1.	**<u>Task: 99 Bottles problem</u>**
	
	The problem is discussed in the theory class. You have to generate the following output.
	
	`99 bottles of beer on the wall, 99 bottles of beer.`
	`Take one down and pass it around, 98 bottles of beer on the wall.`
	`98 bottles of beer on the wall, 98 bottles of beer.`
	`Take one down and pass it around, 97 bottles of beer on the wall.`
	`...`
	`3 bottles of beer on the wall, 3 bottles of beer.`
	`Take one down and pass it around, 2 bottles of beer on the wall.`
	`2 bottles of beer on the wall, 2 bottles of beer.`
	`Take one down and pass it around, 1 bottle of beer on the wall.`
	`1 bottle of beer on the wall, 1 bottle of beer.`
	`Take one down and pass it around, no more bottles of beer on the wall.`
	`No more bottles of beer on the wall, no more bottles of beer.`
	`Go to the store and buy some more, 99 bottles of beer on the wall.`
	
	Goals
	
	1. To give you hands-on experience in **unit testing**.
	2. To let you program considering just **readability**.
	3. To convince you that it is easier to develop and **debug using unit tests than by manual testing**
	
	Materials
	
	You are given a test code snippet containing a test case
		20

```
class BottleProblemTest {
    @Test
    void verse0() {
        String expected = "No more bottles of beer on the wall, no more bottles of beer.\n" +
                "Go to the store and buy some more, 99 bottles of beer on the wall.\n";
        BottleProblem bottleProblem = new BottleProblem();
        assertEquals(expected, BottleProblem.verse(0));
    }
}
```

To do

1. Implement the 99 bottles of beer code that can generate the **full song**, **a range of verses**, and **any single verse**. The goal is to make the code as readable as possible. As per discussion in the class, one way to define readability is, *how well the code reflects the problem it is solving*. **From the code, we want to easily identify** -
    a. How many verse variants are there?
    b. Which verses are most alike? In what way?
    c. Which verses are most different, and in what way?
2. Check the correctness of the code using test cases. There must be test cases to test the **full song**, **a range of verses**, and **any single verse**.
3. In your code, you should handle the wrong number of bottles like 150.

Restrictions

You are **not allowed to write the main method**. Run the test cases instead to check the correctness.

Rubrics for marking

Complete code by following all instructions ● If code contains main method ● If code is not readable	12 -2 -3

<table>
<tr><td></td><td>
<table>
<tr><td>Test code having 2-4 test cases</td><td>4</td></tr>
<tr><td>Test code having all test cases that can test the full song, a range of verses and a verse with variants.</td><td>4</td></tr>
</table>
</td><td></td></tr>
<tr><td>2.</td><td>

Task: Shape problem

You have to draw different shapes like circles, squares, etc. in a GUI application. A list of the circles and squares will be created in the appropriate order and the program must walk the list in that order and draw each circle or square.

Goals

1. To understand the problems of a code if the future change request is not considered.
2. To understand the benefit of applying the Open Close Principle (OCP) to facilitate future change requests.
3. To check the correctness of the code by using unit testing.

Materials

- You have to consider circles and squares are types of shapes.
- You do not have to actually draw different shapes. Instead, you can have methods that print text “drawing circle” or “drawing rectangle”.
- You have to use a List to store all shapes.

To do

1. You have to write a solution that can solve the shape problem.
2. Write test cases to check the correctness of your code.
3. A new change request has been made that your program should support rectangle shapes along with circles and squares. Write new code and test cases that can satisfy the new change request.

</td><td>20</td></tr>
</table>

4. Write the class and method names that you had to change to support the new functionality in a text file. Mention the problems of your code from the OCP perspective.
5. Write a code by following the OCP that can solve the shape problem.

Restrictions

You are not allowed to write the main method. Run the test cases instead.

Rubrics for marking

Each to-do item has 4 marks ● If to-do 1 is partially complete **-2** ● If test cases are missing for to-do items **-2**	5*4 = 20