Neio: TeX for the 21st century

Titouan Vervack

Promotor: Professor Marko van Dooren

What is it?

- Markup language
- Inspired by TeX and markdown

What’s wrong with Markdown

- Not extensible
- Too simple (no customisation)
- No programming model

What’s wrong with TeX

- Steep initial learning curve
- Allows redefenition of commands
- Overly complex
- Not statically typed

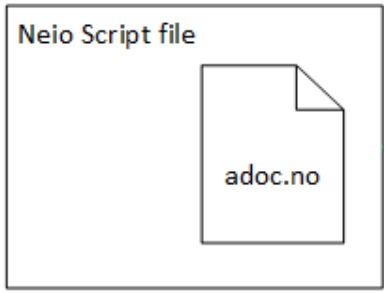
What’s wrong with Word

- Corruptable fileformat
- Not platform independent
- Overkill for very simple documents
- Not well suited for very complex documents

Goals

- Easy to get started with
- As powerful as TeX
- As simple as markdown
- Static typesystem

Compile flow



Neio class files

- Javalike syntax
  - Adoptability
  - Easy translation

```
namespace neio.stdlib;

import neio.lang.Content;

class Chapter extends Content;

String title;
Integer level;

Chapter(String title, Integer level) {
    this.title = title;
    this.level = level;
}

Paragraph newline(String parText) {
    Paragraph p = new Paragraph(parText);
    addContent(p);

    return p;
}

Image image(String caption, String imageName) {
    Image neioImage = new Image(caption, imageName);
    addContent(neioImage);

    return neioImage;
}

nested Chapter #(String title, Integer level) {
    if (level <= this.level) {
        return nearestAncestor(Chapter.class).#(title, level);
    }
    Chapter chapter = new Chapter(title, level);
    addContent(chapter);

    return chapter;
}
```

Neio script files

- Markdownlike syntax
  - Adoptability
  - Easy to read/write
  - Non-corruptable

```
[Document]

# Chapter 1
This is the first paragraph of our simple document.

## Chapter 1.1
This is a chapter of the next indentation level.

# Chapter 2
And here we have our last chapter.
```

Why output in java?

- Chameleon can output Java
- Java is statically typed
- Java is platform independent
- Java semantic ≈ Neio semantic

Building a document

- Start out with documentclass
- Everything is a methodcall

```
[Document]

# Chapter 1
This is the first paragraph of our simple document.

## Chapter 1.1
This is a chapter of the next indentation level.

# Chapter 2
And here we have our last chapter.
```

```
new Document()
        .#("Chapter 1")
            .newline("This is the first paragraph of our simple document.")
            .##("Chapter 1.1")
                .newline("This is a chapter of the next indentation level.")
        .#("Chapter 2")
            .newline("And here we have our last chapter.");
```

Building a document

```
new Document()
        .#("Chapter 1")
            .newline("This is the first paragraph of our simple document.")
            .##("Chapter 1.1")
                .newline("This is a chapter of the next indentation level.")
        .#("Chapter 2")
            .newline("And here we have our last chapter.");
```

```
public class simpleDocument {
    public static void main(String[] args) {
        neio.stdlib.Document $var0 = new Document();

        neio.stdlib.Chapter $var1 = $var0.hash("Chapter 1");
        neio.stdlib.Paragraph $var2 = $var1.newline("This is the first paragraph of our simple document.");

        neio.stdlib.Chapter $var3 = $var1.hash("Chapter 1.1", 2);
        neio.stdlib.Paragraph $var4 = $var3.newline("This is a chapter of the next indentation level.");

        neio.stdlib.Chapter $var5 = $var0.hash("Chapter 2");
        neio.stdlib.Paragraph $var6 = $var5.newline("And here we have our last chapter.");

        java.lang.String $var7 = new TexFileWriter($var0).write("simpleDocument");
        new TexToPDFBuilder().build($var7);
    }
}
```

Building a document: result

Chapter 1

This is the first paragraph of our simple document.

Chapter 1.1

This is a chapter of the next indentation level.

Chapter 2

And here we have our last chapter.

Nested methods

- Allow for nesting of method
- nested type f(arg, Integer)

```
29  nested Chapter #(String title, Integer level) {
30      if (level <= this.level) {
31          return nearestAncestor(Chapter.class).#(title, level);
32      }
33      Chapter chapter = new Chapter(title, level);
34      addContent(chapter);
35
36      return chapter;
37  }
```

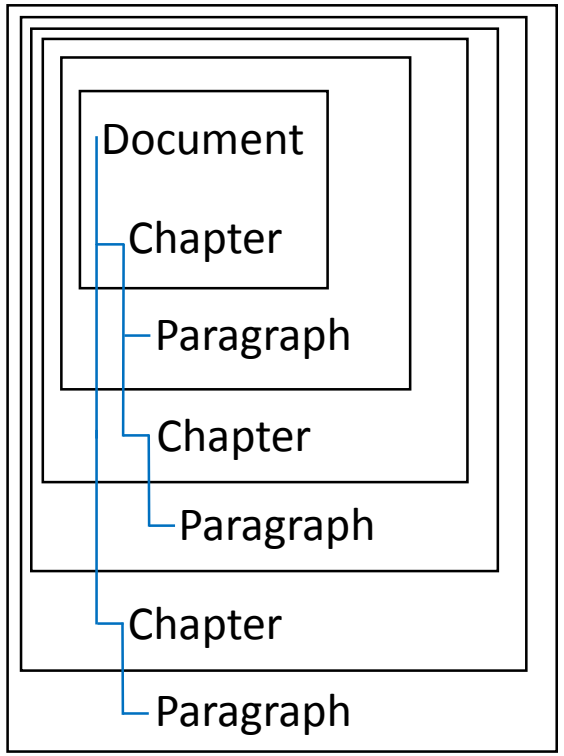
```
## Chapter 1.1
This is a chapter of the next indentation level.

# Chapter 2
And here we have our last chapter.
```

```
neio.stdlib.Chapter $var3 = $var1.hash("Chapter 1.1", 2);
neio.stdlib.Paragraph $var4 = $var3.newline("This is a chapter of the next indentation level.");

neio.stdlib.Chapter $var5 = $var0.hash("Chapter 2");
neio.stdlib.Paragraph $var6 = $var5.newline("And here we have our last chapter.");
```

