Faktor-IPS Tutorial

Table of Contents

Teil 1: Modellierung und Produktkonfigurierung

Einleitung

Dieses Tutorial führt in die Konzepte von und die Arbeitsweise mit Faktor-IPS ein. Faktor-IPS ist ein OpenSource-Werkzeug zur modellgetriebenen Entwicklung [1] versicherungsfachlicher Softwaresysteme mit Fokus auf der einheitlichen Abbildung des Produktwissens. Insbesondere können mit Faktor-IPS nicht nur die Modelle der Systeme bearbeitet, sondern auch die Produktinformationen selbst verwaltet werden.

Neben reinen Produktdaten können einzelne Produktaspekte auch über eine Excel ähnliche Formelsprache definiert werden. Darüber hinaus können Tabellen verwaltet und fachliche Testfälle definiert und ausgeführt werden.

Als durchgängiges Beispiel verwenden wir dazu eine stark vereinfachte Hausratversicherung. Die grundlegenden Konstruktions- und Modellierungsprinzipien lassen sich auch anhand dieses sehr fachlichen Modells darstellen. Insbesondere behandeln wir in dem Tutorial die Möglichkeiten zur Produktkonfiguration.

[1] Model driven software development (MDSD). Eine sehr gute Beschreibung der zugrundeliegenden Konzepte findet sich in Stahl, Völter: Modellgetriebene Softwareentwicklung.

Das Tutorial ist in zwei Teile gegliedert:

1. Der erste Teil führt in die Arbeit mit dem Modellierungswerkzeug und dem generierten Sourcecode ein und zeigt wie konkrete Produkte konfiguriert werden.
2. Der zweite Teil beschreibt die Verwendung von Tabellen, die Implementierung der Beitragsberechnung und zeigt, wie mit Hilfe von Formeln das Hausratmodell flexibel gestaltet werden kann.

Das Tutorial ist für Softwarearchitekten und Entwickler mit fundierten Kenntnissen über objektorientiere Modellierung mit der UML geschrieben. Erfahrungen mit der Entwicklung von Java-Anwendungen in Eclipse sind hilfreich, aber nicht unbedingt erforderlich.

Wenn Sie die einzelnen Schritte des Tutorials nicht selber durchführen wollen, können Sie das Endergebnis auch von https://www.faktorzehn.org/de/dokumentation/ herunterladen und installieren.

Überblick über Teil 1 des Tutorials

Der erste Teil des Tutorials ist wie folgt gegliedert:

- **Hello Faktor-IPS**
  In diesem Kapitel wird ein erstes Faktor-IPS Projekt angelegt und eine erste Klasse definiert.
- **Arbeiten mit Modell und Sourcecode**
  Anhand eines Modells einer Hausratversicherung wird der Umgang mit dem Modellierungswerkzeug und dem generierten Sourcecode erläutert.
- **Erweitern des Hausratmodells**
  Das Hausratsmodell wird vervollständigt indem der Klasse HausratVertrag weitere Attribute hinzugefügt werden.
- **Aufnahme von Produktaspekten ins Modell**
  In diesem Kapitel wird erläutert, wie der Produktaspekt im Modell abgebildet wird.
- **Definition der Hausratprodukte**
  Auf Basis des Modells werden nun zwei Hausratprodukte erfasst. Hierzu wird die Produktdefinitionsperspektive verwendet, die speziell für die Fachabteilung konzipiert ist.
- **Zugriff auf Produktinformationen zur Laufzeit**
  In dem Kapitel wird erläutert wie man zur Laufzeit, also in einer Anwendung oder einem Testfall auf Produktinformationen zugreift.

Hello Faktor-IPS

Im ersten Schritt dieses Tutorials legen wir ein Faktor-IPS Projekt an, definieren eine Modellklasse und generieren Java-Sourcecode zu dieser Modellklasse.

Falls Sie Faktor-IPS noch nicht installiert haben, tun Sie das jetzt. Die Software und die Installationsanleitung finden Sie auf https://www.faktorzehn.org/de/download/. In diesem Tutorial verwenden wir Eclipse 4.20 (2021-06) und Faktor-IPS 21.6. Eclipse wird auf Englisch verwendet. Faktor-IPS hingegen auf Deutsch (also mit installiertem Language Pack).

Starten Sie Eclipse. Am besten verwenden Sie für dieses Tutorial einen eigenen Workspace. Wenn Faktor-IPS korrekt installiert ist, sollten Sie bei geöffneter *Java-Perspektive* [2] in der Toolbar folgende Symbole sehen [3]:

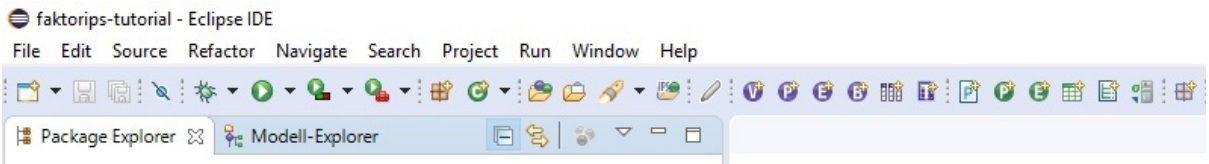

Figure 1. Faktor-IPS Icons

[2] In der Menüleiste *Window* ► *Perspective* ► *Open Perspective* ► *Java* auswählen
[3] In den Tutorials von Faktor-IPS wird von einer Installation mit Faktor-IPS German

Language-Pack ausgegangen. Wenn Sie das Language-Pack installiert haben, aber trotzdem ohne die Übersetzung arbeiten wollen, können sie einfach Eclipse mit einer anderen Locale starten, z.B. mit eclipse -vmargs -Duser.language=en

Falls der *Modell-Explorer* nicht sichtbar ist, liegt das daran, dass Sie diesen Workspace bereits vor der Installation von Faktor-IPS verwendet haben. Rufen Sie in diesem Fall im Menü *Window* ► *Perspective* ► *Reset Perspective* auf.

Faktor-IPS-Projekte sind normale Java-Projekte oder Maven-Projekte mit einer zusätzlichen Faktor-IPS-Nature. Als erstes legen Sie also ein neues Maven-Projekt mit dem Namen `„Hausratmodell“` an. Hierzu kann der Maven-Archetype für Faktor-IPS genutzt werden, der ein Maven-Projekt mit Faktor-IPS Nature erstellt.

Öffnen Sie dazu die Kommandozeile und navigieren Sie in den Ordner in dem Sie das Projekt anlegen möchten. Führen Sie dort den folgenden Befehl aus:

```
mvn archetype:generate -DarchetypeGroupId=org.faktorips
-DarchetypeArtifactId=faktorips-maven-archetype -DarchetypeVersion=21.6.0
-DgroupId=org.faktorips.tutorial -DartifactId=Hausratmodell -Dversion=1.0
-Dpackage=org.faktorips.tutorial.model -DJavaVersion=1.8 -DIPS-Language=de -DIPS
-IsModelProject=true -DIPS-IsProductDefinitionProject=false -DIPS-SourceFolder=model
-DIPS-RuntimeIdPrefix=hausrat. -DIPS-ConfigureIPSBuild=true
```

Dieser Befehl hat mit Hilfe des Faktor-IPS Archetype das Maven-Projekt `„Hausratmodell“` erstellt. Dem Projekt wurde die Faktor-IPS Nature und damit die Laufzeitbibliotheken von Faktor-IPS hinzugefügt.
In diesem Projekt werden wir die Modellklassen anlegen. Das Sourceverzeichnis in dem die Modelldefinitionen abgelegt werden haben wir `„model“` genannt (durch den Parameter `-DIPS-SourceFolder=model`). Unterhalb dieses Verzeichnisses kann die Modellbeschreibung wie in Java durch Packages (Pakete) strukturiert werden. Faktor-IPS verwendet wie Java qualifizierte Namen zur Identifikation der Klassen des Modells.
Das Basispackage für die generierten Java-Klassen wurde `„org.faktorips.tutorial.model“` genannt (durch den Parameter `-Dpackage=org.faktorips.tutorial.model`). Als Runtime-ID-prefix haben wir `„hausrat.“` gewählt. Die Bedeutung des Runtime-ID-Prefixes wird im Kapitel „Definition der Produkte" erläutert.
Die genauen Funktionen der verwendeten Parameter können in der Dokumentation des Archetype nachgelesen werden: https://www.faktorzehn.org/dokumentation/faktor-ips-projekte-mit-maven-archetypes-erstellen/.

Anschließend muss das neu erstellte Projekt in den Eclipse Workspace importiert werden. Rufen Sie dazu *File* ► *Import* ► *Maven*► *Existing Maven Projects* auf. Im Dialog wählen Sie dann den Projektordner als Root Directory aus und importieren dann das Projekt mit einem Klick auf `Finish`.

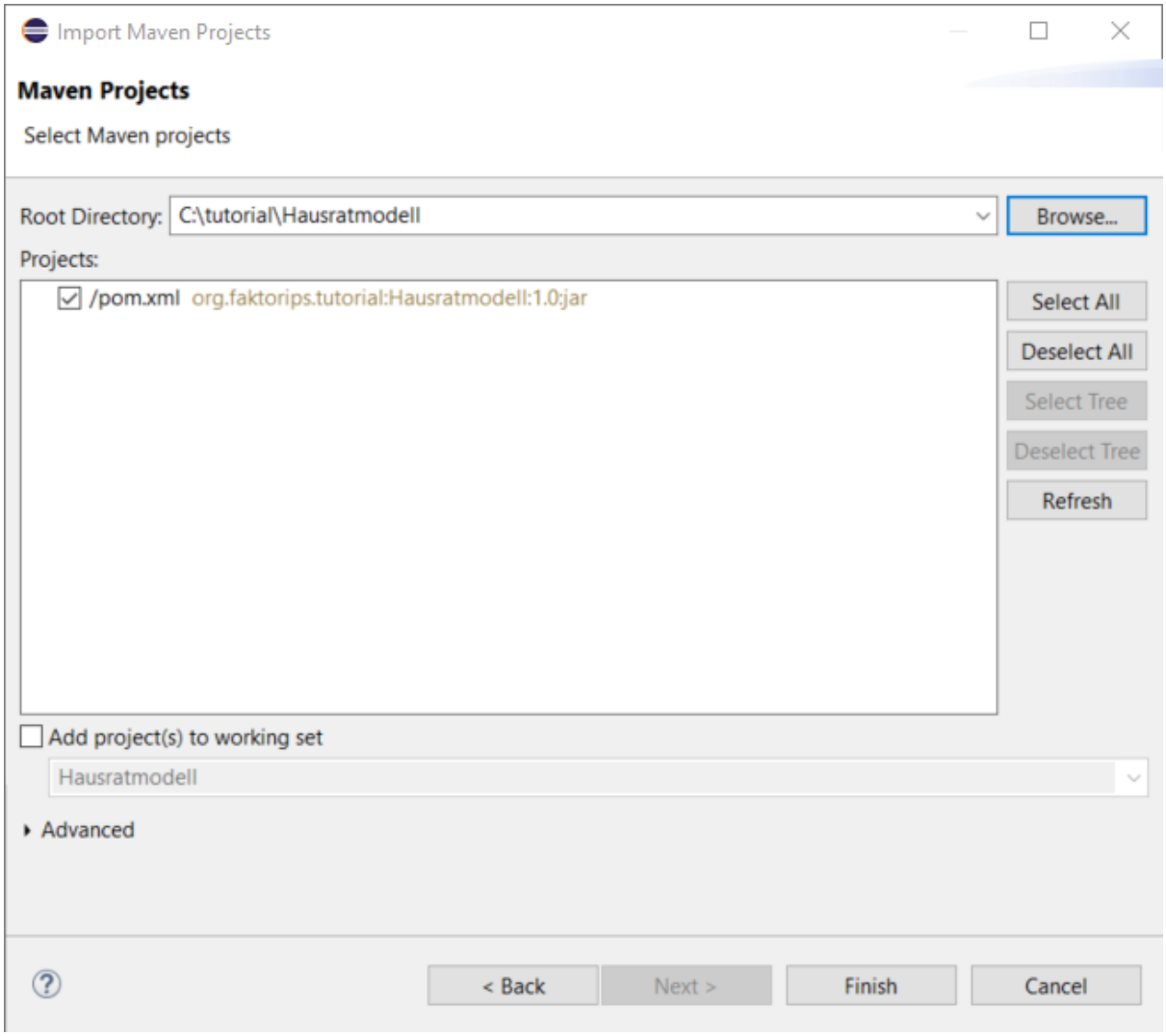

Figure 2. Maven Projekt in Eclipse importieren

Bevor wir die erste Klasse „`HausratVertrag`“ definieren, stellen Sie noch ein, dass der Workspace automatisch gebaut wird (im Menü: *Project* ▶ *Build automatically*).

Wechseln Sie zunächst in den *Modell-Explorer* von Faktor-IPS direkt neben dem *Package-Explorer*.

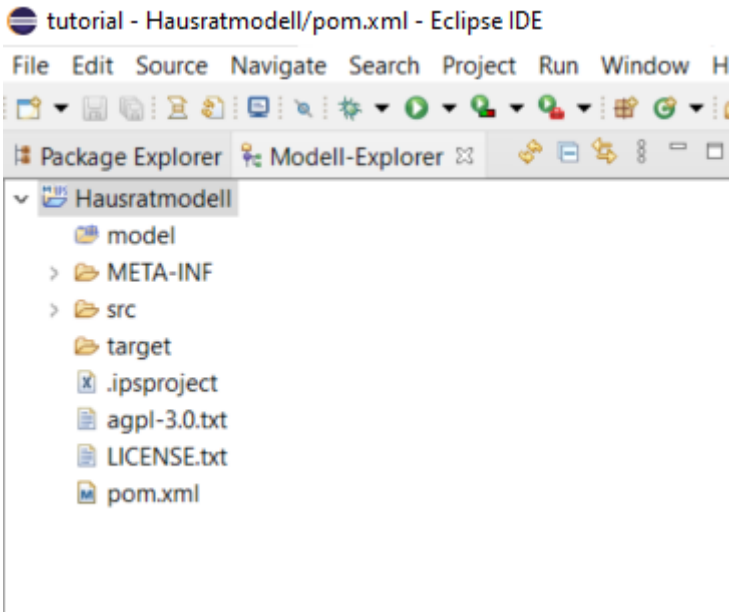

Figure 3. Ansicht der Projekte im Modell-Explorer

Im *Modell-Explorer* wird die Modelldefinition ohne die Java-Details dargestellt. In der Datei „`.ipsproject`“ sind die Eigenschaften des Faktor-IPS Projektes gespeichert. Hierzu gehören zum Beispiel die gerade im *Add IpsNature*-Dialog eingegebenen Informationen, Einstellungen für die Codegenerierung, die erlaubten Datentypen etc. Der Inhalt ist in XML abgelegt und ausführlich in der Datei dokumentiert.

Die Klassen werden wir in einem Package mit dem Namen „`hausrat`“ ablegen. Zum Anlegen des IPS Packages Rechtsklick auf das Sourceverzeichnis „`model`“, *neu* ▶ *IPS Package*. Danach den Namen des neuen Packages („`hausrat`“) eingeben und anschließend auf Button Finish drücken. Alternativ können Sie IPS Packages über den Button  , in der Toolbar, anlegen.

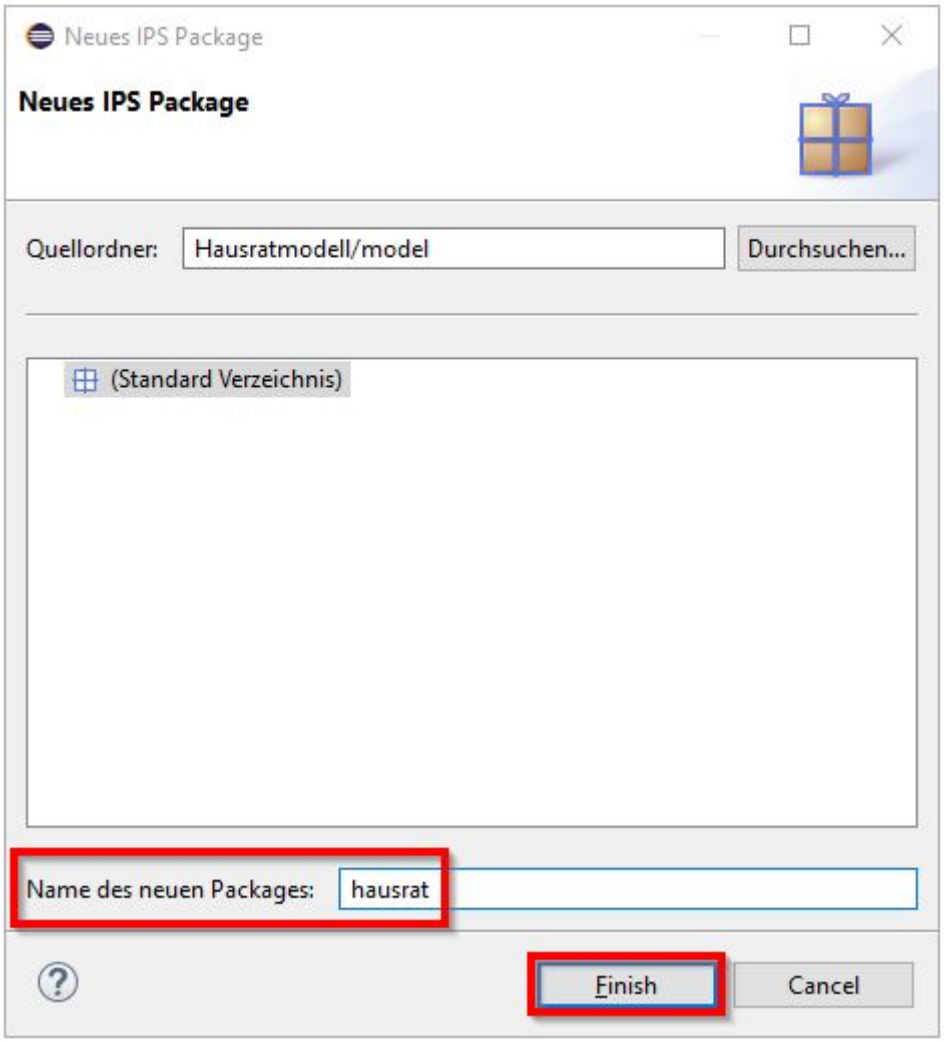

Figure 4. Anlegen eines IPS Packages

Als nächstes wollen wir eine Klasse anlegen, die unseren Hausratvertrag repräsentiert. Markieren Sie dazu das neu angelegte Package im Package Explorer und drücken auf den Button in der Toolbar.

