Name	Definition	Example	Differences

Big Integer Instantiation	Don't create instances of already existing BigInteger (BigInteger.ZERO, BigInteger.ONE) andfor Java 1.5 onwards, BigInteger.TEN and BigDecimal (BigDecimal.ZERO, BigDecimal.ONE, BigDecimal.TEN).	**FaultHunter-only true positives**
		testValue(rootNode, 1.0d, 10.0f);
		testValue(rootNode, 1.0d, 10.0d);
		testValue(rootNode, 1.0d, newBigDecimal("1.0"));
		o1.setEstimatedPrice(new BigDecimal(1.0d));
		BigDecimalrefundTotal = new BigDecimal(0.0);
			FaultHunter finds those BigInteger and BigDecimal initializations, which occur inside a method call, or which do not consist solely of digits (e.g. 1.0d).

Empty Catch Block	Empty Catch Block finds instances where an exception is caught, but nothing is done. In most circumstances, this swallows an exception which should either be acted on or reported.	**FaultHunter-only true positives**
		Thread t = new Thread(r);
		t.start();
		t.join();
		} catch (InterruptedException e) {
		// noop
		} finally {
		reconnectLock.unlock();
		}
			FaultHunter finds those empty catch blocks, which only contain a single one-line comment.

Empty If Stmt	Empty If Statement finds instances where a condition is checked but nothing is done about it.	**FaultHunter-only true positives:**
		if (jj_2_30(2)) {
		;
		} else {
			FaultHunter finds those empty if blocks, which only contain a single empty statement.

If Else Stmts Must Use Braces	Avoid using if..else statements without using curly braces. If the code formatting or indentation is lost then it becomes difficult to separate the code being controlled from the rest.	**FaultHunter-only true positives:**
		if (foo)
		else
		x = x-1;
			FaultHunter finds those rule violations which are only considered "Empty If Statements" by PMD.

Avoid Instanceof Checks In Catch Clause	Each caught exception type should be handled in its own catch clause.	**FaultHunter-only true positives**
		catch(Throwable jjte000){
		if (jjte000 instanceof RuntimeException) {
		{if (true) throw (RuntimeException)jjte000;}
		}
			PMD doesn't find cases where RuntimeException is compared.

Name	Definition	Example	Differences

Close Resource	Ensure that resources (like Connection, Statement, and ResultSet objects) are always closed after use.	**PMD-only false positives**
		ResultSet rs = pstmt.executeQuery();
		List<PrimaryDataStoreVO> pools =
		newArrayList<PrimaryDataStoreVO>();
		while (rs.next()) {
		  pools.add(toEntityBean(rs, false));
			If the resource is used as a parameter to a function or as a return statement, FaultHunter treats the connection as closed.

		FaultHunter-only true positives PreparedStatement pstmt = null;	PMD misses the kind of types which are child classes of the specific types (e.g. PreparedStatement).
Position Literals First In Comparisons	Position literals first in comparisons, if the second argument is null then NullPointerExceptions can be avoided, they will just return false.	**FaultHunter-only true positives** boolean connected = words[1].equals("0");	PMD cannot find cases where an element of an array is compared to a constant value using equals().

		if (!found&&f.getName().equals("template.properties")) {
		  found = true;
		}
			PMD cannot find cases where the caller is not an identifier, but a complex type.

		(userPublicTemplateEnabled == null || userPublicTemplateEnabled.equals("false") ? false : true));	PMD cannot find certain complex comparisons either.

		if (cidr.equals(NetUtils.ALL_CIDRS)) {
		  continue;
		}
			FaultHunter finds cases where an object is compared with a static final variable.

Name	Definition	Example	Differences

Preserve Stack Trace	Throwing a new exception from a catch block without passing the original exception into thenew exception will cause the original stack trace to be lost making it difficult to debug effectively.	**PMD-only false positives**
		} catch (NaException e) {
		throw new ServerException("Unabletocreatevolume", e);
			PMD incorrectly finds cases where a custom Exception class is instantiated with an exception as parameter.

