Übersicht

Die Package-Sammlung besteht aus den folgenden Packages. Eine kurze Beschreibung folgt danach.

0.1 Beschreibung

Hier ist nur eine Test-Beschreibung, um zu sehen, wie das aussieht, und was genau in der `overview.html` drinstehen soll.

@version Id

@author Paul Ebermann

Siehe auch `de.dclj.paul.ltxdoclet`

1 Package de.dclj.paul.ltxdoclet

Pauls LaTeX-Doclet ist ein Doclet, welches als Ausgabe LaTeX-Quelltext produziert, um eine schön formatierte Ausgabe (etwa als PDF oder gedruckt) zu erhalten.

1.1 Klassen-Liste

1.2 Package-Beschreibung

Pauls LaTeX-Doclet ist ein Doclet, welches als Ausgabe LaTeX-Quelltext produziert, um eine schön formatierte Ausgabe (etwa als PDF oder gedruckt) zu erhalten.

- Die Haupt-Klasse (deren statische Methoden von javadoc aufgerufen werden) ist `DocletStart`.
- Ein Quelltextformatierer ist in `PrettyPrinter` gegeben.
- Umwandlung von HTML in LaTeX ist in der Klasse «Link:@link:de.dclj.paul.ltxdoc.HtmlKonverter|de
  welche eine abgespeckte Version des **TexDoclet** (at http://texdoclet.dev.java.net/)s
  von Gregg Wonderly ist.

@author Paul Ebermann

@version Id

1.3 Klasse de.dclj.paul.ltxdoclet.UniversalLinkCreator

1.3.1 Übersicht

Universeller Link-Creator. Er delegiert je nach Package an einen spezielleren Link-Creator.

@author **Paul Ebermann** (at mailto:paulo@heribert.local)

@version Id

1.3.2 Inhaltsverzeichnis

1.3.3 Variablen

intern Link-Creator für interne Links.

```
private LinkCreator intern;
```

extern Map von Package-Namen zu den entsprechenden LinkCreators für externe Links.

```
private Map⟨String, LinkCreator⟩ extern;
```

noLink Fallback-Linkcreator: wenn in extern keiner für das Package passt, wird hier einer genommen.

```
private LinkCreator noLink;
```

1.3.4 Konstruktoren

UniversalLinkCreator Konstruktor.

```
UniversalLinkCreator()
{
    super();
    this.intern = new InternLinkCreator();
    this.noLink = new NoLinkCreator();
    this.extern = new HashMap⟨String, LinkCreator⟩();
}
```

1.3.5 Methoden

createLink Describe `createLink` method here.

Parameter

string a `String` value

doc a `Doc` value

Rückgabewert a `String` value

```
public final String createLink(final String label,
                               final Doc doc)
{
    if (doc.isIncluded()) {
        return intern.createLink(label, doc);
    }
    LinkCreator lc = extern.get(packageName(doc));
    if (lc ≠ null) {
        return lc.createLink(label, doc);
    }
    return noLink.createLink(label, doc);
}
```

addOption Fügt eine Kommandozeilenoption hinzu.

Parameter

args die Option mitsamt ihren Argumenten in jeweils einem String.

```
public void addOption(String[] args)
{
    LinkCreator lc;
    String packageURL;
    if (args[0].equals("-link") || args[0].equals("-linkhtml")) {
        packageURL = args[1];
        lc = new HTMLInlineLinkCreator(args[1]);
```

```
        }
        else if (args[0].equals("-linkoffline") ||
    args[0].equals("-linkofflinehtml")) {
            lc = new HTMLInlineLinkCreator(args[1]);
            packageURL = args[2];
        }
        else if (args[0].equals("-linkpdf")) {
            packageURL = args[1].substring(0,
    args[1].lastIndexOf("/") + 1);
            lc = new PDFInlineLinkCreator(args[1]);
        }
        else if (args[0].equals("-linkofflinepdf")) {
            lc = new PDFInlineLinkCreator(args[1]);
            packageURL = args[2];
        }
        else {
            return;
        }
        parsePackageList(packageURL, lc);
    }
```

parsePackageList

```
    private void parsePackageList(String packageURL,
                                  LinkCreator lc)
    {
        try {
            URL url = new URL(packageURL + "package-list");
            InputStream in = url.openStream();
            BufferedReader reader = new BufferedReader(new
    InputStreamReader(in));
            String line;
            while((line = reader.readLine()) ≠ null){
                extern.put(line, lc);
            }
            reader.close();
            in.close();
        }
        catch(IOException io){
            io.printStackTrace();
        }
    }
```

packageName

```
    private String packageName(Doc d)
    {
        if (d instanceof PackageDoc) {
            return d.name();
        }
        if (d instanceof ProgramElementDoc) {
            return ((ProgramElementDoc)d).containingPackage().name();
        }
        return "-";
    }
```

packageHtmlAdress

```
    private String packageHtmlAdress(String baseURL,
                                     Doc target)
    {
        return baseURL + "?" + packageName(target).replace(".", "/");
    }
```

htmlAdress

```
    private String htmlAdress(String baseURL,
                              Doc target)
    {
        if (target instanceof PackageDoc) {
            return packageHtmlAdress(baseURL, target) +
    "/package-summary.html";
        }
        else if (target instanceof ClassDoc) {
            return packageHtmlAdress(baseURL, target) + "/" +
    target.name() + ".html";
        }
        else {
            MemberDoc md = (MemberDoc)target;
            return packageHtmlAdress(baseURL, md) + "/" +
    md.containingClass().name() + ".html#" + md.name();
        }
    }
```

1.4 Klasse de.dclj.paul.ltxdoclet.UniversalLinkCreator.HTMLInlineLinkCreator

1.4.1 Übersicht

1.4.2 Inhaltsverzeichnis

1.4.3 Variablen

url

```
private String url;
```

1.4.4 Konstruktoren

UniversalLinkCreator.HTMLInlineLinkCreator

```
UniversalLinkCreator.HTMLInlineLinkCreator(String adresse)
{
    super();
    this.url = adresse;
}
```

1.4.5 Methoden

createLink

```
public String createLink(String label,
                         Doc target)
{
    String linkTarget = htmlAdress(this.url, target);
    return "\\href{" + linkTarget + "}{" + label + "}";
}
```

1.5 Klasse de.dclj.paul.ltxdoclet.UniversalLinkCreator.PDFInlineLinkCreator

1.5.1 Übersicht

Link-Creator für Inline-Links zu anderen PDF-Dateien, die mit diesem Doclet erstellt wurden (oder zumindest entsprechende interne Link-Anker haben).

1.5.2 Inhaltsverzeichnis

1.5.3 Variablen

pdfURL

```
private String pdfURL;
```

1.5.4 Konstruktoren

UniversalLinkCreator.PDFInlineLinkCreator

```
UniversalLinkCreator.PDFInlineLinkCreator(String pdf)
{
    super();
    this.pdfURL = pdf;
}
```

1.5.5 Methoden

createLink

```
public String createLink(String label,
                         Doc target)
{
    String targetName =
LaTeXWriter.configuration.toRefLabel(target);
    String url = pdfURL + "#" + targetName;
    return "\\href{" + url + "}{" + label + "}";
}
```

1.6 Klasse de.dclj.paul.ltxdoclet.UniversalLinkCreator.InternLinkCreator

1.6.1 Übersicht

Ein Link-Creator für interne Links.

1.6.2 Inhaltsverzeichnis

1.6.3 Konstruktoren

UniversalLinkCreator.InternLinkCreator

```
private UniversalLinkCreator.InternLinkCreator()
{
    super();
}
```

1.6.4 Methoden

createLink «@inheritDoc:»

```
public String createLink(String label,
                         Doc target)
{
```

```
        String targetName =
    LaTeXWriter.configuration.toRefLabel(target);
        return "\\hyperlink{" + targetName + "}{" + label + "}";
    }
```

1.7 Klasse de.dclj.paul.ltxdoclet.UniversalLinkCreator.NoLinkCreator

1.7.1 Übersicht

Ein Link-Creator, der keine Links erstellt.

1.7.2 Inhaltsverzeichnis

1.7.3 Konstruktoren

UniversalLinkCreator.NoLinkCreator

```
    private UniversalLinkCreator.NoLinkCreator()
    {
        super();
    }
```

1.7.4 Methoden

createLink

```
    public String createLink(String label,
                             Doc d)
    {
        return label;
    }
```

1.8 Enum de.dclj.paul.ltxdoclet.SpecialToken

1.8.1 Übersicht

Java-Token, die eine spezielle Behandlung benötigen.

1.8.2 Inhaltsverzeichnis

1.8.3 Enum-Konstanten

DUMMY

```
    public static final SpecialToken DUMMY = new
    SpecialToken("üäöü");
```

POSTFIX_INCREMENT

```
    public static final SpecialToken POSTFIX_INCREMENT = new
    SpecialToken("++", Kind.POSTFIX_INCREMENT);
```

PREFIX_INCREMENT

```
    public static final SpecialToken PREFIX_INCREMENT = new
    SpecialToken("++", Kind.PREFIX_INCREMENT);
```

POSTFIX_DECREMENT

```
    public static final SpecialToken POSTFIX_DECREMENT = new
    SpecialToken("--", Kind.POSTFIX_DECREMENT);
```

PREFIX_DECREMENT

```
    public static final SpecialToken PREFIX_DECREMENT = new
    SpecialToken("--", Kind.PREFIX_DECREMENT);
```

UNARY_PLUS

```
    public static final SpecialToken UNARY_PLUS = new
    SpecialToken("+", Kind.UNARY_PLUS);
```

UNARY_MINUS

```
    public static final SpecialToken UNARY_MINUS = new
    SpecialToken("-", Kind.UNARY_MINUS);
```

BITWISE_COMPLEMENT

```
    public static final SpecialToken BITWISE_COMPLEMENT = new
    SpecialToken("~", Kind.BITWISE_COMPLEMENT, 2,
    " \\clap{$\\sim$} ");
```

LOGICAL_COMPLEMENT

```
    public static final SpecialToken LOGICAL_COMPLEMENT = new
    SpecialToken("!", Kind.LOGICAL_COMPLEMENT);
```

MULTIPLY

```
    public static final SpecialToken MULTIPLY = new SpecialToken("*",
    Kind.MULTIPLY, 2, " \\clap{$\\cdot$} ");
```

DIVIDE

```
    public static final SpecialToken DIVIDE = new SpecialToken("/",
    Kind.DIVIDE, 2, " \\clap{$/$} ");
```

REMAINDER

```
    public static final SpecialToken REMAINDER = new
    SpecialToken("%", Kind.REMAINDER, 3, " \\% ");
```

PLUS

```
    public static final SpecialToken PLUS = new SpecialToken("+",
    Kind.PLUS, 2, " \\clap{$+$} ");
```

MINUS

```
    public static final SpecialToken MINUS = new SpecialToken("-",
    Kind.MINUS, 2, " \\clap{$-$} ");
```

LEFT_SHIFT

```
public static final SpecialToken LEFT_SHIFT = new
SpecialToken("<<", Kind.LEFT_SHIFT, 4, " << ");
```

RIGHT_SHIFT

```
public static final SpecialToken RIGHT_SHIFT = new
SpecialToken(">>", Kind.RIGHT_SHIFT, 4, " >> ");
```

UNSIGNED_RIGHT_SHIFT

```
public static final SpecialToken UNSIGNED_RIGHT_SHIFT = new
SpecialToken(">>>", Kind.UNSIGNED_RIGHT_SHIFT, 5, " >>> ");
```

LESS_THAN

```
public static final SpecialToken LESS_THAN = new
SpecialToken("<", Kind.LESS_THAN, 2, " \\clap{$<$} ");
```

GREATER_THAN

```
public static final SpecialToken GREATER_THAN = new
SpecialToken(">", Kind.GREATER_THAN, 2, " \\clap{$>$} ");
```

LESS_THAN_EQUAL

```
public static final SpecialToken LESS_THAN_EQUAL = new
SpecialToken("<=", Kind.LESS_THAN_EQUAL, 2, " \\clap{$\\leq$} ");
```

GREATER_THAN_EQUAL

```
public static final SpecialToken GREATER_THAN_EQUAL = new
SpecialToken(">=", Kind.GREATER_THAN_EQUAL, 2,
" \\clap{$\\geq$} ");
```

EQUAL_TO

```
public static final SpecialToken EQUAL_TO = new
SpecialToken("==", Kind.EQUAL_TO, 3, "\\clapon{$\\equiv$}{   }");
```

NOT_EQUAL_TO

```
public static final SpecialToken NOT_EQUAL_TO = new
SpecialToken("!=", Kind.NOT_EQUAL_TO, 3,
"\\clapon{$\\neq$}{   }");
```

AND

```
public static final SpecialToken AND = new SpecialToken("&",
Kind.AND, 2, " \\clap{\\&} ");
```

XOR

```
public static final SpecialToken XOR = new SpecialToken("^",
Kind.XOR, 2, " \\clap{\\barwedge} ");
```

OR

```
public static final SpecialToken OR = new SpecialToken("|",
Kind.OR, 2, " \\clap{|} ");
```

CONDITIONAL_AND

```
    public static final SpecialToken CONDITIONAL_AND = new
    SpecialToken("&&", Kind.CONDITIONAL_AND, 4, " \\&\\& ");
```

CONDITIONAL_OR

```
    public static final SpecialToken CONDITIONAL_OR = new
    SpecialToken("||", Kind.CONDITIONAL_OR, 4, " || ");
```

ASSIGNMENT

```
    public static final SpecialToken ASSIGNMENT = new
    SpecialToken("=", Kind.ASSIGNMENT, 4,
    "  \\clap{$\\leftarrow$}  ");
```

MULTIPLY_ASSIGNMENT

```
    public static final SpecialToken MULTIPLY_ASSIGNMENT = new
    SpecialToken("*=", Kind.MULTIPLY_ASSIGNMENT);
```

DIVIDE_ASSIGNMENT

```
    public static final SpecialToken DIVIDE_ASSIGNMENT = new
    SpecialToken("/=", Kind.DIVIDE_ASSIGNMENT);
```

REMAINDER_ASSIGNMENT

```
    public static final SpecialToken REMAINDER_ASSIGNMENT = new
    SpecialToken("%=", Kind.REMAINDER_ASSIGNMENT);
```

PLUS_ASSIGNMENT

```
    public static final SpecialToken PLUS_ASSIGNMENT = new
    SpecialToken("+=", Kind.PLUS_ASSIGNMENT);
```

MINUS_ASSIGMENT

```
    public static final SpecialToken MINUS_ASSIGMENT = new
    SpecialToken("-=", Kind.MINUS_ASSIGNMENT);
```

LEFT_SHIFT_ASSIGNMENT

```
    public static final SpecialToken LEFT_SHIFT_ASSIGNMENT = new
    SpecialToken("<<=", Kind.LEFT_SHIFT_ASSIGNMENT);
```

RIGHT_SHIFT_ASSIGNMENT

```
    public static final SpecialToken RIGHT_SHIFT_ASSIGNMENT = new
    SpecialToken(">>=", Kind.RIGHT_SHIFT_ASSIGNMENT);
```

UNSIGNED_RIGHT_SHIFT_ASSIGNMENT

```
    public static final SpecialToken UNSIGNED_RIGHT_SHIFT_ASSIGNMENT
    = new SpecialToken(">>>=", Kind.UNSIGNED_RIGHT_SHIFT_ASSIGNMENT);
```

AND_ASSIGNMENT

```
    public static final SpecialToken AND_ASSIGNMENT = new
    SpecialToken("&=", Kind.AND_ASSIGNMENT);
```

XOR_ASSIGNMENT

```
    public static final SpecialToken XOR_ASSIGNMENT = new
    SpecialToken("^=", Kind.XOR_ASSIGNMENT);
```

OR_ASSIGNMENT

```
public static final SpecialToken OR_ASSIGNMENT = new
SpecialToken("|=", Kind.OR_ASSIGNMENT);
```

LEFT_ANGLE

```
public static final SpecialToken LEFT_ANGLE = new
SpecialToken("<", " \\llap{$\\langle$}");
```

RIGHT_ANGLE

```
public static final SpecialToken RIGHT_ANGLE = new
SpecialToken(">", "\\rlap{$\\rangle$} ");
```

RIGHT_ANGLE_SPACE

```
public static final SpecialToken RIGHT_ANGLE_SPACE = new
SpecialToken("> ", "\\rlap{$\\rangle$}  ");
```

SEMIKOLON

```
public static final SpecialToken SEMIKOLON = new
SpecialToken(";", 2, "~\\clap{\\textbf{;}} ");
```

KOMMA

```
public static final SpecialToken KOMMA = new SpecialToken(",");
```

QUESTION_MARK

```
public static final SpecialToken QUESTION_MARK = new
SpecialToken("?", 3, " \\textbf{?} ");
```

COLON

```
public static final SpecialToken COLON = new SpecialToken(":", 3,
" \\textbf{:} ");
```

KOMMA_SPACE

```
public static final SpecialToken KOMMA_SPACE = new
SpecialToken(", ");
```

LEFT_BRACE Linke geschweifte Klammer.

```
public static final SpecialToken LEFT_BRACE = new
SpecialToken("{", "\\{");
```

LEFT_BRACE_SPACE

```
public static final SpecialToken LEFT_BRACE_SPACE = new
SpecialToken("{ ", "\\{ ");
```

SPACE_LEFT_BRACE

```
public static final SpecialToken SPACE_LEFT_BRACE = new
SpecialToken(" {", " \\{");
```

RIGHT_BRACE rechte geschweifte Klammer.

```
public static final SpecialToken RIGHT_BRACE = new
SpecialToken("}", "\\}");
```

SPACE_RIGHT_BRACE

```
public static final SpecialToken SPACE_RIGHT_BRACE = new
SpecialToken(" }", " \\}");
```

1.8.4 Variablen

len

```
private int len;
```

TeXtext

```
private String TeXtext;
```

javaText

```
private String javaText;
```

kind

```
private Kind kind;
```

javaMap Eine Abbildung von den Java-Token zu unseren Token-Objekten.

```
private static Map⟨String, SpecialToken⟩ javaMap = new HashMap
⟨String, SpecialToken⟩();
```

opMap

```
private static Map⟨Kind, SpecialToken⟩ opMap = new EnumMap⟨Kind,
SpecialToken⟩(Kind.class);
```

1.8.5 Konstruktoren

SpecialToken

```
SpecialToken(String javaText,
             Kind expressionKind,
             int len,
             String TeXtext)
{
    super();
    this.len = len;
    this.TeXtext = TeXtext;
    this.javaText = javaText;
    this.kind = expressionKind;
}
```

SpecialToken

```
SpecialToken(String javaText,
             Kind kind,
             String TeXtext)
{
    this(javaText, kind, javaText.length(), TeXtext);
}
```

SpecialToken

```
SpecialToken(String javaText,
             int len,
             String TeXtext)
{
    this(javaText, null, len, TeXtext);
}
```

SpecialToken

```
SpecialToken(String jText,
             String lText)
{
    this(jText, jText.length(), lText);
}
```

SpecialToken

```
SpecialToken(String text,
             Kind kind)
{
    this(text, kind, text.length(), text);
}
```

SpecialToken

```
SpecialToken(String text)
{
    this(text, text.length(), text);
}
```

1.8.6 Methoden

values

valueOf

getLength

```
public int getLength()
{
    return len;
}
```

getToken

```
public static SpecialToken getToken(Kind kind)
{
    return opMap.get(kind);
}
```

getToken

```
public static SpecialToken getToken(String jText)
{
    return javaMap.get(jText);
}
```

getReplacement

```
public String getReplacement()
{
    return TeXtext;
}
```

1.9 Klasse de.dclj.paul.ltxdoclet.PackageWriter

1.9.1 Übersicht

Schreibt die Dokumentation für ein Package.

