Lab7

April 4, 2024

1 Lab 7

Deadline: **Week 9** in your respective lab session

1.0.1 Name:Hansel

1.0.2 Student ID:230603866

1.1 Question 1 [1 mark]

Write a method called `shiftByN` that accepts an ArrayList `a` containing integers and an integer `n`. This method shifts every element in the ArrayList `a` by the value of `n` and updates the ArrayList a accordingly. For example, if `a` is initially `[1, 2, 3, 4, 5]` and `n` is 2, invoking `shiftByN` will result in `a` being updated to `[4, 5, 1, 2, 3]`.

Write your answer below:

[11]:
```
public static void shiftByN(ArrayList<Integer> a, int n)
{
    int length = a.size();
    ArrayList<Integer> temp = new ArrayList<>(length);
    for (int i = 0; i < length; i++)
    {
        temp.add(null);
    }
    for (int i = 0; i < length; i++)
    {
        temp.set((i + n) % length, a.get(i));
    }
    for (int i = 0; i < length; i++)
    {
        a.set(i, temp.get(i));
    }
}
```

Run your program:

[12]:
```
ArrayList<Integer> a = new ArrayList<>();
a.add(1);
a.add(2);
a.add(3);
a.add(4);
a.add(5);

shiftByN(a, 2);

System.out.println(a.toString());
```

```
[4, 5, 1, 2, 3]
```

1.2 Question 2 [1 mark]

Write a method named `getOverlap` that takes two ArrayLists of integers as input and returns an ArrayList containing integers that appear in both ArrayLists. For example, given input ArrayLists `[1, 2, 3]` and `[3, 2]`, the output will be `[2, 3]`. The order of elements in the output list does not matter.

Write your answer below:

[13]:
```
public static ArrayList<Integer> getOverlap(ArrayList<Integer> a,␣
 ↪ArrayList<Integer> b)
{
    Set<Integer> set_b = new HashSet<>(b);
    ArrayList<Integer> overlap = new ArrayList<>();
    for (int element : a)
    {
        if (set_b.contains(element))
        {
          overlap.add(element);
        }
    }
    return overlap;
}
```

Run your program:

[14]:
```
ArrayList<Integer> a = new ArrayList<>();
a.add(1);
a.add(2);
a.add(3);

ArrayList<Integer> b = new ArrayList<>();
b.add(3);
b.add(2);
```

```
System.out.println(getOverlap(a, b).toString());
```

```
[2, 3]
```

1.3 Question 3 [1 mark]

Write a method `toReversedArrayList` that takes a linked list of strings as input, converts it to an ArrayList in reverse order, and then returns it. For example, given input `["Alice", "Bob", "Charlie"]` the output will be `["Charlie", "Bob", "Alice"]`.

Write your answer below:

[15]:
```
public static ArrayList<String> toReversedArrayList(LinkedList<String>␣
 ↪linkedList)
{
    ArrayList<String> reversedList = new ArrayList<>();
    Stack<String> stack = new Stack<>();
    for (String element : linkedList)
    {
        stack.push(element);
    }
    while (!stack.isEmpty())
    {
        reversedList.add(stack.pop());
    }
    return reversedList;
}
```

Run your program:

[16]:
```
LinkedList<String> a = new LinkedList<>();
a.add("Alice");
a.add("Bob");
a.add("Charlie");

System.out.println(toReversedArrayList(a).toString());
```

```
[Charlie, Bob, Alice]
```

1.4 Question 4 [1 mark]

Write a method named `getCounts` that takes a linked list of generic type as input and returns a HashMap containing counts of each unique value. For example, given input `["Java", "Python", "Java", "C++"]`, the output will be `{Java=2, C++=1, Python=1}`.

Write your answer below:

[17]:
```java
public static <T> HashMap<T, Integer> getCounts(LinkedList<T> linkedList)
{
    HashMap<T, Integer> counts = new HashMap<>();
    for (T element : linkedList)
    {
        Integer count = counts.getOrDefault(element, 0);
        counts.put(element, count + 1);
    }
    return counts;
}
```

Run your program:

[18]:
```java
LinkedList<String> a = new LinkedList<>();
a.add("Java");
a.add("Python");
a.add("Java");
a.add("C++");

System.out.println(getCounts(a).toString());
```

```
{Java=2, C++=1, Python=1}
```

1.5 Question 5 [1 mark]

A bracket refers to any of the following characters: `(`, `)`, `{`, `}`, `[`, or `]`.

Two brackets form a matched pair when the opening bracket (i.e., `(`, `[`, or `{`) precedes a corresponding closing bracket (i.e., `)`, `]`, or `}`) of the exact same type. There are three types of matched pairs: `[]`, `{}`, and `()`.

A matching pair of brackets is considered unbalanced if the set of brackets it encloses are not matched. For example, `{[(])}` is not balanced because the contents in between `{` and `}` are not balanced. Specifically, the pair of square brackets encloses a single, unbalanced opening bracket, `(`, and the pair of parentheses encloses a single, unbalanced closing square bracket, `]`.

By this logic, we say a sequence of brackets is balanced if the following conditions are met: - It contains no unmatched brackets.
- The subset of brackets enclosed within the confines of a matched pair of brackets is also a matched pair of brackets.

Utilising Java Collections, the task is to ascertain whether a string of brackets forms a balanced sequence. If the sequence is indeed balanced, the method should return true; otherwise, it should return false.

Hint: Which data structure operates on a first-in-last-out basis?

Write your answer below:

[19]:
```
static boolean matchingBrackets(String s)
{
    Stack<Character> stack = new Stack<>();
    for (char c : s.toCharArray())
    {
        if (isOpenBracket(c))
        {
            stack.push(c);
        }
        else if (isClosingBracket(c))
        {
            if (stack.isEmpty() || !isMatchingPair(stack.pop(), c))
            {
                return false;
            }
        }
    }
    return stack.isEmpty();
}


private static boolean isOpenBracket(char c)
{
    return c == '(' || c == '[' || c == '{';
}


private static boolean isClosingBracket(char c)
{
    return c == ')' || c == ']' || c == '}';
}

private static boolean isMatchingPair(char opening, char closing)
{
    return (opening == '(' && closing == ')') ||
     (opening == '[' && closing == ']') ||
     (opening == '{' && closing == '}');
}
```

Run your program:

[20]:
```
System.out.println(matchingBrackets("{[()]}"));             // true
System.out.println(matchingBrackets("((({{{[[]]}}})))"));   // true
System.out.println(matchingBrackets("()"));                  // true
System.out.println(matchingBrackets("[]"));                  // true
System.out.println(matchingBrackets("{}"));                  // true
System.out.println();
```

```
System.out.println(matchingBrackets("{[(])}"));              // false
System.out.println(matchingBrackets("[(){}"));               // false
System.out.println(matchingBrackets(")"));                   // false
System.out.println(matchingBrackets("({)}"));                // false
```

```
true
true
true
true
true

false
false
false
false
```

[ ]: