**Projeto: Commons-io**

**Confiança Mínima: 0%**

1. **Automatizado sem estruturas de controle(Copy and paste?)**

**Padrão:**

**Usuários que chamam:**

IOUtils.*closeQuietly*(Writer);

**Também chamam:**

IOUtils.*closeQuietly*(Writer);

Suporte: 0.00634 (7)

Confiança: 0.411

**Método Minerado:**

**private** **static** Writer FileWriterWithEncoding.initWriter(File file, Object encoding, **boolean** append) **throws** IOException {

**if** (file == **null**) {

**throw** **new** NullPointerException("File is missing");

}

**if** (encoding == **null**) {

**throw** **new** NullPointerException("Encoding is missing");

}

**boolean** fileExistedAlready = file.exists();

OutputStream stream = **null**;

Writer writer = **null**;

**try** {

stream = **new** FileOutputStream(file, append);

**if** (encoding **instanceof** Charset) {

writer = **new** OutputStreamWriter(stream, (Charset)encoding);

} **else** **if** (encoding **instanceof** CharsetEncoder) {

writer = **new** OutputStreamWriter(stream, (CharsetEncoder)encoding);

} **else** {

writer = **new** OutputStreamWriter(stream, (String)encoding);

}

} **catch** (IOException ex) {

IOUtils.*closeQuietly*(writer);

IOUtils.*closeQuietly*(stream);

**if** (fileExistedAlready == **false**) {

FileUtils.*deleteQuietly*(file);

}

**throw** ex;

} **catch** (RuntimeException ex) {

IOUtils.*closeQuietly*(writer);

IOUtils.*closeQuietly*(stream);

**if** (fileExistedAlready == **false**) {

FileUtils.*deleteQuietly*(file);

}

**throw** ex;

}

**return** writer;

}

**Padrão sendo utilizado:**

**private** Writer LockableFileWriter.initWriter(File file, Charset encoding, **boolean** append) **throws** IOException {

**boolean** fileExistedAlready = file.exists();

OutputStream stream = **null**;

Writer writer = **null**;

**try** {

**if** (encoding == **null**) {

writer = **new** FileWriter(file.getAbsolutePath(), append);

} **else** {

stream = **new** FileOutputStream(file.getAbsolutePath(), append);

writer = **new** OutputStreamWriter(stream, encoding);

}

} **catch** (IOException ex) {

IOUtils.closeQuietly(writer);

IOUtils.closeQuietly(stream);

FileUtils.deleteQuietly(lockFile);

**if** (fileExistedAlready == **false**) {

FileUtils.deleteQuietly(file);

}

**throw** ex;

} **catch** (RuntimeException ex) {

IOUtils.closeQuietly(writer);

IOUtils.closeQuietly(stream);

FileUtils.deleteQuietly(lockFile);

**if** (fileExistedAlready == **false**) {

FileUtils.deleteQuietly(file);

}

**throw** ex;

}

**return** writer;

}

1. **Erro sem estruturas de controle (Uso normal)**

**Padrão:**

**Usuários que chamam:**

IOUtils.closeQuietly(Writer);

**Também chamam:**

IOUtils.closeQuietly(Writer);

Suporte: 0.00634 (7)

Confiança: 0.411

**Padrão não sendo utilizado:**

**public** **void** TailerTest.testLongFile() **throws** Exception {

**long** delay = 50;

File file = **new** File(getTestDirectory(), "testLongFile.txt");

createFile(file, 0);

Writer writer = **new** FileWriter(file, **true**);

**for** (**int** i = 0; i < 100000; i++) {

writer.write("LineLineLineLineLineLineLineLineLineLine\n");

}

writer.write("SBTOURIST\n");

IOUtils.closeQuietly(writer);

TestTailerListener listener = **new** TestTailerListener();

tailer = **new** Tailer(file, listener, delay, **false**);

**long** start = System.currentTimeMillis();

Thread thread = **new** Thread(tailer);

thread.start();

List<String> lines = listener.getLines();

**while** (lines.isEmpty() || !lines.get(lines.size() - 1).equals("SBTOURIST")) {

lines = listener.getLines();

}

System.out.println("Elapsed: " + (System.currentTimeMillis() - start));

listener.clear();

}

**Confiança Mínima: 20%**

1. **Erro sem estruturas de controle (Uso normal)**

**Padrão**

**Usuários que chamam:**

java.io.File.isDirectory()

**Também chamam:**

java.io.File.exists()

**Confiança:** 0.2121212526721822

**Suporte:** 0.00634

**Método Minerado:**

**public** **static** FileOutputStream FileUtils.openOutputStream(File file) **throws** IOException {

**if** (file.exists()) {

**if** (file.isDirectory()) {

**throw** **new** IOException("File '" + file + "' exists but is a directory");

}

**if** (file.canWrite() == **false**) {

**throw** **new** IOException("File '" + file + "' cannot be written to");

}

} **else** {

File parent = file.getParentFile();

**if** (parent != **null** && parent.exists() == **false**) {

**if** (parent.mkdirs() == **false**) {

**throw** **new** IOException("File '" + file + "' could not be created");

}

}

}

**return** **new** FileOutputStream(file);

}

**Padrão não sendo utilizado:**

/\*\*

\* Extract the directories.

