**CPSC326 | Homework 5: Virtual Machine | 3/30/2025 | Arjuna Herbst**

My first positive test verifies the VM's ability to handle complex mathematical expressions with multiple operations and proper execution order. This was important since the stack-based VM requires careful management of operands across operations, which was very difficult to get working properly. I also created a test for nested function calls, which tests the VM's ability to handle function frames and return values.

For my negative test cases, I focused on error handling. The first negative test ensures that my VM correctly identifies attempts to call undefined functions. This validates my error-checking implementation. The second negative test confirms that my VM properly validates type compatibility rather than failing unexpectedly.

```
1704    // Positive Tests
1705
1706    @Test
1707    void complexCalculationWithMultipleOperations() {
1708      VMFrameTemplate m = new VMFrameTemplate("main");
1709      // Calculate (3 + 4) * 2 - 5
1710      m.add(VMInstr.PUSH(3));
1711      m.add(VMInstr.PUSH(4));
1712      m.add(VMInstr.ADD()); // 3 + 4 = 7
1713      m.add(VMInstr.PUSH(2));
1714      m.add(VMInstr.MUL()); // 7 * 2 = 14
1715      m.add(VMInstr.PUSH(5));
1716      m.add(VMInstr.SUB()); // 14 - 5 = 9
1717      m.add(VMInstr.WRITE());
1718      VM vm = new VM();
1719      vm.add(m);
1720      vm.run();
1721      assertEquals("9", output.toString());
1722    }
1723
1724    @Test
1725    void nestedFunctionCalls() {
1726      // Function that returns double the input
1727      VMFrameTemplate double_func = new VMFrameTemplate("double");
1728      double_func.add(VMInstr.PUSH(2));
1729      double_func.add(VMInstr.MUL());
1730      double_func.add(VMInstr.RET());
1731
1732      // Function that adds 5 to the input
1733      VMFrameTemplate add5_func = new VMFrameTemplate("add5");
1734      add5_func.add(VMInstr.PUSH(5));
1735      add5_func.add(VMInstr.ADD());
1736      add5_func.add(VMInstr.RET());
1737
1738      // Main function that calls double(add5(3))
1739      VMFrameTemplate m = new VMFrameTemplate("main");
1740      m.add(VMInstr.PUSH(3));
1741      m.add(VMInstr.CALL("add5")); // add5(3) = 8
1742      m.add(VMInstr.CALL("double")); // double(8) = 16
1743      m.add(VMInstr.WRITE());
1744
1745      VM vm = new VM();
1746      vm.add(double_func);
1747      vm.add(add5_func);
1748      vm.add(m);
1749      vm.run();
1750      assertEquals("16", output.toString());
1751    }
```

```
1753    // Negative Tests
1754
1755    @Test
1756    void callUndefinedFunction() {
1757      VMFrameTemplate m = new VMFrameTemplate("main");
1758      m.add(VMInstr.CALL("undefined_function"));
1759      VM vm = new VM();
1760      vm.add(m);
1761      Exception e = assertThrows(MyPLException.class, () -> vm.run());
1762      assertTrue(e.getMessage().startsWith("VM_ERROR: "));
1763    }
1764
1765    @Test
1766    void divideStringByNumber() {
1767      VMFrameTemplate m = new VMFrameTemplate("main");
1768      // Push a string and a number
1769      m.add(VMInstr.PUSH("hello"));
1770      m.add(VMInstr.PUSH(5));
1771      // Try to divide a string by a number (invalid operation)
1772      m.add(VMInstr.DIV());
1773      m.add(VMInstr.WRITE());
1774
1775      VM vm = new VM();
1776      vm.add(m);
1777
1778      // Should throw a MyPLException
1779      Exception e = assertThrows(MyPLException.class, () -> vm.run());
1780      assertTrue(e.getMessage().startsWith("VM_ERROR:"));
1781    }
1782
1783  }
```