Javadoc

SS 2019

Dokumentation

- gut dokumentierter Quelltext zum besseren Verstehen eines Programms
  - für andere Entwickler
  - Für sich selbst
  - Für die Projektgruppe

- Dokumentation für Java-Code
  - Klassen
  - Interfaces
  - Methoden

Javadoc

- nützliches Tool in der Entwicklung
- Überblick über Codekommentare
- Generiert HTML-Seiten für alle Klassen-/Interface- und Methodenkommentare
  - In /** */ eingeschlossene Kommentare
  - `<code> spezielle Schriftart </code>`
    - um Klassennamen, Interfacenamen, Methodennamen hervorzuheben
  - In Klassen- und Methodenkommentaren kann man auch HTML benutzen
    - Das wird in das fertige HTML-Dokument integriert.
  - Mit // beginnende Zeilenkommentare und durch /* und */ eingeschlossene Blockkommentare werden nicht ausgewertet.

Tags

- Spezielle Bezeichner mit @ davor:
  - @author - Der Autor der Klasse oder des Interfaces.
    - Bei mehreren Autoren kann man dieses Tag mehrfach verwenden.
  - @version - die Version der Bibliothek angeben, seit der die Klasse, das Interface oder die Methode mit dabei ist.
  - @param - Parameter von Methoden lassen sich damit beschreiben.
    - Nach @param kommt der Name des Parameters und danach die Beschreibung.
  - @return – Bezeichnet den Rückgabewert einer Methode

Tags

- Spezielle Bezeichner mit @ davor:
  - @throws (oder @exception) - Darüber kann dokumentiert werden, welche Exceptions eine Methode oder ein Konstruktor werfen.
    - Nach @throws kommt der Typ der Exception, danach die Beschreibung.
  - @see - für eine Methode angeben, an welchen anderen Stellen im Quelltext noch weitere Informationen über diese Methode zu finden sind.
    - Typischerweise wird das verwendet, um von der Implementierung eines Interfaces auf die schon vorhandene Dokumentation im Interface zu verlinken.

Javadoc und Tags

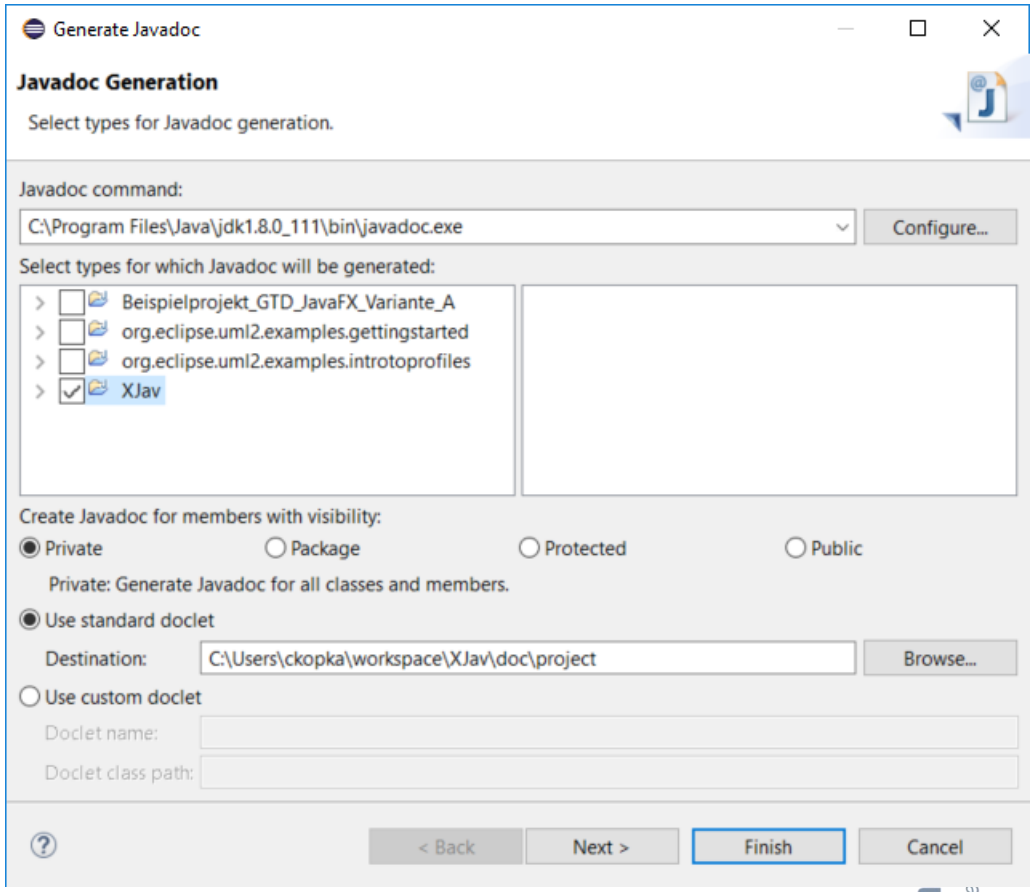
- Es gibt weitere bereits definierte Tags
- Außerdem können mit Javadoc eigene Custom Tags definiert werden
  - Z.B. Custom Tags wie
    - @precondition und @postcondition
    - mit dem Parameter "-tag" wird definiert,
      - wo die Tags erscheinen können
        - Option "cm" steht für "Constructor, Method„
      - durch was für einen String sie ersetzt werden sollen
- Generieren der Javadoc mit Custom Tags aus Netbeans und Eclipse möglich