\*/

**private** List DirectoryWalkerTestCaseJava4.directoriesOnly(Collection results) {

List list = **new** ArrayList(results.size());

**for** (Iterator it = results.iterator(); it.hasNext(); ) {

File file = (File) it.next();

**if** (file.isDirectory()) {

list.add(file);

}

}

**return** list;

}

1. **Erro sem estruturas de controle (Uso normal)**

**Padrão**

**Usuários que chamam:**

java.io.File.isDirectory()

**Também chamam:**

java.io.File.exists()

**Confiança:** 0.2121212526721822

**Suporte:** 0.00634

**Padrão não sendo utilizado:**

**public** **static** **boolean** FileUtils.directoryContains(**final** File directory, **final** File child) **throws** IOException {

// Fail fast against NullPointerException

**if** (directory == **null**) {

**throw** **new** IllegalArgumentException("Directory must not be null");

}

**if** (!directory.isDirectory()) {

**throw** **new** IllegalArgumentException("Not a directory: " + directory);

}

**if** (child == **null**) {

**return** **false**;

}

// Canonicalize paths (normalizes relative paths)

String canonicalParent = directory.getCanonicalPath();

String canonicalChild = child.getCanonicalPath();

**if** (IOCase.SYSTEM.checkEquals(canonicalParent, canonicalChild)) {

**return** **false**;

}

**return** IOCase.SYSTEM.checkStartsWith(canonicalChild, canonicalParent);

}

1. **Erro sem estruturas de controle (Uso normal)**

**Padrão**

**Usuários que chamam:**

java.io.File.isDirectory()

**Também chamam:**

java.io.File.exists()

**Confiança:** 0.2121212526721822

**Suporte:** 0.00634

**Padrão não sendo utilizado:**

**private** **static** **void** FileUtils.innerListFiles(Collection<File> files, File directory,

IOFileFilter filter, **boolean** includeSubDirectories) {

File[] found = directory.listFiles((FileFilter) filter);

**if** (found != **null**) {

**for** (File file : found) {

**if** (file.isDirectory()) {

**if** (includeSubDirectories) {

files.add(file);

}

innerListFiles(files, file, filter, includeSubDirectories);

} **else** {

files.add(file);

}

}

}

}

1. **Erro sem estruturas de controle (Uso normal)**

**Padrão**

**Usuários que chamam:**

java.io.File.isDirectory()

**Também chamam:**

java.io.File.exists()

**Confiança:** 0.2121212526721822

**Suporte:** 0.00634

**Padrão não sendo utilizado:**

**public** **static** Collection<File> FileUtils.listFilesAndDirs(

File directory, IOFileFilter fileFilter, IOFileFilter dirFilter) {

validateListFilesParameters(directory, fileFilter);

IOFileFilter effFileFilter = setupEfectiveFileFilter(fileFilter);

IOFileFilter effDirFilter = setupEffectiveDirFilter(dirFilter);

//Find files

Collection<File> files = **new** java.util.LinkedList<File>();

**if** (directory.isDirectory()) {

files.add(directory);

}

innerListFiles(files, directory,

FileFilterUtils.or(effFileFilter, effDirFilter), **true**);

**return** files;

}

**Confiança Mínima: 40%**

1. **Erro com estruturas de controle (Influência supersequência)**

**Padrão:**

**Usuários que chamam:**

junit.framework.Assert.assertTrue(java.lang.String, Z)

junit.framework.Assert.assertTrue(java.lang.String, Z)

junit.framework.Assert.assertTrue(java.lang.String, Z)

**Também chamam:**

junit.framework.Assert.fail(java.lang.String)

**Suporte:** 0.00543 (6)

**Confiança:** 0.4666

**Método Minerado (Supersequência minerada)**

**public** **void** FileFilterTestCase.assertFiltering(IOFileFilter filter, File file, **boolean** expected) **throws** Exception {

// Note. This only tests the (File, String) version if the parent of

// the File passed in is not null

*assertTrue*("Filter(File) " + filter.getClass().getName() + " not " + expected + " for " + file, (filter.accept(file) == expected));

**if** (file != **null** && file.getParentFile() != **null**) {

*assertTrue*("Filter(File, String) " + filter.getClass().getName() + " not " + expected + " for " + file, (filter.accept(file.getParentFile(), file.getName()) == expected));

} **else** **if** (file == **null**) {

*assertTrue*("Filter(File, String) " + filter.getClass().getName() + " not " + expected + " for null", filter.accept(file) == expected);

}

}

**Padrão não sendo utilizado:**

**public** **void** DirectoryWalkerTestCaseJava4.testFilterAndLimitC() {

List results = **new** TestFileFinder(NOT\_SVN, 3).find(javaDir);

assertEquals("[C] Result Size", 4, results.size());

assertTrue("[C] Start Dir", results.contains(javaDir));

assertTrue("[C] Org Dir", results.contains(orgDir));

assertTrue("[C] Apache Dir", results.contains(apacheDir));

assertTrue("[C] Commons Dir", results.contains(commonsDir)

}

1. **Erro com estruturas de controle(Influência supersequência)**

**Padrão:**

**Usuários que chamam:**

junit.framework.Assert.assertTrue(java.lang.String, Z)

junit.framework.Assert.assertTrue(java.lang.String, Z)

junit.framework.Assert.assertTrue(java.lang.String, Z)

