Java Optimization Rules

Rules available in this category:

Rule 1: Use_String_length_to_compare_empty_string_variables

Severity: High
Rule: The String.equals() method is overkill to test for an empty string. It is quicker to test if the length of the string is 0.
Reason: The String.equals() method is overkill to test for an empty string. It is quicker to test if the length of the string is 0.

Usage Example:

```
package com.rule;
class Use_String_length_to_compare_empty_string_violation
{
        public boolean isEmpty(String str)
        {
                return str.equals("");          // VIOLATION
        }
}
```

Should be written as:

```
package com.rule;
class Use_String_length_to_compare_empty_string_correction
{
        public boolean isEmpty(String str)
        {
                return str.length()==0;         // CORRECTION
        }
}
```

Reference: http://www.onjava.com/pub/a/onjava/2002/03/20/optimization.html?page=4
http://www.javaperformancetuning.com/tips/rawtips.shtml

Rule 2: Avoid_using_Math_class_methods_on_constants

Severity: Medium
Rule: It is quicker to determine the value statically.
Reason: It is quicker to determine the value statically.

Usage Example:

```
public class Test
{
        public void fubar()
        {
                double a;
                a = Math.abs(1.5); // VIOLATION
        }
}
```

Should be written as:

```
public class Test
{
        public void fubar()
        {
                double a;
                a =1.5; // FIXED
        }
}
```

Reference: Not Available.

Rule 3: Avoid_consecutively_invoking_StringBuffer_append_with_string_literals

Severity: Medium
Rule: Doing so reduces the classfile size by 17 bytes and eliminates a few instructions.
Reason: Doing so reduces the classfile size by 17 bytes and eliminates a few instructions.

Usage Example:

```
public class Test
{
        private void fubar()
        {
                StringBuffer buf = new StringBuffer();
                buf.append("Hello").append(" ").append("World");  // VIOLATION
         }
}
```

Should be written as:

```
public class Test
{
        private void fubar()
        {
                StringBuffer buf = new StringBuffer();
                buf.append("Hello World");// FIXED
        }
}
```

Reference: Not Available.

Rule 4: Avoid_creating_thread_without_run_method

Severity: High
Rule: A Thread which is created without specifying a run method does nothing other than a delay in performance.
Reason: A Thread which is created without specifying a run method does nothing other than a delay in performance.

Usage Example:

```
public class Test
{
 public void method() throws Exception
 {
        new Thread().start();  //VIOLATION
 }
}
```

Should be written as:

```
public class Test
{
 public void method(Runnable r) throws Exception
 {
        new Thread(r).start();  //FIXED
 }
}
```

Reference: Not Available.

Rule 5: Always_reuse_immutable_constant_objects_for_better_memory_utilization

Severity: Medium
Rule: Creation of constant immutable objects that are not assigned to static final variables lead to unnecessary memory consumption.
Reason: Creation of constant immutable objects that are not assigned to static final variables lead to unnecessary memory consumption.

Usage Example:

```
public class Test
{
  protected Object[] getObjects()
 {
  return new Object[0];  // VIOLATION
  }

 publicstatic Integer convertToInt(String s)
 {
  if (s == null || s.length() == 0)
          {
return new Integer(-1);  // VIOLATION
        }
  else
          {
return new Integer(s);
        }
 }
}
```

Should be written as:

```
public class Test
{
 public static final Object[] NO_OBJECTS = new Object[0];

 protected Object[] getObjects()
{
  return NO_OBJECTS;  // FIXED
 }

 private static final Integer INT_N1 = new Integer(-1);

 public static Integer convertToIn(String s) {
  if (s == null || s.length() == 0)
          {
return INT_N1;  // FIXED
        }
  else
          {
return new Integer(s);
        }
 }
}
```

Reference: Not available.

Rule 6: Avoid_constant_expressions_in_loops

Severity: Medium
Rule: It is more efficient to either simplify these expressions or move them outside the body of the loop.
Reason: It is more efficient to either simplify these expressions or move them outside the body of the loop.

Usage Example:

```
public class Test
{
 public static final boolean TRUE= true;
 public static final boolean FALSE= false;
 public static final int FOO= 7;

 public void myMethod()
 {
  int[] x= new int[10];
  int j= 0;

  for(int i=0; i<10; i++)
         {
x[0]= 7+(FOO+9);// VIOLATION

for(j=0; TRUE||FALSE;)// VIOLATION
          {
}
  }
 }
}
```

Should be written as:

```
public class Test
{
 public void myMethod()
 {
  int[] x= new int[10];
  int j= 0;

  x[0]= 7+(FOO+9);//FIXED
  for(int i=0; i<10; i++)
         {
for(j=0; TRUE;)//FIXED
          {
}
  }
 }
}
```

Reference: Not Available.

Rule 7: Avoid_invoking_time_consuming_methods_in_loop

Severity: Medium
Rule: Moving method calls which may take a long time outside of loops can improve performance.
Reason: Moving method calls which may take a long time outside of loops can improve performance.

Usage Example:

```
import java.util.Arrays;


public class Test
{
 public int[] sortArray(int[] a)
{
  for(int i=0; i<100; i++)
  {
Arrays.sort(a);  // VIOLATION
//Some other code
  }
  return a;
 }
}
```

Should be written as:

```
public class Test
{
 public int[] sortArray(int[] a)
{
  Arrays.sort(a);  // FIXED
  for(int i=0; i<100; i++)
 {
//Some other code
  }
  return a;
 }
}
```

Reference: Not Available.

Rule 8: Avoid_duplication_of_code

Severity: High
Rule: Avoid duplication of code.
Reason: Avoid duplication of code.

Usage Example:

```
package com.rule;

public class Avoid_duplication_of_code_violation
{
        public void method()
        {
                int x = getValue();

                if(x > 10)
                {                               // Violation.
                        int j = i + 10;
                        int k = j * 2;
                        System.out.println(k);
                }
                else if( x < 20 )
                {                               // Violation.
                        int j = i + 10;
                        int k = j * 2;
                        System.out.println(k);
                }
        }

}
```

Should be written as:

Reference: Reference not available.

Rule 9: Use_entrySet_instead_of_keySet

Severity: Medium
Rule: Use entrySet() instead of keySet().
Reason: Use entrySet() instead of keySet().

Usage Example:

```
package com.rule;
```

```
import java.util.Map;
import java.util.Set;
import java.util.HashMap;
import java.util.Iterator;

public class Use_entrySet_instead_of_keySet_violation
{
        public void method()
        {
                Map m = new HashMap();
                Iterator it = m.keySet().iterator();
                Object key = it.next();
                Object v = m.get(key);          // Violation

        }

}
```

Should be written as:

```
package com.rule;

public class Use_entrySet_instead_of_keySet_correction
{
        public void method()
        {
                Map m = new HashMap();
                Set set = m.entrySet();         //Correction.
                Object keyValuePair = it.next();
        }

}
```

Reference: Reference not available.

Rule 10: Ensure_efficient_removal_of_elements_in_a_collection

Severity: Medium
Rule: Searching a collection to fetch the element to remove is inefficient.
Reason: Searching a collection to fetch the element to remove is inefficient.

Usage Example:

```
public class Test
{
        public void someMethod(Collection collection)
        {
                Iterator iter = collection.iterator();
                while (iter.hasNext())
                {
                        Object element = iter.next();
                        collection.remove(element); // VIOLATION
                }
        }
}
```

Should be written as:

```
public class Test
{
        public void someMethod(Collection collection)
        {
                Iterator iter = collection.iterator();
                while (iter.hasNext())
                {
                        iter.remove(); // FIXED
                }
        }
}
```

Reference: Not available.

Rule 11: Ensure_efficient_removal_of_map_entries

Severity: Medium
Rule: Searching a keyset/entryset to fetch the key to remove the element is inefficient.
Reason: Searching a keyset/entryset to fetch the key to remove the element is inefficient.

Usage Example:

```
import java.util.*;

public class Test
{
 public void someMethod(HashMap collection)
 {
  Set keySet = collection.keySet();
  Iterator keyIter = keySet.iterator();
  while (keyIter.hasNext())
          {
Object key = keyIter.next();
collection.remove(key); // VIOLATION
  }
 }
}
or another case when we iterate on entry set:
```

```
public class Test
{
 public void someMethod(HashMap collection)
 {
  Set entrySet = collection.entrySet();
  Iterator entriesIter = entrySet.iterator();
  while (entriesIter.hasNext())
          {
(Map.Entry) entry = (Map.Entry)entriesIter.next();
Object key = entry.getKey();
collection.remove(key); // VIOLATION
  }
 }
}
```

Should be written as:

```
public class Test
{
 public void someMethod(HashMap collection)
 {
  Set keySet = collection.keySet();
  Iterator keyIter = keySet.iterator();
  while (keyIter.hasNext())
          {
keyIter.remove(); // FIXED
  }
 }
}
```

Reference: Not available.

Rule 12: Ensure_efficient_iteration_over_map_entries

Severity: Medium
Rule: Using a keyset to iterate over a map, and then requesting values for each key is inefficient.
Reason: Using a keyset to iterate over a map, and then requesting values for each key is inefficient.

Usage Example:

```
import java.util.Iterator;
import java.util.Map;

public class Test
{
 public void inefficientIteration(Map map)
 {
  Iterator iter = map.keySet().iterator();
  while (iter.hasNext()) {
Object key = iter.next();
Object value = map.get(key); // VIOLATION
  }
 }
}
```

Should be written as:

```
import java.util.Iterator;
import java.util.Map;

public class Test
{
 public void efficientIteration(Map map)
 {
  Iterator iter = map.entrySet().iterator();
  while (iter.hasNext()) {
Map.Entry entry = (Map.Entry)iter.next();
Object key = entry.getKey();
Object value = entry.getValue(); // FIXED
  }
 }
}
```

Reference: Not available.

Rule 13: Avoid_using_java_lang_Class_forName

Severity: Medium
Rule: Decreases performance and could cause possible bugs.
Reason: Decreases performance and could cause possible bugs.

Usage Example:

```
public class Test
{
        private void foo()
        {
                try
                {
                        System.out.println(Class.forName("java.lang.Integer")
                                        .getName()); // VIOLATION
                }
                catch ( ClassNotFoundException e )
                {
                        e.printStackTrace();
                }
        }

}
```

Should be written as:

```
public class Test
{
        private void foo()
        {
                System.out.println(java.lang.Integer.class.getName()); // CORRECTION
        }

}
```

Reference: No references available.

Rule 14: Do_not_declare_members_accessed_by_inner_class_private

Severity: High
Rule: Do not declare members accessed by inner class private.
Reason: Do not declare members accessed by inner class private.

Usage Example:

```
package com.rule;
public class Do_not_declare_members_accessed_by_inner_class_private_violation
{
        private int iVar = 0; // Violation
        class inner
        {
                int var2;
                public void foo()
                {
                        var2 = iVar;
                        // ...
                }
        }
}
```

Should be written as:

```
package com.rule;
public class Do_not_declare_members_accessed_by_inner_class_private_correction
{
        int iVar = 0; // Correction
        class inner
        {
                int var2;
                public void foo()
                {
                        var2 = iVar;
                        // ...
                }
        }
}
```

Reference: http://www.glenmccl.com/jperf/#Sample1

Rule 15: Avoid_synchronized_modifier_in_method

Severity: High
Rule: Avoid synchronized modifier in method for performace reasons
Reason: Avoid synchronized modifier in method for performace reasons

Usage Example:

```
package com.rule;
import java.io.ObjectOutputStream;
import java.io.IOException;

class Avoid_synchronized_modifier_in_method_violation
{
        public synchronized void writeToStream(String s)throws IOException              // VIOLATION
        {
                //....
        }
}
```

Should be written as:

```
package com.rule;

import java.io.ObjectOutputStream;
import java.io.IOException;

class Avoid_synchronized_modifier_in_method_correction
{
        public void writeToStream(String s)throws IOException           // CORRECTION
        {
                //....
                synchronized (this)// CORRECTION
                {
                        //....
                }

                //....
        }
}
```

Reference: Reference Not Available.

Rule 16: Avoid_empty_if

Severity: Low
Rule: Avoid empty "if" block structure.
Reason: Avoid empty "if" block structure.

Usage Example:

```
package com.rule;

class Avoid_empty_if_violation
{
        public void method()
        {
                final int ZERO = 0;
                int i = 10;
                if (i < ZERO)           // VIOLATION
                {
                }
                i = ZERO;
        }
}
```

Should be written as:

```
package com.rule;

class Avoid_empty_if_correction
{
        public void method()
        {
                final int ZERO = 0;
                int i = 10;
                /*
                if (i < ZERO)           // CORRECTION
                {
                }
                */
                i = ZERO;
        }
}
```

Reference: Reference Not Available.

Rule 17: Avoid_empty_static_initializer

Severity: Low
Rule: Since the static initializer contains no code, it can be safely removed.
Reason: Since the static initializer contains no code, it can be safely removed.

Usage Example:

```
public class Test
{
        static  // VIOLATION
        {
                // empty
        }
}
```

Should be written as:

```
public class Test
{
        // ...
}
```

Reference: Not Available.

Rule 18: Avoid_unnecessary_if

Severity: Medium
Rule: Avoid unnecessary if statements.
Reason: Avoid unnecessary if statements.

Usage Example:

```
package com.rule;

public class Avoid_unnecessary_if_Violation
{
        public void method()
        {
                if (true)               // Violation.
                {
                 //Some Code ...
                }
                if (!true)              // Violation.
                {
                        //Some Code ...
                }
        }
}
```

Should be written as:

```
package com.rule;

public class Avoid_unnecessary_if_Correction
{
        public void method()
        {
                boolean flag = true;

                if (flag)               //Correction.
                {
                 //Some Code ...
                }
        }

}
```

Reference: Reference not available.

Rule 19: Avoid_unnecessary_parentheses

Severity: Medium
Rule: Avoid unnecessary parentheses in an expression.
Reason: Avoid unnecessary parentheses in an expression.

Usage Example:

```
package com.rule;

public class Avoid_unnecessary_parentheses_violation
{
        public void method()
        {
                if((method()))          // Violation.
                {
                        // Do Something..
                }
        }

}
```

Should be written as:

```
package com.rule;

public class Avoid_unnecessary_parentheses_correction
{
        public void method()
        {
                if(method())            //Correction.
                {
                        // Do Something..
                }
        }

}
```

Reference: Reference not available.

Rule 20: Avoid_unnecessary_implementing_Clonable_interface

Severity: Medium
Rule: Avoid unnecessary implementing Clonable interface.
Reason: Avoid unnecessary implementing Clonable interface.

