Click to verify

With this native Java API, you can easily compare documents and obtain the differences in the desired output format. Our Java library is fully self-contained and does not rely on any external tools or services. All document processing features are implemented in this powerful Java solution for a hassle-free experience. Document comparison is a highly sought-after procedure, particularly within automated document workflows. Whether you're working with legal documents, version control systems, or content management systems, the document comparison API for Java can be a game-changer. It compares the contents of documents both at the character level and at the word level. Even if only a single character has been changed, the entire word will be marked as modified. This allows you to detect the smallest changes that would be invisible to the human eye. Compare Word, PDF, web documents using Java There are occasions when you find yourself unsure about whether a document has been modified, and the process of manually comparing two versions of the document can be quite challenging. On the flip side, there are instances where you are confident that the document has been changed, but the task of visually locating the updated areas becomes exceedingly difficult. Let's explore some typical scenarios where automated document comparison can be incredibly useful: Legal Industry. Automating the comparison of contracts, agreements, and legal briefs can save valuable time and ensure accuracy, allowing legal teams to focus on more critical tasks Software Development. With this API, Java developers can effortlessly compare source code, requirements documents, and technical specifications, facilitating efficient version control and streamlined communication Quality Assurance. In industries such as publishing and content creation, ensuring consistency and accuracy across multiple document versions is crucial. This Java solution empowers QA teams to automatically compare drafts, manuscripts, or user manuals, pinpointing discrepancies and facilitating error-free document production Financial Services. Financial institutions deal with extensive documentation, including reports, statements, and contracts. With Java library, financial professionals can automate the comparison of financial statements, detect anomalies, and streamline compliance processes, enhancing operational efficiency Compare two documents programmatically in Java By integrating automatic document comparison into your workflows, you gain the ability to programmatically compare documents, extract differences, and instantly get results in the desired output format. Whether you're a seasoned developer or just getting started with Java, our comprehensive code snippets and online demonstration will guide you through the process. Try out our live demo by uploading two documents, selecting the target format to highlight the differences, and examining the Java code snippet displayed on the screen. This example demonstrates in detail how to perform document comparison programmatically and obtain the results in the required file format. An important point: the compared documents should not have revisions before calling the comparison method. You must first accept all the revisions. We have already taken care of this nuance in the Java code snippet below: Input files Upload the compared file Run code Upload the second file to compare Output format DOCX PDF DOC DOT DOCM DOTX DOTM RTF XPS Select the target format from the list import com.aspose.words.*; Document docA = new Document("Input1.docx"); Document docB = new Document("Input2.docx"); docA.acceptAllRevisions(); docB.acceptAllRevisions(); docA.compare(docB, "Author Name", new Date()); docA.save("Output.docx"); How to compare two text files in Java Install Aspose.Words for Java Add a library reference (import the library) to your Java project Load two documents to compare Accept all revisions before calling the compare() method Call the compare() method to compare two docs Call the Save() method, passing an output filename with required extension Get the result of compression as a separate file We host our Java packages in Maven repositories. 