Stirling-PDF
Final Report

PREPARED FOR

Course: COE891 - Software Testing and Quality Assurance

Prof: Dr. Reza Samavi

TA: Leandra Budau

PREPARED BY

Section 7, Group 2

Victor Do - 501137174

Alaa Yafaoui- 501052383

Benjamin Fredette - 500947897

Wasay Adil - 501112339

Gaurav Divecha - 501034331

Introduction

Background Information

Stirling-PDF is an open-source Java-based web application providing a comprehensive PDF manipulation operation suite. The project implements over 50 different PDF operations, including file splitting, merging, conversion, security management, and document enhancement features. Built with a focus on security and privacy, it processes files locally and implements temporary storage mechanisms to ensure data privacy. The application features a modern web interface with dark mode support, parallel processing capabilities, and an API for external integration. Notable technical aspects include integration with LibreOffice for conversions, Tesseract for OCR capabilities, and PDF.js with Joxit for viewing and editing functions. The project demonstrates enterprise-level architecture with features like SSO support, authentication systems, and database management, making it an ideal candidate for comprehensive software testing across multiple functional areas.

Document Information

This is the final report covering all the testing that was completed on the StirlingPDF application. Included with this report submission is a zip file containing a Gradle Java project containing all the classes that were tested, the test class files, and a build.gradle file that helps to quickly install all the dependencies. The submitted Gradle Java project does not contain the entire application for reasons of brevity and file size constraint. Therefore, it cannot be built into a working application. However, all our tests have been validated to work on the full application as well as in our submission project, which contains only the bare minimum classes.

References

Github: https://github.com/Stirling-Tools/Stirling-PDF?tab=readme-ov-file
Docker: https://hub.docker.com/r/stirlingtools/stirling-pdf
Homepage: https://stirlingpdf.com
All documentation is available at: https://docs.stirlingpdf.com/
Our repo for testing: https://github.com/Wasay76/Stirling-PDF/tree/main

Findings and Results of Testing

Input Space Partitioning (ISP)

Testing tools:

- Pitest
- Junit5
- Gradle
- Springframework
- Java 17 or above
- VSCode

Tested classes:

- ‘stirling.software.SPDF.utils.ErrorUtils'
- 'stirling.software.SPDF.utils.FileInfo'
- 'stirling.software.SPDF.utils.PropertyConfig.java'
- 'stirling.software.SPDF.config.InstallationPathConfig.java'
- 'stirling.software.SPDF.utils.RequesturiUtils.java’

Example 1:

For the InstallationPathConfig class, our ISP testing focused on ensuring that the computed installation paths meet the expected boundaries when operating under default conditions. In our tests, we assumed no system property for "STIRLING_PDF_DESKTOP_UI" was set, so the BASE_PATH defaults to "." plus the file separator. We then applied Boundary Value Analysis by verifying that methods like getLogPath and getSettingsPath correctly concatenate the default BASE_PATH with their respective fixed subpaths ("logs", "configs", etc.) and the proper file separators. This approach effectively partitions the input space by testing critical boundary conditions—specifically, that the string construction behaves as expected in the most basic configuration. This coverage gives us confidence that, regardless of future changes or extensions, the InstallationPathConfig will generate valid paths within the predefined framework.

InstallationPathConfigTest

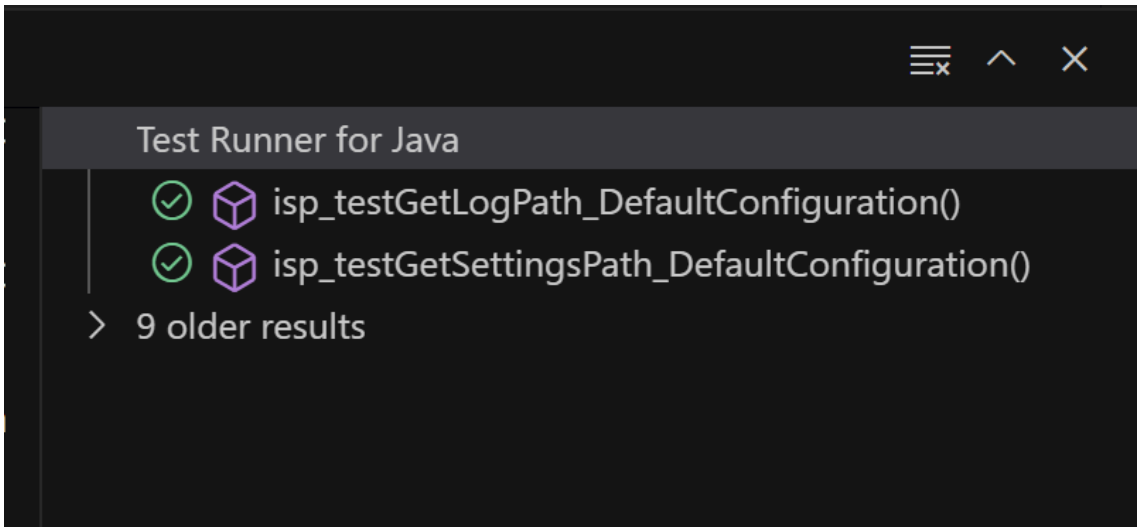
all > stirling.software.SPDF.config > InstallationPathConfigTest

2	0	0	0.494s
tests	failures	ignored	duration

100%
successful

Tests | Standard output

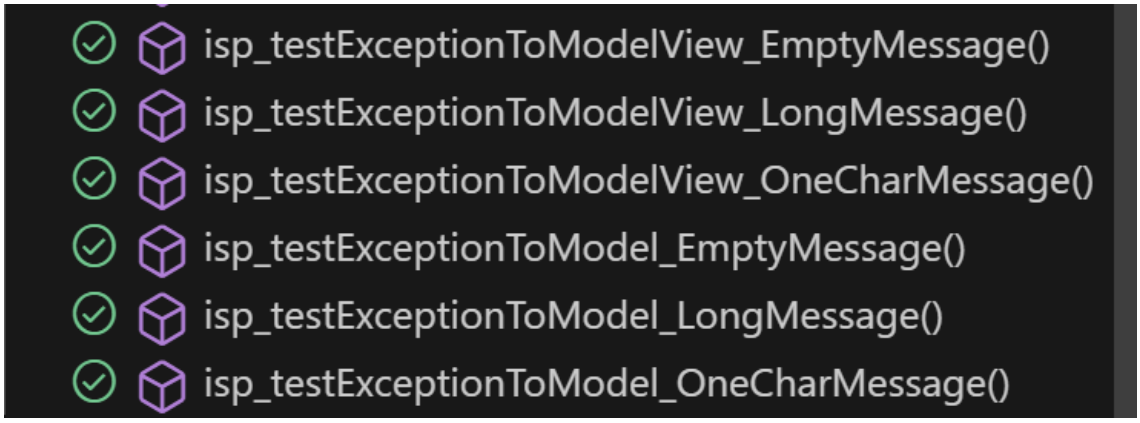
Test	Duration	Result
isp_testGetLogPath_DefaultConfiguration()	0.493s	passed
isp_testGetSettingsPath_DefaultConfiguration()	0.001s	passed



```java
    @Test
    public void isp_testGetLogPath_DefaultConfiguration() {
        // With no STIRLING_PDF_DESKTOP_UI property set,
        // the default BASE_PATH is: "." + File.separator.
        // Therefore, LOG_PATH is computed as BASE_PATH + "logs" + File.separator.
        String expected = "." + File.separator + "logs" + File.separator;
        assertEquals(expected, InstallationPathConfig.getLogPath(),
                message:"Default log path should match expected boundary value when using default BASE_PATH.");
    }

    @Test
    public void isp_testGetSettingsPath_DefaultConfiguration() {
        // With the default configuration:
        // BASE_PATH = "." + File.separator;
        // CONFIG_PATH = BASE_PATH + "configs" + File.separator, so expected CONFIG_PATH is:
        // "." + File.separator + "configs" + File.separator.
        // Then, SETTINGS_PATH = CONFIG_PATH + "settings.yml".
        String expected = "." + File.separator + "configs" + File.separator + "settings.yml";
        assertEquals(expected, InstallationPathConfig.getSettingsPath(),
                message:"Default settings path should match expected boundary value when using default BASE_PATH.");
    }
}

```

Example 2:

In these ISP tests for ErrorUtils, we applied Boundary Value Analysis to ensure that the error-handling methods gracefully handle the full spectrum of exception message lengths. We verified the behavior of both exceptionToModel and exceptionToModelView by testing three critical boundary conditions: an empty string message (to see that it returns an empty error message while still generating a valid stack trace), a one-character message (to confirm that minimal input is captured correctly in both the error message and the resulting stack trace), and an extremely long message (to ensure that the method can fully handle and return large inputs without truncation). This comprehensive testing approach ensures that the utility methods remain robust and predictable, even when the exception messages are at their boundary extremes, thereby enhancing the reliability of error reporting in the application.



ErrorUtilsTest

all > stirling.software.SPDF.utils > ErrorUtilsTest

13	0	0	0.024s
tests	failures	ignored	duration

100% successful

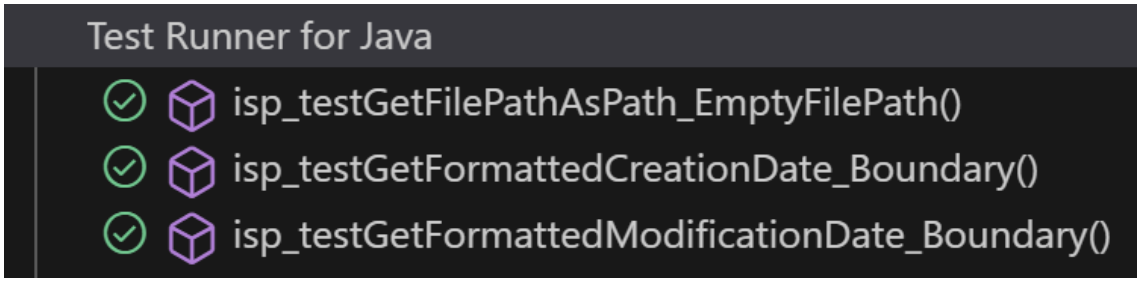
Tests

Test	Duration	Result
isp_testExceptionToModelView_EmptyMessage()	0.002s	passed
isp_testExceptionToModelView_LongMessage()	0.002s	passed
isp_testExceptionToModelView_OneCharMessage()	0.001s	passed
isp_testExceptionToModel_EmptyMessage()	0.001s	passed
isp_testExceptionToModel_LongMessage()	0.001s	passed
isp_testExceptionToModel_OneCharMessage()	0.001s	passed

```java
    // ============================================================
    // ISP TESTING: Additional tests using Boundary Value Analysis (BVA)
    // ============================================================

    @Test
    public void isp_testExceptionToModel_EmptyMessage() {
        // Boundary: Exception with an empty string as message
        Model model = new ConcurrentModel();
        Exception ex = new Exception(message:"");
        Model resultModel = ErrorUtils.exceptionToModel(model, ex);
        assertNotNull(resultModel);
        assertEquals(expected:"", resultModel.getAttribute(attributeName:"errorMessage"), message:"Empty message should return empty string");
        assertNotNull(resultModel.getAttribute(attributeName:"stackTrace"));
    }

    @Test
    public void isp_testExceptionToModel_OneCharMessage() {
        // Boundary: Exception with a one-character message
        Model model = new ConcurrentModel();
        Exception ex = new Exception(message:"A");
        Model resultModel = ErrorUtils.exceptionToModel(model, ex);
        assertNotNull(resultModel);
        assertEquals(expected:"A", resultModel.getAttribute(attributeName:"errorMessage"), message:"One-char message should be returned correctly");
        assertNotNull(resultModel.getAttribute(attributeName:"stackTrace"));
        assertTrue(((String) resultModel.getAttribute(attributeName:"stackTrace")).contains(s:"A"), message:"Stack trace should contain the one-char message");
    }

    @Test
    public void isp_testExceptionToModel_LongMessage() {
        // Boundary: Exception with an extremely long message
        Model model = new ConcurrentModel();
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < 1000; i++) {
            sb.append(str:"X");
        }
        String longMessage = sb.toString();
        Exception ex = new Exception(longMessage);
        Model resultModel = ErrorUtils.exceptionToModel(model, ex);
        assertNotNull(resultModel);
        assertEquals(longMessage, resultModel.getAttribute(attributeName:"errorMessage"), message:"Long message should be returned fully");
        assertNotNull(resultModel.getAttribute(attributeName:"stackTrace"));
    }

    @Test
    public void isp_testExceptionToModelView_EmptyMessage() {
        // Boundary: Using ModelAndView with an exception that has an empty message
        Model model = new ConcurrentModel();
        Exception ex = new Exception(message:"");
        ModelAndView resultModelAndView = ErrorUtils.exceptionToModelView(model, ex);
        assertNotNull(resultModelAndView);
        assertEquals(expected:"", resultModelAndView.getModel().get(key:"errorMessage"), message:"Empty message should return empty string");
        assertNotNull(resultModelAndView.getModel().get(key:"stackTrace"));
    }

    @Test
    public void isp_testExceptionToModelView_OneCharMessage() {
        // Boundary: Using ModelAndView with an exception that has a one-character message
        Model model = new ConcurrentModel();
        Exception ex = new Exception(message:"B");
        ModelAndView resultModelAndView = ErrorUtils.exceptionToModelView(model, ex);
        assertNotNull(resultModelAndView);
        assertEquals(expected:"B", resultModelAndView.getModel().get(key:"errorMessage"), message:"One-char message should be returned correctly");
        assertNotNull(resultModelAndView.getModel().get(key:"stackTrace"));
        assertTrue(((String) resultModelAndView.getModel().get(key:"stackTrace")).contains(s:"B"), message:"Stack trace should contain the one-char message");
    }

    @Test
    public void isp_testExceptionToModelView_LongMessage() {
        // Boundary: Using ModelAndView with an exception that has an extremely long message
        Model model = new ConcurrentModel();
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < 1000; i++) {
            sb.append(str:"Y");
        }
        String longMessage = sb.toString();
        Exception ex = new Exception(longMessage);
        ModelAndView resultModelAndView = ErrorUtils.exceptionToModelView(model, ex);
        assertNotNull(resultModelAndView);
        assertEquals(longMessage, resultModelAndView.getModel().get(key:"errorMessage"), message:"Long message should be returned fully");
        assertNotNull(resultModelAndView.getModel().get(key:"stackTrace"));
    }
}
```