Usage Example:

```
package com.rule;

public class Avoid_unnecessary_implementing_Clonable_interface_violation implements Clonable
{
        public void method()            //Violation.
        {

        }

}
```

Should be written as:

```
package com.rule;

public class Avoid_unnecessary_implementing_Clonable_interface_correction implements Clonable
{
        public void method()
        {
                void_unnecessary_implementing_Clonable_interface theClone = new void_unnecessary_implementing_Clonable_interface();

                Object o = theClone.clone();            //Correction.
        }

}
```

Reference: Reference not available.

Rule 21: Avoid_using_MessageFormat

Severity: Low
Rule: Avoid using MessageFormat as it is slow.
Reason: Avoid using MessageFormat as it is slow.

Usage Example:

```
package com.rule;

import java.text.MessageFormat;

public class Avoid_using_MessageFormat_violation
{
	public void method()
	{
		final int N = 25000;
		Object argvec[] = new Object[2];
		MessageFormat f = new MessageFormat("The square of {0,number,#} is {1,number,#}");
		for (int i = 1; i <= N; i++)
		{
			argvec[0] = new Integer(i);
			argvec[1] = new Integer(i * i);
			String s = f.format(argvec);
			System.out.println(s);
		}
	}
}
```

Should be written as:

```
package com.rule;
public class Avoid_using_MessageFormat_correction
{
	public void method()
	{
		final int N = 25000;
		String s;
		for (int i = 1; i <= N; i++)
		{
			s = "The square of " + i + " is " + (i * i);
			System.out.println(s);
		}
	}
}
```

Reference: Reference Not Available.

Rule 22: Avoid_writeByte_method_in_loop

Severity: High
Rule: Avoid writing single byte in loop
Reason: Avoid writing single byte in loop

Usage Example:

```
package com.rule;
class Avoid_writeByte_method_violation
{
	public static void main(String args[])
	{
		final int ZERO = 0;
		final int ONE = 1;
		final int TEN = 10;
		String strFileName = "C:\\demo.java"; //$NON-NLS-1$
		try
		{
			java.io.FileOutputStream fos = new java.io.FileOutputStream(strFileName);
			java.io.DataOutputStream ds = new java.io.DataOutputStream(fos);

			int i = ZERO;
			while(i < TEN)
			{
				ds.writeByte(ONE);      // VIOLATION
				i++;
			}

			for(i=ZERO; i<TEN; i++)
			{
				ds.writeByte(ONE);      // VIOLATION
			}

			i = ZERO;

			do
			{
				ds.writeByte(ONE);      // VIOLATION
				i++;
			}
			while(i<TEN);
		}
		catch(java.io.IOException e)
		{
			e.printStackTrace();
		}
	}
}
```

Should be written as:

```
package com.rule;
class Avoid_writeByte_method_correction
{
        public static void main(String args[])
        {
                final int ZERO = 0;
                final char ONE = '1';
                final int TEN = 10;

                String strFileName = "C:\\demo.java"; //$NON-NLS-1$
                byte bArr[] = new byte[10];

                try
                {
                        java.io.FileOutputStream fos = new java.io.FileOutputStream(strFileName);
                        java.io.DataOutputStream ds = new java.io.DataOutputStream(fos);

                        int i = ZERO;
                        while(i < TEN)
                        {
                                bArr[i] = ONE;
                                i++;
                        }

                        ds.write(bArr, ZERO, bArr.length);      // CORRECTION
                }
                catch(java.io.IOException e)
                {
                        e.printStackTrace();
                }
        }
}
```

Reference: Reference Not Available.

Rule 23: Avoid_repeated_casting

Severity: High
Rule: Avoid repeated casting by casting it once and keeping its reference
Reason: Avoid repeated casting by casting it once and keeping its reference

Usage Example:

```
package com.rule;

import java.awt.Component;
import java.awt.TextField;

class Avoid_repeated_casting_violation
{
        public void method(Component comp)
        {
                ((TextField) comp).setText("");         // VIOLATION
                ((TextField) comp).setEditable(false);  // VIOLATION
        }
}
```

Should be written as:

```
package com.rule;

import java.awt.Component;
import java.awt.TextField;

class Avoid_repeated_casting_correction
{
        public void method(Component comp)
        {
                final TextField tf = (TextField) comp;  // CORRECTION
                tf.setText("");
                tf.setEditable(false);
        }
}
```

Reference: http://www.javaperformancetuning.com/tips/rawtips.shtml
Java Performance tunning by Jack Shirazi

Rule 24: Use_ternary_operator

Severity: Medium
Rule: Use ternary operator for improved performance
Reason: Use ternary operator for improved performance

Usage Example:

```
package com.rule;
class Use_ternary_operator_correction
{
        public boolean test(String value)
        {
                if(value.equals("AppPerfect"))          // VIOLATION
                {
                        return true;
                }
                else
                {
                        return false;
                }
        }
}
```

Should be written as:

```
package com.rule;
class Use_ternary_operator_correction
{
        public boolean test(String value)
        {
                return value.equals("AppPerfect"); // CORRECTION
        }
}
```

Reference: Reference Not Available.

Rule 25: Remove_unnecessary_if_then_else_statement

Severity: Medium
Rule: It could be simplified to enhance the code's efficiency and reduce its size.
Reason: It could be simplified to enhance the code's efficiency and reduce its size.

Usage Example:

```
public class Test
{
 boolean b;
 public boolean isTrue()
{
  if (b) //VIOLATION
          {
return true;
          }
  else
          {
return false;
          }
 }
}
```

Should be written as:

```
public class Test
{
 boolean b;
 public boolean isTrue()
{
  return b; //FIXED
 }

}
```

Reference: Not available.

Rule 26: Always_dispose_SWT_Control

Severity: High
Rule: Always dispose SWT Control.
Reason: Always dispose SWT Control.

Usage Example:

```
package com.rule;

import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;

public class Always_dispose_SWT_Control_violation
{
        public void createControl(Composite cmp, int style)
        {
                Control ct = new Button(cmp, style);  // Violation
                //...do something with ct

        }

}
```

Should be written as:

```
package com.rule;

import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;

public class Always_dispose_SWT_Control_correction
{
        public void createControl(Composite cmp, int style)
        {
                Control ct = new Button(cmp, style);
                //...
                ct.dispose();  // Correction
        }
}
```

Reference: http://www.eclipse.org/articles/swt-design-2/swt-design-2.html

Rule 27: Always_declare_constant_field_static

Severity: Medium
Rule: The constant fields that are declared final should be declared static.
Reason: The constant fields that are declared final should be declared static.

Usage Example:

```
package com.rule;

public class Always_declare_constant_field_static_violation
{
        final int MAX = 1000; // VIOLATION
        final String NAME = "Noname"; // VIOLATION
}
```

Should be written as:

```
package com.rule;

public class Always_declare_constant_field_static_correction
{
        static final int MAX = 1000; // CORRECTION
        static final String NAME = "Noname"; // VIOLATION
}
```

Reference: http://www.glenmccl.com/jperf/#Sample1

Rule 28: Use_buffered_IO

Severity: Critical
Rule: Use BufferedInputStream and BufferedOutputStream or equivalent buffered methods wherever possible.
Doing I/O a single byte at a time is generally too slow.
Note that I/O operation uses lots of synchronization, hence you can get better performance by reading / writing in bulk.
Reason: Use BufferedInputStream and BufferedOutputStream or equivalent buffered methods wherever possible.
Doing I/O a single byte at a time is generally too slow.
Note that I/O operation uses lots of synchronization, hence you can get better performance by reading / writing in bulk.

Usage Example:

```
package com.rule;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

class Use_buffered_IO_violation
{
        public static void copy(String from, String to) throws IOException
        {
                final int NEGATIVE = -1;
        InputStream in = null;
        OutputStream out = null;
        try
        {
                in = new FileInputStream(from);         // VIOLATION
        out = new FileOutputStream(to);         // VIOLATION
        while (true)
        {
        int data = in.read();
        if (data == NEGATIVE)
        {
                break;
        }
        out.write(data);
        }
        in.close();
        out.close();
        }
        finally
        {
                if (in != null)
        {
        in.close();
        }
        if (out != null)
        {
        out.close();
        }
                }
        }
}
```

Should be written as:

```
package com.rule;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
class Use_buffered_IO_correction
{
	public static void copy(String from, String to) throws IOException
	{
		final int NEGATIVE = -1;
	InputStream in = null;
	OutputStream out = null;
	try
	{
		in = new BufferedInputStream(new FileInputStream(from));			// CORRECTION
	out = new BufferedOutputStream(new FileOutputStream(to));		// CORRECTION
	while (true)
	{
	int data = in.read();
	if (data == NEGATIVE)
	{
	break;
	}
	out.write(data);
	}
		}
	 finally
	 {
		if (in != null)
	  {
		in.close();
	  }
	  if (out != null)
	  {
		out.close();
	  }
		}
	}
}
```

Reference: java.sun.com\docs\books\performance\1st_edition\html\JPIOPerformance.fm.html

Rule 29: Avoid_unnecessary_casting

Severity: High
Rule: Avoid unnecessary casting.
Reason: Avoid unnecessary casting.

Usage Example:

```
package com.rule;

class Avoid_unnecessary_casting_violation
{
	public Object method()
	{
		String str = "AppPerfect";	//$NON-NLS-1$
		Object obj = (Object)str;		// VIOLATION
		return obj;
	}
}
```

Should be written as:

```
package com.rule;
class Avoid_unnecessary_casting_correction
{
	public Object method()
	{
		String str = "AppPerfect";	//$NON-NLS-1$
		Object obj = str;		// CORRECTION
		return obj;
	}
}
```

Reference: http://www.javaworld.com/javaworld/jw-12-1999/jw-12-performance.html

Rule 30: Avoid_instantiation_of_class_with_only_static_members

Severity: Medium
Rule: Avoid instantiation of class with only static members.
Reason: Avoid instantiation of class with only static members.

Usage Example:

```
package com.rule;

public class Avoid_instantiation_of_static_class_violation
{
	MyClassInner mcInner = new MyClassInner();  // Violation
}

class MyClasss
{
	public static void foo()
	{
	}
}
```

Should be written as:

```
Avoid instantiation of static classes having only static members.
```

Reference: Reference not available.

Rule 31: Close_jdbc_connections

Severity: High
Rule: Always close the database connections opened.
Reason: Always close the database connections opened.

Usage Example:

```
package com.rule;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Close_Jdbc_Connections_violation
{
	public void method (String url) throws SQLException
	{
		try
		{
			Connection conn = DriverManager.getConnection(url);		// VIOLATION
			// some operations on connection

		}
		catch (java.lang.Exception e)
		{
			e.printStackTrace();
		}
	}
}
```

Should be written as:

```
package com.rule;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Close_Jdbc_Connections_correction
{
	public void method (String url) throws SQLException
	{
		Connection conn = null;
		try
		{
			conn = DriverManager.getConnection(url);
			// some operations on connection
		}
		catch (java.lang.Exception e)
		{
			e.printStackTrace();
		}
		finally
		{
			conn.close();		// CORRECTION
		}
	}
}
```

Reference: Reference not available.

Rule 32: Close_jdbc_connections_Only_In_finally_Block

Severity: High
Rule: Always close the database connections opened. The one of the places to close the connection is in finally block.
Reason: Always close the database connections opened. The one of the places to close the connection is in finally block.

Usage Example:

```
package com.rule;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Close_Jdbc_Connections_only_in_finllay_block_violation
{
	public void method (String url) throws SQLException
	{
		try
		{
			Connection conn = DriverManager.getConnection(url);
			// some operations on connection

			conn.close ();		// VIOLATION
		}
		catch (java.lang.Exception e)
		{
			e.printStackTrace();
		}
	}
}
```

Should be written as:

```
package com.rule;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Close_Jdbc_Connections_only_in_finllay_block_correction
{
	public void method (String url) throws SQLException
	{
		Connection conn = null;
		try
		{
			conn = DriverManager.getConnection(url);
			// some operations on connection
		}
		catch (java.lang.Exception e)
		{
			e.printStackTrace();
		}
		finally
		{
			conn.close();		// CORRECTION
		}
	}
}
```

Reference: Reference not available.

Rule 33: Avoid_synchronized_readObject_method

Severity: Medium
Rule: Avoid synchronized readObject() method.
Reason: Avoid synchronized readObject() method.

Usage Example:

```
package com.rule;

import java.io.Serializable

class Avoid_synchronized_readObject_method implements Serializable
{
	private synchronized void readObject(ObjectInputStream s) throws IOException,ClassNotFoundException		//Violation
	{
	 s.defaultReadObject();

	 // customized deserialization code
	}

}
```

Should be written as:

```
package com.rule;

import java.io.Serializable

class Avoid_synchronized_readObject_method implements Serializable
{
	private void readObject(java.io.ObjectInputStream in)throws IOException, ClassNotFoundException		//Correction
	{
	 s.defaultReadObject();

	 // customized deserialization code
	}

}
```

Reference: Reference Not Available

Rule 34: Avoid_boolean_array

Severity: Low
Rule: Do not use array of boolean.
Reason: Do not use array of boolean.

Usage Example:

```
package com.rule;

public class Avoid_boolean_array_violation
{
	public void method()
	{
		boolean[] b = new boolean[]{true, false, true}; // VIOLATION
	}
}
```

Should be written as:

```
package com.rule;

public class Avoid_boolean_array_correction
{
        public void method()
        {
                BitSet bs = new BitSet(3); // CORRECTION
                bs.set(0);
                bs.set(2);
        }
}
```

Reference: http://www.javaperformancetuning.com/tips/rawtips.shtml

Rule 35: Avoid_string_concatenation_in_loop

Severity: Critical
Rule: Use 'StringBuffer' instead of 'String' for non-constant strings.
Reason: Use 'StringBuffer' instead of 'String' for non-constant strings.

Usage Example:

```
package com.rule;
class String_concatenation_violation
{
        public void concatValues()
        {
                String result = "";
                for (int i = 0; i < 20; i++)
                {
                 result += getNextString();                             // VIOLATION
                }
        }
}
```

Should be written as:

```
package com.rule;
class String_concatenation_correction
{
        public void concatValues(String strMainString, String strAppend1, String strAppend2)
        {
                String result = "";
                StringBuffer buffer = new StringBuffer();
                for (int i = 0; i < 20; i++)
                {
                 buffer.append(getNextString());                        // CORRECTION
                }
                result = buffer.toString();                             // CORRECTION
        }
}
```

Reference: http://java.sun.com/docs/books/performance/1st_edition/html/JPMutability.fm.html

Rule 36: Avoid_method_calls_in_loop

Severity: High
Rule: If possible avoid using length(), size() etc. method calls in loop condition statement, there can be a performance hit.
Reason: If possible avoid using length(), size() etc. method calls in loop condition statement, there can be a performance hit.

Usage Example:

```
package com.rule;

class Avoid_method_calls_in_loop_violation
{
        public void method()
        {
                String str = "Hello";
                for (int i = 0; i < str.length(); i++)          // VIOLATION
                {
                        i++;
                }
        }
}
```

Should be written as:

```
package com.rule;

class Avoid_method_calls_in_loop_correction
{
        public void method()
        {
                String str = "Hello";
                int len = str.length();         // CORRECTION
                for (int i = 0; i < len ; i++)
                {
                        i++;
                }
 }
}
```

Reference: Reference Not Available.

Rule 37: Avoid_new_with_string

Severity: Medium
Rule: Avoid using new with String objects.
Reason: Avoid using new with String objects.

Usage Example:

```
package com.rule;

class Avoid_new_with_string_violation
{
	public int action(String str)
	{
		String s = new String(str);		// VIOLATION
		return s.length();
	}
}
```

Should be written as:

```
package com.rule;

class Avoid_new_with_string_correction
{
	public int action(String str)
	{
		String s = str;		// CORRECTION
		return s.length();
	}
}
```

Reference: http://www.javaworld.com/javaqa/2002-09/01-qa-0906-strings.html

Rule 38: Loop_invariant_code_motion

Severity: Medium
Rule: The code that is going to result in same value over the iterations of loop, should be moved out of the loop.
Reason: The code that is going to result in same value over the iterations of loop, should be moved out of the loop.

Usage Example:

```
package com.rule;
class Loop_invariant_code_motion_violation
{
	public void method(int x, int y, int[] z)
	{
		for(int i = 0; i < z.length; i++)
		{
			z[i] = x * Math.abs(y);		// VIOLATION
		}
	}
}
```

Should be written as:

```
package com.rule;
class Loop_invariant_code_motion_correction
{
	public void method(int x, int y, int[] z)
	{
		int t1 = x * Math.abs(y);		// CORRECTION
		for(int i = 0; i < z.length; i++)
		{
			z[i] = t1;
		}
	}
}
```

Reference: Java Performance Tunning by Jack Shirazi

Rule 39: Use_short_circuit_boolean_operators

Severity: High
Rule: Short-circuit booleans should be used as that speeds up the test slightly in almost every case.
Reason: Short-circuit booleans should be used as that speeds up the test slightly in almost every case.

Usage Example:

```
package com.rule;
class Use_short_circuit_boolean_operators_violation
{
	public void method()
	{
		if(sValue.equals("true")  | sValue.equals("false"))	// VIOLATION
		{
			System.out.println("valid boolean");
		}
	}
}
```

Should be written as:

```
package com.rule;
class Use_short_circuit_boolean_operators_correction
{
	public void method()
	{
		if(sValue.equals("true") || sValue.equals("false"))	// CORRECTION
		{
			System.out.println("valid boolean");
		}
	}
}
```

Reference: Java Performance Tunning by Jack Shirazi
http://java.oreilly.com/news/javaperf_0900.html

Rule 40: Avoid_using_StringTokenizer

Severity: Critical
Rule: Avoid using StringTokenizer to improve performace
Reason: Avoid using StringTokenizer to improve performace

Usage Example:

```
package com.rule;
class Avoid_using_StringTokenizer_violation
{
        public void method(String str)
        {
                StringTokenizer strtok = new StringTokenizer(str);      // VIOLATION
                while(strtok.hasMoreTokens())
                {
                        System.out.println(strtok.nextToken());
                }
        }
}
```

Should be written as:

```
package com.rule;
class Avoid_using_StringTokenizer_correction
{
        public void method(String str)
        {
                String[] parts = breakUp(str);          // CORRECTION
                int len = parts.length;
                for(int i=len; i>0; i--)
                {
                        System.out.println(parts[len-i]);
                }
        }

        String[] breakUp(String str)
        {
                String strParts[];
                // break the string into parts
                return strParts;
        }
}
```

Reference: http://www.javaperformancetuning.com/tips/rawtips.shtml

Rule 41: Use_instanceof_only_on_interfaces

Severity: Medium
Rule: Use instanceof only on interfaces.
Reason: Use instanceof only on interfaces.

Usage Example:

```
package com.rule;

class MyClass { }

public class Use_instanceof_only_on_interfaces
{
 private void method (Object o)
 {
  if (o instanceof MyClass) { }// VIOLATION
 }
}
```

Should be written as:

```
package com.rule;

interface MyInterface {}
class MyClass implements MyInterface {}

public class Use_instanceof_only_on_interfaces
{
 private void method (Object o)
 {
  if (o instanceof MyInterface) { }// Correction
 }
}
```

Reference: Reference not available.

Rule 42: Avoid_null_check_before_instanceof

Severity: Medium
Rule: Avoid null check before checking instanceof.
Reason: Avoid null check before checking instanceof.

Usage Example:

```
package com.rule;

public class Avoid_null_check_before_instanceof_violation
```

```
{
        public void method(Object o)
        {
                if(o != null &&  o instanceof Object)           // Violation.
                {
                        // Do Something.
                }
        }

}
```

Should be written as:

```
package com.rule;

public class Avoid_null_check_before_instanceof_correction
{
        public void method(Object o)
        {
                if(o instanceof Object)         // Correction
                {
                        // Do Something.
                }
        }

}
```

Reference: Reference not available.

Rule 43: Stream_not_closed

Severity: High
Rule: Always close the streams opened.
Reason: Always close the streams opened.

Usage Example:

```
package com.rule;

import java.io.FileInputStream;
import java.io.IOException;

public class Stream_not_closed
{
        public void method (java.io.File f) throws IOException
        {
                FileInputStream fileInputStream = null;
                try
                {
                        fileInputStream = new java.io.FileInputStream(f);               //Violation
                        fileInputStream.read ();
                }
                catch (java.io.FileNotFoundException e1)
                {
                        System.out.println("Exception : File not found");
                }
        }
}
```

Should be written as:

```
package com.rule;

import java.io.FileInputStream;
import java.io.IOException;

public class Stream_not_closed
{
        public void method (java.io.File f) throws IOException
        {
                FileInputStream fileInputStream = null;
                try
                {
                        fileInputStream = new java.io.FileInputStream(f);
                        fileInputStream.read ();
                        fileInputStream.close ();               // Correction
                }
                catch (java.io.FileNotFoundException e1)
                {
                        System.out.println("Exception : File not found");
                }
        }
}
```

Reference: Reference not available.

Rule 44: Close_streams_only_in_finally

Severity: High
Rule: Always close the streams opened, in finally block.
Reason: Always close the streams opened, in finally block.

Usage Example:

```
package com.rule;
import java.io.FileInputStream;
import java.io.IOException;
```

```
public class Close_streams_only_in_finally
{
	public void method (java.io.File f) throws IOException
	{
		FileInputStream fileInputStream = null;
		try
		{
			fileInputStream = new java.io.FileInputStream(f);
			fileInputStream.read ();
			fileInputStream.close ();		// VIOLATION
		}
		catch (java.io.FileNotFoundException e1)
		{
			System.out.println("Exception : File not found");
		}
		catch (java.lang.Exception e)
		{
			e.printStackTrace();
		}
	}
}
```

Should be written as:

```
package com.rule;
import java.io.FileInputStream;
import java.io.IOException;

public class Close_streams_only_in_finally
{
	public void method (java.io.File f) throws IOException
	{
		FileInputStream fileInputStream = null;
		try
		{
			fileInputStream = new java.io.FileInputStream(f);
			fileInputStream.read ();

		}
		catch (java.io.FileNotFoundException e1)
		{
			System.out.println("Exception : File not found");
		}
		catch (java.lang.Exception e)
		{
			e.printStackTrace();
		}
		finally
		{
			fileInputStream.close ();		// CORRECTION
		}
	}
}
```

Reference: Reference not available.

Rule 45: Avoid_instantiation_for_getClass

Severity: High
Rule: Do not create instances just to call getClass on it.
Reason: Do not create instances just to call getClass on it.

Usage Example:

```
package com.rule;

public class Avoid_instantiation_for_getClass_violation
{
	public void method()
	{
		Class c = (new Avoid_instantiation_for_getClass_violation()).getClass(); // VIOLATION
	}
}
```

Should be written as:

```
package com.rule;

public class Avoid_instantiation_for_getClass_correction
{
	public void method()
	{
		Class c = Avoid_instantiation_for_getClass_correction.class; // CORRECTION
	}
}
```

Reference: http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Class.html
http://java.sun.com/docs/books/jls/second_edition/html/expressions.doc.html#251530

Rule 46: Use_System_arrayCopy

Severity: Critical
Rule: Use 'System.arraycopy ()' instead of a loop to copy array. This rule disallows the copying of arrays inside a loop
Reason: Use 'System.arraycopy ()' instead of a loop to copy array. This rule disallows the copying of arrays inside a loop

Usage Example:

```
package com.rule;
class Use_System_arrayCopy_violation
{
        public int[] copyArray (int[] array)
        {
                int length = array.length;
                int[] copy = new int [length];
                for(int i=0;i<length;i++)
                {
                        copy [i] = array[i];            // VIOLATION
                }
                return copy;
        }
}
```

Should be written as:

```
package com.rule;
class Use_System_arrayCopy_correction
{
        public int[] copyArray (int[] array)
        {
                final int ZERO = 0;
                int length = array.length;
                int[] copy = new int [length];
                System.arraycopy(array, ZERO, copy, ZERO, length);              // CORRECTION
                return copy;
        }
}
```

Reference: http://www.cs.cmu.edu/~jch/java/speed.html

Rule 47: Use_String_length_to_compare_empty_string

Severity: High
Rule: The String.equals() method is overkill to test for an empty string. It is quicker to test if the length of the string is 0.
Reason: The String.equals() method is overkill to test for an empty string. It is quicker to test if the length of the string is 0.

Usage Example:

```
package com.rule;
class Use_String_length_to_compare_empty_string_violation
{
        public boolean isDocEmpty()
        {
                return doc.getContents().equals("");            // VIOLATION
        }
}
```

Should be written as:

```
package com.rule;
class Use_String_length_to_compare_empty_string_correction
{
        public boolean isDocEmpty()
        {
                return doc.getContents().length() == 0;         // CORRECTION
        }
}
```

Reference: http://www.onjava.com/pub/a/onjava/2002/03/20/optimization.html?page=4
http://www.javaperformancetuning.com/tips/rawtips.shtml

Rule 48: Use_String_equalsIgnoreCase

Severity: High
Rule: Use String.equalsIgnoreCase() method.
Reason: Use String.equalsIgnoreCase() method.

Usage Example:

```
package com.rule;

public class Use_String_equalsIgnoreCase_violation
{
        public void method()
        {
                String str = "APPPERFECT";
                String str1 = "appperfect";

                if(str1.toUpperCase().equals(str))              // Violation
                {
                        System.out.println("Strings are equals");
                }
        }

}
```

Should be written as:

```
package com.rule;

public class Use_String_equalsIgnoreCase_correction
{
        public void method()
        {
                String str = "APPPERFECT";
                String str1 = "appperfect";
                if(str1.equalsIgnoreCase(str))          // Correction.
                {
                        System.out.println("Strings are equals");
                }
        }

}
```

Reference: Reference not available.

Rule 49: Place_try_catch_out_of_loop

Severity: Medium
Rule: Placing "try/catch/finally" blocks inside loops can slow down the execution of code.
Reason: Placing "try/catch/finally" blocks inside loops can slow down the execution of code.

Usage Example:

```
package com.rule;

import java.io.InputStream;
import java.io.IOException;

class Place_try_catch_out_of_loop_violation
{
        void method (InputStream is)
         {
                 int ZERO = 0;
                 int TEN = 10;
                 int count = 0;

                 for (int i = ZERO; i < TEN; i++)
                 {
                         try                     // VIOLATION
                         {
                                 count += is.read();
                         }
                         catch (IOException ioe)
                         {
                                ioe.printStackTrace();
                         }
                 }
 }
}
```

Should be written as:

```
package com.rule;

import java.io.InputStream;
import java.io.IOException;

class Place_try_catch_out_of_loop_correction
{
  void method (InputStream is)
         {
                 int ZERO = 0;
                 int TEN = 10;
                 int count = 0;

                 try                     // CORRECTION
                 {
                         for (int i = ZERO; i < TEN; i++)
                         {
                                         count += is.read ();
                         }
                 }
                 catch (IOException ioe)
                 {
                        ioe.printStackTrace();
                 }
 }
}
```

Reference: http://www.precisejava.com/javaperf/j2se/Loops.htm

Rule 50: Declare_inner_class_static

Severity: Medium
Rule: The inner class that doesn't require the outer class reference, should be declared static.
Reason: The inner class that doesn't require the outer class reference, should be declared static.

Usage Example:

```
package com.rule;

public class Declare_inner_class_static_violation
{
        class Declare_inner_class_static_violation_INNER // VIOLATION
        {
                // no reference for Declare_inner_class_static_violation.this or its fields, methods.
        }
}
```

Should be written as:

```
package com.rule;

public class Declare_inner_class_static_correction
{
	static class Declare_inner_class_static_correction_INNER // CORRECTION
	{
		// no reference for Declare_inner_class_static_correction.this or its fields, methods.
	}
}
```

Reference: http://java.sun.com/docs/books/tutorial/java/javaOO/nested.html
http://www.builderau.com.au/program/java/0,39024620,39130230,00.htm

Rule 51: Declare_inner_class_using_outer_class_only_in_constructor_static

Severity: Medium
Rule: The inner class which uses outer class in constructor, should be declared static.
Reason: The inner class which uses outer class in constructor, should be declared static.

Usage Example:

```
package com.rule;

public class Declare_inner_class_using_outer_class_only_in_constructor_static_violation
{
	int x = 0;
	class Declare_inner_class_using_outer_class_only_in_constructor_static_violation_INNER // VIOLATION
	{
		int x = 0;
		public Declare_inner_class_using_outer_class_only_in_constructor_static_violation_INNER()
		{
			x = Declare_inner_class_using_outer_class_only_in_constructor_static_violation.this.x;
		}
	}
}
```

Should be written as:

```
package com.rule;

public class Declare_inner_class_using_outer_class_only_in_constructor_static_correction
{
	int x = 0;
	static class Declare_inner_class_using_outer_class_only_in_constructor_static_correction_INNER // CORRECTION
	{
		public Declare_inner_class_using_outer_class_only_in_constructor_static_correction_INNER(Declare_inner_class_using_outer_class_only_in_con
		{
			this.x = outer.x;
		}
	}
}
```

Reference: Reference not available.