'Aspose.Words for Java' is a common JAR file containing byte-code. Please follow the step-by-step instructions on how to install it to your Java developer environment. Java SE 7 and more recent Java versions are supported. We also provide a separate package for Java SE 6 in case you are obliged to use this outdated JRE. Our Java package is cross-platform and runs on all operating systems with JVM implementation, including Microsoft Windows, Linux, macOS, Android and iOS. For information on optional package dependencies, such as JogAmp JOGL, Harfbuzz font engine, Java Advanced Imaging JAI, please refer to Product Documentation. When working with PDF documents, there are times when you need to compare the content of two documents to identify differences. The Aspose.PDF for .NET library provides a powerful toolset for this purpose. In this article, we'll explore how to compare PDF documents using a couple of simple code snippets. The comparison functionality in Aspose.PDF allows you to compare two PDF documents page by page. You can choose to compare either specific pages or entire documents. The resulting comparison document highlights differences, making it easier to identify changes between the two files. Here is a list of possible ways to compare PDF documents using Aspose.PDF for .NET library: Comparing Specific Pages - Compare the first pages of two PDF documents. Comparing Entire Documents - Compare the entire content of two PDF documents. Compare PDF documents graphically: Compare PDF with GetDifference method - individual images where changes are marked. Compare PDF with CompareDocumentsToPdf method - PDF document with images where changes are marked. Comparing Specific Pages The first code snippet demonstrates how to compare the first pages of two PDF documents. Steps: Document Initialization. The code starts by initializing two PDF documents using their respective file paths (documentPath1 and documentPath2). The paths are specified as empty strings for now, but in practice, you would replace these with the actual file paths. Comparison Process. Page Selection - the comparison is limited to the first page of each document ('getPages().get_Item(1)'). Comparison Options: 'setAdditionalChangeMarks(true)'- this option ensures that additional change markers are displayed. These markers highlight differences that might be present on other pages, even if they are not on the current page being compared. 'setComparisonMode(ComparisonMode.IgnoreSpaces)' - this mode tells the comparer to ignore spaces in the text, focusing only on changes within words. The resulting comparison document, which highlights the differences between the two pages, is saved to the file path specified in 'resultPdfPath'. private static void ComparingSpecificPages() { // The path to the documents directory String dataDir = "C:\\test\\"; // Open PDF documents Document document1 = new Document(dataDir + "ComparingSpecificPages1.pdf"); Document document2 = new Document(dataDir + "ComparingSpecificPages2.pdf"); com.aspose.pdf.comparison.sidebysidecomparison.SideBySideComparisonOptions options = new com.aspose.pdf.comparison.sidebysidecomparison.SideBySideComparisonOptions(); options.setAdditionalChangeMarks(true); options.setComparisonMode(com.aspose.pdf.comparison.sidebysidecomparison.ComparisonMode.IgnoreSpaces); com.aspose.pdf.comparison.sidebysidecomparison.SideBySidePdfComparer.compare(document1.getPages().get_Item(1), document2.getPages().get_Item(1), dataDir + "ComparingSpecificPages_out.pdf", options); document1.close(); document2.close(); } Comparing Entire Documents The second code snippet expands the scope to compare the entire content of two PDF documents. Steps: Document Initialization. Just like in the first example, two PDF documents are initialized with their file paths. Comparison Process. Entire Document Comparison - unlike the first snippet, this code compares the entire content of the two documents. Comparison Options - the options are the same as in the first snippet, ensuring that spaces are ignored, and additional change markers are displayed. The comparison result, which highlights differences across all pages of the two documents, is saved in the file specified by 'resultPdfPath'. private static void ComparingEntireDocuments() { // The path to the documents directory String dataDir = "C:\\test\\"; // Open PDF documents Document document1 = new Document(dataDir + "ComparingSpecificPages1.pdf"); Document document2 = new Document(dataDir + "ComparingSpecificPages2.pdf"); com.aspose.pdf.comparison.sidebysidecomparison.SideBySideComparisonOptions options = new com.aspose.pdf.comparison.sidebysidecomparison.SideBySideComparisonOptions(); options.setAdditionalChangeMarks(true); options.setComparisonMode(com.aspose.pdf.comparison.sidebysidecomparison.ComparisonMode.IgnoreSpaces); com.aspose.pdf.comparison.sidebysidecomparison.SideBySidePdfComparer.compare(document1, document2, dataDir + "ComparingSpecificPages_out.pdf", options); document1.close(); document2.close(); } The comparison results generated by these snippets are PDF documents that you can open in a viewer like Adobe Acrobat. If you use the Two-page view in Adobe Acrobat, you'll see the changes side by side: Deletions - these are noted on the left page. Insertions - these are noted on the right page. By setting 'AdditionalChangeMarks' to 'true', you can also see markers for changes that may occur on other pages, even if those changes aren't on the current page being viewed. Aspose.PDF for Java provides robust tools for comparing PDF documents, whether you need to compare specific pages or entire documents. By using options like 'AdditionalChangeMarks' and different 'ComparisonMode settings', you can tailor the comparison process to your specific needs. The resulting document provides a clear, side-by-side view of changes, making it easier to track revisions and ensure document accuracy. Compare PDF documents using GraphicalPdfComparer When collaborating on documents, especially in professional environments, you often end up with multiple versions of the same file. You can use the [GraphicalPdfComparer] class to compare PDF documents and pages. The class is suitable for comparing changes in a page's graphic content. With Aspose.PDF for Java, it's possible to compare documents and pages and output the comparison result to a PDF document or image file. You can set the following class properties: Resolution - resolution in DPI units for output images, as well as for images generated during the comparison. Color - the color of change marks. Threshold - change threshold in percent. The default value is zero. Setting a value other than zero allows you to ignore graphic changes that are insignificant to you. The class has a method that allows you to get page image differences in a form suitable for further processing: ImagesDifference GetDifference(Page page1, Page page2). This method returns an object of the [ImagesDifference] class, which contains an image of the first page being compared and an array of differences. The array of differences and the original image has the RGB24bpp pixel format. ImagesDifference allows you to generate a different image and get an image of the second page being compared by adding an array of differences to the original image. To do this, use the ImagesDifference.getDestinationImage() and ImagesDifference.differenceToImage() methods. Compare PDF with GetDifference method The provided code defines a method [GetDifference] that compares two PDF documents and generates visual representations of the differences between them. This method compares the first pages of two PDF files and generates two PNG images: One image (diffPngFilePath) highlights the differences between the pages in red. The other image (destPngFilePath) is a visual representation of the destination (second) PDF page. This process can be useful for visually comparing changes or differences between two versions of a document. private static void ComparePDFWithGetDifferenceMethod() throws IOException { // The path to the documents directory String dataDir = "C:\\test\\"; Document document1 = new Document(dataDir + "ComparingSpecificPages1.pdf"); Document document2 = new Document(dataDir + "ComparingSpecificPages2.pdf"); // Create comparer com.aspose.pdf.comparison.graphicalcomparison.GraphicalPdfComparer comparer = new com.aspose.pdf.comparison.graphicalcomparison.GraphicalPdfComparer(); // Compare com.aspose.pdf.comparison.graphicalcomparison.ImagesDifference imagesDifference = comparer.getDifference(document1.getPages().get_Item(1), document2.getPages().get_Item(1)); java.awt.image.BufferedImage diffImg = imagesDifference.differenceToImage(Color.getRed(), Color.getWhite()); // Specify the output file File outputFile = new File(dataDir + "ComparePDFWithGetDifferenceMethodDiffPngFilePath_out.png"); // Save the image as a PNG file ImageIO.write(diffImg, "png", outputFile); java.awt.image.BufferedImage destImg = imagesDifference.getDestinationImage(); //Specify the output DestinationImage file File destOutputFile = new File(dataDir + "ComparePDFWithGetDifferenceMethodDestPngFilePath_out.png"); // Save the image as a PNG file ImageIO.write(destImg, "png", destOutputFile); } Compare PDF with CompareDocumentsToPdf method The provided code snippet used the [CompareDocumentsToPdf] method, which compares two documents and generates a PDF report of the comparison results. private static void ComparePDFWithCompareDocumentsToPdfMethod() { // The path to the documents directory String dataDir = "C:\\test\\"; Document document1 = new Document(dataDir + "ComparingSpecificPages1.pdf"); Document document2 = new Document(dataDir + "ComparingSpecificPages2.pdf"); // Create comparer com.aspose.pdf.comparison.graphicalcomparison.GraphicalPdfComparer comparer = new com.aspose.pdf.comparison.graphicalcomparison.GraphicalPdfComparer(); comparer.setThreshold(3.0); comparer.setColor(Color.getBlue()); comparer.setResolution(new com.aspose.pdf.devices.Resolution(300)); // Compare comparer.compareDocumentsToPdf(document1, document2, dataDir + "compareDocumentsToPdf_out.pdf"); } What are the most effective approaches for comparing two PDF files using Java? Comparing PDF files in Java involves determining differences in content, structure, or layout between two PDF documents. Several libraries available in Java can facilitate this process, enabling developers to implement robust comparison functionalities in their applications. Copied import org.apache.pdfbox.pdmodel.PDDocument; import org.apache.pdfbox.text.PDFTextStripper; public class PDFComparator { public static String getTextFromPDF(String fileName) throws Exception { PDDocument document = PDDocument.load(new java.io.File(fileName)); PDFTextStripper pdfStripper = new PDFTextStripper(); String text = pdfStripper.getText(document); document.close(); return text; } public static boolean comparePDFs(String pdf1, String pdf2) throws Exception { String text1 = getTextFromPDF(pdf1); String text2 = getTextFromPDF(pdf2); return text1.equals(text2); } } Causes Content Variation: Text differences can occur due to formatting, font changes, or even different encoding. Structural Differences: Changes in document layout, such as page order or embedded elements, can also lead to discrepancies. Annotations and Comments: Differences in annotations and comments added to the PDFs may not be immediately obvious. Solutions Apache PDFBox: A popular open-source library that can extract text and compare it programmatically. iText: This commercial library allows for complex PDF manipulation and can be used to compare documents with extensive methods. DiffPDF: While not a Java library, this tool can be integrated with Java applications via command line operations, providing visual comparisons. Mistake: Not handling PDF exceptions properly Solution: Ensure to use try-catch blocks to handle IOExceptions and other potential exceptions when dealing with PDF files. Mistake: Ignoring differences in encoding Solution: Be aware that different variations in text encoding may lead to incorrect comparison results. Mistake: Failing to consider layout differences Solution: If layout is important, consider using libraries that support visual comparisons instead of solely text-based methods. compare PDF files in Java Java PDF comparison library PDFBox comparison iText PDF comparison detect PDF differences © Copyright 2025 - CodingTechRoom.com Closed. This question needs to be more focused. It is not currently accepting answers. Want to improve this question? Update the question so it focuses on one problem only by editing this post. i need to write a java class that compares two pdf files and points out the differences(differences in text/position/font) using some sort of highlighting. my initial approach was use pdfbox to parse the file using pdfbox and store the extracted text using in some data structure that would help me with comparing. Is there any java library that can extract the text,preserve the formatting,help me with indexing and comparing.Can i use tika/ google's diff-match for this. tika extracts text in the form of xhtml but how can i compare two xhtml files? 1 This tutorial explains how to compare PDF files using Java. It includes details on setting the IDE, a list of steps, and sample code for working with PDF comparison tools using Java. You will learn to compare individual pages or complete PDF files based on your requirements.Steps to Compare PDF Documents using JavaSet the environment to use Aspose.PDF for Java to compare PDF documentsLoad two PDF files separately into the Document class objectsAccess the target pages from the collection of pages of both the PDF files for comparisonCall the TextPdfComparer.comparePages() method to compare selected pages from both the PDF filesCompare complete PDF documents using the TextPdfComparer.compareFlatDocuments() methodSave the comparison result into another PDF file using the above methodThese steps summarize the process to compare two PDF docs using Java. You may load the two PDF files for comparison and select the individual pages if you want to compare only the selected pages. Finally, call the compareFlatDocuments() method to compare the complete documents or the comparePages() method to compare the selected pages, along with setting the output PDF file name to save the differences.Code for PDF to PDF Comparison using JavaThis code can be used to develop a simple PDF document comparison tool using Java. Use the ComparisonOptions class object to customize the comparison, such as defining the included/excluded rectangular areas during the comparison, setting a flag for excluding tables, and setting the edit operations orders. The EditOperationsOrder enumerator, which has values InsertFirst or DeleteFirst, can be used.This article has taught us the process to compare two PDFs using Java. If you want to crop a PDF file, refer to the article on how to crop a PDF using Java. Closed. This question needs to be more focused. It is not currently accepting answers. Want to improve this question? Update the question so it focuses on one problem only by editing this post. i need to write a java class that compares two pdf files and points out the differences(differences in text/position/font) using some sort of highlighting. my initial approach was use pdfbox to parse the file using pdfbox and store the extracted text using in some data structure that would help me with comparing. Is there any java library that can extract the text,preserve the formatting,help me with indexing and comparing.Can i use tika/ google's diff-match for this. tika extracts text in the form of xhtml but how can i compare two xhtml files? 