Generieren von Javadoc

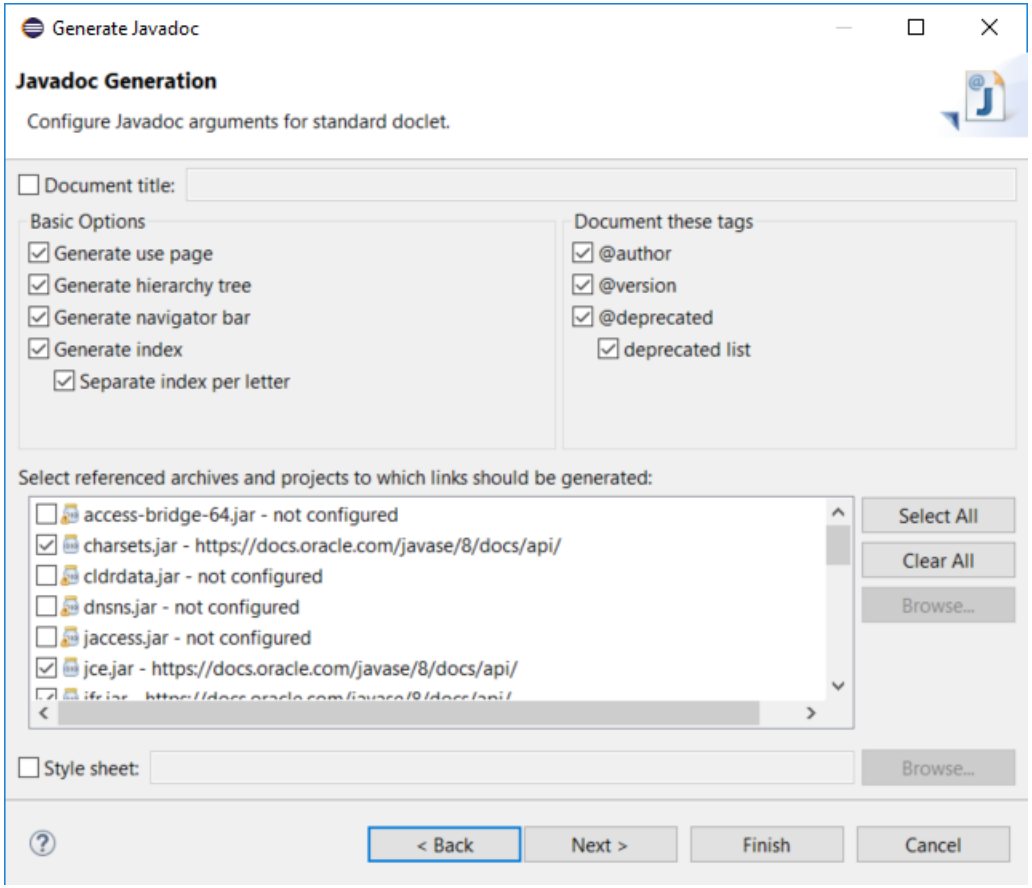
- direkt
  - javadoc –d ..\doc –subpackages <packagename>
  - -d gibt den Zielordner an
    - Aufruf des Befehls im src -Ordner
    - den Ordner doc eine Ebene höher, weil man die Dokumentation nicht im src-Ordner erstellen will
  - –subpackages gibt an, dass die Dokumentation auch für alle Unterpackages des angegebenen Packages mit generiert wird
- Über IDE wie Eclipse

