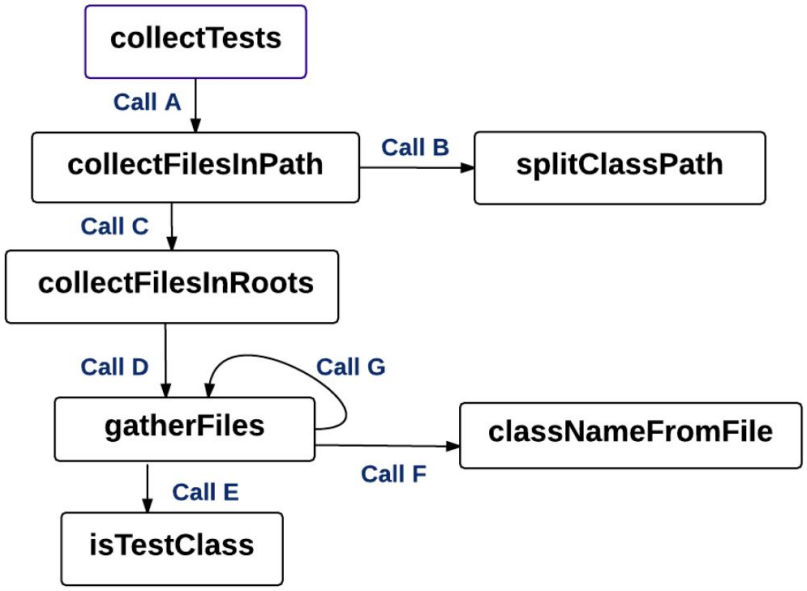
Graph

Class: ClassPathTestCollector



The relationship between seven methods in the class is shown above. The call label will be used throughout my report and other graph.

```
1.  public abstract class ClassPathTestCollector implements TestCollector {
2.     static final int SUFFIX_LENGTH = ".class".length();
3.
4.     public ClassPathTestCollector() {
5.     }
6.
7.     public Enumeration collectTests() {
8.         String classPath = System.getProperty("java.class.path");
9.         Hashtable result = this.collectFilesInPath(classPath);
10.        return result.elements();
11.    }
12.
13.    public Hashtable collectFilesInPath(String classPath) {
14.        Hashtable result = this.collectFilesInRoots(this.splitClassPath(classPath));
15.        return result;
16.    }
17.
18.    Hashtable collectFilesInRoots(Vector roots) {
19.        Hashtable result = new Hashtable(100);
20.        Enumeration e = roots.elements();
21.
22.        while(e.hasMoreElements()) {
```

```
23.             this.gatherFiles(new File((String)e.nextElement()), "", result);
24.         }
25. 
26.         return result;
27.     }
28. 
29.     void gatherFiles(File classRoot, String classFileName, Hashtable result) {
30.         File thisRoot = new File(classRoot, classFileName);
31.         if(thisRoot.isFile()) {
32.             if(this.isTestClass(classFileName)) {
33.                 String var7 = this.classNameFromFile(classFileName);
34.                 result.put(var7, var7);
35.             }
36. 
37.         } else {
38.             String[] contents = thisRoot.list();
39.             if(contents != null) {
40.                 for(int i = 0; i < contents.length; ++i) {
41.                     this.gatherFiles(classRoot, classFileName + File.separatorChar +
    contents[i], result);
42.                 }
43.             }
44. 
45.         }
46.     }
47. 
48.     Vector splitClassPath(String classPath) {
49.         Vector result = new Vector();
50.         String separator = System.getProperty("path.separator");
51.         StringTokenizer tokenizer = new StringTokenizer(classPath, separator);
52. 
53.         while(tokenizer.hasMoreTokens()) {
54.             result.addElement(tokenizer.nextToken());
55.         }
56. 
57.         return result;
58.     }
59. 
60.     protected boolean isTestClass(String classFileName) {
61.         return classFileName.endsWith(".class") && classFileName.indexOf(36) < 0 &&
    classFileName.indexOf("Test") > 0;
62.     }
63. 
64.     protected String classNameFromFile(String classFileName) {
65.         String s = classFileName.substring(0, classFileName.length() - SUFFIX_LENGTH);
66.         String s2 = s.replace(File.separatorChar, '.');
67.         return s2.startsWith(".")?s2.substring(1):s2;
68.     }
69. }
```

last-def and first-use

Call	A	B	C	D	E	F	G
Call Site In Codes	9	14	14	23	32	33	41

Call	**Variable**	**last-def**	**first-use**
A	classPath	8	14
B	classPath	13(8)	51
C	roots	14(49,54)	20
D	result	19	34,41
	classFileName	23	30
	classRoot	23	30
E	classFileName	29(23)	61
F	classFileName	29(23)	65
G	classRoot	29(23)	30
	classFileName	41	30
	result	29(19)	34,41

Sometimes last-def of variable is a bit difficult to identify. For example,

```
void A() {
  int a = 2;   // (1)
  B(a);
}
void B(int a){ // (2)
  C(a);// call site
}
void C(){}
```

So last-def of *a* in labeled call site is either (1) or (2). (1) takes definition outside the method into account. (2) only considers all the defs inside the method.
So in the table above, I put the first condition inside () when I identify last-def. But later, I will focus more on the first condition which takes definition outside the method. I think it is more important to see if it works.

Coupling Paths

Initial coupling paths are just paths connecting last-def with first-use. But sometimes one coupling path can cover another coupling path so that I do not need to count the shorter path. And also, sometimes the last-def of a variable is just a call site which is the initial of another coupling path. I should connect these two coupling paths otherwise I cannot study the whole coupling path property. So here are the final coupling paths:

For classPath,

- *1st* 8->9->(call A)->14->(call B)->51

For roots,

- *2nd* 49->53(while false)->57(return site)->14->(call C)->20
- *3rd* 49->53(while true)->54->57(return site)->14->(call C)-20

For result,

- *4th* 19->22(while true)->23->(call D)->31(if true)->32(if true)->34
- *5th* 19->22(while true)->23->(call D)->31(if false)->39(if true)->41->(call G)->31(if true)->32(if true)->34

For classRoot,

- *6th* 23->(call D)->30
- *7th* 23->(call D)->31(if false)->39(if true)->41->(call G)->30

For classFileName,

- *8th* 23->(call D)->30
- *9th* 23->(call D)->31(if true)->32->(call E)->61
- *10th* 23->(call D)->31(if true)->32(if true)->33->(call F)->65
- *11th* 23->(call D)->31(if false)->39(if true)->41->(call G)->30

Test number

Because I will assign each variable in every test, I can test several variables in one test. Therefore, I can reduce test number by testing several coupling paths in one test. Here are some conditions,

4th coupling path covers 6th, 8th, 9th,10th.

5th coupling path covers 6th, 7th, 8th, 11th.

So I need 5 tests to develop an adequate test set to cover your coupling paths.