

Assignment 4 Report

Sandeep Varma (200050032)
Yeshash Chandra (200050031)

Field Privatization

The optimization we chose is field privatization. Field privatization is an optimization where unnecessary field accesses are minimized after performing necessary points-to analysis over the program. This is mainly useful when there are loops present in the code. So, we have done field privatization and minimized the field accesses within a loop.

Step 1:

First, we have made the fields private and added getter and setter methods to read and write to the fields. This makes field privatization intuitive.

Step 2:

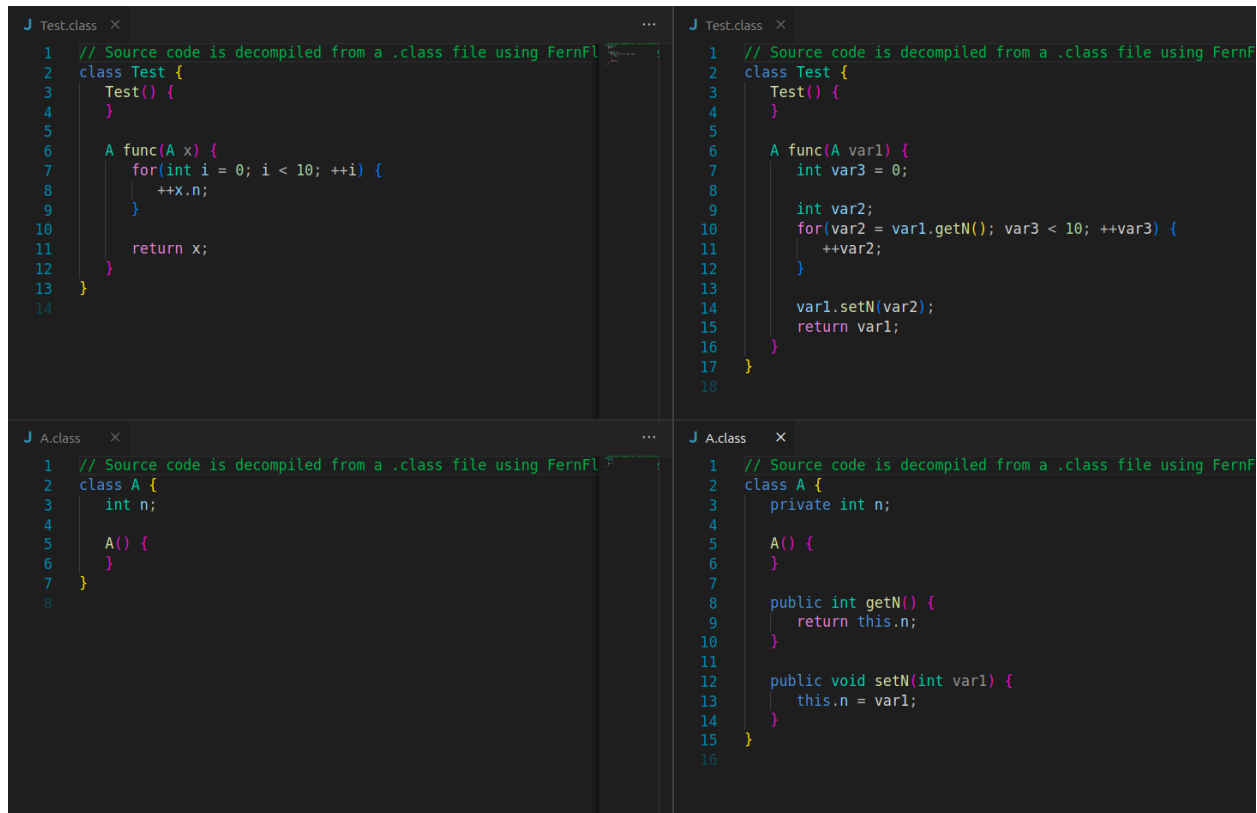
In all the methods, replace the appropriate field accesses with suitable getter and setter method calls.

Step 3:

We build a LoopNestTree (provided by soot) to analyze and transform the loops. With points-to analysis, we filter the fields whose accesses can be localized in that loop. For any method calls made inside the loop body, for all the objects reachable by these methods, field accesses cannot be localized.

Step 4:

For these fields whose accesses can be localized, we make temporary local variables. For all these localized fields, the local variables are added to the method's body. Next, before the beginning of the loop, all these variables are assigned the values of the fields. After the loop, all the values in these variables are stored back in the fields.



The image displays four code snippets in a dark-themed IDE, arranged in a 2x2 grid. The top row shows the 'Test' class, and the bottom row shows the 'A' class. The left column represents the unoptimized state, while the right column represents the state after optimization by Soot.

Top Left (Unoptimized Test.class):

```
1 // Source code is decompiled from a .class file using FernFl
2 class Test {
3     Test() {
4     }
5
6     A func(A x) {
7         for(int i = 0; i < 10; ++i) {
8             ++x.n;
9         }
10
11         return x;
12     }
13 }
14
```

Top Right (Optimized Test.class):

```
1 // Source code is decompiled from a .class file using FernFl
2 class Test {
3     Test() {
4     }
5
6     A func(A var1) {
7         int var3 = 0;
8
9         int var2;
10        for(var2 = var1.getN(); var3 < 10; ++var3) {
11            ++var2;
12        }
13
14        var1.setN(var2);
15        return var1;
16    }
17 }
18
```

Bottom Left (Unoptimized A.class):

```
1 // Source code is decompiled from a .class file using FernFl
2 class A {
3     int n;
4
5     A() {
6     }
7 }
8
```

Bottom Right (Optimized A.class):

```
1 // Source code is decompiled from a .class file using FernFl
2 class A {
3     private int n;
4
5     A() {
6     }
7
8     public int getN() {
9         return this.n;
10    }
11
12    public void setN(int var1) {
13        this.n = var1;
14    }
15 }
16
```

The class files on the left are unoptimized whereas the one on the right is the output from soot. We can see that the class definition itself is being changed. The field has become private and getter and setter methods have been added. In the loop instead of repeated field read and write, a local variable is being used.

Code and how to run

The code is publicly available at <https://github.com/Sandeep-Varma/CS6004-A4/>

To run the soot code over all the testcases, one can simply run the script.sh in the command line. Then run.sh in the metrics directory. counter.py is useful to process the final output and get the metrics.

Performance boost (on running with VM)

Testcase	Actual Time	Optimized Time
Test0.java	0.132s	0.116s
Test1.java	0.127s	0.114s
Test2.java	0.106s	0.128s
Test3.java	0.121s	0.099s
Test4.java	0.129s	0.115s
Test5.java	0.124s	0.119s

THANK YOU

*** THE END ***