Rule 52: Avoid_nested_Synchronized_blocks

Severity: Critical
Rule: Avoid nested Synchronized blocks to improve performance
Reason: Avoid nested Synchronized blocks to improve performance

Usage Example:

```
package com.rule;
class Avoid_nested_synchronized_blocks_violation
{
	public void doTest()
	{
		//......
		synchronized (getClass())
		{
			//.....
			synchronized (this)	// VIOLATION
			{
				//.....
			}
			//.....
		}
		//......
	}
}
```

Should be written as:

```
Avoid nested synchronized blocks.
```

Reference: http://www.darrenhobbs.com/archives/000448.html

Rule 53: Avoid_empty_try_blocks

Severity: Low
Rule: Avoid empty "try" block structure.
Reason: Avoid empty "try" block structure.

Usage Example:

```
package com.rule;
```

```
class Avoid_empty_try_blocks_violation
{
	void method (int i, int j)
	{
		try			// VIOLATION
		{
		}
		catch(ArithmeticException ae)
		{
			i = j;
			ae.printStackTrace();
		}
	}

}
```

Should be written as:

```
package com.rule;

class Avoid_empty_try_blocks_correction
{
	void method (int i, int j)
	{
		/*
		try			// CORRECTION
		{
		}
		catch(ArithmeticException ae)
		{
			i = j;
			ae.printStackTrace();
		}
		*/
	}
}
```

Reference: Reference Not Available.

Rule 54: Declare_accessor_methods_final

Severity: High
Rule: Declare accessor methods for instance fields as "final".
Reason: Declare accessor methods for instance fields as "final".

Usage Example:

```
package com.rule;

public class Decalre_accessor_methods_final_violation
{
	private String name;
	public String getName()		// VIOLATION
	{
		return name;
	}
}
```

Should be written as:

```
package com.rule;

public class Decalre_accessor_methods_final_correction
{
	private String name;
	public final String getName()		// CORRECTION
	{
		return name;
	}
}
```

Reference: http://ccm.redhat.com/doc/core-platform/5.0/engineering-standards/s1-performance-optimization.html

Rule 55: Use_batch_update_for_sql_queries

Severity: Critical
Rule: Instead of calling executeUpdate, use executeBatch.
Reason: Instead of calling executeUpdate, use executeBatch.

Usage Example:

```
package com.rule;

import java.sql.Connection;
import java.sql.Statement;

public class Use_batch_update_for_sql_queries_violation
{
	public void method(Connection conn) throws Exception
	{
		String[] queries = new String[] {"query1", "query2", "query3"};
		Statement stmt = conn.createStatement();
		for (int i = 0; i < queries.length; i++)
		{
			stmt.executeUpdate(queries[i]); // VIOLATION
		}
	}
}
```

Should be written as:

```
package com.rule;

import java.sql.Connection;
import java.sql.Statement;

public class Use_batch_update_for_sql_queries_correction
{
        public void method(Connection conn) throws Exception
        {
                String[] queries = new String[] {"query1", "query2", "query3"};
                Statement stmt = conn.createStatement();
                for (int i = 0; i < queries.length; i++)
                {
                        stmt.addBatch(queries[i]); // CORRECTION
                }
                stmt.executeBatch();
        }
}
```

Reference: http://java.sun.com/docs/books/tutorial/jdbc/jdbc2dot0/batchupdates.html

Rule 56: Declare_private_constant_fields_final

Severity: Medium
Rule: A private constant fields should be declared final for performance reasons
Reason: A private constant fields should be declared final for performance reasons

Usage Example:

```
package com.rule;

class Declare_private_constant_fields_final_violation
{
        private int i = 5;  // VIOLATION
        public void method()
        {
                int j = i;
                j = j + i;
        }
}
```

Should be written as:

```
package com.rule;

class Declare_private_constant_fields_final_correction
{
        private final int i = 5;  // CORRECTION
        public void method()
        {
                int j = i;
                j = j + i;
        }
}
```

Reference: Reference Not Available.

Rule 57: Always_declare_constant_local_variables_final

Severity: Medium
Rule: A constant local variable should be declared final for performance reasons
Reason: A constant local variable should be declared final for performance reasons

Usage Example:

```
public class Test
{
 private int foo (int x)
{
                int size = 5;  // VIOLATION
        return size + x;
 }
}
```

Should be written as:

```
public class Test
{
 private int foo (int x)
{
                final int size = 5;  // FIXED
        return size + x;
 }
}
```

Reference: Not available.

Rule 58: Reduce_number_of_exception_creations

Severity: Low
Rule: Reduce number of exception creations for performace reasons
Reason: Reduce number of exception creations for performace reasons

Usage Example:

```
class Reduce_number_of_exception_creations_violation
{

        public static final int TEN = 10;


        public void method() throws Exception
        {
                if(true)
                {
                        throw new Exception("1");
                }
                if(true)
                {
                        throw new Exception("2");
                }
                if(true)
                {
                        throw new Exception("3");
                }
                if(true)
                {
                        throw new Exception("4");
                }
                if(true)
                {
                        throw new Exception("5");
                }
                if(true)
                {
                        throw new Exception("6");
                }
                if(true)
                {
                        throw new Exception("7");
                }
                if(true)
                {
                        throw new Exception("8");
                }
                if(true)
                {
                        throw new Exception("9");
                }
                if(true)
                {
                        throw new Exception("10");
                }
                if(true)
                {
                        throw new Exception("11");// VIOLATION, it is the 11th exception.
                }
        }

}
```

Should be written as:

```
class Reduce_number_of_exception_creations_correction
{

        public static final int TEN = 10;


        public void method() throws Exception
        {
                if(true)
                {
                        throw new Exception("1");
                }
                if(true)
                {
                        throw new Exception("2");
                }
                if(true)
                {
                        throw new Exception("3");
                }
                if(true)
                {
                        throw new Exception("4");
                }
                if(true)
                {
                        throw new Exception("5");
                }
                if(true)
                {
                        throw new Exception("6");
                }
                if(true)
                {
                        throw new Exception("7");
                }
                if(true)
                {
                        throw new Exception("8");
                }
                if(true)
                {
                        throw new Exception("9");
                }
                if(true)
                {
                        throw new Exception("10");
                }
                // CORRECTION.
        }

}
```

Reference: http://ccm.redhat.com/doc/core-platform/5.0/engineering-standards/performance-optimization.html

Rule 59: Reduce_switch_density

Severity: Medium
Rule: Reduce switch density for performance reasons.
Reason: Reduce switch density for performance reasons.

Usage Example:

```
package com.rule;

public class Reduce_switch_density_violation
{
        public void method()
        {
                switch (x)      // Violation.
                {
                         case 1:
                         {
                                 if(status)
                                 {
                                        // More Statements
                                 }
                                 break;
                         }
                         case 2:
                         {
                                // More Statements
                                break;
                         }
                         default :
                         {

                         }
                }
        }

}
```

Should be written as:

```
package com.rule;

public class Reduce_switch_density_correction
{
        public void method()
        {
                switch (x)                      //Correction.
                {
                         case 1:
                         {
                                 if(status)
                                 {
                                        method1();
                                 }
                                 break;
                         }
                         case 2:
                         {
                                method2();

                                break;
                         }
                         default :
                         {

                         }
                }
                i--;
        }
        public method1()
        {
                // Do Something.
        }
        public method2()
        {
                // Do Something.
        }


}
```

Reference: Reference not available.

Rule 60: Avoid_empty_loops

Severity: Low
Rule: Avoid empty loops.
Reason: Avoid empty loops.

Usage Example:

```
package com.rule;

class Avoid_empty_loops_violation
{
	public void method()
	{
		int i = -5;
		final int ZERO = 0;
		final int NEGATIVE = -1;

		while (i < ZERO)		// VIOLATION
		{
		}

		i = NEGATIVE;

		for(;i < ZERO;)		// VIOLATION
		{
		}
	}
}
```

Should be written as:

```
package com.rule;

class Avoid_empty_loops_correction
{

	public void method()
	{
		int i = -5;
		final int ZERO = 0;
		final int NEGATIVE = -1;

		/*
		while (i < ZERO)		// CORRECTION
		{
		}
		*/

		i = NEGATIVE;

		/*
		for(;i < ZERO;)		// CORRECTION
		{
		}
		*/
	}
}
```

Reference: Reference Not Available.

Rule 61: Avoid_new_Integer_toString

Severity: High
Rule: Avoid creating objects of primitive types to call the toString() method instead use valueOf(...) in String class to convert primitive types into their String equivalent.
Reason: Avoid creating objects of primitive types to call the toString() method instead use valueOf(...) in String class to convert primitive types into their String equivalent.

Usage Example:

```
package com.rule;

class Avoid_new_Integer_toString_violation
{
	public void print()
	{
		String str = new Integer(1).toString();		// VIOLATION
	}
}
```

Should be written as:

```
package com.rule;

class Avoid_new_Integer_toString_correction
{
	public void print()
	{
		String str = String.valueOf(1);		// CORRECTION
	}
}
```

Reference: Reference Not Available.

Rule 62: Avoid_passing_primitive_int_to_Integer_constructor

Severity: High
Rule: Avoid creating objects of primitive types using the constructor.
Reason: Avoid creating objects of primitive types using the constructor.

Usage Example:

```
public class Test
{
	public void fubar()
	{
		Integer i = new Integer(3); // VIOLATION
		//...
	}
}
```

Should be written as:

```
public class Test
{
	public void fubar()
	{
		Integer i = Integer.valueOf(3); // FIXED
		//...
	}
}
```

Reference: Not Available.

Rule 63: Avoid_passing_primitive_long_to_Long_constructor

Severity: High
Rule: Avoid creating objects of primitive types using the constructor.
Reason: Avoid creating objects of primitive types using the constructor.

Usage Example:

```
public class Test
{
	public void fubar()
	{
		Long i = new Long(3); // VIOLATION
		//...
	}
}
```

Should be written as:

```
public class Test
{
	public void fubar()
	{
		Long i = Long.valueOf(3); // FIXED
		//...
	}
}
```

Reference: Not Available.

Rule 64: Avoid_passing_primitive_double_to_Double_constructor

Severity: High
Rule: Avoid creating objects of primitive types using the constructor.
Reason: Avoid creating objects of primitive types using the constructor.

Usage Example:

```
public class Test
{
	public void fubar()
	{
		Double i = new Double(3.0); // VIOLATION
		//...
	}
}
```

Should be written as:

```
public class Test
{
	public void fubar()
	{
		Double i = Double.valueOf(3.0); // FIXED
		//...
	}
}
```

Reference: Not Available.

Rule 65: Avoid_passing_primitive_short_to_Short_constructor

Severity: High
Rule: Avoid creating objects of primitive types using the constructor.
Reason: Avoid creating objects of primitive types using the constructor.

Usage Example:

```
public class Test
{
	public void fubar()
	{
		Short i = new Short(3); // VIOLATION
		//...
	}
}
```

Should be written as:

```
public class Test
{
	public void fubar()
	{
		Short i = Short.valueOf(3); // FIXED
		//...
	}
}
```

Reference: Not Available.

Rule 66: Avoid_passing_primitive_byte_to_Byte_constructor

Severity: High
Rule: Avoid creating objects of primitive types using the constructor.
Reason: Avoid creating objects of primitive types using the constructor.

Usage Example:

```
public class Test
{
	public void fubar()
	{
		Byte i = new Byte(3); // VIOLATION
		//...
	}
}
```

Should be written as:

```
public class Test
{
	public void fubar()
	{
		Byte i = Byte.valueOf(3); // FIXED
		//...
	}
}
```

Reference: Not Available.

Rule 67: Avoid_passing_primitive_char_to_Character_constructor

Severity: High
Rule: Avoid creating objects of primitive types using the constructor.
Reason: Avoid creating objects of primitive types using the constructor.

Usage Example:

```
public class Test
{
	public void fubar()
	{
		Character i = new Character('a'); // VIOLATION
		//...
	}
}
```

Should be written as:

```
public class Test
{
	public void fubar()
	{
		Character i = Character.valueOf('a'); // FIXED
		//...
	}
}
```

Reference: Not Available.

Rule 68: Avoid_using_String_toString

Severity: High
Rule: Avoid calling toString() on Strings.Instead use String.
Reason: Avoid calling toString() on Strings.Instead use String.

Usage Example:

```
package com.rule;

class Avoid_using_String_toString
{
	public void print()
	{
		String str="AppPerfect";

		System.out.println(str.toString());		// Violation.
```

```
        }
}
```

Should be written as:

```
package com.rule;

class Avoid_using_String_toString
{
        public void print()
        {
                String str="AppPerfect";

                System.out.println(str);                // Correction
        }
}
```

Reference: Reference Not Available.

Rule 69: Avoid_unnecessary_substring

Severity: High
Rule: Avoid using String.substring(0).
Reason: Avoid using String.substring(0).

Usage Example:

```
package com.rule;

public class Avoid_unnecessary_substring_violation
{
        public void method()
        {
                String str="AppPerfect";

                String str1 = str.substring(0);         // Violation.
        }

}
```

Should be written as:

```
package com.rule;

public class Avoid_unnecessary_substring_correction
{
        public void method()
        {
                String str="AppPerfect";

                String str1 = str;              //Correction.
        }

}
```

Reference: Reference not available.

Rule 70: Use_toArray_with_array_as_parameter

Severity: High
Rule: Use toArray(Object[]) instead of toArray() on collection.
Reason: Use toArray(Object[]) instead of toArray() on collection.

Usage Example:

```
package com.rule;

import java.util.Collection;
import java.util.List;
import java.util.ArrayList;

class Use_toArray_with_array_as_parameter
{
        public void print()
        {
                Collection c = new ArrayList();

                c.add("AppPerfect");
                c.add("TestStudio");

                Object[] obj  = c.toArray();            // Violation
        }
}
```

Should be written as:

```
package com.rule;

import java.util.Collection;
import java.util.List;
import java.util.ArrayList;

class Use_toArray_with_array_as_parameter
{
        public void print()
        {
                Collection c = new ArrayList();
                c.add("AppPerfect");
                c.add("TestStudio");

                String[] x = (String[]) c.toArray(new String[2]);               //Correction
        }
}
```

Reference: Reference Not Available.