Other Tests:

FileInfoTest.java



FileInfoTest

all > stirling.software.SPDF.utils > FileInfoTest

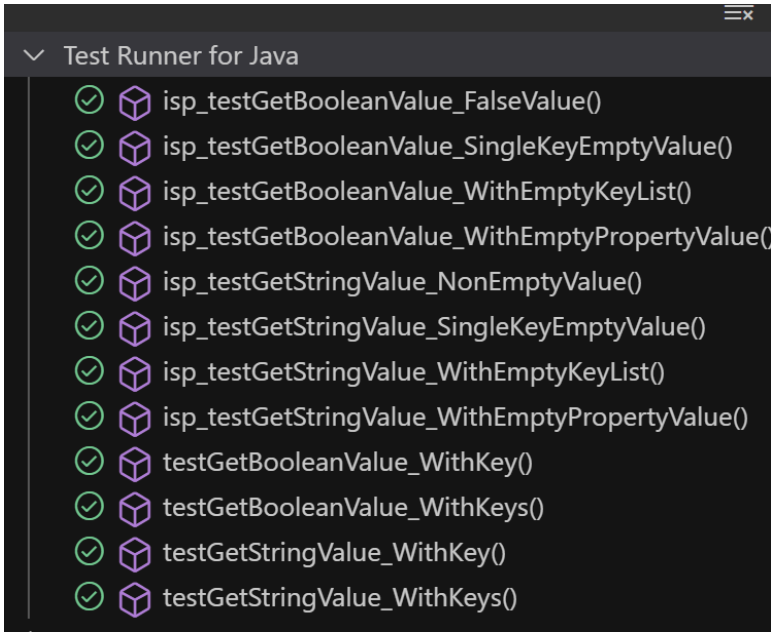
10 tests	0 failures	0 ignored	0.048s duration

100% successful

Tests

Test	Method name	Duration	Result
isp_testGetFilePathAsPath_EmptyFilePath()	isp_testGetFilePathAsPath_EmptyFilePath()	0.001s	passed
isp_testGetFormattedCreationDate_Boundary()	isp_testGetFormattedCreationDate_Boundary()	0.001s	passed
isp_testGetFormattedModificationDate_Boundary()	isp_testGetFormattedModificationDate_Boundary()	0.002s	passed

PropertConfigTest.java



PropertyConfigsTest

all > stirling.software.SPDF.utils > PropertyConfigsTest

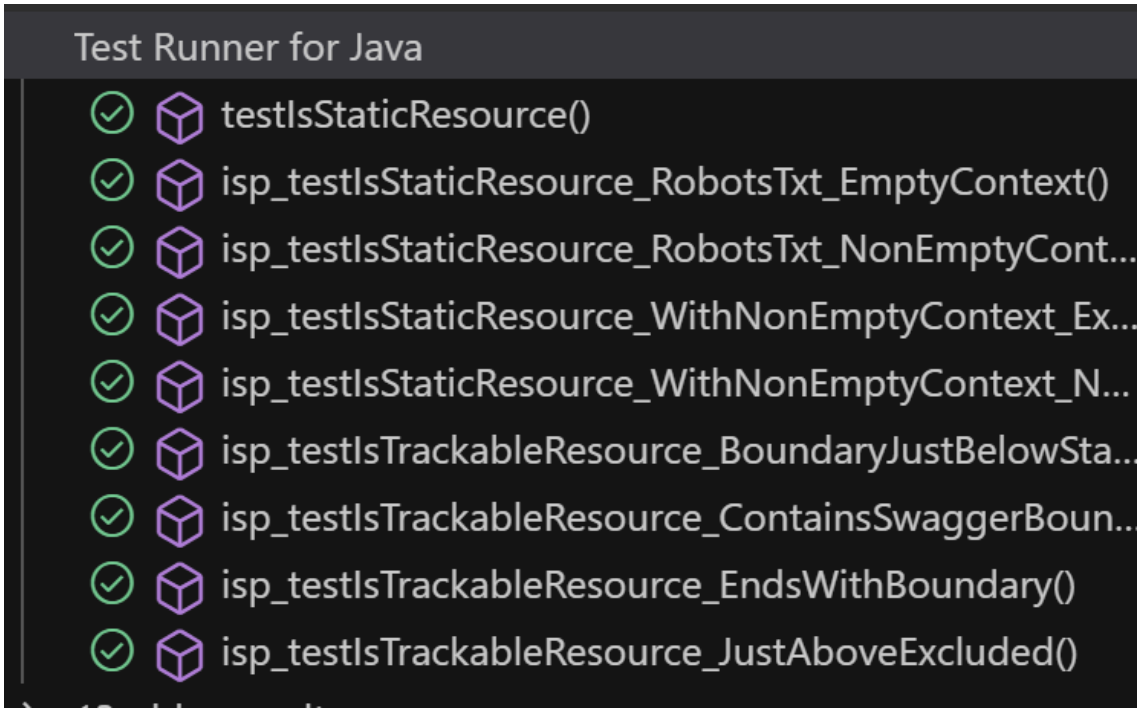
12	0	0	0.015s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
isp_testGetBooleanValue_FalseValue()	0.001s	passed
isp_testGetBooleanValue_SingleKeyEmptyValue()	0.001s	passed
isp_testGetBooleanValue_WithEmptyKeyList()	0.001s	passed
isp_testGetBooleanValue_WithEmptyPropertyValue()	0.001s	passed
isp_testGetStringValue_NonEmptyValue()	0.003s	passed
isp_testGetStringValue_SingleKeyEmptyValue()	0.001s	passed
isp_testGetStringValue_WithEmptyKeyList()	0s	passed
isp_testGetStringValue_WithEmptyPropertyValue()	0.002s	passed
testGetBooleanValue_WithKey()	0.002s	passed
testGetBooleanValue_WithKeys()	0.001s	passed
testGetStringValue_WithKey()	0.001s	passed
testGetStringValue_WithKeys()	0.001s	passed

RequestUriUtilsTest.java



RequestUriUtilsTest

all > stirling.software.SPDF.utils > RequestUriUtilsTest

9	0	0	0.016s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
isp_testIsStaticResource_RobotsTxt_EmptyContext()	0.001s	passed
isp_testIsStaticResource_RobotsTxt_NonEmptyContext()	0.002s	passed
isp_testIsStaticResource_WithNonEmptyContext_ExactMatch()	0s	passed
isp_testIsStaticResource_WithNonEmptyContext_NearMiss()	0.008s	passed
isp_testIsTrackableResource_BoundaryJustBelowStaticThreshold()	0.001s	passed
isp_testIsTrackableResource_ContainsSwaggerBoundary()	0.001s	passed
isp_testIsTrackableResource_EndsWithBoundary()	0.001s	passed
isp_testIsTrackableResource_JustAboveExcluded()	0.001s	passed
testIsStaticResource()	0.001s	passed

Graph-based testing (CFG and DFG)

Example 1:

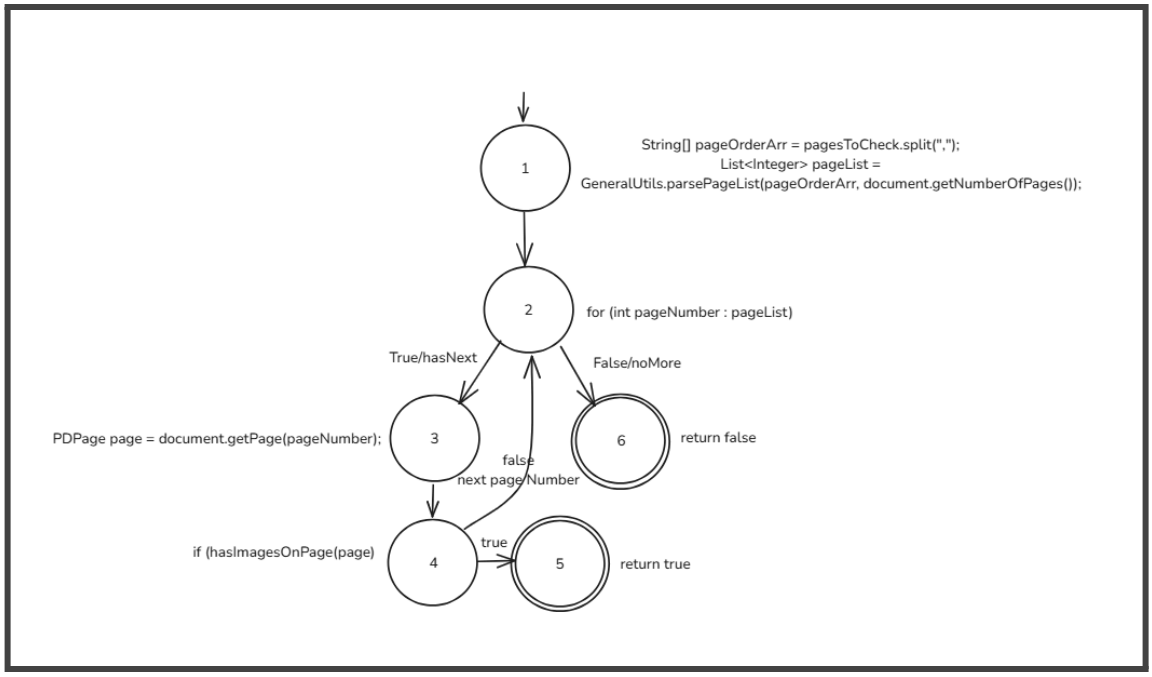
An example of Graph-based testing in our project is testing done on the PdfUtils.java class. The specific example chosen is testing done on the hasImages() function.

hasImages():

```
85    public static boolean hasImages(PDDocument document, String pagesToCheck) throws IOException {
86        String[] pageOrderArr = pagesToCheck.split(regex:",");
87        List<Integer> pageList =
88                GeneralUtils.parsePageList(pageOrderArr, document.getNumberOfPages());
89
90        for (int pageNumber : pageList) {
91            PDPage page = document.getPage(pageNumber);
92            if (hasImagesOnPage(page)) {
93                return true;
94            }
95        }
96
97        return false;
98    }
```

To ensure the line coverage of our test is adequate, a control flow diagram was drawn to understand better how to design test cases that take all possible branches in the function.

Control flow diagram:



After designing the control flow diagram for hasImages(), it was decided to design a set of test cases which satisfy edge coverage in order to increase the coverage of the tests on the PdfUtils.java class.

Test requirements for Edge Coverage:

TR(EC) = Ensure every edge is taken at least once

- TR1: Provide input so that the pagesToCheck string produces a page list with at least one page where hasImagesOnPage(page) returns true. This will exercise the branch in the loop where the condition is met and the method returns true immediately.
- TR2: Provide input so that the pagesToCheck string produces a page list with multiple pages and where hasImagesOnPage(page) returns false for every page. This ensures the loop iterates through all pages and the method finally returns false.
- TR3: Ensure that the page list has more than one element so that the loop's back edge is exercised. This means the loop should iterate over at least two pages (even if both return false), which covers the transition from one iteration to the next.

Test set T to satisfy TR(EC):

T = {

"Document with at least one page containing an image (e.g. a page created with createPageWithImage()) returns true",

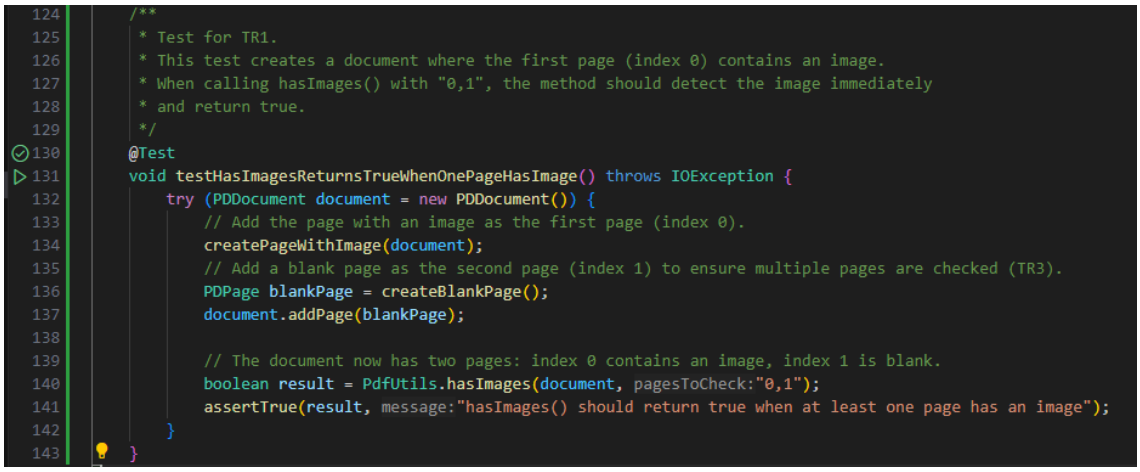
"Document with multiple pages that contain no images (e.g. pages created with createBlankPage()) returns false"

}

Implementation of Junit test for hasImages():