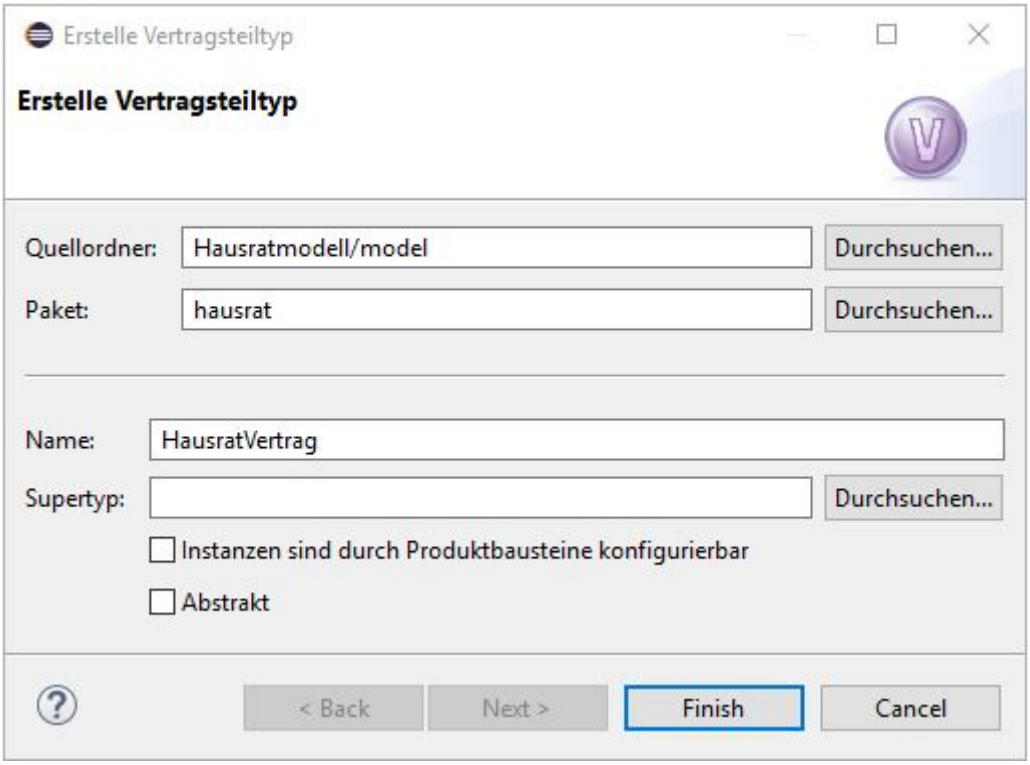

Figure 5. Anlegen einer neuen Vertragsklasse

In dem Dialog sind Sourceverzeichnis und Package bereits entsprechend vorbelegt und Sie geben noch den Namen der Klasse an, also „HausratVertrag“ und klicken auf *Finish*. Faktor-IPS hat jetzt die neue Klasse angelegt und den Editor zur Bearbeitung geöffnet. Wechseln Sie zurück in den Package-Explorer. Sie sehen, dass die Klasse „HausratVertrag“ in einer eigenen Datei mit dem Namen „HausratVertrag.ipspolicycmpttype“ gespeichert ist.

Weiterhin hat der Codegenerator von Faktor-IPS bereits zwei Java-Sourcefiles erzeugt „org.faktorips.tutorial.model.hausrat.HausratVertrag“ und „org.faktorips.tutorial.model.hausrat.HausratVertragBuilder“.

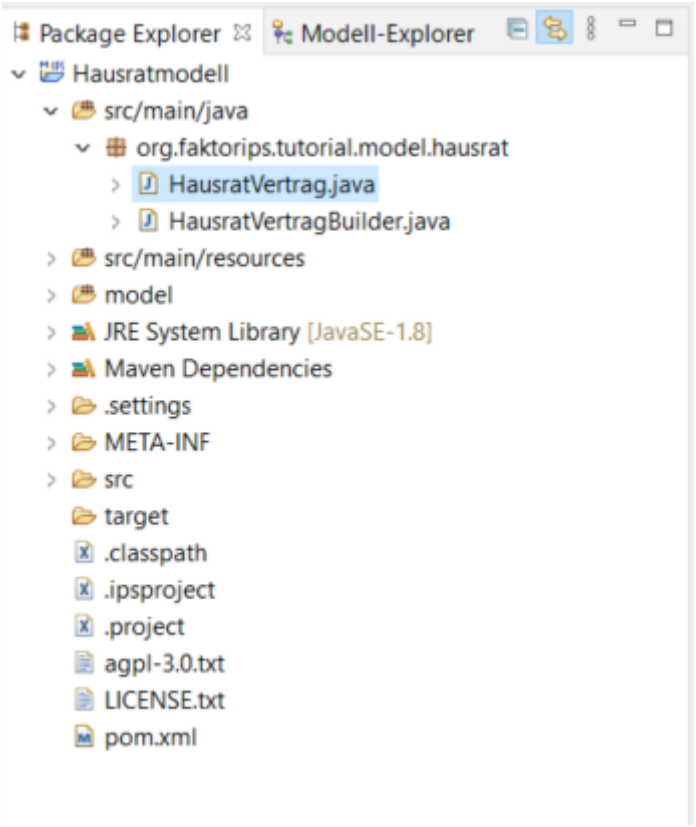

Figure 6. Generierte Klassen

Ein kurzer Blick in den Sourcecode von „`HausratVertrag`“ zeigt, dass hier schon einige Methoden generiert worden sind. Diese Methoden dienen unter anderem zur Konvertierung der Objekte in XML und zur Unterstützung von Prüfungen.

Die Klasse „`HausratVertragBuilder`“ erzeugt nach dem Erbauer-Entwurfsmuster Vertragsstrukturen.

Arbeiten mit Modell und Sourcecode

Im zweitem Schritt des Tutorials erweitern wir unser Modell und arbeiten mit dem generierten Sourcecode.

Als erstes erweitern wir die Klasse „`HausratVertrag`“ um ein Attribut `zahlweise`. Wenn der Editor mit der Vertragsklasse nicht mehr geöffnet ist, öffnen Sie diesen nun durch Doppelklick im Modell-Explorer. In dem Editor klicken Sie auf den Button *Neu...* im Abschnitt *Attribute*. Es öffnet sich der folgende Dialog:

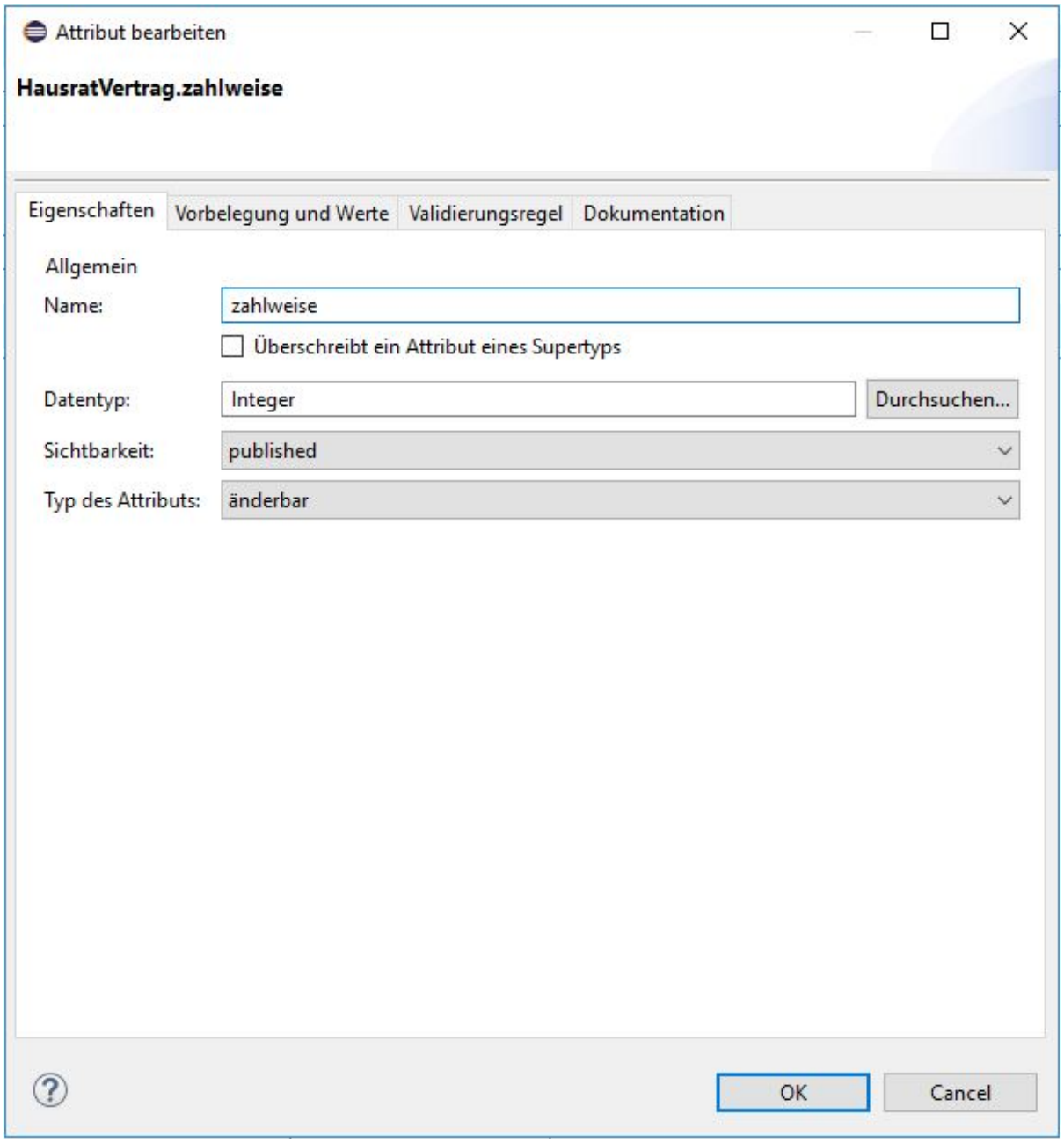

Figure 7. Dialog zum Anlegen eines neuen Attributs

Die Felder haben folgende Bedeutung:

Table 1. Attribut Felder

Feld	**Bedeutung**
Name	Der Name des Attributs
Checkbox	Indikator, ob dieses Attribut bereits in einer Superklasse definiert worden ist und in dieser Klasse lediglich Eigenschaften wie z.B. der Default Value überschrieben werden [5]
Datatype/Datentyp	Datentyp des Attributs

Feld	Bedeutung
Modifier/Sichtbarkeit	Analog zum Modifier in Java. Der zusätzliche Modifier published bedeutet, dass die Eigenschaften ins published Interface aufgenommen wird. [6]
Attribute type/Typ des Attributs	Der Typ des Attributs. * **änderbar** Änderbare Eigenschaften, also solche mit Getter- und Setter- Methoden * **konstant** Konstante, nicht änderbare Eigenschaft * **abgeleitet (cached, Berechnung durch expliziten Methodenaufruf)** Abgeleitete Eigenschaften im UML Sinne. Die Eigenschaft wird durch einen expliziten Methodenaufruf berechnet und das Ergebnis ist danach über die Getter-Methode abfragbar. Zum Beispiel kann die Eigenschaft bruttobeitrag durch eine Methode berechneBeitrag berechnet und danach über `getBruttobeitrag()` abgerufen werden. * **abgeleitet (Berechnung bei jedem Aufruf der Gettermethode)** Abgeleitete Eigenschaften im UML Sinne. Die Eigenschaft wird bei jedem Aufruf der Getter-Methode berechnet. Zum Beispiel kann das Alter einer versicherten Person bei jedem Aufruf von `getAlter()` aus dem Geburtstag ermittelt werden.

[5] Entspricht der @override Annotation in Java 5.

[6] **Hinweis:** Die Aktivierung/Deaktivierung der Generierung der Published-Inferfaces erfolgt über das Kontextmenü > Eigenschaften > Faktor-IPS Code Generator des entsprechenden Projekts

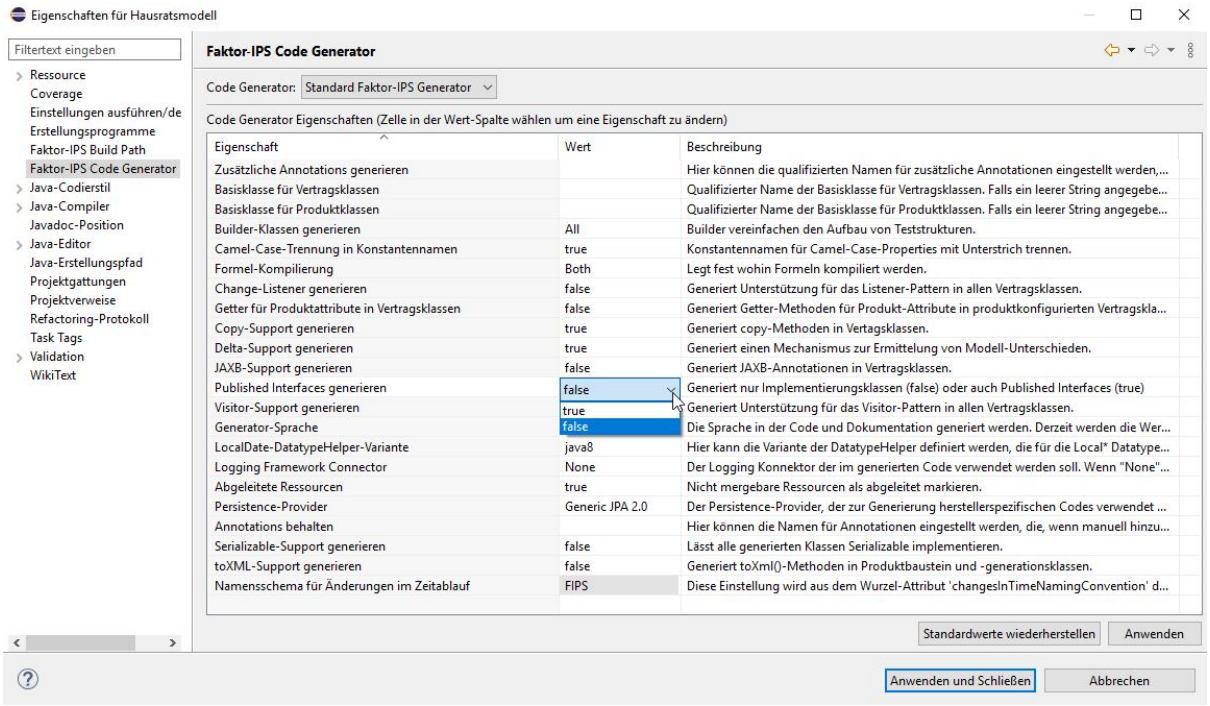

Figure 8. Aktivierung/Deaktivierung der Generierung der Published-Interfaces

Geben Sie als Namen `zahlweise` und als Datentyp *Integer* ein. Wenn Sie auf den Browse Button neben dem Feld klicken, öffnet sich eine Liste mit den verfügbaren Datentypen. Alternativ dazu können Sie wie in Eclipse üblich auch mit STRG-Space eine Vervollständigung durchführen. Wenn Sie zum Beispiel „D“ eingeben und STRG-Space drücken, sehen Sie alle Datentypen, die mit „D“ beginnen. Die anderen Felder lassen Sie wie vorgegeben und drücken jetzt *OK*, danach speichern Sie die geänderte Vertragsklasse.

Der Codegenerator hat nun bereits die Java-Sourcefiles aktualisiert. Die Klasse „`HausratVertrag`“ enthält nun Zugriffsmethoden für das Attribut und speichert den Zustand in einer privaten Membervariable.


```
/**
 * Membervariable fuer zahlweise.
 *
 * @generated
 */
private Integer zahlweise = null;

/**
 * Erzeugt eine neue Instanz von HausratVertrag.
 *
 * @generated
 */
public HausratVertrag() {
    super();
    productConfiguration = new ProductConfiguration();
}

/**
 * Gibt den Wert des Attributs zahlweise zurueck.
 *
 * @generated
 */
@IpsAttribute(name = "zahlweise", kind = AttributeKind.CHANGEABLE,
valueSetKind = ValueSetKind.AllValues)
public Integer getZahlweise() {
    return zahlweise;
}

/**
 * Setzt den Wert des Attributs zahlweise.
 *
 * @generated
 */
@IpsAttributeSetter("zahlweise")
public void setZahlweise(Integer newValue) {
    this.zahlweise = newValue;
}
```

Das JavaDoc für die Membervariable und für die Getter-Methode ist mit einem `@generated` markiert. Dies bedeutet, dass die Methode zu 100% generiert wird. Bei einer erneuten Generierung wird dieser Code genau so wieder erzeugt, unabhängig davon, ob er in der Datei gelöscht oder modifiziert worden ist. Änderungen seitens des Entwicklers werden also überschrieben. Möchten Sie die Methode modifzieren, so fügen Sie hinter die Annotation `@generated` ein `NOT` hinzu. Probieren wir das einmal aus. Fügen Sie jeweils eine Zeile in die Getter- und Setter-Methode ein, und ergänzen bei der Methode `setZahlweise()`

mit `NOT` hinter der Annotation:

```
/**
 * Gibt den Wert des Attributs zahlweise zurueck.
 *
 * @generated
 */
@IpsAttribute(name = "zahlweise", kind = AttributeKind.CHANGEABLE,
valueSetKind = ValueSetKind.Enum)
public Integer getZahlweise() {
    System.out.println("getZahlweise");
    return zahlweise;
}
```

```
/**
 * Setzt den Wert des Attributs zahlweise.
 *
 * @generated NOT
 */
@IpsAttributeSetter("zahlweise")
public void setZahlweise(Integer newValue) {
    System.out.println("setZahlweise");
    this.zahlweise = newValue;
}
```

Generieren Sie jetzt den Sourcecode für die Klasse „`HausratVertrag`“ neu. Dies können Sie wie in Eclipse üblich auf zwei Arten erreichen:

- Entweder bauen Sie mit *Project* ► *Clean* das gesamt Projekt neu, oder
- Sie speichern die Modellbeschreibung der Klasse „`HausratVertrag`“ erneut.

