OOADJ Lab 7 Hackerrank challenge

Assigned by Deepthi

https://www.hackerrank.com/ooadj-lab-7

Abstract Class

You are tasked with designing a Java program to manage a code catalog system, allowing users to add and display repositories. Each repository comprises unique attributes including *ID, project name, description, programming language, and stars.* Create an abstract class `RepositoryItem` and create one or more classes that extend `RepositoryItem`. Implement methods to add and display repositories.

Keep in mind the format specifications while solving the problem. Submissions that do not make use of abstract classes will not be considered for evaluation.

Input Format

Input is given in the form of a single line with comma separated values.

Constraints

Follow the output format correctly to pass test cases.

Output Format

Display the Repository ID, Name, Description, Programming language and the number of stars accordingly. Follow the same format as the sample output.

Sample Input 0

```
352, Abstract Classes, Program to demonstrate the usage of abstract classes, Java, 5
```

Sample Output 0

```
Repository ID: 352
Name: Abstract Classes
Description: Program to demonstrate the usage of abstract classes
Programming Language: Java
Number of Stars: 5
```

```java
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

abstract class RepositoryItem {
   protected int id;
   protected String name;
   protected String description;
   protected String programmingLanguage;
   protected int stars;

   public RepositoryItem(int id, String name, String description, String
programmingLanguage, int stars) {
       this.id = id;
       this.name = name;
       this.description = description;
       this.programmingLanguage = programmingLanguage;
       this.stars = stars;
   }


   public abstract void display();
}


class Repository extends RepositoryItem {
   public Repository(int id, String name, String description, String
programmingLanguage, int stars) {
       super(id, name, description, programmingLanguage, stars);
   }


   @Override
   public void display() {
       System.out.println("Repository ID: " + id);
       System.out.println("Name: " + name);
       System.out.println("Description: " + description);
       System.out.println("Programming Language: " + programmingLanguage);
       System.out.println("Number of Stars: " + stars);
   }
}
```

```
class CodeCatalog {
   private List<RepositoryItem> repositories;

   public CodeCatalog() {
       repositories = new ArrayList<>();
   }


   public void addRepository(RepositoryItem repository) {
       repositories.add(repository);
   }

   public void displayAllRepositories() {
       for (RepositoryItem repository : repositories) {
           repository.display();
           System.out.println(); // Empty line for separation
       }
   }
}

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

       CodeCatalog catalog = new CodeCatalog();
       Scanner scanner = new Scanner(System.in);

       // System.out.println("Enter Repository ID, Project Name, Description,
Programming Language, Number of Stars (comma-separated):");
       String input = scanner.nextLine();

       String[] parts = input.split(",");
       int id = Integer.parseInt(parts[0].trim());
       String name = parts[1].trim();
       String description = parts[2].trim();
       String language = parts[3].trim();
       int stars = Integer.parseInt(parts[4].trim());

       RepositoryItem repository = new Repository(id, name, description, language,
stars);
       catalog.addRepository(repository);
```

```
        catalog.displayAllRepositories();

        scanner.close();
    }
}
```

ArrayList

HackerRank

You are tasked with finding out the highest scorers among the list of students you've been given. Write a program to identify and print the names of students with the highest score, if there's a tie for the highest score, it should print the names of all students with that score.T The program takes two lines of input. The first line contains a list of student names, and the second line contains their corresponding test scores.

Make use of ArrayLists to solve this problem. Submissions that do not make use of ArrayLists classes will not be considered for evaluation.

Input Format

Two lines of input: The first line contains a list of student names(comma separated), and the second line contains their corresponding test scores(comma separated).

Constraints

1 <= n <= 100

Output Format

Display the name(s) that belong to the highest score.

