SWE 4301 - OBJECT ORIENTED CONCEPTS II

OOC-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

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:

```java
public class BottlesTest {

    private Bottles bottles = new Bottles();

    @Test
    void test_a_verse() {
        String expected = "99 bottles of beer on the wall, 99 bottles of beer.\n" +
                "Take one down and pass it around, 98 bottles of beer on the wall.\n";
        assertEquals(expected, this.bottles.verse(99));
    }

    @Test
    void test_another_verse() {
        String expected = "89 bottles of beer on the wall, 89 bottles of beer.\n" +
                "Take one down and pass it around, 88 bottles of beer on the wall.\n";
        assertEquals(expected, this.bottles.verse(89));
    }
    @Test
    void test_zero_verse() {
        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";
        assertEquals(expected, this.bottles.verse(0));
    }

    @Test
    void test_one_verse() {
        String expected =  "1 bottle of beer on the wall, 1 bottle of beer.\n" +
                "Take it down and pass it around, no more bottles of beer on the wall.\n";
        assertEquals(expected, this.bottles.verse(1));
    }

    @Test
    void test_two_verse() {
        String expected = "2 bottles of beer on the wall, 2 bottles of beer.\n" +
                "Take one down and pass it around, 1 bottle of beer on the wall.\n";
        assertEquals(expected, this.bottles.verse(2));
    }

    @Test
    void test_a_couple_verses() {
        String expected = "99 bottles of beer on the wall, 99 bottles of beer.\n" +
                "Take one down and pass it around, 98 bottles of beer on the wall.\n" +

                "98 bottles of beer on the wall, 98 bottles of beer.\n" +
                "Take one down and pass it around, 97 bottles of beer on the wall.\n";
        assertEquals(expected, this.bottles.verses(99, 98));
    }

    @Test
    void test_a_few_verses() {
        String expected = "2 bottles of beer on the wall, 2 bottles of beer.\n" +
                "Take one down and pass it around, 1 bottle of beer on the wall.\n" +

                "1 bottle of beer on the wall, 1 bottle of beer.\n" +
                "Take it down and pass it around, no more bottles of beer on the wall.\n" +

                "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";
        assertEquals(expected, this.bottles.verses(2, 0));
    }

    @Test
    void test_the_whole_song() {
        String expected = this.bottles.verses(99, 0);
        assertEquals(expected, this.bottles.song());
    }
}
```

1.1.2 C# Implementation of Test Cases:

Unit Testing for C# Blog

```
using _99Bottles;

namespace _99BottlesTest
{
    public class Tests
    {
        Bottles _bottles;

        [SetUp]
        public void Setup()
        {
            _bottles = new Bottles();
        }
/*
        [Test]
        public void Test1()
        {
            Assert.Pass();
        }*/

        [Test]
        public void test_a_verse()
        {
            String expected = "99 bottles of beer on the wall, 99 bottles of beer.\n" +
                         "Take one down and pass it around, 98 bottles of beer on the wall.\n";
            Assert.AreEqual(expected, _bottles.verse(99));
        }

        [Test]
        public void test_another_verse()
        {
            String expected = "89 bottles of beer on the wall, 89 bottles of beer.\n" +
                         "Take one down and pass it around, 88 bottles of beer on the wall.\n";
            Assert.AreEqual(expected, _bottles.verse(89));
        }

        [Test]
        public void test_zero_verse()
        {
            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";
            Assert.AreEqual(expected, _bottles.verse(0));
        }

        [Test]
        public void test_one_verse()
        {
            String expected = "1 bottle of beer on the wall, 1 bottle of beer.\n" +
                    "Take it down and pass it around, no more bottles of beer on the wall.\n";
            Assert.AreEqual(expected, _bottles.verse(1));
        }

        [Test]
        public void test_two_verse()
        {
            String expected = "2 bottles of beer on the wall, 2 bottles of beer.\n" +
                    "Take one down and pass it around, 1 bottle of beer on the wall.\n";
            Assert.AreEqual(expected, _bottles.verse(2));
        }

        [Test]
        public void test_a_couple_verses()
        {
            String expected = "99 bottles of beer on the wall, 99 bottles of beer.\n" +
                    "Take one down and pass it around, 98 bottles of beer on the wall.\n" +

                    "98 bottles of beer on the wall, 98 bottles of beer.\n" +
                    "Take one down and pass it around, 97 bottles of beer on the wall.\n";
            Assert.AreEqual(expected, _bottles.verses(99, 98));
        }

        [Test]
        public void test_a_few_verses()
        {
            String expected = "2 bottles of beer on the wall, 2 bottles of beer.\n" +
                    "Take one down and pass it around, 1 bottle of beer on the wall.\n" +

                    "1 bottle of beer on the wall, 1 bottle of beer.\n" +
                    "Take it down and pass it around, no more bottles of beer on the wall.\n" +

                    "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";
            Assert.AreEqual(expected, _bottles.verses(2, 0));
        }

        [Test]
        public void test_the_whole_song()
        {
            String expected = _bottles.verses(99, 0);
            Assert.AreEqual(expected, _bottles.song());
```

```
            }
        }
}
```

