Software Design Document

for

iText Based PDF Viewer

Version 1.0

Prepared by

Dhruvkumar Patel
Jace Robinson
Pranav Pranav

CS 7140 Advanced Software Engineering
July 13th 2016

Table of Contents

1. Introduction

This is a software design document for the product, *iText Based PDF Viewer*. This document will significantly refer to the specification document of the same product [7].

1.1 Purpose

The purpose of this document is to completely describe and define the design for the *iText Based PDF Viewer* product. This design must meet the specification in the previous document [7]. This application will allow users to view and annotate files following the Portable Document Format (PDF) version 1.7 [4]. The viewer will be a subset of functionality and comparable quality of popular PDF viewers *PDF-xchange-editor* [2] and *XODO* [3].

iText Based PDF Viewer will be developed using the free and open source library iText [1]. This extensive library can assist in manipulating the PDF internals while using the Java programming language. As a classroom project, the requirements, specification, design, implementation, and testing documents will all be created for this product.

1.2 Intended Audience and Reading Suggestions

There are several types of readers who may view this document. Below are various categories of users along with suggested sections to read.

Developers:
A developer will be interested in learning the specifications necessary to create the product. This group is suggested to read sections 2.2, 2.5, and all of section 3. Section 2.2 will give a high level summary of the functionality, section 2.5 will discuss important constraints, and section 3 lists the formal specifications.

Project Manager:
A project manager will want to understand a bigger picture view of the product in order to manage the work. He or she is encouraged to read all of section 2. This will give a big picture view of the product, users and constraints.

Users:
Users of the product may read this document for a number of reason such as wanting to know some motivation for the product as discussed in section 2.1, or a list of functionality in section 2.2.

Students:

Students may read this document as an example of requirements document. For educational purposes, it is suggested the student reads the entire document to gain the most understanding.

1.3 Product Scope

The goal of this project is to develop a Java and iText based PDF viewer. The viewer should be able to open files following the PDF format. The user should also be able to annotate the PDF, and save these changes with the file. For more detailed description of functionality see sections 2.2 and 3.

As this is an educational project, the scope will be severely limited when compared to a professional product. The primary focus of the project is to develop the documentation for a software life cycle of requirements, specification, design, implementation, and testing. The product will not allow replacement of text at the level of lines, words, and paragraphs. The viewer is only expected to display textual content. This means the product will not be able to display vector graphics, raster images, or any other types of content commonly stored in PDF. The product will not support hand drawn annotations such as created by a stylus pen on a tablet device.

The product must be built using the iText library to handle transfer of information between PDF and this viewer. iText was chosen due to size, popularity, and quality of user documentation. Some of the other choices considered were jPod, PDFBox, and ICEpdf. An investigation comparing and contrasting the various PDF libraries was not performed and the choice of iText should not imply higher quality.

1.4 References

[1] "iText Developers", *Developers.itextPDF.com*, 2016. [Online]. Available: http://developers.itextPDF.com/. [Accessed: 15- Jun- 2016].

[2] "Tracker Software Products :: PDF-XChange Editor", *Tracker-software.com*, 2016. [Online]. Available: https://www.tracker-software.com/product/PDF-xchange-editor. [Accessed:15- Jun- 2016].

[3] "XODO PDF Reader & Annotator", *Xodo.com*, 2016. [Online]. Available: http://xodo.com/. [Accessed: 15- Jun- 2016].

[4] *PDF Reference*, 6th ed. Adobe Systems Incorporated, 2006.

[5] "Doxygen: Main Page", *Stack.nl*, 2016. [Online]. Available: http://www.stack.nl/~dimitri/doxygen/. [Accessed: 16- Jun- 2016].

[6] D. Patel, J. Robinson, and P. Pranav, “Software Requirements Document for iText Based PDF Viewer”, June 2016.

[7] D. Patel, J. Robinson, and P. Pranav, “Software Specification Document for iText Based PDF Viewer”, June 2016.

1.5 Document Terminology

“PDF File” - For the purposes of this Requirements Document, “PDF file” refers to a file format in compliance with the standards defined in the PDF reference document version 1.7 [4].

