```java
import static org.junit.Assert.assertEquals;

/**
 * JUnit test fixture for {@code Statement}'s constructor and kernel methods.
 *
 * @author Chloe Feller and Krish Patel
 *
 */
public abstract class StatementTest {

    /**
     * The name of a file containing a sequence of BL statements.
     */
    private static final String FILE_NAME_1 = "test/statement/statement1.bl",
            FILE_NAME_2 = "test/statement/statement2.bl",
            FILE_NAME_3 = "test/statement/statement3.bl",
            FILE_NAME_4 = "test/statement/statement4.bl";

    /**
     * Invokes the {@code Statement} constructor for the implementation under
     * test and returns the result.
     *
     * @return the new statement
     * @ensures constructorTest = compose((BLOCK, ?, ?), <>)
     */
    protected abstract Statement constructorTest();

    /**
     * Invokes the {@code Statement} constructor for the reference
     * implementation and returns the result.
     *
     * @return the new statement
     * @ensures constructorRef = compose((BLOCK, ?, ?), <>)
     */
    protected abstract Statement constructorRef();

    /**
     * Test of parse on syntactically valid input.
     */
    @Test
    public final void testParseValidExample() {
        /*
         * Setup
         */
        Statement sRef = this.constructorRef();
        SimpleReader file = new SimpleReader1L(FILE_NAME_1);
        Queue<String> tokens = Tokenizer.tokens(file);
        sRef.parse(tokens);
        file.close();
        Statement sTest = this.constructorTest();
        file = new SimpleReader1L(FILE_NAME_1);
        tokens = Tokenizer.tokens(file);
        file.close();
        /*
         * The call
         */
        sTest.parse(tokens);
```

```java
        /*
         * Evaluation
         */
        assertEquals(sRef, sTest);
    }

    /**
     * Test of parse on syntactically invalid input.
     */
    @Test(expected = RuntimeException.class)
    public final void testParseErrorExample() {
        /*
         * Setup
         */
        Statement sTest = this.constructorTest();
        SimpleReader file = new SimpleReader1L(FILE_NAME_2);
        Queue<String> tokens = Tokenizer.tokens(file);
        file.close();
        /*
         * The call--should result in an error being caught
         */
        sTest.parse(tokens);
    }

    /**
     * Test of parse on syntactically invalid input.
     */
    @Test(expected = RuntimeException.class)
    public final void testParseFile3() {
        /*
         * Setup
         */
        Statement sTest = this.constructorTest();
        SimpleReader file = new SimpleReader1L(FILE_NAME_3);
        Queue<String> tokens = Tokenizer.tokens(file);
        file.close();
        /*
         * The call--should result in an error being caught
         */
        sTest.parse(tokens);
    }

    /**
     * Test of parse on syntactically invalid input.
     */
    @Test(expected = RuntimeException.class)
    public final void testParseFile4() {
        /*
         * Setup
         */
        Statement sTest = this.constructorTest();
        SimpleReader file = new SimpleReader1L(FILE_NAME_4);
        Queue<String> tokens = Tokenizer.tokens(file);
        file.close();
        /*
         * The call--should result in an error being caught
         */
```

```
123         sTest.parse(tokens);
124     }
125
126 }
127
```