

**70094 15**

**Re-use and Extensibility**

## **Submitters**

---


**wss119**  
**sf23**

**Alva Si**  
**Shihan Fu**

# **Emarking**

**Final Tests****TestSummary.txt: 1/1****Alva Si - wss119:v5**

```
1: Final Tests: Summary for wss119 of v5
2: -----
3:
4:   Public Tests:
5:     Compiles:           1 / 1
6:     Tests Pass:         1 / 1
7:     Coverage and Style Checks: 1 / 1
8:
9: Git Repo: git@gitlab.doc.ic.ac.uk:lab2324_spring/SE_Design_Ex5_wss119.git
10: Commit ID: a06a8
```



14/20 - Overall good implementation of the design patterns. See comments below for details.

There is some duplication in your tests, I suggest you compare your approach with the suggested solution.

```

1: package ic.doc.templateMethod;
2:
3: import static ic.doc.matchers.IterableBeginsWith.beginsWith;
4: import static org.hamcrest.CoreMatchers.containsString;
5: import static org.hamcrest.MatcherAssert.assertThat;
6: import static org.hamcrest.core.Is.is;
7: import static org.junit.Assert.fail;
8:
9: import org.junit.Test;
10:
11: public class TriangleNumbersSequenceTest {
12:
13:     final TriangleNumbersSequence sequence = new TriangleNumbersSequence();
14:
15:     @Test
16:     public void definesFirstTermToBeOne() {
17:         assertThat(sequence.term(0), is(1));
18:     }
19:
20:     @Test
21:     public void definesTermToBeSumOfDotsInEquilateralTriangle() {
22:         assertThat(sequence.term(2), is(6));
23:         assertThat(sequence.term(3), is(10));
24:         assertThat(sequence.term(4), is(15));
25:     }
26:
27:     @Test
28:     public void isUndefinedForNegativeIndices() {
29:         try {
30:             sequence.term(-1);
31:             fail("should have thrown exception");
32:         } catch (IllegalArgumentException e) {
33:             assertThat(e.getMessage(), containsString("Not defined for indices < 0"));
34:         }
35:     }
36:
37:     @Test
38:     public void canBeIteratedThrough() {
39:         assertThat(sequence, beginsWith(1, 3, 6, 10, 15));
40:     }
41: }

```

Be mindful of duplication in the tests. You should reason on how many times each part gets tested.

Some of the tests are inherited (and executed twice). A better solution would have been having a (non-abstract) test class for the parent Sequence, which you instantiate with an inline dummy implementation of term(i). You can use it to test common methods

```

1: package ic.doc.templateMethod;
2:
3: import static ic.doc.matchers.IterableBeginsWith.beginsWith;
4: import static org.hamcrest.CoreMatchers.containsString;
5: import static org.hamcrest.MatcherAssert.assertThat;
6: import static org.hamcrest.core.Is.is;
7: import static org.junit.Assert.fail;
8:
9: import org.junit.Test;
10:
11: public class FibonacciSequenceTest {
12:
13:     final FibonacciSequence sequence = new FibonacciSequence();
14:
15:     @Test
16:     public void definesFirstTwoTermsToBeOne() {
17:         assertThat(sequence.term(0), is(1));
18:         assertThat(sequence.term(1), is(1));
19:     }
20:
21:     @Test
22:     public void definesSubsequentTermsToBeTheSumOfThePreviousTwo() {
23:         assertThat(sequence.term(2), is(2));
24:         assertThat(sequence.term(3), is(3));
25:         assertThat(sequence.term(4), is(5));
26:     }
27:
28:     @Test
29:     public void isUndefinedForNegativeIndices() {
30:         try {
31:             sequence.term(-1);
32:             fail("should have thrown exception");
33:         } catch (IllegalArgumentException e) {
34:             assertThat(e.getMessage(), containsString("Not defined for indices < 0"));
35:         }
36:     }
37:
38:     @Test
39:     public void canBeIteratedThrough() {
40:         assertThat(sequence, beginsWith(1, 1, 2, 3, 5));
41:     }
42: }