"text" - For the purposes of this Requirements Document, "text" refers to "Text Objects" described by Section 5.3 of the PDF reference document [4] with the added restriction that ”text" is composed solely of sequences of Latin characters irrespective of font specified in the PDF for display of the characters. Latin characters are defined in Appendix D.1 of the PDF reference [4].

"annotation" - refers to a subset of annotations described in section 8.4 of the PDF reference document [4]. The subset chosen are underline, highlight, strikethrough, rectangular box, and comment. Table 8.28 of the PDF reference [4] describes rectangular box, and table 8.30 of the PDF reference [4] describe underline, highlight and strikethrough, and table 8.23 describes comment. A visual example of each annotation is given in figure 3.2.1 and figure 3.2.2 of this document.

“upload” - refers to the display of only text and annotations encoded within PDF documents selected from within a file system by a user. The text will display to the output monitor device of the computer.

“save” - refers to storing PDF documents with user selected file names at user selected file system locations.

“insert” - “insert” refers to adding content to the current state of the PDF document.

“delete” - refers to removing content from the current state of the PDF document.

“location” - refers to the position in the grid defined in section 4.2 of the PDF reference document [4].

“search” - refers to matching input text with all occurrences of text contained in the current page of document such that textInput = textDocument. The locations of the matching text within the document are determined.

“PDF page” - refers to a grouping of PDF content as defined in section 3.6 of PDF reference document [4].

1.6 Definitions, Acronyms, and Abbreviations

JAR	Java Archive
JDK	Java Development Kit

KISS	Keep it Simple Stupid
PDF	Portable Document Format
UI	User Interface

1.7 Figures

2. Design Considerations

This section describes the issues considered in determining the design.

2.1. Assumptions and Dependencies

There are a few critical dependencies for this product that are listed below.

1. The viewer is developed to display PDF version 1.7 formatted files [4]. It is assumed that future versions of PDF will remain backwards compatible with this version.
2. The viewer is developed using free and open source library developed by iText. It is assumed the library remains free in the future.
3. The viewer is to be developed using JavaFX available in Java 8. It is assumed current and future JRE's past version 1.8.0_91 will be backwards compatible and execute the deliverable .jar file.
4. As a test to the saving and uploading of PDF documents, the PDF viewer XODO [3] will be used. A requirement defined in section 3 is a PDF file created in this product must be viewable in XODO. It is assumed that XODO will continue to be available during the development process. It is also assumed XODO will continue to follow PDF standards.

2.2. General Constraints

There are several constraints affecting the development of this product. The constraints are listed below.

1. The product must be developed in six weeks. Due to this short time frame, the scope has been severely reduced.

2. The software development life cycle will follow a waterfall model. This model has the sequence of deliverables of requirements, specifications, design, implementation, and testing.

2.3. Goals and Guidelines

There were three main design goals for this product.

1) Keep it Simple Stupid (KISS)
2) Design by Contract
3) Maximize Use of iText

For the first design goal, a motivation for this project is the development of the software development lifecycle documents. The code is not intended for professional use, and therefore does have extensive development for a variety of use cases.

In the second design goal, we want to emphasize the principles of design by contract. Therefore, the architecture was designed in a way to give precise responsibility for various classes.

The third design goal is to maximize the use of iText software library. Significant effort has been spent to learn and understand the possible usability of iText in the context of this product. Many of the core functions are implemented in iText.

3. System Architecture

This section provides information related to system architecture which contains major components and their structures and how they are interact or related with each other. Software architecture provides whole system overview and consider following points as advantages:

1) Describe the structure of the system but hide the implementation details.
2) Includes Data Flow and UML diagram to describes all the components and their interactions with each other.
3) satisfy both functional and quality requirements.
4) In figure 3.2 is the UML Class diagram describes all the classes with attributes and operations.