Eclipse - Javadoc

- Javadoc einstellen:
  - Menü "Project"
    - "Generate Javadoc...„
  - Javadoc command
  - Gewünschte Sichtbarkeit
  - Zielordner:
    - Wohin soll generiert werden

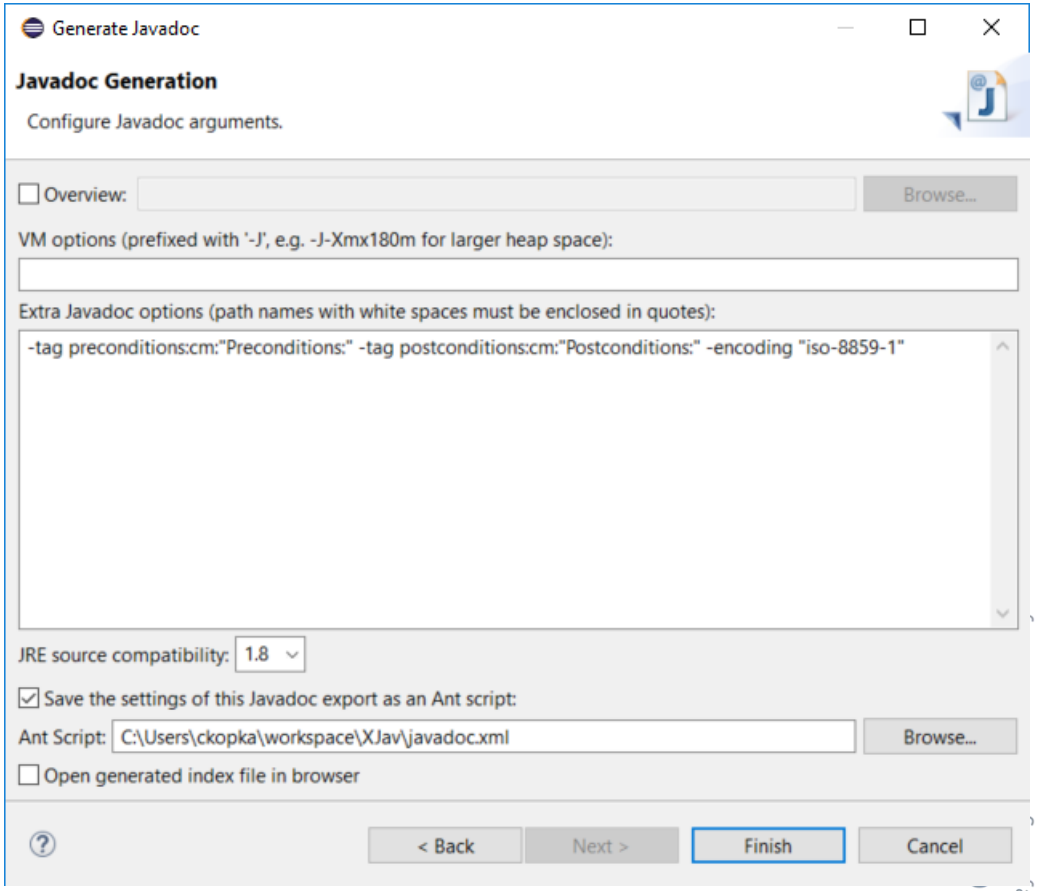


Eclipse - Javadoc



Eclipse - Javadoc

- Custom Tags
- Encoding
- ...