```

```

1: package ic.doc.strategy;
2:
3: import static ic.doc.matchers.IterableBeginsWith.startsWith;
4: import static org.hamcrest.MatcherAssert.assertThat;
5:
6: import org.junit.Test;
7:
8: public class TriangleNumbersSequenceTest extends BaseSequenceTest {
9:
10:     final Sequence sequence = new Sequence(new TriangleNumbersSequence());
11:
12:     @Override
13:     protected Sequence getSequence() {
14:         return sequence;
15:     }
16:
17:     @Test
18:     public void definesFirstTermToBeOne() {
19:
20:         assertEquals(0, 1);
21:     }
22:
23:     @Test
24:     public void definesTermToBeSumOfDotsInEquilateralTriangle() {
25:
26:         assertEquals(2, 6);
27:         assertEquals(3, 10);
28:         assertEquals(4, 15);
29:     }
30:
31:     @Test
32:     public void canBeIteratedThrough() {
33:         assertThat(sequence, beginsWith(1, 3, 6, 10, 15));
34:     }
35: }

```

```

1: package ic.doc.strategy;
2:
3: import static ic.doc.matchers.IterableBeginsWith.startsWith;
4: import static org.hamcrest.MatcherAssert.assertThat;
5:
6: import org.junit.Test;
7:
8: public class FibonacciSequenceTest extends BaseSequenceTest {
9:
10:     final Sequence sequence = new Sequence(new FibonacciSequence());
11:
12:     @Override
13:     protected Sequence getSequence() {
14:         return sequence;
15:     }
16:
17:     @Test
18:     public void definesFirstTwoTermsToBeOne() {
19:
20:         assertEquals(0, 1);
21:         assertEquals(1, 1);
22:     }
23:
24:     @Test
25:     public void definesSubsequentTermsToBeTheSumOfThePreviousTwo() {
26:
27:         assertEquals(2, 2);
28:         assertEquals(3, 3);
29:         assertEquals(4, 5);
30:     }
31:
32:     @Test
33:     public void canBeIteratedThrough() {
34:         assertThat(sequence, beginsWith(1, 1, 2, 3, 5));
35:     }
36: }

```

It looks like "sequence" is tested twice.

## Final Tests

BaseSequenceTest.java: 1/1

Alva Si - wss119:v5

```
1: package ic.doc.strategy;
2:
3: import static org.hamcrest.CoreMatchers.containsString;
4: import static org.hamcrest.MatcherAssert.assertThat;
5: import static org.hamcrest.core.Is.is;
6: import static org.junit.Assert.fail;
7:
8: import org.junit.Test;
9:
10: public abstract class BaseSequenceTest {
11:     protected abstract Sequence getSequence();
12:
13:     protected void assertTermEquals(int index, int expectedValue) {
14:         assertThat(getSequence().term(index), is(expectedValue));
15:     }
16:
17:     @Test
18:     public void isUndefinedForNegativeIndices() {
19:
20:         try {
21:             getSequence().term(-1);
22:             fail("should have thrown exception");
23:         } catch (IllegalArgumentException e) {
24:             assertThat(e.getMessage(), containsString("Not defined for indices < 0"));
25:         }
26:     }
27: }
```

This looks a bit convoluted. In this case, your objective is to reduce duplication of execution (sequence should be tested once). A better solution would be having the following test classes:

- SequenceTest instantiates a Sequence object with a dummy strategy to test methods that are common to both sequences (e.g., canBeIteratedThrough, UndefinedForNegativeIndices).
- A test class for your implementation of the triangle formula.
- A test class for your implementation of the fibonacci formula.