1 Closed. This question needs to be more focused. It is not currently accepting answers. Want to improve this question? Update the question so it focuses on one problem only by editing this post. i need to write a java class that compares two pdf files and points out the differences(differences in text/position/font) using some sort of highlighting. my initial approach was use pdfbox to parse the file using pdfbox and store the extracted text using in some data structure that would help me with comparing. Is there any java library that can extract the text,preserve the formatting,help me with indexing and comparing.Can i use tika/ google's diff-match for this. tika extracts text in the form of xhtml but how can i compare two xhtml files? 1 In this quick tutorial, we demonstrate to you the step-by-step process to compare PDF files using Java. It doesn't require you to install any third-party software and quickly develop the capability for PDF comparison in Java by consuming a few API calls. Here you can find complete instructions and functional example for comparing PDF documents.Steps to Compare PDF Files using JavaInstall Conholdate.Total for Java from the Maven repository in your Java application to compare PDF filesImport the essential classes for performing PDF comparison using JavaCreate an instance of the Comparer class and pass the source PDF file to its constructorCall the add method of the Comparer class and specify the target file path to compare PDF documents using JavaFinally, call the compare method and pass the resultant file pathWe have outlined the steps above for comparing two PDF files for differences using Java. You just need to follow these steps in a order to start developing the functionality to compare two PDF files in Java. However, these instructions can be used on other platforms including MS Windows, Linux, and Mac OS.Code to Compare PDF Files using JavaIn the working example above, you can see the Java code to compare two PDF files. We have developed a sample code for performing the comparison of two PDF documents. Additionally, you may add more documents for comparison by repeating step 4 in the code. You can also use various other document formats for comparisons such as DOCX, XLSX, PPTX, VSDX, ODT, PNG, and many more. A simple Java library to compare two PDF files. Files are rendered and compared pixel by pixel. There is no text comparison. Just include it as a dependency. Please check for the most current version available: de.redsix pdfcompare ... There is a simple interactive UI, when you start the jar file without any additional arguments (which starts the class de.redsix.pdfcompare.Main). It allows you to choose files to compare and also to mark areas to ignore and write those to an ignore-file. Next to the UI you can provide an expected and actual file and additional parameter via a CLI. To get a help for the CLI use the -h or --help option-. usage: java -jar pdfcompare-x.x.x-full.jar [EXPECTED] [ACTUAL] -h,--help Displays this text and exit ... But the focus of PdfCompare is on embedded usage as a library. new PdfComparator("expected.pdf", "actual.pdf").compare().writeTo("diffOutput"); This will produce an output PDF which may include markings for differences found. PdfCompare renders a page from the expected.pdf and the same page from the actual.pdf to a bitmap image and compares these two images pixel by pixel. Pixels that are equal are faded a bit. Pixels that differ are marked in red and green. Green for pixels that where in the expected.pdf, but are not present in the actual.pdf. Red for pixels that are present in the actual.pdf, but were not in the expected.pdf. And there are markings at the edge of the paper in magenta to find areas that differ quickly. Ignored Areas are marked with a yellow background. Pages that were expected, but did not come are marked with a red border. Pages that appear, but were not expected are marked with a green border. The compare-method returns a CompareResult, which can be queried: final CompareResult result = new PdfComparator("expected.pdf", "actual.pdf").compare(); if (result.isNotEqual()) { System.out.println("Differences found!"); } if (result.isEqual()) { System.out.println("No Differences found!"); } if (result.hasDifferenceInExclusion()) { System.out.println("Differences in excluded areas