1.9.2 Inhaltsverzeichnis

1.9.3 Variablen

doc

```
private PackageDoc doc;
```

1.9.4 Konstruktoren

PackageWriter Erstellt einen neuen PackageWriter.

Parameter

pd das zu dokumentierende Paket.

```
public PackageWriter(PackageDoc pd)
    throws IOException
{
    super(new File(configuration.toOutputFileName(pd),
"package-doc.tex"));
    this.doc = pd;
}
```

1.9.5 Methoden

writeDoc Erstellt die Doku für das Paket.

```
public void writeDoc()
{
```

```
        configuration.root.printNotice("ltxdoclet: package-
    doc.tex für \"" + doc +
    "\" wird erstellt ...");
        ClassDoc[] classes = doc.allClasses();
        writeClasses(classes);
        chapter("Package ", doc);
        println();
        writeInlineTags(doc.firstSentenceTags());
        println();
        section("Klassen-Liste");
        println("\\begin{description}");
        for(int i = 0; i < classes.length; i++){
            ClassDoc cd = classes[i];
            print("\\item[{");
            if (cd.isInterface()) {
                print("\\textit{" + createLink(cd) + "}");
            }
            else {
                print(createLink(cd));
            }
            println("}]");
            writeInlineTags(cd.firstSentenceTags());
            print("\\hfill");
            println(referenceTo(cd));
            println();
        }
        println("\\end{description}");
        section("Package-Beschreibung");
        writeDescription(doc);
        writeClassImports(classes);
        close();
    }
```

writeClasses Erstellt die Doku für die Klassen in einzelnen Threads.

Parameter

classes die zu dokumentierenden Klassen.

```
    public void writeClasses(ClassDoc[] classes)
    {
        for(int i = 0; i < classes.length; i++){
            final ClassDoc cd = classes[i];
            Thread tr = new Thread(cd + "-Writer")«Typdeklaration: »;
```

```
            tr.run();
        }
    }
```

writeClassImports Erstellt die Input-Anweisungen, um die parallel erstellten Klassendokus in die Package-Doku einzubinden.

```
public void writeClassImports(ClassDoc[] classes)
{
    String pkgDir = configuration.toInputFileName(doc) + "/";
    for(int i = 0; i < classes.length; i++){
        println("\\input{" + pkgDir + classes[i].name() + "}");
    }
}
```

1.10 Klasse de.dclj.paul.ltxdoclet.MainFileWriter

1.10.1 Übersicht

Ein Writer für die Haupt-Datei.

1.10.2 Inhaltsverzeichnis

1.10.3 Konstruktoren

MainFileWriter

```
public MainFileWriter()
    throws IOException
{
    super(new File(configuration.destdir, "doku-main.tex"));
}
```

1.10.4 Methoden

writeDoku Erstellt die komplette Doku für das Projekt.

Dazu werden in parallelen Threads `PackageWriter` für die einzelnen Packages aufgerufen und dann die Main-Datei erstellt.

```
public void writeDoku()
{
    writePackages();
    configuration.root.printNotice("ltxdoclet: doku-
main.tex wird erstellt ...");
    println("   % Damit beim Compilieren nicht bei jedem Feh-
ler angehalten wird");
    println("\\scrollmode");
```

```
        println();
        writePreamble();
        println("\\begin{document}");
        println();
        chapter("Übersicht", false);
        ltxwrite(configuration.doctitle+
    " besteht aus den folgenden Packages. Eine");
        ltxwrite(" kurze Beschreibung folgt danach.");
        writePackageList();
        section("Beschreibung");
        writeOverview();
        println("\\setcounter{chapter}{0}");
        writePackageImports();
        println("\\appendix");
        println("\\end{document}");
        close();
        configuration.root.printNotice("ltxdoclet: ... doku-
    main.tex fertig.");
        configuration.root.printNotice("ltxdoclet: warte auf Beendi-
    gung der anderen Dateien ...");
        waitForAllThreads();
        configuration.root.printNotice("ltxdoclet: Fertig!");
    }
```

waitForAllThreads Wartet, bis alle Threads in LaTeXWriter.configuration.threads beendet wurden und sich aus der Liste streichen.

```
public void waitForAllThreads()
{
    List threads = configuration.threads;
    while(true){
        Thread akt;
        synchronized((threads)) {
            if (!configuration.threads.isEmpty())
                akt = (Thread)threads.get(0);
            else
                break;
        }
        try {
            akt.join();
        }
        catch(InterruptedException ex){
            Thread.currentThread().interrupt();
```

```
                configuration.wasError = true;
                return;
            }
        }
    }
```

copyPackage Kopiert eine Datei (z.B. ein LaTeX-Paket) aus unseren Programm-Ressourcen in das Ausgabeverzeichnis.

Parameter

packageName der Name der Datei, z.B. `ltxdoclet.sty`.

```
    private void copyPackage(String packageName)
    {
        try {
            InputStream in = new BufferedIn-
putStream(MainFileWriter.class.getResourceAsStream(packageName));
            OutputStream out = new BufferedOutputStream(new
FileOutputStream(new File(configuration.destdir, packageName)));
            pipeInToOut(in, out);
            in.close();
            out.close();
        }
        catch(IOException io){
            throw new
RuntimeException("konnte das Package nicht kopieren: ", io);
        }
    }
```

pipeInToOut Leitet den kompletten Inhalt einen InputStreams in einen OutputStream um.

Diese Methode liest und schreibt jedes byte einzeln - man sollte also gepufferte Streams verwenden, damit es nicht zu langsam wird.

```
    private void pipeInToOut(InputStream in,
                             OutputStream out)
        throws IOException
    {
        int r;
        while(0≤ (r = in.read())){
            out.write(r);
        }
    }
```

writePreamble

```
private void writePreamble()
{
    println("   % Report scheint für eine API jedenfalls bes-
ser als Artikel");
    println("\\documentclass[final, 11pt, a4paper]{scrreprt}");
    println();
    println("\\usepackage{fontspec}");
    println("\\usepackage{xunicode}");
    println("\\setmonofont{Liberation Mono}");
    if (Locale.getDefault().getLanguage().equals("de")) {
        println("  % Neue deutsche Silbentrennung");
        println("\\usepackage[german]{polyglossia}");
        println();
    }
    println("\\usepackage[pdfborderstyle={/S/U/W 1}]{hyperref}");
    println("\\usepackage{enumerate}");
    println();
    println("\\usepackage[dvipsnames]{color}");
    println();
    println("\\usepackage{ltxdoclet}");
    println();
    ;
    copyPackage("ltxdoclet.sty");
}
```

translateEncoding Übersetzt einen Java-Encoding-Namen in den passenden Namen für das inputenc-Paket.

```
private String translateEncoding(Charset javaName)
{
    if (javaName.toString().equals("UTF-8")) {
        return "utf8";
    }
    return javaName.toString().toLowerCase();
}
```

writeOverview Erstellt den Überblick über die Package-Sammlung, mit Beschreibung etc.

```
private void writeOverview()
{
    writeDescription(configuration.root);
}
```

writePackageList Gibt eine Liste der Packages aus.

```
    private void writePackageList()
    {
        println("   % Liste der Packages:");
        println("\\begin{enumerate}[1.]");
        PackageDoc[] pkgs = configuration.packages;
        for(int i = 0; i< pkgs.length; i++){
            PackageDoc pd = pkgs[i];
            println("\\item "+ createLink(pd)+ "\\dotfill "+
referenceTo(pd));
            newLine();
            writeInlineTags(pd.firstSentenceTags());
        }
        println("\\end{enumerate}");
    }
```

writePackageImports

```
    private void writePackageImports()
    {
        PackageDoc[] pkgs = configuration.packages;
        for(int i = 0; i< pkgs.length; i++){
            String pkgName = configuration.toInputFileName(pkgs[i]);
            println("\\input{"+ pkgName+ "/package-doc.tex"+ "}");
        }
    }
```

writePackages

```
    private void writePackages()
    {
        configuration.root.printNotice("Package-
Dokus werden erstellt ...");
        PackageDoc[] pkgs = configuration.packages;
        for(int i = 0; i< pkgs.length; i++){
            final PackageDoc pd = pkgs[i];
            Thread thread = new Thread(pd+
"-Writer")«Typdeklaration: »;
            configuration.threads.add(thread);
            thread.start();
        }
    }
```

writeIndex

```
    private void writeIndex()
    {
    }
```

1.11 Klasse de.dclj.paul.ltxdoclet.HelpTextBundleControl

1.11.1 Übersicht

A resource bundle control implementation for loading a help text in a UTF-8 encoded ".txt"file as a single-key ResourceBundle. (The key is `"help"`).

1.11.2 Inhaltsverzeichnis

1.11.3 Konstruktoren

HelpTextBundleControl

```
HelpTextBundleControl()
{
    super();
}
```

1.11.4 Methoden

getFormats

```
public List⟨String⟩ getFormats(String basename)
{
    return Arrays.asList("helptext");
}
```

newBundle

```
public ResourceBundle newBundle(String baseName,
                                Locale loc,
                                String format,
                                ClassLoader loader,
                                boolean reload)
    throws IOException
{
    if (!format.equals("helptext")) {
        return null;
    }
    String fileName = toResourceName(toBundleName(baseName, loc),
"txt");
    InputStream stream = loader.getResourceAsStream(fileName);
    if (stream ≡ null) {
        return null;
    }
    StringBuilder b = new StringBuilder(stream.available());
```

```
    Reader r = new BufferedReader(new InputStreamReader(stream,
"UTF-8"));
    char[] c = new char[500];
    int l;
    while((l = r.read(c)) > 0){
        b.append(c, 0, l);
    }
    final String val = b.toString();
    return new ListResourceBundle()«Typdeklaration: »;
}
```

1.12 Klasse de.dclj.paul.ltxdoclet.DocletStart

1.12.1 Übersicht

Eintrittspunkt für das Doclet.

1.12.2 Inhaltsverzeichnis

1.12.3 Konstruktoren

DocletStart

```
private DocletStart()
{
    super();
}
```

1.12.4 Methoden

start Diese Methode wird von Javadoc aufgerufen, um unser Doclet arbeiten zu lassen.

Parameter

root alle Infos, die wir brauchen.

Rückgabewert `true` falls erfolgreich, `false` falls es einen Fehler gab.

```
public static boolean start(RootDoc root)
{
    root.printNotice("ltxdoclet, (c) Paul Ebermann 2001, 2010");
    try {
        configuration().setOptions(root);
        if (configuration().includeSource) {
            configuration().startCompiler();
        }
        new MainFileWriter().writeDoku();
```

```
            configuration().root.printNotice("Fertig.");
            configuration().root.printNotice("Fehler: " +
    configuration().wasError);
            return !configuration().wasError;
        }
        catch(Exception ex){
            ex.printStackTrace();
            return false;
        }
    }
```

optionLength

```
    public static int optionLength(String option)
    {
        return configuration().optionLength(option);
    }
```

validOptions

```
    public static boolean validOptions(String[][] ops,
                                       DocErrorReporter rep)
    {
        return configuration().validOptions(ops, rep);
    }
```

languageVersion

```
    public static LanguageVersion languageVersion()
    {
        return LanguageVersion.JAVA_1_5;
    }
```

configuration

```
    static LtxDocletConfiguration configuration()
    {
        if (LaTeXWriter.configuration ≡ null) {
            LaTeXWriter.configuration = new LtxDocletConfiguration();
        }
        return LaTeXWriter.configuration;
    }
```

1.13 Klasse de.dclj.paul.ltxdoclet.SourceFormatter

1.13.1 Übersicht

Ein Formatierer für Quelltext.

@author **Paul Ebermann** (at mailto:paulo@heribert.local)

@version Id

1.13.2 Inhaltsverzeichnis

1.13.3 Variablen

writer Hierhin geben wir die Daten aus.

```
private LaTeXWriter writer;
```

indentStack

```
private Deque⟨Integer⟩ indentStack;
```

currentLine

```
private StringBuilder currentLine;
```

hasIndent

```
private boolean hasIndent = false;
```

currentLineLength

```
private int currentLineLength;
```

verbDelim

```
private static final char[] verbDelim =
"#\'+*&-.,:;<>|!/()=[]1234567890".toCharArray();
```

1.13.4 Konstruktoren

SourceFormatter Creates a new `SourceFormatter` instance.

```
public SourceFormatter(LaTeXWriter wri)
{
    super();
    this.writer = wri;
    this.currentLine = new StringBuilder(80);
    this.indentStack = new ArrayDeque⟨Integer⟩();
    this.indentStack.push(0);
    this.currentLineLength = 0;
}
```

1.13.5 Methoden

flush Sorgt dafür, dass alles an den unterliegenden Stream ausgegeben wird.

```
public void flush()
{
    if (hasIndent) {
        writer.print(currentLine);
    }
}
```

appendInLine Hängt an die aktuelle Zeile einen Text an, dessen Drucklänge gleich seiner Länge ist.

```
private void appendInLine(String text)
{
    appendInLine(text, text.codePointCount(0, text.length()));
}
```

appendInLine Hängt an die aktuelle Zeile einen Text an und registriert dessen Länge.

Parameter

text der auszugebende Text.

len die Druck-Länge dieses Textes in Zeichen.

```
private void appendInLine(String text,
                          int len)
{
    indent();
    currentLine.append(text);
    currentLineLength+=len;
}
```

finishLine Beendet eine Zeile. Der aktuelle Inhalt des Zeilenpuffers wird ausgegeben und danach der Zeilenpuffer geleert.

```
private void finishLine()
{
    indent();
    writer.println(currentLine);
    currentLine.setLength(0);
    hasIndent = false;
}
```

pushIndent Beginnt eine neue Einrückungsebene, welche die aktuelle »Cursorposition« (d.h. die bisherige Länge aktuellen Zeile) als neue Einrücktiefe hat.

Alle bis zum nächsten `popIndent` begonnenen Zeilen erhalten eine Einrückung dieser Länge.

```
public void pushIndent()
{
    currentLine.append("\\ltdSetIndent{" + currentLineLength +
"}");
    this.indentStack.push(currentLineLength);
}
```

addIndent Beginnt eine neue Einrückungsebene.

Alle bis zum nächsten `popIndent` begonnenen Zeilen erhalten eine Einrückung, die 4 Zeichen mehr ist als die aktuelle Einrückung.

```
public void addIndent()
{
    int newIndent = this.indentStack.peek() + 4;
    currentLine.append("\\ltdSetIndent{" + newIndent + "}");
    this.indentStack.push(newIndent);
}
```

popIndent Beendet eine Einrückungsebene.