**Também chamam:**

java.io.File.exists()

**Suporte:** 0.00634 (7)

**Confiança:** 0.4666

**Método Minerado (Supersequência minerada):**

**public** **void** FileFilterTestCase.assertFiltering(IOFileFilter filter, File file, **boolean** expected) **throws** Exception {

// Note. This only tests the (File, String) version if the parent of

// the File passed in is not null

*assertTrue*("Filter(File) " + filter.getClass().getName() + " not " + expected + " for " + file, (filter.accept(file) == expected));

**if** (file != **null** && file.getParentFile() != **null**) {

*assertTrue*("Filter(File, String) " + filter.getClass().getName() + " not " + expected + " for " + file, (filter.accept(file.getParentFile(), file.getName()) == expected));

} **else** **if** (file == **null**) {

*assertTrue*("Filter(File, String) " + filter.getClass().getName() + " not " + expected + " for null", filter.accept(file) == expected);

}

}

**Padrão não sendo utilizado:**

**public** **void** DirectoryWalkerTestCaseJava4.testFilterAndLimitC() {

List results = **new** TestFileFinder(NOT\_SVN, 3).find(javaDir);

assertEquals("[C] Result Size", 4, results.size());

assertTrue("[C] Start Dir", results.contains(javaDir));

assertTrue("[C] Org Dir", results.contains(orgDir));

assertTrue("[C] Apache Dir", results.contains(apacheDir));

assertTrue("[C] Commons Dir", results.contains(commonsDir)

}

**Confiança Mínima: 70%**

1. **Automatizado sem estruturas de controle (Copy and paste?)**

**Padrão:**

**Usuários que chamam:**

junit.framework.Assert.assertEquals(java.lang.String, java.lang.String, java.lang.String)

junit.framework.Assert.fail(java.lang.String)

**Também chamam:**

junit.framework.Assert.assertEquals(java.lang.String, java.lang.String, java.lang.String)

**Suporte:** 0.00543(6)

**Confiança:** 1

**Método Minerado:**

**public** **void** DirectoryWalkerTestCase.testMultiThreadCancel() {

String cancelName = **null**;

TestMultiThreadCancelWalker walker = **null**;

// Cancel on a file

**try** {

cancelName = "DirectoryWalker.java";

walker = **new** TestMultiThreadCancelWalker(cancelName, **false**);

walker.find(*javaDir*);

*fail*("CancelException not thrown for '" + cancelName + "'");

} **catch** (DirectoryWalker.CancelException cancel) {

File last = (File) walker.results.get(walker.results.size() - 1);

*assertEquals*(cancelName, last.getName());

*assertEquals*("Depth: " + cancelName, 5, cancel.getDepth());

} **catch**(IOException ex) {

*fail*("IOException: " + cancelName + " " + ex);

}

// Cancel on a directory

**try** {

cancelName = "commons";

walker = **new** TestMultiThreadCancelWalker(cancelName, **false**);

walker.find(*javaDir*);

*fail*("CancelException not thrown for '" + cancelName + "'");

} **catch** (DirectoryWalker.CancelException cancel) {

*assertEquals*("File: " + cancelName, cancelName, cancel.getFile().getName());

*assertEquals*("Depth: " + cancelName, 3, cancel.getDepth());

} **catch**(IOException ex) {

*fail*("IOException: " + cancelName + " " + ex);

}

// Suppress CancelException (use same file name as preceeding test)

**try** {

walker = **new** TestMultiThreadCancelWalker(cancelName, **true**);

List results = walker.find(*javaDir*);

File lastFile = (File) results.get(results.size() - 1);

*assertEquals*("Suppress: " + cancelName, cancelName, lastFile.getName());

} **catch**(IOException ex) {

*fail*("Suppress threw " + ex);

}

}

**Padrão sendo utilizado:**

**public** **void** DirectoryWalkerTestCaseJava4.testCancel() {

String cancelName = **null**;

// Cancel on a file

**try** {

cancelName = "DirectoryWalker.java";

**new** TestCancelWalker(cancelName, **false**).find(javaDir);

fail("CancelException not thrown for '" + cancelName + "'");

} **catch** (DirectoryWalker.CancelException cancel) {

assertEquals("File: " + cancelName, cancelName, cancel.getFile().getName());

assertEquals("Depth: " + cancelName, 5, cancel.getDepth());

} **catch**(IOException ex) {

fail("IOException: " + cancelName + " " + ex);

}

// Cancel on a directory

**try** {

cancelName = "commons";

**new** TestCancelWalker(cancelName, **false**).find(javaDir);

fail("CancelException not thrown for '" + cancelName + "'");

} **catch** (DirectoryWalker.CancelException cancel) {

assertEquals("File: " + cancelName, cancelName, cancel.getFile().getName());

assertEquals("Depth: " + cancelName, 3, cancel.getDepth());

} **catch**(IOException ex) {

fail("IOException: " + cancelName + " " + ex);

}

// Suppress CancelException (use same file name as preceeding test)

**try** {

List results = **new** TestCancelWalker(cancelName, **true**).find(javaDir);

File lastFile = (File) results.get(results.size() - 1);

assertEquals("Suppress: " + cancelName, cancelName, lastFile.getName());

} **catch**(IOException ex) {

fail("Suppress threw " + ex);