found!"); } result.getDifferences(); // returns page areas, where differences were found For convenience, writeTo also returns the equals status: boolean isEquals = new PdfComparator("expected.pdf", "actual.pdf").compare().writeTo("diffOutput"); if (!isEquals) { System.out.println("Differences found!"); } The compare method can be called with filenames as Strings, Files, Paths or InputStreams. It is also possible to define rectangular areas that are ignored during comparison. For that, a file needs to be created, which defines areas to ignore. The file format is JSON (or actually a superset called HOCON) and has the following form: exclusions: [{ page: 2 x1: 300 // entries without a unit are in pixels. Pdfs are rendered by default at 300DPI y1: 1000 x2: 550 y2: 1300 }, { // page is optional. When not given, the exclusion applies to all pages. x1: 130.5mm // entries can also be given in units of cm, mm or pt (DTP-Point defined as 1/72 Inches) y1: 3.3cm x2: 190mm y2: 3.7cm }, { page: 7 // coordinates are optional. When not given, the whole page is excluded. }] When the provided exclusion file is not found, it is ignored and the compare is done without the exclusions. Exclusions are provided in the code as follows: new PdfComparator("expected.pdf", "actual.pdf").withIgnore("ignore.conf").compare(); Alternatively an Exclusion can be added via the API as follows: new PdfComparator("expected.pdf", "actual.pdf") .withIgnore(new PageArea(1, 230, 350, 450, 420)) .withIgnore(new PageArea(2)) .compare(); When you want to compare password protected PDF files, you can give the password to the Comparator through the withExpectedPassword(String password) or withActualPassword(String password) methods respectively. new PdfComparator("expected.pdf", "actual.pdf") .withExpectedPassword("somePwd") .withActualPassword("anotherPwd") .compare(); PdfCompare can be configured with a config file. The default config file is called "application.conf" and it must be located in the root of the classpath. PdfCompare uses Lightbend Config (previously called TypeSafe Config) to read its configuration files. If you want to specify another configuration file, you can find out more about that here: . In particular you can specify a replacement config file with the -Dconfig.file=path/to/file command line argument. Alternatively you can specify parameters either through a system environment variables or as a Jvm parameter with -DvariableName= Another way to specify a different config location programmatically is to create a new ConfigFileEnvironment(...) and pass it to PdfCompare.withEnvironment(...). All the settings, that can be changed through the application.conf file can also be changed programmatically through the API. To do so you can use the following code: new PdfComparator("expected.pdf", "actual.pdf") .withEnvironment(new SimpleEnvironment() .setActualColor(Color.green) .setExpectedColor(Color.blue)) .compare(); The SimpleEnvironment delegates all settings, that were not assigned, to the default Environment. Through the environment you can configure the memory settings (see above) and the following settings: DPI=300 Sets the DPI that Pdf pages are rendered with. Default is 300. expectedColor=00B400 (GREEN) The expected color is the color that is used for pixels that were expected, but are not there. The colors are specified in HTML-Stlye format (without a leading '#'): The first two characters define the red-portion of the color in hexadecimal. The next two characters define the green-portion of the color. The last two characters define the blue-portion of the color to use. actualColor=D20000 (RED) The actual color is the color that is used for pixels that are there, but were not expected. The colors are specified in HTML-Stlye format (without a leading '#'): The first two characters define the red-portion of the color in hexadecimal. The next two characters define the green-portion of the color. The last two characters define the blue-portion of the color to use. tempDir=System.property("java.io.tmpdir") Sets the directory where to write temporary files. Defaults to the java default for java.io.tmpdir, which usually determines a system specific default, like /tmp on most unix systems. allowedDifferenceInPercentPerPage=0.2 Percent of pixels that may differ per page. Default is 0. If for some reason your rendering is a little off or you allow for some error margin, you can configure a percentage of pixels that are ignored during comparison. That way a difference is only reported, when more than the given percentage of pixels differ. The percentage is calculated per page. Not that the differences are still marked in the output file, when you addEqualPagesToResult. parallelProcessing=true When set to false, disables all parallel processing and process everything in a single thread. addEqualPagesToResult=true