```java
package uk.co.littlestickyleaves;

import org.junit.Before;
import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.ExpectedException;

import static org.hamcrest.core.IsEqual.equalTo;
import static org.mockito.Mockito.doThrow;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.never;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.verifyNoMoreInteractions;
import static org.mockito.Mockito.when;

public class LogicianTest {
    private static final String TEST_INPUT = "testInput";

    @Rule
    public ExpectedException myExpectedException = ExpectedException.none();

    private ScoreProvider mockScoreProvider = mock(ScoreProvider.class);
    private Persister mockPersister = mock(Persister.class);
    private Logician testObject;

    @Before
    public void setUp() {
        testObject = new Logician(mockScoreProvider, mockPersister);
    }

    @Test
    public void testLowScoreNothingStored() {
        // arrange
        when(mockScoreProvider.fetchScore(TEST_INPUT)).thenReturn(3);

        // act
        testObject.handleString(TEST_INPUT);

        // assert
        verify(mockPersister, never()).store(TEST_INPUT);
        /// these two lines are less important
        verify(mockScoreProvider).fetchScore(TEST_INPUT);
        verifyNoMoreInteractions(mockScoreProvider, mockPersister);
    }

    @Test
    public void testHighScoreSomethingStored() {
        // arrange
        when(mockScoreProvider.fetchScore(TEST_INPUT)).thenReturn(24);

        // act
        testObject.handleString(TEST_INPUT);

        // assert
        verify(mockPersister).store(TEST_INPUT);
        verify(mockScoreProvider).fetchScore(TEST_INPUT);
        verifyNoMoreInteractions(mockScoreProvider, mockPersister);
```

```
    }

    @Test
    public void testScoreProviderExceptionHandledNothingStored() {
        // arrange
        IllegalArgumentException illegalArgExc = new IllegalArgumentException("Too many
        vowels");
        when(mockScoreProvider.fetchScore(TEST_INPUT)).thenThrow(illegalArgExc);

        doThrow(new IllegalArgumentException()).when(mockPersister).store("three");

        myExpectedException.expect(LogicianException.class);
        myExpectedException.expectCause(equalTo(illegalArgExc));
        myExpectedException.expectMessage("Input 'testInput' cannot produce score");

        // act
        testObject.handleString(TEST_INPUT);

        // assert
        verify(mockPersister, never()).store(TEST_INPUT);
        verify(mockScoreProvider).fetchScore(TEST_INPUT);
        verifyNoMoreInteractions(mockScoreProvider, mockPersister);
    }
}
```

```
public class Logician {

    private ScoreProvider myScoreProvider;

    private Persister myPersister;

    public Logician(ScoreProvider scoreProvider, Persister persister) {
        myScoreProvider = scoreProvider;
        myPersister = persister;
    }

    public void handleString(String input) {
        int score;
        try {
            score = myScoreProvider.fetchScore(input);
        } catch (IllegalArgumentException e) {
            throw new LogicianException("Input '" + input + "' cannot produce score", e);
        }
        if (score > 20) {
            myPersister.store(input);
        }
    }
}
```