Context types




```
[Document]

# Chapter 1
This is the first paragraph of our simple document.

## Chapter 1.1
This is a chapter of the next indentation level.

# Chapter 2
And here we have our last chapter.
```

```
new Document()
        .#("Chapter 1")
            .newline("This is the first paragraph of our simple document.")
            .##("Chapter 1.1")
                .newline("This is a chapter of the next indentation level.")
        .#("Chapter 2")
            .newline("And here we have our last chapter.");
            neio.stdlib.Paragraph $var6 = $var5.newline("And here we have our last chapter.");

            java.lang.String $var7 = new TexFileWriter($var0).write("simpleDocument");
            new TexToPDFBuilder().build($var7);
        }
    }
```

Customisability: inheritance

```
1  namespace neio.mypackage;
2
3  import neio.stdlib.Document;
4
5  class MyDocument extends Document;
6
7  String header() {
8      String s = super.header();
9      return s + "This is my document!";
10 }
```

```
1  [MyDocument]
2
3  # Chapter 1
4  This is the first paragraph of our simple document.
5
6  ## Chapter 1.1
7  This is a chapter of the next indentation level.
8
9  # Chapter 2
10 And here we have our last chapter.
```

This is my document!

Chapter 1

This is the first paragraph of our simple document.

Chapter 1.1

This is a chapter of the next indentation level.

Chapter 2

And here we have our last chapter.

Customisability: executing code

```
[Document]

# Chapter 1
This is the first paragraph of our simple document.

`#("Chapter 1.1", 2)`

```

```
neio.stdlib.Document $var0 = new Document();
neio.stdlib.Chapter $var1 = $var0.hash("Chapter 1");
neio.stdlib.Paragraph $var2 = $var1.newline("This is the first paragraph of our simple document.");

{
    $var1.hash("Chapter 1.1", 2);
}
```

Customisability: executing code

```
 8 `
 9 Chapter chap = new Chapter("Chapter 2", 1);
10 chap.newline("This is a coded paragraph");
11 addChapter(chap);
12 `
```

```
8 ``
9 Chapter chap = new Chapter("Chapter 2", 1);
10 chap.newline("This is a coded paragraph");
11 addChapter(chap);
12 ``
```

```
{
    Chapter chap = new Chapter("Chapter 2", 1);
    chap.newline("This is a coded paragraph");
    $var0.addChapter(chap);
}
```

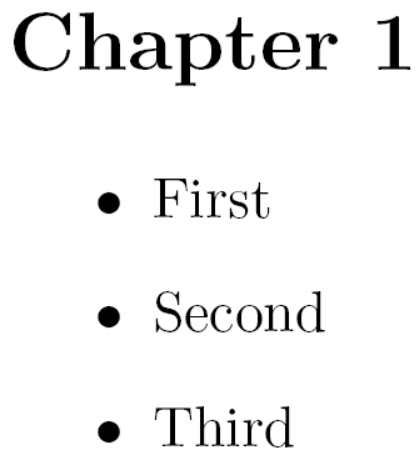
```
Chapter chap = new Chapter("Chapter 2", 1);
chap.newline("This is a coded paragraph");
$var0.addChapter(chap);
```

Add content through code

```
[Document]

# Chapter 1

``return new Itemize()``
* First
* Second
* Third
```




```
neio.stdlib.Document $var0 = new Document();
neio.stdlib.Chapter $var1 = $var0.hash("Chapter 1");

neio.stdlib.Itemize $var2 = $var1.addContent(new Itemize());
neio.stdlib.ItemizeItem $var3 = $var2.star("First");
neio.stdlib.ItemizeItem $var4 = $var2.star("Second");
neio.stdlib.ItemizeItem $var5 = $var2.star("Third");
```

Customisability: custom commands

```
namespace neio.mypackage;

import neio.stdlib.Document;

class MyDocument extends Document;

private Integer defaultFontSize = 12;

void increaseFontSize() {
    defaultFontSize = defaultFontSize + 2;
}

void setFontSize(Integer defaultFontSize) {
    this.defaultFontSize = defaultFontSize;
}

String header() {
    String s = super.header();
    return s + "This is my document!";
}
```

```
[MyDocument]

\increaseFontSize
# Chapter 1
This is the first paragraph of our simple Document.

\setFontSize(12)
## Chapter 1.1
This is a chapter of the next indentation level.
```

Future work before hand in

- Slideshows
- Surround methodes *bold text*  **bold text**
- Further controles on static typing
- Create more documentclassses (article, letter, book,…)

Future work

- Compiler optimizations
- Aliases for methods (e.g. alias star for *)
- Automatic double latex compile detection
- Native output to languages other than latex
- Further implementation of commonly used latex packages