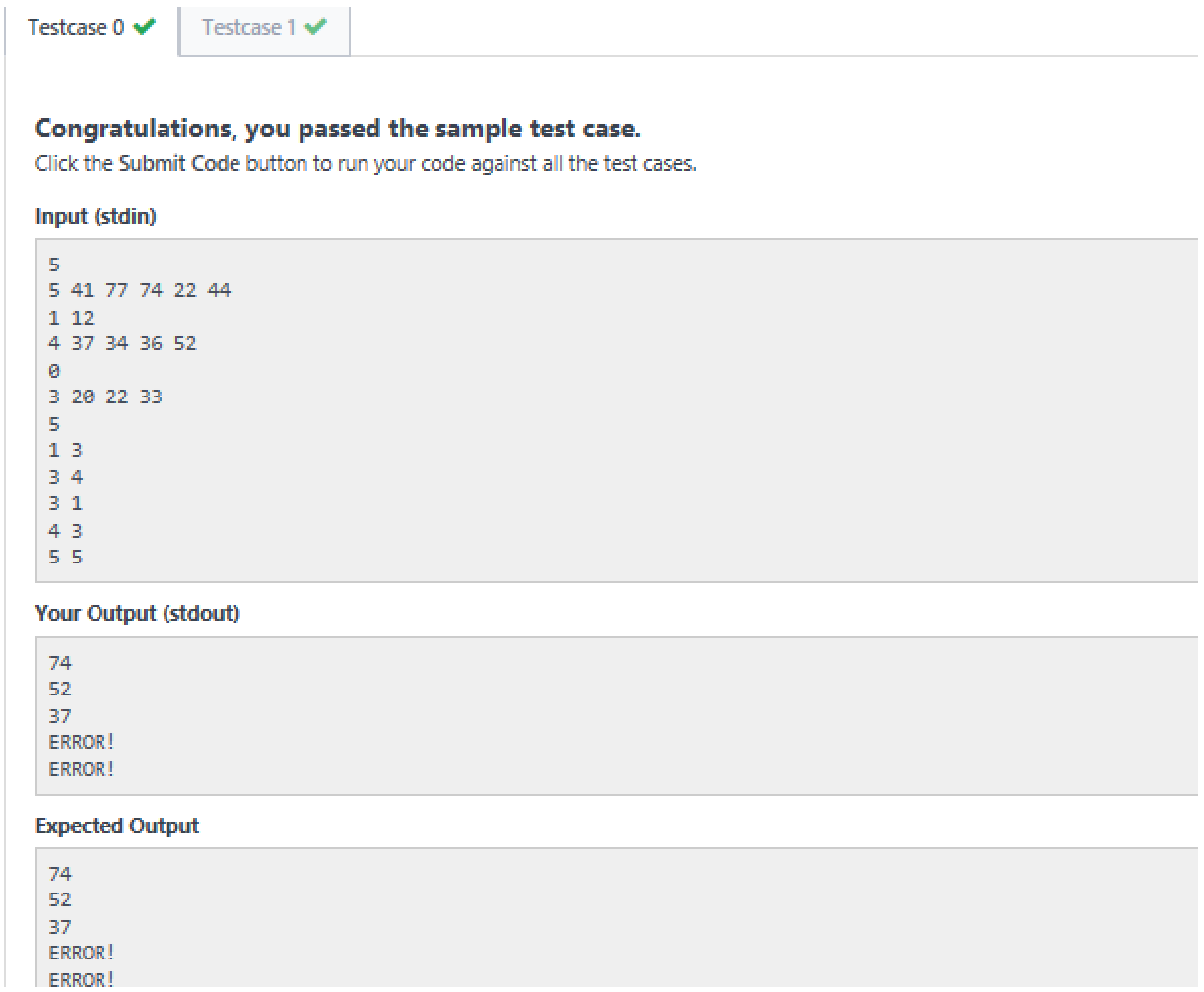
The first line has an integer n. In each of the next n lines there will be an integer d denoting number of integers on that line and then there will be d space-separated integers. In the next line there will be an integer q denoting number of queries. Each query will consist of two integers x and y.

Program

import java.util.Scanner;

```
import java.util.ArrayList;


public class Solution {

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        int n = scan.nextInt();


        /* Save numbers in 2-D ArrayList */

        ArrayList<ArrayList<Integer>> lists = new ArrayList<>();

        for (int row = 0; row < n; row++) {

            int d = scan.nextInt();

            ArrayList<Integer> list = new ArrayList<>();

            for (int col = 0; col < d; col++) {

                list.add(scan.nextInt());

            }
            lists.add(list);
        }

        /* Answer the queries */
        int q = scan.nextInt();
        for (int i = 0; i < q; i++) {
            int x = scan.nextInt();
            int y = scan.nextInt();
            ArrayList<Integer> list = lists.get(x-1);
            if (y <= list.size()) {
                System.out.println(list.get(y-1));
            } else {
                System.out.println("ERROR!");
            }
        }
```

```
        scan.close();

    }

}
```



Testcase 0 ✓ Testcase 1 ✓

Congratulations, you passed the sample test case.

