

FindBugs for Java (<https://goo.gl/0QIqeI>)

```
module FindBugs
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

/**
 * Class is final but declares protected field
 *
 * @param class the class
 */
def CI_CONFUSED_INHERITANCE(class : ClassConcept) : Void = {
  assert class.isFinal == true
  member := class.member
  assert member instanceof FieldDeclaration
  assert member.visibility instanceof ProtectedVisibility
}

/**
 * Abstract class defines covariant equals() method
 *
 * @param class the class
 */
def EQ_ABSTRACT_SELF(class : ClassConcept) : Void = {
  method := class.member
  assert method instanceof InstanceMethodDeclaration

  // check method name and return type
  name := method.name
  assert method.returnType instanceof BooleanType
  assert eval(name.equals("equals"))

  // assert that the method has only one parameter
  assert count getParameters(method) == 1

  // and check that the parameter's type is the same as the class
  parameter := getParameters(method)
  parameterType := parameter.type
  assert parameterType instanceof ClassifierType
  assert class == parameterType.classifier
}

/**
 * Class defines covariant compareTo() method
 *
 * @param class the class
 */
def CO_SELF_NO_OBJECT(class : ClassConcept) : Void = {
  // check that the class implements the Comparable interface
  assert def comparables(class)

  method := class.member
  assert method instanceof InstanceMethodDeclaration

  // check method name and return type
  name := method.name
  assert method.returnType instanceof IntegerType
  assert eval(name.equals("compareTo"))

  // assert that the method has only one parameter
  count := count getParameters(method)
  assert eval(count == 1)

  // and check that the parameter's type is not Object
  parameter := getParameters(method)
  parameterType := parameter.type
  assert parameterType instanceof ClassifierType
  assert eval(!parameterType.classifier.name.equals(Object.class.getSimpleName()))
}
```

```

66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131

/**
  Class is Serializable but its superclass doesn't define a void constructor

  @param class the class
  */
def SE_NO_SUITABLE_CONSTRUCTOR(class : ClassConcept) : Void = {
  superClass := class.superclass.classifier
  assert def serializables(class)
  assert undef serializables(superClass)
  assert undef noArgConstructor(superClass)
}

/**
  Enumerates methods which perform dubious catching of IllegalMonitorStateException

  @param method the method
  */
def IMSE_DONT_CATCH_IMSE(method : BaseMethodDeclaration) : Void = {
  assert catchClause instanceof CatchClause
  type := catchClause.throwable.type
  assert eval(type.isInstanceOf(ClassifierType) && type : ClassifierType.classifier.name.equals(
    IllegalMonitorStateException.class.getSimpleName()))
  method := eval(catchClause.ancestor<concept = BaseMethodDeclaration>)
}

/**
  Enumerates unused fields

  @param field the field
  */
def UUF_UNUSED_FIELD(field : FieldDeclaration) : Void = {
  assert undef referencedVariables(field)
}

/**
  A class's finalize() method should have protected access, not public.

  @param class the class
  */
def FI_PUBLIC_SHOULD_BE_PROTECTED(class : ClassConcept) : Void = {
  method := class.member
  assert method instanceof InstanceMethodDeclaration

  // check method name and return type
  name := method.name
  type := method.returnType
  assert eval(name.equals("finalize") && type.isInstanceOf(VoidType))

  // assert that the method has no parameter
  assert count getParameters(method) == 0

  // assert that the method's visibility is public
  assert method.visibility instanceof PublicVisibility
}

/**
  Method invokes dangerous method runFinalizersOnExit

  @param class the class which contains the method
  */
def DM_RUN_FINALIZERS_ON_EXIT(class : ClassConcept) : Void = {
  assert call instanceof StaticMethodCall
  assert eval(call.staticMethodDeclaration.name.equals("runFinalizersOnExit"))
  classifier := call.classConcept
  assert eval(classifier.name.equals(Runtime.class.getSimpleName()) || classifier.name.equals(
    System.class.getSimpleName()))
  class := eval(call.ancestor<concept = ClassConcept>)
}