}

}

1. **Automatizado com estruturas de controle (Influência supersequência)**

**Padrão:**

**Usuários que chamam:**

junit.framework.Assert.assertTrue(java.lang.String, Z)

junit.framework.Assert.assertTrue(java.lang.String, Z)

junit.framework.Assert.assertTrue(java.lang.String, Z)

**Também chamam:**

junit.framework.Assert.assertTrue(java.lang.String, Z)

**Suporte:** 0.00996 (11)

**Confiança:** 0.73333

**Método Minerado (Supersequência minerada):**

**public** **void** FileFilterTestCase.assertFiltering(IOFileFilter filter, File file, **boolean** expected) **throws** Exception {

// Note. This only tests the (File, String) version if the parent of

// the File passed in is not null

*assertTrue*("Filter(File) " + filter.getClass().getName() + " not " + expected + " for " + file, (filter.accept(file) == expected));

**if** (file != **null** && file.getParentFile() != **null**) {

*assertTrue*("Filter(File, String) " + filter.getClass().getName() + " not " + expected + " for " + file, (filter.accept(file.getParentFile(), file.getName()) == expected));

} **else** **if** (file == **null**) {

*assertTrue*("Filter(File, String) " + filter.getClass().getName() + " not " + expected + " for null", filter.accept(file) == expected);

}

}

**Padrão sendo utilizado:**

**public** **void** DirectoryWalkerTestCaseJava4.testFilterAndLimitC() {

List results = **new** TestFileFinder(*NOT\_SVN*, 3).find(*javaDir*);

*assertEquals*("[C] Result Size", 4, results.size());

*assertTrue*("[C] Start Dir", results.contains(*javaDir*));

*assertTrue*("[C] Org Dir", results.contains(*orgDir*));

*assertTrue*("[C] Apache Dir", results.contains(*apacheDir*));

*assertTrue*("[C] Commons Dir", results.contains(*commonsDir*));

}

1. **Automatizado sem estruturas de controle (Copy and Paste)**

**Padrão:**

**Usuários que chamam:**

junit.framework.Assert.assertEquals(java.lang.String, java.lang.String, java.lang.String)

junit.framework.Assert.fail(java.lang.String)

**Também chamam:**

junit.framework.Assert.assertEquals(java.lang.String, java.lang.String, java.lang.String)

**Suporte:** 0.00543(6)

**Confiança:** 1

**Método Minerado:**

**public** **void** DirectoryWalkerTestCase.testMultiThreadCancel() {

String cancelName = **null**;

TestMultiThreadCancelWalker walker = **null**;

// Cancel on a file

**try** {

cancelName = "DirectoryWalker.java";

walker = **new** TestMultiThreadCancelWalker(cancelName, **false**);

walker.find(*javaDir*);

*fail*("CancelException not thrown for '" + cancelName + "'");

} **catch** (DirectoryWalker.CancelException cancel) {

File last = (File) walker.results.get(walker.results.size() - 1);

*assertEquals*(cancelName, last.getName());

*assertEquals*("Depth: " + cancelName, 5, cancel.getDepth());

} **catch**(IOException ex) {

*fail*("IOException: " + cancelName + " " + ex);

}

// Cancel on a directory

**try** {

cancelName = "commons";

walker = **new** TestMultiThreadCancelWalker(cancelName, **false**);

walker.find(*javaDir*);

*fail*("CancelException not thrown for '" + cancelName + "'");

} **catch** (DirectoryWalker.CancelException cancel) {

*assertEquals*("File: " + cancelName, cancelName, cancel.getFile().getName());

*assertEquals*("Depth: " + cancelName, 3, cancel.getDepth());

} **catch**(IOException ex) {

*fail*("IOException: " + cancelName + " " + ex);

}

// Suppress CancelException (use same file name as preceeding test)

**try** {

walker = **new** TestMultiThreadCancelWalker(cancelName, **true**);

List results = walker.find(*javaDir*);

File lastFile = (File) results.get(results.size() - 1);

*assertEquals*("Suppress: " + cancelName, cancelName, lastFile.getName());

} **catch**(IOException ex) {

*fail*("Suppress threw " + ex);

}

}

**Padrão sendo utilizado:**

**public** **void** DirectoryWalkerTestCaseJava4.testMultiThreadCancel() {

String cancelName = **null**;

TestMultiThreadCancelWalker walker = **null**;

// Cancel on a file

**try** {

cancelName = "DirectoryWalker.java";

walker = **new** TestMultiThreadCancelWalker(cancelName, **false**);

walker.find(*javaDir*);

*fail*("CancelException not thrown for '" + cancelName + "'");

} **catch** (DirectoryWalker.CancelException cancel) {

File last = (File) walker.results.get(walker.results.size() - 1);

*assertEquals*(cancelName, last.getName());

*assertEquals*("Depth: " + cancelName, 5, cancel.getDepth());

} **catch**(IOException ex) {

*fail*("IOException: " + cancelName + " " + ex);

}

// Cancel on a directory

**try** {

cancelName = "commons";

walker = **new** TestMultiThreadCancelWalker(cancelName, **false**);

walker.find(*javaDir*);

*fail*("CancelException not thrown for '" + cancelName + "'");

} **catch** (DirectoryWalker.CancelException cancel) {

*assertEquals*("File: " + cancelName, cancelName, cancel.getFile().getName());

