\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

From PrepInsta’s top 25 HackerRank questions , here are a few examples:

1. **Maximum Passengers Problem** – Matrix traversal with constraints.
2. **Minimum Street Lights** – Greedy algorithm for coverage.
3. **Maximize Earnings** – Job scheduling with profit optimization.
4. **Duplicate characters in a string**
5. **Second largest number in an array**
6. **Anagram check**
7. **Armstrong number validation**

Leetcode:

|  |
| --- |
| FIZZBUZZ  import java.util.ArrayList;  public class Exercise {  public String[] fizzBuzz(int n) {  ArrayList<String> arrayList = new ArrayList<>();  for (int i = 1; i <= n; i++) {  if (i % 3 == 0 && i % 5 == 0) {  arrayList.add("FizzBuzz");  } else if (i % 3 == 0) {  arrayList.add("Fizz");  } else if (i % 5 == 0) {  arrayList.add("Buzz");  } else {  **arrayList.add(String.valueOf(i));**  }  }  **return arrayList.toArray(new String[0]);**  }  } |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Most HackerRank challenges require you to read input from [stdin](https://en.wikipedia.org/wiki/Standard_streams#Standard_input_.28stdin.29) (standard input) and write output to [stdout](https://en.wikipedia.org/wiki/Standard_streams" \l "Standard_output_.28stdout.29) (standard output).

One popular way to read input from stdin is by using the [Scanner class](https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html) and specifying the *Input Stream* as *System.in*. For example:

Scanner scanner = new Scanner(System.in);

String myString = scanner.next();

int myInt = scanner.nextInt();

scanner.close();

System.out.println("myString is: " + myString);

System.out.println("myInt is: " + myInt);

**Why the Fix?**

* nextInt() and nextDouble() do **not consume the newline** (\n) after the number.
* nextLine() reads until the end of the line, so if you don’t consume the leftover newline, it will read an empty string.
* Adding sc.nextLine() before reading the actual string input ensures the buffer is clean.
* import java.io.\*;
* import java.util.\*;
* public class Solution {
* public static void main(String[] args) {
* /\* Enter your code here. Read input from STDIN. Print output to STDOUT. Your class should be named Solution. \*/
* Scanner sc=new Scanner(System.in);
* int c=sc.nextInt();
* Double b=sc.nextDouble();
* sc.nextLine();
* String a=sc.nextLine();

* System.out.println("String: "+a.toString());
* System.out.println("Double: "+b);
* System.out.println("Int: "+c);
* }
* }

A screenshot of a computer

AI-generated content may be incorrect.

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.function.\*;

import java.util.regex.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.joining;

import static java.util.stream.Collectors.toList;

public class Solution {

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

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

        int N = Integer.parseInt(bufferedReader.readLine().trim());

         for(int i=1;i<=10;i++){

             String a=N+" x "+ i+" = ";

             int b=N\*i;

            System.out.println(a+b);

         }

        bufferedReader.close();

    }

}

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**import java.io.\*;**

**import java.util.\*;**

**public class Solution {**

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

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

**Scanner sc = new Scanner(System.in);**

**int q = sc.nextInt(); // number of queries**

**for (int i = 0; i < q; i++) {**

**int a = sc.nextInt();**

**int b = sc.nextInt();**

**int n = sc.nextInt();**

**int sum = a;**

**for (int j = 0; j < n; j++) {**

**sum += b \* (int)Math.pow(2, j);**

**System.out.print(sum + " ");**

**}**

**System.out.println();**

**}**

**sc.close();**

**}**

**}**

**✅ Problem Summary**

**You are given q queries. Each query contains three integers: a, b, and n. For each query, you need to print a series of n space-separated integers using the formula:**

**Si=a+b×(20+21+⋯+2i)*Si*​=*a*+*b*×(20+21+⋯+2*i*)**

**📥 Input Format**

**The first line contains an integer q (number of queries).**

**Each of the next q lines contains three space-separated integers: a, b, n.**

**📤 Output Format**

**For each query, print the series on a new line.**

**💡 Java Code Solution**

**🧪 Example Input**

**2**

**0 2 10**

**5 3 5**

**✅ Example Output**

**2 6 14 30 62 126 254 510 1022 2046**

**8 14 26 50 98**

**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.**

**Input Format**

**The first line contains an integer, , denoting the number of test cases.  
Each test case, , is comprised of a single line with an integer, , which can be arbitrarily large or small.**