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.

```java
public class Bottles {
    public String song(){
        return verses(99, 0);
    }

    public String verses(int upperBound, int lowerBound) {
        String lyrics = "";
        for (int i = upperBound; i >= lowerBound ; i--) {
            lyrics += verse(i);
        }
        return lyrics;
    }

    public String verse(int lineNumber) {
        String lyrics = "";
        switch (lineNumber){
            case 0:
                lyrics += "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";
                break;
            case 1:
                lyrics +=  "1 bottle of beer on the wall, 1 bottle of beer.\n" +
                        "Take it down and pass it around, no more bottles of beer on the wall.\n";
                break;
            case 2:
                lyrics += "2 bottles of beer on the wall, 2 bottles of beer.\n" +
                        "Take one down and pass it around, 1 bottle of beer on the wall.\n";
                break;
            case 89:
                lyrics += "89 bottles of beer on the wall, 89 bottles of beer.\n" +
                        "Take one down and pass it around, 88 bottles of beer on the wall.\n";
                break;
            case 98:
                lyrics += "98 bottles of beer on the wall, 98 bottles of beer.\n" +
                        "Take one down and pass it around, 97 bottles of beer on the wall.\n";
                break;
            case 99:
                lyrics += "99 bottles of beer on the wall, 99 bottles of beer.\n" +
                        "Take one down and pass it around, 98 bottles of beer on the wall.\n";
                break;
        }
        return lyrics;
    }
}
```

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.

```
import java.util.ArrayList;
import java.util.List;

interface Lyrics {
    String generate(Verse verse);
```

```java
}

class NoMore implements Lyrics {
    @Override
    public String generate(Verse verse) {
        return "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";
    }
}

class LastOne implements Lyrics {
    @Override
    public String generate(Verse verse) {
        return "1 bottle of beer on the wall, " +
                "1 bottle of beer.\n" +
                "Take it down and pass it around, " +
                "no more bottles of beer on the wall.\n";
    }
}

class Penultimate implements Lyrics {
    @Override
    public String generate(Verse verse) {
        return "2 bottles of beer on the wall, " +
                "2 bottles of beer.\n" +
                "Take one down and pass it around, " +
                "1 bottle of beer on the wall.\n";
    }
}

class Default implements Lyrics {
    @Override
    public String generate(Verse verse) {
        return verse.getNumber() + " bottles of beer on the wall, " +
                verse.getNumber() + " bottles of beer.\n" +
                "Take one down and pass it around, " +
                (verse.getNumber() - 1) + " bottles of beer on the wall.\n";
    }
}

class SpeculativelyGeneral {
    String song() {
        return verses(99, 0);
    }

    String verses(int finish, int start) {
        List<String> verseList = new ArrayList<>();
        for (int verseNumber = finish; verseNumber >= start; verseNumber--) {
            verseList.add(verse(verseNumber));
        }
        return String.join("", verseList);
    }

    String verse(int number) {
        return verseFor(number).text();
    }

    Verse verseFor(int number) {
        switch (number) {
            case 0:
                return new Verse(number, new NoMore());
            case 1:
                return new Verse(number, new LastOne());
            case 2:
                return new Verse(number, new Penultimate());
            default:
                return new Verse(number, new Default());
            }
        }
}

class Verse {
    private final int number;
    private final Lyrics lyrics;

    Verse(int number, Lyrics lyrics) {
        this.number = number;
        this.lyrics = lyrics;
    }

    int getNumber() {
        return number;
    }

    String text() {
        return lyrics.generate(this);
    }
}
```

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