*assertEquals*("Depth: " + cancelName, 3, cancel.getDepth());

} **catch**(IOException ex) {

*fail*("IOException: " + cancelName + " " + ex);

}

// Suppress CancelException (use same file name as preceeding test)

**try** {

walker = **new** TestMultiThreadCancelWalker(cancelName, **true**);

List results = walker.find(*javaDir*);

File lastFile = (File) results.get(results.size() - 1);

*assertEquals*("Suppress: " + cancelName, cancelName, lastFile.getName());

} **catch**(IOException ex) {

*fail*("Suppress threw " + ex);

}

}

1. **Automatizado com estruturas de controle (Influência supersequência)**

**Padrão:**

**Usuários que chamam:**

junit.framework.Assert.assertTrue(java.lang.String, Z)

junit.framework.Assert.assertTrue(java.lang.String, Z)

junit.framework.Assert.assertTrue(java.lang.String, Z)

**Também chamam:**

junit.framework.Assert.assertTrue(java.lang.String, Z)

**Suporte:** 0.00996 (11)

**Confiança:** 0.73333

**Método Minerado (Supersequência minerada):**

**public** **void** FileFilterTestCase.assertFiltering(IOFileFilter filter, File file, **boolean** expected) **throws** Exception {

// Note. This only tests the (File, String) version if the parent of

// the File passed in is not null

*assertTrue*("Filter(File) " + filter.getClass().getName() + " not " + expected + " for " + file, (filter.accept(file) == expected));

**if** (file != **null** && file.getParentFile() != **null**) {

*assertTrue*("Filter(File, String) " + filter.getClass().getName() + " not " + expected + " for " + file, (filter.accept(file.getParentFile(), file.getName()) == expected));

} **else** **if** (file == **null**) {

*assertTrue*("Filter(File, String) " + filter.getClass().getName() + " not " + expected + " for null", filter.accept(file) == expected);

}

}

**Padrão sendo utilizado:**

/\*\* Test case sensitivity \*/

**public** **void** ExtensionFileComparatorTest.testCaseSensitivity() {

File file3 = **new** File("abc.FOO");

Comparator<File> sensitive = **new** ExtensionFileComparator(**null**); /\* test null as well \*/

assertTrue("sensitive file1 & file2 = 0", sensitive.compare(equalFile1, equalFile2) == 0);

assertTrue("sensitive file1 & file3 > 0", sensitive.compare(equalFile1, file3) > 0);

assertTrue("sensitive file1 & less > 0", sensitive.compare(equalFile1, lessFile) > 0);

Comparator<File> insensitive = ExtensionFileComparator.EXTENSION\_INSENSITIVE\_COMPARATOR;

assertTrue("insensitive file1 & file2 = 0", insensitive.compare(equalFile1, equalFile2) == 0);

assertTrue("insensitive file1 & file3 = 0", insensitive.compare(equalFile1, file3) == 0);

assertTrue("insensitive file1 & file4 > 0", insensitive.compare(equalFile1, lessFile) > 0);

assertTrue("insensitive file3 & less > 0", insensitive.compare(file3, lessFile) > 0);

}

1. **Automatizado com estruturas de controle (Influência supersequência)**

**Padrão:**

**Usuários que chamam:**

junit.framework.Assert.assertTrue(java.lang.String, Z)

junit.framework.Assert.assertTrue(java.lang.String, Z)

junit.framework.Assert.assertTrue(java.lang.String, Z)

**Também chamam:**

junit.framework.Assert.assertTrue(java.lang.String, Z)

**Suporte:** 0.00996 (11)

**Confiança:** 0.73333

**Padrão sendo utilizado:**

/\*\* Test case sensitivity \*/

**public** **void** NameFileComparatorTest.testCaseSensitivity() {

File file3 = **new** File("a/FOO.txt");

Comparator<File> sensitive = **new** NameFileComparator(**null**); /\* test null as well \*/

assertTrue("sensitive file1 & file2 = 0", sensitive.compare(equalFile1, equalFile2) == 0);

assertTrue("sensitive file1 & file3 > 0", sensitive.compare(equalFile1, file3) > 0);

assertTrue("sensitive file1 & less > 0", sensitive.compare(equalFile1, lessFile) > 0);

Comparator<File> insensitive = NameFileComparator.NAME\_INSENSITIVE\_COMPARATOR;

assertTrue("insensitive file1 & file2 = 0", insensitive.compare(equalFile1, equalFile2) == 0);

assertTrue("insensitive file1 & file3 = 0", insensitive.compare(equalFile1, file3) == 0);

assertTrue("insensitive file1 & file4 > 0", insensitive.compare(equalFile1, lessFile) > 0);

assertTrue("insensitive file3 & less > 0", insensitive.compare(file3, lessFile) > 0);

}

1. **Automatizado com estruturas de controle (Influência supersequência)**

**Padrão:**

**Usuários que chamam:**

junit.framework.Assert.assertTrue(java.lang.String, Z)

junit.framework.Assert.assertTrue(java.lang.String, Z)

junit.framework.Assert.assertTrue(java.lang.String, Z)

**Também chamam:**

junit.framework.Assert.assertTrue(java.lang.String, Z)