Alle in Zukunft begonnenen Zeilen haben wieder die Einrückung, die vor dem letzten `addIndent` oder `pushIndent` aktiv war.

```
public void popIndent()
{
    this.indentStack.pop();
    currentLine.append("\\ltdSetIndent{" +
this.indentStack.peek() + "}");
}
```

indent stellt sicher, dass die aktuelle Zeile eingerückt ist.

```
private void indent()
{
    if (hasIndent)
        return;
    currentLine.append("\\ltdIndent");
    int count = indentStack.peek();
    for(int i = 0; i < count; i++)
        currentLine.append(' ');
    currentLine.append(".");
    currentLineLength = count;
    hasIndent = true;
}
```

print gibt puren Text aus (d.h. Text, dessen Drucklänge gleich seiner Länge ist).

```
public void print(String text)
{
    if (text.length() == 0)
        return;
    int index = text.indexOf("\n");
    if (index < 0) {
        appendInLine(text);
        return;
    }
    appendInLine(text.substring(0, index));
    finishLine();
    print(text.substring(index + 1));
}
```

println gibt puren Text aus und beendet danach die Zeile.

```
public void println(String text)
{
    print(text);
    finishLine();
}
```

print gibt die String-Darstellung des Objektes als puren Text aus.

```
public void print(Object o)
{
    print(o.toString());
}
```

println gibt die String-Darstellung des Objektes als puren Text aus und beendet danach die Zeile.

```
public void println(Object o)
{
    println(o.toString());
}
```

println beendet die aktuelle Zeile.

```
public void println()
{
    finishLine();
}
```

printKeyword Druckt ein Schlüsselwort.

```
public void printKeyword(String keyword)
{
    appendInLine("\\markKeyword{" + keyword + "}",
keyword.length());
}
```

printId druckt einen Identifier (dabei werden _ escapet.)

```
public void printId(CharSequence name)
{
    indent();
    int len = name.length();
    for(int i = 0; i < len; i++){
        char c = name.charAt(i);
        if (c == '_') {
            this.currentLine.append("\\_");
        }
        else {
            this.currentLine.append(c);
        }
    }
    this.currentLineLength+=len;
}
```

printSpecial Druckt ein spezielles Token.

```
public void printSpecial(String token)
{
    if (token.equals(""))
        return;
    if (token.charAt(0) == '\n') {
        println();
        token = token.substring(1);
        if (token.equals(""))
            return;
    }
    SpecialToken tok = SpecialToken.getToken(token);
    if (tok == null) {
        appendInLine("\\textbf{" + token + "}", token.length());
    }
    else {
        appendInLine(tok.getReplacement(), tok.getLength());
    }
}
```

printSpecial

```
    public void printSpecial(Kind token)
    {
        SpecialToken tok = SpecialToken.getToken(token);
        if (tok == null) {
            print(token);
        }
        else {
            appendInLine(tok.getReplacement(), tok.getLength());
        }
    }
```

printAsNumber

```
    private void printAsNumber(String text)
    {
        appendInLine("{\\markNumber ", 0);
        appendInLine(text);
        appendInLine("}", 0);
    }
```

printLiteral Druckt ein Literal für einen Wert.

```
    public void printLiteral(Object value,
                             Kind type)
    {
        switch(type) {
            case BOOLEAN_LITERAL:
                String val = value.toString();
                appendInLine("{\\markLiteralKeyword " + val + "}",
    val.length());
                return;
            case INT_LITERAL:
            case DOUBLE_LITERAL:
                printAsNumber(value.toString());
                return;
            case LONG_LITERAL:
                printAsNumber(value.toString() + "L");
                return;
            case FLOAT_LITERAL:
                printAsNumber(value.toString() + "f");
                return;
            case CHAR_LITERAL: {
                String javaString =
    escapeJavaString(value.toString());
```

```
                int len = javaString.length() + 2;
                String lString = escapeLaTeXString("\'" + javaString +
    "\'");
                appendInLine("{\\markNumber " + lString + "}", len);
                return;
            }
            case STRING_LITERAL: {
                String javaString =
    escapeJavaString(value.toString());
                int len = javaString.length() + 2;
                String lString = escapeLaTeXString('\"' + javaString +
    '\"');
                appendInLine("{\\markString " + lString + "}", len);
                return;
            }
            case NULL_LITERAL:
                appendInLine("{\\markLiteralKeyword null}", 4);
                return;
        }
        throw new IllegalArgumentException("Kein Literal: " + type +
    " (" + value + ")");
    }
```

escapeLaTeXString

```
    private String escapeLaTeXString(String org)
    {
        char delim = 0;
        for (char c : verbDelim) {
            if (org.indexOf(c) < 0) {
                delim = c;
                break;
            }
        }
        if (delim == 0) {
            int div = org.length() / 2;
            return escapeLaTeXString(org.substring(0, div)) +
    escapeLaTeXString(org.substring(div));
        }
        return "\\verb" + delim + org + delim;
    }
```

escapeJavaString

```
private String escapeJavaString(String org)
{
    StringBuilder b = new StringBuilder(org);
    for(int i = 0 ; i < b.length() ; i++){
        char c = b.charAt(i);
        String replace;
        switch(c) {
            case '\n':
                replace = "\\n" ;
                break;
            case '\r':
                replace = "\\r" ;
                break;
            case '\t':
                replace = "\\t" ;
                break;
            case '\f':
                replace = "\\f" ;
                break;
            case '\b':
                replace = "\\b" ;
                break;
            case '\\':
                replace = "\\\\" ;
                break;
            case '\'':
                replace = "\\\'" ;
                break;
            case '\"':
                replace = "\\\"" ;
                break;
            default:
                if (c < ' ') {
                    replace = "\\0" + Integer.toOctalString(c) ;
                    break;
                }
                break;
        }
        b.replace(i, i + 1, replace) ;
        i++ ;
    }
    return b.toString() ;
}
```

printLinkedId Druckt einen Identifier als Link.

Parameter

text der Text des Links (purer Text).

el das Element, zu dem gelinkt wird.

```
public void printLinkedId(String text,
                          Doc el)
{
    appendInLine(LaTeXWriter.configuration.linker.createLink(text,
el), text.length());
}
```

1.14 Klasse de.dclj.paul.ltxdoclet.PrettyPrinter

1.14.1 Übersicht

Ein Schön-Drucker für Quelltext.

Wir implementieren `TreeVisitor`, um den Syntaxbaum abzuarbeiten.

Von außen müssen nur `printSource` und der Konstruktor verwendet werden.

@author Paul Ebermann (at mailto:paulo@heribert.local)

@version Id

1.14.2 Inhaltsverzeichnis

1.14.3 Variablen

maxLineLength

```
private int maxLineLength = 50;
```

elements

```
Elements elements;
```

types

```
Types types;
```

trees

```
Trees trees;
```

currentElement

```
private ProgramElementDoc currentElement;
```

1.14.4 Konstruktoren

PrettyPrinter Creates a new `PrettyPrinter` instance.

```
public PrettyPrinter(Elements elements,
                     Types types,
                     Trees trees)
{
    super();
    this.elements = elements;
    this.types = types;
    this.trees = trees;
}
```

1.14.5 Methoden

printSource Druckt den Quelltext zum angegebenen dokumentierten Element aus.

```
public void printSource(ProgramElementDoc doc,
                        LaTeXWriter target)
{
    this.currentElement = doc;
    Element elem = getElementForDoc(doc);
    TreePath path = trees.getPath(elem);
    if (path ≡ null)
        return;
    target.println("\\begin{sourcecode}");
    SourceFormatter f = new SourceFormatter(target);
    this.scan(path, f);
    f.flush();
    target.println("\\end{sourcecode}");
}
```

getElementForDoc Sucht das passende `Element`-Objekt zu einem dokumentierten Programmelement.

```
private Element getElementForDoc(ProgramElementDoc doc)
{
    ClassDoc klasse = doc.containingClass();
    if (klasse ≡ null) {
        Element el =
elements.getTypeElement(doc.qualifiedName());
        if (el ≡ null) {
            throw new RuntimeException("kein Element für "+ doc+
" gefunden.");
        }
```

```
            return el;
        }
        Element parent = getElementForDoc(klasse);
        return findInList(doc, parent.getEnclosedElements());
    }
```

getDocForElement Sucht das Doc für das genannte Element.

```
    private Doc getDocForElement(ExecutableElement element,
                                 ClassDoc classD)
    {
        if (element.getKind() ≡ ElementKind.CONSTRUCTOR) {
            ConstructorDoc[] cons = classD.constructors(false);
            return findInList(element, cons);
        }
        else if (element.getKind() ≡ ElementKind.METHOD) {
            MethodDoc[] meths = classD.methods(false);
            return findInList(element, meths);
        }
        else {
            throw new IllegalArgumentException(element + " (" +
    element.getKind() + ")");
        }
    }
```

findInList

```
    private Doc findInList(ExecutableElement exe,
                           ConstructorDoc[] cons)
    {
        for (ConstructorDoc doc : cons) {
            if (gleicheSignatur(exe, doc)) {
                return doc;
            }
        }
        return null;
    }
```

findInList

```
    private Doc findInList(ExecutableElement exe,
                           MethodDoc[] cons)
    {
        for (MethodDoc doc : cons) {
            if (doc.name().contentEquals(exe.getSimpleName()) &&
    gleicheSignatur(exe, doc)) {
```

```
                return doc;
            }
        }
        return null;
    }
```

findInList

```
    private Element findInList(ProgramElementDoc doc,
                               List<? extends Element> siblings)
    {
        if (doc.isConstructor()) {
            List<ExecutableElement> konstruktoren =
    ElementFilter.constructorsIn(siblings);
            for (ExecutableElement exe : konstruktoren) {
                if (gleicheSignatur(exe, (ConstructorDoc)doc)) {
                    return exe;
                }
            }
            LaTeXWriter.configuration.root.printWarning("Kein Kon-
    struktor zu "+ doc+ " gefunden. "+ "Kandidaten waren: "+
    konstruktoren);
        }
        if (doc.isMethod()) {
            List<ExecutableElement> methoden =
    ElementFilter.methodsIn(siblings);
            methodenSchleife :
                for (ExecutableElement methode : methoden) {
                    if
    (methode.getSimpleName().contentEquals(doc.name()) &&
    gleicheSignatur(methode, (MethodDoc)doc)) {
                        return methode;
                    }
                }
            LaTeXWriter.configuration.root.printWarning("Keine Metho-
    de zu "+ doc+ " gefunden. "+ "Kandidaten waren: "+
    methoden);
        }
        if (doc.isField() || doc.isEnumConstant()) {
            List<VariableElement> fields =
    ElementFilter.fieldsIn(siblings);
            for (VariableElement var : fields) {
                if (var.getSimpleName().contentEquals(doc.name())) {
                    return var;
                }
```

```
            }
        }
        if (doc instanceof ClassDoc) {
            List<TypeElement> classes =
    ElementFilter.typesIn(siblings);
            for (TypeElement klasse : classes) {
                if
    (klasse.getSimpleName().contentEquals(((Type)doc).simpleTypeName()))
    {
                    return klasse;
                }
            }
            LaTeXWriter.configuration.root.printWarning("Keine Klas-
    se zu " + doc + " gefunden. " + "Kandidaten waren: " +
    classes);
        }
        throw new RuntimeException("unbekannter Doc-Typ: " + doc +
    " [" + doc.getClass() + "]");
    }
```

gleicheSignatur Untersucht, ob ein Element und das entsprechende Dokumentationselement zusammenpassen.

```
    private boolean gleicheSignatur(ExecutableElement methode,
                                    ExecutableMemberDoc doc)
    {
        Parameter[] parameters = doc.parameters();
        List<? extends VariableElement> args =
    methode.getParameters();
        if (args.size() ≠ parameters.length)
            return false;
        for(int i = 0; i < parameters.length; i++){
            String argTtext =
    typeAsString(args.get(i).asType()).replace(", ", ",");
            if
    (!argTtext.equals(parameters[i].type().toString().replace(", ",
    ",")))
                return false;
        }
        return true;
    }
```

typeAsString Wandelt einen Typ in einen String um, um Signaturen vergleichen zu können.

```
    private String typeAsString(TypeMirror argT)
    {
```

```
        if (argT.getKind() ≡ TypeKind.TYPEVAR) {
            TypeVariable argTVar = (TypeVariable)argT;
            TypeMirror lowerBound = argTVar.getLowerBound();
            if (lowerBound.getKind() ≠ TypeKind.NULL) {
                return argT + " super " + lowerBound ;
            }
            else {
                TypeMirror upperBound = argTVar.getUpperBound();
                if (upperBound.toString().equals("java.lang.Object"))
    {
                    return argT.toString() ;
                }
                else {
                    return argT + " extends " + upperBound ;
                }
            }
        }
        else {
            return argT.toString() ;
        }
    }
```

makeLink

```
    private void makeLink(String text,
                          Element linkTarget,
                          SourceFormatter target)
    {
        RootDoc root = LaTeXWriter.configuration.root;
        Doc doc;
        switch(linkTarget.getKind()) {
            case PACKAGE: {
                PackageElement pEl = (PackageElement)linkTarget;
                doc =
    root.packageNamed(pEl.getQualifiedName().toString()) ;
                break;
            }
            case CLASS:
            case INTERFACE:
            case ENUM:
            case ANNOTATION_TYPE: {
                TypeElement type = (TypeElement)linkTarget;
                doc =
    root.classNamed(type.getQualifiedName().toString()) ;
```

```
                break;
            }
            case CONSTRUCTOR:
            case METHOD: {
                ExecutableElement method =
    (ExecutableElement)linkTarget;
                TypeElement type =
    (TypeElement)method.getEnclosingElement();
                ClassDoc typeDoc =
    root.classNamed(type.getQualifiedName().toString());
                doc = getDocForElement(method, typeDoc);
                break;
            }
            case FIELD:
            case ENUM_CONSTANT: {
                VariableElement field = (VariableElement)linkTarget;
                TypeElement type =
    (TypeElement)field.getEnclosingElement();
                ClassDoc typeDoc =
    root.classNamed(type.getQualifiedName().toString());
                doc = null;
                break;
            }
            case EXCEPTION_PARAMETER:
            case LOCAL_VARIABLE:
            case PARAMETER: {
                target.printId(text);
                return;
            }
            default:
                target.print("«[" + text + ":" + linkTarget + "]»");
                return;
        }
        makeLink(text, doc, linkTarget, target);
    }
```

makeLink setzt einen Link auf ein Programmelement.

Parameter

text der Text des Links.

doc das Dokumentationselement zum verlinkten Element. Falls `null`, ist keine Dokumentation dazu vorhanden.

linkTarget das Element, auf das gelinkt wird.

target hier wird der Text hingeschrieben.

```
void makeLink(String text,
              Doc doc,
              Element linkTarget,
              SourceFormatter target)
{
    if (doc ≡ null) {
        makeExternalLink(text, linkTarget, doc, target);
    }
    else {
        target.printLinkedId(text, doc);
    }
}
```

makeExternalLink setzt einen externen Link zu einem Element, welches nicht enthalten ist.

```
private void makeExternalLink(String text,
                              Element linkTarget,
                              Doc linkedDoc,
                              SourceFormatter target)
{
    target.print(text);
}
```

arrayLevel Zählt, wie viele Array-Level in diesem Typ sind.

```
private int arrayLevel(Tree tree)
{
    if (tree.getKind() ≠ Tree.Kind.ARRAY_TYPE) {
        return 0;
    }
    ArrayTypeTree array = (ArrayTypeTree)tree;
    return 1+ arrayLevel(array.getType());
}
```

arrayBaseType Sucht den Basistyp nach Entfernen aller Array-Level.

```
private Tree arrayBaseType(Tree tree)
{
    if (tree.getKind() ≠ Tree.Kind.ARRAY_TYPE) {
        return tree;
    }
    ArrayTypeTree array = (ArrayTypeTree)tree;
    return arrayBaseType(array.getType());
}
```

findElement Sucht ein Element in einem Scope.

```
public Element findElement(Element scope,
                           Name name,
                           TypeMirror type)
{
    if (scope.getKind() ≡ ElementKind.TYPE_PARAMETER) {
        List⟨? extends TypeMirror⟩ bndlist =
((TypeParameterElement)scope).getBounds();
        if (bndlist.isEmpty()) {
            bndlist = Collec-
tions.singletonList(elements.getTypeElement("java.lang.Object").asType());
        }
        for (TypeMirror bnd : bndlist) {
            Element subScope = types.asElement(bnd);
            Element found = findElement(subScope, name, type);
            if (found ≠ null)
                return found;
        }
        return null;
    }
    List⟨? extends Element⟩ list = (scope.getKind().isClass() ||
scope.getKind().isInterface()) ?
elements.getAllMembers((TypeElement)scope) :
scope.getEnclosedElements();
    for (Element el : list) {
        if (el.getSimpleName().equals(name)) {
            return el;
        }
    }
    Element enclosing = scope.getEnclosingElement();
    if (enclosing ≡ null)
        return null;
    return findElement(enclosing, name, type);
}
```

findElement Sucht ein Element mit angegebenen Namen und Typ in einem Scope.

```
public Element findElement(Scope scope,
                           Name name,
                           TypeMirror type)
{
    for (Element el : scope.getLocalElements()) {
        if (el.getSimpleName().equals(name)) {
```

```
                return el;
            }
        }
        TypeElement typeEl = scope.getEnclosingClass();
        if (type ≠ null) {
            Element inClass = findElement(typeEl, name, type);
            if (inClass ≠ null)
                return inClass;
        }
        Scope enclosing = scope.getEnclosingScope();
        if (enclosing ≡ null)
            return null;
        return findElement(enclosing, name, type);
    }
```

scanType Scannt einen Tree als Typ. (Dies hat eine Sonderbehandlung bei Identifier, die andernfalls als Expression aufgefasst würden.)

```
    public void scanType(Tree tree,
                         SourceFormatter target)
    {
        if (tree.getKind() ≡ Tree.Kind.IDENTIFIER) {
            TypeMirror t =
trees.getTypeMirror(TreePath.getPath(this.getCurrentPath(),
tree));
            Element el = types.asElement(t);
            this.makeLink(tree.toString(), el, target);
        }
        else {
            this.scan(tree, target);
        }
    }
```

printVariableDecl Druckt eine Variablendeklaration (ohne das Semikolon am Ende).

```
    public void printVariableDecl(VariableTree var,
                                  SourceFormatter target)
    {
        this.scan(var.getModifiers(), target);
        this.scanType(var.getType(), target);
        target.print(" ");
        this.printVariableInitializer(var, target);
    }
```

printVariableInitializer Druckt eine Variablendeklaration ohne Modifier und Typ (d.h. nur Namen und eventuell Initializer).

Dies ist für Variablendeklarationen, die mehrere Variablen gleichzeitig deklarieren, notwendig (insbesondere bei for-Schleifen).

```
public void printVariableInitializer(VariableTree var,
                                      SourceFormatter target)
{
    target.printId(var.getName());
    ExpressionTree init = var.getInitializer();
    if (init ≠ null) {
        target.printSpecial(" = ");
        this.scan(init, target);
    }
}
```

printTypeParameters Schreibt eine Liste von Typparametern oder -Argumenten, falls nicht leer, raus.

