70094 15

Re-use and Extensibility

Submitters

wss119	Alva Si
sf23	Shihan Fu

```
Final Tests: Summary for wss119 of v5
-------------------------------------

  Public Tests:
    Compiles:                  1 / 1
    Tests Pass:                1 / 1
    Coverage and Style Checks: 1 / 1

Git Repo: git@gitlab.doc.ic.ac.uk:lab2324_spring/SE_Design_Ex5_wss119.git
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.

```java
package ic.doc.templatemethod;

import static ic.doc.matchers.IterableBeginsWith.beginsWith;
import static org.hamcrest.CoreMatchers.containsString;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.core.Is.is;
import static org.junit.Assert.fail;

import org.junit.Test;

public class TriangleNumbersSequenceTest {

  final TriangleNumbersSequence sequence = new TriangleNumbersSequence();

  @Test
  public void definesFirstTermToBeOne() {

    assertThat(sequence.term(0), is(1));
  }

  @Test
  public void definesTermToBeSumOfDotsInEquilateralTriangle() {

    assertThat(sequence.term(2), is(6));
    assertThat(sequence.term(3), is(10));
    assertThat(sequence.term(4), is(15));
  }

  @Test
  public void isUndefinedForNegativeIndices() {

    try {
      sequence.term(-1);
      fail("should have thrown exception");
    } catch (IllegalArgumentException e) {
      assertThat(e.getMessage(), containsString("Not defined for indices < 0"));
    }
  }

  @Test
  public void canBeIteratedThrough() {
    assertThat(sequence, beginsWith(1, 3, 6, 10, 15));
  }
}
```



```java
package ic.doc.templatemethod;

import static ic.doc.matchers.IterableBeginsWith.beginsWith;
import static org.hamcrest.CoreMatchers.containsString;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.core.Is.is;
import static org.junit.Assert.fail;

import org.junit.Test;

public class FibonacciSequenceTest {

  final FibonacciSequence sequence = new FibonacciSequence();

  @Test
  public void definesFirstTwoTermsToBeOne() {

    assertThat(sequence.term(0), is(1));
    assertThat(sequence.term(1), is(1));
  }

  @Test
  public void definesSubsequentTermsToBeTheSumOfThePreviousTwo() {

    assertThat(sequence.term(2), is(2));
    assertThat(sequence.term(3), is(3));
    assertThat(sequence.term(4), is(5));
  }

  @Test
  public void isUndefinedForNegativeIndices() {

    try {
      sequence.term(-1);
      fail("should have thrown exception");
    } catch (IllegalArgumentException e) {
      assertThat(e.getMessage(), containsString("Not defined for indices < 0"));
    }
  }

  @Test
  public void canBeIteratedThrough() {
    assertThat(sequence, beginsWith(1, 1, 2, 3, 5));
  }
}
```

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

git@gitlab.doc.ic.ac.uk:lab2324_spring/SE_Design_Ex5_wss119.git a06a8

```java
package ic.doc.strategy;

import static ic.doc.matchers.IterableBeginsWith.beginsWith;
import static org.hamcrest.MatcherAssert.assertThat;

import org.junit.Test;

public class TriangleNumbersSequenceTest extends BaseSequenceTest {

  final Sequence sequence = new Sequence(new TriangleNumbersSequence());

  @Override
  protected Sequence getSequence() {
    return sequence;
  }

  @Test
  public void definesFirstTermToBeOne() {

    assertTermEquals(0, 1);
  }

  @Test
  public void definesTermToBeSumOfDotsInEquilateralTriangle() {

    assertTermEquals(2, 6);
    assertTermEquals(3, 10);
    assertTermEquals(4, 15);
  }

  @Test
  public void canBeIteratedThrough() {
    assertThat(sequence, beginsWith(1, 3, 6, 10, 15));
  }
}
```



```java
package ic.doc.strategy;

import static ic.doc.matchers.IterableBeginsWith.beginsWith;
import static org.hamcrest.MatcherAssert.assertThat;

import org.junit.Test;

public class FibonacciSequenceTest extends BaseSequenceTest {

  final Sequence sequence = new Sequence(new FibonacciSequence());

  @Override
  protected Sequence getSequence() {
    return sequence;
  }

  @Test
  public void definesFirstTwoTermsToBeOne() {

    assertTermEquals(0, 1);
    assertTermEquals(1, 1);
  }

  @Test
  public void definesSubsequentTermsToBeTheSumOfThePreviousTwo() {

    assertTermEquals(2, 2);
    assertTermEquals(3, 3);
    assertTermEquals(4, 5);
  }

  @Test
  public void canBeIteratedThrough() {
    assertThat(sequence, beginsWith(1, 1, 2, 3, 5));
  }
}
```

It looks like “sequence” is tested twice.

```java
package ic.doc.strategy;

import static org.hamcrest.CoreMatchers.containsString;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.core.Is.is;
import static org.junit.Assert.fail;

import org.junit.Test;

public abstract class BaseSequenceTest {
  protected abstract Sequence getSequence();

  protected void assertTermEquals(int index, int expectedValue) {
    assertThat(getSequence().term(index), is(expectedValue));
  }

  @Test
  public void isUndefinedForNegativeIndices() {

    try {
      getSequence().term(-1);
      fail("should have thrown exception");
    } catch (IllegalArgumentException e) {
      assertThat(e.getMessage(), containsString("Not defined for indices < 0"));
    }
  }
}
```

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.