**Suporte:** 0.00996 (11)

**Confiança:** 0.73333

**Padrão sendo utilizado:**

/\*\* Test case sensitivity \*/

**public** **void** PathFileComparatorTest.testCaseSensitivity() {

File file3 = **new** File("FOO/file.txt");

Comparator<File> sensitive = **new** PathFileComparator(**null**); /\* test null as well \*/

assertTrue("sensitive file1 & file2 = 0", sensitive.compare(equalFile1, equalFile2) == 0);

assertTrue("sensitive file1 & file3 > 0", sensitive.compare(equalFile1, file3) > 0);

assertTrue("sensitive file1 & less > 0", sensitive.compare(equalFile1, lessFile) > 0);

Comparator<File> insensitive = PathFileComparator.PATH\_INSENSITIVE\_COMPARATOR;

assertTrue("insensitive file1 & file2 = 0", insensitive.compare(equalFile1, equalFile2) == 0);

assertTrue("insensitive file1 & file3 = 0", insensitive.compare(equalFile1, file3) == 0);

assertTrue("insensitive file1 & file4 > 0", insensitive.compare(equalFile1, lessFile) > 0);

assertTrue("insensitive file3 & less > 0", insensitive.compare(file3, lessFile) > 0);

}

1. **Erro sem estruturas de controle (Uso normal)**

**Padrão**

**Usuários que chamam:**

junit.framework.Assert.fail(java.lang.String)

junit.framework.Assert.assertEquals(java.lang.String, java.lang.String, java.lang.String)

junit.framework.Assert.fail(java.lang.String)

**Também chamam:**

junit.framework.Assert.assertEquals(java.lang.String, java.lang.String, java.lang.String)

**Suporte:** 0.00543(6)

**Confiança:** 1

**Método Minerado:**

**public** **void** DirectoryWalkerTestCase.testMultiThreadCancel() {

String cancelName = **null**;

TestMultiThreadCancelWalker walker = **null**;

// Cancel on a file

**try** {

cancelName = "DirectoryWalker.java";

walker = **new** TestMultiThreadCancelWalker(cancelName, **false**);

walker.find(*javaDir*);

*fail*("CancelException not thrown for '" + cancelName + "'");

} **catch** (DirectoryWalker.CancelException cancel) {

File last = (File) walker.results.get(walker.results.size() - 1);

*assertEquals*(cancelName, last.getName());

*assertEquals*("Depth: " + cancelName, 5, cancel.getDepth());

} **catch**(IOException ex) {

*fail*("IOException: " + cancelName + " " + ex);

}

// Cancel on a directory

**try** {

cancelName = "commons";

walker = **new** TestMultiThreadCancelWalker(cancelName, **false**);

walker.find(*javaDir*);

*fail*("CancelException not thrown for '" + cancelName + "'");

} **catch** (DirectoryWalker.CancelException cancel) {

*assertEquals*("File: " + cancelName, cancelName, cancel.getFile().getName());

*assertEquals*("Depth: " + cancelName, 3, cancel.getDepth());

} **catch**(IOException ex) {

*fail*("IOException: " + cancelName + " " + ex);

}

// Suppress CancelException (use same file name as preceeding test)

**try** {

walker = **new** TestMultiThreadCancelWalker(cancelName, **true**);

List results = walker.find(*javaDir*);

File lastFile = (File) results.get(results.size() - 1);

*assertEquals*("Suppress: " + cancelName, cancelName, lastFile.getName());

} **catch**(IOException ex) {

*fail*("Suppress threw " + ex);

}

}

**Padrão não sendo utilizado:**

**private** **void** LineIteratorTestCase.testFiltering(List<String> lines, Reader reader) {

LineIterator iterator = **new** LineIterator(reader) {

@Override

**protected** **boolean** isValidLine(String line) {

**char** c = line.charAt(line.length() - 1);

**return** ((c - 48) % 3 != 1);

}

};

**try** {

**try** {

iterator.remove();

fail("Remove is unsupported");

} **catch** (UnsupportedOperationException ex) {

// expected

}

**int** idx = 0;

**int** actualLines = 0;

**while** (iterator.hasNext()) {

String line = iterator.next();

actualLines++;

assertEquals("Comparing line " + idx, lines.get(idx), line);

assertTrue("Exceeded expected idx=" + idx + " size=" + lines.size(), idx < lines.size());

idx++;

**if** (idx % 3 == 1) {

idx++;

}

}

assertEquals("Line Count doesn't match", 9, lines.size());

assertEquals("Line Count doesn't match", 9, idx);

assertEquals("Line Count doesn't match", 6, actualLines);

// try calling next() after file processed

**try** {

iterator.next();

fail("Expected NoSuchElementException");

} **catch** (NoSuchElementException expected) {

// ignore, expected result

}

**try** {

iterator.nextLine();

fail("Expected NoSuchElementException");

} **catch** (NoSuchElementException expected) {

// ignore, expected result

}

} **finally** {

LineIterator.closeQuietly(iterator);

}

}

**Confiança Mínima: 90%**

1. **Automatizado com estruturas de controle (Influência supersequência)**

**Padrão:**

**Usuários que chamam:**

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

**Também chamam:**

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

**Suporte:** 0.013587 (15)

**Confiança:** 0.9375

**Método Minerado (Supersequência minerada):**

**public** **void** IOUtilsTestCase.testConstants() **throws** Exception {

*assertEquals*('/', IOUtils.*DIR\_SEPARATOR\_UNIX*);