Beispiel

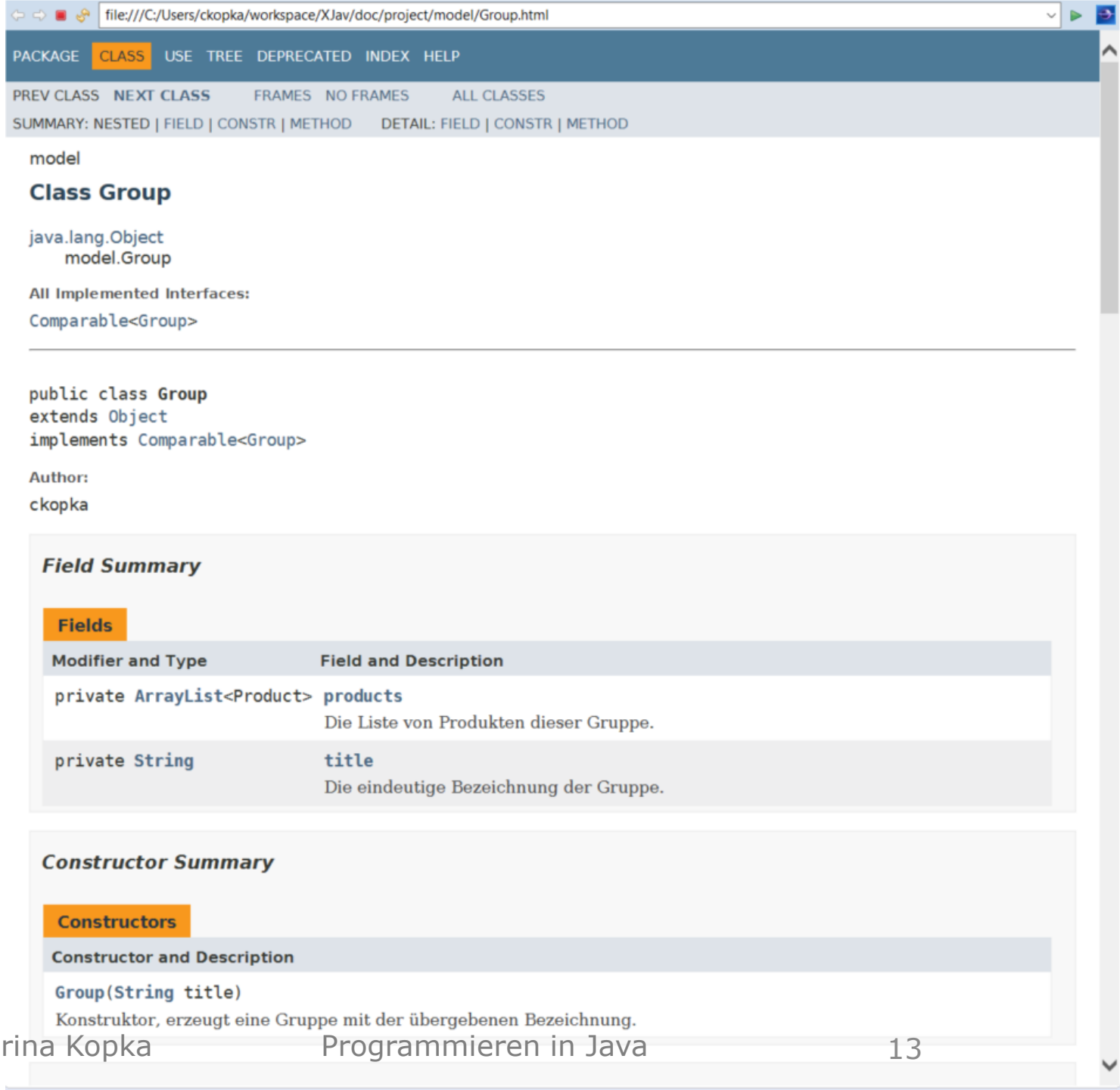
```
package model;
import java.util.ArrayList;
import java.util.Collections;
/**
 * @author ckopka
 *
 */
public class Group implements Comparable<Group> {
            /**
             * Die eindeutige Bezeichnung der Gruppe.
             */
            private String title;
            /**
             * Die Liste von Produkten dieser Gruppe.
             */
            private ArrayList<Product> products;
            /**
             * Konstruktor, erzeugt eine Gruppe mit der übergebenen Bezeichnung.
             *
             * @param title
             *            Die eindeutige Bezeichnung der Gruppe.
             * @preconditions <b>title</b> darf nicht <em>null</em> und nicht leer sein
             *                und muss eindeutig sein.
             */
            public Group(String title) {
                        this.title = title;
                        products = new ArrayList<Product>();
            }
            //…
```

Beispiel II

```
        /**
         * Fügt ein Produkt zu dieser Gruppe hinzu und sortiert die
         * Produktliste der Gruppe.<br>
         * Sortierung anhand: {@link Product#compareTo(Product)}.
         * @param product
         *            Das neue Produkt.
         */
        public void addProduct(Product product) {
                products.add(product);
                Collections.sort(products);
        }
        /**
         * Vergleichsfunktion zweier Gruppen anhand der Bezeichnung für die
         * Sortierung der Gruppen.
         */
        @Override
        public int compareTo(Group group) {
                return this.title.compareTo(group.title);
        }
        /**
         * Gibt die Liste von Produkten dieser Gruppe zurück.
         *
         * @return Die Liste von Produkten dieser Gruppe.
         */
        public ArrayList<Product> getProducts() {
                return products;
        }
//. . .
}
```