Nach dem Generieren ist das `System.out.println(⋯)`-Statement aus der Getter-Methode entfernt worden, in der Setter-Methode ist es erhalten geblieben.

Methoden und Attribute, die neu hinzugefügt werden, bleiben nach der Generierung erhalten. Auf diese Weise kann der Sourcecode beliebig erweitert werden.

Nun erweitern wir noch die Modelldefinition der Zahlweise um die erlaubten Werte. Öffnen Sie dazu den Dialog zum Bearbeiten des Attributes und wechseln auf die zweite Tabseite. Bisher sind alle Werte des Datentyps als zulässige Werte für das Attribute erlaubt. Wir schränken dies nun auf 1, 2, 4, 12 für jährlich, halbjährlich, quartalsweise und monatlich ein. Ändern Sie hierzu den Typ auf „Aufzählung“ und geben Sie in die Tabelle die Werte 1, 2, 4 und 12 ein [7].

[7] Faktor-IPS unterstützt auch die Definition von Enums. Auf dieses Feature wird an dieser Stelle aber verzichtet. Darüber hinaus können über einen Extension Point beliebige Java Klassen als Datentyp registriert werden. Diese Java-Klassen sollten dabei entsprechend des Musters *ValueObject* realisiert sein.

Setzen Sie nun einmal den Default Value auf 0. Faktor-IPS markiert den Default Value mit einer Warnung, da der Wert nicht in der im Modell erlaubten Wertemenge enthalten ist.

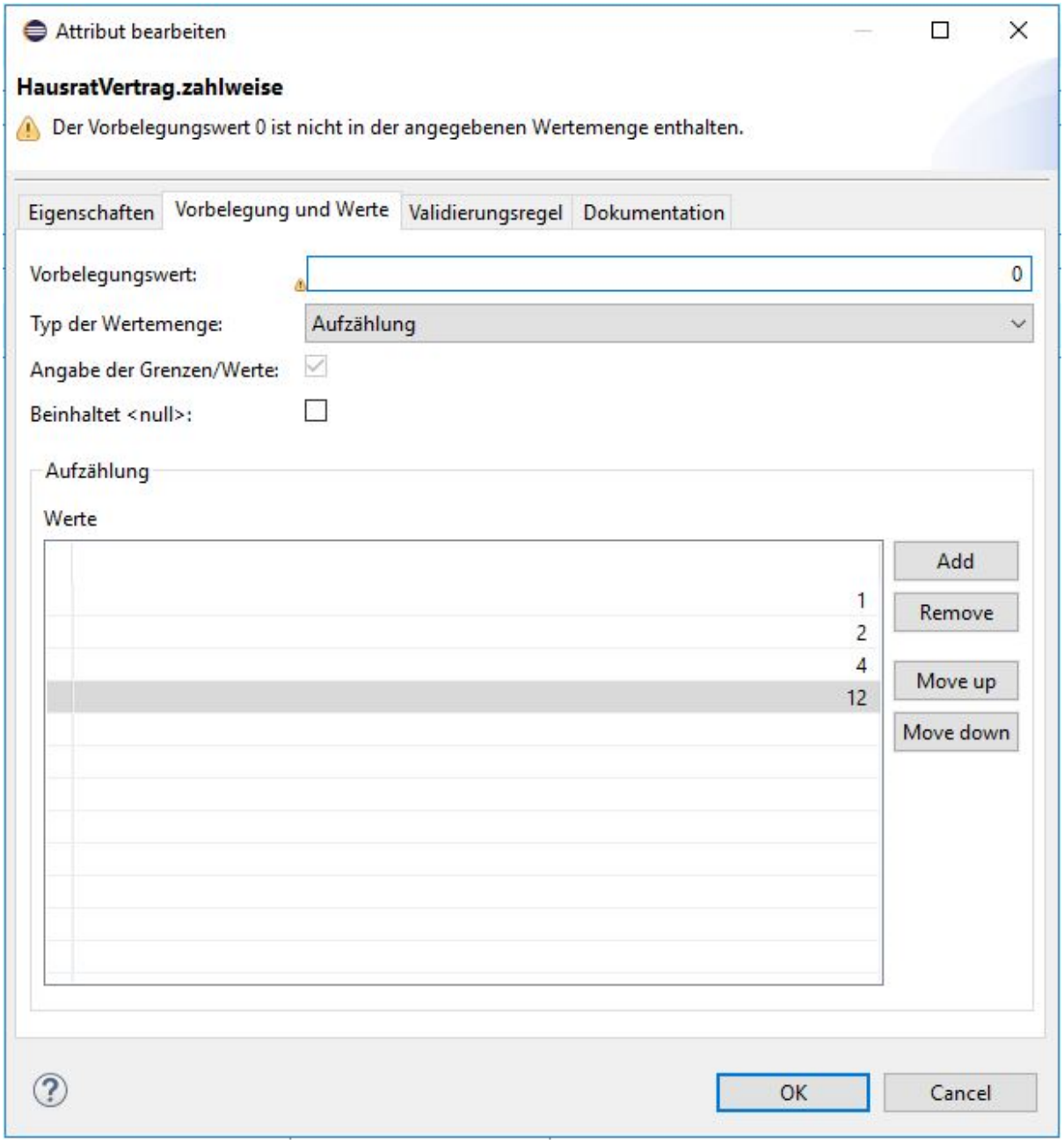

Figure 9. Wertebereich für das Attribut `zahlweise`

Es handelt sich also um einen möglichen Fehler im Modell. Lassen wir das aber für einen Augenblick so stehen. Das gibt uns die Gelegenheit die Fehlerbehandlung von Faktor-IPS zu erläutern. Schließen Sie dazu den Dialog und speichern die Vertragsklasse. Im *Problems-View* von Eclipse wird nun die gleiche Warnung wie im Dialog angezeigt. Faktor-

IPS lässt Fehler und Inkonsistenzen im Modell zu und informiert den Benutzer darüber im Eclipse-Stil, also in den Editoren und als so genannte Problemmarker, die im *Problems-View* und in den Explorern sichtbar sind.

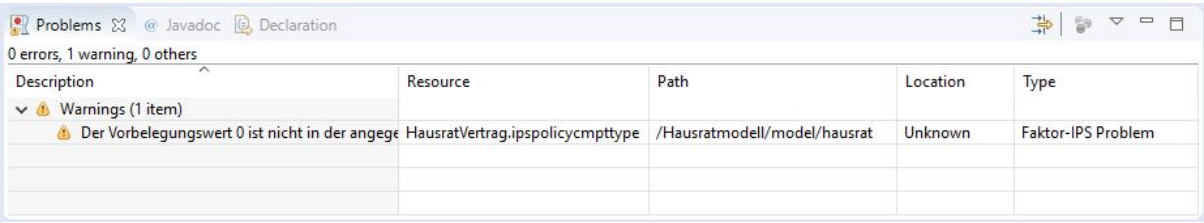

Figure 10. Anzeige von Fehlern im Problems-View

Löschen Sie die „0“ als Vorbelegungswert und speichern Sie die Vertragsklasse. Die Warnung wird damit wieder aus dem Problems-View entfernt.

> *Faktor-IPS generiert eine Warnung und keinen Fehler, da es durchaus sinnvoll sein kann, wenn der Defaultwert nicht im Wertebereich ist. Insbesondere gilt dies für den Defaultwert null. Wird zum Beispiel ein neuer Vertrag angelegt, so kann es gewolltes Verhalten sein, dass die Zahlweise nicht vorbelegt ist sondern noch null ist, um die Eingabe der Zahlweise durch den Benutzer zu erzwingen. Erst wenn der Vertrag vollständig erfasst ist, muss auch die Bedingung erfüllt sein, dass die Eigenschaft Zahlweise einen Wert aus dem Wertebereich enthält.*

Das Kapitel schließen wir mit der Definition einer Klasse „HausratGrunddeckung“ und der Kompositionsbeziehung zwischen „HausratVertrag“ und „HausratGrunddeckung“ gemäß dem folgenden Diagramm:

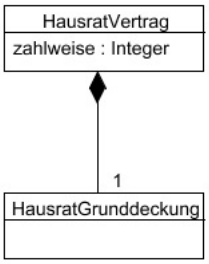

Figure 11. Modell der Vertragsseite

Legen Sie zunächst die Klasse „HausratGrunddeckung“ analog zur Klasse „HausratVertrag“ an. Wechseln Sie anschließend zu der Klasse „HausratVertrag“. Starten Sie den Assistenten zur Anlage einer neuen Beziehung. Klicken Sie auf den Button *Neu...* im Abschnitt *Beziehungen*. [8]

[8] In Faktor-IPS wird in Übereinstimmung mit der UML-Begriff Association verwendet. In dem Tutorial verwenden wir den im Sprachgebrauch üblicheren Begriff Beziehung.

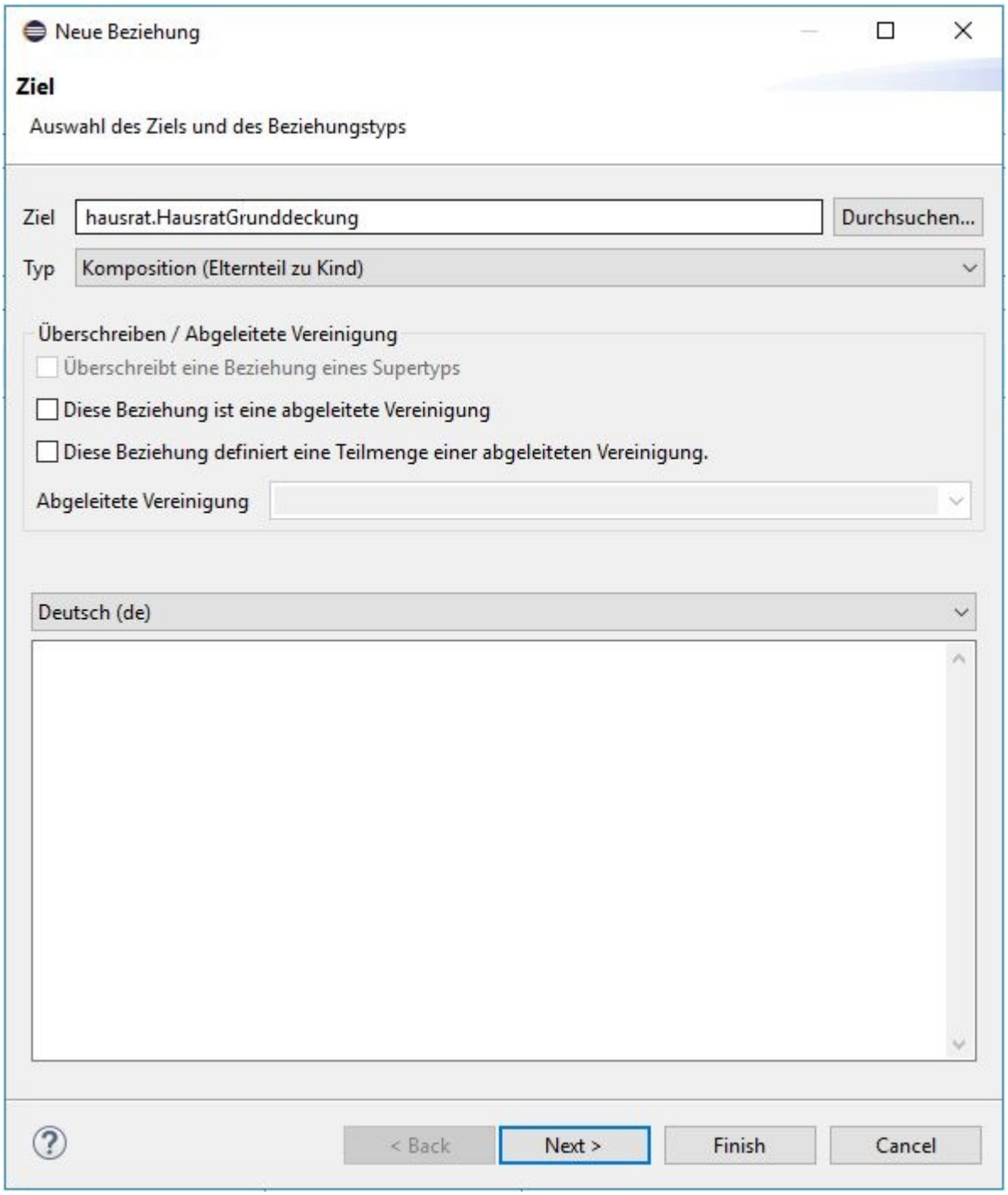

Figure 12. Anlegen einer neuen Beziehung

Als *Target* wählen Sie bitte die gerade angelegte Klasse „HausratGrunddeckung“ aus. Hier steht Ihnen wieder die Vervollständigung mit STRG-Space zur Verfügung. Bereich *Überschreiben / Abgeleitete Vereinigung* ignorieren Sie zunächst. Das Konzept wird im Tutorial zur Modellpartitionierung erläutert. Danach Button Next drücken.

Auf der nächsten Seite geben Sie als *Minimale Kardinalität* 1 und als *Maximale Kardinalität* 1 ein, als *Rollennamen* `HausratGrunddeckung` bzw. `HausratGrunddeckungen`. Die Mehrzahl wird verwandt, damit der Codegenerator verständlichen Sourcecode generieren kann.

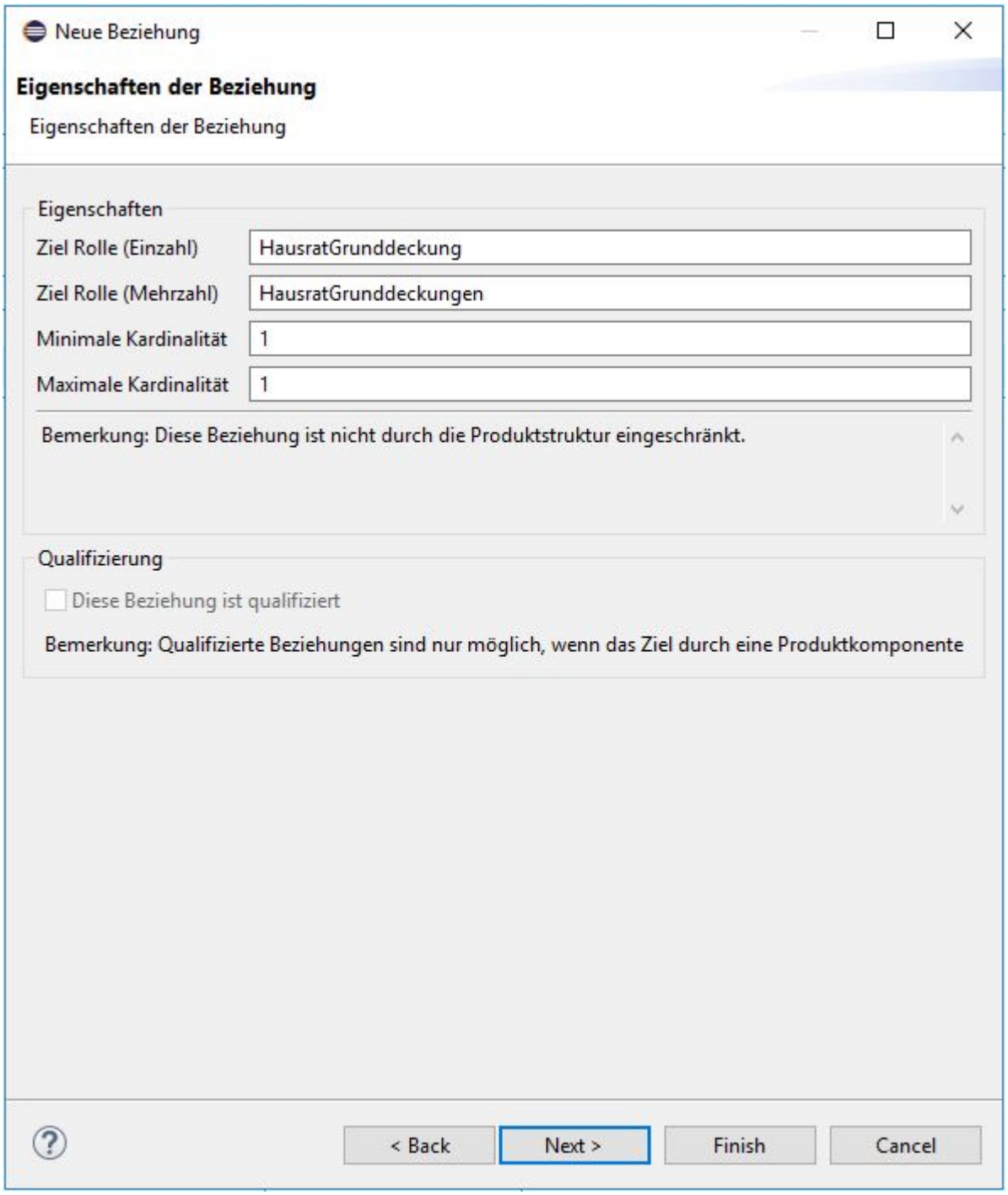

Figure 13. Rollennamen und Kardinalitäten einer Beziehung

Auf der nächsten Seite können Sie auswählen, ob es auch eine Rückwärtsbeziehung (*Inverse Beziehung*) von „HausratGrunddeckung“ zu „HausratVertrag“ geben soll. Beziehungen in Faktor-IPS sind immer gerichtet, so ist es auch möglich die Navigation nur in eine Richtung zuzulassen. Hier wählen Sie bitte *Neue inverse Beziehung* und gehen zur nächsten Seite.