Click the Submit Code button to run your code against all the test cases.

Input (stdin)

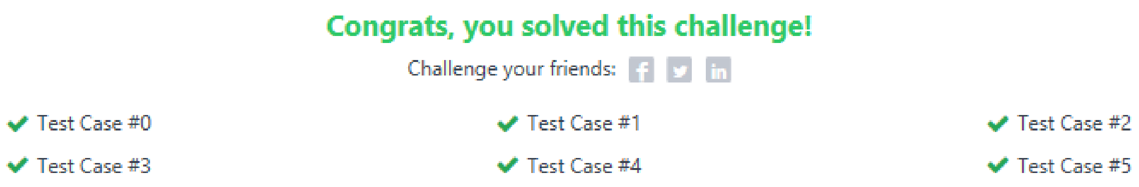
```
5
5 41 77 74 22 44
1 12
4 37 34 36 52
0
3 20 22 33
5
1 3
3 4
3 1
4 3
5 5
```

Your Output (stdout)

```
74
52
37
ERROR!
ERROR!
```

Expected Output

```
74
52
37
ERROR!
ERROR!
```



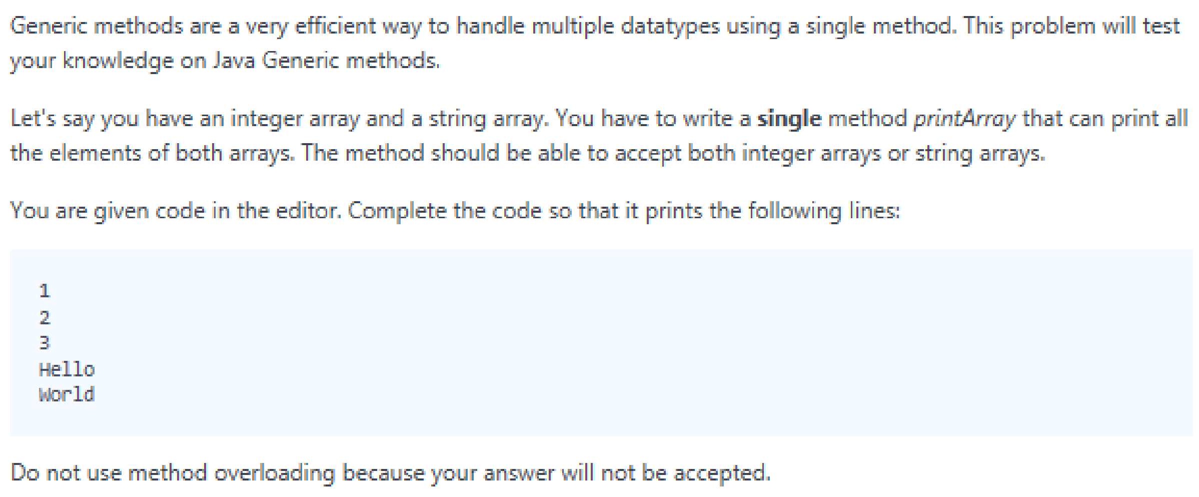
Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2

✓ Test Case #3 ✓ Test Case #4 ✓ Test Case #5

Q60



Generic methods are a very efficient way to handle multiple datatypes using a single method. This problem will test your knowledge on Java Generic methods.

Let's say you have an integer array and a string array. You have to write a **single** method *printArray* that can print all the elements of both arrays. The method should be able to accept both integer arrays or string arrays.

You are given code in the editor. Complete the code so that it prints the following lines:

```
1
2
3
Hello
World
```

