



Haute école d'ingénierie et d'architecture Fribourg  
Hochschule für Technik und Architektur Freiburg

ALGORYTHMIQUE ET STRUCTURE DE DONNÉES

T1A

RÉSEAU ET SÉCURITÉ

---

## **S01 Type Abstrait, piles**

---

ROTEN MARC

2017/2018

# Table des matières

<b>1</b>	<b>CharStack Class</b>	<b>2</b>
<b>2</b>	<b>Parenth Class</b>	<b>4</b>
<b>3</b>	<b>Pre-conditions for charStack Class</b>	<b>6</b>
<b>4</b>	<b>Effect of following Methods</b>	<b>7</b>

# 1 CharStack Class

---

```
public class CharStack {
    private int topIndex;
    public char [] buffer;
    //-----
    private static final int DEFAULT_SIZE = 10;
    //-----
    public CharStack() {
        this(DEFAULT_SIZE);
    }
    //-----
    //this is the constructor of my class charstack
    //we have to create a new array of char and fix the topIndex
    //at -1 (array is now empty)
    public CharStack(int estimatedSize) {
        topIndex = -1;
        buffer = new char[estimatedSize];
    }
    //-----
    public boolean isEmpty() {
        /*
         * we check that the list is empty, if true, return true,
         * else false
         */
        if(topIndex == -1) {
            return true;
        } else return false;
    }
    //-----
    public char top() {
        /*
         * if stack is Empty we return an error value fixed as space
         * we return topValue
         */
        if(isEmpty()) {
            return ' ';
        } else return buffer[topIndex];
    }
    //-----
    public char pop() {
        /*
         * if stack is Empty we return an error value fixed as space
         * we decrement the topIndex and return the top value
         */
        if(isEmpty()) {
            return ' ';
        }
    }
}
```

```

    }
    char temp = buffer[topIndex];
    topIndex--;
    return temp;
}
//-----
public void push(char x) {
    /*
     * no need to have an empty stack for this method
     * we create a new char array
     * we decal all elements with our for iteration
     * we fix the topIndex at the end
     */
    topIndex++;
    if (topIndex==(buffer.length)) {
        char [] temp = new char[(buffer.length)*2];
        for (int i = 0; i < buffer.length; i++) {
            temp[i]=buffer[i];
        }
        buffer = temp;
    }
    buffer[topIndex]=x; }

}
//-----

```

---

// i used the following test class

---

```

public class CharStrackTest {
    public static void main(String[] args) {
        CharStack cs = new CharStack();
        cs.push('v');
        cs.pop();
        /*
         * we test with 1st test that push and top are returning an
         * empty stack without error
         */
        System.out.println("result expected is // True //, result
            obtained is "+ cs.isEmpty());
        cs.push('m');
        cs.push('h');
        cs.push('t');
        cs.push('y');
        cs.push('r');
        cs.push('o');
        cs.push('g');
        cs.push('l');
        cs.push('a');
        /*

```

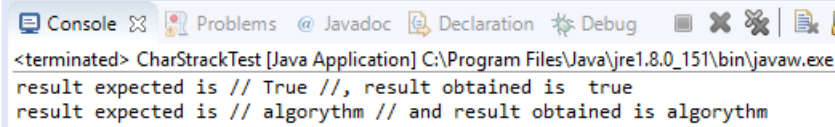
```

    * we test here that the method top() is good implemented
    */
    System.out.print("result expected is // algorithme // and
        result obtained is ");
    for(int i=0;i<cs.buffer.length;i++) {
        System.out.print(cs.top());
        cs.pop();
    }
}
}
}

```

---

the result in the java console is



```

<terminated> CharStackTest [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe
result expected is // True //, result obtained is true
result expected is // algorithme // and result obtained is algorithme

```

## 2 Parenth Class

---

```

public class Parenth {
    public static void main(String [] args) {
        if (args.length!=0) {
            String s = args[0];
            System.out.println(s+" : "+isBalanced(s));
        }
        String[] t = { "((o{()oo})o)",
            "oo((((((((((((((((((((((((((o))))))))))))))o{}",
            "oo())()",
            "oo()((()())",
            "oo()((()})",
            ")()",
            "({)})",
        };
        boolean [] answer = {true, true, false, false, false,
            false, false};
        boolean ok = true;
        for (int i=0; i<t.length; i++) {
            ok = ok & (isBalanced(t[i]) == answer[i]);
            System.out.print(t[i]+" : "+isBalanced(t[i]));
            System.out.println(" (should be "+answer[i]+")");
        }
        if (ok) System.out.println("\nTest passed successfully");
        else System.out.println("\nOops... There's a bug !");
    }
    //-----
    public static boolean isBalanced(String l) {
        /*