Rule 71: Do_lazy_initialization

Severity: High
Rule: Do Lazy initialization.
Reason: Do Lazy initialization.

Usage Example:

```
package com.rule;

public class Do_lazy_initialization_violation
{
        private Do_lazy_initialization_violation  instance  =  new Do_lazy_initialization_violation();  //Violation

}
```

Should be written as:

```
package com.rule;

public class Do_lazy_initialization_correction
{
        private Do_lazy_initialization_violation instance;

        public Do_lazy_initialization_violation getInstance()
        {
                if(doLazy == null)
                        instance  =  new Do_lazy_initialization_violation();  // Correction
                return instance;
        }

}
```

Reference: http://www.javapractices.com/Topic34.cjp

Rule 72: Avoid_using_Thread_yield

Severity: Medium
Rule: Avoid using Thread.yield().
Reason: Avoid using Thread.yield().

Usage Example:

```
package com.rule;
public class Avoid_using_Thread_yield_violation
{
        public void method()
        {
                //......
                Thread.yield(); // VIOLATION
                //......
        }
}
```

Should be written as:

```
package com.rule;
public class Avoid_using_Thread_yield_correction
{
        public void method()
        {
                //......
                this.wait(); // CORRECTION
                //......
        }
}
```

Reference: http://forum.java.sun.com/thread.jspa?threadID=167194&messageID=2645189

Rule 73: Avoid_unread_fields

Severity: Medium
Rule: Avoid unread fields.
Reason: Avoid unread fields.

Usage Example:

```
package com.rule;

public class Avoid_unread_fields_violation
{
        private static String name = "AppPerfect";              //Violation.
        private static int i = 10;

        public static void method()
        {
                name = "AppPerfect USA";
                System.out.println(i);
        }

}
```

Should be written as:

Reference: Reference not available.

Rule 74: Avoid_equality_with_boolean

Severity: Medium
Rule: Avoid comparing a boolean with "true".
Reason: Avoid comparing a boolean with "true".

Usage Example:

```
package com.rule;

class Avoid_equality_with_boolean_violation
{
        boolean method(String value)
        {
                boolean b = false;
                String str = "S";

                if (value.endsWith(str) == true)                // VIOLATION
                {
                        b = true;
                }

                return b;
        }
}
```

Should be written as:

```
package com.rule;

class Avoid_equality_with_boolean_correction
{
        boolean method(String value)
        {
                boolean b = false;
                String str = "S";
                //....
                //........
                if ( value.endsWith(str) )              // CORRECTION
                {
                        b = true;
                }

                return b;
        }
}
```

Reference: Reference Not Available.

Rule 75: Avoid_startsWith

Severity: Critical
Rule: Avoid calling String.startsWith() for performance reasons
Reason: Avoid calling String.startsWith() for performance reasons

Usage Example:

```
package com.rule;
class Avoid_startsWith_violation
{
        public void method()
        {
                String sTemp="Data";
                if (sTemp.startsWith("D")) // VIOLATION
                {
                        sTemp = "data";
                }
        }
}
```

Should be written as:

```
package com.rule;

class Avoid_startsWith_correction
{
        public void method()
        {
                final int ZERO = 0;
                final char D = 'D';
                String sTemp="Data";

                if (sTemp.length () > ZERO && sTemp.charAt(ZERO) == D)  // CORRECTION
                {
                        sTemp = "data";
                }
        }
}
```

Reference: http://www.javacommerce.com/displaypage.jsp?name=java_performance.sql&id=18264

Rule 76: Avoid_readByte_method_in_loop

Severity: High
Rule: Avoid reading single byte in loop
Reason: Avoid reading single byte in loop

Usage Example:

```
package com.rule;

class Avoid_readByte_method_in_loop_violation
{
	public static void main(String args[])
	{
		String strFilePath = "c:\temp.java"; //$NON-NLS-1$
		final int ZERO = 0;
		final int TEN = 10;

		try
		{

			java.io.FileInputStream fis = new java.io.FileInputStream(strFilePath);
			java.io.DataInputStream ds = new java.io.DataInputStream(fis);

			int i = ZERO;
			while(i < TEN)
			{
				ds.readByte();		// VIOLATION
				i++;
			}

			for(i=ZERO; i<TEN; i++)
			{
				ds.readByte();		// VIOLATION
			}

			i = ZERO;

			do
			{
				ds.readByte();		// VIOLATION
				i++;
			}
			while(i<TEN);
		}
		catch(java.io.IOException e)
		{
			e.printStackTrace();
		}
	}
}
```

Should be written as:

```
package com.rule;

class Avoid_readByte_method_in_loop_correction
{
	public static void main(String args[])
	{

		final int TEN = 10;
		byte bArr[] = new byte[TEN];
		String strFilePath = "c:\temp.java"; //$NON-NLS-1$

		try
		{
			java.io.FileInputStream fis = new java.io.FileInputStream(strFilePath);
			java.io.DataInputStream ds = new java.io.DataInputStream(fis);

			ds.read(bArr);		// CORRECTION

		}
		catch(java.io.IOException e)
		{
			e.printStackTrace();
		}
	}
}
```

Reference: Reference Not Available.

Rule 77: Avoid_instantiation_of_boolean

Severity: High
Rule: Avoid instantiation of Boolean instead use the static constants defined in Boolean class.
Reason: Avoid instantiation of Boolean instead use the static constants defined in Boolean class.

Usage Example:

```
package com.rule;

class Avoid_instantiation_of_boolean_violation
{
	public Boolean method()
	{
		Boolean b = new Boolean(true);		// VIOLATION
```

```
                return b;
        }
}
```

Should be written as:

```
package com.rule;

class Avoid_instantiation_of_boolean_correction
{
        public Boolean method()
        {
                Boolean b = Boolean.TRUE;               // CORRECTION
                return b;
 }
}
```

Reference: Reference Not Available.

Rule 78: Avoid_Synchronized_blocks

Severity: Critical
Rule: Do not use synchronized blocks to avoid synchronization overheads
Reason: Do not use synchronized blocks to avoid synchronization overheads

Usage Example:

```
package com.rule;
class Avoid_synchronized_blocks_violation
{
        public void doTest()
        {
                synchronized (getClass())       // VIOLATION
                {
                        //.....
                }
        }
}
```

Should be written as:

```
package com.rule;

class Avoid_synchronized_blocks_correction
{
        public synchronized void doTest()       // CORRECTION
        {
                //.....
        }

}
```

Reference: http://www-2.cs.cmu.edu/~jch/java/speed.html

Rule 79: Declare_package_private_method_final

Severity: Medium
Rule: A package-private method that is not overridden should be declared final.
Reason: A package-private method that is not overridden should be declared final.

Usage Example:

```
package com.rule;

class Declare_package_private_method_final_violation
{
        void method()           // VIOLATION
        {
        }
}
```

Should be written as:

```
package com.rule;

class Declare_package_private_method_final_correction
{
        final void method()     // CORRECTION
        {
        }
}
```

Reference: http://www.javaworld.com/javaworld/jw-12-1998/jw-12-securityrules_p.html

Rule 80: Declare_public_or_protected_method_final

Severity: Medium
Rule: It optimizes the code and makes the code self documenting.
Reason: It optimizes the code and makes the code self documenting.

Usage Example:

```
class Declare_public_or_protected_method_final_violation
{
        public void method()            // VIOLATION
        {
        }
}
```

Should be written as:

```
class Declare_public_or_protected_method_final_correction
{
        public final void method()      // CORRECTION
        {
        }
}
```

Reference: http://www.javaworld.com/javaworld/jw-12-1998/jw-12-securityrules_p.html

Rule 81: Declare_public_or_protected_class_final

Severity: Medium
Rule: It optimizes the code and makes the code self documenting.
Reason: It optimizes the code and makes the code self documenting.

Usage Example:

```
public class Test // VIOLATION
{
        public void fubar()
        {
                //....
        }
}
```

Should be written as:

```
public final class Test // FIXED
{
        public void fubar()
        {
                //....
        }
}
```

Reference: http://www.javaworld.com/javaworld/jw-12-1998/jw-12-securityrules_p.html

Rule 82: Use_array_of_primitive_type_insteadof_collection

Severity: Medium
Rule: Instead of using some 'Collection' for wrapper classes of primitive types, use arrays of primitive types.
Reason: Instead of using some 'Collection' for wrapper classes of primitive types, use arrays of primitive types.

Usage Example:

```
package com.rule;

import java.util.Vector;

public class Use_array_of_primitive_type_insteadof_collection_violation
{
        public void method()
        {
                Vector vInt = new Vector(); // VIOLATION
                vInt.add(new Integer(1));
                vInt.add(new Integer(2));
                vInt.add(new Integer(3));
        }
}
```

Should be written as:

```
package com.rule;

public class Use_array_of_primitive_type_insteadof_collection_correction
{
        public void method()
        {
                int[] arrInt = new int[3]; // CORRECTION
                arrInt[0] = 1;
                arrInt[1] = 2;
                arrInt[2] = 3;
        }
}
```

Reference: No reference available.

Rule 83: Avoid_using_Double_toString

Severity: High
Rule: Avoid using Double toString for performance reasons
Reason: Avoid using Double toString for performance reasons

Usage Example:

```
package com.rule;
class Avoid_using_Double_toString_violation
{
        public void method(double d)
        {
                String dStr = Double.toString(d);               // VIOLATION
                System.out.println(dStr);
        }
}
```

Should be written as:

```
Avoid Using Double.toString() instead use custom conversion algorithm.
```

Reference: http://www.onjava.com/pub/a/onjava/2000/12/15/formatting_doubles.html

Rule 84: Avoid_debugging_code

Severity: Low
Rule: Avoid debugging code.
Reason: Avoid debugging code.

Usage Example:

```
package com.rule;

public void Avoid_debugging_code_violation
{
	private int count =0;
	public int getCount()
	{
		//...
		System.out.println("count ="+count);  // Violation
		return count;
	}
}
```

Should be written as:

```
package com.rule;

public void Avoid_debugging_code_correction
{
	private int count =0;
	public int getCount()
	{
		if (Debug.ON)
		{
			System.out.println("count ="+count);  //correction
		}
		return count;
	}
}
```

Reference: http://www.rgagnon.com/javadetails/java-0130.html

Rule 85: Avoid_using_Thread_dumpStack

Severity: Low
Rule: Avoid using Thread.dumpStack() in production code since it is typically associated at the time of manual profiling activities.
Reason: Avoid using Thread.dumpStack() in production code since it is typically associated at the time of manual profiling activities.

Usage Example:

```
public class Test
{
	public static void main( String[] args )
	{
		for ( int i = 0; i < args.length; i++ )
		{
			System.out.println( args[ i ] );
		}
		Thread.dumpStack(); // VIOLATION
	}
}
```

Should be written as:

```
Use some profiling tool instead.
```

Reference: No references available.

Rule 86: Avoid_using_java_lang_Runtime_freeMemory

Severity: Low
Rule: Avoid using java.lang.Runtime.freeMemory() in production code since it is typically associated at the time of manual profiling activities.
Reason: Avoid using java.lang.Runtime.freeMemory() in production code since it is typically associated at the time of manual profiling activities.

Usage Example:

```
public class Test
{
	public static void main(String[] args)
	{
		System.out.println( Runtime.getRuntime().freeMemory() ); // VIOLATION
		int[] values = new int[ args.length ];
		for ( int i = 0; i < values.length; i++ )
		{
			values[ i ] = Integer.parseInt( args[ i ] );
		}
		System.out.println( Runtime.getRuntime().freeMemory() ); // VIOLATION
	}
}
```

Should be written as:

```
Use some profiling tool instead.
```

Reference: No references available.

Rule 87: Avoid_using_java_lang_Runtime_totalMemory

Severity: Low
Rule: Avoid using java.lang.Runtime.totalMemory() in production code since it is typically associated at the time of manual profiling activities.
Reason: Avoid using java.lang.Runtime.totalMemory() in production code since it is typically associated at the time of manual profiling activities.

Usage Example:

```
public class Test
{
	public static void main(String[] args)
	{
		System.out.println( Runtime.getRuntime().totalMemory() ); // VIOLATION
		int[] values = new int[ args.length ];
		for ( int i = 0; i < values.length; i++ )
		{
			values[ i ] = Integer.parseInt( args[ i ] );
		}
		System.out.println( Runtime.getRuntime().totalMemory() ); // VIOLATION
	}
}
```

Should be written as:

```
Use some profiling tool instead.
```

Reference: No references available.

Rule 88: Avoid_using_java_lang_Runtime_traceInstructions

Severity: Low
Rule: Avoid using java.lang.Runtime.traceInstructions(boolean on) in production code since it is typically associated at the time of manual profiling activities.
Reason: Avoid using java.lang.Runtime.traceInstructions(boolean on) in production code since it is typically associated at the time of manual profiling activities.

Usage Example:

```
public class Test
{
	public static void main(String[] args)
	{
		Runtime.getRuntime().traceInstructions(true); // VIOLATION
		traced();
	}

	private static void traced()
	{
		System.out.println( "Traced" );  //$NON-NLS-1$
	}

}
```

Should be written as:

```
Use some profiling tool instead.
```

Reference: No references available.

Rule 89: Avoid_using_java_lang_Runtime_traceMethodCalls

Severity: Low
Rule: Avoid using java.lang.Runtime.traceMethodCalls(boolean on) in production code since it is typically associated at the time of manual profiling activities.
Reason: Avoid using java.lang.Runtime.traceMethodCalls(boolean on) in production code since it is typically associated at the time of manual profiling activities.

Usage Example:

```
public class Test
{
	public static void main(String[] args)
	{
		Runtime.getRuntime().traceMethodCalls(true); // VIOLATION
		traced();
	}

	private static void traced()
	{
		System.out.println( "Traced" );  //$NON-NLS-1$
	}

}
```

Should be written as:

```
Use some profiling tool instead.
```

Reference: No references available.

Rule 90: Avoid_using_java_lang_Class_getMethod

Severity: Low
Rule: Instead of hardcoding the method name, directly invoking the method would improve performance and reduce the chances of possible bugs.
Reason: Instead of hardcoding the method name, directly invoking the method would improve performance and reduce the chances of possible bugs.