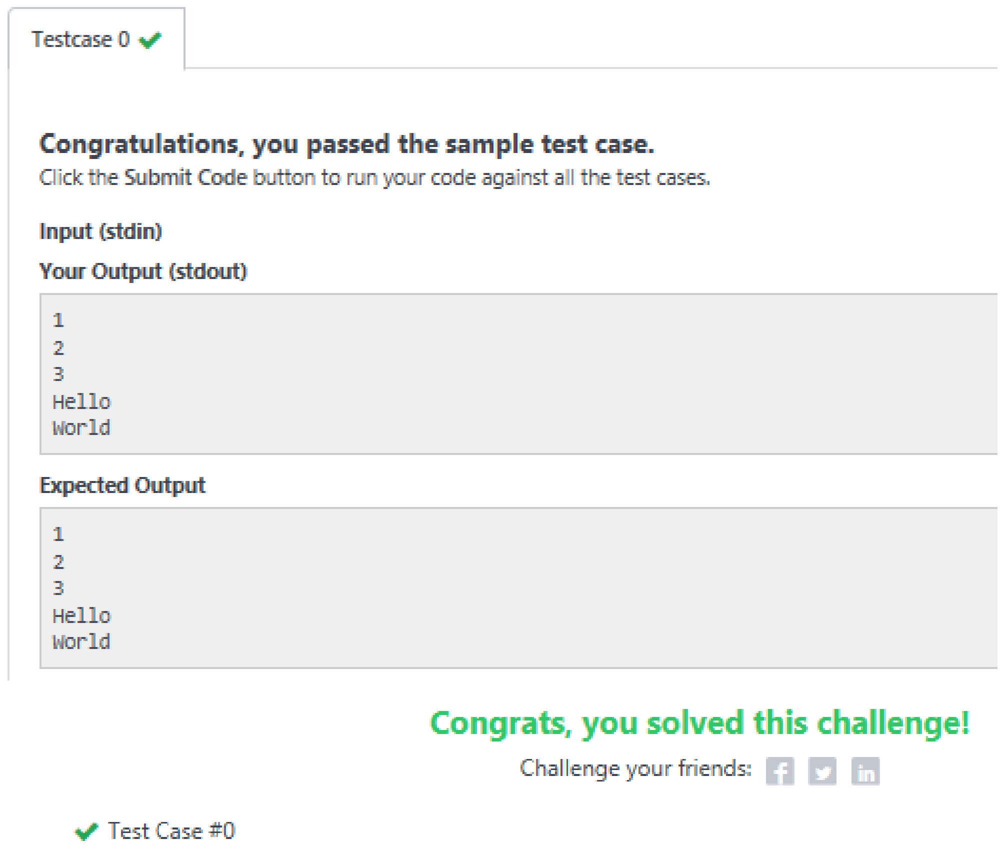
Do not use method overloading because your answer will not be accepted.

Program

```
import
java.io.IOExceptio
n;

                    import java.lang.reflect.Method;

                    class Printer {

                      public <T> void printArray(T[] array) {

                        for (T item : array) {

                          System.out.println(item);
```

```
        }
    }
}

public class Solution {
    public static void main(String args[]) {
        Printer myPrinter    = new Printer();
        Integer[] intArray   = { 1, 2, 3 };
        String[] stringArray = {"Hello", "World"};
        myPrinter.printArray(intArray);
        myPrinter.printArray(stringArray);
        int count = 0;
        for (Method method :
Printer.class.getDeclaredMethods()) {

            String name = method.getName();

            if (name.equals("printArray")) {

                count++;

            }

        }

        if (count > 1) {

            System.out.println("Method overloading is not
allowed!");

        }

    }

}
```

Output



Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the Submit Code button to run your code against all the test cases.

Input (stdin)

Your Output (stdout)

```
1
2
3
Hello
World
```

Expected Output

```
1
2
3
Hello
World
```

Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0

Q61

You are given a $6 * 6$ 2D array. An hourglass in an array is a portion shaped like this:

```
a b c
  d
e f g
```

For example, if we create an hourglass using the number 1 within an array full of zeros, it may look like this:

```
1 1 1 0 0 0
0 1 0 0 0 0
1 1 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
```

Actually, there are many hourglasses in the array above. The three leftmost hourglasses are the following:

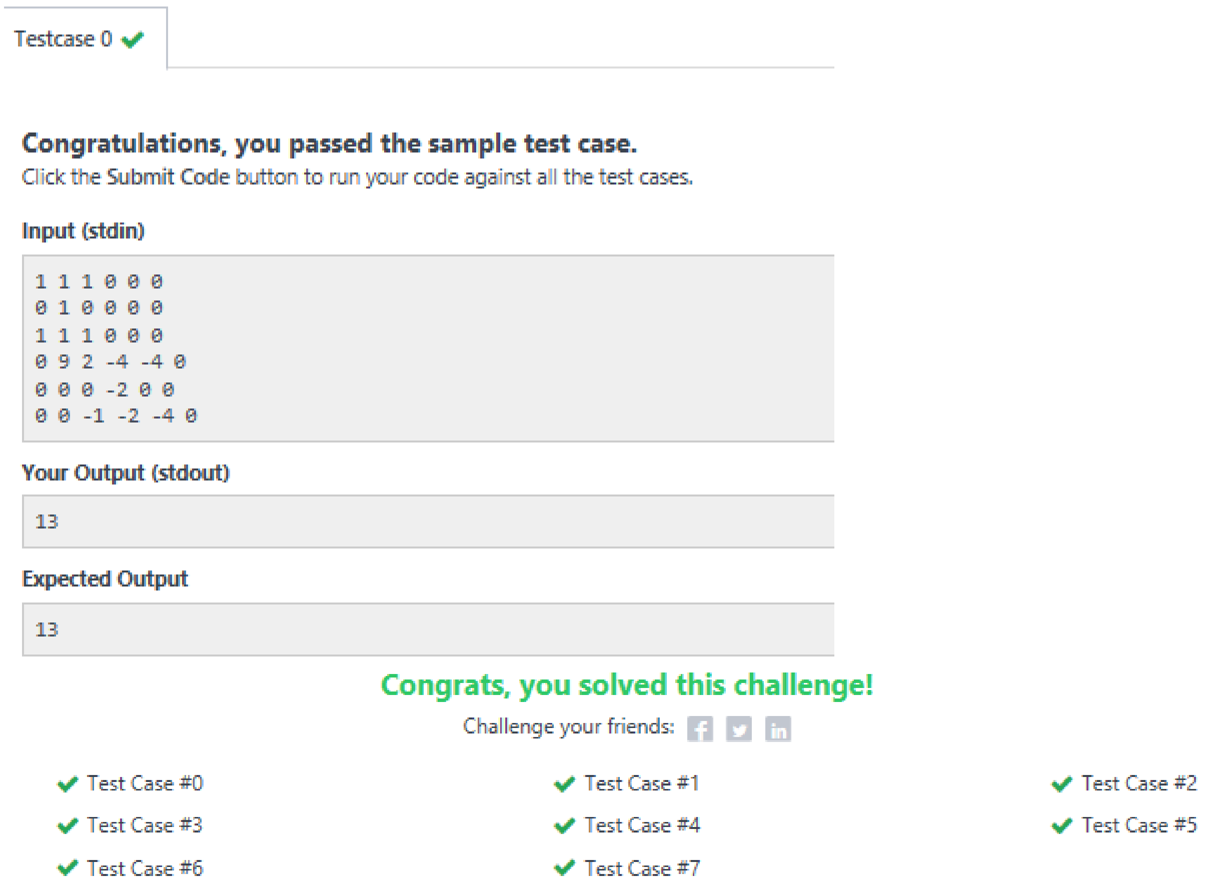
```
1 1 1     1 1 0     1 0 0
  1         0         0
1 1 1     1 1 0     1 0 0
```