Beispiel

Klasse Group


file:///C:/Users/ckopka/workspace/XJav/doc/project/model/Group.html

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

model

Class Group

java.lang.Object
    model.Group

All Implemented Interfaces:

Comparable<Group>

```
public class Group
extends Object
implements Comparable<Group>
```

Author:

ckopka

Field Summary

Fields

Modifier and Type	Field and Description
private ArrayList<Product>	products Die Liste von Produkten dieser Gruppe.
private String	title Die eindeutige Bezeichnung der Gruppe.

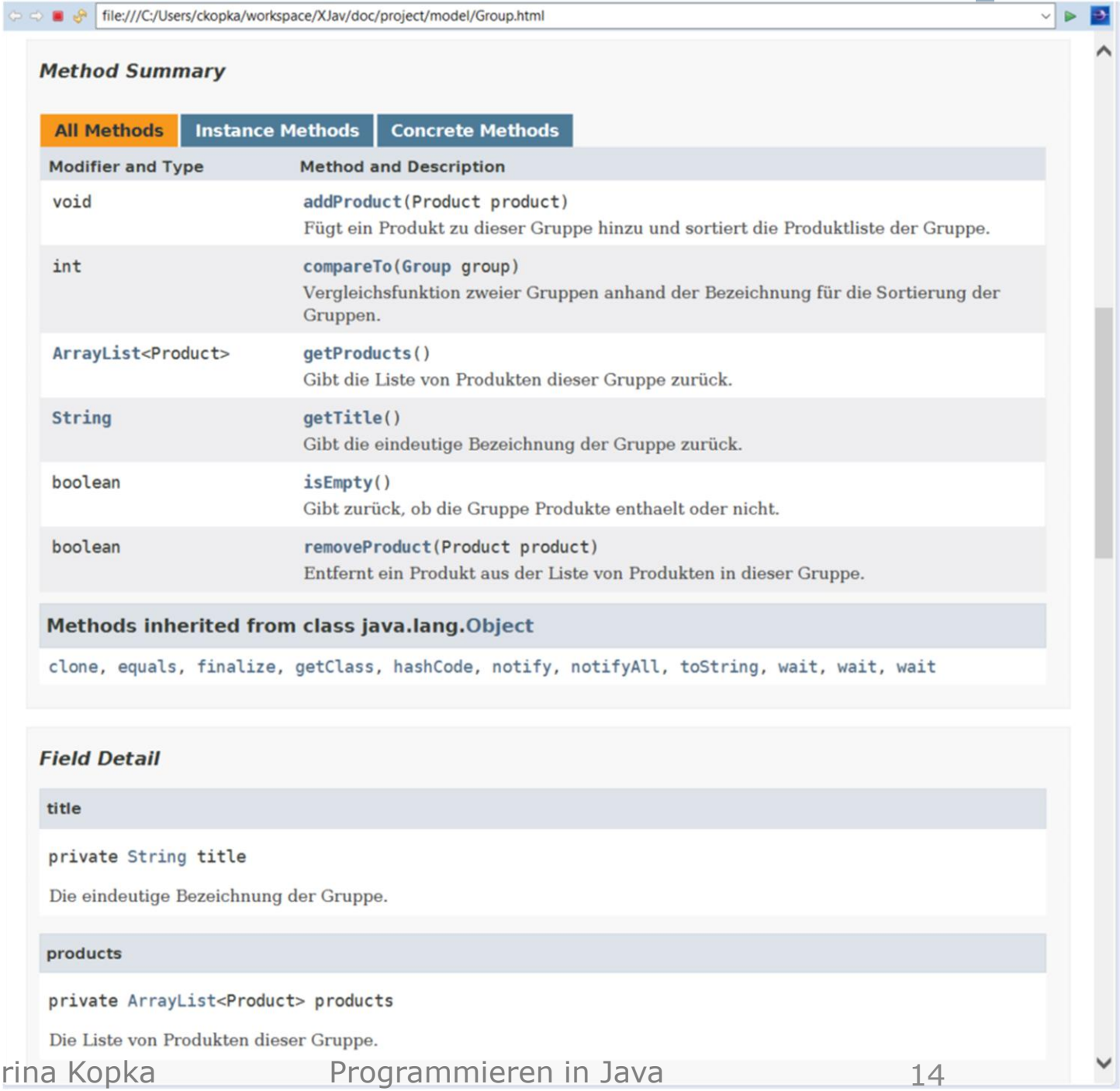
Constructor Summary

Constructors

Constructor and Description
Group(String title) Konstruktor, erzeugt eine Gruppe mit der übergebenen Bezeichnung.