Usage Example:

```
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;

public class Test
{
	public static void main(String[] args)
	{
		try
		{
			Method method = GetMethod.class.getMethod( "setValue", new Class[] { int.class } ); // VIOLATION  //$NON-NLS-1$
			GetMethod obj = new GetMethod();
			method.invoke( obj, new Object[] {new Integer(1)} );
					System.out.println( obj.getValue() );
		}
		catch (IllegalAccessException e)
		{
			System.out.println( "Can't access private method 'getValue'"); //$NON-NLS-1$
		}
		catch (InvocationTargetException e1)
		{
			System.out.println( "Problem calling method" ); //$NON-NLS-1$
		}
		catch (NoSuchMethodException e2)
		{
			System.out.println( "No method getValue"); //$NON-NLS-1$
		}
	}
}

class GetMethod
{
	public void setValue( int value )
	{
		this.value = value;
	}

	public int getValue()
	{
		return value;
	}

	private int value;
}
```

Should be written as:

```
public class Test
{
	public static void main(String[] args)
	{
		GetMethod obj = new GetMethod();
		obj.setValue(1); // CORRECTION
		System.out.println( obj.getValue() );
	}
}

class GetMethod
{
	public void setValue( int value )
	{
		this.value = value;
	}

	public int getValue()
	{
		return value;
	}

	private int value;
}
```

Reference: No references available.

Rule 91: Avoid_using_java_lang_Class_getField

Severity: Low
Rule: Instead of hardcoding the field name, directly accessing the field would improve performance and reduce the chances of possible bugs.
Reason: Instead of hardcoding the field name, directly accessing the field would improve performance and reduce the chances of possible bugs.

Usage Example:

```
import java.lang.reflect.Field;


public class Test
{

	public static void main(String[] args)
	{
		try
		{
			Field field = GetField.class.getField( "value" );  // VIOLATION //$NON-NLS-1$
			GetField obj = new GetField();
			field.set( obj, new Integer( 1 ) );
			System.out.println( obj.getValue() );
		}
		catch (SecurityException e)
		{
			System.out.println( "Can't access field 'value'"); //$NON-NLS-1$
		}
		catch (NoSuchFieldException e1)
		{
			System.out.println( "No field 'value'"); //$NON-NLS-1$
		}
		catch (IllegalArgumentException e2)
		{
			System.out.println( e2.getMessage() );
		}
		catch (IllegalAccessException e3)
		{
			System.out.println( "Can't access private field 'value'" ); //$NON-NLS-1$
		}
	}

}


class GetField
{
	public int value;

	public void setValue( int value )
	{
		this.value = value;
	}

	public int getValue()
	{
		return value;
	}

}
```

Should be written as:

```
public class Test
{

	public static void main(String[] args)
	{
		GetField obj = new GetField();
		obj.setValue(1);
		System.out.println( obj.getValue() );
	}

}


class GetField
{
	public int value;

	public void setValue( int value )
	{
		this.value = value;
	}

	public int getValue()
	{
		return value;
	}

}
```

Reference: No references available.

Rule 92: Avoid_using_java_lang_Class_getDeclaredMethod

Severity: Low
Rule: Instead of hardcoding the method name, directly invoking the method would improve performance and reduce the chances of possible bugs.
Reason: Instead of hardcoding the method name, directly invoking the method would improve performance and reduce the chances of possible bugs.

Usage Example:

```
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;

public class Test
{
	public static void main(String[] args)
	{
		try
		{
			Method method = GetMethod.class.getDeclaredMethod( "setValue", new Class[] { int.class } ); // VIOLATION  //$NON-NLS-1$
			GetMethod obj = new GetMethod();
			method.invoke( obj, new Object[] {new Integer(1)} );
					System.out.println( obj.getValue() );
		}
		catch (IllegalAccessException e)
		{
			System.out.println( "Can't access private method 'getValue'"); //$NON-NLS-1$
		}
		catch (InvocationTargetException e1)
		{
```

```
                        System.out.println( "Problem calling method" ); //$NON-NLS-1$
                }
                catch (NoSuchMethodException e2)
                {
                        System.out.println( "No method getValue"); //$NON-NLS-1$
                }
        }
}

class GetMethod
{
        public void setValue( int value )
        {
                this.value = value;
        }

        public int getValue()
        {
                return value;
        }

        private int value;
}
```

Should be written as:

```
public class Test
{
        public static void main(String[] args)
        {
                GetMethod obj = new GetMethod();
                obj.setValue(1); // CORRECTION
                System.out.println( obj.getValue() );
        }
}

class GetMethod
{
        public void setValue( int value )
        {
                this.value = value;
        }

        public int getValue()
        {
                return value;
        }

        private int value;
}
```

Reference: No references available.

Rule 93: Avoid_using_java_lang_Class_getDeclaredField

Severity: Low
Rule: Instead of hardcoding the field name, directly accessing the field would improve performance and reduce the chances of possible bugs.
Reason: Instead of hardcoding the field name, directly accessing the field would improve performance and reduce the chances of possible bugs.

Usage Example:

```
import java.lang.reflect.Field;

public class Test
{

        public static void main(String[] args)
        {
                try
                {
                        Field field = GetField.class.getDeclaredField( "value" );  // VIOLATION //$NON-NLS-1$
                        GetField obj = new GetField();
                        field.set( obj, new Integer( 1 ) );
                        System.out.println( obj.getValue() );
                }
                catch (SecurityException e)
                {
                        System.out.println( "Can't access field 'value'"); //$NON-NLS-1$
                }
                catch (NoSuchFieldException e1)
                {
                        System.out.println( "No field 'value'"); //$NON-NLS-1$
                }
                catch (IllegalArgumentException e2)
                {
                        System.out.println( e2.getMessage() );
                }
                catch (IllegalAccessException e3)
                {
                        System.out.println( "Can't access private field 'value'" ); //$NON-NLS-1$
                }
        }

}


class GetField
```

```
{
        public int value;

        public void setValue( int value )
        {
                this.value = value;
        }

        public int getValue()
        {
                return value;
        }

}
```

Should be written as:

```
public class Test
{

        public static void main(String[] args)
        {
                GetField obj = new GetField();
                obj.setValue(1);
                System.out.println( obj.getValue() );
        }

}


class GetField
{
        public int value;

        public void setValue( int value )
        {
                this.value = value;
        }

        public int getValue()
        {
                return value;
        }

}
```

Reference: No references available.

Rule 94: Always_Access_Fields_In_Same_Class_Directly

Severity: Medium
Rule: Avoid accessing fields in the same class through accessor methods.
Reason: Avoid accessing fields in the same class through accessor methods.

Usage Example:

```
class Always_Access_Fields_In_SameClass_Directly_Violation
{
        boolean bModified = false;

        public boolean isModified()
        {
                return bModified;
        }

        public void save()
        {
                if (isModified()) //violation
                {
                        //..save
                }
        }
}
```

Should be written as:

```
class Always_Access_Fields_In_SameClass_Directly_Correction
        {
                boolean bModified = false;

                public boolean isModified()
                {
                        return bModified;
                }

                public void save()
                {
                        if (bModified) //correction
                        {
                                //..save
                        }
                }
        }
```

Reference: No reference

Rule 95: Use_arrayList_inplace_of_vector

Severity: Medium
Rule:
Use 'ArrayList' in place of 'Vector' wherever possible ArrayList is faster than Vector except when there is no lock acquisition required in HotSpot JVMs (when they have about the same perfc
Reason:
Use 'ArrayList' in place of 'Vector' wherever possible ArrayList is faster than Vector except when there is no lock acquisition required in HotSpot JVMs (when they have about the same perfc

Usage Example:

```
package com.rule;
import java.util.Vector;

class Use_arrayList_inplace_of_vector_violation
{
  final int SIZE = 10;
  private Vector v = new Vector(SIZE);          // VIOLATION

  public int method()
  {
              return v.size();
         }
}
```

Should be written as:

```
package com.rule;
import java.util.ArrayList;
class Use_arrayList_inplace_of_vector_correction
{
        final int SIZE = 10;
        private ArrayList al = new ArrayList(SIZE);             // CORRECTION

        public int method()
        {
                return al.size();
        }
}
```

Reference: www.javaperformancetuning.com/tips/rawtips.shtml
http://www.onjava.com/pub/a/onjava/2001/05/30/optimization.html
http://www.glenmccl.com/jperf/

Rule 96: Avoid_unnecessary_exception_throwing

Severity: Low
Rule: Avoid unnecessary exception throwing for performance reasons
Reason: Avoid unnecessary exception throwing for performance reasons

Usage Example:

```
class Avoid_unnecessary_exception_throwing_violation
{
        public void method() throws java.io.IOException         // VIOLATION
        {
                // no code that throws IOException
        }
}
```

Should be written as:

```
class Avoid_unnecessary_exception_throwing_correction
{
        public void method() // CORRECTION
        {
                // no code that throws IOException
        }
}
```

Reference: Reference Not Available.

Rule 97: Avoid_LinkedLists

Severity: High
Rule: Avoid LinkedLists instead use Vector / ArrayList as LinkedList implementation has a performance overhead for indexed access.
Reason: Avoid LinkedLists instead use Vector / ArrayList as LinkedList implementation has a performance overhead for indexed access.

Usage Example:

```
package com.rule;

import java.util.LinkedList;            // VIOLATION

class Avoid_LinkedLists_violation
{
        public void method()
        {
                LinkedList list = new LinkedList();
                if(list != null)
                {
                        list = null;
                }
        }
}
```

Should be written as:

```
package com.rule;

import java.util.ArrayList;             // CORRECTION
```

```
class Avoid_LinkedLists_correction
{
        public void method()
        {
                 final int SIZE = 10;
                 ArrayList list = new ArrayList(SIZE);

                 if(list != null)
                 {
                        list = null;
                 }
        }
}
```

Reference: www.onjava.com/pub/a/onjava/2001/05/30/optimization.html

Rule 98: Use_single_quotes_when_concatenating_character_to_String

Severity: Medium
Rule: Use single quotes instead of double qoutes when concatenating single character to a String.
Reason: Use single quotes instead of double qoutes when concatenating single character to a String.

Usage Example:

```
package com.rule;
public class Use_single_quotes_when_concatenating_character_to_String_violation
{
        Use_single_quotes_when_concatenating_character_to_String_violation()
        {
                String s = "a";
                s = s + "a"; // VIOLATION
        }
}
```

Should be written as:

```
package com.rule;
public class Use_single_quotes_when_concatenating_character_to_String_correction
{
        Use_single_quotes_when_concatenating_character_to_String_correction()
        {
                String s = "a";
                s = s + 'a'; // CORRECTION
        }
}
```

Reference: Reference Not Available.

Rule 99: Use_PreparedStatement_instead_of_Statement

Severity: High
Rule: PreparedStatements should be used where dynamic queries are involved.
Reason: PreparedStatements should be used where dynamic queries are involved.

Usage Example:

```
package com.rule;

import java.sql.Connection;
import java.sql.Statement;

public class Use_PreparedStatement_instead_of_Statement_violation
{
        public void method(Connection conn) throws Exception
        {
                Statement stmt = conn.createStatement();
                for (int i = 0; i < empCount; i++)
                {
                        String sQry = "SELECT * FROM employee WHERE id = " + i;
                        ReultSet rs = stmt.execute(sQry);
                        //...
                }
        }
}
```

Should be written as:

```
package com.rule;

import java.sql.Connection;
import java.sql.PreparedStatement;

public class Use_PreparedStatement_instead_of_Statement_correction
{
        public void method(Connection conn) throws Exception
        {
                String sQry = "SELECT * FROM employee WHERE id = ?";
                PreparedStatement pstmt = con.prepareStatement(sQry);
                for (int i = 0; i < empCount; i++)
                {
                        pstmt.setString(1,""+i);
                        ResultSet rs = pstmt.executeQuery();
                        //...
                }
        }
}
```

Reference: http://www.javaworld.com/javaworld/jw-01-2002/jw-0125-overpower.html

Rule 100: Avoid_multi_dimensional_arrays

Severity: Medium
Rule: Try to use single dimensional arrays inplace of multidimensional arrays.
Reason: Try to use single dimensional arrays inplace of multidimensional arrays.

Usage Example:

```
package com.rule;

public class Aviod_multidimensional_arrays_violation
{
        public void method1(int[][] values) // VIOLATION
        {
                for (int i = 0; i < values.length; i++)
                {
                        System.out.println(values[i][0] + ":" + values[i][1]);
                }
        }

        public void method2()
        {
                int[][] arr = new int[][]{ // VIOLATION
                        {1,2},
                        {2,4},
                        {3,6},
                        {4,8}
                };
                method1(arr);
        }
}
```

Should be written as:

```
package com.rule;

public class Aviod_multidimensional_arrays_correction
{
        public void method1(int[] values1, int[] values2) // CORRECTION
        {
                for (int i = 0; i < values1.length; i++)
                {
                        System.out.println(values1[i] + ":" + values2[i]);
                }
        }

        public void method2()
        {
                int[] arr1 = new int[]{1,2,3,4}; // CORRECTION
                int[] arr2 = new int[]{2,4,6,8}; // CORRECTION
                method1(arr1, arr2);
        }
}
```

Reference: http://www.wildtangent.com/developer/downloads/overviews/BestPractices/chapter5

Rule 101: Avoid_empty_synchronized_block

Severity: Low
Rule: Remove empty synchronized blocks to avoid unnecessary overheads
Reason: Remove empty synchronized blocks to avoid unnecessary overheads

Usage Example:

```
package com.rule;

class Avoid_empty_synchronized_block_violation
{
        void method()
        {
                synchronized(this)
                {
                }
        }
}
```

Should be written as:

```
package com.rule;

class Avoid_empty_synchronized_block_violation
{
        void method()
        {
                /*
                synchronized(this)
                {
                }
                */
        }
}
```

Reference: Reference Not Available.

Rule 102: Use_DataSource_instead_of_DriverManager

Severity: High
Rule: Using DataSource is a preferred way for obtaining the Connection objects.
Reason: Using DataSource is a preferred way for obtaining the Connection objects.

Usage Example:

```
package com.rule;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Use_DataSource_instead_of_DriverManager_violation
{
        public void method(String url) throws SQLException
        {
                Connection conn = DriverManager.getConnection(url); // VIOLATION
                // use conn
        }

        public void initDriverManager()
        {
                // load driver class
        }
}
```

Should be written as:

```
package com.rule;

import java.sql.Connection;
import java.sql.SQLException;
import javax.sql.DataSource;

class Use_DataSource_instead_of_DriverManager_correction
{
        DataSource ds;

        public void method() throws SQLException
        {
                Connection conn = ds.getConnection(); // CORRECTION
                // use conn
        }

        public void initDataSource()
        {
                // initialize ds
        }
}
```

Reference: http://java.sun.com/j2se/1.4.2/docs/guide/jdbc/getstart/drivermanager.html
http://java.sun.com/j2se/1.4.2/docs/guide/jdbc/getstart/datasource.html
http://www.javapractices.com/Topic127.cjp

Rule 103: Avoid_call_to_Thread.sleep()

Severity: Critical
Rule: Do not call Thread.sleep() for performance reasons
Reason: Do not call Thread.sleep() for performance reasons

Usage Example:

```
package com.rule;
class Avoid_call_to_Thread_sleep_violation
{
        public void doTest() throws InterruptedException
        {
                //......
                Thread.sleep(100); // VIOLATION
                //......
        }
}
```

Should be written as:

```
package com.rule;
class Avoid_call_to_Thread_sleep_correction
{
        public void doTest()
        {
                //......
                this.wait(100); // CORRECTION
                //......
        }
}
```

Reference: Reference Not Available.

