List, Set, Map, Stack, Queue and Tree

- Write a program which solves the problem quoted below.

We have a list consisting of all numbers from 1 to 100. Each time; we remove two numbers randomly from the list, add them up, subtract 2 from the sum and add the result back in the list. We continue this process until there is exactly one number left in the list. What i that last number?

solution:

```java
public static void main(String[] args) {
    LinkedList<Integer> list = new LinkedList<>();
    Random r = new Random();
    // initialize
    for(int i=1; i<=100; i++)
        list.add(i);
    while(list.size() > 1) {
        // draw first
        int randX1 = r.nextInt(list.size());
        int num1 = list.remove(randX1);
        // draw second
        int randX2 = r.nextInt(list.size());
        int num2 = list.remove(randX2);
        // add their sum - 2
        list.add(num1+num2-2);
    }
    System.out.println(list);
}
```

* Consider a set of matryushka dolls each of which has a unique name. Assume that each doll is either small or medium or large in size. You are given two mappings. The first mapping maps big dolls to medium dolls, indicating which doll contains which. The second mapping maps medium dolls to small dolls, with the same meaning as that of first. Your task is to come up with a map from big dolls to small dolls showing containment information.

Note that there may be a big doll containing a medium doll which does not contain any small doll. For such cases you should map big doll to void.

In Java terms, you should write a function which takes two parameters of type `HashMap<String, String>` and returns an object of type `HashMap<String, String>`.

Main method:

```
public static void main(String[] args) {
    // BtM: big to medium
    HashMap<String, String> BtM = new HashMap<>();
    BtM.put("big1", "medium1");
    BtM.put("big2", "medium2");
    BtM.put("big3", "medium3");
    // MtS: medium to small
    HashMap<String, String> MtS = new HashMap<>();
    MtS.put("medium1", "small1");
    MtS.put("medium2", "small2");
    MtS.put("medium3", "small3");
    System.out.println(matryushka(BtM, MtS));
}
```

solution:

```
static HashMap<String, String> matryushka(HashMap<String, String> BtM, HashMap<String, String> MtS) {
    // BtS: big to small
    HashMap<String, String> BtS = new HashMap<>();
    for(String bigDoll: BtM.keySet()) {
        String mediumDoll = BtM.get(bigDoll);
        if(MtS.containsKey(mediumDoll))
            BtS.put(bigDoll, MtS.get(mediumDoll));
        else
            BtS.put(bigDoll, null);
    }
    return BtS;
}
```

- Write a program which provides a command-line interface to a stack of strings. Your program should push a string to a stack when user enters “push string”. Likewise it should pop a string from the stack and prints it to the screen when user enters “pop”. The program should warn the user when she tries to pop from an empty stack. Lastly, when user enters “quit”, the program should report the number of items remaining in the stack and terminate.

solution:

```java
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    Stack<String> st = new Stack<>();
    String in;
    while(!(in = sc.next()).equals("quit"))
        if(in.equals("push"))
            st.push(sc.next());
        else if(in.equals("pop"))
            if(st.empty())
                System.out.println("stack is empty!");
            else
                System.out.println(st.pop());
    System.out.println(st.size() + " elements left.");
}
```

- Write a function which takes a string and checks if all of the parentheses (if any) in it are balanced.

Hint: Use stack

example:

- `isBalanced("(yeah)") = true`
- `isBalanced("(yeah)(i)am(ba(lan)ced)") = true`
- `isBalanced("((something)") = false`
- `isBalanced("no parenthesis") = true`
- `isBalanced("we(i)rd") = true`

Main method:

```java
public static void main(String[] args) {
    System.out.println(isBalanced("(yeah)"));
    System.out.println(isBalanced("(yeah)(i)am(ba(lan)ced)"));
    System.out.println(isBalanced("((something)"));
    System.out.println(isBalanced("no parenthesis"));
    System.out.println(isBalanced("we(i)rd"));
}
```

solution:

```java
static boolean isBalanced(String s) {
    // the kind of elements that you
    // push to the stack has no importance
    // in this question
    Stack<Object> st = new Stack<>();
    for(int i=0; i<s.length(); i++)
        if(s.charAt(i)=='(')
            st.push(null);
        else if(s.charAt(i) == ')')
            if(st.empty())
                return false;
            else
                st.pop();
    return st.empty();
}
```

- Write a function which takes a `HashMap<String, Integer> map1` and a `HashMap<Integer, String> map2` as parameters and returns their composite mapping, which is of type `HashMap<String, String>`. The composition operation is the same with function composition in mathematics.

Main method:

```
public static void main(String[] args) {
    HashMap<String, Integer> map1 = new HashMap<>();
    map1.put("a", 1);
    map1.put("b", 2);
    map1.put("c", 3);
    HashMap<Integer, String> map2 = new HashMap<>();
    map2.put(1, "one");
    map2.put(2, "two");
    map2.put(3, "three");
    HashMap<String, String> res = compose(map1, map2);
    System.out.println(res);
}
```

solution:

```
static HashMap<String, String> compose(HashMap<String, Integer> map1, HashMap<Integer, String> map2) {
    HashMap<String, String> res = new HashMap<>();
    for(String key: map1.keySet()) {
        Integer n = map1.get(key);
        if(map2.containsKey(n))
            res.put(key, map2.get(n));
    }
    return res;
}
```

- Implement the same program for a queue structure, use addLast and removeFirst methods of LinkedList class.

solution:

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    LinkedList<String> q = new LinkedList<>();
    String in;
    while(!(in = sc.next()).equals("quit"))
        if(in.equals("enqueue"))
            q.addLast(sc.next());
        else if(in.equals("dequeue"))
            if(q.size() == 0)
                System.out.println("queue is empty!");
            else
                System.out.println(q.removeFirst());
    System.out.println(q.size() + " elements left.");
}
```