FindBugs for Java

```
module FindBugs

/**
  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()))
}
```

```
/**
  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 performdDubious 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>)
```

```
}

/**
  Class defines equals() and does not define Object.hashCode()

  @param class the class
 */
def HE_EQUALS_USE_HASHCODE(class : ClassConcept) : Void = {
  assert def getEqualsMethod(class)
  assert undef getHashCodeMethod(class)
}

/**
  Enumerates binary operations which compare String objects using == or !=

  @param op the binary operation
 */
def ES_COMPARING_STRINGS_WITH_EQ(op : BinaryOperation) : Void = {
  assert def ES_COMPARING_STRINGS_WITH_EQ_0(op)
  assert op instanceOf NotEqualsExpression
} alt {
  assert def ES_COMPARING_STRINGS_WITH_EQ_0(op)
  assert op instanceOf EqualsExpression
}

/**
  Helper function of ES_COMPARING_STRINGS_WITH_EQ
  Looks up string literals or string-typed variable references on the
  two sides of the binary operation

  @param op the binary operation
 */
private def ES_COMPARING_STRINGS_WITH_EQ_0(op : BinaryOperation) : Void = {
  (left, right) := getBinaryParts(op)
  assert left instanceOf StringLiteral
  assert right instanceOf StringLiteral
} alt {
  (left, right) := getBinaryParts(op)
  assert left instanceOf VariableReference
  var := left.variableDeclaration
  assert def getVariablesWithStringType(var)
  assert right instanceOf StringLiteral
} alt {
  (left, right) := getBinaryParts(op)
  assert right instanceOf VariableReference
  var := right.variableDeclaration
  assert def getVariablesWithStringType(var)
  assert left instanceOf StringLiteral
} alt {
  (left, right) := getBinaryParts(op)
  assert left instanceOf VariableReference
  v1 := left.variableDeclaration
  assert def getVariablesWithStringType(v1)
  assert right instanceOf VariableReference
  v2 := right.variableDeclaration
  assert def getVariablesWithStringType(v2)
}

// HELPER FUNCTIONS

/**
  Returns the left and right hand side expressions of a binary operation

  @param op the binary operation
  @return the left and right hand side expressions
 */
private def getBinaryParts(op : BinaryOperation) : (Expression, Expression) = {
  left := op.leftExpression
```

```
  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
}

// returns the super class of given class
/**
  Returns the superclass of a class

  @param c the subclass
  @return the superclass
 */
private def getSuperClass(c : ClassConcept) : ClassConcept = {
  s := c.superclass.classifier
  assert s instanceOf ClassConcept
  return s
}

/**
  Returns the no-arg constructor of a class.

  @param class the class
  @return the no-arg constructor
 */
private def noArgConstructor(class : ClassConcept) : ConstructorDeclaration = {
  constructor := class.member
  assert constructor instanceOf ConstructorDeclaration
  count := count getParameters(constructor)
  assert count == 0
  return constructor
}

/**
  Enumerates the classes which implement the Comparable interface

  @param class the class
 */
private def comparables(class : ClassConcept) : << ... >> = {
  interface := class.implementedInterface
  assert eval(interface.classifier.name.equals(Comparable.class.getSimpleName()))
}

/**
  Enumerates the classes which implement the Serializable interface

  @param class the class
 */
private def serializables(class : ClassConcept) : << ... >> = {
  interface := class.implementedInterface
  assert eval(interface.classifier.name.equals(Serializable.class.getSimpleName()))
}

/**
  Returns the parameter(s) of a method

  @param method the method
  @return the parameter(s)
 */
private def getParameters(method : BaseMethodDeclaration) : ParameterDeclaration = {
  return method.parameter
}
```