## Final Tests

TriangleNumbersSequence.java: 1/1

Alva Si - wss119:v5

```
1: package ic.doc.templatemethod;
2:
3: public class TriangleNumbersSequence extends Sequence {
4:
5:     @Override
6:     public int term(int i) {
7:         validateIndex(i);
8:         return ((i + 1) * (i + 2)) / 2;
9:     }
10: }
```



## Final Tests

Sequence.java: 1/1

Alva Si - wss119:v5

```
1: package ic.doc.templatemethod;
2:
3: import java.util.Iterator;
4:
5: public abstract class Sequence implements Iterable<Integer> {
6:     public abstract int term(int i);
7:
8:     protected void validateIndex(int i) {
9:         if (i < 0) {
10:             throw new IllegalArgumentException("Not defined for indices < 0");
11:         }
12:     }
13:
14:     public Iterator<Integer> iterator() {
15:         return new SequenceIterator();
16:     }
17:
18:     private class SequenceIterator implements Iterator<Integer> {
19:
20:         private int index = 0;
21:
22:         @Override
23:         public boolean hasNext() {
24:             return true;
25:         }
26:
27:         @Override
28:         public Integer next() {
29:             return term(index++);
30:         }
31:
32:         @Override
33:         public void remove() {
34:             throw new UnsupportedOperationException("remove is not implemented");
35:         }
36:     }
37: }
```



## Final Tests

FibonacciSequence.java: 1/1

Alva Si - wss119:v5

```
1: package ic.doc.templatemethod;
2:
3: public class FibonacciSequence extends Sequence {
4:
5:     @Override
6:     public int term(int i) {
7:         validateIndex(i);
8:         if (i < 2) {
9:             return 1;
10:        }
11:        return term(i - 1) + term(i - 2);
12:    }
13: }
```



Final Tests

TriangleNumbersSequence.java: 1/1

Alva Si - wss119:v5

```
1: package ic.doc.strategy;
2:
3: public class TriangleNumbersSequence implements Strategy {
4:     @Override
5:     public int term(int i) {
6:         validateIndex(i);
7:         return ((i + 1) * (i + 2)) / 2;
8:     }
9: }
```

This is not really a sequence. I would rather call this class "TriangleTermGenerator".

Final Tests

Strategy.java: 1/1

Alva Si - wss119:v5

```
1: package ic.doc.strategy;
2:
3: public interface Strategy {
4:     int term(int i);
5:
6:     default void validateIndex(int i) {
7:         if (i < 0) {
8:             throw new IllegalArgumentException("Not defined for indices < 0");
9:         }
10:     }
11: }
```

Be mindful of how you name different components. You should think about the role that each object plays - but avoid using pattern names or "interface" in their type names

## Final Tests

Sequence.java: 1/1

Alva Si - wss119:v5

```
1: package ic.doc.strategy;
2:
3: import java.util.Iterator;
4:
5: public class Sequence implements Iterable<Integer> {
6:
7:     private final Strategy strategy;
8:
9:     public Sequence(Strategy strategy) {
10:         this.strategy = strategy;
11:     }
12:
13:     public int term(int i) {
14:         return strategy.term(i);
15:     }
16:
17:     public Iterator<Integer> iterator() {
18:         return new SequenceIterator();
19:     }
20:
21:     private class SequenceIterator implements Iterator<Integer> {
22:
23:         private int index = 0;
24:
25:         @Override
26:         public boolean hasNext() {
27:             return true;
28:         }
29:
30:         @Override
31:         public Integer next() {
32:             return term(index++);
33:         }
34:
35:         @Override
36:         public void remove() {
37:             throw new UnsupportedOperationException("remove is not implemented");
38:         }
39:     }
40: }
```

This reads rather ambiguous. Try choose different names.

