CSE 2231 – Software 2: Software Development and Design

Professor: Rob LaTour

Project #7

Program and Statement Kernel Implementations / Implementation of Program and Statement Kernels

---

Date of Submission: March 24th, 2023

The Ohio State University

College of Engineering

Columbus, Ohio

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

import org.junit.Test;

import components.map.Map;
import components.map.Map.Pair;
import components.program.Program;
import components.simplereader.SimpleReader;
import components.simplereader.SimpleReader1L;
import components.statement.Statement;

/**
 * JUnit test fixture for {@code Program}'s constructor and kernel methods.
 *
 * @author Wayne Heym (heym.1@osu.edu)
 * @author Danny Kan (kan.74@osu.edu)
 * @author Jatin Mamtani (mamtani.6@osu.edu)
 *
 */
public abstract class ProgramTest {

    /**
     * The name of a file containing a BL program.
     */
    private static final String PROGRAM_SAMPLE = "data/program-sample.bl";
    /**
     * The name of a file containing a BL program.
     */
    private static final String PROGRAM_TEST_1 = "data/program-test1.bl";
    /**
     * The name of a file containing a BL program.
     */
```

```
private static final String PROGRAM_TEST_2 = "data/program-test2.bl";
/**
 * The name of a file containing a BL program.
 */
private static final String PROGRAM_TEST_3 = "data/program-test3.bl";

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

/**
 *
 * Creates and returns a {@code Program}, of the type of the implementation
 * under test, from the file with the given name.
 *
 * @param filename
 *            the name of the file to be parsed to create the program
 * @return the constructed program
```

```
     * @ensures createFromFile = [the program as parsed from the file]
     */
    private Program createFromFileTest(String filename) {
        Program p = this.constructorTest();
        SimpleReader file = new SimpleReader1L(filename);
        p.parse(file);
        file.close();
        return p;
    }

    /**
     *
     * Creates and returns a {@code Program}, of the reference implementation
     * type, from the file with the given name.
     *
     * @param filename
     *            the name of the file to be parsed to create the program
     * @return the constructed program
     * @ensures createFromFile = [the program as parsed from the file]
     */
    private Program createFromFileRef(String filename) {
        Program p = this.constructorRef();
        SimpleReader file = new SimpleReader1L(filename);
        p.parse(file);
        file.close();
        return p;
    }

    /**
     * Test constructor.
     */
    @Test
```

```
public final void testConstructor() {
    /*
     * Setup
     */
    Program pRef = this.constructorRef();

    /*
     * The call
     */
    Program pTest = this.constructorTest();

    /*
     * Evaluation
     */
    assertEquals(pRef, pTest);
}

/**
 * Test name.
 */
@Test
public final void testName() {
    /*
     * Setup
     */
    Program pTest = this.createFromFileTest(PROGRAM_SAMPLE);
    Program pRef = this.createFromFileRef(PROGRAM_SAMPLE);

    /*
     * The call
     */
    String result = pTest.name();
```

```
    /*
     * Evaluation
     */
    assertEquals(pRef, pTest);
    assertEquals("Test", result);
}

/**
 * Test name.
 */
@Test
public final void testNameProgramTest1() {
    Program pTest = this.createFromFileTest(PROGRAM_TEST_1);
    Program pRef = this.createFromFileRef(PROGRAM_TEST_1);
    String result = pTest.name();
    assertEquals(pRef, pTest);
    assertEquals("programTest1", result);
}

/**
 * Test name.
 */
@Test
public final void testNameProgramTest2() {
    Program pTest = this.createFromFileTest(PROGRAM_TEST_2);
    Program pRef = this.createFromFileRef(PROGRAM_TEST_2);
    String result = pTest.name();
    assertEquals(pRef, pTest);
    assertEquals("programTest2", result);
}
```

```
/**
 * Test name.
 */
@Test
public final void testNameProgramTest3() {
    Program pTest = this.createFromFileTest(PROGRAM_TEST_3);
    Program pRef = this.createFromFileRef(PROGRAM_TEST_3);
    String result = pTest.name();
    assertEquals(pRef, pTest);
    assertEquals("programTest3", result);
}

/**
 * Test setName.
 */
@Test
public final void testSetName() {
    /*
     * Setup
     */
    Program pTest = this.createFromFileTest(PROGRAM_SAMPLE);
    Program pRef = this.createFromFileRef(PROGRAM_SAMPLE);
    String newName = "Replacement";
    pRef.setName(newName);

    /*
     * The call
     */
    pTest.setName(newName);

