```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";

    /**
     * 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 testParseValidExample() {
        /*
         * 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
         */
```

```
66          assertEquals(pRef, pTest);
67      }
68
69      /**
70       * Test of parse on syntactically invalid input.
71       */
72      @Test(expected = RuntimeException.class)
73      public final void testParseErrorExample() {
74          /*
75           * Setup
76           */
77          Program pTest = this.constructorTest();
78          SimpleReader file = new SimpleReader1L(FILE_NAME_2);
79          Queue<String> tokens = Tokenizer.tokens(file);
80          file.close();
81          /*
82           * The call--should result in a syntax error being found
83           */
84          pTest.parse(tokens);
85      }
86
87      // TODO - add more test cases for valid inputs
88      // TODO - add more test cases for as many distinct syntax errors as possible
89
90      /**
91       * Test of parse on syntactically invalid input.
92       */
93      @Test(expected = RuntimeException.class)
94      public final void testParseSameInstructionError() {
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
```