Parameter

liste die Liste der Parameter. Falls die Liste leer ist, wird gar nichts getan.

target dort schreiben wir die Daten hin.

space falls `true` und die Liste nicht leer ist, wird am Ende (nach dem `>`) noch ein Leerzeichen angehängt.

```
public void printTypeParameters(List⟨? extends Tree⟩ liste,
                                SourceFormatter target,
                                boolean space)
{
    printTypeList(liste, target, "<", ", ", space ? "> " : ">");
}
```

printForInit Druckt die Initialisierungsanweisungen einer For-Schleife. Das ist entweder eine Variablendeklaration (eventuell mehrere Variablen gleichen Typs), oder eine Reihe von StatementExpressions (d.h. die Sorte Expressions, die in ExpressionStatements vorkommen kann), verpackt in je ein ExpressionStatement. Wir müssen beim Drucken aufpassen, dass keine `;`, sondern `,` dazwischen sind.

```
public void printForInit(List⟨? extends StatementTree⟩ liste,
                         SourceFormatter target)
{
    if (liste.isEmpty()) {
        return;
    }
```

```
        StatementTree first = liste.get(0);
        switch(first.getKind()) {
            case VARIABLE: {
                VariableTree var = (VariableTree)first;
                this.printVariableDecl(var, target);
                for (StatementTree stat : liste.subList(1,
    liste.size())) {
                    var = (VariableTree)stat;
                    target.printSpecial(", ");
                    this.printVariableInitializer(var, target);
                }
                return;
            }
            case EXPRESSION_STATEMENT: {
                ExpressionStatementTree exp =
    (ExpressionStatementTree)first;
                this.scan(exp.getExpression(), target);
                for (StatementTree stat : liste.subList(1,
    liste.size())) {
                    exp = (ExpressionStatementTree)stat;
                    target.printSpecial(", ");
                    this.scan(exp.getExpression(), target);
                }
                return;
            }
            default:
                throw new
    IllegalArgumentException(first.getClass().getName());
        }
    }
```

printForUpdate Druckt den Update-Teil eines For-Statements. Dies ist eine Liste von StatementExpressions, für uns leider verpackt in je ein ExpressionStatement. Wir müssen Kommas dazwischensetzen.

```
    public void printForUpdate(List<? extends ExpressionStatementTree>
    liste,
                               SourceFormatter target)
    {
        if (liste.isEmpty()) {
            return;
        }
        ExpressionStatementTree exp = liste.get(0);
        this.scan(exp.getExpression(), target);
```

```
        for (ExpressionStatementTree stat : liste.subList(1,
    liste.size())) {
            target.print(", ");
            this.scan(stat.getExpression(), target);
        }
    }
```

printParameterList Druckt eine Parameterliste (d.h. eine Liste von Parameterdeklarationen einer Methode/eines Konstruktors).

```
    public void printParameterList(List⟨? extends VariableTree⟩
    params,
                                   SourceFormatter target)
    {
        if (params.isEmpty())
            return;
        target.pushIndent();
        this.printVariableDecl(params.get(0), target);
        for (VariableTree arg : params.subList(1, params.size())) {
            target.println(",");
            this.printVariableDecl(arg, target);
        }
        target.popIndent();
    }
```

printThrows Druckt eine Liste von throws-Deklarationen für eine Methode.

```
    public void printThrows(List⟨? extends ExpressionTree⟩
    exceptions,
                            SourceFormatter target)
    {
        target.addIndent();
        this.printTypeList(exceptions, target, "\nthrows ",
    ",\n       ", "");
        target.popIndent();
    }
```

printList Druckt eine Liste. Falls die Liste leer ist, wird gar nichts getan, ansonsten wird zunächst `prefix` ausgegeben, dann werden die einzelnen Elemente gescannt, getrennt durch `infix`, danach wird `postfix` ausgegeben.

Parameter

liste die zu druckenden Elemente.

target der Formatierer, auf dem der Inhalt gedruckt werden soll

prefix ein String, der am Anfang der Liste (falls nicht leer) gedruckt werden soll

infix ein String, der zwischen den einzelnen Listenelementen gedruckt werden soll.

postfix ein String, der am Ende der Liste (falls nicht leer) gedruckt werden soll.

Siehe auch «Link:@see:printTypeList für eine analoge Methode, die die Einträge als Typen scannt.|printTypeList|für eine analoge Methode, die die Einträge als Typen scannt.»

```
public void printList(List⟨? extends Tree⟩ liste,
                      SourceFormatter target,
                      String prefix,
                      String infix,
                      String postfix)
{
    if (liste.isEmpty())
        return;
    target.printSpecial(prefix);
    this.scan(liste.get(0), target);
    for (Tree elem : liste.subList(1, liste.size())) {
        target.printSpecial(infix);
        this.scan(elem, target);
    }
    target.printSpecial(postfix);
}
```

printTypeList Druckt eine Liste als Typen.

Falls die Liste leer ist, wird gar nichts getan, ansonsten wird zunächst `prefix` ausgegeben, dann werden die einzelnen Elemente als Typen gescannt, getrennt durch `infix`, danach wird `postfix` ausgegeben.

Parameter

liste die zu druckenden Elemente. Sie werden als Typen aufgefasst.

target der Formatierer, auf dem der Inhalt gedruckt werden soll

prefix ein String, der am Anfang der Liste (falls nicht leer) gedruckt werden soll

infix ein String, der zwischen den einzelnen Listenelementen gedruckt werden soll.

postfix ein String, der am Ende der Liste (falls nicht leer) gedruckt werden soll.

Siehe auch «Link:@see:printList für eine analoge Methode, die die Einträge nicht als Typen scannt.|printList|für eine analoge Methode, die die Einträge nicht als Typen scannt.»

```
public void printTypeList(List⟨? extends Tree⟩ liste,
                          SourceFormatter target,
                          String prefix,
                          String infix,
                          String postfix)
{
    if (liste.isEmpty())
        return;
    target.printSpecial(prefix);
    this.scanType(liste.get(0), target);
    for (Tree elem : liste.subList(1, liste.size())) {
        target.printSpecial(infix);
        this.scanType(elem, target);
    }
    target.printSpecial(postfix);
}
```

printBounds Druckt die Liste der Grenzen für einen Typparameter (eines Types oder einer Methode).

```
public void printBounds(List⟨? extends Tree⟩ liste,
                        SourceFormatter target)
{
    printTypeList(liste, target, " extends ", "&", "");
}
```

printIndented Druckt ein Statement eingerückt.

Ist dies ein Block-Statement, muss nichts weiter getan werden (außer dieses auszugeben), da es ja selbst seinen Inhalt einrückt, und eine weitere Einrückung überflüssig ist.

Andernfalls wird die Einrückung erhöht, eine neue Zeile begonnen, das Statement gedruckt und die Einrückung wieder zurückgesetzt.

```
private void printIndented(StatementTree statement,
                           SourceFormatter target)
{
    if (statement instanceof BlockTree) {
        this.scan(statement, target);
    }
    else {
        target.addIndent();
        target.println();
        this.scan(statement, target);
```

```
            target.popIndent();
        }
    }
```

printIndented Druckt eine Reihe von Anweisungen eingerückt.

```
    private void printIndented(List<? extends StatementTree>
    statements,
                               SourceFormatter target)
    {
        switch(statements.size()) {
            case 0:
                return;
            case 1:
                printIndented(statements.get(0), target);
                return;
            default:
                target.addIndent();
                this.printList(statements, target, "\n", "\n", "");
                target.popIndent();
        }
    }
```

visitMethod Druckt eine Methode (oder einen Konstruktor o.ä.).

```
    @Override
    public Void visitMethod(MethodTree meth,
                            SourceFormatter target)
    {
        this.scan(meth.getModifiers(), target);
        this.printTypeParameters(meth.getTypeParameters(), target,
    true);
        if (meth.getName().contentEquals("<init>")) {
            target.print(currentElement.name());
        }
        else {
            this.scanType(meth.getReturnType(), target);
            target.print(" ");
            target.print(meth.getName());
        }
        target.print("(");
        printParameterList(meth.getParameters(), target);
        target.print(")");
        printThrows(meth.getThrows(), target);
        target.println();
```

```
        this.scan(meth.getBody(), target);
        target.println();
        return null;
    }
```

visitVariable Druckt eine Variablendeklaration (als Statement, nicht in Parameterdeklarationen, also mit `;` am Ende.

```
    @Override
    public Void visitVariable(VariableTree var,
                              SourceFormatter target)
    {
        printVariableDecl(var, target);
        target.print(";");
        return null;
    }
```

visitClass Druckt eine Typdeklaration (Klasse, Interface, etc.)

```
    @Override
    public Void visitClass(ClassTree klasse,
                           SourceFormatter target)
    {
        target.print("«Typdeklaration: " + klasse.getSimpleName() +
    "»");
        return null;
    }
```

visitAssert Druckt ein Assert-Statement.

```
    @Override
    public Void visitAssert(AssertTree tree,
                            SourceFormatter target)
    {
        target.printSpecial("assert ");
        this.scan(tree.getCondition(), target);
        ExpressionTree detail = tree.getDetail();
        if (detail ≠ null) {
            target.printSpecial(":");
            this.scan(detail, target);
        }
        target.printSpecial(";");
        return null;
    }
```

visitBlock Druckt einen Block.

```
    @Override
    public Void visitBlock(BlockTree block,
                           SourceFormatter target)
    {
        target.printSpecial("{");
        target.addIndent();
        target.println();
        this.printList(block.getStatements(), target, "", "\n",
    "\n");
        target.popIndent();
        target.printSpecial("}");
        return null;
    }
```

visitBreak Druckt ein Break-Statement.

```
    @Override
    public Void visitBreak(BreakTree br,
                           SourceFormatter target)
    {
        Name label = br.getLabel();
        if (label ≡ null)
            target.printSpecial("break;");
        else {
            target.printSpecial("break ");
            target.printId(label);
        }
        return null;
    }
```

visitContinue Druckt ein Continue-Statement.

```
    @Override
    public Void visitContinue(ContinueTree br,
                              SourceFormatter target)
    {
        Name label = br.getLabel();
        if (label ≡ null)
            target.printSpecial("break;");
        else {
            target.printSpecial("break ");
            target.printId(label);
        }
        return null;
    }
```

visitDoWhileLoop Druckt eine Do-While-Schleife.

```
@Override
public Void visitDoWhileLoop(DoWhileLoopTree loop,
                             SourceFormatter target)
{
    StatementTree stat = loop.getStatement();
    target.printSpecial("do ");
    this.printIndented(stat, target);
    target.print(stat.getKind() ≡ Tree.Kind.BLOCK ? "\n" : " ");
    target.printSpecial("while");
    this.scan(loop.getCondition(), target);
    target.printSpecial(";");
    return null;
}
```

visitEmptyStatement Druckt ein leeres Statement.

```
@Override
public Void visitEmptyStatement(EmptyStatementTree empty,
                                SourceFormatter target)
{
    target.printSpecial(";");
    return null;
}
```

visitEnhancedForLoop Druckt eine for-each-Schleife.

```
@Override
public Void visitEnhancedForLoop(EnhancedForLoopTree loop,
                                 SourceFormatter target)
{
    target.printSpecial("for ");
    target.printSpecial("(");
    this.printVariableDecl(loop.getVariable(), target);
    target.printSpecial(":");
    this.scan(loop.getExpression(), target);
    target.printSpecial(") ");
    this.printIndented(loop.getStatement(), target);
    return null;
}
```

visitExpressionStatement Druckt ein Expression-Statement.

```
@Override
public Void visitExpressionStatement(ExpressionStatementTree
stat,
                                     SourceFormatter target)
{
    this.scan(stat.getExpression(), target);
    target.printSpecial(";");
    return null;
}
```

visitForLoop Druckt eine for-Schleife.

```
@Override
public Void visitForLoop(ForLoopTree loop,
                         SourceFormatter target)
{
    target.printSpecial("for");
    target.printSpecial("(");
    this.printForInit(loop.getInitializer(), target);
    target.printSpecial(";");
    this.scan(loop.getCondition(), target);
    target.printSpecial(";");
    this.printForUpdate(loop.getUpdate(), target);
    target.printSpecial(")");
    this.printIndented(loop.getStatement(), target);
    return null;
}
```

visitIf Druckt ein if-Statement.

```
@Override
public Void visitIf(IfTree ifS,
                    SourceFormatter target)
{
    target.printSpecial("if ");
    this.scan(ifS.getCondition(), target);
    target.print(" ");
    this.printIndented(ifS.getThenStatement(), target);
    StatementTree elseTree = ifS.getElseStatement();
    if (elseTree ≠ null) {
        target.println();
        target.printSpecial("else ");
        if (elseTree instanceof IfTree) {
```

```
                this.scan(elseTree, target);
            }
            else {
                this.printIndented(elseTree, target);
            }
        }
        return null;
    }
```

visitLabeledStatement Druckt ein benanntes Statement.

```
    @Override
    public Void visitLabeledStatement(LabeledStatementTree stat,
                                      SourceFormatter target)
    {
        target.printId(stat.getLabel());
        target.printSpecial(":");
        target.addIndent();
        target.println();
        this.scan(stat.getStatement(), target);
        target.popIndent();
        return null;
    }
```

visitReturn Druckt ein Return-Statement.

```
    @Override
    public Void visitReturn(ReturnTree ret,
                            SourceFormatter target)
    {
        ExpressionTree exp = ret.getExpression();
        if (exp == null) {
            target.printSpecial("return;");
        }
        else {
            target.printSpecial("return ");
            this.scan(exp, target);
            target.printSpecial(";");
        }
        return null;
    }
```

visitSwitch Druckt ein Switch-Statement.

```
@Override
public Void visitSwitch(SwitchTree st,
                        SourceFormatter target)
{
    target.printSpecial("switch");
    this.scan(st.getExpression(), target);
    target.print(" ");
    target.printSpecial("{");
    target.addIndent();
    target.println();
    this.scan(st.getCases(), target);
    target.popIndent();
    target.printSpecial("}");
    return null;
}
```

visitSynchronized Druckt ein synchronized-Statement.

```
@Override
public Void visitSynchronized(SynchronizedTree syn,
                              SourceFormatter target)
{
    target.printSpecial("synchronized");
    target.printSpecial("(");
    this.scan(syn.getExpression(), target);
    target.printSpecial(")");
    target.print(" ");
    this.scan(syn.getBlock(), target);
    return null;
}
```

visitThrow Druckt ein Throw-Statement.

```
@Override
public Void visitThrow(ThrowTree tree,
                       SourceFormatter target)
{
    target.printSpecial("throw ");
    this.scan(tree.getExpression(), target);
    target.printSpecial(";");
    return null;
}
```

visitTry Druckt ein Try-Statement.

```
@Override
public Void visitTry(TryTree tree,
                     SourceFormatter target)
{
    target.printSpecial("try ");
    this.scan(tree.getBlock(), target);
    printList(tree.getCatches(), target, "\n", "\n", "");
    BlockTree fin = tree.getFinallyBlock();
    if (fin ≠ null) {
        target.println();
        target.printSpecial("finally ");
        this.scan(fin, target);
    }
    return null;
}
```

visitWhileLoop Druckt eine While-Schleife.

```
@Override
public Void visitWhileLoop(WhileLoopTree loop,
                           SourceFormatter target)
{
    target.printSpecial("while");
    this.scan(loop.getCondition(), target);
    this.printIndented(loop.getStatement(), target);
    return null;
}
```

visitAnnotation Druckt eine Annotation.

```
@Override
public Void visitAnnotation(AnnotationTree tree,
                            SourceFormatter target)
{
    target.printSpecial("@");
    this.scanType(tree.getAnnotationType(), target);
    this.printList(tree.getArguments(), target, "(", ",", ")");
    return null;
}
```

visitArrayAccess Druckt einen Array-Zugriff.

```
@Override
public Void visitArrayAccess(ArrayAccessTree tree,
                             SourceFormatter target)
{
    this.scan(tree.getExpression(), target);
    target.printSpecial("[");
    this.scan(tree.getIndex(), target);
    target.printSpecial("]");
    return null;
}
```

visitAssignment Druckt eine Zuweisung.

```
@Override
public Void visitAssignment(AssignmentTree tree,
                            SourceFormatter target)
{
    this.scan(tree.getVariable(), target);
    target.printSpecial(" = ");
    this.scan(tree.getExpression(), target);
    return null;
}
```

visitBinary Druckt einen binären Ausdruck.

```
@Override
public Void visitBinary(BinaryTree bin,
                        SourceFormatter target)
{
    this.scan(bin.getLeftOperand(), target);
    target.printSpecial(bin.getKind());
    this.scan(bin.getRightOperand(), target);
    return null;
}
```

visitCompoundAssignment Druckt einen Rechnung-und-Zuweisungs-Ausdruck.

```
@Override
public Void visitCompoundAssignment(CompoundAssignmentTree tree,
                                    SourceFormatter target)
{
    this.scan(tree.getVariable(), target);
    target.printSpecial(tree.getKind());
    this.scan(tree.getExpression(), target);
    return null;
}
```

visitConditionalExpression Druckt eine bedingten Ausdruck `a ? b : c` .

```
@Override
public Void visitConditionalExpression(ConditionalExpressionTree
tree,
                                       SourceFormatter target)
{
    this.scan(tree.getCondition(), target);
    target.printSpecial("?");
    this.scan(tree.getTrueExpression(), target);
    target.printSpecial(":");
    this.scan(tree.getFalseExpression(), target);
    return null;
}
```

visitIdentifier Druckt einen Identifier (als Expression).

```
@Override
public Void visitIdentifier(IdentifierTree id,
                            SourceFormatter target)
{
    target.printId(id.getName());
    return null;
}
```

visitInstanceOf Druckt einen instanceof-Ausdruck.

```
@Override
public Void visitInstanceOf(InstanceOfTree tree,
                            SourceFormatter target)
{
    this.scan(tree.getExpression(), target);
    target.printSpecial(" instanceof ");
    this.scanType(tree.getType(), target);
    return null;
}
```

visitLiteral Druckt ein Literal.

```
@Override
public Void visitLiteral(LiteralTree literal,
                         SourceFormatter target)
{
    target.printLiteral(literal.getValue(), literal.getKind());
    return null;
}
```

visitMemberSelect Druckt einen Variablen-Zugriffs-Ausdruck.

```
@Override
public Void visitMemberSelect(MemberSelectTree tree,
                              SourceFormatter target)
{
    this.scan(tree.getExpression(), target);
    target.printSpecial(".");
    target.printId(tree.getIdentifier());
    return null;
}
```

visitMethodInvocation Druckt einen Methodenaufruf.