		FaultHunter-only true positives
		}catch(java.lang.ClassCastException e){
		// we cannot intantiate the class
		throw f;
		} catch (java.lang.ClassNotFoundException e) {
		// we cannot intantiate the class
		throw f;
			FaultHunter finds cases when a specific exception is thrown many times.

		FaultHunter-only true positives
		} catch (Exception e) {
		_selector.close();
		throw new IOException("SSL: Failtoinit SSL! " + e);
		}
			FaultHunter finds cases where the message of the caught exception is appended to the message of newly thrown exception.

Simple Date Format Needs Locale	Be sure to specify a Locale when creating SimpleDateFormat instances to ensure that locale-appropriateformatting is used.	**PMD-only false positives** import com.ibm.icu.text.SimpleDateFormat; ... SimpleDateFormat dateFormat2 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS");	PMD incorrectly finds cases where the developer instantiates a different, custom SimpleDateFormat class not necessarily requiring a locale.

Unnecessary Local Before Return	Avoid the creation of unnecessary local variables: consider simply returning the value vs storing it in a local variable.	**FaultHunter-only true positives**
		public int ulbr(){
		int unnecessary = 6; // WARNING
		return unnecessary;
		}
			FaultHunter only finds cases where the declaration of a variable is directly before the return statement and where the return operand is a simple identifier.

Name	Definition	Example	Differences
Use Locale With Case Conversions	When doing String.toLowerCase()/toUpperCase() conversions, use Locales to avoids problems with languages thathave unusual conventions, i.e. Turkish.	**PMD-only false positives** super(String.format("All of thefollowingarguments must be set: %1$s, %2$s, %3$s", "--" + CommandLineOptions.Client.WSDLDNS.toLowerCase(), "--" + CommandLineOptions.Client.WSDLPORT.toLowerCase(), "--" + CommandLineOptions.Client.ROOTCONTEXT.toLowerCase()));	PMD incorrectly finds cases where the toLowerCase method was not called on a String object.
		FaultHunter-only true positives if (resourceAdapterId != newResourceId) {	PMD cannot find cases, where the caller expression is more complex than a simple identifier.
Method Naming Conventions	Method names should always begin with a lower case character, and should not contain underscores.	**FaultHunter-only true positives** final public List<Object>JSONArray() throws ParseException {	In multiple cases PMD cannot find methods starting with capital letters and those which contatin the underscore (_) character.
		FaultHunter-only true positives public void ReInit(java.io.InputStreamstream)	
		FaultHunter-only true positives private String _formatPackage(Strings Package_) {	

At Least One Constructor	Each class should declare at least one constructor.	**PMD-only false positives**
		public class StartTestRoundResponse {
		
		}
			PMD incorrectly finds every empty class.

Name	Definition	Example	Differences

Suspicious Hashcode Method Name	The method name and return type are suspiciously close to hashCode(), which may denote an intentionto override the hashCode() method.	**FaultHunter-only true positives**
		Public void hashCode(){ // WARNING
		int x = 5;
		return x;
		}
			PMD only examines if the method name has typos or not (lowercase, uppercase character check). FaultHunter checks if there is a feedback between the hashCode() method and the object's hash code.

		FaultHunter-only true positives
		public int hashCode(int i){ // WARNING
		int x = 5 + i;
		return x;
		}	

		FaultHunter-only true positives
		public int hashcode(){ // WARNING
		int x = 5;
		return x;
		}	

Array Is Stored Directly	Constructors and methods receiving arrays should clone objects and store the copy.This prevents future changes from the user from affecting the original array.	**PMD-only false positives**
		Token(TokenKind tokenKind, char[] tokenData, int pos, int endpos) {
		this(tokenKind,pos,endpos);
		this.data = new String(tokenData);
		}
			PMD incorrectly finds cases where there is a member on the left side of the equation and a complex expression containing an array-parameter on the right.