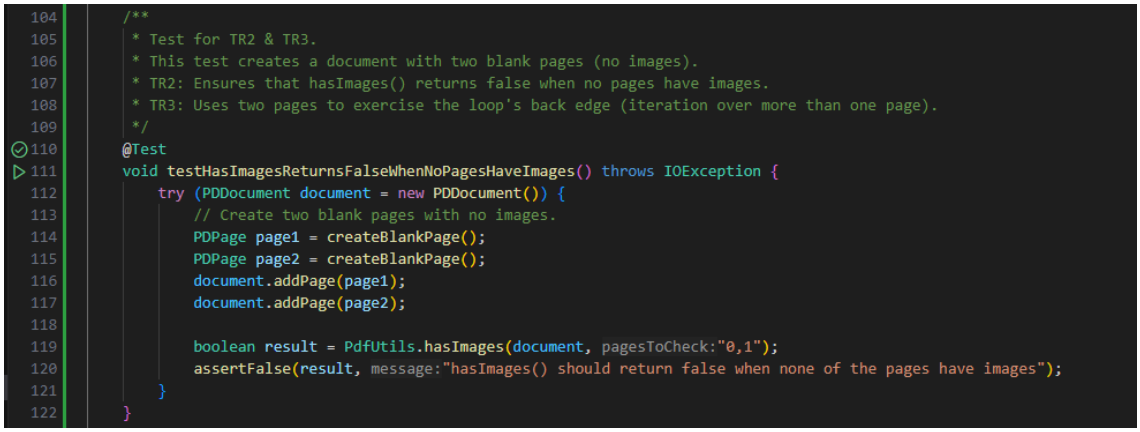
Two tests were written to satisfy edge coverage for the hasImages function,

Test 1:



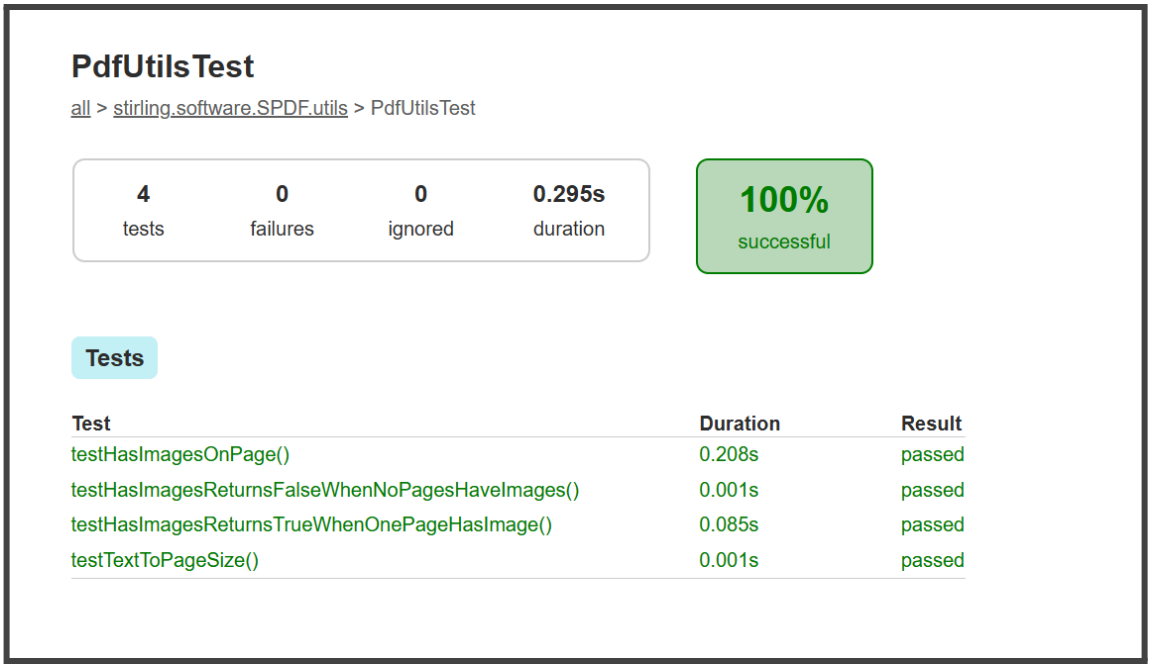
```
124     /**
125      * Test for TR1.
126      * This test creates a document where the first page (index 0) contains an image.
127      * When calling hasImages() with "0,1", the method should detect the image immediately
128      * and return true.
129      */
130     @Test
131     void testHasImagesReturnsTrueWhenOnePageHasImage() throws IOException {
132         try (PDDocument document = new PDDocument()) {
133             // Add the page with an image as the first page (index 0).
134             createPageWithImage(document);
135             // Add a blank page as the second page (index 1) to ensure multiple pages are checked (TR3).
136             PDPage blankPage = createBlankPage();
137             document.addPage(blankPage);
138 
139             // The document now has two pages: index 0 contains an image, index 1 is blank.
140             boolean result = PdfUtils.hasImages(document, pagesToCheck:"0,1");
141             assertTrue(result, message:"hasImages() should return true when at least one page has an image");
142         }
143     }
```

Test 2:



```
104     /**
105      * Test for TR2 & TR3.
106      * This test creates a document with two blank pages (no images).
107      * TR2: Ensures that hasImages() returns false when no pages have images.
108      * TR3: Uses two pages to exercise the loop's back edge (iteration over more than one page).
109      */
110     @Test
111     void testHasImagesReturnsFalseWhenNoPagesHaveImages() throws IOException {
112         try (PDDocument document = new PDDocument()) {
113             // Create two blank pages with no images.
114             PDPage page1 = createBlankPage();
115             PDPage page2 = createBlankPage();
116             document.addPage(page1);
117             document.addPage(page2);
118 
119             boolean result = PdfUtils.hasImages(document, pagesToCheck:"0,1");
120             assertFalse(result, message:"hasImages() should return false when none of the pages have images");
121         }
122     }
```

Test Results:



PdfUtilsTest

all > stirling.software.SPDF.utils > PdfUtilsTest

4	0	0	0.295s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
testHasImagesOnPage()	0.208s	passed
testHasImagesReturnsFalseWhenNoPagesHaveImages()	0.001s	passed
testHasImagesReturnsTrueWhenOnePageHasImage()	0.085s	passed
testTextToPageSize()	0.001s	passed

As you can see, the tests added to cover edge coverage passed. The added benefit of edge coverage is that it provides a more comprehensive testing strategy that not only confirms that all parts of the code run but also that each decision point behaves correctly under all circumstances.

Example 2:

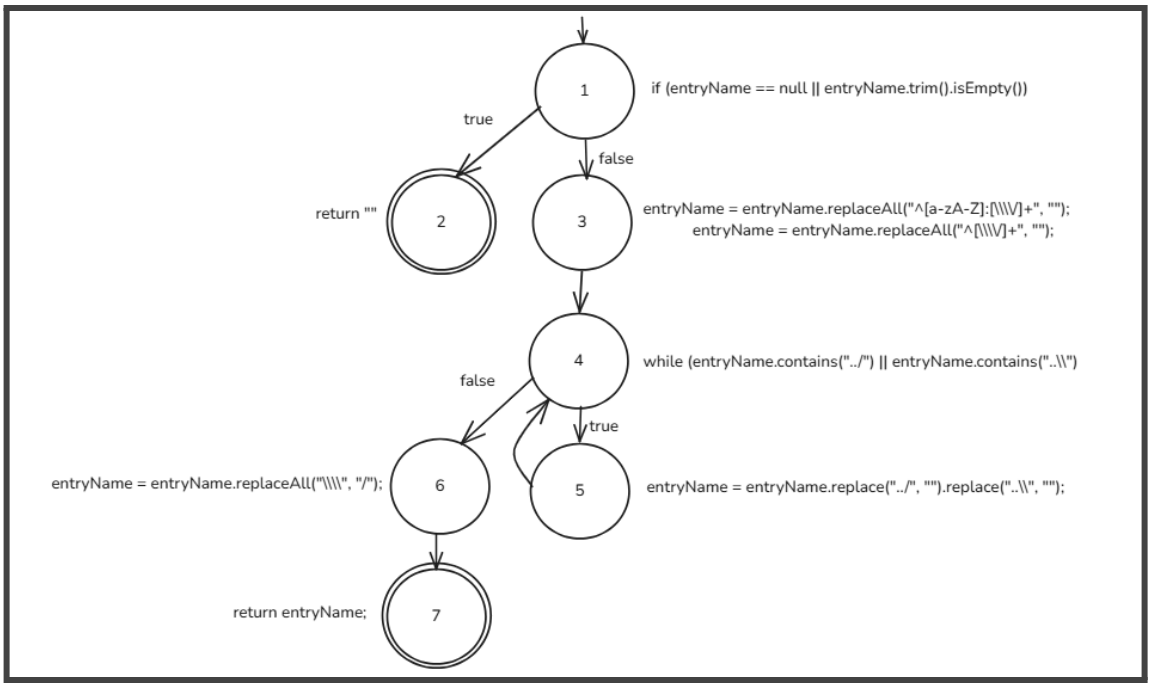
Another example of graph-based testing in our project is done on the FileToPdf.java class. The specific example chosen is testing done on the sanitizeZipFilename() method.

sanitizeZipFilename():

```
194     static String sanitizeZipFilename(String entryName) {
195         if (entryName == null || entryName.trim().isEmpty()) {
196             return "";
197         }
198         // Remove any drive letters (e.g., "C:\") and leading forward/backslashes
199         entryName = entryName.replaceAll(regex:"^[a-zA-Z]:[\\\\/]+", replacement:"");
200         entryName = entryName.replaceAll(regex:"^[\\\\/]+", replacement:"");
201 
202         // Recursively remove path traversal sequences
203         while (entryName.contains(s:"../") || entryName.contains(s:"..\\")) {
204             entryName = entryName.replace(target:"../", replacement:"").replace(target:"..\\", replacement:"");
205         }
206         // Normalize all backslashes to forward slashes
207         entryName = entryName.replaceAll(regex:"\\\\", replacement:"/");
208         return entryName;
209     }
210 }
```

The sanitizeZipFilename was chosen because it has many redefinitions of a variable, which is also used in various parts of the code. This makes this method a good candidate for developing tests based on a data flow diagram.

Control flow diagram:



We are interested in testing the dataflow in the context of the ‘entryName’ variable. To do this effectively, we define the definitions and use nodes for the variable ‘entryName’

Definition nodes for ‘entryName’ = {3,5,6}

Use nodes for ‘entryName’ = {1,3,4,5,6,7}

We will develop a set of test cases that achieve all def, and all use coverage.

Test requirements for All-Def/All-Use coverage:

TR(ADU) = Ensure each definition of entryName (at nodes 3, 5, 6) reaches every possible usage (at nodes 1, 3, 4, 5, 6, 7).

- TR1: Provide a null or empty string (entryName == null || entryName.trim().isEmpty()), so we return immediately at node 1 with no redefinition. This covers usage at node 1 but does not introduce a new definition (nodes 3, 5, and 6 are skipped).

- TR2: Supply a non-empty string without any "../" or "..\\" so the while loop never executes (node 4 is false on the first check). This exercises definition node 3 (the replaceAll calls) and ensures it flows to usage nodes 4, 6, and 7.

- TR3: Provide a string containing path-traversal substrings, e.g. "../secret", so the while loop's body runs at least once, triggering definition node 5. This new definition then reaches usage nodes 4 (the loop check), 5 (the inside assignment), 6 (final slash replacement), and 7 (return).

- TR4: Include multiple path-traversal sequences, e.g. "../../data/../config", so the loop at node 4 runs more than once, re-defining entryName repeatedly at node 5. This covers the repeated def–use chain: 5 → 4 → 5 → ...

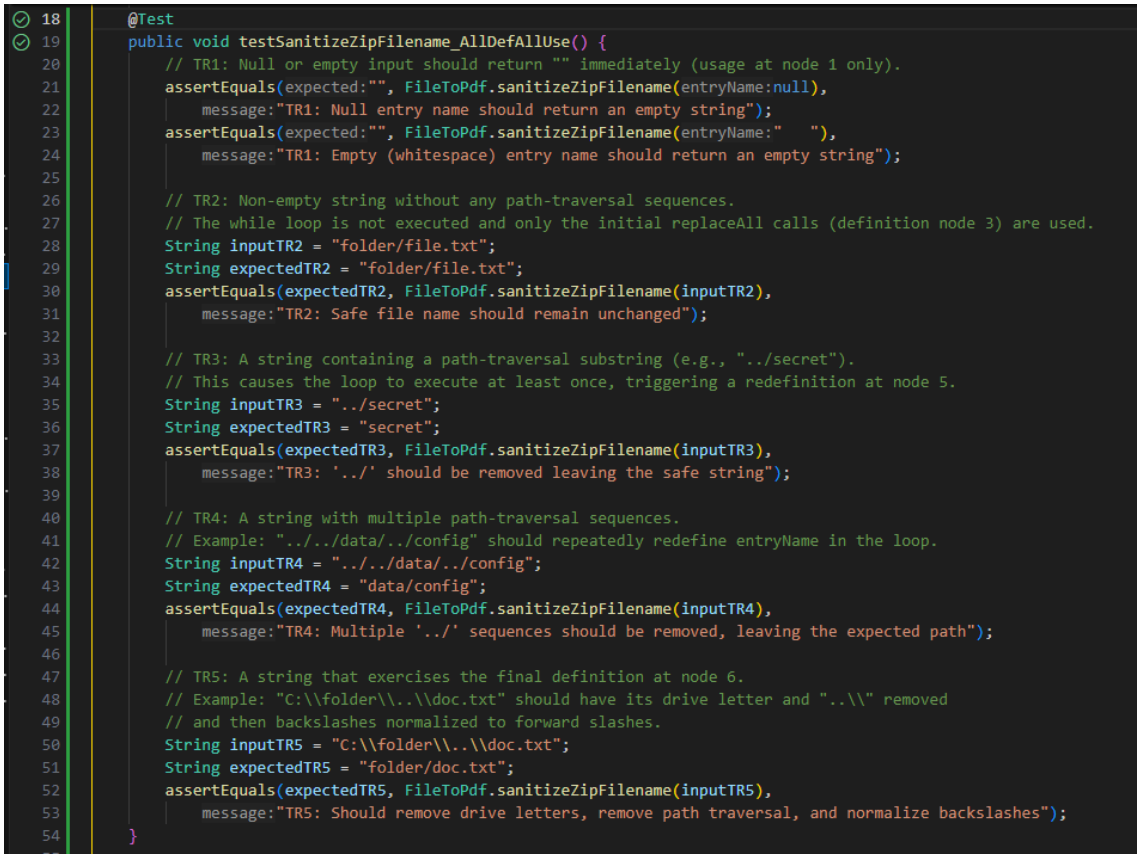
- TR5: Use a string that exercises the final definition at node 6 (backslash normalization), e.g. "C:\\folder\\test" or "\\leadingSlash". While you'll typically hit node 6 in both TR2 and TR3 anyway, specifying this ensures you confirm the final .replaceAll("\\\\", "/") is reached and tested.