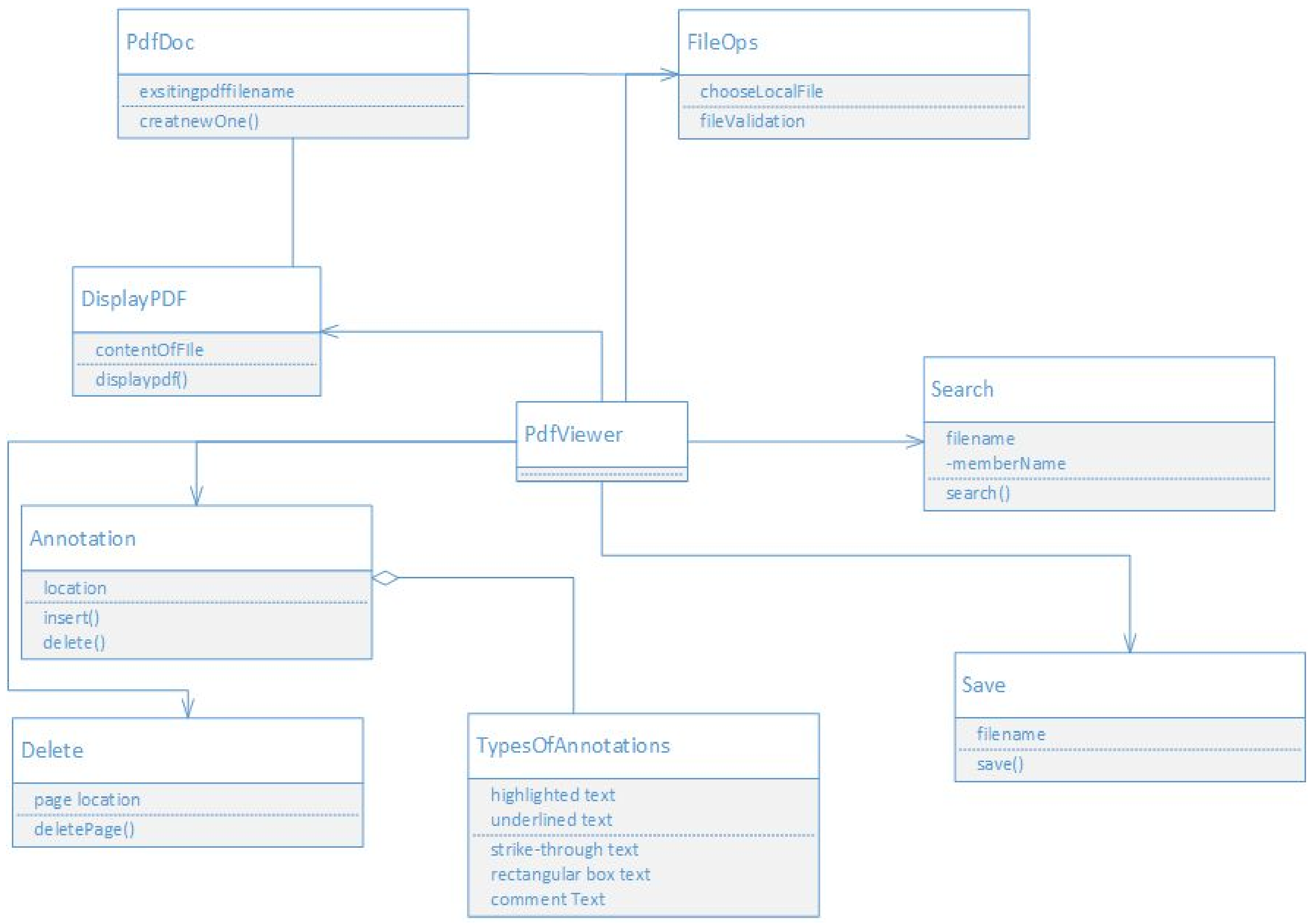

Figure 3.1.1: UML class diagram

Here, in Class diagram TypesOfAnnotations class is a strong part of Annotation class so it describes with aggregation relationship and all the other are connected with direct connection link.

5) Here is the Work Flow Diagram

Work Flow Diagram describes the whole flow of the system. It is start with the initial step and follow each and every step to develop all the features.

