CSE 2231 – Software 2: Software Development and Design

Professor: Rob LaTour

Project #8: Program and Statement Parser Implementation(s)

Date of Submission: April 7th, 2023

The Ohio State University

College of Engineering

Columbus, Ohio

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

import org.junit.Test;

import components.program.Program;
import components.queue.Queue;
import components.simplereader.SimpleReader;
import components.simplereader.SimpleReader1L;
import components.utilities.Tokenizer;

/**
 * JUnit test fixture for {@code Program}'s constructor and kernel methods.
 *
 * @author Danny Kan (kan.74@osu.edu)
 * @author Jatin Mamtani (mamtani.6@osu.edu)
 *
 */
public abstract class ProgramTest {

    /**
     * The name(s) of file(s) containing (possibly invalid) BL program(s).
     */
    private static final String FILE_NAME_1 = "data/program1.bl",
            FILE_NAME_2 = "data/program2.bl",
            FILE_NAME_3 = "data/program-sample.bl",
            FILE_NAME_4 = "data/program-test1.bl",
            FILE_NAME_5 = "data/program-test2.bl",
            FILE_NAME_6 = "data/program-test3.bl",
            FILE_NAME_7 = "data/program-sample-1.bl",
            FILE_NAME_8 = "data/program-test1-1.bl",
            FILE_NAME_9 = "data/program-test2-1.bl",
            FILE_NAME_10 = "data/program-test3-1.bl";
```

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

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

/**
 * Test of parse on syntactically valid input.
 */
@Test
public final void testParseValidExample1() {
    /*
     * Setup
     */
    Program pRef = this.constructorRef();
    SimpleReader file = new SimpleReader1L(FILE_NAME_1);
    pRef.parse(file);
    file.close();
    Program pTest = this.constructorTest();
```

```
        file = new SimpleReader1L(FILE_NAME_1);
        Queue<String> tokens = Tokenizer.tokens(file);
        file.close();
        /*
         * The call
         */
        pTest.parse(tokens);
        /*
         * Evaluation
         */
        assertEquals(pRef, pTest);
    }

    /**
     * Test of parse on syntactically valid input.
     */
    @Test
    public final void testParseValidExample3() {
        /*
         * Setup
         */
        Program pRef = this.constructorRef();
        SimpleReader file = new SimpleReader1L(FILE_NAME_3);
        pRef.parse(file);
        file.close();
        Program pTest = this.constructorTest();
        file = new SimpleReader1L(FILE_NAME_3);
        Queue<String> tokens = Tokenizer.tokens(file);
        file.close();
        /*
         * The call
         */
        pTest.parse(tokens);
        /*
         * Evaluation
         */
        assertEquals(pRef, pTest);
    }

    /**
     * Test of parse on syntactically valid input.
     */
    @Test
    public final void testParseValidExample4() {
        /*
         * Setup
         */
        Program pRef = this.constructorRef();
        SimpleReader file = new SimpleReader1L(FILE_NAME_4);
        pRef.parse(file);
        file.close();
        Program pTest = this.constructorTest();
        file = new SimpleReader1L(FILE_NAME_4);
        Queue<String> tokens = Tokenizer.tokens(file);
        file.close();
        /*
         * The call
         */
        pTest.parse(tokens);
        /*
         * Evaluation
         */
        assertEquals(pRef, pTest);
    }
```

```
/**
 * Test of parse on syntactically valid input.
 */
@Test
public final void testParseValidExample5() {
    /*
     * Setup
     */
    Program pRef = this.constructorRef();
    SimpleReader file = new SimpleReader1L(FILE_NAME_5);
    pRef.parse(file);
    file.close();
    Program pTest = this.constructorTest();
    file = new SimpleReader1L(FILE_NAME_5);
    Queue<String> tokens = Tokenizer.tokens(file);
    file.close();
    /*
     * The call
     */
    pTest.parse(tokens);
    /*
     * Evaluation
     */
    assertEquals(pRef, pTest);
}

/**
 * Test of parse on syntactically valid input.
 */
@Test
public final void testParseValidExample6() {
    /*
     * Setup
     */
    Program pRef = this.constructorRef();
    SimpleReader file = new SimpleReader1L(FILE_NAME_6);
    pRef.parse(file);
    file.close();
    Program pTest = this.constructorTest();
    file = new SimpleReader1L(FILE_NAME_6);
    Queue<String> tokens = Tokenizer.tokens(file);
    file.close();
    /*
     * The call
     */
    pTest.parse(tokens);
    /*
     * Evaluation
     */
    assertEquals(pRef, pTest);
}

/**
 * Test of parse on syntactically invalid input.
 */
@Test(expected = RuntimeException.class)
public final void testParseErrorExample2() {
    /*
     * Setup
     */
    Program pTest = this.constructorTest();
    SimpleReader file = new SimpleReader1L(FILE_NAME_2);
    Queue<String> tokens = Tokenizer.tokens(file);
    file.close();
    /*
     * The call--should result in a syntax error being found
     */
    pTest.parse(tokens);
}

/**
 * Test of parse on syntactically invalid input.
 */
@Test(expected = RuntimeException.class)
public final void testParseErrorExample7() {
    /*
     * Setup
     */
    Program pTest = this.constructorTest();
    SimpleReader file = new SimpleReader1L(FILE_NAME_7);
    Queue<String> tokens = Tokenizer.tokens(file);
    file.close();
    /*
     * The call--should result in a syntax error being found
     */
    pTest.parse(tokens);
}

/**
 * Test of parse on syntactically invalid input.
 */
@Test(expected = RuntimeException.class)
public final void testParseErrorExample8() {
    /*
     * Setup
```

```
         */
        Program pTest = this.constructorTest();
        SimpleReader file = new SimpleReader1L(FILE_NAME_8);
        Queue<String> tokens = Tokenizer.tokens(file);
        file.close();
        /*
         * The call--should result in a syntax error being found
         */
        pTest.parse(tokens);
    }

    /**
     * Test of parse on syntactically invalid input.
     */
    @Test(expected = RuntimeException.class)
    public final void testParseErrorExample9() {
        /*
         * Setup
         */
        Program pTest = this.constructorTest();
        SimpleReader file = new SimpleReader1L(FILE_NAME_9);
        Queue<String> tokens = Tokenizer.tokens(file);
        file.close();
        /*
         * The call--should result in a syntax error being found
         */
        pTest.parse(tokens);
    }

    /**
     * Test of parse on syntactically invalid input.
     */
    @Test(expected = RuntimeException.class)
    public final void testParseErrorExample10() {
        /*
         * Setup
         */
        Program pTest = this.constructorTest();
        SimpleReader file = new SimpleReader1L(FILE_NAME_10);
        Queue<String> tokens = Tokenizer.tokens(file);
        file.close();
        /*
         * The call--should result in a syntax error being found
         */
        pTest.parse(tokens);
    }

}
```