**Output Format**

**For each input variable  and appropriate primitive , you must determine if the given primitives are capable of storing it. If yes, then print:**

**n can be fitted in:**

**\* dataType**

**If there is more than one appropriate data type, print each one on its own line and order them by size (i.e.: ).**

**If the number cannot be stored in one of the four aforementioned primitives, print the line:**

**n can't be fitted anywhere.**

**Sample Input**

**5**

**-150**

**150000**

**1500000000**

**213333333333333333333333333333333333**

**-100000000000000**

**Sample 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**

**Explanation**

**can be stored in a *short*, an *int*, or a *long*.**

**is very large and is outside of the allowable range of values for the primitive data types discussed in this problem.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**import java.io.\*;**

**import java.util.\*;**

**public class Solution {**

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

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

**Scanner sc = new Scanner(System.in);**

**int t = sc.nextInt(); // number of test cases**

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

**try {**

**long n = sc.nextLong();**

**System.out.println(n + " can be fitted in:");**

**if (n >= Byte.MIN\_VALUE && n <= Byte.MAX\_VALUE)**

**System.out.println("\* byte");**

**if (n >= Short.MIN\_VALUE && n <= Short.MAX\_VALUE)**

**System.out.println("\* short");**

**if (n >= Integer.MIN\_VALUE && n <= Integer.MAX\_VALUE)**

**System.out.println("\* int");**

**if (n >= Long.MIN\_VALUE && n <= Long.MAX\_VALUE)**

**System.out.println("\* long");**

**} catch (Exception e) {**

**System.out.println(sc.next() + " can't be fitted anywhere.");**

**}**

**}**

**sc.close();**

**}**

**}**

**Here’s the complete Java solution for the problem you described, which checks whether a given number can fit into standard Java primitive data types: byte, short, int, and long.**

**✅ Java Code**

**📥 Input Format**

**5**

**-150**

**150000**

**1500000000**

**213333333333333333333333333333333333**

**-100000000000000**

**📤 Output Format**

**-150 can be fitted in:**

**\* short333333333333333333333333333333333 can't be fitted anywhere.**

**-100000000000000 can be fitted in:**

**\* long**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**"In computing, *End Of File* (commonly abbreviated *EOF*) is a condition in a computer operating system where no more data can be read from a data source." — (**[**Wikipedia: End-of-file**](https://en.wikipedia.org/wiki/End-of-file)**)**

**The challenge here is to read  lines of input until you reach *EOF*, then number and print all  lines of content.**

**Hint: Java's *Scanner.hasNext()* method is helpful for this problem.**

**Input Format**

**Read some unknown  lines of input from *stdin(System.in)* until you reach *EOF*; each line of input contains a non-empty *String*.**

**Output Format**

**For each line, print the line number, followed by a single space, and then the line content received as input.**

**Sample Input**

**Hello world**

**I am a file**

**Read me until end-of-file.**

**Sample Output**

**1 Hello world**

**2 I am a file**

**3 Read me until end-of-file.**

**Here’s the complete Java solution for reading input until EOF (End of File) and printing each line with its line number:**

**✅ Java Code Using Scanner.hasNext()**

**import java.util.\*;**

**public class Solution {**

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

**Scanner sc = new Scanner(System.in);**

**int lineNumber = 1;**

**while (sc.hasNext()) {**

**String line = sc.nextLine();**

**System.out.println(lineNumber + " " + line);**

**lineNumber++;**

**}**

**sc.close();**

**}**

**}**

**📥 Input Format**

**You don’t know how many lines will be provided. Each line contains a non-empty string.**

**📤 Output Format**

**For each line, print:**

**<line\_number> <line\_content>**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

[**Initializing Fields (The Java™ Tutorials > Learning the Java Language > Classes and Objects)**](https://docs.oracle.com/javase/tutorial/java/javaOO/initial.html)

**The**[**Calendar class**](https://docs.oracle.com/javase/7/docs/api/java/util/Calendar.html)**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, , which returns the *day* on that date. To simplify your task, we have provided a portion of the code in the editor.**