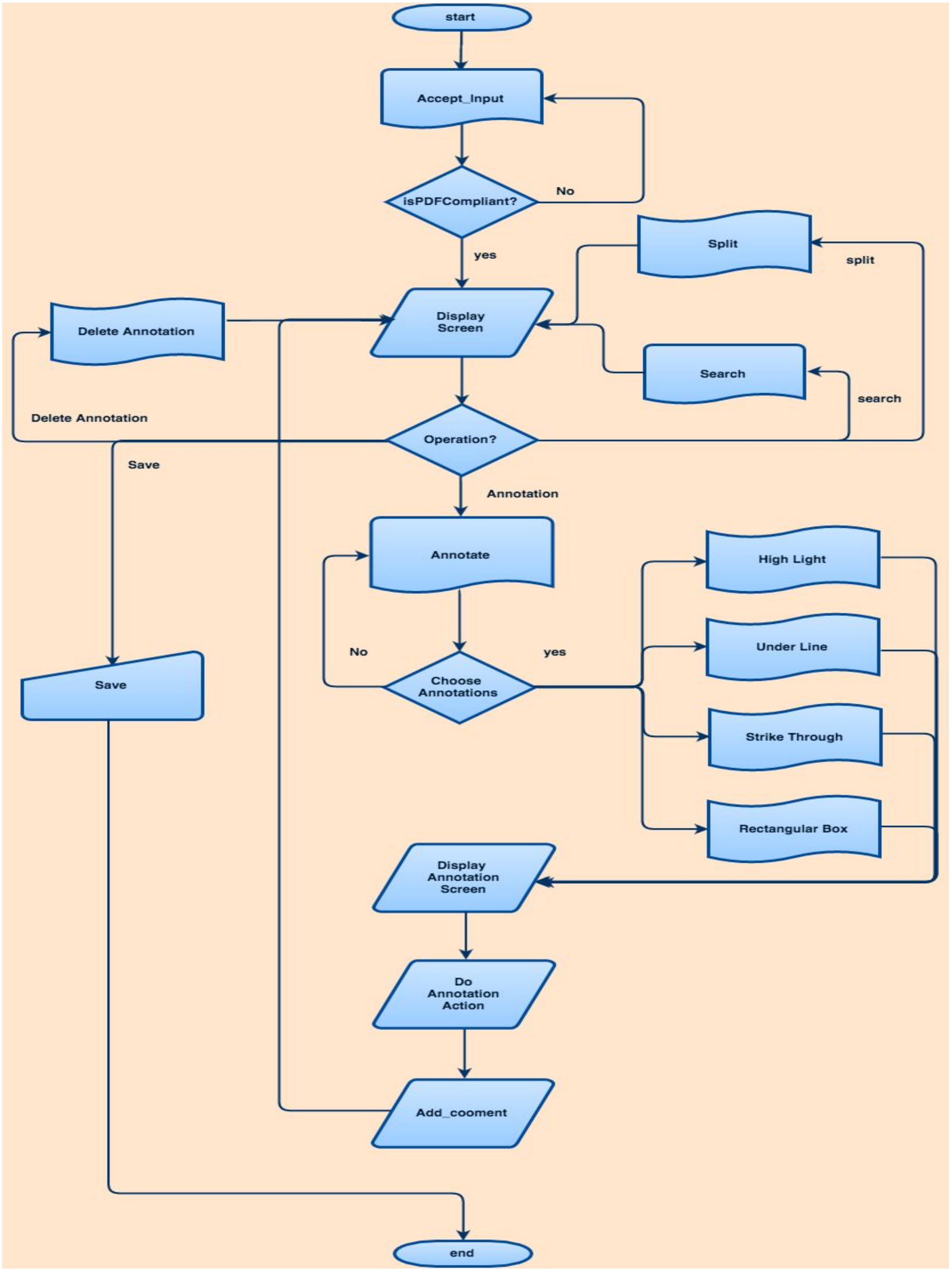

Figure 3.1.2: Work Flow Diagram

Steps of Work Flow Diagram:

1) Start with take pdf file as an input from local machine.
2) Check the file validation it must be a pdf file otherwise exit.
3) Display pdf file in javafx scrollpane screen.
4) Now one can select operations to perform specific things and display the document with particular operation succeeded.
    1) search option to search particular word and it will display with highlighted text
    2) split option to arrange one document by pages
    3) To Insert an Annotation select particular type of annotation among types of annotation.
    4) display pdf on the screen with particular annotation succeeded.
    5) To delete an annotation or delete a page with annotations.
    6) Save option is to save file with existing file name and exit.