    /*
     * Evaluation
```

```
     */
    assertEquals(pRef, pTest);
}

/**
 * Test setName.
 */
@Test
public final void testSetNameProgramTest1() {
    Program pTest = this.createFromFileTest(PROGRAM_TEST_1);
    Program pRef = this.createFromFileRef(PROGRAM_TEST_1);
    String newName = "Replacement";
    pRef.setName(newName);
    pTest.setName(newName);
    assertEquals(pRef, pTest);
}

/**
 * Test setName.
 */
@Test
public final void testSetNameProgramTest2() {
    Program pTest = this.createFromFileTest(PROGRAM_TEST_2);
    Program pRef = this.createFromFileRef(PROGRAM_TEST_2);
    String newName = "Replacement";
    pRef.setName(newName);
    pTest.setName(newName);
    assertEquals(pRef, pTest);
}

/**
 * Test setName.
```

```
 */
@Test
public final void testSetNameProgramTest3() {
    Program pTest = this.createFromFileTest(PROGRAM_TEST_3);
    Program pRef = this.createFromFileRef(PROGRAM_TEST_3);
    String newName = "Replacement";
    pRef.setName(newName);
    pTest.setName(newName);
    assertEquals(pRef, pTest);
}

/**
 * Test newContext.
 */
@Test
public final void testNewContext() {
    /*
     * Setup
     */
    Program pTest = this.createFromFileTest(PROGRAM_SAMPLE);
    Program pRef = this.createFromFileRef(PROGRAM_SAMPLE);
    Map<String, Statement> cRef = pRef.newContext();

    /*
     * The call
     */
    Map<String, Statement> cTest = pTest.newContext();

    /*
     * Evaluation
     */
    assertEquals(pRef, pTest);
```

```
    assertEquals(cRef, cTest);
}

/**
 * Test newContext.
 */
@Test
public final void testNewContextProgramTest1() {
    Program pTest = this.createFromFileTest(PROGRAM_TEST_1);
    Program pRef = this.createFromFileRef(PROGRAM_TEST_1);
    Map<String, Statement> cRef = pRef.newContext();
    Map<String, Statement> cTest = pTest.newContext();
    assertEquals(pRef, pTest);
    assertEquals(cRef, cTest);
}

/**
 * Test newContext.
 */
@Test
public final void testNewContextProgramTest2() {
    Program pTest = this.createFromFileTest(PROGRAM_TEST_2);
    Program pRef = this.createFromFileRef(PROGRAM_TEST_2);
    Map<String, Statement> cRef = pRef.newContext();
    Map<String, Statement> cTest = pTest.newContext();
    assertEquals(pRef, pTest);
    assertEquals(cRef, cTest);
}

/**
 * Test newContext.
 */
```

```
@Test
public final void testNewContextProgramTest3() {
    Program pTest = this.createFromFileTest(PROGRAM_TEST_3);
    Program pRef = this.createFromFileRef(PROGRAM_TEST_3);
    Map<String, Statement> cRef = pRef.newContext();
    Map<String, Statement> cTest = pTest.newContext();
    assertEquals(pRef, pTest);
    assertEquals(cRef, cTest);
}

/**
 * Test swapContext.
 */
@Test
public final void testSwapContext() {
    /*
     * Setup
     */
    Program pTest = this.createFromFileTest(PROGRAM_SAMPLE);
    Program pRef = this.createFromFileRef(PROGRAM_SAMPLE);
    Map<String, Statement> contextRef = pRef.newContext();
    Map<String, Statement> contextTest = pTest.newContext();
    String oneName = "one";
    pRef.swapContext(contextRef);
    Pair<String, Statement> oneRef = contextRef.remove(oneName);
    /* contextRef now has just "two" */
    pRef.swapContext(contextRef);
    /* pRef's context now has just "two" */
    contextRef.add(oneRef.key(), oneRef.value());
    /* contextRef now has just "one" */

    /* Make the reference call, replacing, in pRef, "one" with "two": */
```

```
    pRef.swapContext(contextRef);

    pTest.swapContext(contextTest);
    Pair<String, Statement> oneTest = contextTest.remove(oneName);
    /* contextTest now has just "two" */
    pTest.swapContext(contextTest);
    /* pTest's context now has just "two" */
    contextTest.add(oneTest.key(), oneTest.value());
    /* contextTest now has just "one" */

    /*
     * The call
     */
    pTest.swapContext(contextTest);

    /*
     * Evaluation
     */
    assertEquals(pRef, pTest);
    assertEquals(contextRef, contextTest);
}