Hier liegt das Problem darin, dass wir einen Selector, einige Typparameter und eine Argumentenliste haben, und die Typparameter, falls vorhanden, syntaktisch eigentlich in der Mitte des Selectors (d.h. vor seinem letzten Identifier) stehen müssen.

```
@Override, @SuppressWarnings(value = "ungültig")
public Void visitMethodInvocation(MethodInvocationTree tree,
                                  SourceFormatter target)
{
    ExpressionTree selector = tree.getMethodSelect();
    Name identifier;
    TypeMirror containing;
    Element methodElement;
    TreePath selectorPath =
TreePath.getPath(this.getCurrentPath(), selector);
    TypeMirror t = trees.getTypeMirror(selectorPath);
    switch(selector.getKind()) {
        case MEMBER_SELECT:
            MemberSelectTree msTree = (MemberSelectTree)selector;
            ExpressionTree expr = msTree.getExpression();
            TreePath exprPath = TreePath.getPath(selectorPath,
expr);
            containing = trees.getTypeMirror(exprPath);
            this.scan(expr, target);
            target.printSpecial(".");
            identifier = msTree.getIdentifier();
            methodElement =
findElement(types.asElement(containing), identifier, t);
            break;
        case IDENTIFIER:
            identifier = ((IdentifierTree)selector).getName();
            Scope s;
```

```
                s = trees.getScope(selectorPath);
                methodElement = findElement(s, identifier, t);
                break;
            default:
                throw new
    IllegalArgumentException("Methoden-Selector: [" + selector +
    "] (" + selector.getKind() + ")");
        }
        this.printTypeList(tree.getTypeArguments(), target, "<",
    ", ", ">");
        this.makeLink(identifier.toString(), methodElement, target);
        target.print("(");
        this.printList(tree.getArguments(), target, "", ", ", "");
        target.print(")");
        return null;
    }
```

visitNewArray Druckt einen Array-Erstellungs-Ausdruck.

```
    @Override
    public Void visitNewArray(NewArrayTree tree,
                              SourceFormatter target)
    {
        int[][] beispiel = new int[2][];
        int[][] beispiel1 = new int[2][3];
        int[][] beispiel2 = new int[][]{ { 1, 1 }, { 2, 2 } };
        Tree type = tree.getType();
        if (type ≠ null) {
            target.printSpecial("new ");
            Tree baseType = arrayBaseType(type);
            int level = arrayLevel(type) + 1;
            List⟨? extends ExpressionTree⟩ dims =
    tree.getDimensions();
            this.scanType(baseType, target);
            this.printList(dims, target, "[", "][", "]");
            for(int i = dims.size(); i < level; i++){
                target.printSpecial("[]");
            }
        }
        List⟨? extends ExpressionTree⟩ init = tree.getInitializers();
        if (init ≠ null) {
            this.printList(init, target, "{ ", ", ", " }");
        }
        return null;
    }
```

visitNewClass Druckt einen Exemplarerstellungsausdruck.

```
@Override
public Void visitNewClass(NewClassTree tree,
                          SourceFormatter target)
{
    ExpressionTree enclosing = tree.getEnclosingExpression();
    if (enclosing ≠ null) {
        this.scan(enclosing, target);
        target.printSpecial(".");
    }
    target.printSpecial("new ");
    this.printTypeList(tree.getTypeArguments(), target, "<",
", ", "> ");
    this.scanType(tree.getIdentifier(), target);
    target.printSpecial("(");
    this.printList(tree.getArguments(), target, "", ", ", "");
    target.printSpecial(")");
    ClassTree body = tree.getClassBody();
    if (body ≠ null) {
        this.scan(body, target);
    }
    return null;
}
```

visitParenthesized Druckt einen eingeklammerten Ausdruck.

```
@Override
public Void visitParenthesized(ParenthesizedTree tree,
                               SourceFormatter target)
{
    target.printSpecial("(");
    target.pushIndent();
    this.scan(tree.getExpression(), target);
    target.printSpecial(")");
    target.popIndent();
    return null;
}
```

visitTypeCast Druckt einen Type-Cast-Ausdruck.

```
@Override
public Void visitTypeCast(TypeCastTree tree,
                          SourceFormatter target)
{
```

```
        target.printSpecial("(");
        this.scanType(tree.getType(), target);
        target.printSpecial(")");
        this.scan(tree.getExpression(), target);
        return null;
    }
```

visitUnary Druckt einen unären Ausdruck.

```
    @Override
    public Void visitUnary(UnaryTree tree,
                           SourceFormatter target)
    {
        switch(tree.getKind()) {
            case POSTFIX_DECREMENT:
            case POSTFIX_INCREMENT:
                this.scan(tree.getExpression(), target);
                target.printSpecial(tree.getKind());
                return null;
        }
        target.printSpecial(tree.getKind());
        this.scan(tree.getExpression(), target);
        return null;
    }
```

visitArrayType Druckt einen Array-Typ.

```
    @Override
    public Void visitArrayType(ArrayTypeTree arrayType,
                               SourceFormatter target)
    {
        this.scan(arrayType.getType(), target);
        target.print("[]");
        return null;
    }
```

visitParameterizedType Druckt einen parametrisierten Typ.

```
    @Override
    public Void visitParameterizedType(ParameterizedTypeTree pType,
                                       SourceFormatter target)
    {
        this.scanType(pType.getType(), target);
        List<? extends Tree> params = pType.getTypeArguments();
        this.printTypeParameters(params, target, false);
        return null;
    }
```

visitPrimitiveType Druckt den Namen eines primitiven Typs.

```
@Override
public Void visitPrimitiveType(PrimitiveTypeTree prim,
                               SourceFormatter target)
{
    target.print(prim.getPrimitiveTypeKind().toString().toLowerCase());
    return null;
}
```

visitWildcard Druckt einen Wildcard-Typ (ohne folgendes Leerzeichen).

```
@Override
public Void visitWildcard(WildcardTree wildcard,
                          SourceFormatter target)
{
    target.print("?");
    switch(wildcard.getKind()) {
        case UNBOUNDED_WILDCARD:
            return null;
        case SUPER_WILDCARD:
            target.print(" super ");
            break;
        case EXTENDS_WILDCARD:
            target.print(" extends ");
            break;
        default:
            throw new IllegalArgumentException(wildcard +
" hat Kind " + wildcard.getKind());
    }
    this.scanType(wildcard.getBound(), target);
    return null;
}
```

visitModifiers Druckt die Modifiers aus, mit je einem Leerzeichen dazwischen und einem danach.

```
@Override
public Void visitModifiers(ModifiersTree mods,
                           SourceFormatter target)
{
    this.printList(mods.getAnnotations(), target, "", ", ",
"\n");
    Set<Modifier> modSet = mods.getFlags();
    for (Modifier m : modSet) {
        target.printSpecial(m.toString());
```

```
            target.print(" ");
        }
        return null;
    }
```

visitCase Druckt einen Zweig eines Switch-Statements.

```
    @Override
    public Void visitCase(CaseTree cTree,
                          SourceFormatter target)
    {
        ExpressionTree exp = cTree.getExpression();
        if (exp ≡ null) {
            target.printSpecial("default: ");
        }
        else {
            target.printSpecial("case ");
            this.scan(exp, target);
            target.printSpecial(": ");
        }
        printIndented(cTree.getStatements(), target);
        target.println();
        return null;
    }
```

visitCatch Druckt einen Catch-Zweig eines Throw-Statements.

```
    @Override
    public Void visitCatch(CatchTree cat,
                           SourceFormatter target)
    {
        target.printSpecial("catch");
        target.printSpecial("(");
        this.printVariableDecl(cat.getParameter(), target);
        target.printSpecial(")");
        this.scan(cat.getBlock(), target);
        return null;
    }
```

visitTypeParameter Druckt eine Typparameterdeklaration.

```
    @Override
    public Void visitTypeParameter(TypeParameterTree param,
                                   SourceFormatter target)
    {
```

```
        target.print(param.getName()) ;
        printBounds(param.getBounds(), target) ;
        return null ;
    }
```

1.15 Interface de.dclj.paul.ltxdoclet.LinkCreator

1.15.1 Übersicht

Eine Schnittstelle für Objekte, welche Links auf Programmelemente erstellen können.

@author **Paul Ebermann** (at mailto:paulo@heribert.local)

@version Id

1.15.2 Inhaltsverzeichnis

1.15.3 Methoden

createLink

```
public String createLink(String label,
                         Doc target)
```

1.16 Klasse de.dclj.paul.ltxdoclet.TableInfo

1.16.1 Übersicht

Unterstützung für die Umwandlung von HTML-Tabellen in LaTeX-Tabellen. (Aus dem **TexDoclet** (at https://texdoclet.dev.java.net/) von Gregg Wonderly, leicht angepasst an unser LaTeX-Doclet.)

This class provides support for converting HTML tables into LaTeX tables. Some of the things **NOT** implemented include the following:

- valign attributes are not procesed, but align= is.
- rowspan attributes are not processed, but colspan= is.
- the argument to border= in the table tag is not used to control line size

Here is an example table.

Column 1 Heading	Column two heading	Column three heading
data	Span two columns	
more data	right	left

A nested table example

Column 1 Heading	Column two heading	Column three heading
data	Span two columns	
more data	right	left
1 2 3 4	first line second line third line fourth line	

@version Id

@author Gregg Wonderly (at mailto:gregg.wonderly@pobox.com) (small changes by Paul Ebermann to integrate it in my LaTeX-Doclet.)

Siehe auch `HtmlKonverter`

1.16.2 Inhaltsverzeichnis

1.16.3 Variablen

rowcnt

```
private int rowcnt = 0;
```

colcnt

```
private int colcnt = 0;
```

border

```
private boolean border = false;
```

colopen

```
private boolean colopen = false;
```

props

```
private Map⟨String, String⟩ props;
```

bordwid

```
private int bordwid;
```

parboxed

```
private boolean parboxed;
```

rowopen

```
private boolean rowopen;
```

tblcnt

```
static int tblcnt;
```

tblno

```
int tblno;
```

tc

```
String tc;
```

1.16.4 Konstruktoren

TableInfo Constructs a new table object and starts processing of the table by scanning the `<table>` passed to count columns.

Parameter

p properties found on the `<table>` tag

ret the result buffer that will contain the output

table the input string that has the entire table definition in it.

off the offset into `<table>` where scanning should start

```
public TableInfo(Map⟨String, String⟩ p,
                 StringBuilder ret,
                 String table,
                 int off)
{
    super();
    props = p;
    if (p ≡ null)
        return;
    tblno = tblcnt++;
    tc = countName(tblno);
    String val = p.get("border");
    border = false;
    if (val ≠ null) {
        border = true;
        bordwid = 2;
        if (val.equals("") ≡ false) {
            try {
                bordwid = Integer.parseInt(val);
            }
            catch(Exception ex){
            }
            if (bordwid ≡ 0)
```

```
                border = false;
            }
        }
        ret.append("\n% Table #" + tblno + "\n");
        int col = 0;
        int row = 0;
        for(int i = off; i < table.length(); ++i){
            if (table.charAt(i) == '<') {
                if (table.length() > i + 6 && table.substring(i, i +
7).equalsIgnoreCase("</table")) {
                    ret.append("\\noprint{</table}");
                    break;
                }
                else if (table.length() > i + 3 && table.substring(i,
i + 4).equalsIgnoreCase("</tr")) {
                    ret.append("\\noprint{</tr}");
                    break;
                }
                else if (table.length() > i + 2 && table.substring(i,
i + 3).equalsIgnoreCase("<tr")) {
                    ret.append("\\noprint{<tr}");
                    if (row++ > 0) {
                        break;
                    }
                }
                else if (table.length() > i + 2 && table.substring(i,
i + 3).equalsIgnoreCase("<td")) {
                    ret.append("\\noprint{<td}");
                    Map<String, String> pp = new HashMap<String,
String>();
                    int idx = HtmlKonverter.getTagAttrs(table, pp, i +
3);
                    int v = hasNumProp("colspan", pp);
                    if (v > 0)
                        col+=v;
                    else
                        col++;
                    i = idx - 1;
                }
                else if (table.length() > i + 2 && table.substring(i,
i + 3).equalsIgnoreCase("<th")) {
                    ret.append("\\noprint{<th}");
```

```
                Map<String, String> pp = new HashMap<String,
String>();
                int idx = HtmlKonverter.getTagAttrs(table, pp, i+
3);
                int v = hasNumProp("colspan", pp);
                if (v > 0)
                    col+=v;
                else
                    col++;
                i = idx - 1;
            }
        }
    }
    ret.append("\\noprint{col=" + col + "}");
    if (col == 0)
        col = 1;
    for(int i = 0; i < col; ++i){
        String cc = countName(i);
        ret.append("\\newlength{\\tbl" + tc + "c" + cc + "w}\n");
        ret.append("\\setlength{\\tbl" + tc + "c" + cc + "w}{" +
(1.0 / col) + "\\hsize}\n");
    }
    ret.append("\\begin{tabular}{");
    if (border)
        ret.append("|");
    for(int i = 0; i < col; ++i){
        String cc = countName(i);
        ret.append("p{\\tbl" + tc + "c" + cc + "w}");
        if (border)
            ret.append("|");
    }
    ret.append("}\n");
}
```

1.16.5 Methoden

hasNumProp

```
int hasNumProp(String prop,
               Map<String, String> p)
{
    String val = p.get(prop);
    if (val == null)
```

```
            return -1;
        try {
            return Integer.parseInt(val);
        }
        catch(Exception ex){
            return -1;
        }
    }
```

countName Wandelt eine Nummer in die dazugehörige Zeichenkette im 26-er-System (a-z) um.

```
public String countName(int i)
{
    return ""+ (char)('a'+ ((i / 26) / 26))+ (char)('a'+ ((i /
26) % 26))+ (char)('a'+ (i % 26));
}
```

colSize Gibt zu einer Spaltennummer (in der aktuellen Tabelle) die dazugehörige Breiten-Kontrollsequenz zurück.

```
public String colSize(int colNum)
{
    return "\\tbl"+ tc+ "c"+ countName(colNum)+ "w";
}
```

startCol Starts a new column, possibly closing the current column if needed

Parameter

ret the output buffer to put LaTeX into

p the properties from the `<td>` tag

```
public void startCol(StringBuilder ret,
                     Map⟨String, String⟩ p)
{
    endCol(ret);
    int span = hasNumProp("colspan", p);
    if (colcnt> 0) {
        ret.append(" & ");
    }
    String align = p.get("align");
    if (align ≠ null && span< 0)
        span = 1;
    if (span> 0) {
        StringBuilder spanSize = new StringBuilder();
```

```
            spanSize.append("\\dimexpr");
            spanSize.append(colSize(colcnt));
            for(int i = 1; i < span; i++){
                spanSize.append(" + ");
                spanSize.append(colSize(colcnt + i));
            }
            spanSize.append(" - 2ex");
            spanSize.append("\\relax");
            ret.append("\\multicolumn{" + span + "}{");
            if (border && colcnt == 0)
                ret.append("|");
            String cc = countName(colcnt);
            if (align != null) {
                String h = align.substring(0, 1);
                if ("rR".indexOf(h) >= 0)
                    ret.append("r");
                else if ("lL".indexOf(h) >= 0)
                    ret.append("p{" + spanSize + "}");
                else if ("cC".indexOf(h) >= 0)
                    ret.append("p{" + spanSize + "}");
            }
            else {
                ret.append("p{" + spanSize + "}");
            }
            if (border)
                ret.append("|");
            ret.append("}");
        }
        String wid = p.get("texwidth");
        ret.append("{");
        if (wid != null) {
            ret.append("\\parbox{" + wid + "}{\\vskip 1ex ");
            parboxed = true;
        }
        colcnt++;
        colopen = true;
    }
```

startHeadCol Starts a new Heading column, possibly closing the current column if needed. A Heading column has a Bold Face font directive around it.

Parameter

ret the output buffer to put LaTeX into

p the properties from the `<th>` tag

```
public void startHeadCol(StringBuilder ret,
                         Map⟨String, String⟩ p)
{
    startCol(ret, p);
    ret.append("\\bfseries ");
}
```

endCol Ends the current column.

Parameter

ret the output buffer to put LaTeX into

```
public void endCol(StringBuilder ret)
{
    if (colopen) {
        colopen = false;
        if (parboxed)
            ret.append("\\vskip 1ex}");
        parboxed = false;
        ret.append("}");
    }
}
```

startRow Starts a new row, possibly closing the current row if needed

Parameter

ret the output buffer to put LaTeX into

p the properties from the `<tr>` tag

```
public void startRow(StringBuilder ret,
                     Map⟨String, String⟩ p)
{
    endRow(ret);
    if (rowcnt ≡ 0) {
        if (border)
            ret.append(" \\hline ");
    }
    colcnt = 0;
    ++rowcnt;
    rowopen = true;
}
```

endRow Ends the current row.

Parameter

ret the output buffer to put LaTeX into

```
public void endRow(StringBuilder ret)
{
    if (rowopen) {
        endCol(ret);
        ret.append(" \\tabularnewline");
        if (border)
            ret.append(" \\hline");
        rowopen = false;
        ret.append("\n");
    }
}
```

endTable Ends the table, closing the last row as needed

Parameter

ret the output buffer to put LaTeX into

```
public void endTable(StringBuilder ret)
{
    endRow(ret);
    ret.append("\\end{tabular}\n");
}
```

1.17 Klasse de.dclj.paul.ltxdoclet.LtxDocletConfiguration

1.17.1 Übersicht

Konfiguration für unser Doclet.

1.17.2 Inhaltsverzeichnis

1.17.3 Variablen

destdir Das Verzeichnis, in dem alle erzeugten Daten abgelegt werden sollen.