Test set T to satisfy TR(ADU):

Test	Input(entry Name)	Description	TR1	TR2	TR3	TR4	TR5
T1	null	Covers returning early at node 1 (no definitions).	✓				
T2	"folder/file.txt"	No ../ → while loop never runs → covers definition 3.		✓			✓
T3	"../secret"	Has ../ → while loop runs once, covering definition 5.			✓		✓
T4	"../../data/../config"	Multiple ../ → loop re-executes (definition 5 repeated).			✓	✓	✓
T5	"C:\\folder\\..\\doc.txt"	Includes drive\\ and multiple ..\\ → covers def 3, loop at 5, and final def 6.			✓	✓	✓

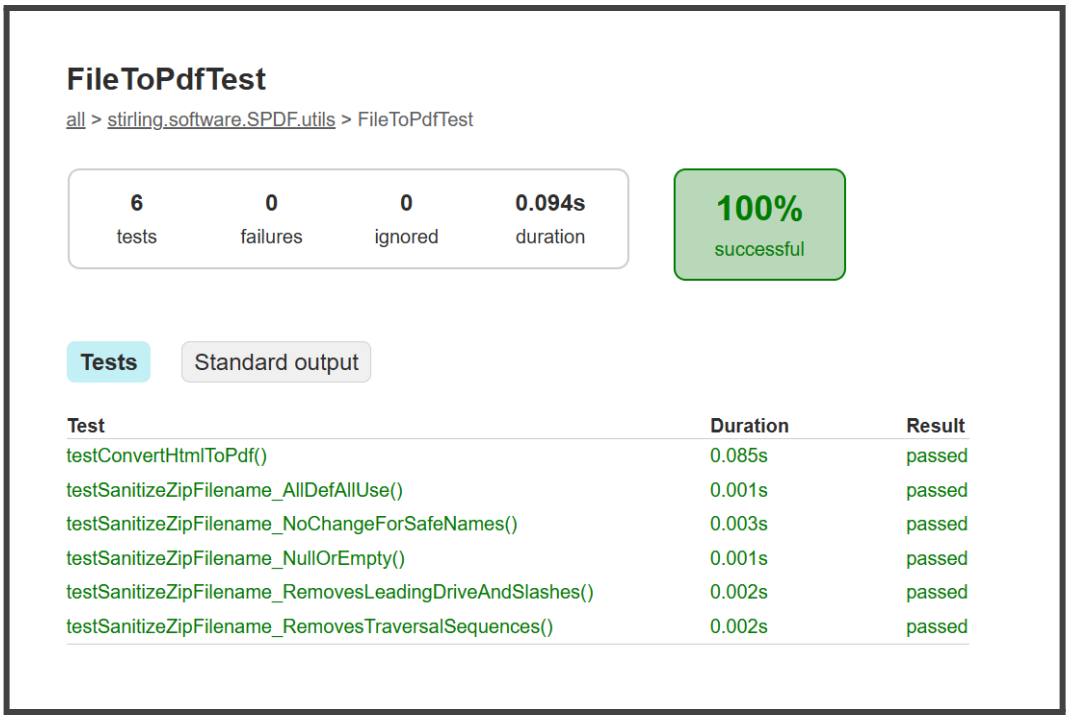
T = {T1, T2, T3, T4, T5}

The following JUnit test was developed to implement the test cases defined above.



```java
@Test
public void testSanitizeZipFilename_AllDefAllUse() {
    // TR1: Null or empty input should return "" immediately (usage at node 1 only).
    assertEquals(expected:"", FileToPdf.sanitizeZipFilename(entryName:null),
        message:"TR1: Null entry name should return an empty string");
    assertEquals(expected:"", FileToPdf.sanitizeZipFilename(entryName:"   "),
        message:"TR1: Empty (whitespace) entry name should return an empty string");

    // TR2: Non-empty string without any path-traversal sequences.
    // The while loop is not executed and only the initial replaceAll calls (definition node 3) are used.
    String inputTR2 = "folder/file.txt";
    String expectedTR2 = "folder/file.txt";
    assertEquals(expectedTR2, FileToPdf.sanitizeZipFilename(inputTR2),
        message:"TR2: Safe file name should remain unchanged");

    // TR3: A string containing a path-traversal substring (e.g., "../secret").
    // This causes the loop to execute at least once, triggering a redefinition at node 5.
    String inputTR3 = "../secret";
    String expectedTR3 = "secret";
    assertEquals(expectedTR3, FileToPdf.sanitizeZipFilename(inputTR3),
        message:"TR3: '../' should be removed leaving the safe string");

    // TR4: A string with multiple path-traversal sequences.
    // Example: "../../data/../config" should repeatedly redefine entryName in the loop.
    String inputTR4 = "../../data/../config";
    String expectedTR4 = "data/config";
    assertEquals(expectedTR4, FileToPdf.sanitizeZipFilename(inputTR4),
        message:"TR4: Multiple '../' sequences should be removed, leaving the expected path");

    // TR5: A string that exercises the final definition at node 6.
    // Example: "C:\\folder\\..\\doc.txt" should have its drive letter and "..\\" removed
    // and then backslashes normalized to forward slashes.
    String inputTR5 = "C:\\folder\\..\\doc.txt";
    String expectedTR5 = "folder/doc.txt";
    assertEquals(expectedTR5, FileToPdf.sanitizeZipFilename(inputTR5),
        message:"TR5: Should remove drive letters, remove path traversal, and normalize backslashes");
}
```

Test results:



FileToPdfTest

all > stirling.software.SPDF.utils > FileToPdfTest

6	0	0	0.094s
tests	failures	ignored	duration

100% successful

Tests | Standard output

Test	Duration	Result
testConvertHtmlToPdf()	0.085s	passed
testSanitizeZipFilename_AllDefAllUse()	0.001s	passed
testSanitizeZipFilename_NoChangeForSafeNames()	0.003s	passed
testSanitizeZipFilename_NullOrEmpty()	0.001s	passed
testSanitizeZipFilename_RemovesLeadingDriveAndSlashes()	0.002s	passed
testSanitizeZipFilename_RemovesTraversalSequences()	0.002s	passed

As you can see, the added test case for all def and all use coverage passed.

For the remaining Graph-Based testing approaches, increasing line coverage for classes ‘FileInfo.java’, ‘PropertyConfigs.java’, and ‘UrlUtils.java’ was prioritized. This was done by considering the classes' control flow and adding tests that covered more nodes and edges. Overall, the Graph-based testing approaches added to the project helped increase line coverage and ensure that.

Logic-based Testing

PropSync.java

The PropSync class is a utility class that synchronizes property files and is specifically designed to handle translation files.

Key Logical Predicates & Paths

1. File Matching Filter

```
file.getName().matches("messages_.*\\.properties")
```

True Case: Files like messages_de_DE.properties

False Case: Files like README.md, config.txt

2. Skipping Base Language File

```
!"messages_en_GB.properties".equals(file.getName())
```

True Case: Continue processing

False Case: Skip processing this file

3. UTF-8 Reading Error Handling

```
try {
   lines = Files.readAllLines(file.toPath(), StandardCharsets.UTF_8);
} catch (MalformedInputException e) { ... }
```

True Case: The file is not UTF-8, and should be skipped

False Case: File is readable

4. Property Key Existence

```
if (currentProps.containsKey(key))
```

True Case: The key exists in translation file

False Case: Add “TODO: Translate” block

5. Line Parsing

if (!line.trim().isEmpty() && line.contains("="))

True Case: Parse key-value

False Case: Skip or treat as comment/empty

Paths

Path A: Valid UTF-8 file → All keys matched → No "TODO"

Path B: Valid UTF-8 file → Some missing keys → Adds "TODO" section

Path C: Invalid UTF-8 file → Skipped with warning

Path D: File is messages_en_GB.properties → Skipped

Testing

The method we will show testing for is: linesToProps (lines 43-46):

if (!line.trim().isEmpty() && line.contains("="))

This predicate consists of two major clauses:

A: !line.trim().isEmpty()

B: line.contains("=")

So the full predicate is A && B

1. **Truth Table**

Test #	A	B	Predicate (A && B)
T1	T	T	T
T2	T	F	F
T3	F	T	F
T4	F	F	F

2. **GACC – General Active Clause Coverage**

Major Clause	Set of possible tests
A	(T1,T3)
B	(T1,T2)

3. **CACC (Correlated Active Clause Coverage)**

Major Clause	Set of possible tests
A	(T1,T3)
B	(T1,T2)

4. **RACC (Restricted Active Clause Coverage)**

Major Clause	Set of possible tests
A	(T1,T3)
B	(T1,T2)

5. **GICC (General Inactive Clause Coverage)**

Major Clause	Set of possible tests	
A	No feasible pairs for P = T	P = F: (T2, T4)
B	No feasible pairs for P = T	P = F: (T3, T4)

6. **RICC (Restricted Inactive Clause Coverage)**

Major Clause	Set of possible tests	
A	No feasible pairs for P = T	P = F: (T2, T4)
B	No feasible pairs for P = T	P = F: (T3, T4)

In order to test a script like PropSync that's outside the source directory, we need to make a few modifications, as it involves file system operations and is meant to be run as a standalone application. To start with, we relocate the file with src for easier testing with Gradle and Junit. Then, we refactor the code to separate the core logic from file system operations, making it more testable. We also make the methods accessible when we separate the file processing logic out into its own method from the main method.

Below is the modified version of PropSync. The original version consisted of a main function with two private functions.

```
public class PropSync {

    public static void main(String[] args) throws IOException {
        File folder = new File("C:\\Users\\systo\\git\\Stirling-PDF\\src\\main\\resources");
        processFiles(folder, "messages_en_GB.properties");
    }

    // Extracted method to make testing possible
    public static void processFiles(File folder, String baseFileName) throws IOException {
        File[] files = folder.listFiles((dir, name) -> name.matches("messages_.*\\.properties"));

        List<String> enLines = Files.readAllLines(Paths.get(folder.getPath(), baseFileName), StandardCharsets.UTF_8);
        Map<String, String> enProps = linesToProps(enLines);

        for (File file : files) {
            if (!baseFileName.equals(file.getName())) {
                System.out.println("Processing file: " + file.getName());
                List<String> lines;
                try {
                    lines = Files.readAllLines(file.toPath(), StandardCharsets.UTF_8);
                } catch (MalformedInputException e) {
                    System.out.println("Skipping due to not UTF8 format for file: " + file.getName());
                    continue;
                } catch (IOException e) {
                    throw new UncheckedIOException(e);
                }

                Map<String, String> currentProps = linesToProps(lines);
                List<String> newLines = syncPropsWithLines(enProps, currentProps, enLines);

                Files.write(file.toPath(), newLines, StandardCharsets.UTF_8);
                System.out.println("Finished processing file: " + file.getName());
            }
        }
    }

    // Changed to public for testing
    public static Map<String, String> linesToProps(List<String> lines) {
        Map<String, String> props = new LinkedHashMap<>();
        for (String line : lines) {
            if (!line.trim().isEmpty() && line.contains("=")) {
                String[] parts = line.split("=", 2);
                props.put(parts[0].trim(), parts[1].trim());
            }
        }
        return props;
    }

    // Changed to public for testing
```

```
    public static List<String> syncPropsWithLines(Map<String, String> enProps, Map<String, String> currentProps,
                                                   List<String> enLines) {
        List<String> newLines = new ArrayList<>();
        boolean needsTranslateComment = false; // flag to check if we need to add "TODO: Translate"

        for (String line : enLines) {
            if (line.contains("=")) {
                String key = line.split("=", 2)[0].trim();

                if (currentProps.containsKey(key)) {
                    newLines.add(key + "=" + currentProps.get(key));
                    needsTranslateComment = false;
                } else {
                    if (!needsTranslateComment) {
                        newLines.add("##########################");
                        newLines.add("###  TODO: Translate   ###");
                        newLines.add("##########################");
                        needsTranslateComment = true;
                    }
                    newLines.add(line);
                }
            } else {
                // handle comments and other non-property lines
                newLines.add(line);
                needsTranslateComment = false;  // reset the flag when we encounter comments or empty lines
            }
        }

        return newLines;
    }
}
```

Methods:

main(String[] args): The main entry point of the class, which calls the processFiles method with a specific folder and base file name.

processFiles(File folder, String baseFileName): Processes all property files in the given folder, excluding the base file, by reading their contents, synchronizing them with the base file, and writing the updated contents back to the files.

linesToProps(List lines): Converts a list of lines from a property file into a map of key-value pairs, where each key is a property name, and each value is the corresponding property value.

syncPropsWithLines(Map<String, String> enProps, Map<String, String> currentProps, List enLines): Synchronizes the properties from the base file (enProps) with the properties from another file (currentProps), using the original lines from the base file (enLines) as a template, and adds "TODO: Translate" comments for missing translations.

