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

/**
 * JUnit test fixture for {@code Program}'s constructor and kernel methods.
 *
 * @author Put your name here
 *
 */
public abstract class ProgramTest {

    /**
     * The names of a files containing a (possibly invalid) BL programs.
     */
    private static final String FILE_NAME_1 = "test/program1.bl",
            FILE_NAME_2 = "test/program2.bl", FILE_NAME_3 = "test/program3.bl",
            FILE_NAME_4 = "test/program4.bl", FILE_NAME_5 = "test/program5.bl",
            FILE_NAME_6 = "test/program6.bl", FILE_NAME_7 = "test/program7.bl",
            FILE_NAME_8 = "test/program8.bl", FILE_NAME_9 = "test/program9.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 testParseValid1() {
        /*
         * 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 invalid input.
     */
    @Test(expected = RuntimeException.class)
    public final void testParseError2() {
        /*
         * 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.
```

```
 92      */
 93     @Test(expected = RuntimeException.class)
 94     public final void testParseError3() {
 95         /*
 96          * Setup
 97          */
 98         Program pTest = this.constructorTest();
 99         SimpleReader file = new SimpleReader1L(FILE_NAME_3);
100         Queue<String> tokens = Tokenizer.tokens(file);
101         file.close();
102         /*
103          * The call--should result in a syntax error being found
104          */
105         pTest.parse(tokens);
106     }
107
108     /**
109      * Test of parse on syntactically invalid input.
110      */
111     @Test(expected = RuntimeException.class)
112     public final void testParseError4() {
113         /*
114          * Setup
115          */
116         Program pTest = this.constructorTest();
117         SimpleReader file = new SimpleReader1L(FILE_NAME_4);
118         Queue<String> tokens = Tokenizer.tokens(file);
119         file.close();
120         /*
121          * The call--should result in a syntax error being found
122          */
123         pTest.parse(tokens);
124     }
125
126     /**
127      * Test of parse on syntactically valid input.
128      */
129     @Test
130     public final void testParseValid5() {
131         /*
132          * Setup
133          */
134         Program pRef = this.constructorRef();
135         SimpleReader file = new SimpleReader1L(FILE_NAME_5);
136         pRef.parse(file);
137         file.close();
138         Program pTest = this.constructorTest();
139         file = new SimpleReader1L(FILE_NAME_5);
140         Queue<String> tokens = Tokenizer.tokens(file);
141         file.close();
142         /*
143          * The call
144          */
145         pTest.parse(tokens);
146         /*
147          * Evaluation
148          */
149         assertEquals(pRef, pTest);
150     }
151
152     /**
153      * Test of parse on syntactically invalid input.
154      */
155     @Test(expected = RuntimeException.class)
156     public final void testParseError6() {
157         /*
158          * Setup
159          */
160         Program pTest = this.constructorTest();
161         SimpleReader file = new SimpleReader1L(FILE_NAME_6);
162         Queue<String> tokens = Tokenizer.tokens(file);
163         file.close();
164         /*
165          * The call--should result in a syntax error being found
166          */
167         pTest.parse(tokens);
168     }
169
170     /**
171      * Test of parse on syntactically invalid input.
172      */
173     @Test(expected = RuntimeException.class)
174     public final void testParseError7() {
```

```java
        /*
         * 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 testParseError8() {
        /*
         * 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 testParseError9() {
        /*
         * 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);
    }
    // TODO - add more test cases for valid inputs
    // TODO - add more test cases for as many distinct syntax errors as possible

}
```