Rule 104: Use_String_instead_StringBuffer_for_constant_strings

Severity: Critical
Rule: Use String instead of StringBuffer for constant Strings
Reason: Use String instead of StringBuffer for constant Strings

Usage Example:

```
package com.rule;
```

```
class Use_String_instead_StringBuffer_for_constant_strings_violation
{
	public String getSign(int i)
	{
		final StringBuffer even = new StringBuffer("EVEN");	// violation
		final StringBuffer odd = new StringBuffer("ODD");	// violation
		final StringBuffer msg = new StringBuffer("The number is ");
		if ((i / 2)*i == i)
		{
			msg.append(even);
		}
		else
		{
			msg.append(odd);
		}
		return msg.toString();
	}
}
```

Should be written as:

```
package com.rule;

class Use_String_instead_StringBuffer_for_constant_strings_correction
{
	public String getSign(int i)
	{
		final String even = "EVEN";	// correction
		final String odd = "ODD";	// correction
		final StringBuffer msg = new StringBuffer("The number is ");
		if ((i / 2)*i == i)
		{
			msg = msg.append(even);
		}
		else
		{
			msg.append(odd);
		}
		return msg.toString();
	}
}
```

Reference: http://java.sun.com/docs/books/tutorial/java/data/whytwo.html

Rule 105: Avoid_using_String_charAt

Severity: High
Rule: Use char[] representation of the string instead of using String.charAt().
Reason: Use char[] representation of the string instead of using String.charAt().

Usage Example:

```
package com.rule;
class Avoid_using_String_charAt_violation
{
	public void method(String str)
	{
		for(int i=0; i<str.length(); i++)
		{
			System.out.println(str.charAt(i));	// VIOLATION
		}
	}
}
```

Should be written as:

```
package com.rule;
class Avoid_using_String_charAt_correction
{
	public void method(String str)
	{
		char[] carr = str.toCharArray();		// CORRECTION
		for(int i=0; i<carr.length; i++)
		{
			System.out.println(carr[i]);		// CORRECTION
		}
	}
}
```

Reference: ftp://ftp.glenmccl.com/pub/free/jperf.pdf
http://www.javaperformancetuning.com/tips/rawtips.shtml

Rule 106: Avoid_Extending_java_lang_Object

Severity: Low
Rule: Avoid empty "finally" block structure.
Reason: Avoid empty "finally" block structure.

Usage Example:

```
package com.rule;

class Avoid_empty_finally_blocks_violation
{
	void method (int i)
	{
		final int ZERO =0;
```

```
                try
                {
                i = ZERO;
                }
                catch(ArithmeticException ae)
                {
                        ae.printStackTrace();
                }
                finally         // VIOLATION
                {
                }
        }
}
```

Should be written as:

```
package com.rule;

class Avoid_empty_finally_blocks_correction
{
        void method (int i)
        {
                final int ZERO = 0;
                try
                {
                i = ZERO;
                }
                catch(ArithmeticException ae)
                {
                        ae.printStackTrace();
                }
                /*
                finally         // COORECTION
                {

                }
                */
        }
}
```

Reference: Reference Not Available.

Rule 107: Use_compound_operators

Severity: Medium
Rule: Use compound operators for improved performance
Reason: Use compound operators for improved performance

Usage Example:

```
package com.rule;
class Use_compound_operators_violation
{
        public void method(int[] a, int x)
        {
                for (int i = 0; i < a.length; i++)
                {
                        a[i] = a[i] + x;                // VIOLATION
                }
        }
}
```

Should be written as:

```
package com.rule;
class Use_compound_operators_correction
{
        public void method(int[] a, int x)
        {
                for (int i = 0; i < a.length; i++)
                {
                        a[i] += x;                      // CORRECTION
                }
        }
}
```

Reference: http://www.patrick.net/jpt/index.html

Rule 108: Avoid_concatenating_Strings_in_StringBuffer

Severity: Medium
Rule: Avoid concatenating Strings in StringBuffer's constructor or append(..) method.
Reason: Avoid concatenating Strings in StringBuffer's constructor or append(..) method.

Usage Example:

```
public class AvoidStringConcat
{
        public void aMethod()
        {
                StringBuffer sb = new StringBuffer("Hello" + getWorld()); // VIOLATION
                sb.append("Calling From " + getJava()); // VIOLATION
        }

        public String getWorld()
        {
                return " World";
        }

        public String getJava()
        {
                return " Java";
         }
}
```

Should be written as:

```
public class AvoidStringConcat
{
        public void aMethod()
        {
                StringBuffer sb = new StringBuffer("Hello");
                sb.append(getWorld());
                sb.append("Calling From ");
                sb.append(getJava());
        }

        public String getWorld()
        {
                return " World";
        }

        public String getJava()
        {
                return " Java";
         }
}
```

Reference: http://java.sun.com/developer/JDCTechTips/2002/tt0305.html#tip1

Rule 109: Declare_package_private_class_final

Severity: Medium
Rule: Declare package private class as final.
Reason: Declare package private class as final.

Usage Example:

```
package com.rule;

class Declare_package_private_method_final_Violation  // Violation
{
}
```

Should be written as:

```
package com.rule;

final class Declare_package_private_method_final_Correction  // Correction
{
}
```

Reference: http://www.javaworld.com/javaworld/jw-12-1998/jw-12-securityrules_p.html

Rule 110: Use_hashMap_inplace_of_hashTable

Severity: Medium
Rule: Use 'HashMap' in place of 'HashTable' wherever possible
Reason: Use 'HashMap' in place of 'HashTable' wherever possible

Usage Example:

```
package com.rule;
import java.util.Hashtable;
class Use_hashMap_inplace_of_hashTable_violation
{
        private static final int SIZE = 10;
        private Hashtable ht = new Hashtable(SIZE);             // VIOLATION

        public void method()
        {
                ht.clear();
        }
}
```

Should be written as:

```
package com.rule;
import java.util.HashMap;
class Use_hashMap_inplace_of_hashTable_correction
{
        private static final int SIZE = 10;
        private HashMap ht = new HashMap(SIZE);                 // CORRECTION

        public void method()
        {
                ht.clear();
        }
}
```

Reference: Reference Not Available.

Rule 111: Avoid_creating_double_from_string

Severity: High
Rule: Avoid creating double from string for improved performance
Reason: Avoid creating double from string for improved performance

Usage Example:

```
public class Avoid_creating_double_from_string_violation
{

                public void method()
                {
                        Double db = new Double("3.44");  // VIOLATION
                        Double.valueOf("3.44");                 // VIOLATION

                        if(db == null)
                        {
                                // Do Something
                                db = null;
                        }
                }
}
```

Should be written as:

```
Avoid converting String to Double
```

Reference: http://www.javaperformancetuning.com/tips/rawtips.shtml
http://www.glenmccl.com/jperf/

Rule 112: Always_use_right_shift_operator_for_division_by_powers_of_two

Severity: Low
Rule: It is more efficient and improves performance if shift operators are used.
Reason: It is more efficient and improves performance if shift operators are used.

Usage Example:

```
public class Test
{
        public int calculate (int num)
        {
          return num / 4;  // VIOLATION
        }
}
```

Should be written as:

```
public class Test
{
        public int calculate (int num)
        {
         return num >> 2;  // FIXED *
}

}

// * Replace the division with an equivalent '>> power', where 'power' is the power
// of two such that 2 ^ power = divisor.
```

Reference: Not available.

Rule 113: Use_shift_operators

Severity: High
Rule: Shift operators are faster than multiplication and division
Reason: Shift operators are faster than multiplication and division

Usage Example:

```
package com.rule;
class Use_shift_operators_violation
{
        public void method()
        {
                int x = 0;
                int X = x / 4;          // VIOLATION
                int Y = x * 2;          // VIOLATION
                X++;
                Y++;
        }
}
```

Should be written as:

```
package com.rule;
class Use_shift_operators_correction
{
        public void method()
        {
                int x = 0;
                int X = x >> 2;         // CORRECTION
                int Y = x << 1;         // CORRECTION
                X++;
                Y++;
        }
}
```

Reference: www.javaworld.com/javaworld/jw-04-1997/jw-04-optimize_p.html

Rule 114: Avoid_java_lang_reflect_package

Severity: High
Rule: Avoid java.lang.reflect package.
Reason: Avoid java.lang.reflect package.

Usage Example:

```
package com.rule;

public class Avoid_java_lang_reflect_package_violation
{
 public Object getData(String classname)
 {
  try
  {
         Class c = Class.forName(classname);
                 Method m = c.getMethod("getData",null);
                 m.invoke(c.newInstance(), null);// Violation
  }
  catch (Exception e)
  {
                        //....
  }
 }
}
```

Should be written as:

```
package com.rule;

public class Avoid_java_lang_reflect_package_correction
{
 public Object getData(String classname)
 {
  try {
Class c = Class.forName(classname);
IDataProvider dataprovider = (IDataProvider) c.newInstance();  // Correction
return dataprovider.getData();
  }
  catch (Exception e)
  {
  }
 }
}

interface IDataProvider
{
        Object getData(String name);
}
```

Reference: Reference not available.

Rule 115: Use_NIO_in_server

Severity: High
Rule: Using NIO for server is better than using traditional java.net package.
Reason: Using NIO for server is better than using traditional java.net package.

Usage Example:

```
package com.rule;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class Use_NIO_in_server_violation
{
        public void method(ServerSocket ss) throws IOException
        {
                Socket soc = ss.accept(); // VIOLATION
                // use soc
        }
}
```

Should be written as:

```
Use classes in java.nio.channels in the server code.
```

Reference: http://www-106.ibm.com/developerworks/java/library/j-javaio/
http://www.onjava.com/pub/a/onjava/2002/10/02/javanio.html?page=4

Rule 116: Avoid_object_instantiation_in_loops

Severity: Critical
Rule: Avoid object instantiation in frequently executed code for performance reasons
Reason: Avoid object instantiation in frequently executed code for performance reasons

Usage Example:

```
package com.rule;

import java.util.ArrayList;

class Avoid_object_instantiation_violation
{
        public void action()
        {
                ArrayList al = getNameAndValues();
                for (int i = 0; i < al.size(); i++)
                {
                        ClassName cn = (ClassName) al.get(i);
                        String sArr[] = new String[] { cn.getName(), cn.getValue() }; //Violation
                        //...
                }
        }

        private ArrayList getNameAndValues()
        {
                ArrayList al = new ArrayList();
                // populate list
                return al;
        }

        private class ClassName
        {
                String name;
                String value;

                String getName()
                {
                        return name;
                }
                String getValue()
                {
                        return value;
                }
        }
}
```

Should be written as:

```
package com.rule;

import java.util.ArrayList;

class Avoid_object_instantiation_correction
{
        public void action()
        {
                ArrayList al = getNameAndValues();
                String sArr[] = new String[2]; //Correction
                for (int i = 0; i < al.size(); i++)
                {
                        ClassName cn = (ClassName) al.get(i);
                        sArr[0] = cn.getName();
                        sArr[1] = cn.getValue();
                        //...
                }
        }

        private ArrayList getNameAndValues()
        {
                ArrayList al = new ArrayList();
                // populate list
                return al;
        }

        private class ClassName
        {
                String name;
                String value;

                String getName()
                {
                        return name;
                }
                String getValue()
                {
                        return value;
                }
        }
}
```

Reference: http://www.oreilly.com/catalog/javapt/chapter/ch04.html

Rule 117: Avoid_unnecessary_instanceof

Severity: High
Rule: Avoid unnecessary "instanceof" evaluations.
Reason: Avoid unnecessary "instanceof" evaluations.

Usage Example:

```
package com.rule;

class Avoid_unnecessary_instanceof_violation
{
        private String obj = "String"; //$NON-NLS-1$
        public void method()
        {
                if (obj instanceof Object)              // VIOLATION
                {
                        obj.getClass();
                }
        }
}
```

Should be written as:

```
package com.rule;

class Avoid_unnecessary_instanceof_correction
{
        private String obj = "String"; //$NON-NLS-1$

        public void method()
        {
                /*                                                      // CORRECTION
                if (obj instanceof Object)
                {
                        obj.getClass();
                }
                */
                obj.getClass();
        }
}
```

Reference: Reference Not Available.

Rule 118: Define_initial_capacities

Severity: Medium
Rule: Expansion of array capacity involves allocating a larger array and copying the contents of the old array to a new one. Eventually, the old array object gets reclaimed by the garbage collector. Array expansion is an expensive operation. Usually one may have a pretty good guess at the expected size which should be used instead of the default.
Reason: Expansion of array capacity involves allocating a larger array and copying the contents of the old array to a new one. Eventually, the old array object gets reclaimed by the garbage collector. Array expansion is an expensive operation. Usually one may have a pretty good guess at the expected size which should be used instead of the default.

Usage Example:

```
package com.rule;

import java.util.ArrayList;

class Define_initial_capacities_violation
{
        private ArrayList al = new ArrayList();         // VIOLATION

        public int method()
        {
                return al.size();
        }
}
```

Should be written as:

```
package com.rule;

import java.util.ArrayList;

class Define_initial_capacities_correction
{
        private final int SIZE = 10;
        private ArrayList al = new ArrayList(SIZE);             // CORRECTION

        public int method()
        {
                return al.size();
        }
}
```

Reference: http://www.oreilly.com/catalog/javapt/chapter/ch04.html
Dov Bulka, "Java Performance and Scalability Volume 1: Server-Side Programming Techniques" Addison Wesley, ISBN: 0-201-70429-3 pp.55 57
Neal Ford, "Performance Tuning With Java Technology" JavaOne 2001 Conference

Rule 119: Avoid_empty_catch_blocks

Severity: Low
Rule: Avoid empty "catch" block structure.
Reason: Avoid empty "catch" block structure.