The sum of an hourglass is the sum of all the numbers within it. The sum for the hourglasses above are 7, 4, and 2, respectively.

In this problem you have to *print the largest sum among all the hourglasses* in the array.

Program

```
import java.util.Scanner;




public class Solution {


   public static void main(String[] args) {


       Scanner scan = new Scanner(System.in);


       int arr[][] = new int[6][6];


       for (int row = 0; row < 6; row++) {
            for (int col = 0; col < 6; col++) {

                arr[row][col] = scan.nextInt();

            }

        }

        scan.close();



        System.out.println(maxHourglass(arr));

    }



    public static int maxHourglass(int [][] arr) {

        int max = Integer.MIN_VALUE;

        for (int row = 0; row < 4; row++) {

            for (int col = 0; col < 4; col++) {

                int sum = findSum(arr, row, col);

                max = Math.max(max, sum);
            }

        }

        return max;

    }



    private static int findSum(int [][] arr, int r, int c) {

        int sum = arr[r+0][c+0] + arr[r+0][c+1] + arr[r+0][c+2]

                              + arr[r+1][c+1] +

                    arr[r+2][c+0] + arr[r+2][c+1] + arr[r+2][c+2];

        return sum;

    }

}
```
Output


Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

```
1 1 1 0 0 0
0 1 0 0 0 0
1 1 1 0 0 0
0 9 2 -4 -4 0
0 0 0 -2 0 0
0 0 -1 -2 -4 0
```

