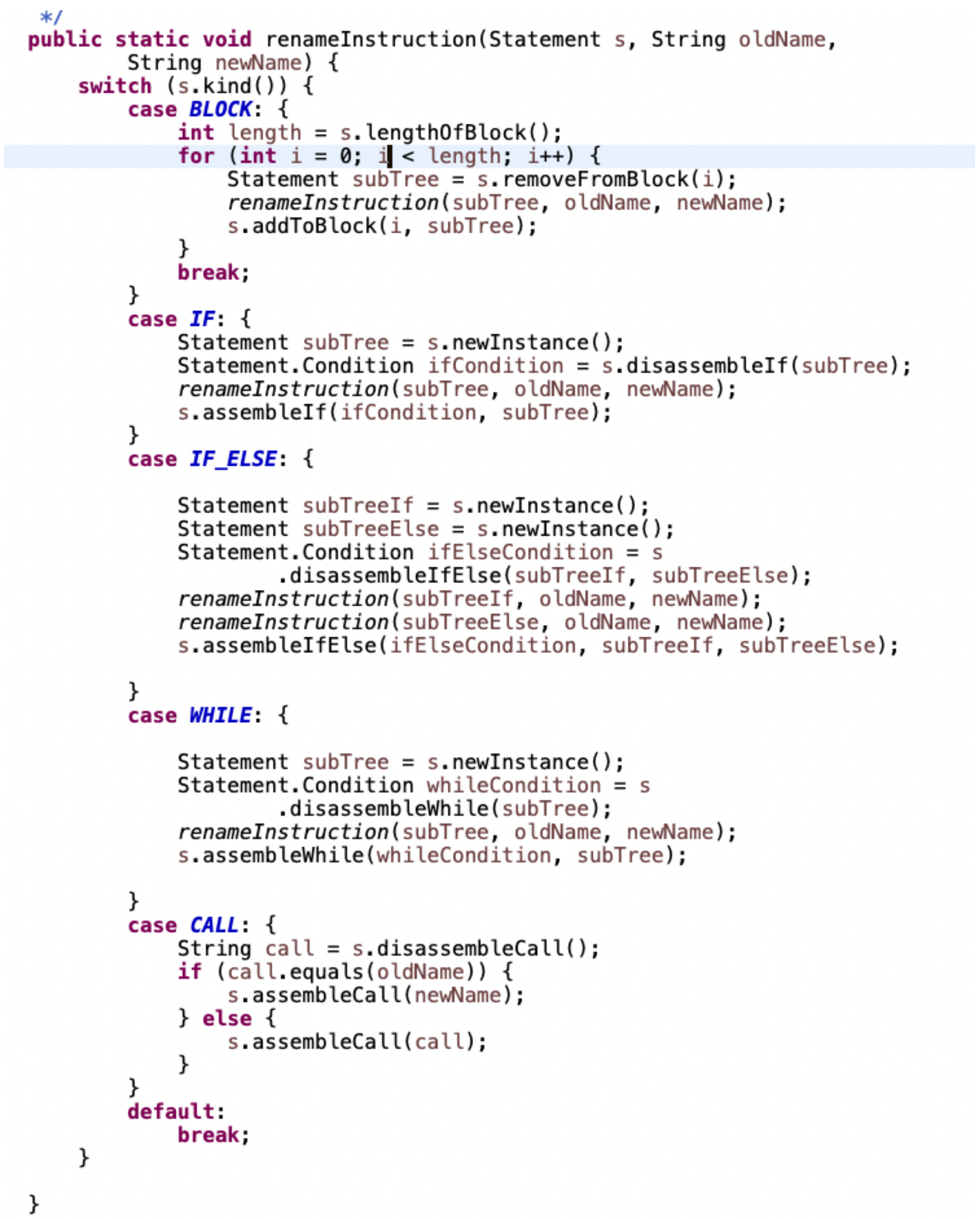
Homework 25

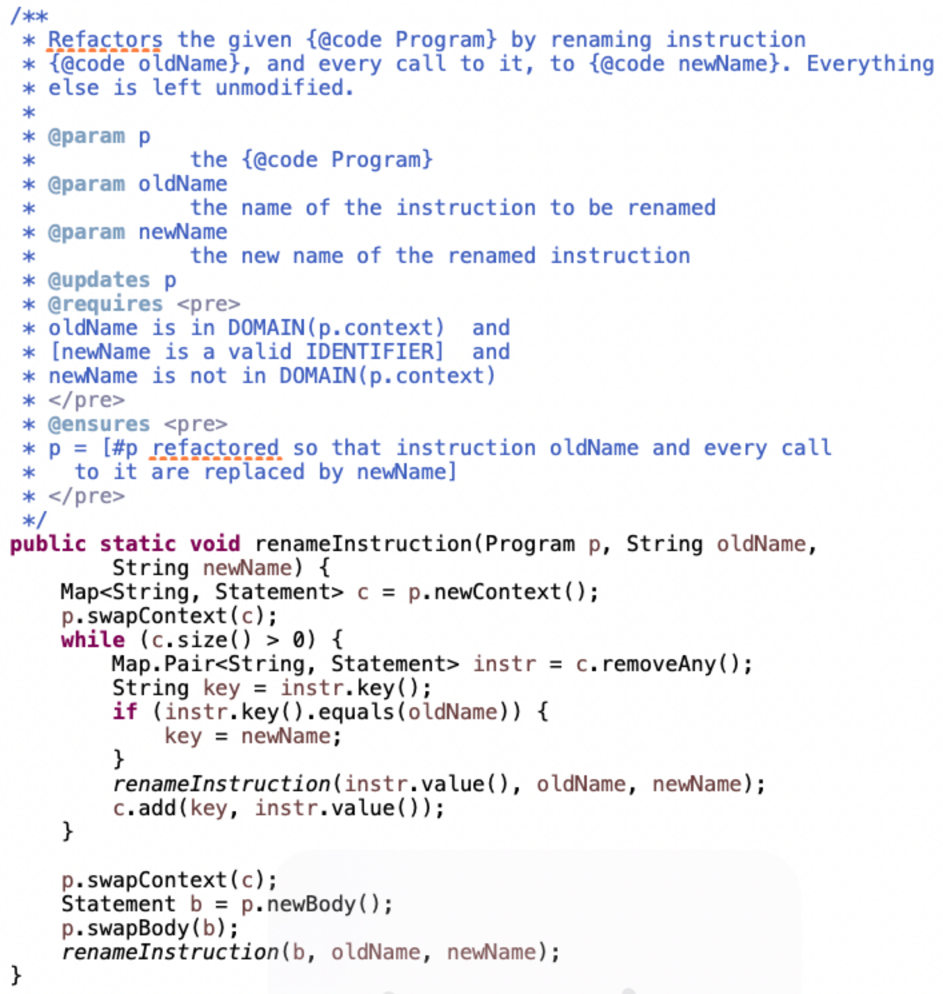
Shyam Sai Bethina

23 March 2022



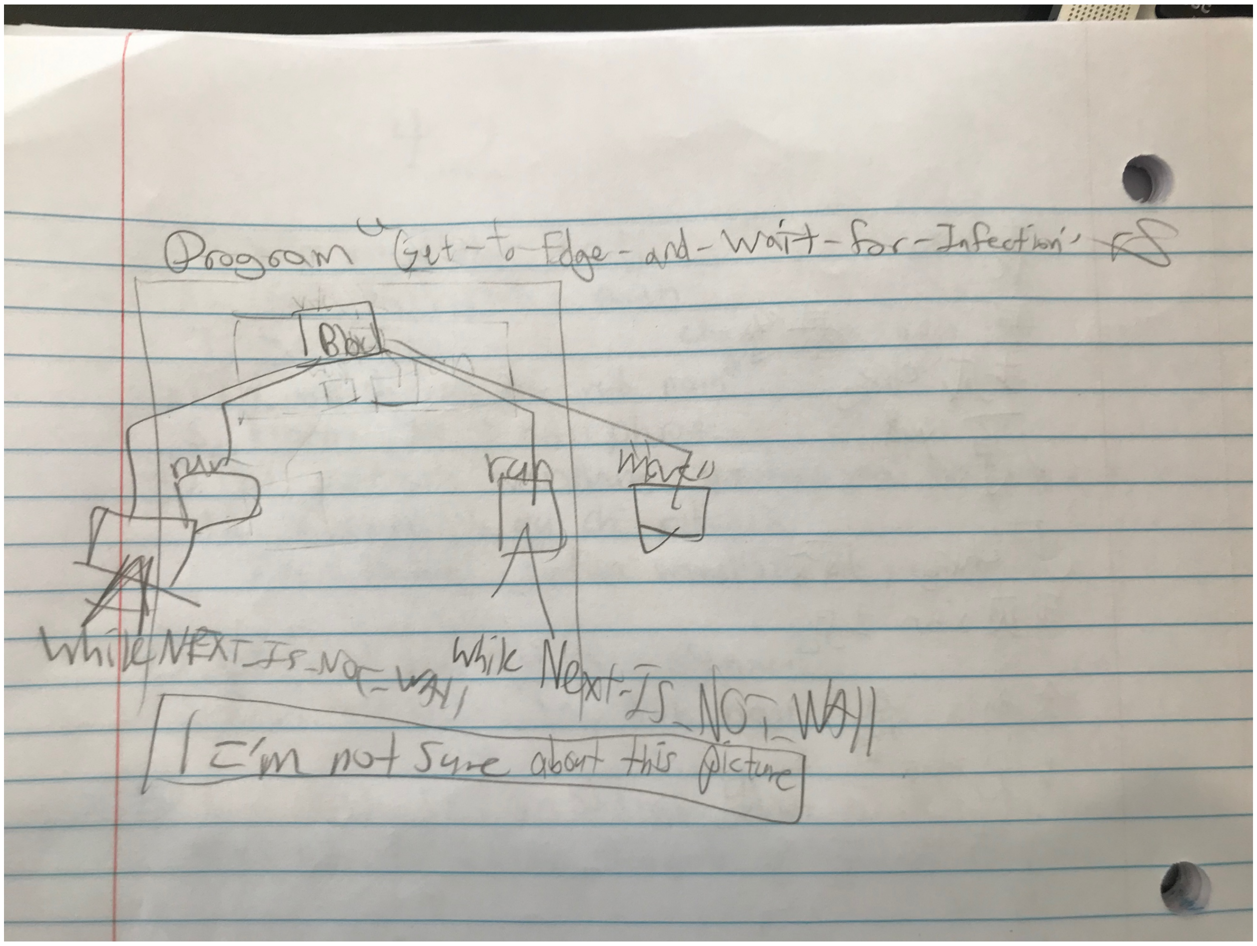
```java
 */
public static void renameInstruction(Statement s, String oldName,
        String newName) {
    switch (s.kind()) {
        case BLOCK: {
            int length = s.lengthOfBlock();
            for (int i = 0; i < length; i++) {
                Statement subTree = s.removeFromBlock(i);
                renameInstruction(subTree, oldName, newName);
                s.addToBlock(i, subTree);
            }
            break;
        }
        case IF: {
            Statement subTree = s.newInstance();
            Statement.Condition ifCondition = s.disassembleIf(subTree);
            renameInstruction(subTree, oldName, newName);
            s.assembleIf(ifCondition, subTree);
        }
        case IF_ELSE: {

            Statement subTreeIf = s.newInstance();
            Statement subTreeElse = s.newInstance();
            Statement.Condition ifElseCondition = s
                    .disassembleIfElse(subTreeIf, subTreeElse);
            renameInstruction(subTreeIf, oldName, newName);