```

```

}
132
133
/**
134
Class defines equals() and does not define Object.hashCode()
135
@param class the class
136
*/
137
def HE_EQUALS_USE_HASHCODE(class : ClassConcept) : Void = {
138
139
    assert def getEqualsMethod(class)
140
    assert undef getHashCodeMethod(class)
141
}
142
143
/**
144
Enumerates binary operations which compare String objects using == or !=
145
@param op the binary operation
146
*/
147
def ES_COMPARING_STRINGS_WITH_EQ(op : BinaryOperation) : Void = {
148
149
    assert def ES_COMPARING_STRINGS_WITH_EQ_0(op)
150
    assert op instanceof NotEqualsExpression
151
} alt {
152
    assert def ES_COMPARING_STRINGS_WITH_EQ_0(op)
153
    assert op instanceof EqualsExpression
154
}
155
156
/**
157
Helper function of ES_COMPARING_STRINGS_WITH_EQ
158
Looks up string literals or string-typed variable references on the
159
two sides of the binary operation
160
@param op the binary operation
161
*/
162
private def ES_COMPARING_STRINGS_WITH_EQ_0(op : BinaryOperation) : Void = {
163
164
    (left, right) := getBinaryParts(op)
165
    assert left instanceof StringLiteral
166
    assert right instanceof StringLiteral
167
} alt {
168
    (left, right) := getBinaryParts(op)
169
    assert left instanceof VariableReference
170
    var := left.variableDeclaration
171
    assert def getVariablesWithStringType(var)
172
    assert right instanceof StringLiteral
173
} alt {
174
    (left, right) := getBinaryParts(op)
175
    assert right instanceof VariableReference
176
    var := right.variableDeclaration
177
    assert def getVariablesWithStringType(var)
178
    assert left instanceof StringLiteral
179
} alt {
180
    (left, right) := getBinaryParts(op)
181
    assert left instanceof VariableReference
182
    v1 := left.variableDeclaration
183
    assert def getVariablesWithStringType(v1)
184
    assert right instanceof VariableReference
185
    v2 := right.variableDeclaration
186
    assert def getVariablesWithStringType(v2)
187
}
188
189
// HELPER FUNCTIONS
190
191
/**
192
Returns the left and right hand side expressions of a binary operation
193
@param op the binary operation
194
@return the left and right hand side expressions
195
*/
196
private def getBinaryParts(op : BinaryOperation) : (Expression, Expression) = {
197
    left := op.leftExpression
198
199

```

```

    right := op.rightExpression
    return (left, right)
}

/**
Enumerates the variable declarations with string type

@param var the variable declaration
*/
private def getVariablesWithStringType(var : VariableDeclaration) : Void = {
    assert var.type instanceof StringType
}

/**
Returns the equals method of a class

@param class the class
@return the equals method
*/
private def getEqualsMethod(class : ClassConcept) : InstanceMethodDeclaration = {
    method := class.member
    assert method instanceof InstanceMethodDeclaration

    // check method name and return type
    name := method.name
    assert method.returnType instanceof BooleanType
    assert eval(name.equals("equals"))

    // assert that the method has only one parameter
    assert count getParameters(method) == 1

    // and check that the parameter's type is the same as the class
    parameter := getParameters(method)
    parameterType := parameter.type
    assert parameterType instanceof ClassifierType
    assert eval(parameterType.classifier.name.equals(Object.class.getSimpleName()))

    return method
}

/**
Returns the hashCode method of a class

@param class the class
@return the hashCode method
*/
private def getHashCodeMethod(class : ClassConcept) : InstanceMethodDeclaration = {
    method := class.member
    assert method instanceof InstanceMethodDeclaration

    // check method name and return type
    name := method.name
    assert method.returnType instanceof IntegerType
    assert eval(name.equals("hashCode"))

    // assert that the method has only one parameter
    assert count getParameters(method) == 1

    return method
}

/**
Enumerates variables which are referenced

@param variable the variable
*/
private def referencedVariables(variable : VariableDeclaration) : Void = {
    assert reference instanceof VariableReference

```

assert variable == reference.variableDeclaration	268
}	269
// returns the super class of given class	270
/**	271
Returns the superclass of a class	272
	273
@param c the subclass	274
@return the superclass	275
*/	276
private def getSuperClass(c : ClassConcept) : ClassConcept = {	277
s := c.superclass.classifier	278
assert s instanceOf ClassConcept	279
return s	280
}	281
	282
/**	283
Returns the no-arg constructor of a class.	284
	285
@param class the class	286
@return the no-arg constructor	287
*/	288
private def noArgConstructor(class : ClassConcept) : ConstructorDeclaration = {	289
constructor := class.member	290
assert constructor instanceOf ConstructorDeclaration	291
count := count getParameters(constructor)	292
assert count == 0	293
return constructor	294
}	295
	296
/**	297
Enumerates the classes which implement the Comparable interface	298
	299
@param class the class	300
*/	301
private def comparables(class : ClassConcept) : << ... >> = {	302
interface := class.implementedInterface	303
assert eval(interface.classifier.name.equals(Comparable.class.getSimpleName()))	304
}	305
	306
/**	307
Enumerates the classes which implement the Serializable interface	308
	309
@param class the class	310
*/	311
private def serializables(class : ClassConcept) : << ... >> = {	312
interface := class.implementedInterface	313
assert eval(interface.classifier.name.equals(Serializable.class.getSimpleName()))	314
}	315
	316
/**	317
Returns the parameter(s) of a method	318
	319
@param method the method	320
@return the parameter(s)	321
*/	322
private def getParameters(method : BaseMethodDeclaration) : ParameterDeclaration = {	323
return method.parameter	324
}	325
	326