/**
 * Test swapContext.
 */
@Test
public final void testSwapContextProgramTest1() {
    Program pTest = this.createFromFileTest(PROGRAM_TEST_1);
    Program pRef = this.createFromFileRef(PROGRAM_TEST_1);
    Map<String, Statement> contextRef = pRef.newContext();
    Map<String, Statement> contextTest = pTest.newContext();
    String oneName = "instOne";
```

```
    pRef.swapContext(contextRef);
    Pair<String, Statement> oneRef = contextRef.remove(oneName);
    pRef.swapContext(contextRef);
    contextRef.add(oneRef.key(), oneRef.value());
    pRef.swapContext(contextRef);
    pTest.swapContext(contextTest);
    Pair<String, Statement> oneTest = contextTest.remove(oneName);
    pTest.swapContext(contextTest);
    contextTest.add(oneTest.key(), oneTest.value());
    pTest.swapContext(contextTest);
    assertEquals(pRef, pTest);
    assertEquals(contextRef, contextTest);
}

/**
 * Test swapContext.
 */
@Test
public final void testSwapContextProgramTest2() {
    Program pTest = this.createFromFileTest(PROGRAM_TEST_2);
    Program pRef = this.createFromFileRef(PROGRAM_TEST_2);
    Map<String, Statement> contextRef = pRef.newContext();
    Map<String, Statement> contextTest = pTest.newContext();
    String oneName = "instOne";
    pRef.swapContext(contextRef);
    Pair<String, Statement> oneRef = contextRef.remove(oneName);
    pRef.swapContext(contextRef);
    contextRef.add(oneRef.key(), oneRef.value());
    pRef.swapContext(contextRef);
    pTest.swapContext(contextTest);
    Pair<String, Statement> oneTest = contextTest.remove(oneName);
    pTest.swapContext(contextTest);
```

```
    contextTest.add(oneTest.key(), oneTest.value());
    pTest.swapContext(contextTest);
    assertEquals(pRef, pTest);
    assertEquals(contextRef, contextTest);
}

/**
 * Test swapContext.
 */
@Test
public final void testSwapContextProgramTest3() {
    Program pTest = this.createFromFileTest(PROGRAM_TEST_3);
    Program pRef = this.createFromFileRef(PROGRAM_TEST_3);
    Map<String, Statement> contextRef = pRef.newContext();
    Map<String, Statement> contextTest = pTest.newContext();
    String oneName = "instOne";
    pRef.swapContext(contextRef);
    Pair<String, Statement> oneRef = contextRef.remove(oneName);
    pRef.swapContext(contextRef);
    contextRef.add(oneRef.key(), oneRef.value());
    pRef.swapContext(contextRef);
    pTest.swapContext(contextTest);
    Pair<String, Statement> oneTest = contextTest.remove(oneName);
    pTest.swapContext(contextTest);
    contextTest.add(oneTest.key(), oneTest.value());
    pTest.swapContext(contextTest);
    assertEquals(pRef, pTest);
    assertEquals(contextRef, contextTest);
}

/**
 * Test newBody.
```

```
 */
@Test
public final void testNewBody() {
    /*
     * Setup
     */
    Program pTest = this.createFromFileTest(PROGRAM_SAMPLE);
    Program pRef = this.createFromFileRef(PROGRAM_SAMPLE);
    Statement bRef = pRef.newBody();

    /*
     * The call
     */
    Statement bTest = pTest.newBody();

    /*
     * Evaluation
     */
    assertEquals(pRef, pTest);
    assertEquals(bRef, bTest);
}

/**
 * Test newBody.
 */
@Test
public final void testNewBodyProgramTest1() {
    Program pTest = this.createFromFileTest(PROGRAM_TEST_1);
    Program pRef = this.createFromFileRef(PROGRAM_TEST_1);
    Statement bRef = pRef.newBody();
    Statement bTest = pTest.newBody();
    assertEquals(pRef, pTest);
```

```
    assertEquals(bRef, bTest);
}

/**
 * Test newBody.
 */
@Test
public final void testNewBodyProgramTest2() {
    Program pTest = this.createFromFileTest(PROGRAM_TEST_2);
    Program pRef = this.createFromFileRef(PROGRAM_TEST_2);
    Statement bRef = pRef.newBody();
    Statement bTest = pTest.newBody();
    assertEquals(pRef, pTest);
    assertEquals(bRef, bTest);
}