Wird durch »-d« festgelegt.

```
public File destdir;
```

root Die zu dokumentierende Software ist in diesem `RootDoc`-Objekt versteckt.

```
public RootDoc root;
```

packages Die Liste der Packages.

```
public PackageDoc[] packages;
```

doctitle Der Titel des Dokumentes. Wird durch »-doctitle« festgelegt.

```
public String doctitle;
```

docencoding Das zu verwendende Encoding für die Ausgabe-Dateien. Wird durch »-docencoding« festgelegt, wie auch beim Standarddoclet.

```
public Charset docencoding;
```

includeSource Sollen Quelltexte mit aufgenommen werden? Falls kein Compiler gefunden wird, wird das automatisch auf false gesetzt, auch wenn die entsprechende Option gesetzt war.

```
public boolean includeSource = false;
```

threads

```
public List⟨Thread⟩ threads = Collections.synchronizedList(new
ArrayList⟨Thread⟩());
```

wasError

```
public boolean wasError;
```

javacOptions

```
Set⟨String⟩ javacOptions = new HashSet⟨String⟩(Arrays.asList(new
String[]{ "-classpath", "-sourcepath", "-encoding", "-source"
}));
```

linker Dieses Objekt erstellt Links.

```
public LinkCreator linker;
```

pp

```
PrettyPrinter pp;
```

optionLengths

```
private Map⟨String, Integer⟩ optionLengths;
```

1.17.4 Konstruktoren

LtxDocletConfiguration

```
public LtxDocletConfiguration()
{
    super();
}
```

1.17.5 Methoden

startCompiler Startet den Java-Compiler, um Quelltexte einfügen zu können. Diese Methode wird nur aufgerufen, wenn die entsprechende Option gesetzt war.

Falls es beim Compilieren einen Fehler gibt, wird das Quelltextanzeigen abgeschaltet und ein Fehler ausgegeben.

```
void startCompiler()
{
    JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();
    if (compiler ≡ null) {
        root.printError("Es gibt keinen Java-Compiler.");
        this.includeSource = false;
        return;
    }
    StandardJavaFileManager fileManager =
compiler.getStandardFileManager(null, null, null);
    Set⟨File⟩ files = new HashSet⟨File⟩();
    for (ClassDoc klasse : root.classes()) {
        files.add(klasse.position().file());
    }
    for (PackageDoc pckg : root.specifiedPackages()) {
        if (pckg.position() ≠ null) {
            files.add(pckg.position().file());
        }
    }
    Iterable⟨? extends JavaFileObject⟩ jFiles =
fileManager.getJavaFileObjectsFromFiles(files);
    List⟨String⟩ params = new ArrayList⟨String⟩();
    for (String[] option : root.options()) {
        if (javacOptions.contains(option[0])) {
            params.addAll(Arrays.asList(option));
        }
    }
    CompilationTask task = compiler.getTask(null, fileManager,
null, params, null, jFiles);
    if (!(task instanceof JavacTask)) {
        root.printError("Der CompilerTask "+ task+ " ist kein "
+ "JavacTask, damit können wir leider "+
"die Tree-API nicht verwenden und daher "+
"keinen Quelltext ausdrucken.");
        this.includeSource = false;
        return;
```

```
        }
        JavacTask jtask = (JavacTask)task;
        try {
            jtask.parse();
            jtask.analyze();
        }
        catch(IOException io){
            throw new RuntimeException(io);
        }
        this.pp = new PrettyPrinter(jtask.getElements(),
    jtask.getTypes(), Trees.instance(jtask));
        root.printNotice("Javac-Task: " + jtask);
    }
```

setOptions Merkt sich die Optionen aus `rd`.

```
    public void setOptions(RootDoc rd)
    {
        this.docencoding = Charset.defaultCharset();
        this.destdir = new File(System.getProperty("user.dir"));
        this.doctitle = "Die Package-Sammlung";
        UniversalLinkCreator lc = new UniversalLinkCreator();
        this.linker = lc;
        this.root = rd;
        root.printNotice("Lese Optionen ...");
        for (String[] op : rd.options()) {
            root.printNotice("Option: " + Arrays.toString(op));
            if (op[0].equals("-d")) {
                this.destdir = new File(op[1]);
                destdir.mkdirs();
            }
            if (op[0].equals("-doctitle")) {
                this.doctitle = op[1];
            }
            if (op[0].equals("-docencoding")) {
                this.docencoding = Charset.forName(op[1]);
            }
            if (op[0].equals("-includesource")) {
                this.includeSource = true;
            }
            if (op[0].startsWith("-link")) {
                lc.addOption(op);
```

```
            }
        }
        this.packages = rd.specifiedPackages();
        root.printNotice("... Optionen gelesen.");
    }
```

optionLength Ermittelt, ob dieses Doclet eine Option annimmt, und wenn ja, wie viele Argumente sie nimmt.

Rückgabewert die Anzahl der Kommandozeilenargumente, die diese Option darstellen, inklusive der Option selbst.

```
    public int optionLength(String option)
    {
        if ("-help".equals(option)) {
            System.out.println(optionHelp());
            return 1;
        }
        Integer r = optionLengths.get(option);
        return r ≡ null ? 0 : r;
    }
```

optionHelp prints some help about the options. We will load the text from the resource bundle.

```
    public String optionHelp()
    {
        ResourceBundle bundle =
ResourceBundle.getBundle("de.dclj.paul.ltxdoclet.help", new
HelpTextBundleControl());
        Charset cs = Charset.defaultCharset();
        String text = bundle.getString("help");
        if (!"␣...␣".contentEquals(cs.decode(cs.encode("␣...␣")))) {
            text = text.replace('␣', '<').replace('␣', '>');
        }
        return MessageFormat.format(text, cs);
    }
```

validOptions

```
    public boolean validOptions(String[][] options,
                                DocErrorReporter rep)
    {
        return true;
    }
```

toRefLabel Erstellt den Label-Namen für das angegebene Programmelement.

```
public String toRefLabel(Doc doc)
{
    if (doc instanceof PackageDoc) {
        return doc + "-package";
    }
    if (doc instanceof ClassDoc) {
        return doc + "-class";
    }
    if (doc instanceof RootDoc) {
        return "over-view";
    }
    return removeSpaces(doc.toString());
}
```

removeSpaces

```
private String removeSpaces(String t)
{
    StringBuilder b = new StringBuilder(t);
    int index = 0;
    while((index = b.indexOf(" ", index)) ≥ 0){
        b.deleteCharAt(index);
    }
    return b.toString();
}
```

toRefLabel Erstellt den Label-Namen für das angegebene Programmelement.

```
public String toRefLabel(Element element)
{
    switch(element.getKind()) {
        case CLASS:
        case INTERFACE:
        case ENUM:
        case ANNOTATION_TYPE:
            return element + "-class";
        case PACKAGE:
            return element + "-package";
        default:
            return element.toString();
    }
}
```

toInputFileName

```
String toInputFileName(PackageDoc d)
{
    return d.toString().replace('.', '/');
}
```

toOutputFileName

```
File toOutputFileName(PackageDoc d)
{
    String newName = d.toString().replace('.', '/');
    File dir = new File(destdir, newName);
    dir.mkdirs();
    return dir;
}
```

1.18 Klasse de.dclj.paul.ltxdoclet.HtmlKonverter

1.18.1 Übersicht

Ein Konverter von HTML zu LaTeX. Hergestellt auf Basis von Teilen von TeXDoclet von Gregg Wonderly, im Original zu finden unter **texdoclet.dec.java.net** (at https://texdoclet.dev.java.net/).

Supported HTML tags within comments include the following

<dl> with the associated <dt><dd></dl>tags

<p> but not align=center...yet

 but not clear=xxx

<table> including all the associcated <td><th><tr></td></th></tr>

 ordered lists

 unordered lists

<font> font coloring

<pre> preformatted text

<code> fixed point fonts

<i> italized fonts

<b> bold fonts

@version Id

@author **Gregg Wonderly** (at mailto:gregg.wonderly@pobox.com) (TeXDoclet), Paul Ebermann (Umwandelung zu HtmlKonverter, einige Anpassungen)

Siehe auch `TableInfo`

1.18.2 Inhaltsverzeichnis

1.18.3 Variablen

tblstk Die zur Zeit offenen und von inneren Tabellen verdeckten Tabellen.

```
Deque⟨TableInfo⟩ tblstk;
```

tblinfo Die aktuell geöffnete Tabelle.

```
TableInfo tblinfo;
```

verbat Wie viele verschachtelte <pre>/<code>-Umgebungen sind gerade offen?

```
int verbat = 0;
```

colors Die im aktuellen Dokument definierten Farben. Schlüssel sind die im HTML definierten Farben (red, blue, ...) sowie hexadezimal kodierte Farben (aab732), Werte die jeweils dazugehörigen Farbnamen im LaTeX-Dokument.

```
static Map⟨String, String⟩ colors;
```

colorIdx Die Nummer der nächsten zu definierenden Farbe.

```
static int colorIdx = 0;
```

labno

```
static int labno = 0;
```

refs

```
static Hashtable⟨String, String⟩ refs = new Hashtable⟨String,
String⟩();
```

block

```
static String block = "";
```

refurl

```
static String refurl = "";
```

refimg

```
static String refimg = "";
```

collectBlock

```
static boolean collectBlock;
```

chapt

```
static int chapt = 0;
```

textdepth

```
static int textdepth = 0;
```

1.18.4 Konstruktoren

HtmlKonverter Konstruktor.

```
public HtmlKonverter()
{
    super();
    tblinfo = new TableInfo(null, null, "", 0);
    tblstk = new ArrayDeque⟨TableInfo⟩();
}
```

1.18.5 Methoden

init Initialisiert die Farb-Tabelle. Dies wird automatisch vor der ersten Verwendung der Klasse aufgerufen, ist also nur notwendig, falls innerhalb einer Ausführung des Programmes (ohne Neuladen dieser Klasse) mehrere LaTeX-Dokumente erstellt werden sollen.

```
static void init()
{
    colors = new HashMap⟨String, String⟩(13);
    for (String colName : new String[]{ "red", "green", "blue",
"white", "yellow", "black", "cyan", "magenta" }) {
        colors.put(colName, colName);
    }
}
```

refName

```
static String refName(String key)
{
    String lab;
    if ((lab = refs.get(key)) ≡ null) {
        lab = "l" + labno++;
        refs.put(key, lab);
    }
    return lab;
}
```

stackTable

```
void stackTable(Map⟨String, String⟩ p,
                StringBuilder ret,
                String txt,
                int off)
{
    tblstk.push(tblinfo);
    tblinfo = new TableInfo(p, ret, txt, off);
}
```

processBlock

```
    void processBlock(String block,
                      StringBuilder ret)
    {
        if (block.substring(0, 6).equalsIgnoreCase("@link ")) {
            block = block.substring(6).trim() ;
            StringTokenizer st = new StringTokenizer(block,
    " \n\r\t");
            String key = st.nextToken();
            String text = key;
            if (st.hasMoreTokens())
                text = st.nextToken("\01").trim() ;
            ret.append(toTeX(text.trim())+ "\\refdefined{" +
    refName(makeRefKey(key))+ "} ") ;
        }
        else {
            ret.append("{"+ block+ "}") ;
        }
    }
```

makeRefKey

```
    static String makeRefKey(String key)
    {
        return key ;
    }
```