4. Detailed System Design

Additional details on each class as introduced in section 3 are provided below. The purpose of each class, its class invariants, and pseudo code for each function are given. Given that the design is built around iText, functions that are provided by iText will be referenced as iText.function(args).

4.1. PDFViewer

The PDFViewer class is the starting object for this program. The display is started from here.

```
Function run() {
displayPDF()
}
```

4.2. PDFDoc

The PDFDoc class maintains the content of the PDF. This content includes all text and annotation elements.

The class invariant is the list of TextElements must not increase size. TextElements can be deleted as entire PDF pages are deleted, but content cannot be edited.

Fields:
List iText.TextElements

List iText.AnnotationElements

Functions:

```
addAnnot(annot) {
    Add Annot to AnnotationElements
}

deleteAnnot(cursor) {
    if(intersect(cursor, iText.AnnotationElements))
        removeAnnotation from AnnotationElements
}
```

4.3. DisplayPDF

The DisplayPDF class creates the external interface visible by the user. The user will interact with the interface using a cursor. The cursor can be used to select one of the several GUI buttons of save, search, upload, or a comment area of an annotation. The interaction with a GUI element will trigger the corresponding event.

The class invariant is all elements of PDFDoc must be within bounds of grid.

Fields:
PDFDoc
grid[MAX_x][MAX_y]
cursorLocation (x,y)
saveField
searchField
uploadField

Functions:

```
displayPDF() {
    Initialize GUI elements
}

displayPDF(PDFDoc) {
    For all elements in PDFDoc
        Grid[element.location.x][element.location.y] = element.content
}

moveCursor(cursorLoc) {
    cursorLocation = cursorLoc
}
```

```
saveEvent(saveField.text) {
        fileName = saveField.text
        Save.save(fileName)
}

searchEvent(searchField.text) {
        Keyword = searchField.text
        Search.search(keyword)
}

annotationEvent(annotType) {
        Annotation.insert(annotType)
}

uploadEvent(uploadField.text) {
        fileName = uploadField.text
        FileOps.extract(fileName)
}
```

4.4. Annotation

The annotation class describes a single annotation. The annotation will have a 2D position within the display maintained by field Location[]. The user has the ability to insert annotations of a specific type and at a specific location (x,y). The user also has the ability to delete annotations at a specific location if they exist.

The class invariant is location (x,y) must be within display bounds of 0 <= x <= DisplayPDF.max_x and 0 <= y <= DisplayPDF.max_y

Fields:
Location = (float x, float y)
PDFDoc

Functions:

```
insert(annotType, cursorPosition) {
if(annotType == highlightedText ||
annotType == underlinedText ||
annotType == strikethroughText)
   PDFDoc.addAnnot(new iText.PDFTextMarkupAnnotation(cursorPosition, annotType))
  } else if (annotType == rectangularBox) {
   PDFDoc.addAnnot(new iText.PDFSquareAnnotation(cursorPosition))
  }
```

```
}

delete(cursorPosition) {
PDFDoc.removeAnnot(cursorPosition)
}
```

4.5. Save

The save class handles the saving of PDF file to the disk and exiting the process.

Filename

```
save(PDFDoc, filename) {
    savePDF to disk
    exit()
}
```

4.6. Delete

The delete class will handle the deletion of individual pages from the PDFDoc.

Fields:
pageLocation

Functions:

```
deletePage(pageNumber) {
    For all elements in PDF Doc
        If element page number == pageNumber
            Remove element from PDFDoc
}
```

4.7. Search

The search class handles the searching operation, such that a keyword is searched for in the document, and the cursor is updated to the first location if it exists.

Fields:
PDFDoc

Functions:

```
search(keyword) {
    For all elements in PDFDoc
        if(element.content == keyword)
```

```
                displayPDF.moveCursor(element.location)
}
```

4.8. FileOps

The fileOps class handles the uploading from document and checking for validity of PDF format.