Your Output (stdout)

```
13
```

Expected Output

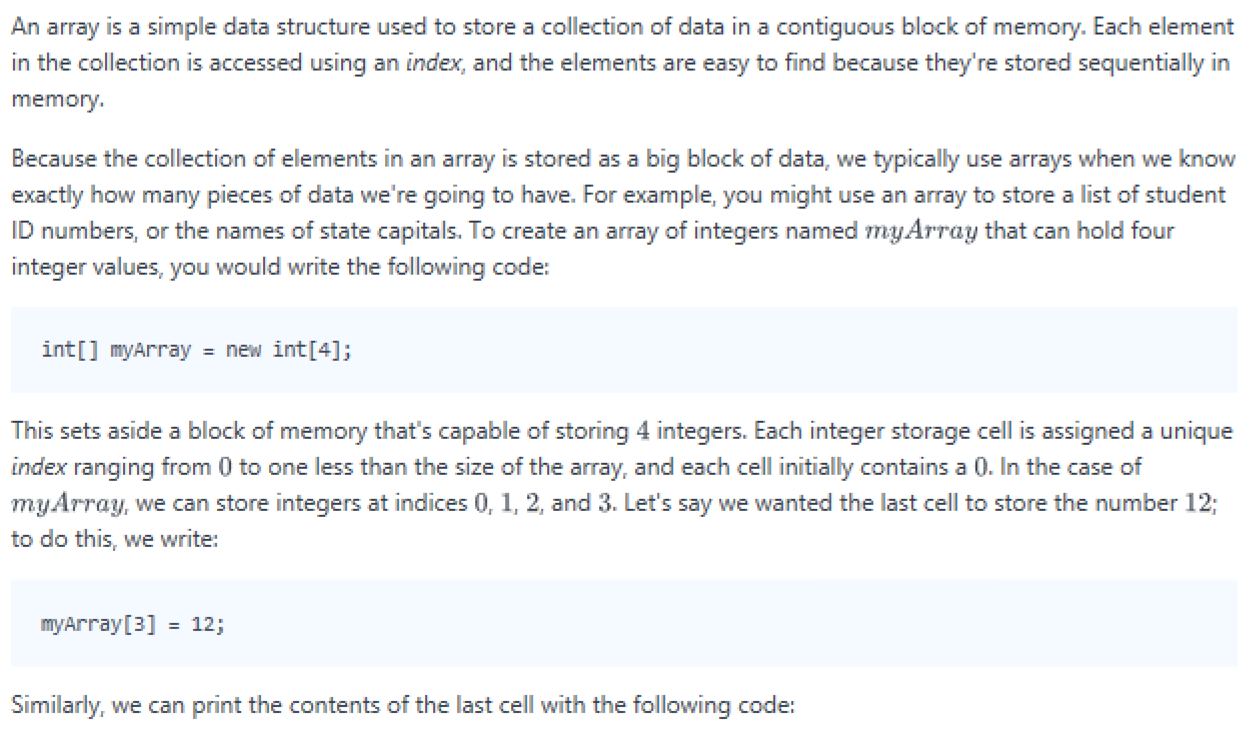
```
13
```

Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0	✓ Test Case #1	✓ Test Case #2
✓ Test Case #3	✓ Test Case #4	✓ Test Case #5
✓ Test Case #6	✓ Test Case #7	

Q62



An array is a simple data structure used to store a collection of data in a contiguous block of memory. Each element in the collection is accessed using an *index*, and the elements are easy to find because they're stored sequentially in memory.

Because the collection of elements in an array is stored as a big block of data, we typically use arrays when we know exactly how many pieces of data we're going to have. For example, you might use an array to store a list of student ID numbers, or the names of state capitals. To create an array of integers named $myArray$ that can hold four integer values, you would write the following code:

```
int[] myArray = new int[4];
```

This sets aside a block of memory that's capable of storing 4 integers. Each integer storage cell is assigned a unique *index* ranging from 0 to one less than the size of the array, and each cell initially contains a 0. In the case of $myArray$, we can store integers at indices 0, 1, 2, and 3. Let's say we wanted the last cell to store the number 12; to do this, we write:

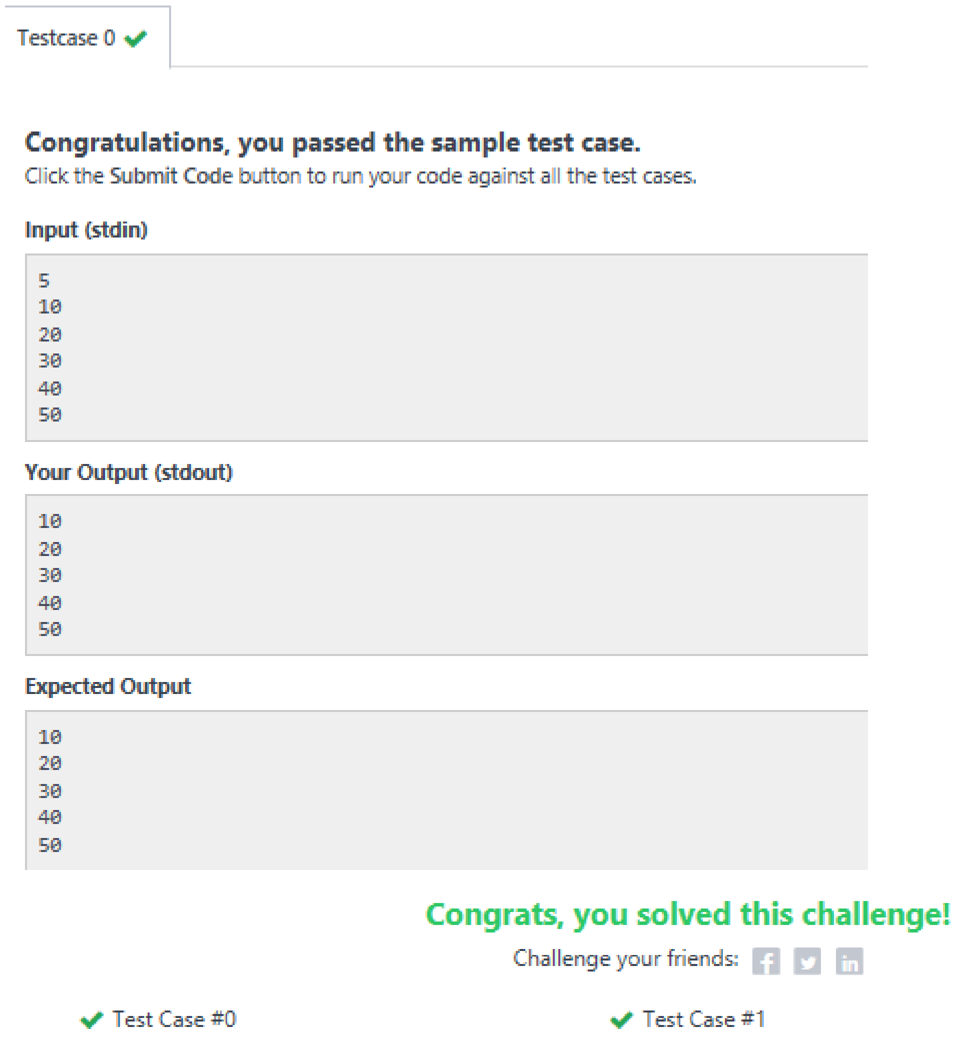
```
myArray[3] = 12;
```

Similarly, we can print the contents of the last cell with the following code:

Program

```
import java.util.*;


public class Solution {


    public static void main(String[] args) {


        Scanner scan = new Scanner(System.in);

        int n = scan.nextInt();


        int [] a = new int[n];


        for (int i = 0 ; i < n; i++) {


            a[i] = scan.nextInt();


        }
        scan.close();

      // Prints each sequential element in array a
      for (int i = 0; i < a.length; i++) {
         System.out.println(a[i]);
      }
   }
}
```

Output



Testcase 0 ✓

Congratulations, you passed the sample test case.

Click the Submit Code button to run your code against all the test cases.

Input (stdin)

```
5
10
20
30
40
50
```

Your Output (stdout)

```
10
20
30
40
50
```

Expected Output

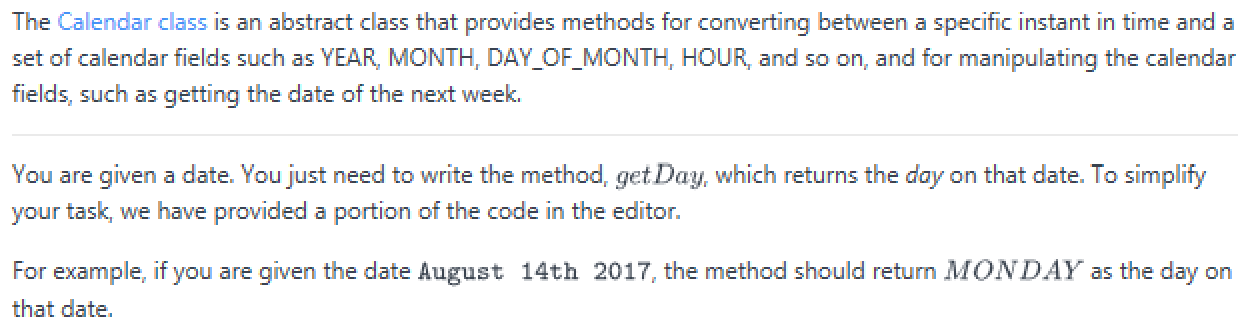
```
10
20
30
40
50
```

Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1

Q63



The Calendar class is an abstract class that provides methods for converting between a specific instant in time and a set of calendar fields such as YEAR, MONTH, DAY_OF_MONTH, HOUR, and so on, and for manipulating the calendar fields, such as getting the date of the next week.

You are given a date. You just need to write the method, $getDay$, which returns the *day* on that date. To simplify your task, we have provided a portion of the code in the editor.