## Final Tests

FibonacciSequence.java: 1/1

Alva Si - wss119:v5

```
1: package ic.doc.strategy;
2:
3: public class FibonacciSequence implements Strategy {
4:     @Override
5:     public int term(int i) {
6:         validateIndex(i);
7:         if (i < 2) {
8:             return 1;
9:         }
10:         return term(i - 1) + term(i - 2);
11:     }
12: }
```



## Final Tests

testResults.txt: 1/2

Alva Si - wss119:v5

```
1: ----- Test Output -----
2: Running LabTS build... (Wed 14 Feb 13:01:20 UTC 2024)
3:
4: Submission summary...
5: You made 8 commits
6:   - f43d7a5 tests: modified fibonacci sequence source code and tests to match triangle numbers sequence spec [4 files changed, 168 insertions]
7:   - d466419 refactor: extracted method from Fibonacci and Triangle to superclass Sequence [3 files changed, 42 insertions, 61 deletions]
8:   - ca5efdd refactor: extracted method from Fibonacci and Triangle to superclass Sequence [9 files changed, 97 insertions, 127 deletions]
9:   - b74ef02 Merge remote-tracking branch 'origin/master' [Merge remote-tracking branch 'origin/master']
10:  - e699343 refactor: removed redundant if statement in TriangleNumberSequence [1 file changed, 3 deletions]
11:  - 67acd7 refactor: extracted throw new illegal argument exception to sequence superclass [4 files changed, 9 insertions, 8 deletions]
12:  - 7141be6 refactor: extracted throw new illegal argument exception to strategy interface [3 files changed, 8 insertions, 6 deletions]
13:  - a06a896 refactor: extracted negative indices test to superclass BaseSequenceTest [3 files changed, 48 insertions, 39 deletions]
14:
15: Preparing...
16:
17: BUILD SUCCESSFUL in 568ms
18:
19: Compiling...
20: BUILD SUCCESSFUL in 5s
21:
22: Running tests...
23:
24: ic.doc.strategy.FibonacciSequenceTest > canBeIteratedThrough PASSED
25:
26: ic.doc.strategy.FibonacciSequenceTest > definesSubsequentTermsToBeTheSumOfThePreviousTwo PASSED
27:
28: ic.doc.strategy.FibonacciSequenceTest > definesFirstTwoTermsToBeOne PASSED
29:
30: ic.doc.strategy.FibonacciSequenceTest > isUndefinedForNegativeIndices PASSED
31:
32: ic.doc.strategy.TriangleNumbersSequenceTest > canBeIteratedThrough PASSED
33:
34: ic.doc.strategy.TriangleNumbersSequenceTest > definesTermToBeSumOfDotsInEquilateralTriangle PASSED
35:
36: ic.doc.strategy.TriangleNumbersSequenceTest > definesFirstTermToBeOne PASSED
37:
38: ic.doc.strategy.TriangleNumbersSequenceTest > isUndefinedForNegativeIndices PASSED
39:
40: ic.doc.templatemethod.FibonacciSequenceTest > canBeIteratedThrough PASSED
41:
42: ic.doc.templatemethod.FibonacciSequenceTest > definesSubsequentTermsToBeTheSumOfThePreviousTwo PASSED
43:
44: ic.doc.templatemethod.FibonacciSequenceTest > isUndefinedForNegativeIndices PASSED
45:
46: ic.doc.templatemethod.FibonacciSequenceTest > definesFirstTwoTermsToBeOne PASSED
47:
48: ic.doc.templatemethod.TriangleNumbersSequenceTest > canBeIteratedThrough PASSED
49:
50: ic.doc.templatemethod.TriangleNumbersSequenceTest > definesTermToBeSumOfDotsInEquilateralTriangle PASSED
51:
52: ic.doc.templatemethod.TriangleNumbersSequenceTest > definesFirstTermToBeOne PASSED
53:
54: ic.doc.templatemethod.TriangleNumbersSequenceTest > isUndefinedForNegativeIndices PASSED
55:
56: BUILD SUCCESSFUL in 2s
57:
58: Checking test coverage and code style...
59: BUILD SUCCESSFUL in 5s
60: Finished auto test. (Wed 14 Feb 13:01:45 UTC 2024)
61:
```



## Final Tests

testResults.txt: 2/2

Alva Si - wss119:v5

```
62: ----- Test Errors -----  
63:
```