Sample Input 0

```
Sophia, Jackson, Olivia, Liam, Emma, Noah, Ava, Aiden, Isabella, Lucas, Mia, Mason, Harper, Ethan, Evelyn,
Alexander, Amelia, Benjamin, Charlotte, Logan, Oliver, Lily, James, Sophia, Jackson, Olivia, Liam, Emma,
Noah, Ava, Aiden, Isabella, Lucas, Mia, Mason, Harper, Ethan, Evelyn, Alexander, Amelia
94, 98, 91, 86, 90, 87, 92, 89, 93, 90, 95, 85, 98, 91, 94, 88, 92, 86, 93, 90, 94, 88, 91, 86, 90, 87, 98,
89, 93, 90, 94, 88, 91, 98, 90, 87, 98, 89, 93, 90
```

Sample Output 0

```
Jackson
Harper
Liam
Mia
Ethan
```

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

public class HighestScorer {
   public static void main(String[] args) {
       Scanner scanner = new Scanner(System.in);

       String[] namesInput = scanner.nextLine().split(",");
       String[] scoresInput = scanner.nextLine().split(",");

       ArrayList<String> studentNames = new ArrayList<>();
       ArrayList<Integer> scores = new ArrayList<>();
       for (String name : namesInput) {
           studentNames.add(name.trim());
       }
       for (String score : scoresInput) {
           scores.add(Integer.parseInt(score.trim()));
       }

       int highestScore = findHighestScore(scores);
       ArrayList<String> highestScorers = findHighestScorers(studentNames, scores,
highestScore);
       for (String scorer : highestScorers) {
           System.out.println(scorer);
       }
   }

   // Function to find the highest score
   private static int findHighestScore(ArrayList<Integer> scores) {
       int highestScore = scores.get(0);
       for (int score : scores) {
           if (score > highestScore) {
               highestScore = score;
           }
       }
       return highestScore;
   }

   // Function to find the names of students with the highest score
   private static ArrayList<String> findHighestScorers(ArrayList<String> names,
ArrayList<Integer> scores, int highestScore) {
       ArrayList<String> highestScorers = new ArrayList<>();
```

```
        for (int i = 0; i < scores.size(); i++) {
            if (scores.get(i) == highestScore) {
                highestScorers.add(names.get(i));
            }
        }
        return highestScorers;
    }
}
```

Linked List

You are given two linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

List 1: 2 -> 4 -> 3 (represents the number 342) List 2: 5 -> 6 -> 4 (represents the number 465)

Result: 7 -> 0 -> 8 (represents the number 807)

Input Format

Two lines with space separated integers, each representing one number.

Constraints

- The input linked lists are non-empty and represent non-negative integers.
- The length of each linked list is in the range [1, 100].
- The value of each node in both linked lists is in the range [0, 9].
- It is guaranteed that the input linked lists do not have leading zeros, except the number 0 itself.

Output Format

Print the expected sum of the given number.

Sample Input 0

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

Sample Output 0

```
7 0 8
```

Sample Input 1

```
1 3 4 5
3 4
```

Sample Output 1

```
4 7 4 5
```

```java
import java.util.Scanner;

class ListNode {
   int val;
   ListNode next;

   ListNode(int val) {
       this.val = val;
   }
}

public class AddTwoNumbers {
   public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
       ListNode dummyHead = new ListNode(0);
       ListNode p = l1, q = l2, current = dummyHead;
       int carry = 0;

       while (p != null || q != null) {
           int x = (p != null) ? p.val : 0;
           int y = (q != null) ? q.val : 0;
           int sum = carry + x + y;
           carry = sum / 10;
           current.next = new ListNode(sum % 10);
           current = current.next;
           if (p != null) p = p.next;
           if (q != null) q = q.next;
       }

       if (carry > 0) {
           current.next = new ListNode(carry);
       }

       return dummyHead.next;
   }

   // Helper method to print the result linked list
   public void printList(ListNode node) {
       while (node != null) {
           System.out.print(node.val + " ");
           node = node.next;
       }
       System.out.println();
```