In order to build the project with Gradle and achieve the 100% success rate for the test suite, we needed to discard tests which involved handling null or empty string key/value combinations, as the original functions handled them incorrectly.

PropSyncTest

all > stirling.software.SPDF.stats > PropSyncTest

13	0	0	0.754s
tests	failures	ignored	duration

100%
successful

Tests | Standard output | Standard error

Test	Duration	Result
testBaseFileSkipping()	0.006s	passed
testEdgeCaseEmptyEqualsSign()	0.004s	passed
testEndToEndSync()	0.005s	passed
testFileMatchingFilter()	0.004s	passed
testFullCombinedTest()	0.004s	passed
testIOExceptionHandling()	0.700s	passed
testLineParsingCACCAndRACC()	0.004s	passed
testLineParsingGACC()	0.003s	passed
testLineParsingGICCAndRICC()	0.005s	passed
testLinesToProps()	0.004s	passed
testMalformedInputHandling()	0.007s	passed
testSyncPropsWithLines()	0.004s	passed
testTodoMarkerResetLogic()	0.004s	passed

PropertyConfigs.java

Key Logical Predicates & Paths

1. getBooleanValue(String key, boolean defaultValue)

```
String value = System.getProperty(key);

if (value == null) value = System.getenv(key);

return (value != null) ? Boolean.valueOf(value) : defaultValue;
```

Logical Predicates

1. System.getProperty(key) == null
2. System.getenv(key) == null
3. value != null

Paths

Path #	Description
P1	Property is set → return parsed Boolean
P2	Property is null, env is set → return parsed Boolean
P3	Property and env are both null → return default

2. getStringValue(String key, String defaultValue)

```
String value = System.getProperty(key);

if (value == null) value = System.getenv(key);

return (value != null) ? value : defaultValue;
```

Logical Predicates

Same as method #1.

Paths

Path #	Description
P1	Property is set → return it
P2	Property null, env is set → return it
P3	Both null → return default

3. getBooleanValue(List<String> keys, boolean defaultValue)

```
for (String key : keys) {
   String value = System.getProperty(key);
   if (value == null) value = System.getenv(key);
   if (value != null) return Boolean.valueOf(value);
}
return defaultValue;
```

Logical Predicates (inside loop)

1. System.getProperty(key) == null
2. System.getenv(key) == null
3. value != null
4. Loop exhaustion (no key matched)

Paths

Path #	Description
P1	First key returns property → return parsed Boolean
P2	First key is null, env set → return parsed Boolean
P3	Multiple keys tried, one eventually matches
P4	All keys fail → return default

4. getStringValue(List<String> keys, String defaultValue)

```
for (String key : keys) {
   String value = System.getProperty(key);
   if (value == null) value = System.getenv(key);
   if (value != null) return value;
}
return defaultValue;
```

Logical Predicates

Same as method #3.

Paths

Path #	Description
P1	First key returns property value
P2	First key returns env value
P3	Later key in list returns value
P4	No keys yield value → return default

Testing

The method we will show testing for is:getBooleanValue (lines 27-31):

```
(value != null) ? Boolean.valueOf(value) : defaultValue
```

This predicate consists of two major clauses:

A: System.getProperty(key) != null

B: System.getenv(key) != null

So the full predicate is A or B (A ∨ B)

1. **Truth Table**

Test #	A	B	Predicate (A && B)
T1	T	T	T
T2	T	F	T
T3	F	T	T
T4	F	F	F

2. **GACC – General Active Clause Coverage**

Major Clause	Set of possible tests
A	(T2, T4)
B	(T3, T4)

3. **CACC (Correlated Active Clause Coverage)**

Major Clause	Set of possible tests
A	(T2, T4)
B	(T3, T4)

4. **RACC (Restricted Active Clause Coverage)**

Major Clause	Set of possible tests
A	(T2, T4)
B	(T3, T4)

5. **GICC (General Inactive Clause Coverage)**

Major Clause	Set of possible tests	
A	P = T: (T1, T3)	No feasible pairs for P = T
B	P = T: (T1, T2)	No feasible pairs for P = T

6. **RICC (Restricted Inactive Clause Coverage)**

Major Clause	Set of possible tests	
A	P = T: (T1, T3)	No feasible pairs for P = T
B	P = T: (T1, T2)	No feasible pairs for P = T

To test PropertyConfigs.java, we need to simulate system properties and environment variables (which are typically read-only). To do this, we mock them by temporarily setting/unsetting system properties using System.setProperty(...). Mocking System.getenv() is more complex, so we simulate scenarios.

PropertyConfigsTest

all > stirling.software.SPDF.utils > PropertyConfigsTest

24	0	0	0.006s
tests	failures	ignored	duration

100%
successful

Tests

Test	Duration	Result
isp_testGetBooleanValue_FalseValue()	0s	passed
isp_testGetBooleanValue_SingleKeyEmptyValue()	0.001s	passed
isp_testGetBooleanValue_WithEmptyKeyList()	0s	passed
isp_testGetBooleanValue_WithEmptyPropertyValue()	0s	passed
isp_testGetStringValue_NonEmptyValue()	0s	passed
isp_testGetStringValue_SingleKeyEmptyValue()	0s	passed
isp_testGetStringValue_WithEmptyKeyList()	0s	passed
isp_testGetStringValue_WithEmptyPropertyValue()	0.001s	passed
testBooleanValueList_firstPropertyMatch()	0s	passed
testBooleanValueList_noneFound()	0s	passed
testBooleanValueList_secondEnvMatch_simulated()	0.001s	passed
testBooleanValue_envSet_simulated()	0.002s	passed
testBooleanValue_noneSet()	0s	passed
testBooleanValue_propertySet()	0s	passed
testGetBooleanValue_WithKey()	0s	passed
testGetBooleanValue_WithKeys()	0s	passed
testGetStringValue_WithKey()	0s	passed
testGetStringValue_WithKeys()	0s	passed
testStringValueList_firstMatch()	0s	passed
testStringValueList_laterMatch_simulatedEnv()	0s	passed
testStringValueList_noneMatch()	0s	passed
testStringValue_envSet_simulated()	0s	passed
testStringValue_noneSet()	0.001s	passed
testStringValue_propertySet()	0s	passed

UIScaling.java

Key Logical Predicates & Paths

1. **Null Check in scaleIcon**
    - Predicate:
        - if (icon == null) return null;
    - Expected Cases:
        - True: When icon == null --> immediately returns null.
        - False: When icon != null --> continues processing.
2. **Aspect Ratio Decision in scaleIcon**
    - Predicate:

      if (scaledWidth / scaledHeight > aspectRatio) {

            scaledWidth = (int)(scaledHeight * aspectRatio);

      } else {

            scaledHeight = (int)(scaledWidth / aspectRatio);

      }

    - Expected Cases:
        - True Branch: (scaledWidth/scaledHeight) > aspectRatio --> adjust width.
        - False Branch: (scaledWidth/scaledHeight) <= aspectRatio --> adjust height.
3. **Scale Factor Selection in scaleFont**
    - Operation:
        - double scaleFactor = Math.min(getWidthScaleFactor(), getHeightScaleFactor());
    - Expected Cases:
        - Case 1: When getWidthScaleFactor() < getHeightScaleFactor(), use width factor.
        - Case 2: When getHeightScaleFactor() <= getWidthScaleFactor(), use height factor.

A. **Null Check (Predicate: icon == null)**
   - **Truth Table**

Test #	icon is null?	Predicate Value	Outcome
T1	True	True	Return null
T2	False	False	Proceed with scaling

- **GACC (General Active Clause Coverage):**
  - Since the predicate consists of a single clause, using T1 (icon is null) and T2 (icon is non-null) demonstrates the independent effect of that clause.
- **CACC (Correlated Active Clause Coverage):**
  - With one clause, the test pair (T1, T2) is sufficient, so the same pair satisfies CACC.
- **RACC (Restricted Active Clause Coverage):**
  - Again, the same test pair (T1, T2) fulfills RACC.
- **GICC (General Inactive Clause Coverage) and RICC (Restricted Inactive Clause Coverage):**
  - Not applicable for single-clause predicates

B. **Aspect Ratio Decision (Predicate: (scaledWidth / scaledHeight) > aspectRatio)**
    - **Truth Table**

Test #	(scaledWidth/scaledHeight) > aspectRatio?	Outcome
T1	True	scaledWidth = (int)(scaledHeight * aspectRatio)
T2	False	scaledHeight = (int)(scaledWidth / aspectRatio)

- GACC (General Active Clause Coverage):
    - The test pair (T1, T2) demonstrates that a change in the (only) clause's truth value (true vs. false) independently affects the outcome.
- CACC (Correlated Active Clause Coverage):
    - Because the predicate is a single clause, the same pair (T1, T2) satisfies CACC
- RACC (Restricted Active Clause Coverage):
    - Again, the same test pair (T1, T2) suffices, meeting RACC requirements.
- GICC (General Inactive Clause Coverage) and RICC (Restricted Inactive Clause Coverage):
    - Not applicable here, as there is only one clause in the predicate.

Mutation Testing

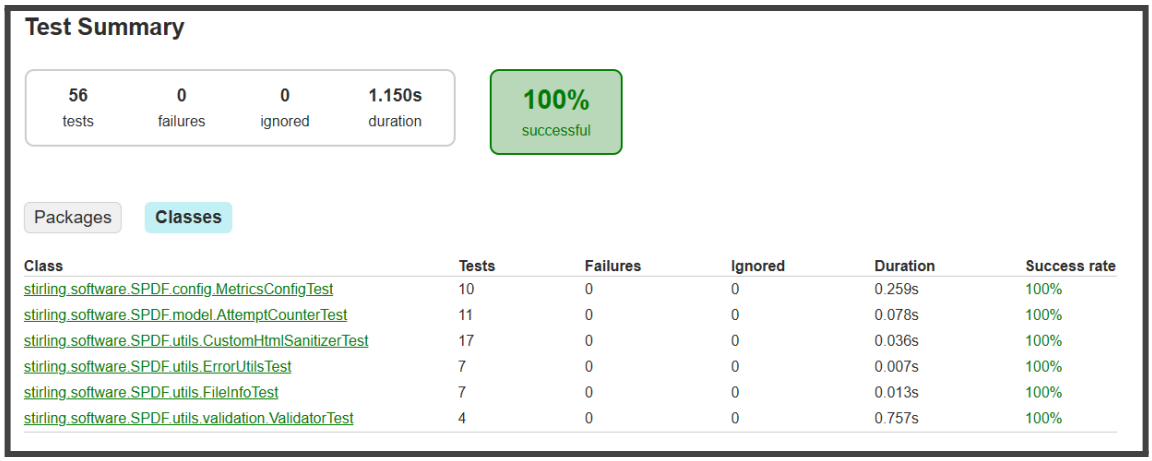
Testing tools:

- Pitest
- Junit5
- Gradle
- Springframework
- Java 17 or above
- VSCode

Tested classes:

- ‘stirling.software.SPDF.utils.ErrorUtils'
- 'stirling.software.SPDF.utils.FileInfo'
- 'stirling.software.SPDF.utils.CustomHtmlSanitizer'
- 'stirling.software.SPDF.model.AttemptCounter'
- 'stirling.software.SPDF.config.MetricsConfig

Overall 15 functions were tested across these 5 classes. These are the final results and we will further discuss how we achieved this test coverage below.



Test Summary

56 tests	0 failures	0 ignored	1.150s duration

100% successful

Packages **Classes**

Class	Tests	Failures	Ignored	Duration	Success rate
stirling.software.SPDF.config.MetricsConfigTest	10	0	0	0.259s	100%
stirling.software.SPDF.model.AttemptCounterTest	11	0	0	0.078s	100%
stirling.software.SPDF.utils.CustomHtmlSanitizerTest	17	0	0	0.036s	100%
stirling.software.SPDF.utils.ErrorUtilsTest	7	0	0	0.007s	100%
stirling.software.SPDF.utils.FileInfoTest	7	0	0	0.013s	100%
stirling.software.SPDF.utils.validation.ValidatorTest	4	0	0	0.757s	100%

Initial mutation testing:

The initial mutation coverage of these 5 classes revealed that there was poor mutational and line coverage; most of the selected classes did not initially have any Junit test classes at all except for the ErrorUtils.java file in the utils package.

