
OOO-2 LAB 03 HANDOUT AND TASKS

Prepared by:

Md. Jubair Ibna Mostafa

Assistant Professor, IUT CSE



Department of Computer Science and Engineering
Islamic University of Technology

Contents

1	99 Bottles of OOP	3
1.1	Test Cases	3
1.1.1	Java Implementation of Test Cases:	4
1.1.2	C# Implementation of Test Cases:	6
1.2	Possible Solutions	10
1.2.1	Shameless Green	11
1.2.2	Speculatively General	12
2	Tasks based on previous classes	15
3	References	17

1 99 BOTTLES OF OOP

In this scenario, you have to write a solution that can generate lyrics of the famous song of the mid-20th century - **99 Bottles of Beer**.

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.

(—————)

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.

1.1 Test Cases

To consider the 99 bottles of beer scenario, we can use the following test cases to check the correctness of the program.

1. first verse
2. another verse
3. verse 0

4. verse 1
5. verse 2
6. a couple of verses
7. a few verses
8. whole song

1.1.1 Java Implementation of Test Cases:

```
1 public class BottlesTest {
2
3     private Bottles bottles = new Bottles();
4
5     @Test
6     void test_a_verse() {
7         String expected = "99 bottles of beer on the wall, 99 bottles
8         of beer.\n" +
9         "Take one down and pass it around, 98 bottles of beer
10        on the wall.\n";
11        assertEquals(expected, this.bottles.verse(99));
12    }
13
14    @Test
15    void test_another_verse() {
16        String expected = "89 bottles of beer on the wall, 89 bottles
17        of beer.\n" +
18        "Take one down and pass it around, 88 bottles of beer
19        on the wall.\n";
20        assertEquals(expected, this.bottles.verse(89));
21    }
22
23    @Test
24    void test_zero_verse() {
25        String expected = "No more bottles of beer on the wall, no more
26        bottles of beer.\n" +
```

```
21         "Go to the store and buy some more, 99 bottles of beer
on the wall.\n";
22         assertEquals(expected, this.bottles.verse(0));
23     }
24
25     @Test
26     void test_one_verse() {
27         String expected = "1 bottle of beer on the wall, 1 bottle of
beer.\n" +
28             "Take it down and pass it around, no more bottles of
beer on the wall.\n";
29         assertEquals(expected, this.bottles.verse(1));
30     }
31
32     @Test
33     void test_two_verse() {
34         String expected = "2 bottles of beer on the wall, 2 bottles of
beer.\n" +
35             "Take one down and pass it around, 1 bottle of beer on
the wall.\n";
36         assertEquals(expected, this.bottles.verse(2));
37     }
38
39     @Test
40     void test_a_couple_verses() {
41         String expected = "99 bottles of beer on the wall, 99 bottles
of beer.\n" +
42             "Take one down and pass it around, 98 bottles of beer
on the wall.\n" +
43
44             "98 bottles of beer on the wall, 98 bottles of beer.\n"
+
45             "Take one down and pass it around, 97 bottles of beer
on the wall.\n";
46         assertEquals(expected, this.bottles.verses(99, 98));
```

```
47     }
48
49     @Test
50     void test_a_few_verses() {
51         String expected = "2 bottles of beer on the wall, 2 bottles of
52         beer.\n" +
53             "Take one down and pass it around, 1 bottle of beer on
54             the wall.\n" +
55             "1 bottle of beer on the wall, 1 bottle of beer.\n" +
56             "Take it down and pass it around, no more bottles of
57             beer on the wall.\n" +
58             "No more bottles of beer on the wall, no more bottles
59             of beer.\n" +
60             "Go to the store and buy some more, 99 bottles of beer
61             on the wall.\n";
62         assertEquals(expected, this.bottles.verses(2, 0));
63     }
64
65     @Test
66     void test_the_whole_song() {
67         String expected = this.bottles.verses(99, 0);
68         assertEquals(expected, this.bottles.song());
69     }
70 }
```