*assertEquals*('\\', IOUtils.*DIR\_SEPARATOR\_WINDOWS*);

*assertEquals*("\n", IOUtils.*LINE\_SEPARATOR\_UNIX*);

*assertEquals*("\r\n", IOUtils.*LINE\_SEPARATOR\_WINDOWS*);

**if** (*WINDOWS*) {

*assertEquals*('\\', IOUtils.*DIR\_SEPARATOR*);

*assertEquals*("\r\n", IOUtils.*LINE\_SEPARATOR*);

} **else** {

*assertEquals*('/', IOUtils.*DIR\_SEPARATOR*);

*assertEquals*("\n", IOUtils.*LINE\_SEPARATOR*);

}

}

**Padrão sendo utilizado:**

**public** **void** FileUtilsTestCase.testDecodeUrl() {

assertEquals("", FileUtils.decodeUrl(""));

assertEquals("foo", FileUtils.decodeUrl("foo"));

assertEquals("+", FileUtils.decodeUrl("+"));

assertEquals("% ", FileUtils.decodeUrl("%25%20"));

assertEquals("%20", FileUtils.decodeUrl("%2520"));

assertEquals("jar:file:/C:/dir/sub dir/1.0/foo-1.0.jar!/org/Bar.class", FileUtils

.decodeUrl("jar:file:/C:/dir/sub%20dir/1.0/foo-1.0.jar!/org/Bar.class"));

}

1. **Automatizado com estruturas de controle (Influência supersequência)**

**Padrão:**

**Usuários que chamam:**

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

**Também chamam:**

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

**Suporte:** 0.013587 (15)

**Confiança:** 0.9375

**Padrão sendo utilizado:**

**public** **void** FileUtilsTestCase. testDecodeUrlLenient() {

assertEquals(" ", FileUtils.decodeUrl(" "));

assertEquals("\u00E4\u00F6\u00FC\u00DF", FileUtils.decodeUrl("\u00E4\u00F6\u00FC\u00DF"));

assertEquals("%", FileUtils.decodeUrl("%"));

assertEquals("% ", FileUtils.decodeUrl("%%20"));

assertEquals("%2", FileUtils.decodeUrl("%2"));

assertEquals("%2G", FileUtils.decodeUrl("%2G"));

}

1. **Automatizado com estruturas de controle (Influência supersequência)**

**Padrão:**

**Usuários que chamam:**

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

**Também chamam:**

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

**Suporte:** 0.013587 (15)

**Confiança:** 0.9375

**Padrão sendo utilizado:**

**public** **void** FilenameUtilsTestCase.testGetFullPathNoEndSeparator\_IO\_248() {

// Test single separator

assertEquals("/", FilenameUtils.getFullPathNoEndSeparator("/"));

assertEquals("\\", FilenameUtils.getFullPathNoEndSeparator("\\"));

// Test one level directory

assertEquals("/", FilenameUtils.getFullPathNoEndSeparator("/abc"));

assertEquals("\\", FilenameUtils.getFullPathNoEndSeparator("\\abc"));

// Test one level directory

assertEquals("/abc", FilenameUtils.getFullPathNoEndSeparator("/abc/xyz"));

assertEquals("\\abc", FilenameUtils.getFullPathNoEndSeparator("\\abc\\xyz"));

}

1. **Automatizado com estruturas de controle (Influência supersequência)**

**Padrão:**

**Usuários que chamam:**

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

**Também chamam:**

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

**Suporte:** 0.013587 (15)

**Confiança:** 0.9375

**Padrão sendo utilizado:**

**protected** **void** XmlStreamReaderTest.\_testRawNoBomValid(String encoding) **throws** Exception {

InputStream is = getXmlStream("no-bom", XML1, encoding, encoding);

XmlStreamReader xmlReader = **new** XmlStreamReader(is, **false**);

assertEquals(xmlReader.getEncoding(), "UTF-8");

is = getXmlStream("no-bom", XML2, encoding, encoding);

xmlReader = **new** XmlStreamReader(is);

assertEquals(xmlReader.getEncoding(), "UTF-8");

is = getXmlStream("no-bom", XML3, encoding, encoding);

xmlReader = **new** XmlStreamReader(is);

assertEquals(xmlReader.getEncoding(), encoding);

is = getXmlStream("no-bom", XML4, encoding, encoding);

xmlReader = **new** XmlStreamReader(is);

assertEquals(xmlReader.getEncoding(), encoding);

is = getXmlStream("no-bom", XML5, encoding, encoding);

xmlReader = **new** XmlStreamReader(is);

assertEquals(xmlReader.getEncoding(), encoding);