**Example**

**The method should return  as the day on that date.**

****

**Function Description**

**Complete the *findDay* function in the editor below.**

***findDay* has the following parameters:**

* ***int:* month**
* ***int:* day**
* ***int:* year**

**Returns**

* ***string:* the day of the week in capital letters**

**Input Format**

**A single line of input containing the space separated month, day and year, respectively, in    format.**

**Constraints**

**Sample Input**

**08 05 2015**

**Sample Output**

**WEDNESDAY**

**Explanation**

**The day on August th  was WEDNESDAY.**

**import java.io.\*;**

**import java.math.\*;**

**import java.security.\*;**

**import java.text.\*;**

**import java.util.\*;**

**import java.util.concurrent.\*;**

**import java.util.function.\*;**

**import java.util.regex.\*;**

**import java.util.stream.\*;**

**import static java.util.stream.Collectors.joining;**

**import static java.util.stream.Collectors.toList;**

**class Result {**

**/\***

**\* Complete the 'findDay' function below.**

**\***

**\* The function is expected to return a STRING.**

**\* The function accepts following parameters:**

**\*  1. INTEGER month**

**\*  2. INTEGER day**

**\*  3. INTEGER year**

**\*/**

**public static String findDay(int month, int day, int year) {**

**Calendar cal = Calendar.getInstance();**

**cal.set(Calendar.YEAR, year);**

**cal.set(Calendar.MONTH, month - 1); // Months are 0-based in Calendar**

**cal.set(Calendar.DAY\_OF\_MONTH, day);**

**String[] days = { "SUNDAY", "MONDAY", "TUESDAY", "WEDNESDAY", "THURSDAY", "FRIDAY", "SATURDAY" };**

**int dayOfWeek = cal.get(Calendar.DAY\_OF\_WEEK);**

**return days[dayOfWeek - 1];**

**}**

**}**

**public class Solution {**

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

**BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));**

**BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));**

**String[] firstMultipleInput = bufferedReader.readLine().replaceAll("\\s+$", "").split(" ");**

**int month = Integer.parseInt(firstMultipleInput[0]);**

**int day = Integer.parseInt(firstMultipleInput[1]);**

**int year = Integer.parseInt(firstMultipleInput[2]);**

**String res = Result.findDay(month, day, year);**

**bufferedWriter.write(res);**

**bufferedWriter.newLine();**

**bufferedReader.close();**

**bufferedWriter.close();**

**}**

**}**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Given a double-precision number, , denoting an amount of money, use the NumberFormat class' getCurrencyInstance method to convert into the US, Indian, Chinese, and French currency formats. Then print the formatted values as follows: US: formattedPayment India: formattedPayment China: formattedPayment France: formattedPayment where is 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). Input Format A single double-precision number denoting . Constraints Output Format On the first line, print US: u where is formatted for US currency. On the second line, print India: i where is formatted for Indian currency. On the third line, print China: c where is formatted for Chinese currency. On the fourth line, print France: f, where is formatted for French currency. Sample Input 12324.134 Sample Output US: $12,324.13 India: Rs.12,324.13 China: ￥12,324.13 France: 12 324,13 € Explanation Each line contains the value of formatted according to the four countries' respective currencies.**

**Copilot said:**

**Copilot**

**Here’s the complete Java solution using NumberFormat.getCurrencyInstance() to format a given double value into US, India, China, and France currency formats:**

**✅ Java Code**

**import java.util.\*;**

**import java.text.\*;**

**public class Solution {**

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

**Scanner sc = new Scanner(System.in);**

**double payment = sc.nextDouble();**

**sc.close();**

**// US Locale**

**Locale us = Locale.US;**

**// India Locale (custom, since Java doesn't have a built-in one)**

**Locale india = new Locale("en", "IN");**

**// China Locale**

**Locale china = Locale.CHINA;**

**// France Locale**

**Locale france = Locale.FRANCE;**

**// Format currency**

**NumberFormat usFormat = NumberFormat.getCurrencyInstance(us);**

**NumberFormat indiaFormat = NumberFormat.getCurrencyInstance(india);**

**NumberFormat chinaFormat = NumberFormat.getCurrencyInstance(china);**

**NumberFormat franceFormat = NumberFormat.getCurrencyInstance(france);**

**// Output**

**System.out.println("US: " + usFormat.format(payment));**

**System.out.println("India: " + indiaFormat.format(payment));**

**System.out.println("China: " + chinaFormat.format(payment));**

**System.out.println("France: " + franceFormat.format(payment));**

**}**

**}**

**🧪 Sample Input**

**12324.134**

**✅ Sample Output**

**US: $12,324.13**

**India: ₹12,324.13**

**China: ￥12,324.13**

**France: 12 324,13 €**

**Note: The Indian currency symbol may appear as ₹ or Rs. depending on your system's locale support.**

If the HackerRank test case expects ￥, but the platform doesn't allow it, this might be a **test case issue**. You can raise it with HackerRank support or your test administrator.