Usage Example:

```
package com.rule;

class Avoid_empty_catch_blocks_violation
{
        void method(int i)
        {
                try
                {
                        i++;
                }
                catch(ArithmeticException ae)           // VIOLATION
                {
                }
        }
}
```

Should be written as:

```
package com.rule;

class Avoid_empty_catch_blocks_correction
{
        void method (int i)
        {
                try
                {
                        i++;
                }
                catch(ArithmeticException ae)
                {
                        ae.printStackTrace();           // CORRECTION
                }
        }
}
```

Reference: Reference Not Available.

Rule 120: Avoid_synchronized_methods_in_loop

Severity: High
Rule: Avoid synchronized methods in loop for performance reasons
Reason: Avoid synchronized methods in loop for performance reasons

Usage Example:

```
package com.rule;
class Avoid_synchronized_methods_in_loop_violation
{
        public synchronized Object remove()
        {
                Object obj = null;
                //...
                return obj;
        }
        public void removeAll()
        {
                for(;;)
                {
                        remove();               // VIOLATION
                }
        }
}
```

Should be written as:

```
package com.rule;
class Avoid_synchronized_methods_in_loop_correction
{
        public Object remove()          // CORRECTION
        {
                Object obj = null;
                //....
                return obj;
        }
        public synchronized void removeAll()
        {
                for(;;)
                {
                        remove();               // CORRECTION
                }
        }
}
```

Reference: Reference Not Available.

Rule 121: Avoid_synchronized_blocks_in_loop

Severity: High
Rule: Avoid synchronized blocks in loop for performance reasons
Reason: Avoid synchronized blocks in loop for performance reasons

Usage Example:

```
package com.rule;
class Avoid_synchronized_blocks_in_loop_violation
{
        public static void main(String[] args)
        {
                Object lock = new Object();

                for ( int i = < 0; i < args.length; i ++)
                {
                        synchronized ( lock ) // VIOLATION
                        {
                                System.out.println( args[ i ] );
                        }
                }
        }

}
```

Should be written as:

```
package com.rule;
class Avoid_synchronized_blocks_in_loop_correction
{
	public static void main(String[] args)
	{
		Object lock = new Object();

		synchronized ( lock ) // CORRECTION
		{
			for ( int i = < 0; i < args.length; i ++)
			{
				System.out.println( args[ i ] );
			}
		}
	}

}
```

Reference: Reference Not Available.

Rule 122: Close_jdbc_resources

Severity: High
Rule: Always close the JDBC resources opened.
Reason: Always close the JDBC resources opened.

Usage Example:

```
package com.rule;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class Close_jdbc_resources_violation
{
	public void method(String url, String query)
	{
		Connection conn = null;
		Statement stmt = null;
		ResultSet rs = null;
		try
		{
			conn = DriverManager.getConnection(url);
			stmt = conn.createStatement();  // VIOLATION
			rs = stmt.executeQuery(query);  // VIOLATION
		}
		catch (Exception e)
		{
		}
		finally
		{
			try
			{
				conn.close();
			}
			catch (Exception e)
			{
			}
		}
	}
}
```

Should be written as:

```
package com.rule;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class Close_jdbc_resources_correction
{
	public void method(String url, String query)
	{
		Connection conn = null;
		Statement stmt = null;
		ResultSet rs = null;

		try
		{
			conn = DriverManager.getConnection(url);
			stmt = conn.createStatement();
			rs = stmt.executeQuery(query);
		}
		catch (Exception e)
		{
		}
		finally
		{
			try
			{
				rs.close();		// CORRECTION
				stmt.close();	// CORRECTION
				conn.close();
```

```
                        }
                        catch (Exception e)
                        {
                        }
                }
        }
}
```

Reference: Reference not available.

Rule 123: Close_jdbc_resources_only_in_finally_block

Severity: High
Rule: The place to close jdbc resources is finally block.
Reason: The place to close jdbc resources is finally block.

Usage Example:

```
package com.rule;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class Close_jdbc_resources_only_in_finally_block_violation
{
        public void method(String url, String query)
        {
                Connection conn = null;
                Statement stmt = null;
                ResultSet rs = null;
                try
                {
                        conn = DriverManager.getConnection(url);
                        stmt = conn.createStatement();
                        rs = stmt.executeQuery(query);

                        rs.close();     // VIOLATION
                        stmt.close();   // VIOLATION
                        conn.close();   // VIOLATION

                }
                catch (Exception e)
                {
                }
                finally
                {

                }
        }
}
```

Should be written as:

```
package com.rule;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class Close_jdbc_resources_only_in_finally_block_correction
{
        public void method(String url, String query)
        {
                Connection conn = null;
                Statement stmt = null;
                ResultSet rs = null;

                try
                {
                        conn = DriverManager.getConnection(url);
                        stmt = conn.createStatement();
                        rs = stmt.executeQuery(query);
                }
                catch (Exception e)
                {
                }
                finally
                {
                        try
                        {
                                rs.close();             // CORRECTION
                                stmt.close();   // CORRECTION
                                conn.close();   // CORRECTION
                        }
                        catch (Exception e)
                        {
                        }
                }
        }
}
```

Reference: Reference not available.

Rule 124: Specify_StringBuffer_capacity

Severity: Medium
Rule: The Default 'StringBuffer' constructor will create a character array of a default size(16).
When size exceeds its capacity, it has to allocate a new character array with a larger capacity, it copies the old contents into the new array, and eventually discard the old array.By specifying enough capacity during construction prevents the 'StringBuffer' from ever needing expansion.
Reason: The Default 'StringBuffer' constructor will create a character array of a default size(16).
When size exceeds its capacity, it has to allocate a new character array with a larger capacity, it copies the old contents into the new array, and eventually discard the old array.By specifying enough capacity during construction prevents the 'StringBuffer' from ever needing expansion.

Usage Example:

```
package com.rule;

class Specify_StringBuffer_capacity_violation
{
        private StringBuffer sb = new StringBuffer();           // VIOLATION

        public void method(int i)
        {
                sb.append(i);
        }
}
```

Should be written as:

```
package com.rule;

class Specify_StringBuffer_capacity_correction
{
        private final int SIZE = 10;
        private StringBuffer sb = new StringBuffer(SIZE);               // CORRECTION

        public void method(int i)
        {
                sb.append(i);
        }
}
```

Reference: Reference Not Available.

Rule 125: Avoid_using_exponentiation

Severity: Medium
Rule: Do not use exponentiation.
Reason: Do not use exponentiation.

Usage Example:

```
package com.rule;

public class Avoid_using_exponentiation_violation
{
        public int getPower(int iBase, int iPow)
        {
                int iRet = (int) Math.pow(iBase, iPow); // VIOLATION
                return iRet;
        }
}
```

Should be written as:

```
package com.rule;

public class Avoid_using_exponentiation_correction
{
        public int getPower(int iBase, int iPow)
        {
                int iRet = 1;
                for (int i = 0; i < iPow; i++) // CORRECTION
                {
                        iRet *= iBase;
                }
                return iRet;
        }
}
```

Reference: www.javaperformancetuning.com/tips/rawtips.shtml

Rule 126: Use_transient_keyword

Severity: High
Rule: Serialization can be very costly. Using the transient keyword reduces the amount of data serialized
Reason: Serialization can be very costly. Using the transient keyword reduces the amount of data serialized

Usage Example:

```
package com.rule;
import java.io.Serializable;
class Use_transient_keyword_violation implements Serializable
{
        final String field1;            // VIOLATION
        private String field2;
        private String field3;

        Use_transient_keyword_violation(int i, int j)
        {
                field2 = i;
                field3 = j;
                field1 = field2 + field3;
        }
}
```

Should be written as:

```
package com.rule;

import java.io.Serializable;
class Use_transient_keyword_correction implements Serializable
{
        final transient String field1;  // CORRECTION
        private String field2;
        private String field3;

        Use_transient_keyword_violation(int i, int j)
        {
                field2 = i;
                field3 = j;
                field1 = field2 + field3;
        }
}
```

Reference: java.sun.com\docs\books\performance\1st_edition\html\JPIOPerformance.fm.html

Rule 127: Avoid_new_inside_getTableCellRendererComponent

Severity: Critical
Rule: Avoid new inside the method getTableCellRendererComponent.
Reason: Avoid new inside the method getTableCellRendererComponent.

Usage Example:

```
package com.rule;

import javax.swing.table.*;
import javax.swing.*;
import java.awt.*;

public class AvoidNewInside_getTableCellRendererComponent_violation extends DefaultTableCellRenderer
{
  public Component getTableCellRendererComponent(JTable aTable, Object  aNumberValue,boolean aIsSelected,boolean aHasFocus,int aRow, int aColumn)
  {
                test tt =new test();  // Violation
                tt.fillColour();
                return aTable;
  }
}
class test
{
        public void fillColour()
        {
                //...
        }
}
```

Should be written as:

```
Avoid new inside getTableCellRendererComponent.
```

Reference: http://www.javapractices.com/Topic168.cjp

Rule 128: Avoid_Integer_valueOf_intValue

Severity: High
Rule: Avoid using Integer.valueOf(String).intValue() instead call Integer.parseInt(String).
Reason: Avoid using Integer.valueOf(String).intValue() instead call Integer.parseInt(String).

Usage Example:

```
package com.rule;

class Avoid_Integer_valueOf_intValue_violation
{
        public int getValue(String s)
        {
                return Integer.valueOf(s).intValue();           // VIOLATION
        }
}
```

Should be written as:

```
package com.rule;

class Avoid_Integer_valueOf_intValue_correction
{
        public int getValue(String s)
        {
                return Integer.parseInt(s);             // CORRECTION
        }
}
```

Reference: Reference Not Available.

Rule 129: Use_ternary_operator_for_assignment

Severity: Medium
Rule: Use ternary operator for assignment.

Reason: Use ternary operator for assignment.

Usage Example:

```
package com.rule;

public class Use_ternary_operator_for_assignment_violation
{
        int iVal;
        private void foo(int a, int b)
        {
                if(a<b) // Violation
                {
                        iVal = a;
                }
                else
                {
                        iVal = b;
                }
        }
}
```

Should be written as:

```
package com.rule;

public class Use_ternary_operator_for_assignment_violation
{
        int iVal;
        private void foo(int a, int b)
        {
                iVal = (a<b )? a:b;// Correction
        }
}
```

Reference: Reference not available.

Rule 130: Avoid_Serialized_class_with_no_instance_variables

Severity: Low
Rule: A class with no instance variables, doesn't have any state information that needs to be preserved
Reason: A class with no instance variables, doesn't have any state information that needs to be preserved

Usage Example:

```
package com.rule;

import java.io.Serializable;

class Avoid_Serialized_class_with_no_instance_variables_violation implements Serializable
{
}
```

Should be written as:

```
package com.rule;

import java.io.Serializable;

class Avoid_Serialized_class_with_no_instance_variables_violation // implements Serializable
{
}
```

Reference: Reference Not Available.

Rule 131: Avoid_Vector_elementAt_inside_loop

Severity: Medium
Rule: Avoid calling Vector.elementAt() inside loop
Reason: Avoid calling Vector.elementAt() inside loop

Usage Example:

```
package com.rule;

import java.util.Vector;

class Avoid_Vector_elementAt_inside_loop_violation
{
        public void method(Vector v)
        {
                int size = v.size();
                for(int i=size; i>0; i--)
                {
                        System.out.println((String) v.elementAt(size-i));       // VIOLATION
                }
        }
}
```

Should be written as:

```
package com.rule;

import java.util.Vector;

class Avoid_Vector_elementAt_inside_loop_correction
{
        public void method(Vector v)
```

```
	{
		int size = v.size();
		Object vArr[] = v.toArray();
		for(int i=0; i<size; i++)
		{
			System.out.println((String) vArr[i]);		// CORRECTION
		}
	}
}
```

Reference: http://www.javaperformancetuning.com/tips/rawtips.shtml
http://jsl.jcon.org/javaperformance.html

Rule 132: Declare_variable_final

Severity: Medium
Rule: Any variable that is initialized and never assigned to should be declared final.
Reason: Any variable that is initialized and never assigned to should be declared final.

Usage Example:

```
package com.rule;

class Declare_variable_final_violation
{
	public void method()
	{
		int i = 5;  // VIOLATION
		int j = i;
		j = j + i;
	}
}
```

Should be written as:

```
package com.rule;

class Declare_variable_final_correction
{
	public void method()
	{
		final int i = 5;  // CORRECTION
		int j = i;
		j = j + i;
	}
}
```

Reference: Reference Not Available.

Rule 133: Declare_method_arguments_final

Severity: Medium
Rule: Any method argument which is neither used nor assigned should be declared final.
Reason: Any method argument which is neither used nor assigned should be declared final.

Usage Example:

```
package com.rule;

class Declare_method_arguments_final_violation
{
	public void method(int i,int j)  // VIOLATION
	{
		j = j + i;
	}
}
```

Should be written as:

```
package com.rule;

class Declare_method_arguments_final_correction
{
	public void method(final int i,int j)  // CORRECTION
	{
		j = j + i;
	}
}
```

Reference: Reference not available.

Rule 134: Avoid_polling_loops

Severity: High
Rule: Do not use polling loops / busy waiting.
Reason: Do not use polling loops / busy waiting.

Usage Example:

```
package com.rule;

public class Avoid_polling_loops_violation
{
	boolean bContinue;

	void doSomething()
	{
```

```
                while(!bContinue)
                {
                        try
                        {
                                Thread.sleep(250); // VIOLATION
                        }
                        catch (InterruptedException e)
                        { }
                }

                // take action
        }
}
```

Should be written as:

```
package com.rule;

public class Avoid_polling_loops_correction
{
        boolean bContinue;
        Object objLock; // notifyAll() will be called on this object from some other thread

        void doSomething()
        {
                synchronized(objLock)
                {
                        while(!bContinue)
                        {
                                try
                                {
                                        objLock.wait(); // CORRECTION
                                }
                                catch (InterruptedException e)
                                { }
                        }
                }

                // take action
        }
}
```

Reference:

Rule 135: Avoid_Serialized_class_with_only_transient_fields

Severity: Low
Rule: The class with only transient fields doesn't have any state information that needs to be preserved
Reason: The class with only transient fields doesn't have any state information that needs to be preserved

Usage Example:

```
package com.rule;

import java.io.Serializable;

class Avoid_Serialized_class_with_only_transient_fields_violation implements Serializable
 //Violation
{
        transient int count = 0;
}
```

Should be written as:

```
package com.rule;

import java.io.Serializable;

class Avoid_Serialized_class_with_only_transient_fields_violation // implements Serializable
//Correction
{
        transient int count = 0;
}
```

Reference: Reference Not Available.