}

1. **Automatizado com estruturas de controle (Influência supersequência)**

**Padrão:**

**Usuários que chamam:**

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

**Também chamam:**

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

**Suporte:** 0.013587 (15)

**Confiança:** 0.9375

**Padrão sendo utilizado:**

**public** **void** FileUtilsTestCase.testByteCountToDisplaySizeLong() {

assertEquals(FileUtils.byteCountToDisplaySize(0), "0 bytes");

assertEquals(FileUtils.byteCountToDisplaySize(1), "1 bytes");

assertEquals(FileUtils.byteCountToDisplaySize(1023), "1023 bytes");

assertEquals(FileUtils.byteCountToDisplaySize(1024), "1 KB");

assertEquals(FileUtils.byteCountToDisplaySize(1025), "1 KB");

assertEquals(FileUtils.byteCountToDisplaySize(1024 \* 1023), "1023 KB");

assertEquals(FileUtils.byteCountToDisplaySize(1024 \* 1024), "1 MB");

assertEquals(FileUtils.byteCountToDisplaySize(1024 \* 1025), "1 MB");

assertEquals(FileUtils.byteCountToDisplaySize(1024 \* 1024 \* 1023), "1023 MB");

assertEquals(FileUtils.byteCountToDisplaySize(1024 \* 1024 \* 1024), "1 GB");

assertEquals(FileUtils.byteCountToDisplaySize(1024 \* 1024 \* 1025), "1 GB");

assertEquals(FileUtils.byteCountToDisplaySize(1024L \* 1024 \* 1024 \* 2), "2 GB");

assertEquals(FileUtils.byteCountToDisplaySize(1024 \* 1024 \* 1024 \* 2 - 1), "1 GB");

assertEquals(FileUtils.byteCountToDisplaySize(1024L \* 1024 \* 1024 \* 1024), "1 TB");

assertEquals(FileUtils.byteCountToDisplaySize(1024L \* 1024 \* 1024 \* 1024 \* 1024), "1 PB");

assertEquals(FileUtils.byteCountToDisplaySize(1024L \* 1024 \* 1024 \* 1024 \* 1024 \* 1024), "1 EB");

assertEquals(FileUtils.byteCountToDisplaySize(Long.MAX\_VALUE), "7 EB");

// Other MAX\_VALUEs

assertEquals(FileUtils.byteCountToDisplaySize(Character.MAX\_VALUE), "63 KB");

assertEquals(FileUtils.byteCountToDisplaySize(Short.MAX\_VALUE), "31 KB");

assertEquals(FileUtils.byteCountToDisplaySize(Integer.MAX\_VALUE), "1 GB");

}

1. **Automatizado com estruturas de controle (Influência supersequência)**

**Padrão:**

**Usuários que chamam:**

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

**Também chamam:**

junit.framework.Assert.assertEquals(java.lang.String,java.lang.String)

**Suporte:** 0.013587 (15)

**Confiança:** 0.9375

**Padrão sendo utilizado:**

**public** **void** FileUtilsTestCase.testByteCountToDisplaySizeBigInteger() {

**final** BigInteger b1023 = BigInteger.valueOf(1023);

**final** BigInteger b1025 = BigInteger.valueOf(1025);

**final** BigInteger KB1 = BigInteger.valueOf(1024);

**final** BigInteger MB1 = KB1.multiply(KB1);

**final** BigInteger GB1 = MB1.multiply(KB1);

**final** BigInteger GB2 = GB1.add(GB1);

**final** BigInteger TB1 = GB1.multiply(KB1);

**final** BigInteger PB1 = TB1.multiply(KB1);

**final** BigInteger EB1 = PB1.multiply(KB1);

assertEquals(FileUtils.byteCountToDisplaySize(BigInteger.ZERO), "0 bytes");

assertEquals(FileUtils.byteCountToDisplaySize(BigInteger.ONE), "1 bytes");

assertEquals(FileUtils.byteCountToDisplaySize(b1023), "1023 bytes");

assertEquals(FileUtils.byteCountToDisplaySize(KB1), "1 KB");

assertEquals(FileUtils.byteCountToDisplaySize(b1025), "1 KB");

assertEquals(FileUtils.byteCountToDisplaySize(MB1.subtract(BigInteger.ONE)), "1023 KB");

assertEquals(FileUtils.byteCountToDisplaySize(MB1), "1 MB");

assertEquals(FileUtils.byteCountToDisplaySize(MB1.add(BigInteger.ONE)), "1 MB");

assertEquals(FileUtils.byteCountToDisplaySize(GB1.subtract(BigInteger.ONE)), "1023 MB");

assertEquals(FileUtils.byteCountToDisplaySize(GB1), "1 GB");

assertEquals(FileUtils.byteCountToDisplaySize(GB1.add(BigInteger.ONE)), "1 GB");

assertEquals(FileUtils.byteCountToDisplaySize(GB2), "2 GB");

assertEquals(FileUtils.byteCountToDisplaySize(GB2.subtract(BigInteger.ONE)), "1 GB");

assertEquals(FileUtils.byteCountToDisplaySize(TB1), "1 TB");

assertEquals(FileUtils.byteCountToDisplaySize(PB1), "1 PB");

assertEquals(FileUtils.byteCountToDisplaySize(EB1), "1 EB");

assertEquals(FileUtils.byteCountToDisplaySize(Long.MAX\_VALUE), "7 EB");

// Other MAX\_VALUEs

assertEquals(FileUtils.byteCountToDisplaySize(BigInteger.valueOf(Character.MAX\_VALUE)), "63 KB");

assertEquals(FileUtils.byteCountToDisplaySize(BigInteger.valueOf(Short.MAX\_VALUE)), "31 KB");

assertEquals(FileUtils.byteCountToDisplaySize(BigInteger.valueOf(Integer.MAX\_VALUE)), "1 GB");

}