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
5	22%	13/58	7%	2/29	50%	2/4

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
stirling.software.SPDF.config	1	0%	0/2	0%	0/1	100%	0/0
stirling.software.SPDF.model	1	0%	0/13	0%	0/7	100%	0/0
stirling.software.SPDF.utils	3	30%	13/43	10%	2/21	50%	2/4

Report generated by PIT 1.15.0

Enhanced functionality available at arcmutate.com

This is the initial mutation testing report for the Utils package:

Pit Test Coverage Report

Package Summary

stirling.software.SPDF.utils

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
3	30%	13/43	10%	2/21	50%	2/4

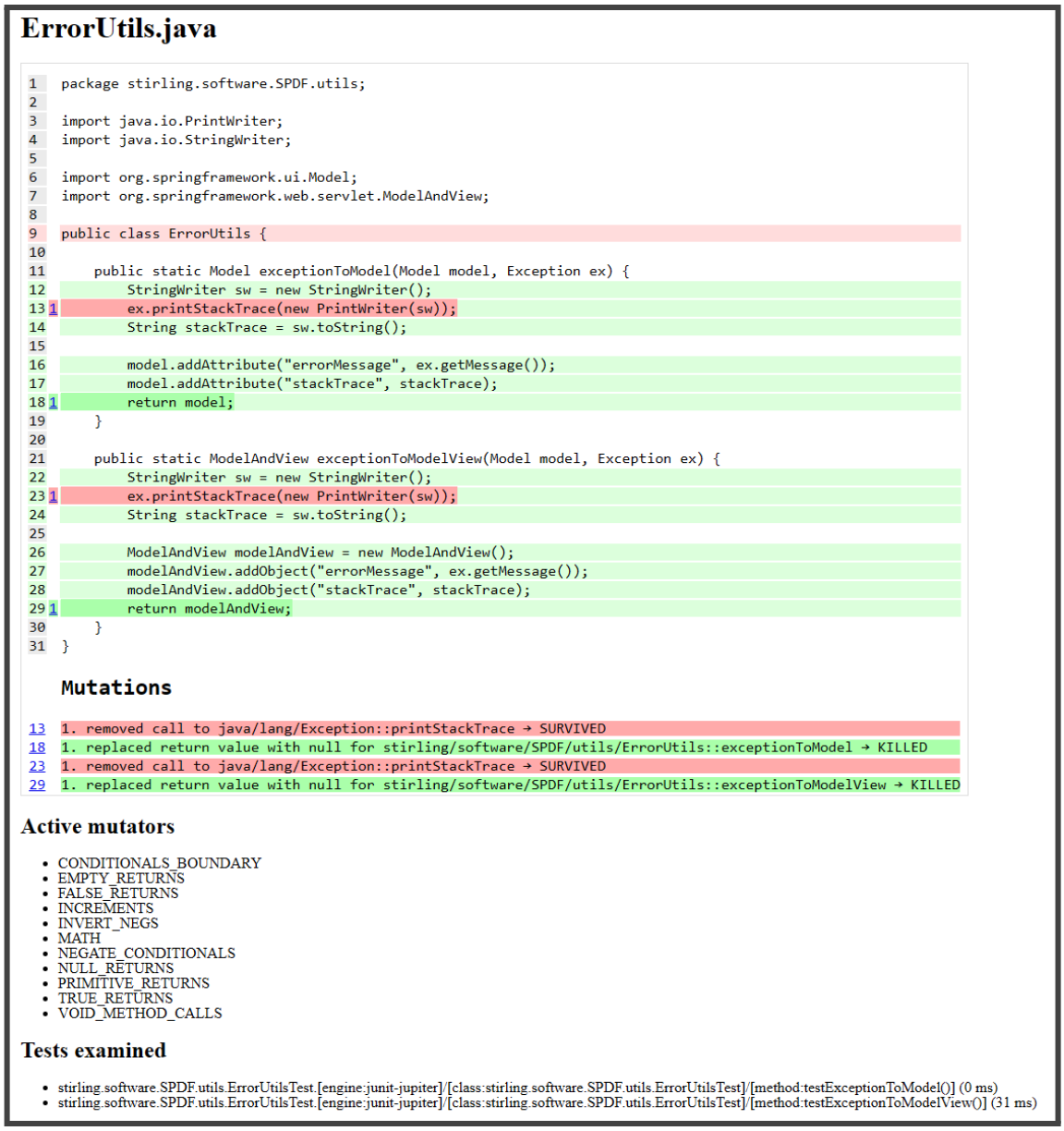
Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
CustomHtmlSanitizer.java	0%	0/10	0%	0/1	100%	0/0
ErrorUtils.java	93%	13/14	50%	2/4	50%	2/4
FileInfo.java	0%	0/19	0%	0/16	100%	0/0

Report generated by PIT 1.15.0

Example #1:

The only class with any mutation coverage initially was ErrorUtils.java. This is because the repository contained two tests for it. The initial two tests focus on the basic functionality of converting an exception into a useful model representation. In one test, testExceptionToModel takes a simple exception with a set error message and verifies that the returned model contains both the correct "errorMessage" and a non-null "stackTrace". Similarly, testExceptionToModelView performs the same verification but for a ModelAndView object. Essentially, both tests confirm that the utility methods embed the expected error information into the model or view, ensuring that the core conversion logic works as intended.



ErrorUtils.java

```
1  package stirling.software.SPDF.utils;
2
3  import java.io.PrintWriter;
4  import java.io.StringWriter;
5
6  import org.springframework.ui.Model;
7  import org.springframework.web.servlet.ModelAndView;
8
9  public class ErrorUtils {
10
11     public static Model exceptionToModel(Model model, Exception ex) {
12         StringWriter sw = new StringWriter();
13 1       ex.printStackTrace(new PrintWriter(sw));
14         String stackTrace = sw.toString();
15
16         model.addAttribute("errorMessage", ex.getMessage());
17         model.addAttribute("stackTrace", stackTrace);
18 1       return model;
19     }
20
21     public static ModelAndView exceptionToModelView(Model model, Exception ex) {
22         StringWriter sw = new StringWriter();
23 1       ex.printStackTrace(new PrintWriter(sw));
24         String stackTrace = sw.toString();
25
26         ModelAndView modelAndView = new ModelAndView();
27         modelAndView.addObject("errorMessage", ex.getMessage());
28         modelAndView.addObject("stackTrace", stackTrace);
29 1       return modelAndView;
30     }
31 }
```

Mutations

```
13  1. removed call to java/lang/Exception::printStackTrace → SURVIVED
18  1. replaced return value with null for stirling/software/SPDF/utils/ErrorUtils::exceptionToModel → KILLED
23  1. removed call to java/lang/Exception::printStackTrace → SURVIVED
29  1. replaced return value with null for stirling/software/SPDF/utils/ErrorUtils::exceptionToModelView → KILLED
```

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

Tests examined

- stirling.software.SPDF.utils.ErrorUtilsTest.[engine:junit-jupiter]/[class:stirling.software.SPDF.utils.ErrorUtilsTest]/[method:testExceptionToModel()] (0 ms)
- stirling.software.SPDF.utils.ErrorUtilsTest.[engine:junit-jupiter]/[class:stirling.software.SPDF.utils.ErrorUtilsTest]/[method:testExceptionToModelView()] (31 ms)

However, these initial tests are not enough because they only cover a single, straightforward scenario where the exception contains a predefined message. They do not account for potential edge cases, such as when an exception has a null message or when subtle differences between various model implementations might affect the outcome. Moreover, they miss verifying that the same model instance is maintained throughout the conversion process, which could be critical for preserving state in more complex applications. This narrow focus means that any changes or mutations in the underlying functionality might go unnoticed until they manifest in production.

The added tests improve the test class package by broadening the coverage and ensuring robustness. They introduce scenarios with null messages, ensuring that the error handling gracefully deals with exceptions that lack a message while still providing a complete stack trace. Additional tests confirm that the model instance consistency is preserved, validating that the same model object is returned without unexpected mutations. Furthermore, by directly comparing the generated stack trace with an expected output from standard exception-handling utilities, these tests catch any potential discrepancies arising from changes in the string or print writer functionality. Collectively, these enhancements provide comprehensive assurance that the error conversion utilities will behave reliably under a wide range of realistic conditions.

We can see the direct improvements in line coverage and mutation coverage by looking at the report for the ErrorUtils class.

ErrorUtils.java

```
1  package stirling.software.SPDF.utils;
2
3  import java.io.PrintWriter;
4  import java.io.StringWriter;
5
6  import org.springframework.ui.Model;
7  import org.springframework.web.servlet.ModelAndView;
8
9  public class ErrorUtils {
10
11     public static Model exceptionToModel(Model model, Exception ex) {
12         StringWriter sw = new StringWriter();
13 1       ex.printStackTrace(new PrintWriter(sw));
14         String stackTrace = sw.toString();
15
16         model.addAttribute("errorMessage", ex.getMessage());
17         model.addAttribute("stackTrace", stackTrace);
18 1       return model;
19     }
20
21     public static ModelAndView exceptionToModelView(Model model, Exception ex) {
22         StringWriter sw = new StringWriter();
23 1       ex.printStackTrace(new PrintWriter(sw));
24         String stackTrace = sw.toString();
25
26         ModelAndView modelAndView = new ModelAndView();
27         modelAndView.addObject("errorMessage", ex.getMessage());
28         modelAndView.addObject("stackTrace", stackTrace);
29 1       return modelAndView;
30     }
31 }
```

Mutations

```
13  1. removed call to java/lang/Exception::printStackTrace → KILLED
18  1. replaced return value with null for stirling/software/SPDF/utils/ErrorUtils::exceptionToModel → KILLED
23  1. removed call to java/lang/Exception::printStackTrace → KILLED
29  1. replaced return value with null for stirling/software/SPDF/utils/ErrorUtils::exceptionToModelView → KILLED
```

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

Tests examined

- stirling.software.SPDF.utils.ErrorUtilsTest.[engine:junit-jupiter]/[class:stirling.software.SPDF.utils.ErrorUtilsTest]/[method:testExceptionToModel2()] (0 ms)
- stirling.software.SPDF.utils.ErrorUtilsTest.[engine:junit-jupiter]/[class:stirling.software.SPDF.utils.ErrorUtilsTest]/[method:testExceptionToModel()] (0 ms)
- stirling.software.SPDF.utils.ErrorUtilsTest.[engine:junit-jupiter]/[class:stirling.software.SPDF.utils.ErrorUtilsTest]/[method:testExceptionToModel_NullMessage()] (0 ms)
- stirling.software.SPDF.utils.ErrorUtilsTest.[engine:junit-jupiter]/[class:stirling.software.SPDF.utils.ErrorUtilsTest]/[method:testStackTraceGeneration()] (3 ms)
- stirling.software.SPDF.utils.ErrorUtilsTest.[engine:junit-jupiter]/[class:stirling.software.SPDF.utils.ErrorUtilsTest]/[method:testExceptionToModelView2()] (0 ms)
- stirling.software.SPDF.utils.ErrorUtilsTest.[engine:junit-jupiter]/[class:stirling.software.SPDF.utils.ErrorUtilsTest]/[method:testExceptionToModelView_NullMessage()] (0 ms)
- stirling.software.SPDF.utils.ErrorUtilsTest.[engine:junit-jupiter]/[class:stirling.software.SPDF.utils.ErrorUtilsTest]/[method:testExceptionToModelView()] (0 ms)

The second class I will discuss is relevant to **security**.

Example #2:

Next example is CustomHtmlSanitizer.java which is designed to clean HTML input by removing potentially harmful or unwanted content while preserving a set of safe, allowed elements and attributes. Using the OWASP HTML Sanitizer library, it builds a composite policy that allows basic formatting (e.g., paragraphs, bold and italic text), table structures, styles, links, and images, but deliberately excludes elements like <noscript>. This sanitation ensures that user-supplied HTML is stripped of any malicious code—such as embedded scripts, event handlers, or dangerous protocols—thus safeguarding web applications from cross-site scripting (XSS) attacks and similar vulnerabilities.

Despite its crucial nature as a layer of security, it did not have any test classes with it in the initial repository. Thus, we created our own test suite with junit5 to ensure its robustness.

The accompanying testing class we created, CustomHtmlSanitizerTest, exists to rigorously verify that the sanitizer operates correctly under a variety of conditions. Its purpose is to validate the expected behavior of the sanitizer by running a comprehensive suite of tests. These tests not only check that the sanitizer returns a valid, non-null output even when given null or blank inputs but also ensure that it properly preserves allowed HTML tags and attributes as defined in the policy. This guarantees that normal and safe HTML content passes through unchanged, thereby retaining the intended formatting and structure.

The testing class specifically examines multiple aspects of the sanitizer's functionality. For example, it tests that a null input yields an empty string and that empty or blank inputs remain unchanged, preventing errors related to null or empty returns. It verifies that simple texts are unaffected and that allowed elements—like <p>, <b>, <i>, table and image tags, as well as style attributes—are preserved. Additionally, the tests scrutinize the sanitizer’s ability to strip out disallowed elements such as <script>, <noscript>, and any malicious attributes like onclick or javascript: URLs. Finally, it confirms the idempotency of the operation by checking that repeated sanitization produces the same result. Together, these tests offer robust coverage to ensure that the sanitizer consistently produces secure yet visually intact HTML output in diverse scenarios.