```java
    }

    // Method to handle user input and create linked lists
    public ListNode[] getUserInput() {
        Scanner scanner = new Scanner(System.in);
        String[] nums1Str = scanner.nextLine().trim().split(" ");
        String[] nums2Str = scanner.nextLine().trim().split(" ");

        ListNode l1 = createList(nums1Str);
        ListNode l2 = createList(nums2Str);

        return new ListNode[]{l1, l2};
    }

    // Helper method to create a linked list from an array of integers represented as
strings
    public ListNode createList(String[] numsStr) {
        ListNode dummyHead = new ListNode(0);
        ListNode current = dummyHead;
        for (String numStr : numsStr) {
            int num = Integer.parseInt(numStr);
            current.next = new ListNode(num);
            current = current.next;
        }
        return dummyHead.next;
    }

    public static void main(String[] args) {
        AddTwoNumbers solution = new AddTwoNumbers();
        ListNode[] lists = solution.getUserInput();
        ListNode result = solution.addTwoNumbers(lists[0], lists[1]);
        solution.printList(result);
    }
}
```

Stack

Write a Java function that takes an integer `n` as input and returns a list of strings representing all valid combinations of parentheses.

Use stacks to solve the problem. Submissions that do not make use of stacks will not be considered for evaluation.

Input Format

Single integer `n`

Constraints

Please use Java 15 or later versions.

1 <= n <= 9

Output Format

A string displaying all valid combinations of paranthesis.

Sample Input 0

```
1
```

Sample Output 0

```
()
```

Sample Input 1

```
2
```

Sample Output 1

```
(())
()()
```

Sample Input 2

```
5
```

Sample Output 2

```
((((()))))
(((()())))
(((())()))
```

```
( ( ( () ) ) () )
( ( ( () ) ) ) ()
( ( () ( () ) ) )
( ( () () () ) )
( ( () () ) () )
( ( () () ) ) ()
( ( () ) ( () ) )
( ( () ) () () )
( ( () ) () ) ()
( ( () ) ) ( () )
( ( () ) ) () ()
( () ( ( () ) ) )
( () ( () () ) )
( () ( () ) () )
( () ( () ) ) ()
( () () ( () ) )
( () () () () )
( () () () ) ()
( () () ) ( () )
( () () ) () ()
( () ) ( ( () ) )
( () ) ( () () )
( () ) ( () ) ()
( () ) () ( () )
( () ) () () ()
() ( ( ( () ) ) )
() ( ( () () ) )
() ( ( () ) () )
() ( ( () ) ) ()
() ( () ( () ) )
() ( () () () )
() ( () () ) ()
() ( () ) ( () )
() ( () ) () ()
() () ( ( () ) )
() () ( () () )
() () ( () ) ()
() () () ( () )
() () () () ()
```

```java
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.util.Stack;

public class stackproblem {

   public static List<String> generateParentheses(int n) {
       List<String> result = new ArrayList<>();
       generateParenthesesHelper(n, 0, 0, new Stack<>(), "", result);
       return result;
   }

   private static void generateParenthesesHelper(int n, int openCount, int closeCount,
Stack<Character> stack, String current, List<String> result) {
       // Base case: if both open and close parentheses count reach n
       if (openCount == n && closeCount == n) {
           result.add(current);
           return;
       }

       // Add '(' if open count is less than n
       if (openCount < n) {
           stack.push('(');
           generateParenthesesHelper(n, openCount + 1, closeCount, stack, current +
"(", result);
           stack.pop();
       }

       // Add ')' if close count is less than open count
       if (closeCount < openCount) {
           stack.push(')');
           generateParenthesesHelper(n, openCount, closeCount + 1, stack, current +
")", result);
           stack.pop();
       }
   }

   public static void main(String[] args) {
       Scanner scanner = new Scanner(System.in);
       int n = scanner.nextInt();
       scanner.close();
```

```
        List<String> result = generateParentheses(n);
        for (String parentheses : result) {
            System.out.println(parentheses);
        }
    }
}
```