Report

The algorithm is split into two main sections. That which optimises the loops and that which optimise the instructions.

For the loop optimization, loops are detected by calling the getLoopPositions method with the code's instruction list as its argument. This returns an array of integers representing the position of the loops. This works by iterating through the method's instruction handles. If the handle's instruction is an instance of the IINC (Increment Local Variable By Constant) class, then its index and the next instruction are stored. If this next instruction is an instance of GotoInstruction, then its target position minus 2, position and original instruction index are appended to an array containing the loop positions. After the for loop has run, this array is returned.

The body of the optimiseMethod function is contained within a for loop that iterates through each instruction in the method. Before this however some variables are declared and defined:

- methodGen: a new instance of a MethodGen object which is used to replace the old method with the optimised method.
- skipNextArithmeticOperation: a boolean flag for if the next arithmetic operation should be skipped
- constants: an integer that stores the constant count.
- loopArray: the position of loops as calculated by the getLoopPositions method.

The for loop itself starts by checking if the current instruction is null and if it is looping to the next instruction. Then it determines if the current instruction is in a loop by checking if its handle is between the start and end of a loop for each set of three values that defines a loop in loopArray. If it is, the inLoop flag is set to true.

Then the type of instruction is determined by checking whether the instruction itself is an instance of any of the broad classes of instruction that constitute any of the following. If it is, then a flag indicating that is set to true:

- LDC or Push Instruction: An instruction that pushes an item from or onto the constant pool/stack.
- Arithmetic Instruction
- Constant
- StoreInstruction: Denotes an unparameterized instruction to store a value into a local variable.
- LoadInstruction: Denotes an unparameterized instruction to load a value from a local variable.

If the inLoop flag is true, then the algorithm first checks that the current variable is not the variable incremented by the loop and sets the skipNextArithmeticOperation variable to true if it is. If the current variable is not the incrementing variable, then the variable with the index of the current instruction's handle is pushed onto the constant stack. If this variable is an integer then the handle's current instruction is replaced by a new LDC instruction that pushes the integer to a new constant pool generator. If it isn't an integer then a new

instruction is inserted to the instruction list before the variable. If the instruction was an arithmetic instruction and not the incrementing variable, then the operation is performed using the doArithmeticOperation method. The constant on the top of the constant stack is inserted into the constant pool as a PUSH compound instruction before the handle. The handle is then deleted from the instruction list. The loop optimisation then ends.

The code executed next is dependent on the instruction's type:

- If it's a constant then it is pushed onto the constant stack.
- If it's an arithmetic instruction then the same is done as in the loop optimisation.
- If it's a store instruction, then the constant at the top of the constant stack is inserted into the variables array at the position of the current instruction's handle.
- If it's a load instruction then the variable at the index of the current instruction handle is pushed onto the constant stack and is inserted into the constant pool as a PUSH compound instruction before the handle. The handle is then deleted from the instruction list.

Errors are then caught using a try-except block and the old method is replaced by the optimised method

Joshua has written the loop detection and optimisation of constant assignment inside, Agnieszka wrote the optimisation for the arithmetic, constant, store and load operations and Sam wrote this report and the removeLDC method.