When set to false, only pages with differences are added to the result and this the resulting difference PDF document. failOnMissingIgnoreFile=false When set to true, a missing ignore file leads to an exception. Otherwise it is ignored and only an info level log messages is written. There are a few different Implementations of CompareResults with different characteristics. The can be used to control certain aspects of the system behaviour, in particular memory consumption. It is good to know a few internals, when using the PdfCompare. Here is in a nutshell, what PdfCompare does, when it compares two PDFs. PdfCompare uses the Apache PdfBox Library to read and write Pdfs. The Two Pdfs to compare are opened with PdfBox. A page from each Pdf is read and rendered into a BufferedImage by default at 300dpi. A new empty BufferedImage is created to take the result of the comparison. It has the maximum size of the expected and the actual image. When the comparison is finished, the new BufferedImage, which holds the result of the comparison, is kept in memory in a CompareResult object. Holding on to the CompareResult means, that the images are also kept in memory. If memory consumption is a problem, a CompareResultWithPageOverflow or a CompareResultWithMemoryOverflow can be used. Those classes store images to a temporary folder on disk, when certain thresholds are reached. After all pages are compared, a new Pdf is created and the images are written page by page into the new Pdf. So comparing large Pdfs can use up a lot of memory. I didn't yet find a way to write the difference Pdf page by page incrementally with PdfBox, but there are some workarounds. There are currently two different CompareResults, that have different strategies for swapping pages to disk and thereby limiting memory consumption. CompareResultWithPageOverflow - stores a bunch of pages into a partial Pdf and merges the resulting Pdfs in the end. The default is to swap every 10 pages, which is a good balance between memory usage and performance. CompareResultWithMemoryOverflow - tries to keep as many images in memory as possible and swaps, when a critical amount of memory is consumed by the JVM. As a default, pages are swapped, when 70% of the maximum available heap is filled. A different CompareResult implementation can be used as follows: new PdfComparator("expected.pdf", "actual.pdf", new CompareResultWithPageOverflow()).compare(); Also there are some internal settings for memory limits, that can be changed. Just add a file called "application.conf" to the root of the classpath. This file can have some or all of the following settings to overwrite the defaults given here: imageCacheSizeCount=30 How many images are cached by PdfBox maxImageSizeInCache=100000 A rough maximum size of images that are cached, to prevent very big images from being cached mergeCacheSizeMB=100 When Pdfs are partially written and later merged, this is the memory cache that is configured for the PdfBox instance that does the merge. swapCacheSizeMB=100 When Pdfs are partially written, this is the memory cache that is configured for the PdfBox instance that does the partial writes. documentCacheSizeMB=200 This is the cache size configured for the PdfBox instance, that loads the documents that are compared. parallelProcessing=true When set to false, disables all parallel processing and process everything in a single thread. overallTimeoutInMinutes=15 Set the overall timeout. This is a safety measure to detect possible deadlocks. Complex comparisons might take longer, so this value might have to be increased. executorTimeoutInSeconds=60 Sets the timeout to wait for the executors to finish after the overallTimeout was reached. It's unlikely that you ever need to change this. So in this default configuration, PdfBox should use up to 400MB of Ram for it's caches, before swapping to disk. I have good experience with granting a 2GB heap space to the JVM. Big thanks to Chethan Rao meetchethan@gmail.com for helping me diagnose out of memory problems and providing the idea of partial writes and merging of the generated PDFs. Closed. This question needs to be more focused. It is not currently accepting answers. Want to improve this question? Update the question so it focuses on one problem only by editing this post. i need to write a java class that compares two pdf files and points out the differences(differences in text/position/font) using some sort of highlighting. my initial approach was use pdfbox to parse the file using pdfbox and store the extracted text using in some data structure that would help me with comparing. Is there any java library that can extract the text,preserve the formatting,help me with indexing and comparing.Can i use tika/ google's diff-match for this. tika extracts text in the form of xhtml but how can i compare two xhtml files? 1