This was the line and mutation coverage for CustomHtmlSanitizer before and after creating all of our tests:

CustomHtmlSanitizer.java

```
1  package stirling.software.SPDF.utils;
2
3  import org.owasp.html.HtmlPolicyBuilder;
4  import org.owasp.html.PolicyFactory;
5  import org.owasp.html.Sanitizers;
6
7  public class CustomHtmlSanitizer {
8      private static final PolicyFactory POLICY =
9              Sanitizers.FORMATTING
10                     .and(Sanitizers.BLOCKS)
11                     .and(Sanitizers.STYLES)
12                     .and(Sanitizers.LINKS)
13                     .and(Sanitizers.TABLES)
14                     .and(Sanitizers.IMAGES)
15                     .and(new HtmlPolicyBuilder().disallowElements("noscript").toFactory());
16
17     public static String sanitize(String html) {
18         String htmlAfter = POLICY.sanitize(html);
19 1       return htmlAfter;
20     }
21 }
```

Mutations

```
19  1. replaced return value with "" for stirling/software/SPDF/utils/CustomHtmlSanitizer::sanitize → NO_COVERAGE
```

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

CustomHtmlSanitizer.java

```
1  package stirling.software.SPDF.utils;
2
3  import org.owasp.html.HtmlPolicyBuilder;
4  import org.owasp.html.PolicyFactory;
5  import org.owasp.html.Sanitizers;
6
7  public class CustomHtmlSanitizer {
8      private static final PolicyFactory POLICY =
9              Sanitizers.FORMATTING
10                     .and(Sanitizers.BLOCKS)
11                     .and(Sanitizers.STYLES)
12                     .and(Sanitizers.LINKS)
13                     .and(Sanitizers.TABLES)
14                     .and(Sanitizers.IMAGES)
15                     .and(new HtmlPolicyBuilder().disallowElements("noscript").toFactory());
16
17     public static String sanitize(String html) {
18         String htmlAfter = POLICY.sanitize(html);
19 1       return htmlAfter;
20     }
21 }
```

Mutations

```
19  1. replaced return value with "" for stirling/software/SPDF/utils/CustomHtmlSanitizer::sanitize → KILLED
```

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

Tests examined

- stirling.software.SPDF.utils.CustomHtmlSanitizerTest [engine:junit-jupiter]/[class:stirling.software.SPDF.utils.CustomHtmlSanitizerTest]/[method:sanitize_WithTableElements_KeepsTableElements()] (0 ms)
- stirling.software.SPDF.utils.CustomHtmlSanitizerTest [engine:junit-jupiter]/[class:stirling.software.SPDF.utils.CustomHtmlSanitizerTest]/[method:sanitize_WithComplexNestedStructure_PreservesStructure()] (0 ms)
- stirling.software.SPDF.utils.CustomHtmlSanitizerTest [engine:junit-jupiter]/[class:stirling.software.SPDF.utils.CustomHtmlSanitizerTest]/[method:sanitize_WithImageTag_KeepsImageTag()] (0 ms)
- stirling.software.SPDF.utils.CustomHtmlSanitizerTest [engine:junit-jupiter]/[class:stirling.software.SPDF.utils.CustomHtmlSanitizerTest]/[method:sanitize_WithSimpleText_ReturnsSameText()] (0 ms)
- stirling.software.SPDF.utils.CustomHtmlSanitizerTest [engine:junit-jupiter]/[class:stirling.software.SPDF.utils.CustomHtmlSanitizerTest]/[method:sanitize_ReturnValueVerification_ReturnsActualSanitizedValue()] (0 ms)
- stirling.software.SPDF.utils.CustomHtmlSanitizerTest [engine:junit-jupiter]/[class:stirling.software.SPDF.utils.CustomHtmlSanitizerTest]/[test-template:sanitize_WithEmptyOrBlankInput_ReturnsEmptyOrBlankString(java.lang.String)]/[test-template-invocation:#4] (0 ms)
- stirling.software.SPDF.utils.CustomHtmlSanitizerTest [engine:junit-jupiter]/[class:stirling.software.SPDF.utils.CustomHtmlSanitizerTest]/[test-template:sanitize_WithEmptyOrBlankInput_ReturnsEmptyOrBlankString(java.lang.String)]/[test-template-invocation:#3] (0 ms)
- stirling.software.SPDF.utils.CustomHtmlSanitizerTest [engine:junit-jupiter]/[class:stirling.software.SPDF.utils.CustomHtmlSanitizerTest]/[test-template:sanitize_WithEmptyOrBlankInput_ReturnsEmptyOrBlankString(java.lang.String)]/[test-template-invocation:#2] (0 ms)
- stirling.software.SPDF.utils.CustomHtmlSanitizerTest [engine:junit-jupiter]/[class:stirling.software.SPDF.utils.CustomHtmlSanitizerTest]/[method:sanitize_WithNullInput_ReturnsEmptyString()] (0 ms)
- stirling.software.SPDF.utils.CustomHtmlSanitizerTest [engine:junit-jupiter]/[class:stirling.software.SPDF.utils.CustomHtmlSanitizerTest]/[test-template:sanitize_WithEmptyOrBlankInput_ReturnsEmptyOrBlankString(java.lang.String)]/[test-template-invocation:#1] (0 ms)
- stirling.software.SPDF.utils.CustomHtmlSanitizerTest [engine:junit-jupiter]/[class:stirling.software.SPDF.utils.CustomHtmlSanitizerTest]/[method:sanitize_WithMaliciousInput_SanitizesCorrectly()] (0 ms)
- stirling.software.SPDF.utils.CustomHtmlSanitizerTest [engine:junit-jupiter]/[class:stirling.software.SPDF.utils.CustomHtmlSanitizerTest]/[test-template:sanitize_WithEmptyOrBlankInput_ReturnsEmptyOrBlankString(java.lang.String)]/[test-template-invocation:#6] (0 ms)
- stirling.software.SPDF.utils.CustomHtmlSanitizerTest [engine:junit-jupiter]/[class:stirling.software.SPDF.utils.CustomHtmlSanitizerTest]/[test-template:sanitize_WithEmptyOrBlankInput_ReturnsEmptyOrBlankString(java.lang.String)]/[test-template-invocation:#5] (0 ms)
- stirling.software.SPDF.utils.CustomHtmlSanitizerTest [engine:junit-jupiter]/[class:stirling.software.SPDF.utils.CustomHtmlSanitizerTest]/[method:sanitize_WithAllowedHtml_KeepsAllowedElements()] (0 ms)
- stirling.software.SPDF.utils.CustomHtmlSanitizerTest [engine:junit-jupiter]/[class:stirling.software.SPDF.utils.CustomHtmlSanitizerTest]/[method:sanitize_WithDisallowedElements_RemovesDisallowedElements()] (0 ms)
- stirling.software.SPDF.utils.CustomHtmlSanitizerTest [engine:junit-jupiter]/[class:stirling.software.SPDF.utils.CustomHtmlSanitizerTest]/[method:sanitize_WithStyleAttributes_KeepsStyleAttributes()] (3 ms)
- stirling.software.SPDF.utils.CustomHtmlSanitizerTest [engine:junit-jupiter]/[class:stirling.software.SPDF.utils.CustomHtmlSanitizerTest]/[method:sanitize_IdempotencyTest_MultipleSanitizationsYieldSameResult()] (25 ms)

Example #3:

The final example is the FileInfo class, which is designed to encapsulate metadata about a file. It stores details such as the file name, file path, modification date, file size, and creation date. In addition to holding data, it provides utility methods for converting a file path string into a Path object, formatting the file size into a human-readable string with appropriate units (Bytes, KB, MB, or GB), and formatting the dates consistently using a predefined pattern. This utility not only standardizes file size representations across the application but also makes it easier for end users to interpret file information.

The accompanying testing class, FileInfoTest, verifies that the FileInfo class correctly formats file size values. Using JUnit's parameterized testing capabilities with CSV source data, this test class ensures that, for varying input sizes, the getFormattedFileSize() method produces the expected string output. The tests focus on several edge cases—such as sizes just before and after the boundaries of Bytes, KB, MB, and GB—to validate that the conversion logic manages both standard cases and boundary conditions accurately.

By systematically testing multiple file size inputs—including zero bytes, sizes just under or over the conversion thresholds, and larger sizes—the tests confirm that the formatting logic correctly handles unit conversion and rounding. This thorough approach helps catch potential issues, such as rounding mishandling or misapplied conversion factors and ultimately contributes to a robust and reliable implementation. The testing strategy ensures that any nuances in the conversion from raw numerical file sizes to a user-friendly representation are well accounted for, helping maintain consistency and accuracy in the application's file information display.

These are the results before mutation testing:

FileInfo.java

```
1  package stirling.software.SPDF.utils;
2
3  import java.nio.file.Path;
4  import java.nio.file.Paths;
5  import java.time.LocalDateTime;
6  import java.time.format.DateTimeFormatter;
7  import java.util.Locale;
8
9  import lombok.AllArgsConstructor;
10 import lombok.Data;
11
12 @AllArgsConstructor
13 @Data
14 public class FileInfo {
15     private static final DateTimeFormatter DATE_FORMATTER =
16             DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
17     private String fileName;
18     private String filePath;
19     private LocalDateTime modificationDate;
20     private long fileSize;
21     private LocalDateTime creationDate;
22
23     // Converts the file path string to a Path object.
24     public Path getFilePathAsPath() {
25 1       return Paths.get(filePath);
26     }
27
28     // Formats the file size into a human-readable string.
29     public String getFormattedFileSize() {
30 2       if (fileSize >= 1024 * 1024 * 1024) {
31 2           return String.format(Locale.US, "%.2f GB", fileSize / (1024.0 * 1024 * 1024));
32 2       } else if (fileSize >= 1024 * 1024) {
33 2           return String.format(Locale.US, "%.2f MB", fileSize / (1024.0 * 1024));
34 2       } else if (fileSize >= 1024) {
35 2           return String.format(Locale.US, "%.2f KB", fileSize / 1024.0);
36        } else {
37 1           return String.format("%d Bytes", fileSize);
38        }
39     }
40
41     // Formats the modification date to a string.
42     public String getFormattedModificationDate() {
43 1       return modificationDate.format(DATE_FORMATTER);
44     }
45
46     // Formats the creation date to a string.
47     public String getFormattedCreationDate() {
48 1       return creationDate.format(DATE_FORMATTER);
49     }
50 }
```

Mutations

```
25 1. replaced return value with null for stirling/software/SPDF/utils/FileInfo::getFilePathAsPath → NO_COVERAGE
30 1. changed conditional boundary → NO_COVERAGE
   2. negated conditional → NO_COVERAGE
31 1. Replaced double division with multiplication → NO_COVERAGE
   2. replaced return value with "" for stirling/software/SPDF/utils/FileInfo::getFormattedFileSize → NO_COVERAGE
32 1. negated conditional → NO_COVERAGE
   2. changed conditional boundary → NO_COVERAGE
33 1. replaced return value with "" for stirling/software/SPDF/utils/FileInfo::getFormattedFileSize → NO_COVERAGE
   2. Replaced double division with multiplication → NO_COVERAGE
34 1. negated conditional → NO_COVERAGE
   2. changed conditional boundary → NO_COVERAGE
35 1. replaced return value with "" for stirling/software/SPDF/utils/FileInfo::getFormattedFileSize → NO_COVERAGE
   2. Replaced double division with multiplication → NO_COVERAGE
37 1. replaced return value with "" for stirling/software/SPDF/utils/FileInfo::getFormattedFileSize → NO_COVERAGE
43 1. replaced return value with "" for stirling/software/SPDF/utils/FileInfo::getFormattedModificationDate → NO_COVERAGE
48 1. replaced return value with "" for stirling/software/SPDF/utils/FileInfo::getFormattedCreationDate → NO_COVERAGE
```

These are the results after creating the test suite from mutation testing:

FileInfo.java

```
1  package stirling.software.SPDF.utils;
2
3  import java.nio.file.Path;
4  import java.nio.file.Paths;
5  import java.time.LocalDateTime;
6  import java.time.format.DateTimeFormatter;
7  import java.util.Locale;
8
9  import lombok.AllArgsConstructor;
10 import lombok.Data;
11
12 @AllArgsConstructor
13 @Data
14 public class FileInfo {
15     private static final DateTimeFormatter DATE_FORMATTER =
16             DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
17     private String fileName;
18     private String filePath;
19     private LocalDateTime modificationDate;
20     private long fileSize;
21     private LocalDateTime creationDate;
22
23     // Converts the file path string to a Path object.
24     public Path getFilePathAsPath() {
25 1       return Paths.get(filePath);
26     }
27
28     // Formats the file size into a human-readable string.
29     public String getFormattedFileSize() {
30 2       if (fileSize >= 1024 * 1024 * 1024) {
31 2           return String.format(Locale.US, "%.2f GB", fileSize / (1024.0 * 1024 * 1024));
32 2       } else if (fileSize >= 1024 * 1024) {
33 2           return String.format(Locale.US, "%.2f MB", fileSize / (1024.0 * 1024));
34 2       } else if (fileSize >= 1024) {
35 2           return String.format(Locale.US, "%.2f KB", fileSize / 1024.0);
36        } else {
37 1           return String.format("%d Bytes", fileSize);
38        }
39     }
40
41     // Formats the modification date to a string.
42     public String getFormattedModificationDate() {
43 1       return modificationDate.format(DATE_FORMATTER);
44     }
45
46     // Formats the creation date to a string.
47     public String getFormattedCreationDate() {
48 1       return creationDate.format(DATE_FORMATTER);
49     }
50 }
```

Mutations

```
25 1. replaced return value with null for stirling/software/SPDF/utils/FileInfo::getFilePathAsPath → NO_COVERAGE
30 1. changed conditional boundary → KILLED
   2. negated conditional → KILLED
31 1. Replaced double division with multiplication → KILLED
   2. replaced return value with "" for stirling/software/SPDF/utils/FileInfo::getFormattedFileSize → KILLED
32 1. negated conditional → KILLED
   2. changed conditional boundary → KILLED
33 1. replaced return value with "" for stirling/software/SPDF/utils/FileInfo::getFormattedFileSize → KILLED
   2. Replaced double division with multiplication → KILLED
34 1. negated conditional → KILLED
   2. changed conditional boundary → KILLED
35 1. replaced return value with "" for stirling/software/SPDF/utils/FileInfo::getFormattedFileSize → KILLED
   2. Replaced double division with multiplication → KILLED
37 1. replaced return value with "" for stirling/software/SPDF/utils/FileInfo::getFormattedFileSize → KILLED
43 1. replaced return value with "" for stirling/software/SPDF/utils/FileInfo::getFormattedModificationDate → NO_COVERAGE
48 1. replaced return value with "" for stirling/software/SPDF/utils/FileInfo::getFormattedCreationDate → NO_COVERAGE
```