```java
package ic.doc.templatemethod;

public class TriangleNumbersSequence extends Sequence {

  @Override
  public int term(int i) {
    validateIndex(i);
    return ((i + 1) * (i + 2)) / 2;
  }
}
```

✔

```java
package ic.doc.templatemethod;

import java.util.Iterator;

public abstract class Sequence implements Iterable<Integer> {
  public abstract int term(int i);

  protected void validateIndex(int i) {
    if (i < 0) {
      throw new IllegalArgumentException("Not defined for indices < 0");
    }
  }

  public Iterator<Integer> iterator() {
    return new SequenceIterator();
  }

  private class SequenceIterator implements Iterator<Integer> {

    private int index = 0;

    @Override
    public boolean hasNext() {
      return true;
    }

    @Override
    public Integer next() {
      return term(index++);
    }

    @Override
    public void remove() {
      throw new UnsupportedOperationException("remove is not implemented");
    }
  }
}
```

✔


```java
package ic.doc.templatemethod;

public class FibonacciSequence extends Sequence {

  @Override
  public int term(int i) {
    validateIndex(i);
    if (i < 2) {
      return 1;
    }
    return term(i - 1) + term(i - 2);
  }
}
```

✔

```java
package ic.doc.strategy;

public class TriangleNumbersSequence implements Strategy {
  @Override
  public int term(int i) {
    validateIndex(i);
    return ((i + 1) * (i + 2)) / 2;
  }
}
```

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


```java
package ic.doc.strategy;

public interface Strategy {
  int term(int i);

  default void validateIndex(int i) {
    if (i < 0) {
      throw new IllegalArgumentException("Not defined for indices < 0");
    }
  }
}
```

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

```java
package ic.doc.strategy;

import java.util.Iterator;

public class Sequence implements Iterable<Integer> {

  private final Strategy strategy;

  public Sequence(Strategy strategy) {
    this.strategy = strategy;
  }

  public int term(int i) {
    return strategy.term(i);
  }

  public Iterator<Integer> iterator() {
    return new SequenceIterator();
  }

  private class SequenceIterator implements Iterator<Integer> {

    private int index = 0;

    @Override
    public boolean hasNext() {
      return true;
    }

    @Override
    public Integer next() {
      return term(index++);
    }

    @Override
    public void remove() {
      throw new UnsupportedOperationException("remove is not implemented");
    }
  }
}
```

This reads rather ambiguous. Try choose different names.


```java
package ic.doc.strategy;

public class FibonacciSequence implements Strategy {
  @Override
  public int term(int i) {
    validateIndex(i);
    if (i < 2) {
      return 1;
    }
    return term(i - 1) + term(i - 2);
  }
}
```

```
-------- Test Output --------
Running LabTS build... (Wed 14 Feb 13:01:20 UTC 2024)

Submission summary...
You made 8 commits
  - f43d7a5 tests: modified fibonacci sequence source code and tests to match triangle numbers sequence spec [4 files changed, 168 insertions]
  - d466419 refactor: extracted method from Fibonacci and Triangle to superclass Sequence [3 files changed, 42 insertions, 61 deletions]
  - ca5efdd refactor: extracted method from Fibonacci and Triangle to superclass Sequence [9 files changed, 97 insertions, 127 deletions]
  - b74ef02 Merge remote-tracking branch 'origin/master' [Merge remote-tracking branch 'origin/master']
  - e699343 refactor: removed redundant if statement in TriangleNumberSequence [1 file changed, 3 deletions]
  - 67acdd7 refactor: extracted throw new illegal argument exception to sequence superclass [4 files changed, 9 insertions, 8 deletions]
  - 7141be6 refactor: extracted throw new illegal argument exception to strategy interface [3 files changed, 8 insertions, 6 deletions]
  - a06a896 refactor: extracted negative indices test to superclass BaseSequenceTest [3 files changed, 48 insertions, 39 deletions]

Preparing...

BUILD SUCCESSFUL in 568ms

Compiling...
BUILD SUCCESSFUL in 5s

Running tests...

ic.doc.strategy.FibonacciSequenceTest > canBeIteratedThrough PASSED

ic.doc.strategy.FibonacciSequenceTest > definesSubsequentTermsToBeTheSumOfThePreviousTwo PASSED

ic.doc.strategy.FibonacciSequenceTest > definesFirstTwoTermsToBeOne PASSED

ic.doc.strategy.FibonacciSequenceTest > isUndefinedForNegativeIndices PASSED

ic.doc.strategy.TriangleNumbersSequenceTest > canBeIteratedThrough PASSED

ic.doc.strategy.TriangleNumbersSequenceTest > definesTermToBeSumOfDotsInEquilateralTriangle PASSED

ic.doc.strategy.TriangleNumbersSequenceTest > definesFirstTermToBeOne PASSED

ic.doc.strategy.TriangleNumbersSequenceTest > isUndefinedForNegativeIndices PASSED

ic.doc.templatemethod.FibonacciSequenceTest > canBeIteratedThrough PASSED

ic.doc.templatemethod.FibonacciSequenceTest > definesSubsequentTermsToBeTheSumOfThePreviousTwo PASSED

ic.doc.templatemethod.FibonacciSequenceTest > isUndefinedForNegativeIndices PASSED

ic.doc.templatemethod.FibonacciSequenceTest > definesFirstTwoTermsToBeOne PASSED

ic.doc.templatemethod.TriangleNumbersSequenceTest > canBeIteratedThrough PASSED

ic.doc.templatemethod.TriangleNumbersSequenceTest > definesTermToBeSumOfDotsInEquilateralTriangle PASSED

ic.doc.templatemethod.TriangleNumbersSequenceTest > definesFirstTermToBeOne PASSED

ic.doc.templatemethod.TriangleNumbersSequenceTest > isUndefinedForNegativeIndices PASSED

BUILD SUCCESSFUL in 2s

Checking test coverage and code style...
BUILD SUCCESSFUL in 5s
Finished auto test. (Wed 14 Feb 13:01:45 UTC 2024)
```

```
-------- Test Errors --------

```