```

```

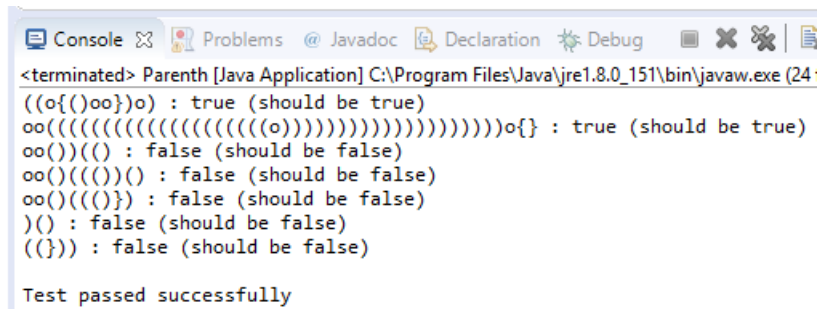
    * lets start with a nonBalanced boolean
    * we use our charStack class of exercise 1
    */
    char c;
    CharStack s = new CharStack();
    boolean isBalanced = false;
    for (int i=0; i<l.length(); i++) {
        c = l.charAt(i);
        if(isOpeningParenth(c)||isClosingParenth(c)) {
            /*
             * if we got an opening parenth, we use our push method
             */
            if(isOpeningParenth(c)) {
                s.push(c);
            }
            /*
             * if we got a closing parenth, we check if s.pop() is
             matching c
             */
            else {
                if(s.isEmpty()==false && isMatchingParenth(s.pop(),
                    c)) {
                    isBalanced =true;
                }else return false;
            }
        }
    }
    /*
     * particular if our stack is empty, we consider it as
     balanced
     */
    if (s.isEmpty()) return true;
    return false;
}

//-----
private static boolean isOpeningParenth(char c) {
    return (c == '(') || (c == '{');
}
private static boolean isClosingParenth(char c) {
    return (c == ')') || (c == '}');
}
private static boolean isMatchingParenth(char c1, char c2) {
    return ( (c1=='(' && (c2=='))')
        || ( (c1=='{' && (c2=='}')));
}
}

```

---

the console java give the following result



The screenshot shows a Java IDE console window with the following output:

```
<terminated> Parenth [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (24:
((o{()oo})o) : true (should be true)
oo((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((
))))))))))))))))))))))))))))))))))))))))))))o{} : true (should be true)
oo())() : false (should be false)
oo()(((()) : false (should be false)
oo()(((})) : false (should be false)
)() : false (should be false)
(({})) : false (should be false)

Test passed successfully
```

### 3 Pre-conditions for charStack Class

---

```
public class CharStack
{
    private int topIndex;
    private char[] buffer;
    public CharStack() { this(10); }
    public CharStack(int estSize) {...}
    public boolean isEmpty () {...}
    //no preConditions for this method

    public char top () {...}
    //the stack cant be empty!

    public char pop () {...}
    //the stack cant be empty!

    public void push (char x) {...}
    //no pre-conditions for this method
}
```

---

## 4 Effect of following Methods

---

```
static String f (String w) {  
    String r="";  
    CharStack s = new CharStack();  
    for (int i=0; i<w.length(); i++)  
        s.push(w.charAt(i));  
    while(!s.isEmpty())  
        r += s.pop();  
    return r;  
}
```

---

this method reverse the string we give in parameters

---

```
static int g (CharStack s, char e) {  
    CharStack r = new CharStack();  
    int c=0;  
    while(!s.isEmpty()) r.push(s.pop());  
    while(!r.isEmpty()) {  
        if (r.top()==e) c++;  
        s.push(r.pop());  
    }  
    return c;  
}
```

---

we transfer the content of the charstack s to a new charstack r, this method return the number of repetition of the given char in our initial charStack. After this, we switch the content of r stack back in the initial stack.