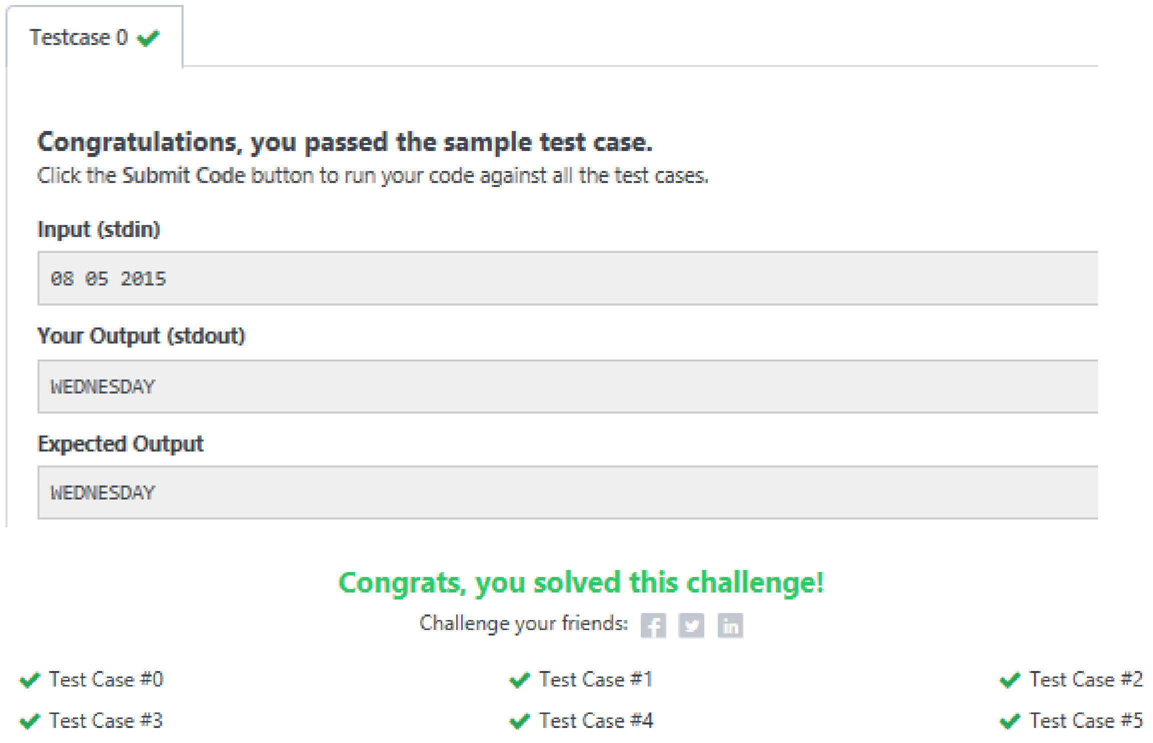
For example, if you are given the date `August 14th 2017`, the method should return $MONDAY$ as the day on that date.

Program

```
import java.util.Scanner;


import java.time.LocalDate;




public class Solution {


   public static String getDay(String day, String month, String year) {


       int d = Integer.valueOf(day);


       int m = Integer.valueOf(month);


       int y = Integer.valueOf(year);


       LocalDate date = LocalDate.of(y, m, d);


       return date.getDayOfWeek().toString();


   }
```

```
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        String month = in.next();
        String day = in.next();
        String year = in.next();

        System.out.println(getDay(day, month, year));
    }
}
```

Output



Testcase 0 ✓

Congratulations, you passed the sample test case.
Click the Submit Code button to run your code against all the test cases.

Input (stdin)

```
08 05 2015
```

Your Output (stdout)

```
WEDNESDAY
```

Expected Output

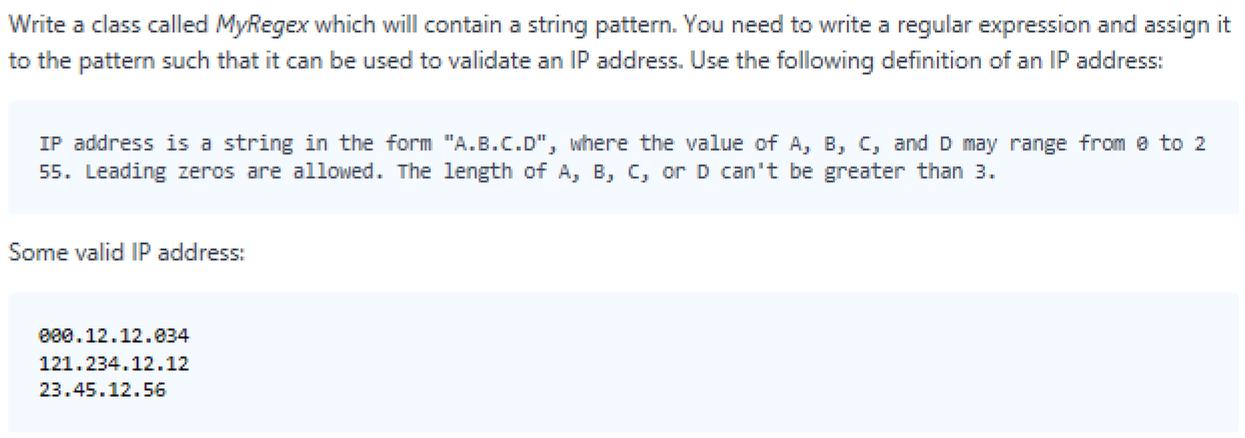
```
WEDNESDAY
```

Congrats, you solved this challenge!

Challenge your friends:

✓ Test Case #0 ✓ Test Case #1 ✓ Test Case #2
✓ Test Case #3 ✓ Test Case #4 ✓ Test Case #5

Q64


Write a class called *MyRegex* which will contain a string pattern. You need to write a regular expression and assign it to the pattern such that it can be used to validate an IP address. Use the following definition of an IP address:

```
IP address is a string in the form "A.B.C.D", where the value of A, B, C, and D may range from 0 to 2
55. Leading zeros are allowed. The length of A, B, C, or D can't be greater than 3.
```

Some valid IP address:

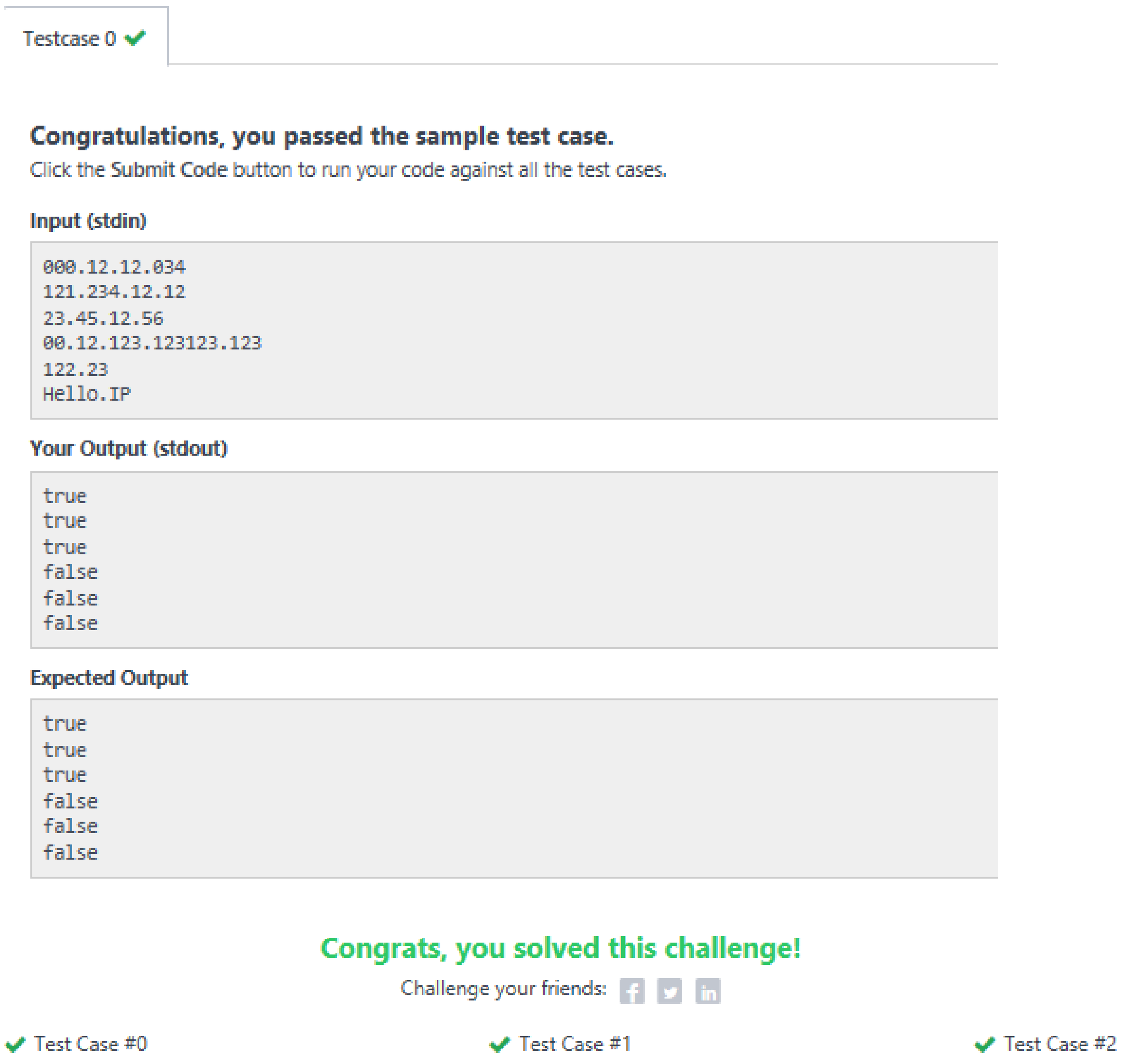
```
000.12.12.034
121.234.12.12
23.45.12.56
```

Program

```
import java.util.regex.Matcher;

import java.util.regex.Pattern;

import java.util.Scanner;


class Solution{


    public static void main(String[] args){

        Scanner in = new Scanner(System.in);

        while(in.hasNext()){

            String IP = in.next();

            System.out.println(IP.matches(new MyRegex().pattern));

        }


    }

}





/*
```

```
    [01]?\\d{1,2}   matches numbers 0-199.

    2[0-4]\\d       matches numbers 200-249

    25[0-5]         matches numbers 250-255

*/

class MyRegex {

   String num = "([01]?\\d{1,2}|2[0-4]\\d|25[0-5])";

   String pattern = num + "." +  num + "." +  num + "." + num;

}
```

Output



Congratulations, you passed the sample test case.

Click the Submit Code button to run your code against all the test cases.

Input (stdin)

```
000.12.12.034
121.234.12.12
23.45.12.56
00.12.123.123123.123
122.23
Hello.IP
```

Your Output (stdout)

```
true
true
true
false
false
false
```

Expected Output

```
true
true
true
false
false
false
```

Congrats, you solved this challenge!

Thank you