Constraints

- $1 \leq length(a), length(b) \leq 50$
- Strings a and b consist of English alphabetic characters.
- The comparison should NOT be case sensitive.

Sample Input 0

```
anagram
margana
```

Sample Output 0

```
Anagrams
```

Explanation 0

Character	Frequency: anagram	Frequency: margana
A or a	3	3
G or g	1	1
N or n	1	1
M or m	1	1
R or r	1	1

The two strings contain all the same letters in the same frequencies, so we print "Anagrams".

Sample Input 1

```
anagramm
marganaa
```

Sample Output 1

```
Not Anagrams
```

Explanation 1

Change Theme Language Java 7

```java
import java.util.Scanner;

public class Solution {
    static boolean isAnagram(String a, String b) {
        char[] arr1 = a.replaceAll("\\s", "").toLowerCase().toCharArray();
        char[] arr2 = b.replaceAll("\\s", "").toLowerCase().toCharArray();

        if (arr1.length != arr2.length)
            return false;
        int[] value = new int[26];
        for (int i = 0; i < arr1.length; i++)
        {
            value[arr1[i] - 97]++;
            value[arr2[i] - 97]--;
        }
        for (int i = 0; i < 26; i++)
            if (value[i] != 0)
                return false;
        return true;
    }

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);
        String a = scan.next();
        String b = scan.next();
        scan.close();
        boolean ret = isAnagram(a, b);
        System.out.println( (ret) ? "Anagrams" : "Not Anagrams" );
    }
}
```

Line: 15 Col: 10

Upload Code as File Test against custom input Run Code Submit Code

Constraints

- $1 \leq length(a), length(b) \leq 50$
- Strings a and b consist of English alphabetic characters.
- The comparison should NOT be case sensitive.

Sample Input 0

```
anagram
margana
```

Sample Output 0

```
Anagrams
```

Explanation 0

Character	Frequency: anagram	Frequency: margana
A or a	3	3
G or g	1	1
N or n	1	1
M or m	1	1
R or r	1	1

The two strings contain all the same letters in the same frequencies, so we print "Anagrams".

Sample Input 1

```
anagramm
marganaa
```

Sample Output 1

```
Not Anagrams
```

Explanation 1

Line: 15 Col: 10

Upload Code as File | Test against custom input | Run Code | Submit Code

You have earned 10.00 points!
You are now 25 points away from the 3rd star for your java badge.
17% 55/80

Congratulations

You solved this challenge. Would you like to challenge your friends?

Next Challenge

Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6

Compiler Message

Success

Input (stdin) Download

```
anagram
margana
```

Expected Output Download

```
Anagrams
```

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.

Note: We recommend completing Java Stdin and Stdout I before attempting this challenge.

Input Format

There are three lines of input:

1. The first line contains an integer.
2. The second line contains a double.
3. The third line contains a String.

Output Format

There are three lines of output:

1. On the first line, print `String:` followed by the unaltered String read from stdin.
2. On the second line, print `Double:` followed by the unaltered double read from stdin.
3. On the third line, print `Int:` followed by the unaltered integer read from stdin.

To make the problem easier, a portion of the code is already provided in the editor.

Note: If you use the nextLine() method immediately following the nextInt() method, recall that nextInt() reads integer tokens; because of this, the last newline character for that line of integer input is still queued in the input buffer and the next nextLine() will be reading the remainder of the integer line (which is empty).

Sample Input

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

Sample Output

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

Change Theme Language Java 7

```
1  import java.util.Scanner;
2
3  public class Solution {
4
5      public static void main(String[] args) {
6          Scanner scan = new Scanner(System.in);
7          int i = scan.nextInt();
8          double d=scan.nextDouble();
9          scan.nextLine();
10         String s=scan.nextLine();
11
12         System.out.println("String: " + s);
13         System.out.println("Double: " + d);
14         System.out.println("Int: " + i);
15     }
16 }
```

Line: 10 Col: 34

Upload Code as File Test against custom input Run Code Submit Code

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.

Note: We recommend completing Java Stdin and Stdout I before attempting this challenge.

Input Format

There are three lines of input:

1. The first line contains an integer.
2. The second line contains a double.
3. The third line contains a String.

Output Format

There are three lines of output:

1. On the first line, print `String:` followed by the unaltered String read from stdin.
2. On the second line, print `Double:` followed by the unaltered double read from stdin.
3. On the third line, print `Int:` followed by the unaltered integer read from stdin.

To make the problem easier, a portion of the code is already provided in the editor.

Note: If you use the nextLine() method immediately following the nextInt() method, recall that nextInt() reads integer tokens; because of this, the last newline character for that line of integer input is still queued in the input buffer and the next nextLine() will be reading the remainder of the integer line (which is empty).

Sample Input

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

Sample Output

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

Line: 10 Col: 34

Upload Code as File | Test against custom input | Run Code | Submit Code

Test case 0
Test case 1
Test case 2
Test case 3
Test case 4

Compiler Message

Success

Input (stdin) Download

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

Expected Output Download

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

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:

```
System.out.println(myArray[3]);
```

The code above prints the value stored at index 3 of $myArray$, which is 12 (the value we previously stored there). It's important to note that while Java initializes each cell of an array of integers with a 0, not all languages do this.

Task

The code in your editor does the following:

1. Reads an integer from stdin and saves it to a variable, n, denoting some number of integers.
2. Reads n integers corresponding to $a_0, a_1, \ldots, a_{n-1}$ from stdin and saves each integer a_i to a variable, val.
3. Attempts to print each element of an array of integers named a.

Write the following code in the unlocked portion of your editor:

1. Create an array, a, capable of holding n integers.
2. Modify the code in the loop so that it saves each sequential value to its corresponding location in the array.

Change Theme Language Java 7

```java
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]);
        }
    }
}
```

Line: 12 Col: 32

UploadCodeasFile Test against custom input Run Code Submit Code

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:

```
System.out.println(myArray[3]);
```

The code above prints the value stored at index 3 of $myArray$, which is 12 (the value we previously stored there). It's important to note that while Java initializes each cell of an array of integers with a 0, not all languages do this.

Task

The code in your editor does the following:

1. Reads an integer from stdin and saves it to a variable, n, denoting some number of integers.
2. Reads n integers corresponding to $a_0, a_1, \ldots, a_{n-1}$ from stdin and saves each integer a_i to a variable, val.
3. Attempts to print each element of an array of integers named a.

Write the following code in the unlocked portion of your editor:

1. Create an array, a, capable of holding n integers.
2. Modify the code in the loop so that it saves each sequential value to its corresponding location in the array.

Line: 12 Col: 32

Upload Code as File | Test against custom input | Run Code | Submit Code

You have earned 5.00 points!
You are now 10 points away from the 3rd star for your Java badge.
67% 70/80

Congratulations
You solved this challenge. Would you like to challenge your friends? Next Challenge

Test case 0
Test case 1

Compiler Message
Success

Input (stdin) Download

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

Expected Output Download

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:

```
class Soccer extends Sports{
    @Override
    String getName(){
        return "Soccer Class";
    }
}
```

Note: When overriding a method, you should precede it with the `@Override` annotation. The parameter(s) and return type of an overridden method must be exactly the same as those of the method inherited from the supertype.

Task

Complete the code in your editor by writing an overridden getNumberOfTeamMembers method that prints the same statement as the superclass' getNumberOfTeamMembers method, except that it replaces n with 11 (the number of players on a Soccer team).

Output Format

When executed, your completed code should print the following:

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

Change Theme Language Java 7

```
1  import java.util.*;
2  class Sports{
3
4      String getName(){
5          return "Generic Sports";
6      }
7
8      void getNumberOfTeamMembers(){
9          System.out.println( "Each team has n players in " + getName() );
10     }
11 }
12
13 class Soccer extends Sports{
14     @Override
15     String getName(){
16         return "Soccer Class";
17     }
18
19     void getNumberOfTeamMembers(){
20         System.out.println( "Each team has 11 players in " + getName() );
21     }
22
23 }
24
25 public class Solution{
26
27     public static void main(String []args){
28         Sports c1 = new Sports();
29         Soccer c2 = new Soccer();
30         System.out.println(c1.getName());
31         c1.getNumberOfTeamMembers();
32         System.out.println(c2.getName());
33         c2.getNumberOfTeamMembers();
34     }
35 }
36
```

Line: 1 Col: 20

Upload Code as File Test against custom input Run Code Submit Code

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:

```
class Soccer extends Sports{
    @Override
    String getName(){
        return "Soccer Class";
    }
}
```

Note: When overriding a method, you should precede it with the `@Override` annotation. The parameter(s) and return type of an overridden method must be exactly the same as those of the method inherited from the supertype.

Task

Complete the code in your editor by writing an overridden getNumberOfTeamMembers method that prints the same statement as the superclass' getNumberOfTeamMembers method, except that it replaces n with 11 (the number of players on a Soccer team).

Output Format

When executed, your completed code should print the following:

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

```
35  }
36
```

Line: 1 Col: 20

Upload Code as File Test against custom input Run Code Submit Code

You have earned 10.00 points!
You are now 70 points away from the 4th star for your java badge.

0% 80/150

Congratulations

You solved this challenge. Would you like to challenge your friends?

Next Challenge

Test case 0

Compiler Message

Success

Expected Output Download

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