1.1.2 C# Implementation of Test Cases:

[Unit Testing for C# Blog](#)

```
1 using _99Bottles;
2
3 namespace _99BottlesTest
4 {
5     public class Tests
```

```
6      {
7          Bottles _bottles;
8
9          [SetUp]
10         public void Setup()
11         {
12             _bottles = new Bottles();
13         }
14     /*
15
16         [Test]
17         public void Test1()
18         {
19             Assert.Pass();
20         }*/
21
22         [Test]
23         public void test_a_verse()
24         {
25             String expected = "99 bottles of beer on the wall, 99
26 bottles of beer.\n" +
27                 "Take one down and pass it around, 98 bottles of
28 beer on the wall.\n";
29             Assert.AreEqual(expected, _bottles.verse(99));
30         }
31
32         [Test]
33         public void test_another_verse()
34         {
35             String expected = "89 bottles of beer on the wall, 89
36 bottles of beer.\n" +
37                 "Take one down and pass it around, 88 bottles of
38 beer on the wall.\n";
39             Assert.AreEqual(expected, _bottles.verse(89));
40         }
41     }
```

```
37     [Test]
38     public void test_zero_verse()
39     {
40         String expected = "No more bottles of beer on the wall, no
more bottles of beer.\n" +
41             "Go to the store and buy some more, 99 bottles of
beer on the wall.\n";
42         Assert.AreEqual(expected, _bottles.verse(0));
43     }
44
45     [Test]
46     public void test_one_verse()
47     {
48         String expected = "1 bottle of beer on the wall, 1 bottle
of beer.\n" +
49             "Take it down and pass it around, no more bottles
of beer on the wall.\n";
50         Assert.AreEqual(expected, _bottles.verse(1));
51     }
52
53     [Test]
54     public void test_two_verse()
55     {
56         String expected = "2 bottles of beer on the wall, 2 bottles
of beer.\n" +
57             "Take one down and pass it around, 1 bottle of beer
on the wall.\n";
58         Assert.AreEqual(expected, _bottles.verse(2));
59     }
60
61     [Test]
62     public void test_a_couple_verses()
63     {
64         String expected = "99 bottles of beer on the wall, 99
bottles of beer.\n" +
```



```
65         "Take one down and pass it around, 98 bottles of
beer on the wall.\n" +
66
67         "98 bottles of beer on the wall, 98 bottles of beer
.\n" +
68         "Take one down and pass it around, 97 bottles of
beer on the wall.\n";
69         Assert.AreEqual(expected, _bottles.verses(99, 98));
70     }
71
72     [Test]
73     public void test_a_few_verses()
74     {
75         String expected = "2 bottles of beer on the wall, 2 bottles
of beer.\n" +
76         "Take one down and pass it around, 1 bottle of beer
on the wall.\n" +
77
78         "1 bottle of beer on the wall, 1 bottle of beer.\n"
+
79         "Take it down and pass it around, no more bottles
of beer on the wall.\n" +
80
81         "No more bottles of beer on the wall, no more
bottles of beer.\n" +
82         "Go to the store and buy some more, 99 bottles of
beer on the wall.\n";
83         Assert.AreEqual(expected, _bottles.verses(2, 0));
84     }
85
86     [Test]
87     public void test_the_whole_song()
88     {
89         String expected = _bottles.verses(99, 0);
90         Assert.AreEqual(expected, _bottles.song());
```

```
91     }  
92   }  
93 }
```

1.2 Possible Solutions

We can solve the problem by writing different solutions. As the author suggested four of those.

- Incomprehensibly Concise
- Speculatively General
- Concretely Abstract
- Shameless Green

Each of the aforementioned solutions needs to answer a set of questions to understand how good the solution is. From the comprehensibility perspective:

1. How many verse variants are there?
2. Which verses are most alike? In what way?
3. Which verses are most different? In what way?
4. What is the rule to determine which verse should be sung next?

From the maintenance perspective.

1. How difficult was this to write?
2. How hard is it to understand?
3. How expensive will it be to change?

1.2.1 Shameless Green

The below code will produce the shameless green solution of the code. As the author suggested the shameless green should be the initial solution before going for more abstraction.

```
1 public class Bottles {
2     public String song(){
3         return verses(99, 0);
4     }
5
6     public String verses(int upperBound, int lowerBound) {
7         String lyrics = "";
8         for (int i = upperBound; i >= lowerBound ; i--) {
9             lyrics += verse(i);
10        }
11        return lyrics;
12    }
13
14    public String verse(int lineNumber) {
15        String lyrics = "";
16        switch (lineNumber){
17            case 0:
18                lyrics += "No more bottles of beer on the wall, no more
19                bottles of beer.\n" +
20                "Go to the store and buy some more, 99 bottles
21                of beer on the wall.\n";
22                break;
23            case 1:
24                lyrics += "1 bottle of beer on the wall, 1 bottle of
25                beer.\n" +
26                "Take it down and pass it around, no more
27                bottles of beer on the wall.\n";
28                break;
29            case 2:
30                lyrics += "2 bottles of beer on the wall, 2 bottles of
31                beer.\n" +
```

```
27         "Take one down and pass it around, 1 bottle of  
beer on the wall.\n";  
28         break;  
29         case 89:  
30             lyrics += "89 bottles of beer on the wall, 89 bottles  
of beer.\n" +  
31                 "Take one down and pass it around, 88 bottles  
of beer on the wall.\n";  
32             break;  
33             case 98:  
34                 lyrics += "98 bottles of beer on the wall, 98 bottles  
of beer.\n" +  
35                     "Take one down and pass it around, 97 bottles  
of beer on the wall.\n";  
36                 break;  
37                 case 99:  
38                     lyrics += "99 bottles of beer on the wall, 99 bottles  
of beer.\n" +  
39                         "Take one down and pass it around, 98 bottles  
of beer on the wall.\n";  
40                     break;  
41             }  
42             return lyrics;  
43     }  
44 }
```