toTeX

```
    String toTeX(String str)
    {
        StringBuilder ret = new StringBuilder(str.length());
        long start = System.currentTimeMillis();
        boolean svcoll = false;
        String svblock = null;
        if (textdepth > 0) {
            svcoll = collectBlock ;
            svblock = block ;
        }
        ++textdepth ;
        for(int i = 0 ; i < str.length() ; ++i){
            int c = str.charAt(i);
            if (collectBlock ≡ true && c ≠ '}') {
                block+=str.charAt(i) ;
```

```
                break;
            }
            switch(c) {
                case ' ':
                    if (verbat > 0) {
                        ret.append("\\mbox{ }");
                    }
                    else {
                        ret.append(' ');
                    }
                    break;
                case '\"':
                    ret.append("\\textquotedbl ");
                    break;
                case '_':
                case '%':
                case '$':
                case '#':
                    ret.append('\\');
                    ret.append((char)c);
                    break;
                case '^':
                    ret.append("$\\wedge$");
                    break;
                case '}':
                    if (collectBlock == false) {
                        ret.append("$\\}$");
                        break;
                    }
                    collectBlock = false;
                    processBlock(block, ret);
                    break;
                case '{':
                    if (str.length() > i + 5 && str.substring(i, i +
6).equalsIgnoreCase("{@link")) {
                        block = "@link";
                        collectBlock = true;
                        i+=5;
                    }
                    else {
                        ret.append("$\\{$");
                    }
```

```
                break;
            case '<':
                if (str.length() > i + 4 && str.substring(i, i +
5).equalsIgnoreCase("<pre>")) {
                    ret.append("{\\ttfamily\n");
                    verbat++;
                    i+=4;
                    if (str.charAt(i + 1) == '\r') {
                        i++;
                    }
                    if (str.charAt(i + 1) == '\n') {
                        i++;
                    }
                }
                else if (str.length() > i + 5 && str.substring(i,
i + 6).equalsIgnoreCase("</pre>")) {
                    verbat--;
                    ret.append("}\n");
                    i+=5;
                }
                else if (str.length() > i + 3 && str.substring(i,
i + 4).equalsIgnoreCase("<h1>")) {
                    ret.append("\\headref{1}{\\Huge}{");
                    i+=3;
                }
                else if (str.length() > i + 4 && str.substring(i,
i + 5).equalsIgnoreCase("</h1>")) {
                    ret.append("}\\bl ");
                    i+=4;
                }
                else if (str.length() > i + 3 && str.substring(i,
i + 4).equalsIgnoreCase("<h2>")) {
                    ret.append("\\headref{2}{\\huge}{");
                    i+=3;
                }
                else if (str.length() > i + 4 && str.substring(i,
i + 5).equalsIgnoreCase("</h2>")) {
                    ret.append("}\\bl ");
                    i+=4;
                }
                else if (str.length() > i + 3 && str.substring(i,
i + 4).equalsIgnoreCase("<h3>")) {
                    ret.append("\\headref{3}{\\Large}{");
```

```
                    i+=3;
                }
                else if (str.length()> i+ 4 && str.substring(i,
i+ 5).equalsIgnoreCase("</h3>")) {
                    ret.append("}\\bl ");
                    i+=4;
                }
                else if (str.length()> i+ 3 && str.substring(i,
i+ 4).equalsIgnoreCase("<h4>")) {
                    ret.append("\\headref{4}{\\normalsize}{");
                    i+=3;
                }
                else if (str.length()> i+ 4 && str.substring(i,
i+ 5).equalsIgnoreCase("</h4>")) {
                    ret.append("}\\bl ");
                    i+=4;
                }
                else if (str.length()> i+ 3 && str.substring(i,
i+ 4).equalsIgnoreCase("<h5>")) {
                    ret.append("\\headref{5}{\\small}{");
                    i+=3;
                }
                else if (str.length()> i+ 4 && str.substring(i,
i+ 5).equalsIgnoreCase("</h5>")) {
                    ret.append("}\\bl ");
                    i+=4;
                }
                else if (str.length()> i+ 3 && str.substring(i,
i+ 4).equalsIgnoreCase("<h6>")) {
                    ret.append("\\headref{6}{\\footnotesize}{");
                    i+=3;
                }
                else if (str.length()> i+ 4 && str.substring(i,
i+ 5).equalsIgnoreCase("</h6>")) {
                    ret.append("}\\bl ");
                    i+=4;
                }
                else if (str.length()> i+ 3 && str.substring(i,
i+ 4).equalsIgnoreCase("<h7>")) {
                    ret.append("\\headref{7}{\\scriptsize}{");
                    i+=3;
                }
```

```
                else if (str.length() > i + 4 && str.substring(i,
i + 5).equalsIgnoreCase("</h7>")) {
                    ret.append("}\\bl ");
                    i+=4;
                }
                else if (str.length() > i + 3 && str.substring(i,
i + 4).equalsIgnoreCase("<h8>")) {
                    ret.append("\\headref{8}{\\tiny}{");
                    i+=3;
                }
                else if (str.length() > i + 4 && str.substring(i,
i + 5).equalsIgnoreCase("</h8>")) {
                    ret.append("}\\bl ");
                    i+=4;
                }
                else if (str.length() > i + 5 && str.substring(i,
i + 6).equalsIgnoreCase("<html>")) {
                    i+=5;
                }
                else if (str.length() > i + 6 && str.substring(i,
i + 7).equalsIgnoreCase("</html>")) {
                    if (chapt > 0) {
                        ret.append("}");
                        --chapt;
                    }
                    i+=6;
                }
                else if (str.length() > i + 5 && str.substring(i,
i + 6).equalsIgnoreCase("<head>")) {
                    i+=5;
                }
                else if (str.length() > i + 6 && str.substring(i,
i + 7).equalsIgnoreCase("</head>")) {
                    i+=6;
                }
                else if (str.length() > i + 7 && str.substring(i,
i + 8).equalsIgnoreCase("<center>")) {
                    ret.append("\\makebox[\\hsize]{ ");
                    i+=7;
                }
                else if (str.length() > i + 8 && str.substring(i,
i + 9).equalsIgnoreCase("</center>")) {
                    ret.append("}");
```

```
                    i+=8;
                }
                else if (str.length() > i + 4 && str.substring(i,
i + 5).equalsIgnoreCase("<meta")) {
                    Map<String, String> p = new HashMap<String,
String>();
                    int idx = getTagAttrs(str, p, i + 3);
                    i = idx;
                }
                else if (str.length() > i + 6 && str.substring(i,
i + 7).equalsIgnoreCase("<title>")) {
                    i+=6;
                    ret.append("\\chapter{");
                }
                else if (str.length() > i + 7 && str.substring(i,
i + 8).equalsIgnoreCase("</title>")) {
                    ret.append("}{");
                    i+=7;
                }
                else if (str.length() > i + 4 && str.substring(i,
i + 5).equalsIgnoreCase("<form")) {
                    Map<String, String> p = new HashMap<String,
String>();
                    int idx = getTagAttrs(str, p, i + 3);
                    i = idx;
                }
                else if (str.length() > i + 6 && str.substring(i,
i + 7).equalsIgnoreCase("</form>")) {
                    i+=6;
                }
                else if (str.length() > i + 5 && str.substring(i,
i + 6).equalsIgnoreCase("<input")) {
                    Map<String, String> p = new HashMap<String,
String>();
                    int idx = getTagAttrs(str, p, i + 3);
                    i = idx;
                }
                else if (str.length() > i + 7 && str.substring(i,
i + 8).equalsIgnoreCase("</input>")) {
                    i+=7;
                }
                else if (str.length() > i + 4 && str.substring(i,
i + 5).equalsIgnoreCase("<body")) {
```

```
                    Map<String, String> p = new HashMap<String,
String>();
                    int idx = getTagAttrs(str, p, i + 3);
                    i = idx;
                }
                else if (str.length() > i + 6 && str.substring(i,
i + 7).equalsIgnoreCase("</body>")) {
                    i+=6;
                }
                else if (str.length() > i + 5 && str.substring(i,
i + 6).equalsIgnoreCase("<code>")) {
                    ret.append("{\\ttfamily ");
                    i+=5;
                }
                else if (str.length() > i + 6 && str.substring(i,
i + 7).equalsIgnoreCase("</code>")) {
                    ret.append("}");
                    i+=6;
                }
                else if (str.length() > i + 4 && str.substring(i,
i + 5).equalsIgnoreCase("</br>")) {
                    i+=4;
                }
                else if (str.length() > i + 3 && str.substring(i,
i + 4).equalsIgnoreCase("<br>")) {
                    ret.append("\\mbox{}\\newline\n");
                    i+=3;
                }
                else if (str.length() > i + 4 && str.substring(i,
i + 5).equalsIgnoreCase("<br/>")) {
                    ret.append("\\mbox{}\\newline\n");
                    i+=3;
                }
                else if (str.length() > i + 5 && str.substring(i,
i + 6).equalsIgnoreCase("<br />")) {
                    ret.append("\\mbox{}\\newline\n");
                    i+=3;
                }
                else if (str.length() > i + 3 && str.substring(i,
i + 4).equalsIgnoreCase("</p>")) {
                    ret.append("\\par ");
                    i+=3;
                }
```

```
                else if (str.length() > i + 2 && str.substring(i,
i + 3).equalsIgnoreCase("<p>")) {
                    ret.append("\\par ");
                    i+=2;
                }
                else if (str.length() > i + 2 && str.substring(i,
i + 3).equalsIgnoreCase("<hr")) {
                    Map<String, String> p = new HashMap<String,
String>();
                    int idx = getTagAttrs(str, p, i + 3);
                    String sz = p.get("size");
                    int size = 1;
                    if (sz ≠ null)
                        size = Integer.parseInt(sz);
                    ret.append("\\newline\\rule[2mm]{\\hsize}{" +
(1 · size · 0.5) + "mm}\\newline\n");
                    i = idx;
                }
                else if (str.length() > i + 2 && str.substring(i,
i + 3).equalsIgnoreCase("<b>")) {
                    ret.append("{\\bfseries ");
                    i+=2;
                }
                else if (str.length() > i + 3 && str.substring(i,
i + 4).equalsIgnoreCase("</b>")) {
                    ret.append("}");
                    i+=3;
                }
                else if (str.length() > i + 6 && str.substring(i,
i + 7).equalsIgnoreCase("<strong>")) {
                    ret.append("{\\bfseries ");
                    i+=6;
                }
                else if (str.length() > i + 7 && str.substring(i,
i + 8).equalsIgnoreCase("</strong>")) {
                    ret.append("}");
                    i+=7;
                }
                else if (str.length() > i + 5 && str.substring(i,
i + 6).equalsIgnoreCase("</img>")) {
                    i+=5;
                }
                else if (str.length() > i + 4 && str.substring(i,
i + 4).equalsIgnoreCase("<img")) {
```

```
                    Map<String, String> p = new HashMap<String,
String>();
                    int idx = getTagAttrs(str, p, i + 4);
                    refimg = p.get("src");
                    ret.append("(see image at " + toTeX(refimg) +
")");
                    i = idx;
                }
                else if (str.length() > i + 3 && str.substring(i,
i + 4).equalsIgnoreCase("</a>")) {
                    if (refurl ≠ null) {
                        ret.append("} ");
                        if (refurl.charAt(0) ≡ '#') {
                            ret.append("\\refdefined{" +
refName(makeRefKey(refurl.substring(1))) + "}");
                        }
                        else {
                            ret.append("(at " + toTeX(refurl) +
")");
                        }
                    }
                    i+=3;
                }
                else if (str.length() > i + 2 && str.substring(i,
i + 2).equalsIgnoreCase("<a")) {
                    Map<String, String> p = new HashMap<String,
String>();
                    int idx = getTagAttrs(str, p, i + 3);
                    refurl = p.get("href");
                    String refname = p.get("href");
                    i = idx;
                    if (refurl ≠ null)
                        ret.append("{\\bf ");
                    else if (refname ≠ null)
                        ret.append("\\label{" +
refName(makeRefKey(refname)) + "}");
                }
                else if (str.length() > i + 3 && str.substring(i,
i + 3).equalsIgnoreCase("<ol")) {
                    Map<String, String> p = new HashMap<String,
String>();
                    int idx = getTagAttrs(str, p, i + 3);
                    i = idx;
```

```
                    ret.append("\\begin{enumerate}\n");
                }
                else if (str.length() > i + 2 && str.substring(i,
i + 3).equalsIgnoreCase("<dl")) {
                    Map<String, String> p = new HashMap<String,
String>();
                    int idx = getTagAttrs(str, p, i + 3);
                    i = idx;
                    ret.append("\\begin{itemize}\n");
                }
                else if (str.length() > i + 3 && str.substring(i,
i + 4).equalsIgnoreCase("<li>")) {
                    ret.append("\n");
                    ret.append("\\item ");
                    i+=3;
                }
                else if (str.length() > i + 4 && str.substring(i,
i + 5).equalsIgnoreCase("</li>")) {
                    i+=4;
                }
                else if (str.length() > i + 3 && str.substring(i,
i + 4).equalsIgnoreCase("<dt>")) {
                    ret.append("\n");
                    ret.append("\\item[{");
                    i+=3;
                }
                else if (str.length() > i + 3 && str.substring(i,
i + 4).equalsIgnoreCase("<dd>")) {
                    ret.append("}]");
                    i+=3;
                }
                else if (str.length() > i + 4 && str.substring(i,
i + 5).equalsIgnoreCase("</dd>")) {
                    i+=4;
                }
                else if (str.length() > i + 4 && str.substring(i,
i + 5).equalsIgnoreCase("</dt>")) {
                    i+=4;
                }
                else if (str.length() > i + 4 && str.substring(i,
i + 5).equalsIgnoreCase("</dl>")) {
                    ret.append("\n\\end{itemize}");
                    i+=4;
```

```
                }
                else if (str.length() > i + 4 && str.substring(i,
i + 5).equalsIgnoreCase("</ol>")) {
                    ret.append("\n\\end{enumerate}");
                    i+=4;
                }
                else if (str.length() > i + 3 && str.substring(i,
i + 3).equalsIgnoreCase("<ul")) {
                    Map<String, String> p = new HashMap<String,
String>();
                    int idx = getTagAttrs(str, p, i + 3);
                    i = idx;
                    ret.append("\\begin{itemize}");
                }
                else if (str.length() > i + 4 && str.substring(i,
i + 5).equalsIgnoreCase("</ul>")) {
                    ret.append("\n\\end{itemize}\n");
                    i+=4;
                }
                else if (str.length() > i + 2 && str.substring(i,
i + 3).equalsIgnoreCase("<i>")) {
                    ret.append("{\\it ");
                    i+=2;
                }
                else if (str.length() > i + 3 && str.substring(i,
i + 4).equalsIgnoreCase("</i>")) {
                    ret.append("}");
                    i+=3;
                }
                else if (str.length() > i + 3 && str.substring(i,
i + 4).equalsIgnoreCase("<em>")) {
                    ret.append("{\\em ");
                    i+=3;
                }
                else if (str.length() > i + 4 && str.substring(i,
i + 5).equalsIgnoreCase("</em>")) {
                    ret.append("}");
                    i+=4;
                }
                else if (str.length() > i + 7 && str.substring(i,
i + 8).equalsIgnoreCase("</table>")) {
                    tblinfo.endTable(ret);
                    tblinfo = tblstk.pop();
```

```
                    i+=7;
                }
                else if (str.length() > i + 4 && str.substring(i,
i + 5).equalsIgnoreCase("</th>")) {
                    tblinfo.endCol(ret);
                    i+=4;
                }
                else if (str.length() > i + 4 && str.substring(i,
i + 5).equalsIgnoreCase("</td>")) {
                    tblinfo.endCol(ret);
                    i+=4;
                }
                else if (str.length() > i + 4 && str.substring(i,
i + 5).equalsIgnoreCase("</tr>")) {
                    tblinfo.endRow(ret);
                    i+=4;
                }
                else if (str.length() > i + 5 && str.substring(i,
i + 6).equalsIgnoreCase("<table")) {
                    Map<String, String> p = new HashMap<String,
String>();
                    int idx = getTagAttrs(str, p, i + 6);
                    i = idx;
                    stackTable(p, ret, str, i);
                }
                else if (str.length() > i + 2 && str.substring(i,
i + 3).equalsIgnoreCase("<tr")) {
                    Map<String, String> p = new HashMap<String,
String>();
                    int idx = getTagAttrs(str, p, i + 3);
                    i = idx;
                    tblinfo.startRow(ret, p);
                }
                else if (str.length() > i + 2 && str.substring(i,
i + 3).equalsIgnoreCase("<td")) {
                    Map<String, String> p = new HashMap<String,
String>();
                    int idx = getTagAttrs(str, p, i + 3);
                    i = idx;
                    tblinfo.startCol(ret, p);
                }
                else if (str.length() > i + 2 && str.substring(i,
i + 3).equalsIgnoreCase("<th")) {
```

```
                    Map<String, String> p = new HashMap<String,
String>();
                    int idx = getTagAttrs(str, p, i + 3);
                    i = idx;
                    tblinfo.startHeadCol(ret, p);
                }
                else if (str.length() > i + 4 && str.substring(i,
i + 5).equalsIgnoreCase("<font")) {
                    Map<String, String> p = new HashMap<String,
String>();
                    int idx = getTagAttrs(str, p, i + 5);
                    i = idx;
                    String sz = p.get("size");
                    String col = p.get("color");
                    ret.append("{");
                    if (col ≠ null) {
                        String colName = colors.get(col);
                        if (colName ≡ null) {
                            colName = "color" + colorIdx;
                            colorIdx++;
                            Color cc = new
Color((int)Long.parseLong(col, 16));
                            ret.append("\\definecolor{" + colName
+ "}[rgb]{" + (cc.getRed() / 255.0) + "," + (cc.getBlue() / 255.0)
+ "," + (cc.getGreen() / 255.0) + "}");
                            colors.put(col, colName);
                        }
                        ret.append("\\color{" + colName + "}");
                    }
                }
                else if (str.length() > i + 6 && str.substring(i,
i + 7).equalsIgnoreCase("</font>")) {
                    ret.append("}");
                    i+=6;
                }
                else {
                    ret.append("\\textless ");
                }
                break;
            case '\r':
            case '\n':
                if (tblstk.size() > 0) {
                    if (verbat > 0) {
```

```
                        ret.append("\\newline\n");
                    }
                    else
                        ret.append(" ");
                }
                else {
                    if ((i + 1) < str.length() && str.charAt(i +
1) == 10) {
                        ret.append("\\bl ");
                        ++i;
                    }
                    else {
                        if (verbat > 0)
                            ret.append("\\mbox{}\\newline\n");
                        else
                            ret.append((char)c);
                    }
                }
                break;
            case '/':
                ret.append("$/$");
                break;
            case '&':
                if (str.length() > i + 4 && str.substring(i, i +
2).equals("&#")) {
                    String it = str.substring(i + 2);
                    int stp = it.indexOf(';');
                    if (stp > 0) {
                        String v = it.substring(0, stp);
                        int ch = -1;
                        try {
                            ch = Integer.parseInt(v);
                        }
                        catch(NumberFormatException ex){
                            ch = -1;
                        }
                        if (ch >= 0 && ch < 128) {
                            ret.append("\\verb" + ((char)(ch +
1)) + ((char)ch) + ((char)(ch + 1)));
                        }
                        else {
                            ret.append("\\&\\#" + v);
```

```
                        }
                        i+=v.length() + 2;
                    }
                    else {
                        ret.append("\\&\\#");
                        i++;
                    }
                }
                else if (str.length() > i + 4 && str.substring(i,
i + 5).equalsIgnoreCase("&amp;")) {
                    ret.append("\\&");
                    i+=4;
                }
                else if (str.length() > i + 5 && str.substring(i,
i + 6).equalsIgnoreCase(" ")) {
                    ret.append("\\phantom{ }");
                    i+=5;
                }
                else if (str.length() > i + 3 && str.substring(i,
i + 4).equalsIgnoreCase("&lt;")) {
                    ret.append("\\textless ");
                    i+=3;
                }
                else if (str.length() > i + 3 && str.substring(i,
i + 4).equalsIgnoreCase("&gt;")) {
                    ret.append("\\textgreater ");
                    i+=3;
                }
                else
                    ret.append("\\&");
                break;
            case '>':
                ret.append("\\textgreater ");
                break;
            case '\\':
                ret.append("\\bslash ");
                break;
            default:
                ret.append((char)c);
                break;
        }
    }
```

```
        if (textdepth > 0) {
            collectBlock = svcoll;
            block = svblock;
        }
        --textdepth;
        long to = System.currentTimeMillis() - start;
        if (to > 1000) {
            System.out.print("(text @" + to + " msecs)");
            System.out.flush();
        }
        return ret.toString();
    }
```

getTagAttrs This method parses HTML tags to extract the tag attributes and place them into a Map<String,String>object.

Parameter

str the string that is the whole HTML start tag (at least)

i the offset in the string where the tag starts

Rückgabewert the offset in the String after the end of the tag (not element).

```
    static int getTagAttrs(String str,
                           Map<String, String> p,
                           int i)
    {
        char[] b = str.toCharArray();
        String name = "";
        String value = "";
        int state = 0;
        while(i < b.length){
            switch(b[i]) {
                case ' ':
                    if (state == 2) {
                        p.put(name.toLowerCase(), value);
                        state = 1;
                        name = "";
                        value = "";
                    }
                    else if (state == 3) {
                        value+=" ";
                    }
                    break;
```

```
                case '=':
                    if (state == 1) {
                        state = 2;
                        value = "";
                    }
                    else if (state > 1) {
                        value+='=';
                    }
                    break;
                case '\"':
                    if (state == 2) {
                        state = 3;
                    }
                    else if (state == 3) {
                        state = 1;
                        p.put(name.toLowerCase(), value);
                        name = "";
                        value = "";
                    }
                    break;
                case '>':
                    if (state == 1) {
                        p.put(name.toLowerCase(), "");
                    }
                    else if (state == 2) {
                        p.put(name.toLowerCase(), value);
                    }
                    return i;
                default:
                    if (state == 0)
                        state = 1;
                    if (state == 1) {
                        name = name + b[i];
                    }
                    else {
                        value = value + b[i];
                    }
            }
            ++i;
        }
        return i;
    }
```

1.19 Klasse de.dclj.paul.ltxdoclet.LaTeXWriter

1.19.1 Übersicht

Einige generelle Methoden zum Schreiben von LaTeX-Dokumenten. Unterklassen können diese dann verwenden.

1.19.2 Inhaltsverzeichnis

1.19.3 Variablen

ltxsymb Die Liste der Ersetzungen.

```
static final String[][] ltxsymb = { { "\\", "\\textbackslash " },
{ "\"", "{\\textquotedbl}" }, { "_", "{\\textunderscore}" }, {
"¡", "!`" }, { "£", "{\\pounds}" }, { "§", "{\\S}" }, { "©",
"{\\copyright}" }, { "±", "{\\pm}" }, { "¶", "{\\P}" }, { "·",
"{\\cdot}" }, { "¿", "?`" }, { "À", "\\`{A}" }, { "Á", "\\'{A}"
}, { "Â", "\\^{A}" }, { "Ã", "\\~{A}" }, { "Ä", "\\"{A}" }, {
"Å", "{\\AA}" }, { "Æ", "{\\AE}" }, { "Ç", "\\c{C}" }, { "È",
"\\`{E}" }, { "É", "\\'{E}" }, { "Ê", "\\^{E}" }, { "Ë",
"\\"{E}" }, { "Ì", "\\`{I}" }, { "Í", "\\'{I}" }, { "Î",
"\\^{I}" }, { "Ï", "\\"{I}" }, { "Ñ", "\\~{N}" }, { "Ò",
"\\`{O}" }, { "Ó", "\\'{O}" }, { "Ô", "\\^{O}" }, { "Õ",
"\\~{O}" }, { "Ö", "\\"{O}" }, { "×", "{\\times}" }, { "Ø",
"{\\O}" }, { "Ù", "\\`{U}" }, { "Ú", "\\'{U}" }, { "Û", "\\^{U}"
}, { "Ü", "\\"{U}" }, { "Ý", "\\'{Y}" }, { "ß", "{\\ss}" }, {
"à", "\\`{a}" }, { "á", "\\'{a}" }, { "â", "\\^{a}" }, { "ã",
"\\~{a}" }, { "ä", "\\"{a}" }, { "å", "{\\aa}" }, { "æ",
"{\\ae}" }, { "ç", "\\c{c}" }, { "è", "\\`{e}" }, { "é",
"\\'{e}" }, { "ê", "\\^{e}" }, { "ë", "\\"{e}" }, { "ì",
"\\`{\\i}" }, { "í", "\\'{\\i}" }, { "î", "\\^{\\i}" }, { "ï",
"\\"{\\i}" }, { "ñ", "\\~{n}" }, { "ò", "\\`{o}" }, { "ó",
"\\'{o}" }, { "ô", "\\^{o}" }, { "õ", "\\~{o}" }, { "ö",
"\\"{o}" }, { "÷", "{\\div}" }, { "ø", "{\\o}" }, { "ù",
"\\`{u}" }, { "ú", "\\'{u}" }, { "û", "\\^{u}" }, { "ü",
"\\"{u}" }, { "ý", "\\'{y}" }, { "ÿ", "\\"{y}" } };
```

nonNormalTagKinds

```
private final Set⟨String⟩ nonNormalTagKinds = new HashSet⟨String⟩
(Arrays.asList("@throws", "@see", "@param"));
```

configuration

```
static LtxDocletConfiguration configuration;
```

1.19.4 Konstruktoren

LaTeXWriter Erstellt einen neuen LaTeXWriter, der in die angegebene Datei schreibt.