            renameInstruction(subTreeElse, oldName, newName);
            s.assembleIfElse(ifElseCondition, subTreeIf, subTreeElse);

        }
        case WHILE: {

            Statement subTree = s.newInstance();
            Statement.Condition whileCondition = s
                    .disassembleWhile(subTree);
            renameInstruction(subTree, oldName, newName);
            s.assembleWhile(whileCondition, subTree);

        }
        case CALL: {
            String call = s.disassembleCall();
            if (call.equals(oldName)) {
                s.assembleCall(newName);
            } else {
                s.assembleCall(call);
            }
        }
        default:
            break;
    }

}
```

1)



```java
/**
 * Refactors the given {@code Program} by renaming instruction
 * {@code oldName}, and every call to it, to {@code newName}. Everything
 * else is left unmodified.
 *
 * @param p
 *            the {@code Program}
 * @param oldName
 *            the name of the instruction to be renamed
 * @param newName
 *            the new name of the renamed instruction
 * @updates p
 * @requires <pre>
 * oldName is in DOMAIN(p.context)  and
 * [newName is a valid IDENTIFIER]  and
 * newName is not in DOMAIN(p.context)
 * </pre>
 * @ensures <pre>
 * p = [#p refactored so that instruction oldName and every call
 *   to it are replaced by newName]
 * </pre>
 */
public static void renameInstruction(Program p, String oldName,
        String newName) {
    Map<String, Statement> c = p.newContext();
    p.swapContext(c);
    while (c.size() > 0) {
        Map.Pair<String, Statement> instr = c.removeAny();
        String key = instr.key();
        if (instr.key().equals(oldName)) {
            key = newName;
        }
        renameInstruction(instr.value(), oldName, newName);
        c.add(key, instr.value());
    }

    p.swapContext(c);
    Statement b = p.newBody();
    p.swapBody(b);
    renameInstruction(b, oldName, newName);
}
```

2)

3) 
Program "Get-to-Edge-and-Wait-for-Infection"
Block
run
run
move()
While NEXT_IS_NOT_WALL
While Next_IS_NOT_WALL
I'm not sure about this picture

4) The original block would've lost it's context when .clear is called. newInstance() creates an empty block that doesn't change the original block, preserving the state of the original block.