1.2.2 Speculatively General

The below code will produce the Speculatively General solution of the code. As the name suggested the below code is more concerned about future changes.

```
1 import java.util.ArrayList;  
2 import java.util.List;  
3  
4 interface Lyrics {  
5     String generate(Verse verse);
```

```
6 }
7
8 class NoMore implements Lyrics {
9     @Override
10    public String generate(Verse verse) {
11        return "No more bottles of beer on the wall, " +
12              "no more bottles of beer.\n" +
13              "Go to the store and buy some more, " +
14              "99 bottles of beer on the wall.\n";
15    }
16 }
17
18 class LastOne implements Lyrics {
19     @Override
20    public String generate(Verse verse) {
21        return "1 bottle of beer on the wall, " +
22              "1 bottle of beer.\n" +
23              "Take it down and pass it around, " +
24              "no more bottles of beer on the wall.\n";
25    }
26 }
27
28 class Penultimate implements Lyrics {
29     @Override
30    public String generate(Verse verse) {
31        return "2 bottles of beer on the wall, " +
32              "2 bottles of beer.\n" +
33              "Take one down and pass it around, " +
34              "1 bottle of beer on the wall.\n";
35    }
36 }
37
38 class Default implements Lyrics {
39     @Override
40    public String generate(Verse verse) {
```

```
41     return verse.getNumber() + " bottles of beer on the wall, " +
42         verse.getNumber() + " bottles of beer.\n" +
43         "Take one down and pass it around, " +
44         (verse.getNumber() - 1) + " bottles of beer on the wall
45     .\n";
46 }
47
48 class SpeculativelyGeneral {
49     String song() {
50         return verses(99, 0);
51     }
52
53     String verses(int finish, int start) {
54         List<String> verseList = new ArrayList<>();
55         for (int verseNumber = finish; verseNumber >= start;
56             verseNumber--) {
57             verseList.add(verse(verseNumber));
58         }
59         return String.join("", verseList);
60     }
61
62     String verse(int number) {
63         return verseFor(number).text();
64     }
65
66     Verse verseFor(int number) {
67         switch (number) {
68             case 0:
69                 return new Verse(number, new NoMore());
70             case 1:
71                 return new Verse(number, new LastOne());
72             case 2:
73                 return new Verse(number, new Penultimate());
74             default:
```

```
74         return new Verse(number, new Default());
75     }
76 }
77 }
78
79 class Verse {
80     private final int number;
81     private final Lyrics lyrics;
82
83     Verse(int number, Lyrics lyrics) {
84         this.number = number;
85         this.lyrics = lyrics;
86     }
87
88     int getNumber() {
89         return number;
90     }
91
92     String text() {
93         return lyrics.generate(this);
94     }
95 }
```

2 TASKS BASED ON PREVIOUS CLASSES

Imagine you are developing a Vehicle management system. Every vehicle uses different fuels namely petrol, diesel, or Gasoline. Vehicles can be driveable or rideable. For example, Buses, cars, etc., can be driveable while BiCycle is rideable. Every vehicle has start and stop functionalities. However, driveable vehicles can accelerate the engine as well.

A vehicle manager can manage any vehicle without knowing the actual vehicle.

1. You have to design a system by following OOC.
2. You have to use dynamic polymorphism to represent the system.

3. Write C# or Java code with Driver class to run and check the correctness of your program.

3 REFERENCES

- **99 Bottles of OOP** A Practical Guide to Object-Oriented Design, 2nd Edition, by Sandi Metz, Katrina Owen & TJ Stankus
- [Blog of jackhoy](#)