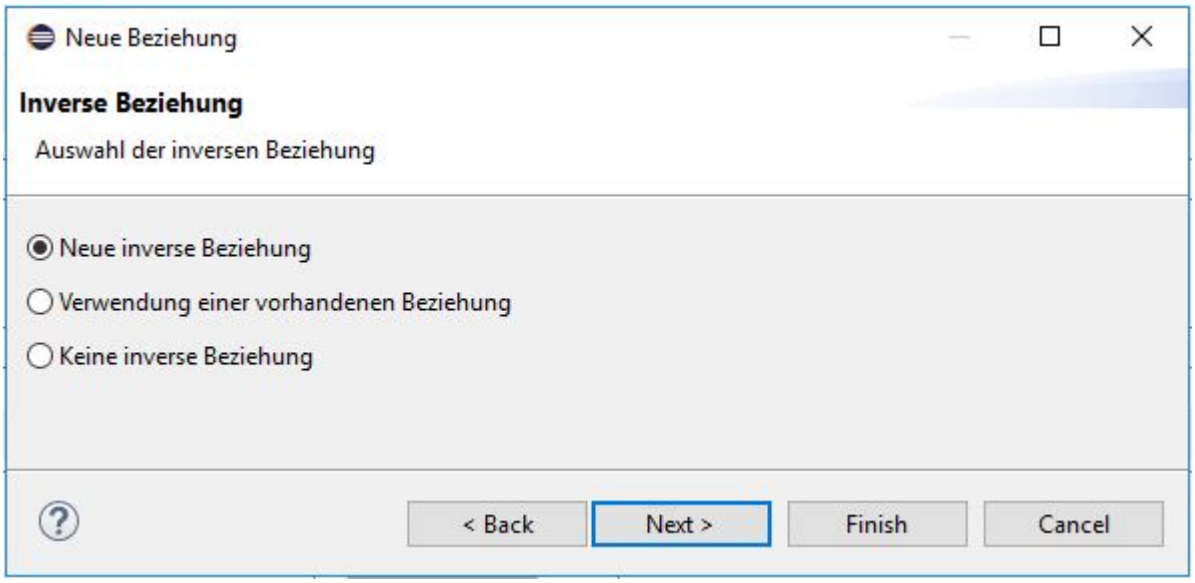

Figure 14. Neue inverse Beziehungs anlegen

Im nächsten Fenster geben Sie die Rollenbezeichnung der inversen Beziehung ein.

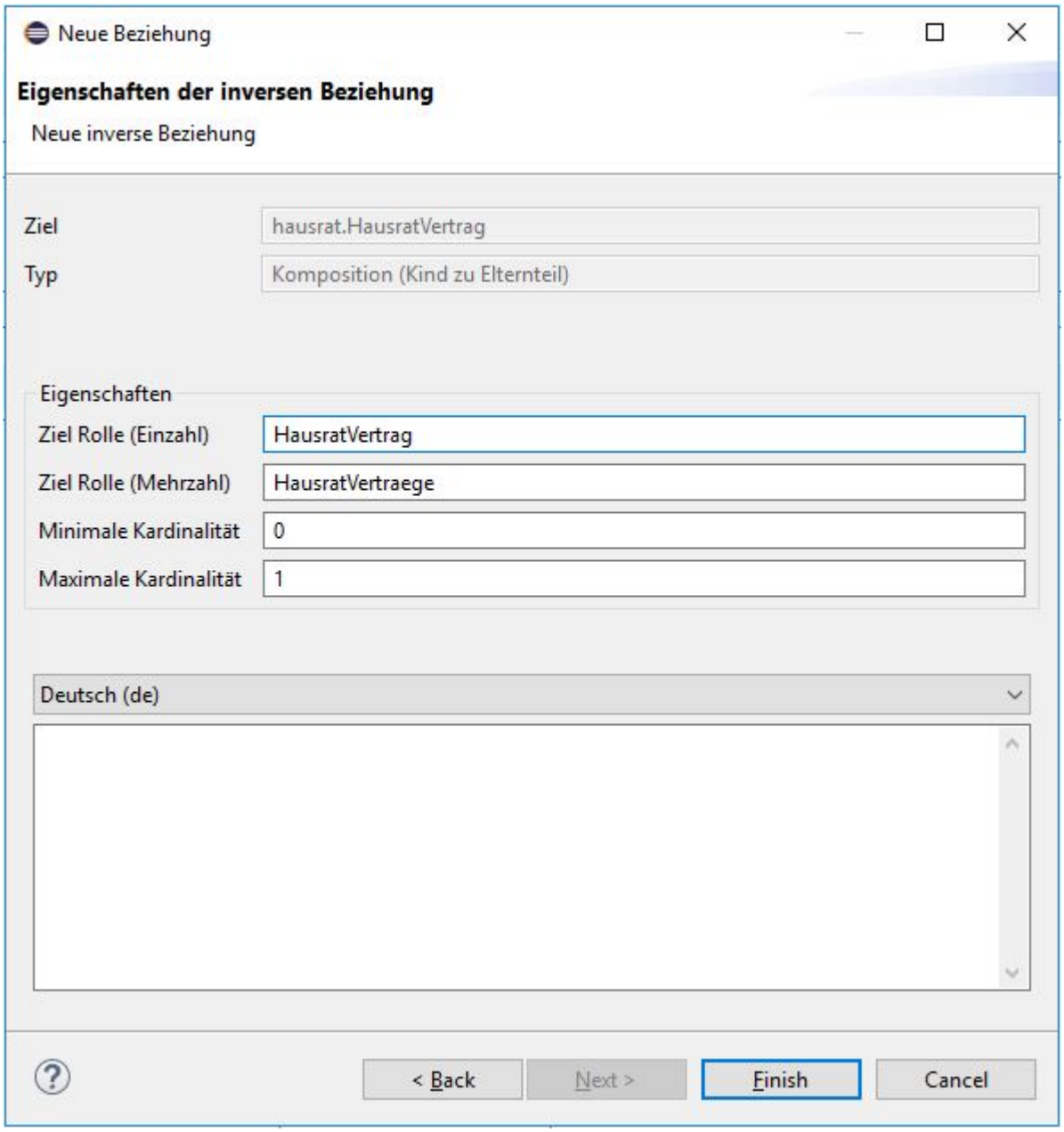

Figure 15. Eigenschaften der inversen Beziehung festlegen

Mit *Finish* legen Sie die beiden Beziehungen (vorwärts und rückwärts) an und speichern danach noch die Klasse „HausratVertrag“. Wenn Sie sich jetzt die Klasse „HausratGrunddeckung“ ansehen, ist dort die Rückwärtsbeziehung eingetragen.

Zum Abschluss werfen wir noch einen kurzen Blick in den generierten Sourcecode. In die Klasse „HausratVertrag“ wurden Methoden generiert, die Grunddeckung dem „HausratVertrag“ hinzuzufügen. In der Klasse „HausratGrunddeckung“ gibt es eine Methode, um zum „HausratVertrag“ zu navigieren. Wenn sowohl die Vorwärts- als auch die Rückwärtsbeziehung im Modell definiert ist, werden hierbei beide Richtungen berücksichtigt. Das heißt nach dem Aufruf von `setHausratGrunddeckung(HausratGrunddeckung d)` auf einer Instanz `v` von „HausratVertrag“ liefert `d.getHausratVertrag()` wieder `v` zurück. Dies zeigt ein kurzer Blick in die

Implementierung der Methode `setHausratGrunddeckung()` in der Klasse „`HausratVertrag`“:

```
public void setHausratGrunddeckung(HausratGrunddeckung newObject) {
    if (hausratGrunddeckung != null) {
        hausratGrunddeckung.setHausratVertragInternal(null);
    }
    if (newObject != null) {
        newObject.setHausratVertragInternal(this);
    }
    hausratGrunddeckung = newObject;
}
```

Der Vertrag wird in der Deckung als der Vertrag gesetzt, zu dem die Deckung gehört (zweites `if`-Statement der Methode).

Erweiterung des Hausratmodells

In diesem Abschnitt werden wir unser Hausratmodell vervollständigen. Die folgende Abbildung zeigt das Modell.

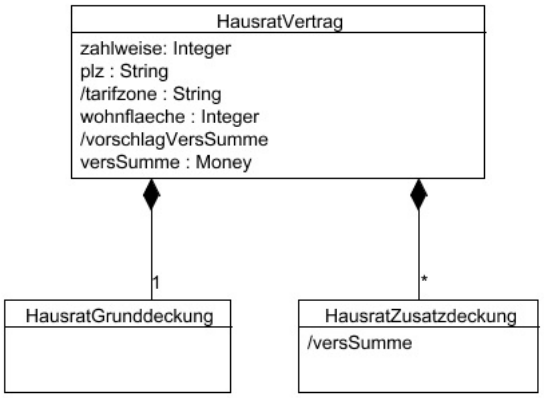

Figure 16. Hausratmodell mit Grunddeckung und Zusatzudeckung

Jeder Hausratvertrag hat genau eine Grunddeckung und kann beliebig viele Zusatzdeckungen haben. Die Grunddeckung deckt immer die im Vertrag definierte Versicherungssumme. Darüber hinaus kann ein Hausratvertrag optional Zusatzdeckungen enthalten, typischerweise sind dies Deckungen gegen Risiken wie zum Beispiel Fahrraddiebstahl oder Überspannungschäden. Die Zusatzdeckung werden wir im zweiten Teil des Tutorials näher betrachten.

Wir öffnen die Klasse „`HausratVertrag`“ und definieren die Attribute der Klasse (analog `zahlweise`):

Table 2. Attribut Felder der Klasse „`HausratVertrag`“

Name : Datentyp	Beschreibung, Bemerkung
`plz`: *String*	Postleitzahl des versicherten Hausrats
`/tarifzone` : *String*	Die Tarifzone (I, II, III, IV, V oder VI) ergibt sich aus der Postleitzahl und ist maßgeblich für den zu zahlenden Beitrag. → Achten Sie also bei der Eingabe darauf *Typ des Attributs auf abgeleitet (Berechnung bei jedem Aufruf der Gettermethode)* zu setzen!
`wohnflaeche` : *Integer*	Die Wohnfläche des versicherten Hausrats in Quadratmetern. Der erlaubte Wertebereich ist min=0 und unbeschränkt. Den Wertebereich definieren Sie auf der zweiten Seite des Dialogs *„Vorbelegung und Werte“*. In der Auswahlbox *„Typ der Wertemenge“* wählen Sie Bereich aus. Für *„Minimum“* geben Sie eine 0 ein, Felder *„Maximum“* und *„Schrittweite“* lassen Sie leer.
`/vorschlagVersSumme` : *Money*	Vorschlag für die Versicherungssumme. Wird auf Basis der Wohnfläche bestimmt. → Achten Sie bei der Eingabe darauf den *Typ des Attributs auf abgeleitet (Berechnung bei jedem Aufruf der Gettermethode)* zu setzen!
`versSumme` : *Money*	Die Versicherungssumme. Der erlaubte Wertebereich ist min=0 EUR und max lassen Sie leer.

Der Editor, der die Klasse `„HausratVertrag“` anzeigt, sieht wie folgt aus:

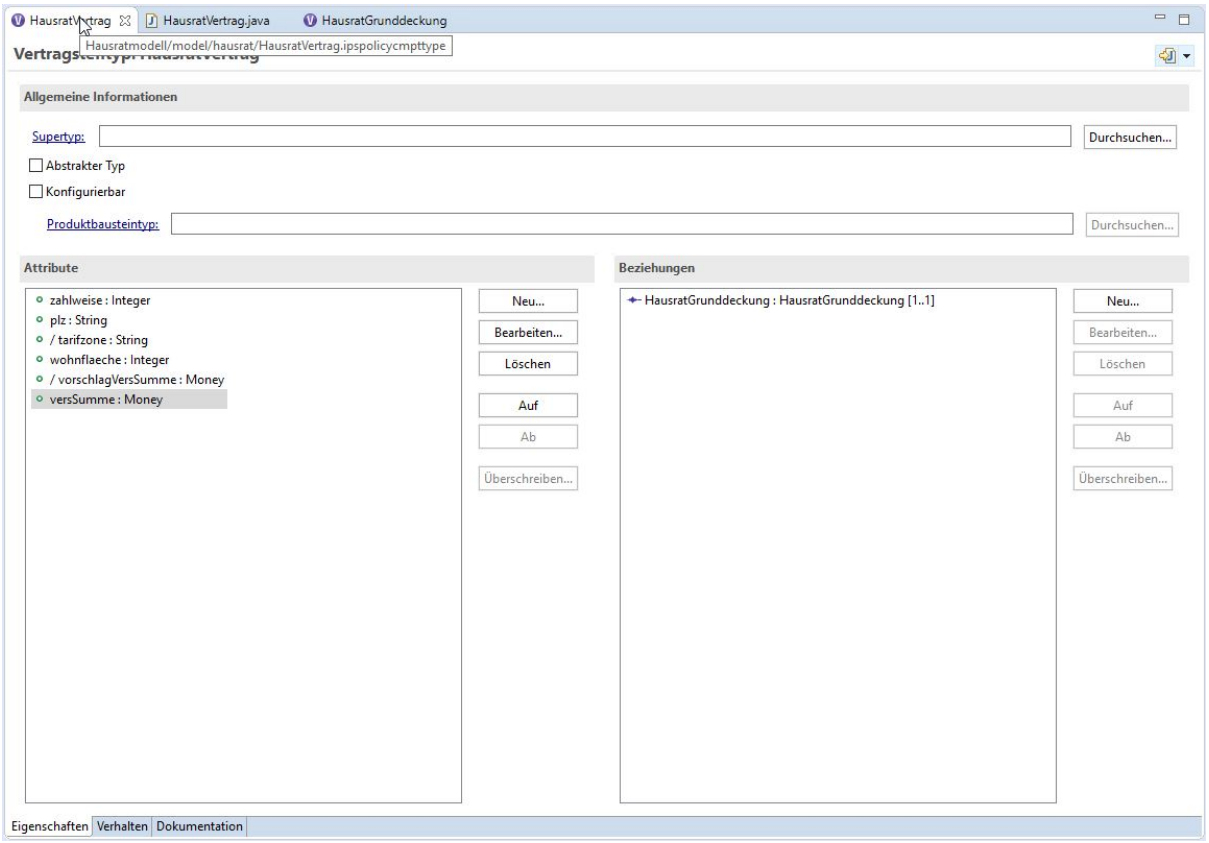

Figure 17. Klasse HausratVertrag

Die abgeleiteten Attribute werden UML-konform mit einem vorangestellten Schrägstrich angezeigt.

Öffnen Sie nun die Klasse „`HausratVertrag`“ im Java-Editor und implementieren die Gettermethoden für die beiden abgeleiteten Attribute `tarifzone` und `vorschlagVersSumme` wie folgt:

```
/**
 * Gibt den Wert des Attributs tarifzone zurueck.
 *
 * @restrainedmodifiable
 */
@IpsAttribute(name = "tarifzone", kind = AttributeKind.DERIVED_ON_THE_FLY,
valueSetKind = ValueSetKind.AllValues)
public String getTarifzone() {
    // begin-user-code
    // TODO: Wird spaeter anhand einer Tarifzonentabelle ermittelt
    return "I";
    // end-user-code
}

/**
 * Gibt den Wert des Attributs vorschlagVersSumme zurueck.
 *
 * @restrainedmodifiable
 */
@IpsAttribute(name = "vorschlagversSumme", kind =
AttributeKind.DERIVED_ON_THE_FLY, valueSetKind = ValueSetKind.AllValues)
public Money getVorschlagversSumme() {
    // begin-user-code
    // TODO: Der Multiplikator wird spaeter aus den Produktdaten ermittelt
    return Money.euro(650).multiply(wohnflaeche);
    // end-user-code
}
```

`@restrainedmodifiable` wird bei bestimmten Methoden (z.B. in generierten Testklassen oder Regeln) vom Generator anstelle von `@generated` erzeugt und weist darauf hin, dass der Entwickler eigenen Code hinzufügen kann. Der Abschnitt, in dem der eigene Code stehen darf, wird durch Kommentare gekennzeichnet. `@restrainedmodifiable` kann nur verwendet werden, wenn die Annotation vom Generator erzeugt wurde. Ein Ersetzen von `@generated` und Einfügen der entsprechenden Kommentarzeilen funktioniert nicht und wird vom Generator überschrieben.

Aufnahme von Produktaspekten ins Modell

Nun beschäftigen wir uns damit, wie Produktaspekte im Modell abgebildet werden. Bevor wir dies mit Faktor-IPS tun, diskutieren wir das Design auf Modellebene.

Schauen wir uns die bisher definierten Attribute unserer Klasse „HausratVertrag“ an und überlegen, was in einem Produkt konfigurierbar sein soll:

Table 3. Eigenschaften der Klasse „HausratVertrag“

Eigenschaften von HausratVertrag	Konfigurationsmöglichkeiten
`zahlweise`	Die im Vertrag erlaubten Zahlweisen. Der Vorgabewert für die Zahlweise bei Erzeugung eines neuen Vertrags.
`wohnflaeche`	Bereich (min, max), in dem die Wohnfläche liegen muss.
`vorschlagVersSumme`	Vorgeschlagener Wert für einen Quadratmeter Wohnfläche. Der Vorschlag für die komplette Versicherungssumme ergibt sich dann durch Multiplikation mit der Wohnfläche [9].
`versSumme`	Bereich, in dem die Versicherungssumme liegen muss.