Exceptions

IOException falls die Datei nicht geöffnet bzw. beschrieben werden kann.

```
public LaTeXWriter(File filename)
    throws IOException
{
    super(new OutputStreamWriter(new FileOutputStream(filename),
configuration.docencoding));
    configuration.root.printNotice("Schreibe in "+ filename+
"...");
    println("   % /-----------------------------------------
\\");
    println("  % | API-Dokumentation für einige Java-Packages |");
    println("   % |    (genaueres siehe doku-main.tex).       |");
    println("  % | LaTeX-Ausgabe erstellt von \'ltxdoclet\'.   |");
    println("   % | Dieses Programm stammt von Paul Ebermann.  |");
    println("   % \\-----------------------------------------
/");
    println();
}
```

1.19.5 Methoden

replace

```
private String replace(String org,
                       String sub1,
                       String sub2)
{
    StringBuffer buf = new StringBuffer(org);
    int slen = sub1.length();
    for(int i = 0; i< buf.length(); i++){
        if (buf.substring(i, i+ slen).equals(sub1)) {
            buf.replace(i, i+ slen, sub2);
        }
    }
    return buf.toString();
}
```

asLaTeXString wandelt einen Unicode-String in die entsprechenden LaTeX-Symbole um.

```
public String asLaTeXString(String s)
{
    return new HtmlKonverter().toTeX(s);
}
```

asLaTeXString Wandelt den Namen eines Dokumentations-Elementes in einen LaTeX-String um.

```
public String asLaTeXString(Doc d)
{
    return asLaTeXString(d.toString());
}
```

ltxwrite gibt den angegebenen Text, umgewandelt in LaTeX-Befehle, aus.

Parameter

text der umzukodierende Text.

```
public void ltxwrite(String text)
{
    print(asLaTeXString(text));
}
```

chapter Beginnt ein neues Kapitel.

Parameter

name Name bzw. Überschrift des Kapitels

num Numerierung erwünscht?

```
public void chapter(String name,
                    boolean num)
{
    println("\\chapter" + (num ? "" : "*") + "{" +
asLaTeXString(name) + "}");
}
```

chapter Beginnt ein neues Kapitel mit Numerierung.

Parameter

name Name bzw. Überschrift des Kapitels

```
public void chapter(String name)
{
    chapter(name, true);
}
```

chapter Eine Kapitelüberschrift mit Link-Ziel.

```
public void chapter(String prefix,
                    Doc doc,
                    String shortName)
{
    String ref = configuration.toRefLabel(doc);
    println("\\chapter[" + asLaTeXString(shortName) + "]{" +
prefix + "\\ltdHypertarget{" + ref + "}{" +
asLaTeXString(doc.toString()) + "}}\\label{" + ref + "}");
}
```

chapter

```
public void chapter(String prefix,
                    Doc doc)
{
    chapter(prefix, doc, prefix + doc);
}
```

section Beginnt einen neuen Abschnitt.

```
public void section(String name)
{
    println("\\section{" + asLaTeXString(name) + "}");
}
```

section

```
public void section(String prefix,
                    Doc doc,
                    String shortName)
{
    String ref = configuration.toRefLabel(doc);
    println("\\section[" + asLaTeXString(shortName) + "]{" +
prefix + "\\ltdHypertarget{" + ref + "}{" +
asLaTeXString(doc.toString()) + "}}\\label{" + ref + "}");
}
```

section Beginnt einen neuen Abschnitt, der im Inhaltsverzeichnis einen alternativen Namen hat.

```
public void section(String name,
                    String shortName)
{
    println("\\section[" + asLaTeXString(shortName) + "]{" +
asLaTeXString(name) + "}");
}
```

subsection Beginnt einen neuen Unterabschnitt.

```
public void subsection(String name)
{
    println("\\subsection{" + asLaTeXString(name) + "}");
}
```

subsubsection Beginnt einen neuen Unterunterabschnitt.

```
public void subsubsection(String name)
{
    println("\\subsubsection{" + asLaTeXString(name) + "}");
}
```

italic gibt den angegebenen Text kursiv aus.

```
public void italic(String name)
{
    println("\\textit{" + asLaTeXString(name) + "}");
}
```

bold gibt den angegebenen Text fett aus.

```
public void bold(String text)
{
    println("\\textbf{" + asLaTeXString(text) + "}");
}
```

typeRef Schreibt einen Typnamen (mit Verlinkungen) in den angegebenen StringBuilder (am Ende).

```
public void typeRef(Type t,
                    StringBuilder app)
{
    if (t.isPrimitive()) {
        app.append(t.toString());
        return;
    }
    String dimension = t.dimension();
    TypeVariable tv = t.asTypeVariable();
    if (tv ≠ null) {
        if (tv.owner().isClass()) {
            app.append(createLink(tv.typeName(), tv.owner()));
        }
        else {
            app.append(tv.typeName());
        }
        app.append(dimension);
        return;
    }
    WildcardType wt = t.asWildcardType();
    if (wt ≠ null) {
        app.append(wt);
        app.append(dimension);
        return;
    }
    ParameterizedType pt = t.asParameterizedType();
    if (pt ≠ null) {
```

```
            if (pt.containingType() ≠ null) {
                app.append(typeRef(pt.containingType()));
                app.append(".");
            }
            app.append(typeRef(pt.asClassDoc()));
            app.append("<");
            for (Type ta : pt.typeArguments()) {
                app.append(typeRef(ta));
                app.append(", ");
            }
            if (pt.typeArguments().length > 0) {
                app.delete(app.length() - 2, app.length());
            }
            app.append(">");
            app.append(dimension);
            return;
        }
        ClassDoc cd = t.asClassDoc();
        if (cd ≠ null) {
            if (cd.containingClass() ≠ null) {
                typeRef(cd.containingClass(), app);
                app.append(".");
            }
            app.append(createLink(cd));
            app.append(dimension);
            return;
        }
        app.append(t);
    }
```

typeRef Erstellt eine verlinkte Version des Typnamens von t.

```
    public String typeRef(Type t)
    {
        StringBuilder b = new StringBuilder();
        typeRef(t, b);
        return b.toString();
    }
```

referenceTo Ermittelt eine Referenz zu dem angegebenen Programmelement.

```
    public String referenceTo(Doc doc)
    {
        return "\\pageref{" + configuration.toRefLabel(doc) + "}";
    }
```

referenceTarget Erstellt ein LaTeX-Label für das angegebene Programmelement.

```
public String referenceTarget(Doc doc)
{
    return referenceTarget(doc, doc.toString());
}
```

referenceTarget

```
public String referenceTarget(Doc doc,
                              String label)
{
    String ref = configuration.toRefLabel(doc);
    return "\\ltdHypertarget{" + ref + "}{" + label + "}" +
"\\label{" + ref + "}";
}
```

writeDescription Schreibt die Beschreibung dieses zu dokumentierenden Elementes.

Zuerst werden die Inline-Tags des Elementes ausgegeben, danach die »normalen« Tags.

```
public void writeDescription(Doc d)
{
    writeInlineTags(d.inlineTags());
    Tag[] tags = d.tags();
    if (tags.length > 0) {
        println("\\begin{description}");
        if (d instanceof ExecutableMemberDoc) {
            ExecutableMemberDoc emd = (ExecutableMemberDoc)d;
            writeTypeParams(emd.typeParamTags());
            writeParams(emd.paramTags());
            writeThrows(emd.throwsTags());
        }
        writeNormalTags(normalTags(tags));
        writeSeeTags(seeTags(tags));
        println("\\end{description}");
    }
}
```

writeTypeParams

```
public void writeTypeParams(ParamTag[] tags)
{
    if (tags.length > 0) {
        println("\\item[Typparameter] ~");
```

```
            println("\\begin{description}");
            for (ParamTag pt : tags) {
                writeParamTag(pt);
            }
            println("\\end{description}");
        }
    }
```

writeParams

```
    public void writeParams(ParamTag[] tags)
    {
        if (tags.length > 0) {
            println("\\item[Parameter] ~");
            println("\\begin{description}");
            for (ParamTag pt : tags) {
                writeParamTag(pt);
            }
            println("\\end{description}");
        }
    }
```

writeThrows

```
    public void writeThrows(ThrowsTag[] tags)
    {
        if (tags.length > 0) {
            println("\\item[Exceptions] ~");
            println("\\begin{description}");
            for (ThrowsTag tt : tags) {
                writeThrowsTag(tt);
            }
            println("\\end{description}");
        }
    }
```

writeNormalTags

```
    public void writeNormalTags(Tag[] tags)
    {
        if (tags.length > 0) {
            for (Tag tag : tags) {
                println("\\item[" + tagName(tag) + "] ");
                writeInlineTags(tag.inlineTags());
            }
        }
    }
```

writeSeeTags

```
public void writeSeeTags(SeeTag[] tags)
{
    if (tags.length > 0) {
        println("\\item[Siehe auch] ~");
        print("\\noprint");
        for (SeeTag t : tags) {
            print(", ");
            writeSeeTag(t);
        }
    }
}
```

normalTags Filtert alle Tags heraus, die speziell behandelt werden, und gibt die übrigen zurück. Die herausgefilterten sind:

- @throws/@ecxeption
- @param
- @see

```
public Tag[] normalTags(Tag[] tags)
{
    Tag[] result = new Tag[tags.length];
    int j = 0;
    for (Tag t : tags) {
        if (!nonNormalTagKinds.contains(t.kind())) {
            result[j] = t;
            j++;
        }
    }
    return j == tags.length ? result : Arrays.copyOfRange(result,
0, j);
}
```

seeTags

```
private SeeTag[] seeTags(Tag[] tags)
{
    SeeTag[] result = new SeeTag[tags.length];
    int j = 0;
    for (Tag t : tags) {
        if ("@see".equals(t.kind())) {
            result[j] = (SeeTag)t;
            j++;
```

```
            }
        }
        return j == tags.length ? result : Arrays.copyOfRange(result,
    0, j);
    }
```

writeTag schreibt ein generisches Block-Tag.

```
    public void writeTag(Tag tag)
    {
        println("\\item[" + tagName(tag) + "]");
        writeInlineTags(tag.inlineTags());
    }
```

writeThrowsTag

```
    public void writeThrowsTag(ThrowsTag tag)
    {
        println("\\item[" + typeRef(tag.exceptionType()) + "]");
        writeInlineTags(tag.inlineTags());
    }
```

writeParamTag

```
    public void writeParamTag(ParamTag tag)
    {
        println("\\item[" + tag.parameterName() + "]");
        writeInlineTags(tag.inlineTags());
    }
```

writeSeeTag

```
    public void writeSeeTag(SeeTag tag)
    {
        String className = tag.referencedClassName();
        if (className == null) {
            print(asLaTeXString(tag.toString()));
        }
        else {
            writeLinkTag(tag);
        }
    }
```

tagName Ermittelt den Namen eines Tags in einer Tag-Liste.

```
    public String tagName(Tag t)
    {
        String kind = t.kind();
```

```
        if (kind.equals("@throws"))
            return "throws " + ((ThrowsTag)t).exceptionType();
        if (kind.equals("@see"))
            return "Siehe auch";
        if (kind.equals("@return"))
            return "Rückgabewert";
        if (kind.equals("@param")) {
            ParamTag pt = (ParamTag)t;
            if (pt.isTypeParameter())
                return "Typeparameter " + pt.parameterName();
            return "Parameter " + pt.parameterName();
        }
        return kind;
    }
```

createLink

```
    public String createLink(String labelText,
                             String target)
    {
        return "\\hyperlink{" + target + "}{" + labelText + "}";
    }
```

createExternalLink

```
    public String createExternalLink(String label,
                                     Doc target)
    {
        return null;
    }
```

createLink Erstellt einen Link.

```
    public String createLink(String label,
                             Doc target)
    {
        if (label.equals("")) {
            label = target.name();
        }
        return configuration.linker.createLink(label, target);
    }
```

createLink Erstellt einen Link.

```
    public String createLink(Doc target)
    {
        return createLink("", target);
    }
```

writeLinkTag

```
void writeLinkTag(SeeTag tag,
                  Doc doc)
{
    if (tag.name().equals("@linkplain")) {
        print(createLink(tag.label(), doc));
    }
    else {
        print("\\texttt{"+ createLink(tag.label(), doc)+ "}");
    }
}
```

writeLinkTag Schreibt ein Link-Tag (bzw. den Inhalt eines `SeeTag`s) hinaus.

Hier noch ein Testlink und `ClassWriter`.

Siehe auch `PrettyPrinter`

```
public void writeLinkTag(SeeTag st)
{
    print("\\noprint{"+ st+ "}");
    MemberDoc md = st.referencedMember();
    if (md ≠ null) {
        writeLinkTag(st, md);
        return;
    }
    ClassDoc cd = st.referencedClass();
    if (cd ≠ null) {
        writeLinkTag(st, cd);
        return;
    }
    PackageDoc pd = st.referencedPackage();
    if (pd ≠ null) {
        writeLinkTag(st, pd);
        return;
    }
    print("{«Link:"+ st+ "|"+ st.referencedClassName()+ "|"+
st.label()+ "»}");
}
```

writeInlineTag

```
public void writeInlineTag(Tag t,
                           HtmlKonverter konverter)
{
```

```
        if (t.kind().equalsIgnoreCase("Text"))
            print(konverter.toTeX(t.text()));
        else if (t.name().equals("@LaTeX"))
            print(t.text());
        else if (t.name().equalsIgnoreCase("@code")) {
            print("\\verb!" + t.text() + "!");
        }
        else if (t instanceof SeeTag) {
            SeeTag st = (SeeTag)t;
            writeLinkTag(st);
        }
        else {
            configuration.root.printNotice("Unbekanntes Inline-Tag: "
    + t);
            print("<<" + t + ">>");
        }
    }
```

writeInlineTags Schreibt mehrere Tags nacheinander. Dies ist gedacht für Inline- und Text-Tags. Diese werden z.B. von Doc.inlineTags() und Doc.firstSentenceTags() zurückgegeben.

```
    public void writeInlineTags(Tag[] tags)
    {
        HtmlKonverter konverter = new HtmlKonverter();
        for (Tag tag : tags) {
            writeInlineTag(tag, konverter);
        }
        println();
    }
```

newParagraph

```
    public void newParagraph()
    {
        println();
        println();
    }
```

newLine

```
    public void newLine()
    {
        println("\\\\");
    }
```

1.20 Klasse de.dclj.paul.ltxdoclet.ClassWriter

1.20.1 Übersicht

Ein Writer zum Schreiben einer Klasse (inklusive der Methoden).

1.20.2 Inhaltsverzeichnis

1.20.3 Variablen

doc

```
private ClassDoc doc;
```

1.20.4 Konstruktoren

ClassWriter

```
public ClassWriter(ClassDoc cd)
    throws IOException
{
    super(new
File(configuration.toOutputFileName(cd.containingPackage()),
cd.name() + ".tex"));
    this.doc = cd;
}
```

1.20.5 Methoden

writeDoc

```
public void writeDoc()
{
    try {
        configuration.root.printNotice("ltxdoclet: Klassen-
Doku für \"" + doc +
"\" wird erstellt ...");
        println("   % Api-Dokumentation für Klasse " + doc +
" (noch nicht fertig). ");
        String refTarget = referenceTarget(doc);
        if (doc.isInterface()) {
            section("Interface ", doc, doc.name());
        }
        else if (doc.isOrdinaryClass()) {
            section("Klasse ", doc, doc.name());
        }
        else if (doc.isException()) {
            section("Exception ", doc, doc.name());
```

```
            }
            else if (doc.isError()) {
                section("Error ", doc, doc.name());
            }
            else if (doc.isEnum()) {
                section("Enum ", doc, doc.name());
            }
            Type[] interfaces = doc.interfaceTypes();
            Type superClass = doc.superclassType();
            subsection("Übersicht");
            writeDescription(doc);
            ConstructorDoc[] konstr = doc.constructors();
            MethodDoc[] meth = doc.methods();
            FieldDoc[] fields = doc.fields();
            FieldDoc[] consts = doc.enumConstants();
            subsection("Inhaltsverzeichnis");
            writeMemberList(consts, "Enum-Konstanten");
            writeMemberList(fields, "Variablen");
            writeMemberList(konstr, "Konstruktoren");
            writeMemberList(meth, "Methoden");
        }
        finally {
            configuration.root.printNotice("ltxdoclet: ... Klassen-
    Doku für \"" + doc +
    "\" beendet.");
            close();
        }
    }
```

writeMemberList

```
    public ⟨X extends MemberDoc⟩ void writeMemberList(X[] liste,
                                                      String titel)
    {
        if (liste.length > 0) {
            subsection(titel);
            println("\\begin{description}");
            for («[X:X]» d : liste) {
                writeMemberDoc(d);
            }
            println("\\end{description}");
        }
    }
```

writeMemberDoc

```
    public ⟨X extends MemberDoc⟩ void writeMemberDoc(«[X:X]» d)
    {
        println("\\item[{" + referenceTarget(d,
    asLaTeXString(d.name())) + "}]");
        print("~ ");
        writeDescription(d);
        if (configuration.includeSource) {
            try {
                configuration.pp.printSource(d, this);
            }
            finally {
                newParagraph();
            }
        }
    }
```

writeDeclaration

```
    public void writeDeclaration(FieldDoc d)
    {
        print("\\texttt{");
        print(d.modifiers());
        print(" ");
        Type t = d.type();
        print(typeRef(t));
        print(" ");
        print(d.name());
        print(";");
        print("}");
    }
```