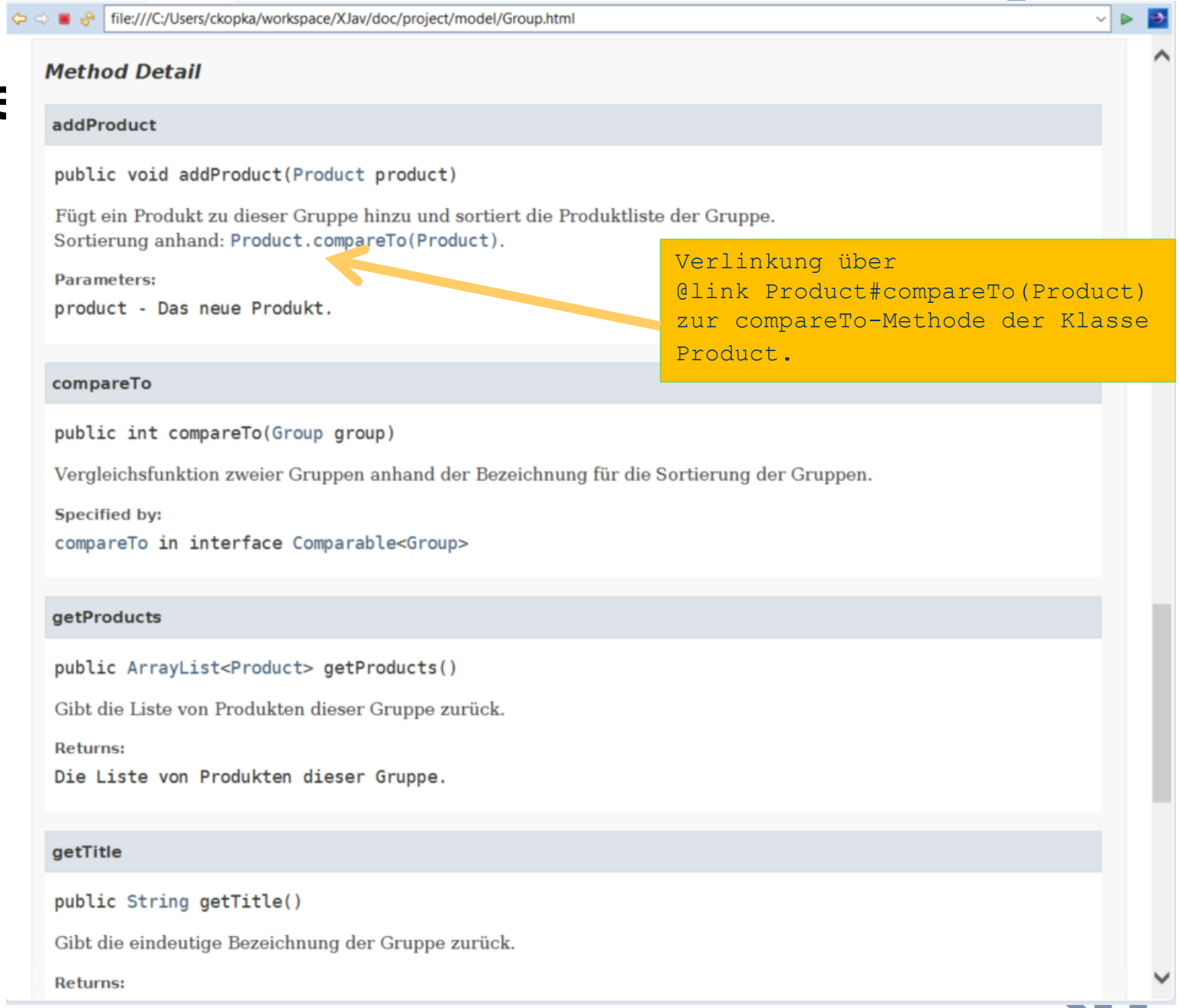
Beispiel

Klasse Group



Beispie

Klasse Group



Verlinkung über
@link Product#compareTo(Product)
zur compareTo-Methode der Klasse Product.