/**
 * Test newBody.
 */
@Test
public final void testNewBodyProgramTest3() {
    Program pTest = this.createFromFileTest(PROGRAM_TEST_3);
    Program pRef = this.createFromFileRef(PROGRAM_TEST_3);
    Statement bRef = pRef.newBody();
    Statement bTest = pTest.newBody();
    assertEquals(pRef, pTest);
    assertEquals(bRef, bTest);
}

/**
 * Test swapBody.
 */
```

```
@Test
public final void testSwapBody() {
    /*
     * Setup
     */
    Program pTest = this.createFromFileTest(PROGRAM_SAMPLE);
    Program pRef = this.createFromFileRef(PROGRAM_SAMPLE);
    Statement bodyRef = pRef.newBody();
    Statement bodyTest = pTest.newBody();
    pRef.swapBody(bodyRef);
    Statement firstRef = bodyRef.removeFromBlock(0);
    /* bodyRef now lacks the first statement */
    pRef.swapBody(bodyRef);
    /* pRef's body now lacks the first statement */
    bodyRef.addToBlock(0, firstRef);
    /* bodyRef now has just the first statement */

    /* Make the reference call, replacing, in pRef, remaining with first: */
    pRef.swapBody(bodyRef);

    pTest.swapBody(bodyTest);
    Statement firstTest = bodyTest.removeFromBlock(0);
    /* bodyTest now lacks the first statement */
    pTest.swapBody(bodyTest);
    /* pTest's body now lacks the first statement */
    bodyTest.addToBlock(0, firstTest);
    /* bodyTest now has just the first statement */

    /*
     * The call
     */
    pTest.swapBody(bodyTest);
```

```
    /*
     * Evaluation
     */
    assertEquals(pRef, pTest);
    assertEquals(bodyRef, bodyTest);
}

/**
 * Test swapBody.
 */
@Test
public final void testSwapBodyProgramTest1() {
    Program pTest = this.createFromFileTest(PROGRAM_TEST_1);
    Program pRef = this.createFromFileRef(PROGRAM_TEST_1);
    Statement bodyRef = pRef.newBody();
    Statement bodyTest = pTest.newBody();
    pRef.swapBody(bodyRef);
    Statement firstRef = bodyRef.removeFromBlock(0);
    pRef.swapBody(bodyRef);
    bodyRef.addToBlock(0, firstRef);
    pRef.swapBody(bodyRef);
    pTest.swapBody(bodyTest);
    Statement firstTest = bodyTest.removeFromBlock(0);
    pTest.swapBody(bodyTest);
    bodyTest.addToBlock(0, firstTest);
    pTest.swapBody(bodyTest);
    assertEquals(pRef, pTest);
    assertEquals(bodyRef, bodyTest);
}

/**
```

```
 * Test swapBody.
 */
@Test
public final void testSwapBodyProgramTest2() {
    Program pTest = this.createFromFileTest(PROGRAM_TEST_2);
    Program pRef = this.createFromFileRef(PROGRAM_TEST_2);
    Statement bodyRef = pRef.newBody();
    Statement bodyTest = pTest.newBody();
    pRef.swapBody(bodyRef);
    Statement firstRef = bodyRef.removeFromBlock(0);
    pRef.swapBody(bodyRef);
    bodyRef.addToBlock(0, firstRef);
    pRef.swapBody(bodyRef);
    pTest.swapBody(bodyTest);
    Statement firstTest = bodyTest.removeFromBlock(0);
    pTest.swapBody(bodyTest);
    bodyTest.addToBlock(0, firstTest);
    pTest.swapBody(bodyTest);
    assertEquals(pRef, pTest);
    assertEquals(bodyRef, bodyTest);
}

/**
 * Test swapBody.
 */
@Test
public final void testSwapBodyProgramTest3() {
    Program pTest = this.createFromFileTest(PROGRAM_TEST_3);
    Program pRef = this.createFromFileRef(PROGRAM_TEST_3);
    Statement bodyRef = pRef.newBody();
    Statement bodyTest = pTest.newBody();
    pRef.swapBody(bodyRef);
```

```
        Statement firstRef = bodyRef.removeFromBlock(0);
        pRef.swapBody(bodyRef);
        bodyRef.addToBlock(0, firstRef);
        pRef.swapBody(bodyRef);
        pTest.swapBody(bodyTest);
        Statement firstTest = bodyTest.removeFromBlock(0);
        pTest.swapBody(bodyTest);
        bodyTest.addToBlock(0, firstTest);
        pTest.swapBody(bodyTest);
        assertEquals(pRef, pTest);
        assertEquals(bodyRef, bodyTest);
    }

}
```