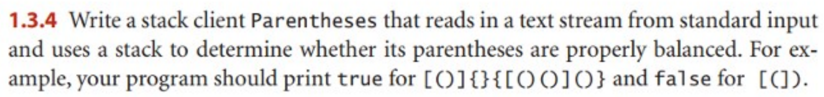
Exercise 4 *

Tuesday, September 13, 2022 9:20 AM

Extremely difficult problem.



1.3.4 Write a stack client `Parentheses` that reads in a text stream from standard input and uses a stack to determine whether its parentheses are properly balanced. For example, your program should print `true` for `[()]{}{[()()]()}` and `false` for `[(])`.

Trail and error process

Thought bubble #1: I have no clues. There are parentheses inside of a parenthesis.

Thought bubble #2:

create three methods, each method handle different kinds of parenthesis. Failed. Some of the parentheses are nested and some are not. Each method requires to deal with both nested and non-nested parenthesis, and such implementation is way too complicated for me to implement.

Thought bubble #3: No more thoughts. I'm stucked.

Correct steps to approach this problem:

Step #1: Analyze the problem.

Requirements:

Uses only ONE Stack to determine whether the parentheses are balanced.

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

Is a combination of non-nested and **nested** parentheses, which indicates the problem is made of smaller similar problems. Therefore, to solve such type of problem, we have to start by solving the problem of the smallest unit, then solve the bigger problem after it one by one.

Next, examining the properties of these parenthesis.

Each pair of parenthesis is composed of one left bracket and one right bracket
Even though some of the parentheses contains other parentheses, we can still identify each of them by looking at the corresponding one-side bracket. For example, [( )] is a nested parentheses, I can identify the two parentheses by looking at the opening brackets and see where the closing brackets exist. The first bracket is ' [' which is an opening square bracket (indicating that there should be a closing bracket that matches with it), then it is followed by an opening bracket ' (', followed by a closing bracket ') ', and this closing bracket match with the opening bracket ' (' before it. Since they are matched, we can say that this is a correct pair of parenthesis. And then ') ' is followed by '] ', which this closing bracket matches '] ' at the very beginning.

closing bracket ') ', and this closing bracket match with the opening bracket ' (' before it. Since they are matched, we can say that this is a correct pair of parenthesis. And then ') ' is followed by '] ', which this closing bracket matches '] ' at the very beginning.

Another example: [( ])

The first character is an opening square bracket, followed by a opening (bracket, next is a closing square bracket, however, this closing bracket doesn't match with (. So, this is an invalid parenthesis.

In this example, we started with two different opening brackets, according to the rules of parenthesis paring, the most inner opening bracket should get paired up first.

As a conclusion, we as human, identify a pair of parenthesis by first looking at the opening bracket and assume there will be a closing bracket matches it later on. If we don't see a closing bracket that matches it, we will say that this is not a valid parenthesis.

Think further:

Since the question only ask us to use ONE SINGLE stack to solve the problem, and according to the properties of a stack, pushing the entire text stream into the stack will not help us to solve the problem. Therefore, we can safely assume that we only want to push some of the characters of the string into the stack.

We already examined how human identify a pair of parenthesis, now we can apply this human method into our program.

Final approach:

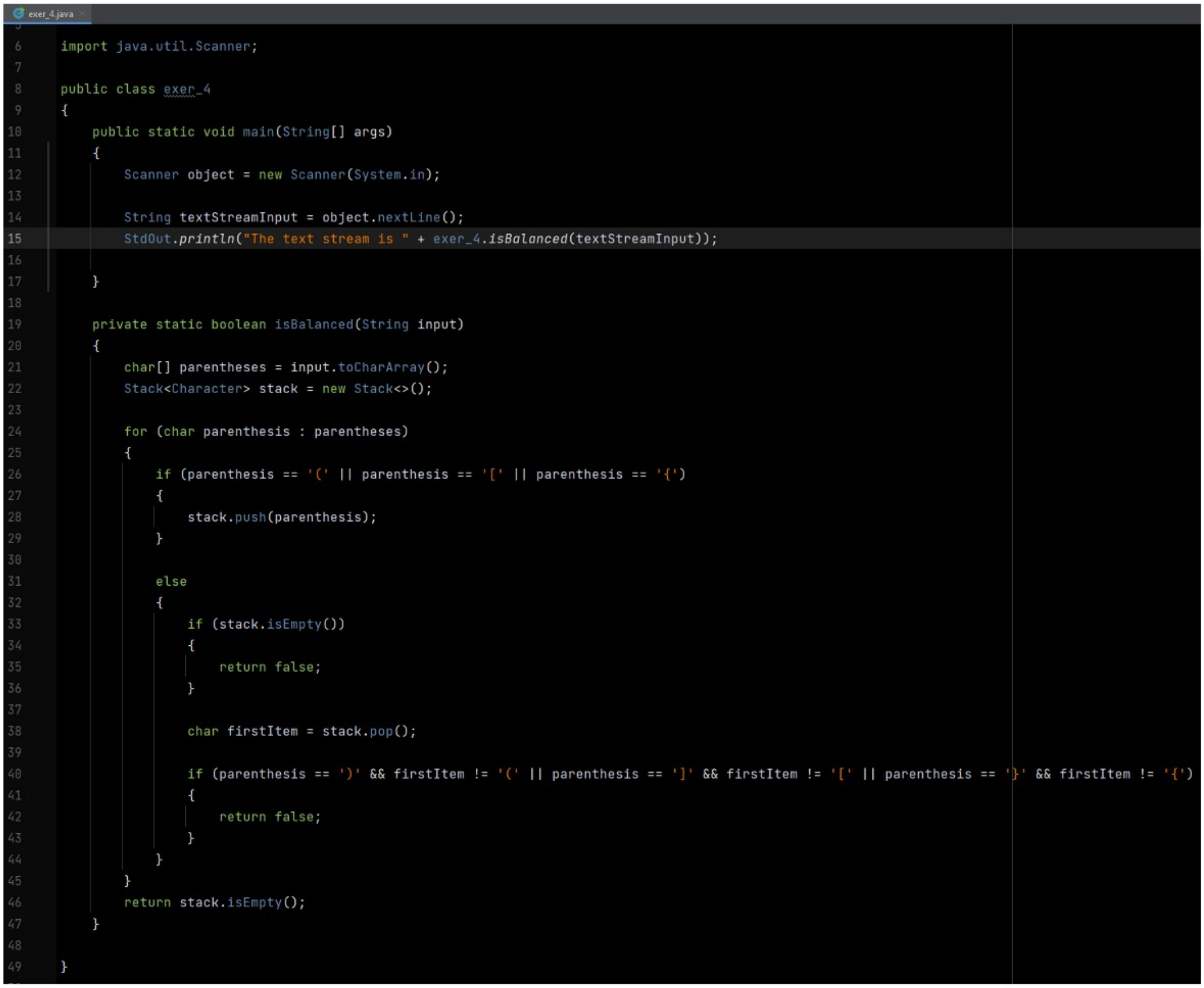
Human identify parentheses based on their opening brackets. So in our stack, we can push the opening brackets and compare them when we sees the closing brackets.

Example:

a) [()]{}{[()()]()}

Pushes opening bracket	The current stack	pops	Compare with closing bracket	The current stack
[(	[, (	(	) == matched	[
	[	[	] == matched	
{	{	{	} == matched	
{ [(	{ , [, (	(	) == matched	{ [
{ [(	{ , [, (	(	) == matched	{ [
{ [	{ , [	[	] == matched	{
{ (	{ , (	(	) == matched	{
{	{	{	} == matched	**The end**

Summary: You should always approach nested problem from the innermost small problem inside of the entire problem. If you can't identify which is the innermost most small problem, try to look and analyze the pattern of the whole problem and get some ideas from there. Practice will make you more sophisticated at this kind of problems



```java
import java.util.Scanner;

public class exer_4
{
    public static void main(String[] args)
    {
        Scanner object = new Scanner(System.in);

        String textStreamInput = object.nextLine();
        StdOut.println("The text stream is " + exer_4.isBalanced(textStreamInput));

    }

    private static boolean isBalanced(String input)
    {
        char[] parentheses = input.toCharArray();
        Stack<Character> stack = new Stack<>();

        for (char parenthesis : parentheses)
        {
            if (parenthesis == '(' || parenthesis == '[' || parenthesis == '{')
            {
                stack.push(parenthesis);
            }

            else
            {
                if (stack.isEmpty())
                {
                    return false;
                }

                char firstItem = stack.pop();

                if (parenthesis == ')' && firstItem != '(' || parenthesis == ']' && firstItem != '[' || parenthesis == '}' && firstItem != '{')
                {
                    return false;
                }
            }
        }
        return stack.isEmpty();
    }

}
```