Functions:

```
extract(input_filename) {
    If(input_filename exists) {
        File = readFile(input_filename)
        if (iText.validPDF(file))
            PDFDoc = file
        Else
            PDFDoc = null
    }
}
```

5. Author Journals

Below are the journals of each of the authors maintained throughout the development process.

5.1 Dhruvkumar Patel

Start Day 7-11-16

we decided to implement specification with the same template of document what we used for Requirement documents.
I learned lifecycle of software engineering system how the system is build? in real world. As of now what i have developed in my career i just start with the
analysis and then directly move on implementation part.

Start Day 7-12-16
we start with specification document and discussed all the math logic, pre and postcondition of all the objects used throughout specification document.
also start to think about design of the system we discussed classes, attributes and operations of the system.
designed UML and Data Flow Diagram using Microsoft Visio tool.
learned how to design whole system and documents.

meanwhile we started coding to make product and which will be helpful to write implementation documents.
discussed design of all the features what we mentioned in the requirement and specification document.
start writing specification and design document.

<u>Start Day 7-13-16</u>
We had a difficulty to display pdf in javafx scrollpane but, then piazza discussion and talked with jace and pranav. Atlast we reached the solutions.
we all divided the section of both documents and we did within time limit.

5.2 Jace Robinson

Note since the specs and design were developed at same time, I used the same journal for both.

<u>Start Day 7-11-16</u>
Today we are beginning the specification document. The first task is to find and understand an appropriate template. I am confident in our requirements document so now it is a matter of writing a spec which meets this. We decided to use basically the same template as requirements but create specifications that meet each requirement.
<u>Start Day 7-12-16</u>
Today we are beginning the design document. We have a rough outline of the main components of our viewer.
A pdf is a collection of objects completely describing the content. Specifically, there exist *text objects* which completely describe the text (this includes font).
PDF *Document Structure* describes pages, fonts, and annotations.
3.4 File Structure contains body which make up document. Sequence of indirect objects
File Structure
How is text stored?? Text is font, position, and characters (text objects)
Glyphs are organized into fonts
Character is abstract entity such as “A” (and not its font)
How does text “wrap” and not extend past border of page?? Using matrix operations to determine “text space”
Annotations
Text Markup Annotation (highlight, underline, strikethrough), 4 (x,y) pairs describing coordinates of rectangle to markup (text will be in the rectangle).

Square and Circle Annotation – rectangle is represented as difference of two rectangles, along with their positions

Annotation are separate objects from the text, they are simply given the same coordinates.

How are pages determined? A page is a collection of content stream

Entire pdf is collection of pages (with other stuff but not relevant to us)

Now looking at iText

Relevant Classes/Functions:

Canvas: Where content is placed, does not know what page it is

Document: root object

DocumentRenderer:

LayoutPosition:

Paragraph: self contained block of text

IElement: takes physical space on canvas

PDFAnnotation

PDFArray

PDFDocument

PDFMarkupAnnotation

PDFPage

PDFSquareAnnotation

PDFTextMarkupAnnotation

Rectangle

Examples and APIs worth keeping track of:

com.itextpdf.kernel.pdf.annot

com.itextpdf.kernel.pdf.canvas

TextMarkup example (with highlight)

http://developers.itextpdf.com/content/itext-7-jump-start-tutorial/chapter-4-making-pdf-interactive

TextExtractionStrategy() to read text from pdf

http://developers.itextpdf.com/examples/content-extraction-and-redaction/parsing-pdfs

PDFStamper to use modify/create new pdf from existing pdf

Start Day 7-13-16

First task is to continue writing specification. I have decided to overhaul a large portion of our existing specification to give more rigorous “math” solution. The main idea to array of pages, where each page is array of text and annotations. The grid is a two-grid. All other operations are built around these basic ideas.

There were some difficulties writing the specification for how the content is “displayed” but a reasonable solution was met.

5.3 Pranav Pranav

<u>Start Day 7-12-16</u>

Group meeting 1: Used rough draft of specs to write first draft of pseudo code.
List down all the classes and its corresponding field.
Draw the first version of the class level diagram.
Draw work from diagram.
Split the task. Formal writing remaining.

<u>Start Day 7-13-16</u>

Group meeting 2: Made formal workflow diagram.
Review document.