[9] Alternativ könnten wir auch die Formel zur Berechnung des Vorschlags konfigurierbar machen. Zunächst beschränken wir uns aber auf den Vorschlag für den Quadratmeterwert.

Wir wollen zwei Hausratprodukte erstellen. „`HR-Optimal`“ soll einen umfangreichen Versicherungsschutz gewähren während „`HR-Kompakt`“ einen Basisschutz zu einem günstigen Beitrag bietet. Die folgende Tabelle zeigt die Eigenschaften der beiden Produkte bzgl. der oben aufgeführten Konfigurationsmöglichkeiten:

Konfigurationsmöglichkeit	HR-Kompakt	HR-Optimal
Vorgabewert Zahlweise	jährlich	jährlich
Erlaubte Zahlweisen	halbjährlich, jährlich	monatlich, vierteljährlich, halbjährlich, jährlich
Erlaubte Wohnfläche	0-1000 qm	0-2000 qm
Vorschlag Versicherungssumme pro qm Wohnfläche	600 Euro	900 Euro
Versicherungssumme	10 Tsd - 2 Mio Euro	10 Tsd - 5 Mio Euro

Wir bilden dies im Modell ab, indem wir eine Klasse „`HausratProdukt`“ einführen. Das Produkt enthält die Eigenschaften und Konfigurationsmöglichkeiten, die bei allen Hausratverträgen, die auf dem gleichen Produkt basieren, identisch sind. Die beiden Produkte „`HR-Optimal`“ und „`HR-Kompakt`“ sind Instanzen der Klasse „HausratProdukt“. Das folgende UML Diagramm zeigt das Modell:

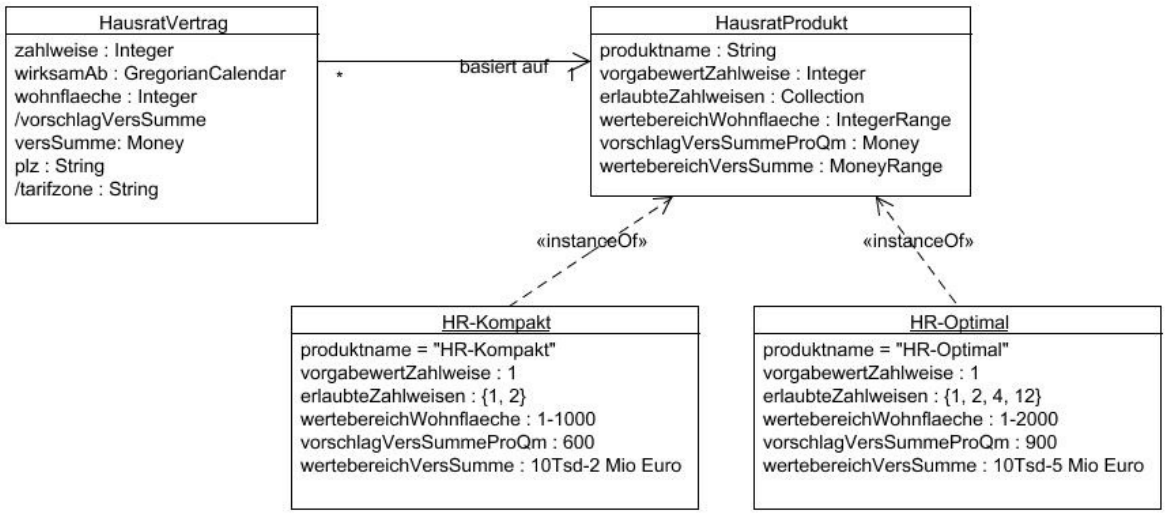

Figure 18. Hausratmodell mit Produktklassen

Erweitern wir unser Modell in Faktor-IPS um die Produktklassen. Als erstes definieren wir die Klasse „HausratProdukt“. Zum Erzeugen klicken Sie in der Toolbar auf den Button  . In dem Wizard geben Sie den Namen der neuen Klasse an („HausratProdukt“) und geben Sie im Feld *Policy Component Type (Vertragsteiltyp)* an, welche Klasse konfiguiert wird, in diesem Fall also „HausratVertrag“, danach den Button *Finish* drücken.

Es öffnet sich der Editor zur Bearbeitung von Produktklassen.

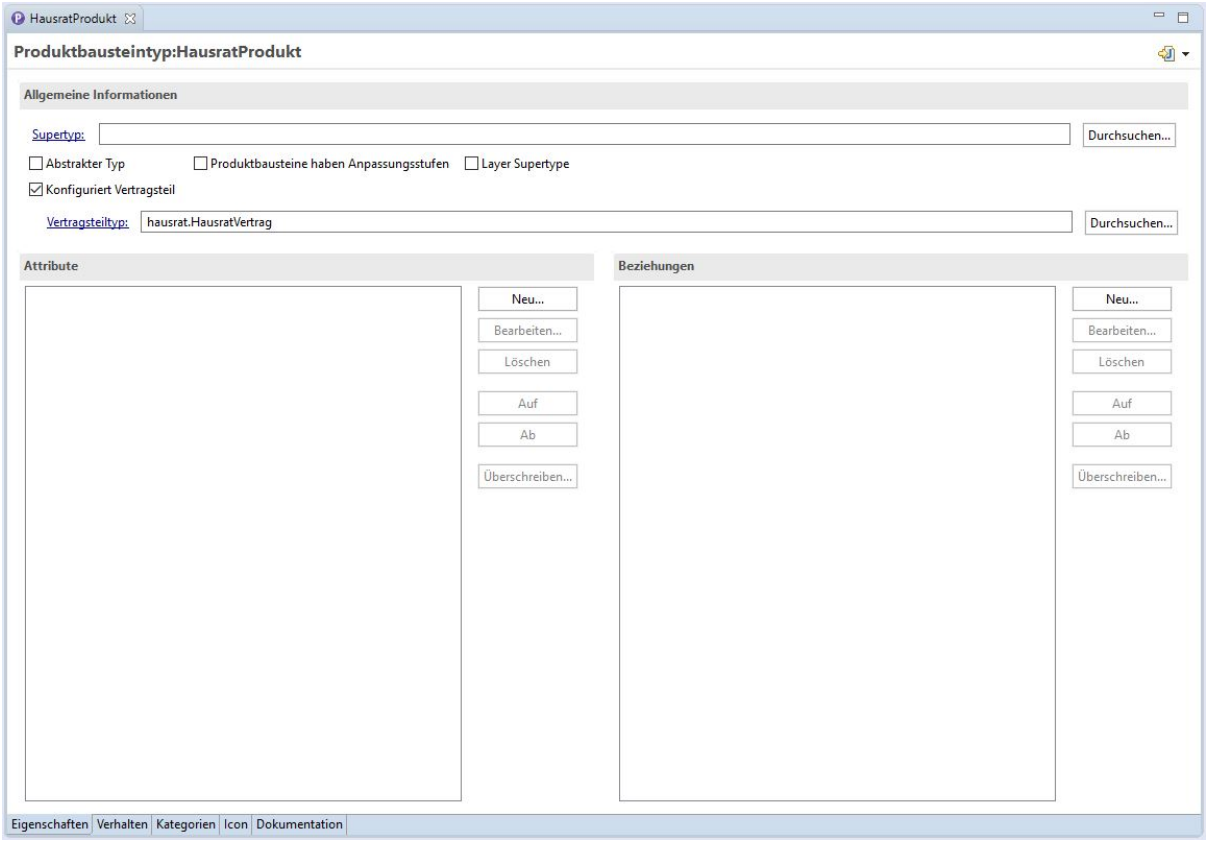

Figure 19. Editor für Produktklassen

Im Abschnitt *Allgemeine Informationen* sehen wir die gerade im Wizard eingegebene Information, dass die Klasse „HausratProdukt“ die Klasse „HausratVertrag“ konfiguriert. Ansonsten ist der Aufbau der ersten Seite des Editors analog zum Editor für Vertragsklassen [10].

[10] Sie können in den Preferences einstellen, ob Sie alle Informationen zu einer Klasse auf einer Seite oder auf zwei Seiten dargestellt bekommen möchten.

Gleichzeitig hat Faktor-IPS nun die Implementierungsklasse „HausratProdukt“ generiert.

Folgende Aspekte sollen in der Klasse „HausratProdukt“ konfigurierbar sein:

- der Name des Produktes
- die erlaubten Zahlweisen und der Vorbelegungswert für die Zahlweise.

Beginnen wir mit dem Produktnamen. Hierzu legen Sie ein neues Attribut `produktname` vom Datentyp *String* an. Dies geschieht analog zum Anlegen eines Attributes für Vertragsklassen. Die folgende Abbildung zeigt den entsprechenden Dialog:

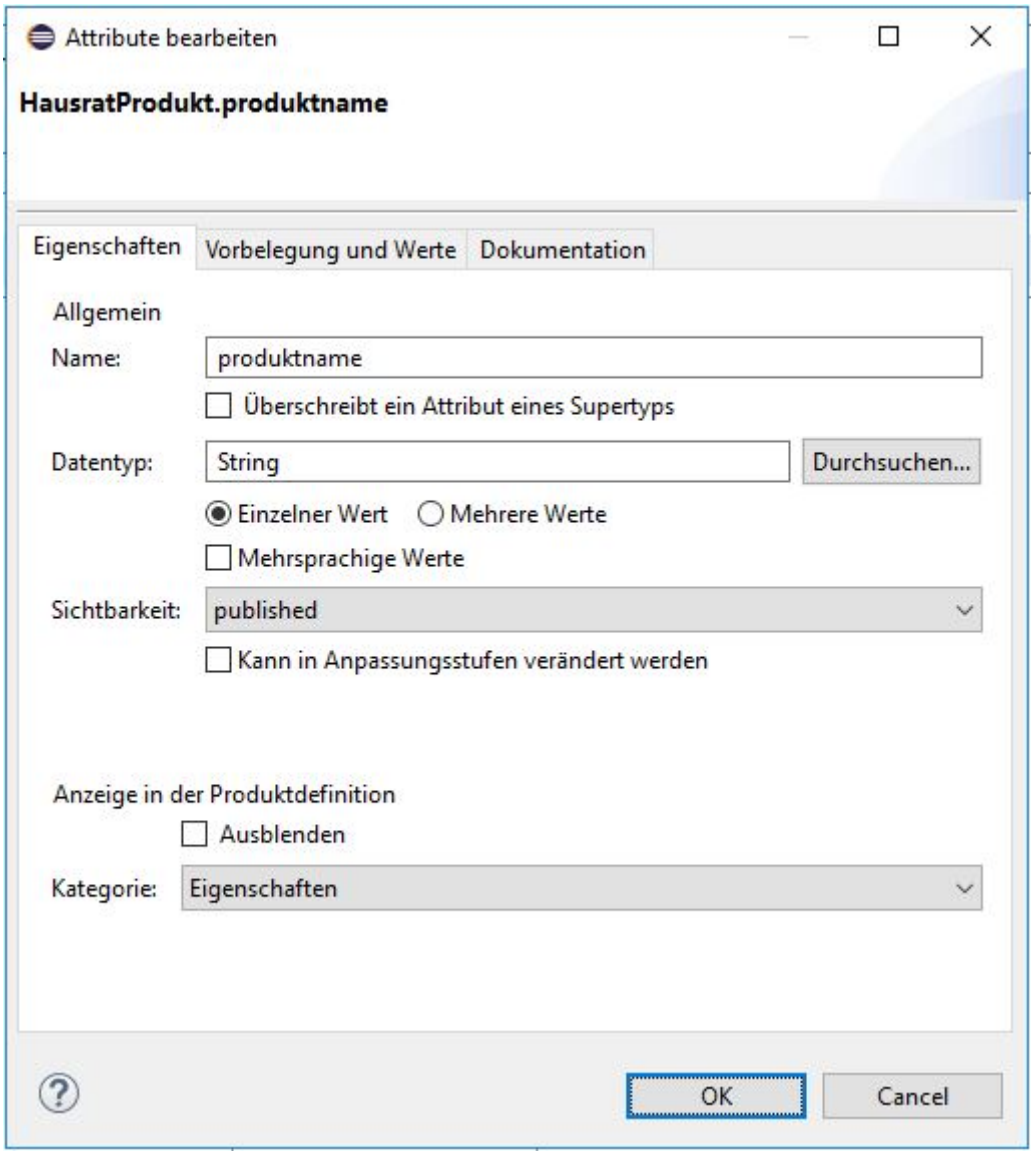

Figure 20. Dialog zum Editieren von Produktattributen

Nun konfigurieren wir die in einem Hausratvertrag erlaubten Zahlungsweisen und den Vorgabewert für die Zahlweise im Produkt. Hierzu öffnen wir zunächst den Editor für die Klasse „HausratVertrag“. In den Abschnitt *Allgemeine Informationen* hat der Wizard eingetragen, dass die Klasse „HausratVertrag“ durch die Klasse „HausratProdukt“ konfiguriert wird.

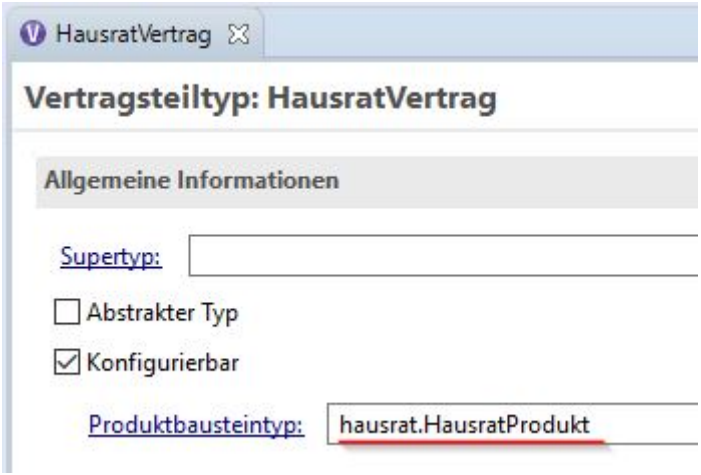

Figure 21. General Information Abschnitt im Editor für die Klasse Vertrag

Entsprechend ist der Vertragsteiltyp als konfigurierbar eingestellt. Öffnen Sie das Attribut `zahlweise` im Bereich *Attribute*. Im Dialogfenster unter dem Bereich *Konfiguration* können Sie dieses Attribut editieren.

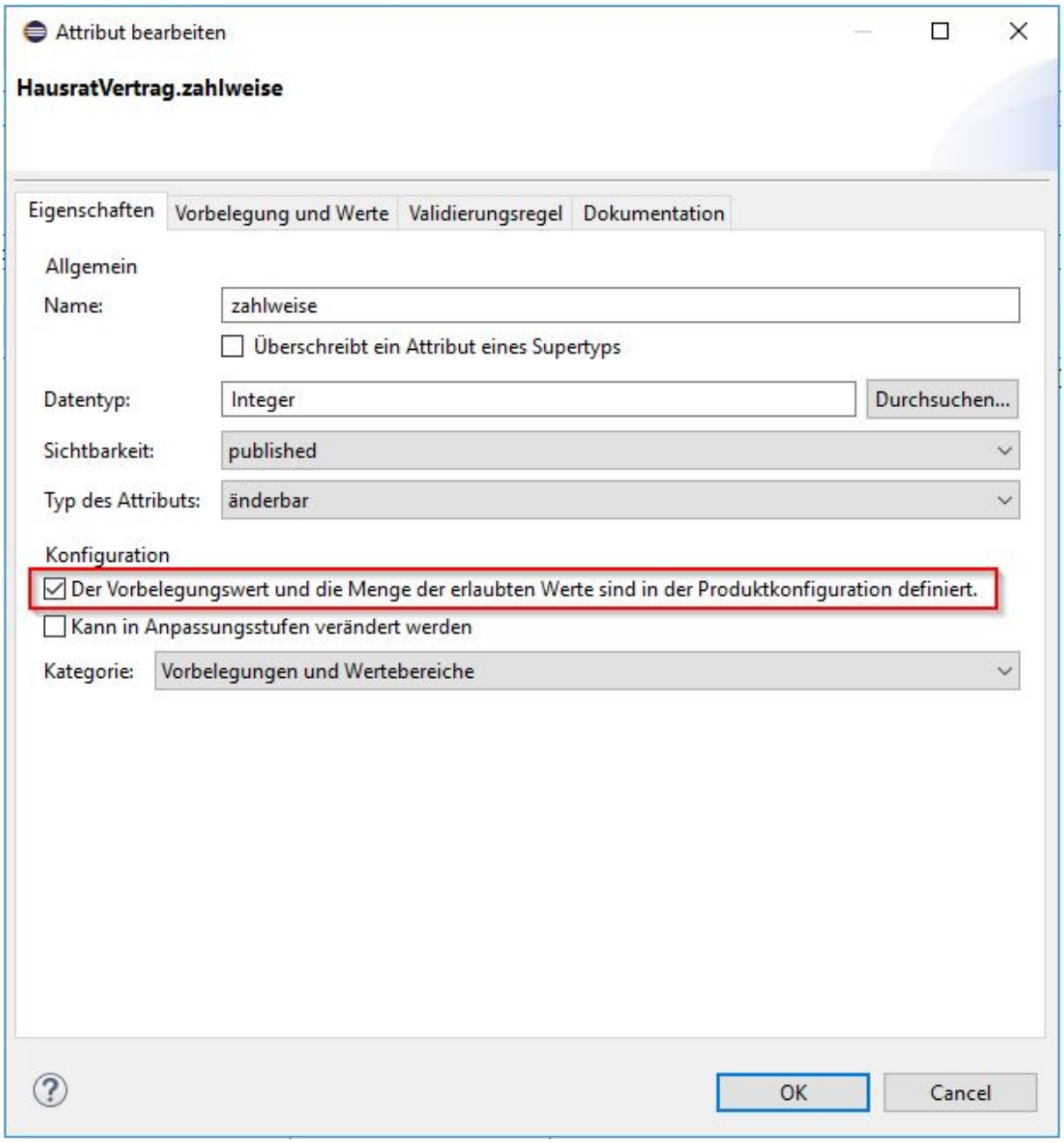

Figure 22. Dialog für ein Vertragsklassenattribut mit Konfigurationsmöglichkeit

Um die erlaubten Zahlweisen und den Vorgabewert für die Zahlweisen im Produkt definieren zu können, müssen Sie die entsprechende Checkbox anhaken. Button *OK* drücken und anschließend Änderungen speichern.