		PMD-only false positives
		public MockHttpInputMessage(byte[] contents) {
		Assert.notNull(contents, "'contents' must not be null");
		this.body = new ByteArrayInputStream(contents);
		}	

		FaultHunter-only true positives
		public void setTemplateLoaderPaths(String...
		templateLoaderPaths) {
		this.templateLoaderPaths = templateLoaderPaths;
		}
			PMD can’t find varargs (variable arguments).

Name	Definition	Example	Differences
Avoid Catching NPE	Code should never throw NullPointerExceptions under normal circumstances. A catch block may hide the original error, causing other, more subtle problems later on.	**FaultHunter-only true positives:** } catch (java.lang.NullPointerException npe) {	PMD cannot find java.lang.NullPointerExceptions.
Avoid Catching Throwable	Catching Throwable errors is not recommended since its scope is very broad. It includes runtime issues such as OutOfMemoryError that should be exposed and managed separately.	**FaultHunter-only true positives** } catch (java.lang.Throwable ivjExc) {	PMD cannot find java.lang.Throwable.

Avoid Rethrowing Exception	Catch blocks that merely rethrow a caught exception only add to code size and runtime complexity. In cases when the rethrowed exception can be caught by following catches are allowed.	**PMD-only false positives**
		} catch (CodecException e) {
		throw e;
		} catch (Throwable t) {
		throw new DecoderException(t);
		
		PMD-only false positives
		} catch (RuntimeException re) {
		throw re;
		} catch (Throwable t) {
		/* ignored */
		}
			FaultHunter doesn't indicate some throws on purpose, when the goal seems to be making the catch block ignore the given type of exception.

Avoid Throwing Null Pointer Exception	Avoid throwing NullPointerExceptions. These are confusing because most people will assume that thevirtual machine threw it. Consider using an IllegalArgumentException instead; this will beclearly seen as a programmer-initiated exception. The rule also warns on null pointer exception instantiations.	**PMD-only false positives:**
		throw new ClassNotFoundException("null classname", new NullPointerException());
		
		PMD-only false positives:
		result = handleException(currentJob(), new NullPointerException());
		
		FaultHunter-only true positives:
		throw new java.lang.NullPointerException("Codingerror; Don't try to mark/rewind an indexed DTM");
			PMD incorrectly finds every new NullPointerException, for example if the NullPointerException is passed as a parameter.
			
			FaultHunter finds exceptions given with full package path.

Name	Definition	Example	Differences
Avoid Throwing Raw Exception Types	Avoid throwing certain exception types. Rather than throw a raw RuntimeException, Throwable,Exception, or Error, use a subclassed exception or error instead. The rule also warns on raw exception instantiations.	**PMD-only false positives** data.errors = new Error [count];	FaultHunter does not find error array initializations.

		FaultHunter-only true positives
		} catch (javax.xml.stream.XMLStreamException e) {
		throw new java.lang.Exception(e);
		}
			PMD can't find cases, when exceptions are given with the complete package path.

		FaultHunter-only true positives if (resourceAdapterId != new ResourceId) {	PMD cannot find cases where comparing occurs with a non-literal.
Unused Imports	Avoid unused import statements. This rule will find unused on demand imports, i.e. import com.foo.*.	**PMD-only false positives** import staticorg.junit.Assert.*; ... assertNotNull(...);	PMD usually can't find imports given with wildcard.
		FaultHunter-only true positives xl2 = (x1+x2)>>1;// WARNING	FaultHunter indicates cases which cannot be found by PMD.
		FaultHunter-only true positives ObjectInfoinfo = new ObjectInfo(obj, new CoordinateSystem(orig, zdir, ydir), "Light "+(counter++));	
		FaultHunter-only true positives out.writeByte((data[i]>>8)&0xFF);	

Too Many Methods	A class with too many methods is probably a good target for refactoring, in order to reduce its complexity and find a way to have more fine grained objects.	**FaultHunter identifies as getter:**
		public String getString1() {
		return this.string1;
		}
		FaultHunter identifies as setter:
		public void setString1(String string1) {
		this.string1 = string1;
		}
			PMD and FaultHunter recognize getters and setters differently. PMD scans for method names containing "get" and "set", while FaultHunter deems methods as getters or setters based on syntax.