Active mutators

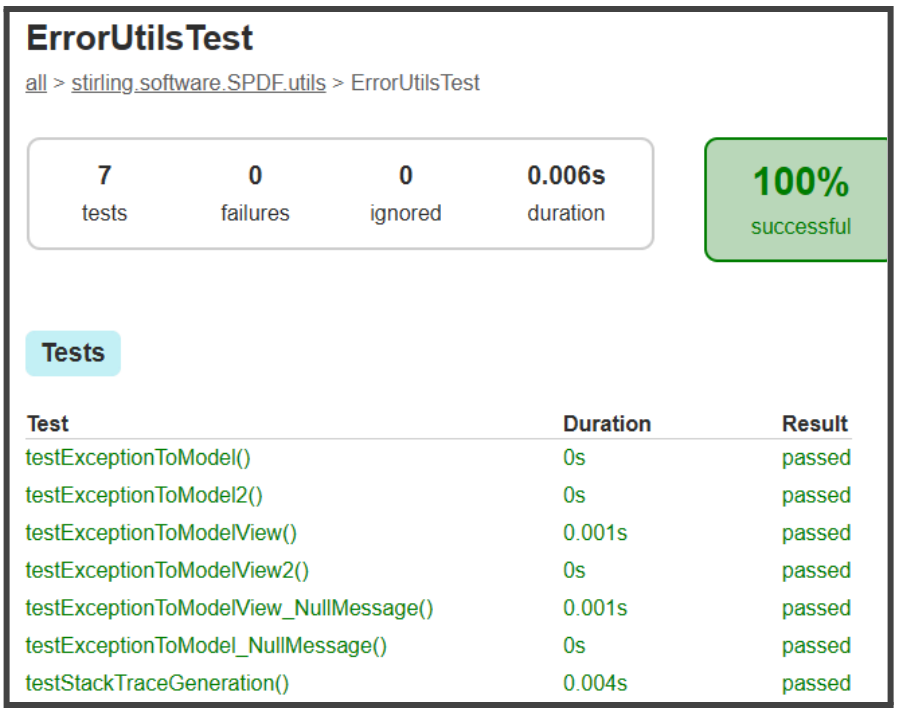
- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

Tests examined

- stirling.software.SPDF.utils.FileInfoTest.[engine:junit-jupiter]/[class:stirling.software.SPDF.utils.FileInfoTest]/[test-template:testGetFormattedFileSize(long, java.lang.String)]/[test-template-invocation:#4] (0 ms)
- stirling.software.SPDF.utils.FileInfoTest.[engine:junit-jupiter]/[class:stirling.software.SPDF.utils.FileInfoTest]/[test-template:testGetFormattedFileSize(long, java.lang.String)]/[test-template-invocation:#3] (0 ms)
- stirling.software.SPDF.utils.FileInfoTest.[engine:junit-jupiter]/[class:stirling.software.SPDF.utils.FileInfoTest]/[test-template:testGetFormattedFileSize(long, java.lang.String)]/[test-template-invocation:#5] (0 ms)
- stirling.software.SPDF.utils.FileInfoTest.[engine:junit-jupiter]/[class:stirling.software.SPDF.utils.FileInfoTest]/[test-template:testGetFormattedFileSize(long, java.lang.String)]/[test-template-invocation:#2] (0 ms)
- stirling.software.SPDF.utils.FileInfoTest.[engine:junit-jupiter]/[class:stirling.software.SPDF.utils.FileInfoTest]/[test-template:testGetFormattedFileSize(long, java.lang.String)]/[test-template-invocation:#6] (0 ms)
- stirling.software.SPDF.utils.FileInfoTest.[engine:junit-jupiter]/[class:stirling.software.SPDF.utils.FileInfoTest]/[test-template:testGetFormattedFileSize(long, java.lang.String)]/[test-template-invocation:#7] (0 ms)
- stirling.software.SPDF.utils.FileInfoTest.[engine:junit-jupiter]/[class:stirling.software.SPDF.utils.FileInfoTest]/[test-template:testGetFormattedFileSize(long, java.lang.String)]/[test-template-invocation:#1] (13 ms)

Testing Results For Two Examples:

Example 1: ErrorUtils:



ErrorUtilsTest

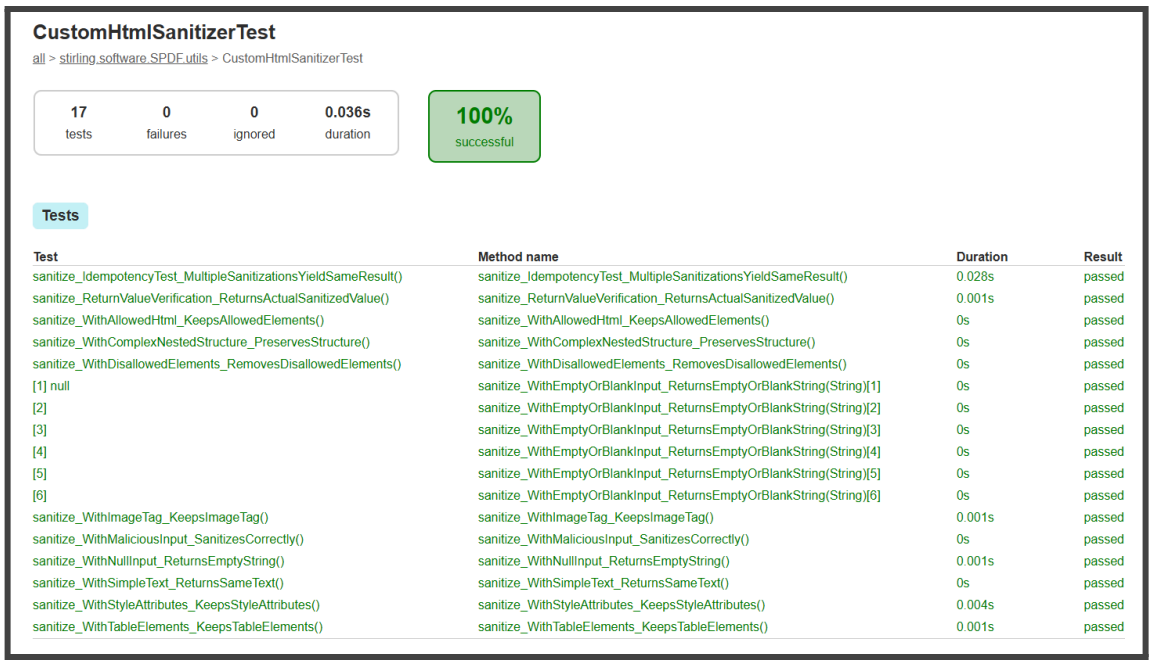
all > stirling.software.SPDF.utils > ErrorUtilsTest

7 tests	0 failures	0 ignored	0.006s duration	100% successful

Tests

Test	Duration	Result
testExceptionToModel()	0s	passed
testExceptionToModel2()	0s	passed
testExceptionToModelView()	0.001s	passed
testExceptionToModelView2()	0s	passed
testExceptionToModelView_NullMessage()	0.001s	passed
testExceptionToModel_NullMessage()	0s	passed
testStackTraceGeneration()	0.004s	passed

Example 2: CustomHtmlSanitizer



CustomHtmlSanitizerTest

all > stirling.software.SPDF.utils > CustomHtmlSanitizerTest

17 tests	0 failures	0 ignored	0.036s duration	100% successful

Tests

Test	Method name	Duration	Result
sanitize_IdempotencyTest_MultipleSanitizationsYieldSameResult()	sanitize_IdempotencyTest_MultipleSanitizationsYieldSameResult()	0.028s	passed
sanitize_ReturnValueVerification_ReturnsActualSanitizedValue()	sanitize_ReturnValueVerification_ReturnsActualSanitizedValue()	0.001s	passed
sanitize_WithAllowedHtml_KeepsAllowedElements()	sanitize_WithAllowedHtml_KeepsAllowedElements()	0s	passed
sanitize_WithComplexNestedStructure_PreservesStructure()	sanitize_WithComplexNestedStructure_PreservesStructure()	0s	passed
sanitize_WithDisallowedElements_RemovesDisallowedElements()	sanitize_WithDisallowedElements_RemovesDisallowedElements()	0s	passed
[1] null	sanitize_WithEmptyOrBlankInput_ReturnsEmptyOrBlankString(String)[1]	0s	passed
[2]	sanitize_WithEmptyOrBlankInput_ReturnsEmptyOrBlankString(String)[2]	0s	passed
[3]	sanitize_WithEmptyOrBlankInput_ReturnsEmptyOrBlankString(String)[3]	0s	passed
[4]	sanitize_WithEmptyOrBlankInput_ReturnsEmptyOrBlankString(String)[4]	0s	passed
[5]	sanitize_WithEmptyOrBlankInput_ReturnsEmptyOrBlankString(String)[5]	0s	passed
[6]	sanitize_WithEmptyOrBlankInput_ReturnsEmptyOrBlankString(String)[6]	0s	passed
sanitize_WithImageTag_KeepsImageTag()	sanitize_WithImageTag_KeepsImageTag()	0.001s	passed
sanitize_WithMaliciousInput_SanitizesCorrectly()	sanitize_WithMaliciousInput_SanitizesCorrectly()	0s	passed
sanitize_WithNullInput_ReturnsEmptyString()	sanitize_WithNullInput_ReturnsEmptyString()	0.001s	passed
sanitize_WithSimpleText_ReturnsSameText()	sanitize_WithSimpleText_ReturnsSameText()	0s	passed
sanitize_WithStyleAttributes_KeepsStyleAttributes()	sanitize_WithStyleAttributes_KeepsStyleAttributes()	0.004s	passed
sanitize_WithTableElements_KeepsTableElements()	sanitize_WithTableElements_KeepsTableElements()	0.001s	passed

Example 3: FileInfo

FileInfoTest

all > stirling.software.SPDF.utils > FileInfoTest

7	0	0	0.013s
tests	failures	ignored	duration

100% successful

Tests

Test	Method name	Duration	Result
1: fileSize=0	testGetFormattedFileSize(long, String)[1]	0.011s	passed
2: fileSize=1023	testGetFormattedFileSize(long, String)[2]	0.001s	passed
3: fileSize=1024	testGetFormattedFileSize(long, String)[3]	0.001s	passed
4: fileSize=1048575	testGetFormattedFileSize(long, String)[4]	0s	passed
5: fileSize=1048576	testGetFormattedFileSize(long, String)[5]	0s	passed
6: fileSize=1073741823	testGetFormattedFileSize(long, String)[6]	0s	passed
7: fileSize=1073741824	testGetFormattedFileSize(long, String)[7]	0s	passed

Final Results: (After Improvements based on mutation testing were implemented)

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
5	81%	47/58	86%	25/29	96%	25/26

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
stirling.software.SPDF.config	1	100%	2/2	100%	1/1	100%	1/1
stirling.software.SPDF.model	1	100%	13/13	86%	6/7	86%	6/7
stirling.software.SPDF.utils	3	74%	32/43	86%	18/21	100%	18/18

Reflection:

Achievements (What went well):

Overall, the project was a major success as we were successfully able to apply the knowledge we acquired in this course to improve the test quality of a mid-sized open-source software application. We were able to apply tests to a variety of classes using testing methods such as Input Space Partitioning (ISP), Graph-based testing (CFG and DFG), Logic-based testing, and Mutation testing. It was satisfying to see that the methods that we learned in class can help to improve the test quality of an application developed by a large open-source software team.

Another achievement worth mentioning is the successful collaboration that we were able to do to be able to apply the various testing methods. We were successfully able to split the tasks into manageable portions for each team member and established open communication channels in order to be on the same page. Each member was open and willing to help each other and answer any important questions that the group may have had.

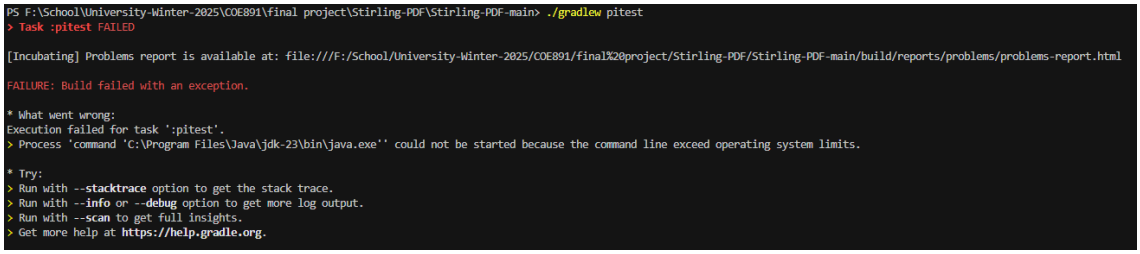
Difficulties and Barriers:

One challenge that we had to overcome was the size of the application. There were two issues due to this. The first is that the submission requirements stipulated a maximum of 10 MB for the zip file. The second was that when trying to run PIT Mutation Testing through the Pitest plugin on the full project, even on a single class within the entire project, it would attempt to import every single dependency, causing an error and failing as the project was immense and exceed the maximum command line argument that the Windows OS supported. Our workaround for this was creating a new project and importing only the required classes to accomplish our testing.

The biggest difficulty of this testing approach was the interconnected nature of the application. Many classes referenced each other or had multiple dependencies that made it difficult to test individual classes without needing to import large amounts of the codebase which brought us closer to the 10mb limit and to the limitations of Pitest.

Pitest Error:

This is an image of the error that we were confronted with when attempting to run Pitest on the full project:



> Process 'command 'C:\Program Files\Java\jdk-23\bin\java.exe'' could not be started because the command line exceed operating system limits.

Many hours were spent trying to reduce the classpath in multiple ways, but these attempts were futile. As a reminder, it does not matter how many classes you configure Pitest to test for; it uses the same classpath regardless. This is an image of half of the debugging stack trace. As you can see, the length of the classpath is immense because of the number of interconnected classes and dependencies:

Improvements (how we could improve the project):

We could improve the project by further increasing the test coverage to cover other classes for which the developers have not written tests. In doing this, there is a good chance that bugs could be uncovered, which could be reported to the developers. In the official Stirling PDF test folder, there are tests written already for many classes, but there are some omissions. For example, only two out of the nineteen classes in the \src\main\java\stirling\software\SPDF\controller\api folder have test classes. Uncovering bugs in these very important Controller classes could significantly improve the reliability of the project.

Another improvement that could be made is to improve the quality of the documentation. As we had never worked on this application before, there was a relatively large learning curve to understand some of the classes and methods in this project. Many of the implemented classes have little to no comments. Improving the quality of the comments and documentation would incentivize more new developers to contribute to the open-source software as it would be a lot more approachable to get into.