Werfen wir nun einen Blick in den Sourcecode. In der Klasse `„HausratProdukt“` gibt es jeweils eine Methode, um den Produktnamen, den Vorgabewert für die Zahlweise und die erlaubten Werte für die Zahlweise abzufragen.


```
/**
 * Gibt den Wert der Eigenschaft produktname zurueck.
 *
 * @generated
 */
@IpsAttribute(name = "produktname", kind = AttributeKind.CONSTANT,
valueSetKind = ValueSetKind.AllValues)
public String getProduktname() {
    return produktname;
}

/**
 * Gibt den Defaultwert fuer die Eigenschaft zahlweise zurueck.
 *
 * @generated
 */
@IpsDefaultValue("zahlweise")
public Integer getDefaultValueZahlweise() {
    return defaultValueZahlweise;
}

/**
 * Gibt den erlaubten Wertebereich fuer das Attribut zahlweise zurueck.
 *
 * @generated
 */
@IpsAllowedValues("zahlweise")
public OrderedValueSet<Integer>
getAllowedValuesForZahlweise(IValidationContext context) {
    return allowedValuesForZahlweise;
}
```

In der Klasse „HausratVertrag“ gibt es Methoden, um auf das „HausratProdukt“ zuzugreifen.

```
/**
 * Gibt HausratProdukt zurueck, welches HausratVertrag konfiguriert.
 *
 * @generated
 */
public HausratProdukt getHausratProdukt() {
    return (HausratProdukt) getProductComponent();
}

/**
 * Setzt neuen HausratProdukt.
 *
 * @param hausratProdukt
 *            Der neue HausratProdukt.
 * @param initPropertiesWithConfiguratedDefaults
 *            <code>true</code> falls die Eigenschaften mit den Defaultwerten
aus
 *            HausratProdukt belegt werden sollen.
 *
 * @generated
 */
public void setHausratProdukt(HausratProdukt hausratProdukt, boolean
initPropertiesWithConfiguratedDefaults) {
    setProductComponent(hausratProdukt);
    if (initPropertiesWithConfiguratedDefaults) {
        initialize();
    }
}
```

Markieren Sie abschließend noch die Attribute `wohnflaeche` und `versSumme` analog zur `zahlweise` als konfigurierbar.

Nun überarbeiten wir die Berechnung des Vorschlags der Versicherungssumme. Im Kapitel „Erweiterung des Hausratmodells“ hatten wir die Methode `getVorschlagVersSumme()` der Klasse „`HausratVertrag`“ bisher wie folgt implementiert:


```
/**
 * Gibt den Wert des Attributs vorschlagversSumme zurueck.
 *
 * @restrainedmodifiable
 */
@IpsAttribute(name = "vorschlagversSumme", kind =
AttributeKind.DERIVED_ON_THE_FLY, valueSetKind = ValueSetKind.AllValues)
public Money getVorschlagversSumme() {
    // begin-user-code
    // TODO: Der Multiplikator wird spaeter aus den Produktdaten ermittelt.
    return Money.euro(650).multiply(wohnflaeche);
    // end-user-code
}
```

Nun wollen wir die Höhe des Multiplikators im Hausratprodukt konfigurieren können. Hierzu legen Sie zunächst an der Klasse „HausratProdukt“ ein neues Attribut vorschlagVersSummeProQm vom Datentyp *Money* an. Dies ist der Vorschlagswert für einen Quadratmeter Wohnfläche. Nach dem Speichern der Klasse HausratProdukt hat Faktor -IPS an der Klasse „HausratProdukt“ die entsprechende Gettermethode getVorschlagVersSummeProQm() generiert. Diese nutzen wir nun in der Berechnung des Vorschlags für die Versicherungssumme. Passen Sie den Sourcecode in der Klasse „HausratVertrag“ dazu wie folgt an:

```
/**
 * Gibt den Wert des Attributs vorschlagversSumme zurueck.
 *
 * @restrainedmodifiable
 */
@IpsAttribute(name = "vorschlagversSumme", kind =
AttributeKind.DERIVED_ON_THE_FLY, valueSetKind = ValueSetKind.AllValues)
public Money getVorschlagversSumme() {
    // begin-user-code
    HausratProdukt prod = getHausratProdukt();
    if (prod == null) {
        return Money.NULL;
    }
    return prod.getVorschlagVersSummeProQm().multiply(wohnflaeche);
    // end-user-code
}
```

Definieren wir nun noch die Produktseite des Modells für die Grunddeckung. Dazu markieren wir die Klasse „HausratGrunddeckung“ als Konfigurierbar. Die neu anzulegende Produktbausteinklasse nennen wir „HausratGrunddeckungstyp“. Wir definieren zunächst nur ein Attribut bezeichnung mit Datentyp *String* an dieser Klasse.

Figure 23. Konfiguration der HausratGrunddeckung

Zum Abschluss dieses Kapitels beschäftigen wir uns noch mit den Beziehungen zwischen den Klassen der Produktseite. Wir wollen hierüber abbilden, welche (Hausrat-)Deckungstypen in welchen (Hausrat-)Produkten enthalten sind. Das folgende UML Diagramm zeigt das Modell:

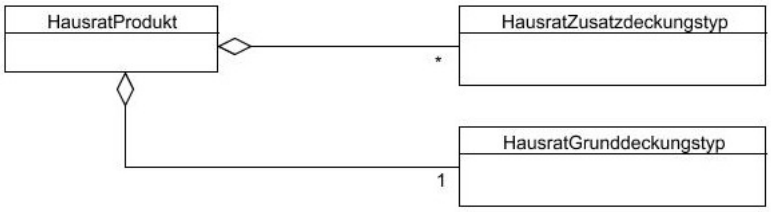

Figure 24. Modell der Produktkonfigurationsklassen

Das Hausratprodukt verwendet genau einen Grunddeckungstyp und beliebig viele Zusatzdeckungstypen. Andersherum kann ein Grunddeckungstyp bzw. ein Zusatzdeckungstyp in beliebig vielen Hausratprodukten verwendet werden. Die primäre Navigation ist immer vom Hausratprodukt zum Grund- bzw. Zusatzdeckungstyp, nicht umgekehrt, da ein Deckungstyp immer unabhängig von den Produkten sein sollte, die ihn verwenden.

Definieren wir die Beziehung zwischen `„HausratProdukt“` und `„HausratGrunddeckungstyp“` also nun in Faktor-IPS. Öffnen Sie hierzu zunächst den Editor für die Klasse `„HausratProdukt“` und legen im Bereich *Beziehungen* durch klicken auf *New* eine neue Beziehung an. Es öffnet sich der folgende Dialog, in den Sie die Daten wie hier abgebildet eintragen. Achten Sie darauf die maximale Kardinalität auf eins zu setzen. Den Zusatzdeckungstypen legen wir im Teil 2 des Tutorials an.

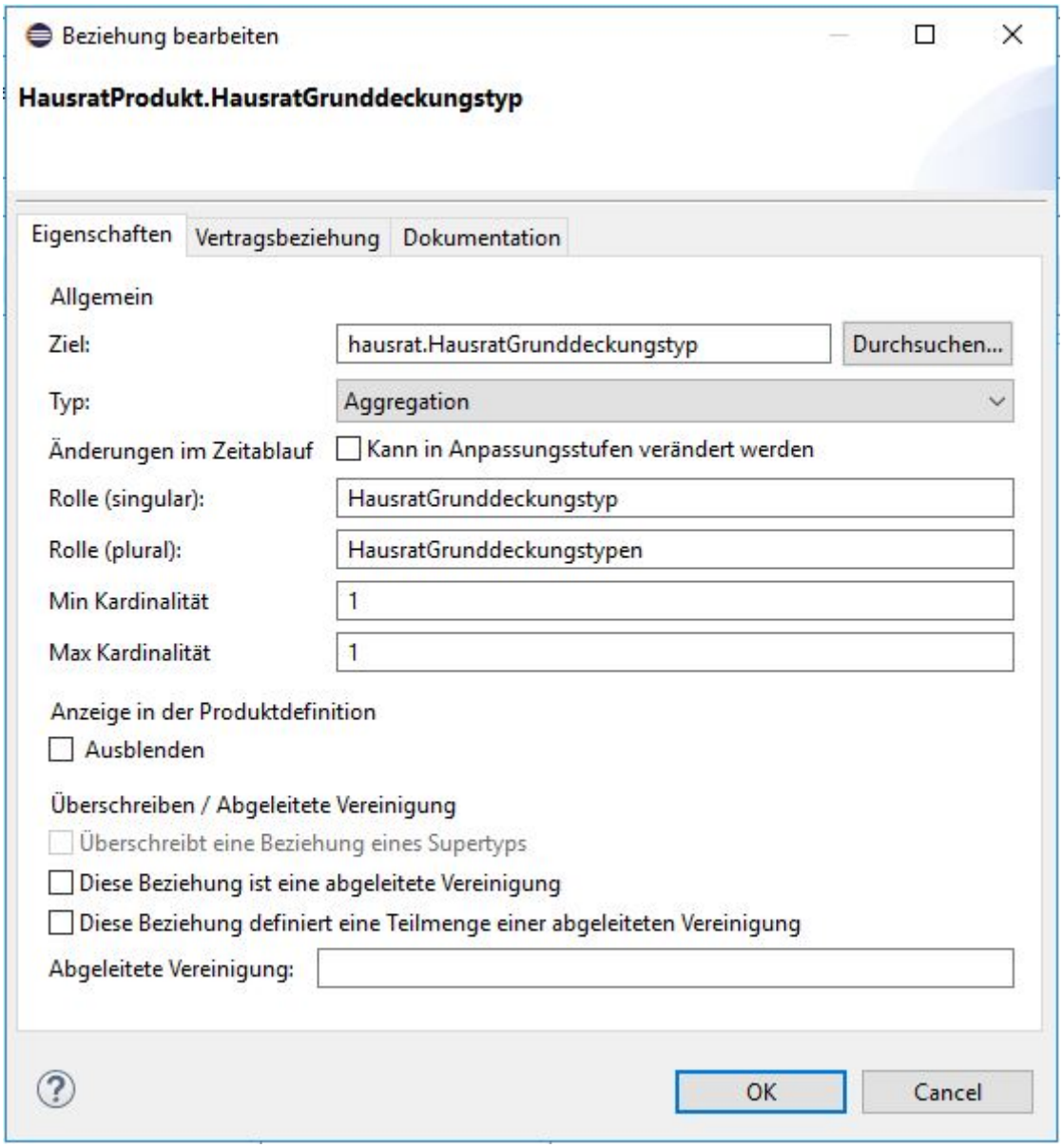

Figure 25. Dialog für Beziehungen zwischen Produktklassen

Definition der Hausratprodukte

In diesem Kapitel definieren wir nun die Produkte „HR-Optimal“ und „HR-Kompakt“ in Faktor- IPS. Hierzu wird die speziell für den Fachbereich entwickelte *Produktdefinitionsperspektive* verwendet.

Als erstes richten Sie ein neues Projekt mit dem Namen „Hausratprodukte“ und dem Sourceverzeichnis „produktdaten“ ein. Dazu wieder mit Hilfe des Archetype über die Kommandozeile ein neues Maven-Projekt mit der Faktor-IPS Nature erzeugen.

```
mvn archetype:generate -DarchetypeGroupId=org.faktorips
-DarchetypeArtifactId=faktorips-maven-archetype -DarchetypeVersion=21.6.0
-DgroupId=org.faktorips.tutorial -DartifactId=Hausratprodukte -Dversion=1.0
-Dpackage=org.faktorips.tutorial.produktdaten -DJavaVersion=1.8 -DIPS-Language=de
```

```
-DIPS-IsModelProject=false -DIPS-IsProductDefinitionProject=true -DIPS
-SourceFolder=produktdaten -DIPS-RuntimeIdPrefix=hausrat. -DIPS
-ConfigureIPSBuild=true
```

Als Typ wählen Sie diesmal *Produktdefinitions-Project* (`"-DIPS-IsProductDefinitionProject=true"`), als Packagename `„org.faktorips.tutorial.produktdaten“` und als Runtime-ID Prefix `„hausrat.“`. Achten Sie darauf, dass der Prefix mit einem Punkt (.) endet. Faktor-IPS erzeugt für jeden neuen Produktbaustein eine Id, mit der der Baustein zur Laufzeit identifiziert wird. Standardmäßig setzt sich diese Runtime-ID aus dem Prefix gefolgt von dem (unqualifizierten) Namen zusammen [11]. Der qualifizierte Name eines Bausteines wird nicht zur Identifikation zur Laufzeit verwendet, da die Packagestruktur zur Organisation der Produktdaten zur Entwicklungszeit dient. Auf diese Weise können die Produktdaten umstrukturiert (refactored) werden, ohne dass dies Auswirkungen auf die nutzenden operativen Systeme hat.

[11] Für die Implementierung eigener Verfahren zur Vergabe der Runtime-ID wird ein entsprechender Extension Point bereitgestellt werden.

Das Projekt muss anschließend über *File* ► *Import* ► *Maven* ► *Existing Maven Projects* in den Eclipse Workspace importiert werden.

Faktor-IPS generiert Java Sourcefiles und kopiert XML-Dateien, die zu 100% generiert werden, in das Verzeichnis `"src/main/resources"`. Der Inhalt des Verzeichnisses kann also jederzeit gelöscht und neu erzeugt werden. Da in diesem Ordner im Verlauf dieses Tutorials auch Source-Code für Formeln generiert wird, muss Maven angewiesen werden, diesen auch im `"src/main/resources"`-Ordner zu bauen. Dazu muss in der `pom.xml`-Datei unter `<plugins>` der folgende Code ergänzt werden:


```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>build-helper-maven-plugin</artifactId>
    <executions>
        <execution>
            <id>add-source</id>
            <phase>generate-sources</phase>
            <goals>
                <goal>add-source</goal>
            </goals>
            <configuration>
                <sources>
                    <source>${project.basedir}/src/main/resources</source>
                </sources>
            </configuration>
        </execution>
    </executions>
</plugin>
```

Die Verwaltung der Produktdaten in einem eigenen Projekt erfolgt vor dem Hintergrund, dass die Verantwortung für die Produktdaten bei anderen Personen liegt und sie auch einen anderen Releasezyklus haben können. So könnte die Fachabteilung zum Beispiel ein neues Produkt „HR-Flexibel“ erstellen und freigeben, ohne dass das Modell geändert wird. Damit in dem neuen Projekt auf die Klassen des Hausratmodells zugegriffen werden kann, muss in Faktor -IPS im Produktdatenprojekt eine Referenz auf das Hausratmodell-Projekt definiert werden. Das funktioniert in Faktor-IPS analog zur Definition des Build Path in Java. Der Dialog zur Definition des Faktor-IPS Build Path ist auch genauso aufgebaut wie der entsprechende Dialog für den *Java Build Path* und kann über die *Projekteigenschaften* aufgerufen werden.

Um eine Referenz auf das Modell anzulegen, gehen Sie wie folgt vor:

- Rechtsklick auf das Projekt „Hausratprodukte“
- Im Dialogfenster *Properties* auswählen
- Danach *Faktor-IPS Build Path* und Ordner *Projects* auswählen
- Button *Hinzufügen* anklicken und *Hausratmodell* auswählen. Abschließend *OK* drücken

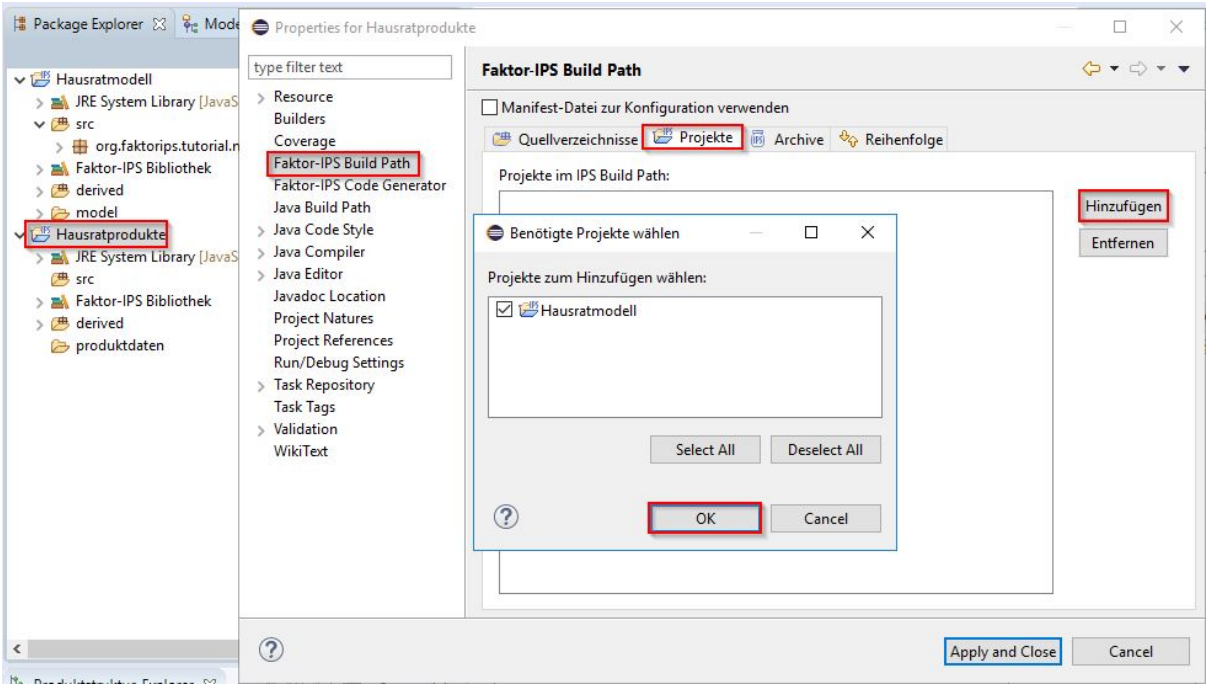

Figure 26. Referenz auf das Modell anlegen

Die folgende Abbildung zeigt den Dialog für den *Faktor-IPS Build Path* des Produktdatenprojektes mit Referenz auf das Modellprojekt.

