

10.3-old A simple text formatter system

Here we present another example of using nested classes. The domain is text formatting as known from e.g. Microsoft Word, and LaTeX. The complexity of most of these systems is huge. Here we will thus stick to a very simple text formatting system.

The phenomena of our domain are documents that consists of a number of paragraphs. The paragraphs of a document are formatted according to a given template, which defines a number of styles for forming the paragraphs in the document. A style may e.g. define the font being used, the size of the font, the form of the font, etc. A paragraph may be left-aligned, centred, right-aligned, og with flushed margins.

In addition to styles, there are also properties associated with a document like page orientation (portrait or landscape), paper size (like A4, A3, and US letter), and number of columns.

To avoid ending up with a large example, we will keep the possible template and style properties to a minimum. The text formatter we define, can handle documents with two types of paragraphs: headings and body text. Headings are automatically numbered sequentially and body texts are indented by three spaces. A document has only one property which is the maximum length of a line in a paragraph.

We start by defining class Document:

```
class Document(temp: ref Template, lineMax: var integer):  
    class Paragraph(theContent: var String, theStyle: ref Style):  
        format -> theStyledPar: var String:  
            theStyledPar :=  
                theStyle.format(theContent.asString, lineMax)  
    content: obj OrderedList(Paragraph)  
    addPar(P: ref Paragraph):  
        content.insert(P)  
    format -> formattedContent: var String:  
        theStyledPar := ""  
        content.scan  
            formattedContent := formattedContent + current.format  
    print:  
        print(format)
```

- Class Document has two parameters temp, and lineMax where temp is the Template defining the format of the Document, and lineMax which defines the maximum length of a line in a paragraph. Temp and lineMax are used when adding paragraphs to a given document.
- It has a local class Paragraph, which has two parameters: theContent which is a String, and theStyle which is a reference to a Style object defining the style of the paragraph.
- A Paragraph has a format method that invokes the format method of the Style with the String of the Paragraph and the lineMax as an argument. Class Paragraph is an example of a nested class.
- The data-item content holds the paragraphs of the Document. The object content is of type OrderedList(Paragraph). OrderedList is similar to class Set as described in Section 4. The difference is that there is no ordering of the elements in a Set object whereas the

elements of an `OrderedList` are ordered in the order they are inserted into the list. For details about `OrderedList`, [see here](#).

- The method `addPar` adds a `Paragraph` to the `Document` by inserting it into `OrderedList`.
- The `print` method of `Document` invokes the `format` method and prints the `String` returned by `format`.

Next, we define class `Template`:

```
class Template:
```

```
  styles: obj OrderedList(Style)
  heading: ref Style
  bodyText: ref Style
  inner(Template)
```

- The `Styles` defined for a given `Template` are held in a `styles` object of type `OrderedList`.
- A `Template` object has two styles `heading` and `bodyText` which must be defined in subclasses of `Template`

Next we define class `Style`:

```
class Style -> S: ref Style:
```

```
  format(content: var String, lineMax: var integer)
    -> formattedContent: var String:<
    inner(format)
  inner(Style)
  S := this(Style)
```

In this simple example, a `Style`-object only has a virtual `format` method. Note that a generation of a `Style` object returns a reference to the generated object as expressed by assigning `this(Style)` to the return variable `S`.

We may now define `DefaultHeading` and `DefaultBodyText` as subclasses of `Style`:

```
class DefaultHeading: Style
```

```
  secNo: var integer
  format:<<
    secNo := secNo + 1
    formattedContent := "\n" + secNo + ". " + content
```

```
class DefaultBodyText: Style
```

```
  format:<<
    pos: var integer
    pos := 3
    formattedContent := "\n  "
    for (1):to(content.length):repeat
      pos := pos + 1
      if (pos > lineMax):then
        formattedContent := formattedContent + "\n  "
        pos := 3
```

```
formattedContent := formattedContent + content[inx]
```

A `DefaultHeading` has a local variable `secNo` keeping track of the heading numbers. The `format` method increments `secNo` by one and returns a `String` consisting of a newline ("`\n`"), followed by `secNo` followed by a dot and a blank ("`.` ") and the string hold in the `content` parameter of `format`.

The `format` method of class `DefaultBodyText` appends the characters in `content` to `formattedContent`. A newline and three blanks ("`\n` ") are inserted so each line has a maximum of `lineMax` characters.

We may define a standard template as a subclass of `Template`:

```
class StandardTemplate: Template
    heading := DefaultHeading
    bodyText:= DefaultBodyText
```

A `StandardTemplate` object initializes the predefined styles `heading` and `bodyText` to `defaultHeading` and `defaultBodyText` respectively.

We may enclose the above classes in a class `SimpleTextFormatter`:

```
class SimpleTextFormatter:
    class Document:
        ...
    class Template:
        ...
    class Style:
        ...
    class StandardTemplate: Template
        ...
```

We may now show a simple example of using the text formatter:

```
myFormatter: obj SimpleTextFormatter
doc: obj Document(StandardTemplate,10)
    addCont(T: ref Text, S: ref Style):
        addPar(Paragraph(T,S))
doc.addCont("Introduction",temp.heading)
doc.addCont("Once upon a time, ...",temp.bodyText)
doc.addCont("The problem",temp.heading)
doc.addCont("There was a programmer, ...",temp.bodyText)
doc.print
```

- The singular object `myFormatter` is subclassed from `SimpleTextFormatter`.
- It defines a singular object `doc` subclassed from `Document` with arguments `StandardTemplate` and `10` - the latter setting `lineMax` to 10 characters. Note that `StandardTemplate` generates an object and a reference to this object is passed as the first argument to `Document`.
- The `doc`-object has a method `addCont`, which inserts a `Paragraph` of a given `Style` in the `Document`.
- Then `doc.addCont` is invoked four times putting two headings and two body texts into the

document. For the headings, the `theStyle.bodyText` is used and for the body texts `theStyle.bodyText` is used.

- Finally it prints the document which gives the following output:

1. Introduction

Once up
on a tim
e, ...

2. The problem

There w
as a pro
grammer,
...

As can be seen, the `format` method may insert newlines inside a word – a real text formatting system should of course not do that.

Discussion

In the example above, we have only defined a single subclass, `Document` of `Document` and a single `StandardTemplate`. We may of course defined several subclasses of `Document` and `Template`. Most publishers, professional societies, etc., have their own document and template formats. For MS Word and LaTeX there are numerous possibilities depending on the use of the document.

The above text formatter is very simple. If you consider at systems like MS Word, it contains a large amount of options for forming a given document. This includes fonts and font families – class `Style` may be extended to include a font, a font size, etc. A `Paragraph` may include how it is to be adjusted. A `Document` may have a size (A4, US Letter, ...) and it may have a page orientation (landscape or portrait).

In MS Word its is possible to change the template of a given document. This implies that there must be a procedure/algorithm for converting the style of each paragraph to a similar style in the new tempalte. In MS Word this is mainly done by selecting a style of the same name.

It is a major task to develop a text system that resembles MS Word. The above classes may be extended to some extent, but it is unlikely that the structure of these classes may expand into something like MS Word. We leave it is a challenge!

Exercise

Add methods that generates a HTML-document from the content of a `Document` object.