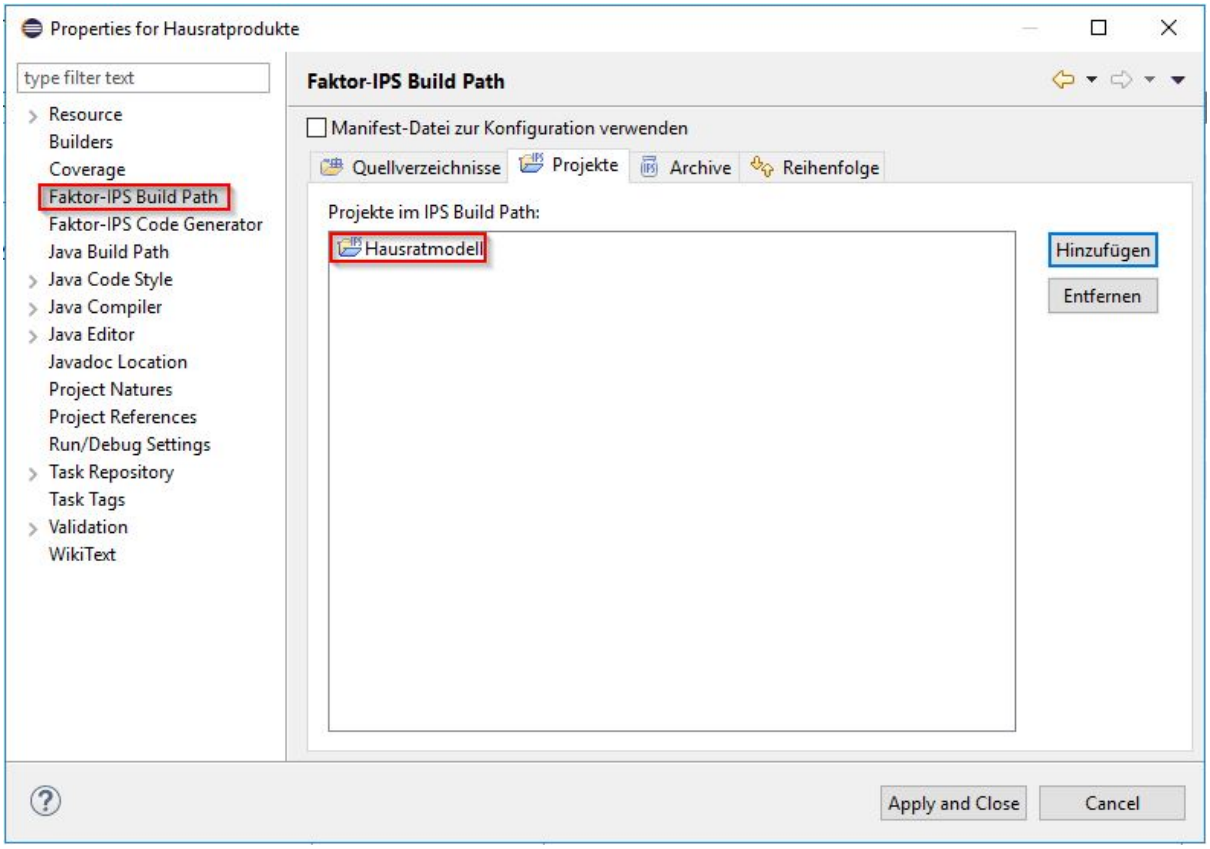

Figure 27. Dialog für den Faktor-IPS Build Path

Gespeichert wird der Build Path in der „.ipsproject“ Datei im XML-Element „IpsObjectPath“. Nach dem Anlegen der Referenz enthält das Element den folgenden neuen Entry.

<Entry reexported="true" referencedIpsProject="Hausratmodell" type="project"/>

Neben Sourceverzeichnissen und Projektreferenzen unterstützt Faktor-IPS (wieder analog zu Java) auch Archive/Bibliotheken (z.B. Maven) im Build Path. Erstellt werden können Faktor-IPS Archive analog zu JARs über einen Export-Wizard.

Damit auch die Javaklassen in dem Projekt verfügbar sind, muss die `pom.xml`-Datei des Projekts `„Hausratprodukte“` um eine Abhängigkeit zum Projekt `„Hausratmodell“` ergänzt werden. Dazu fügen Sie einfach unter `<dependencies>` in der `pom.xml`-Datei den folgenden Eintrag hinzu:

```
<dependency>
    <groupId>org.faktorips.tutorial</groupId>
    <artifactId>Hausratmodell</artifactId>
    <version>1.0</version>
</dependency>
```

Anschließend müssen noch die JUnit 5 Bibliotheken zu dem Projekt hinzugefügt werden, da wir später eine Testklasse schreiben werden. Dazu ergänzen Sie in der `pom.xml`-Datei die folgenden Abhängigkeiten:

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.7.2</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.platform</groupId>
    <artifactId>junit-platform-runner</artifactId>
    <version>1.7.2</version>
    <scope>test</scope>
</dependency>
```

Die Tutorial-Projekte sollen mit Java 8 erstellt werden. Falls Java 11 genutzt wurde, muss die folgende Abhängigkeit ergänzt werden:

```
<dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.1</version>
</dependency>
```

Damit die Änderungen in der `pom.xml`-Datei wirksam werden, führen Sie einen Rechtklick auf das Projekt „Hausratprodukte“ aus und wählen *Maven* ► *Update Project* ► *Ok*.

Öffnen Sie nun zunächst die Produktdefinitionsperspektive über *Window* ► *Perspective* ► *Open Perspective* ► *Other* ► *Produktdefintion* auswählen [12]. Falls Sie noch Editoren geöffnet haben, schließen Sie diese jetzt, um die Sichtweise der Fachabteilung auf das System zu haben. Damit Sie im *Problems-View* ausschließlich die Marker von Faktor-IPS sehen (und nicht auch Java-Marker u.a.) müssen Sie im *Problems-View* den Faktor-IPS Filter ein- und alle anderen Filter (standardmäßig mindestens der Defaultfilter) ausschalten.

[12] Für die Verwendung von Faktor-IPS durch die Fachabteilung gibt es auch eine eigene Installation (in Eclipse Terminologie: ein eigenes Produkt), bei der ausschließlich die Produktdefinitionsperspektive verfügbar ist.

Zunächst legen wir zwei IPS-Packages an, eines für die Produkte und eines für die Deckungen. Dies geschieht wie in der Java Perspektive entweder über das Kontextmenü oder die Toolbar (als Quellordner „Hausratprodukte“ wählen).

Bemerkung: Sie können in dem Projekt beliebig viele Verzeichnisse anlegen. Zum Beispiel ein doc- Verzeichnis zur Verwaltung von Produktdokumenten.

Als erstes legen wir jetzt das Produkt „HR-Optimal“ an. Markieren Sie dazu das gerade angelegte Package „produkte“ und klicken dann in der Toolbar  an . Es öffnet sich der Wizard zum Erzeugen eines neuen Produktbausteins. Der Wizard bietet Ihnen nun die im Modell verfügbaren Produktklassen zur Auswahl, zu denen Sie Produktbausteine erstellen können. Wählen Sie `HausratProdukt`.

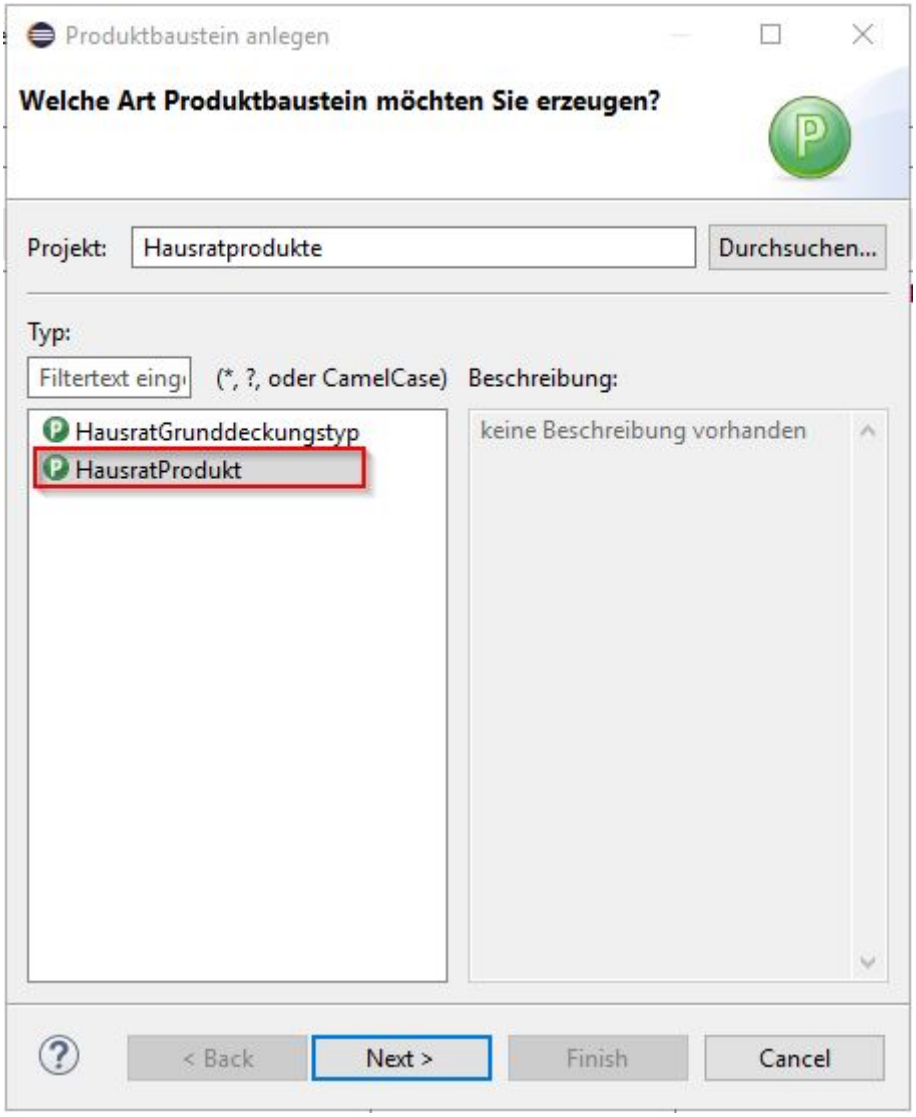

Figure 28. Auswahl der Produktbaustein-Klasse

Falls Sie keine Produktklasse finden, fehlt noch die Referenz auf das Hausratmodell-Projekt im *Faktor-IPS Build Path* (s.o.). Gehen Sie mit Next zur nächsten Seite des Wizards.

Hier geben Sie den Namen des Bausteins ein. Wenn Sie nun *Finish* drücken wird der Produktbaustein im Dateisystem angelegt.

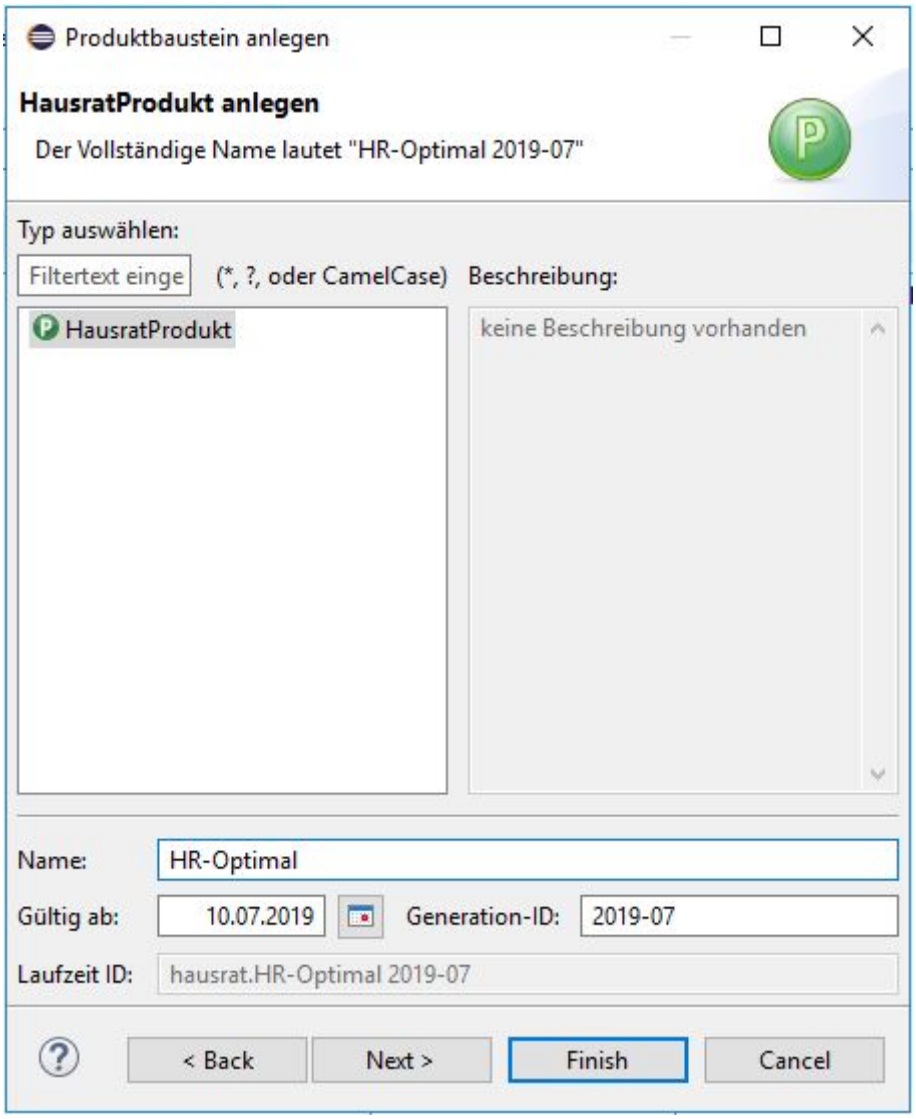

Figure 29. Anlegen eines neuen Produkts

Mit Doppelklick auf den Produktbaustein im Produktdefinitions-Explorer öffnen Sie den Editor für den Baustein.

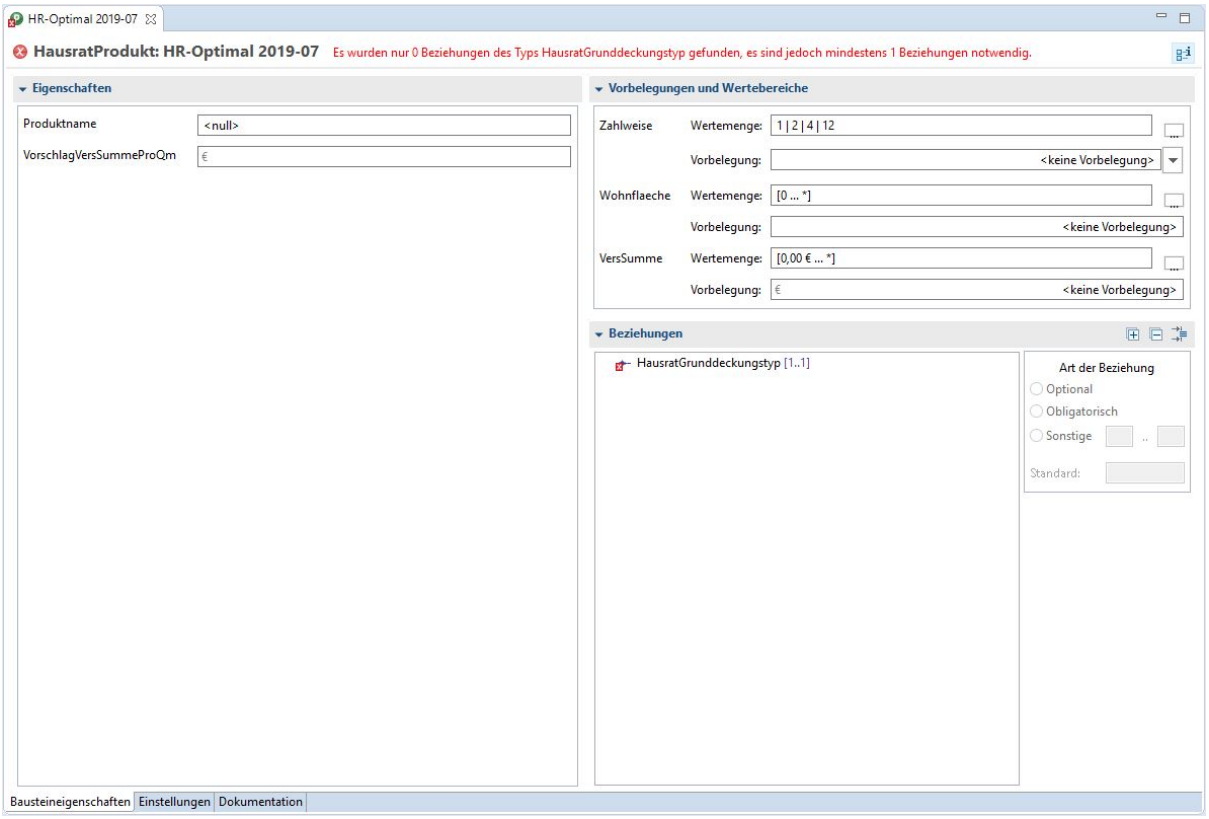

Figure 30. Produktbaustein-Editor für HR-Optimal

Die erste Seite des Editors zeigt:

- *Eigenschaften*
  Enthält die Eigenschaften der Anpassungsstufe. Hier ist jedes in der Produktklasse definierte Attribut aufgelistet.
- *Vorbelegungen und Wertebereiche*
  Enthält die Vorbelegungswerte und Wertebereiche für die Vertragseigenschaften.
- *Beziehungen*
  Enthält die verwendeten anderen Produktbausteine.

Geben Sie Daten für das Produkt HR-Optimal entsprechend der folgenden Tabelle ein:

Konfigurationsmöglichkeit	**HR-Optimal**
Produktname	Hausrat Optimal
Vorschlag Versicherungssumme pro qm Wohnfläche	900 EUR
Vorgabewert Zahlweise	1 (jährlich)
Erlaubte Zahlweisen	1, 2, 4, 12
Vorgabewert Wohnflaeche	<keine Vorbelegung>
Erlaubte Wohnflaeche	0-2000

Konfigurationsmöglichkeit	HR-Optimal
Vorgabewert Versicherungssumme	<keine Vorbelegung>
Versicherungssumme	10000 EUR - 5000000 EUR

Beim Anlegen des Produktbausteins haben Sie folgende Fehlermeldung bekommen:

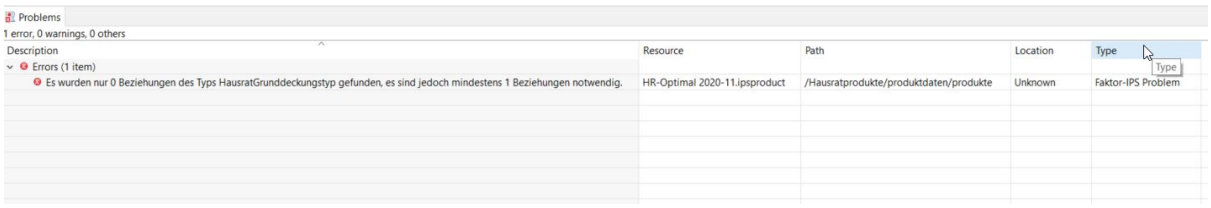

Figure 31. Fehlermeldung beim Anlegen des Produktbausteins

Um diese zu beheben, legen wir nun den Grunddeckungstyp für das Produkt an. Markieren Sie hierzu den Ordner Deckungen und legen einen neuen Produktbaustein mit Namen „HRD-Grunddeckung-Optimal“ an basierend auf der Klasse „HausratGrunddeckungstyp “.

Nun müssen wir noch den Deckungstyp dem Produkt „HR-Optimal“ zuordnen. Dies kann man bequem per Drag&Drop aus dem Produktdefinitions-Explorer erledigen. Öffnen Sie das Produkt „HR-Optimal“. Ziehen Sie die „HRD-Grunddeckung-Optimal“ aus dem Explorer auf den Knoten HausratGrundeckungstyp im Bereich *Beziehungen*.

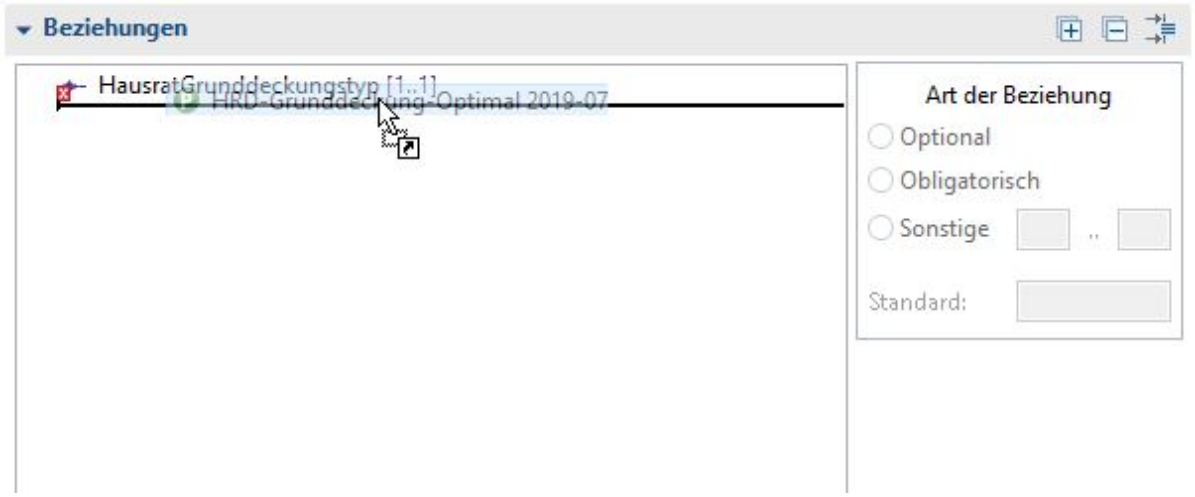

Figure 32. Zuordnung von Deckungstypen zu einem Produkt

Unter „Art der Beziehung“ muss *Obligatorisch* gewählt werden, da es sich um eine 1..1(1) Relation handelt. (Grund dafür ist die definierte Beziehung zwischen „HausratProdukt“ und „HausratGrunddeckungstyp“. *Siehe Abbildung 25*)

Im nächsten Schritt legen wir das Produkt „HR-Kompakt“ inklusive der Grunddeckung „HRD-GrunddeckungKompakt“ an. Dies können Sie analog zum Produkt „HR-Optimal“ machen. Alternativ können Sie einen Kopierassistenten verwenden, mit dem Sie einen Produktbaustein inklusive aller verwendeter Bausteine kopieren können. Wenn Sie dies

ausprobieren möchten, markieren Sie das Produkt „HR-Optimal“ im Produktdefinitions-Explorer und wählen im Kontextmenü *New* ► *Copy Product* ...

Im geöffneten Dialog geben Sie als Search Pattern (Suchen nach) „Optimal“ und als *Replace Pattern* (*Ersetzen durch*) „Kompakt“ ein, klicken Next und dann Finish. Faktor-IPS legt die Bausteine „HR-Kompakt“ und „HRD-Grunddeckung“ neu an. Öffnen Sie „HR-Kompakt“ und geben Sie Daten entsprechend der nachfolgenden Tabelle ein:

Konfigurationsmöglichkeit	**HR-Kompakt**
Produktname	Hausrat Kompakt
Vorschlag Versicherungssumme pro qm Wohnfläche	600 EUR
Vorgabewert Zahlweise	jährlich
Erlaubte Zahlweisen	halbjährlich, jährlich
Vorgabewert Wohnflaeche	<keine Vorbelegung>
Erlaubte Wohnflaeche	0-1000 qm
Vorgabewert Versicherungssumme	<keine Vorbelegung>
Versicherungssumme	10000 EUR - 2000000 EUR

Damit ist die Definition der beiden Produkte zunächst abgeschlossen. Im *Produktdefinitions-Explorer* sollten Sie folgendes sehen.

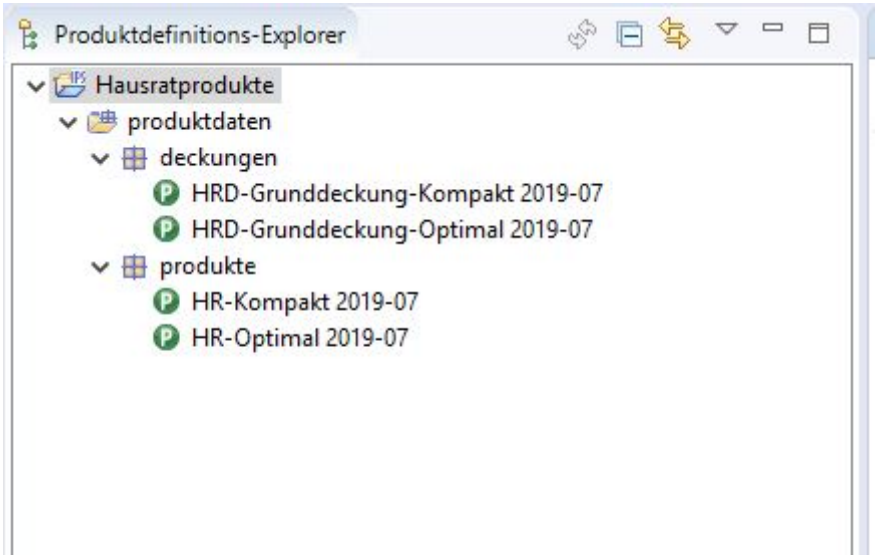

Figure 33. Darstellung der Produkte im Produktdefinitions-Explorer

Neben dem *Produktdefinitions-Explorer* stehen Ihnen zwei weitere Werkzeuge zur Analyse der Produktdefinition zur Verfügung. Die Struktur eines Produktes können Sie sich mit *Show Structure* im Kontextmenü anzeigen lassen. Die unterschiedliche Verwendung eines Bausteins mit *Search References*. Darüber hinaus können Sie die Reihenfolge der Pakete über den Menüpunkt *Edit Sort Order* frei festlegen.

Rechts neben dem Editor zur Eingabe der Produktdaten befindet sich der *Model Description View*. Dieser zeigt passend zum in Bearbeitung befindlichen Produktbaustein die Dokumentation der zugehörigen Produktklasse. Wenn Sie die einmal ausprobieren wollen, dokumentieren Sie z.B. das Attribut `produktname` im Modell, schließen Sie den Bausteineditor und öffnen ihn erneut.

Zugriff auf Produktinformationen zur Laufzeit

Nachdem wir die Produktdaten erfasst haben, beschäftigen wir uns nun damit, wie man zur Laufzeit (in einer Anwendung/einem Testfall) auf diese zugreift. Hierzu werden wir einen *JUnit-Test* schreiben, den wir im Laufe des Tutorials weiter ausbauen.

Zum Zugriff auf Produktdaten stellt Faktor-IPS das Interface `„IRuntimeRepository“` bereit. Die Implementierung `ClassloaderRuntimeRepository` erlaubt den Zugriff auf die mit Faktor-IPS erfassten Produktdaten und lädt die Daten über einen Classloader. Damit dies möglich ist, macht Faktor-IPS zwei Dinge:

1. Die Dateien, die die Produktinformationen enthalten, werden in den Java Sourcefolder mit dem Namen `„derived“` kopiert. Damit sind diese Dateien im Build Path des Projektes enthalten und können über den Classloader geladen werden.
2. Welche Daten sich im ClassloaderRuntimeRepository befinden, ist in einem Inhaltsverzeichnis vermerkt. Dieses Inhaltsverzeichnis (Englisch: table of contents, toc) wird von Faktor-IPS ebenfalls in eine Datei generiert, die als Toc-File bezeichnet wird. Die Datei heißt standardmäßig `„faktorips-repository-toc.xml“`. Die Namen lassen sich in der „.ipsproject“ Datei im Abschnitt IpsObjectPath konfigurieren.

Ein ClassloaderRuntimeRepository wird über die statische `create`(...)-Methode der Klasse erzeugt. Als Parameter wird der Pfad zum Toc-File übergeben. Das Toc-File wird direkt beim Erzeugen des Repositorys über `Classloader.getResourceAsStream`() gelesen. Alle weiteren Daten werden erst (wiederum über den Classloader) geladen, wenn auf sie zugegriffen wird.

> *Das Laden der Daten über den Classloader hat im Gegensatz zum Laden aus dem Filesystem den großen Vorteil, dass es völlig plattformunabhängig ist. So kann der Programmcode z.B. ohne Änderungen auf z/OS laufen.*

Einen Produktbaustein kann man über die Methode `getProductComponent`(...) erhalten. Als Parameter übergibt man die Runtime-ID des Bausteins. Da das Interface `„IRuntimeRepository“` unabhängig vom konkreten Modell (in unserem Fall also dem `Hausratmodell`) ist, muss man das Ergebnis noch auf die konkrete Produktklasse casten.

Probieren wir dies doch einmal in einem JUnit Testfall aus. Legen Sie hierzu zunächst im Projekt Hausratprodukte einen neuen Java Sourcefolder „test“ an. Am einfachsten geht

dies, indem Sie im `Package-Explorer` das Projekt markieren und im Kontextmenü *Build Path* ▸ *New Source Folder*...aufrufen.

Danach markieren Sie den neuen Sourcefolder und legen einen JUnit Testfall an, indem Sie in der Toolbar auf den Pfeil neben dem Icon

klicken und dann *JUnit Test Case* auswählen. In dem Dialog geben Sie als Namen für die Testfallklasse „TutorialTest“ ein und haken an, dass auch die `setUp()` Methode generiert werden soll. Die Warnung, dass die Verwendung des Defaultpackages nicht empfohlen wird, ignorieren wir in dem Tutorial. Für unseren Test verwenden wir die JUnit 5 `JUnit Jupiter`. Beim Beenden des Wizards werden wir gefragt, ob wir die JUnit Library zum Build Path des Projektes hinzufügen wollen. Klicken Sie in diesem Dialog auf *OK*. Der nächste Kasten enthält den Sourcecode der Testfallklasse.

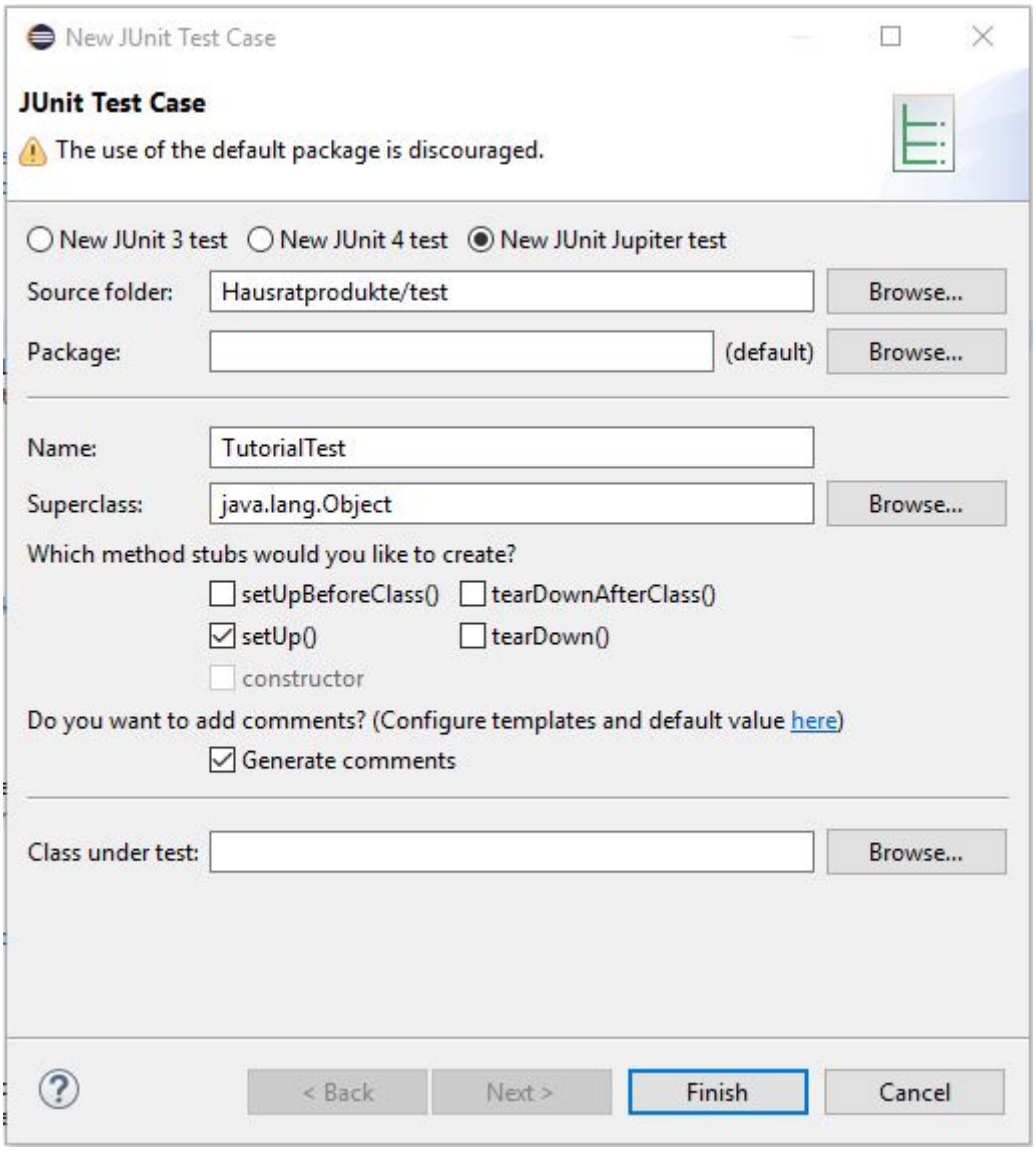

Figure 34. Wizard zur Erstellung eines JUnit Tests

Anstatt die Korrektheit der Produktdaten mit `assert`*-Statements zu testen, geben wir sie hier mit `println` auf der Konsole aus.

```
public class TutorialTest {

    private IRuntimeRepository repository;
    private HausratProdukt kompaktProdukt;

    @BeforeEach
    public void setUp() throws Exception {
        // Repository erzeugen
        repository = ClassloaderRuntimeRepository

.create("org/faktorips/tutorial/produktdaten/internal/faktorips-repository-
toc.xml");

        // Referenz auf das Kompaktprodukt aus dem Repository holen
        IProductComponent pc = repository.getProductComponent("hausrat.HR-
Kompakt 2019-07");

        // Auf die eigenen Modellklassen casten
        kompaktProdukt = (HausratProdukt) pc;
    }

    @Test
    public void test() {
        System.out.println("Produktname: " + kompaktProdukt.getProduktname());
        System.out.println("Vorschlag Vs pro 1qm: " +
kompaktProdukt.getVorschlagVersSummeProQm());
        System.out.println("Default Zahlweise: " +
kompaktProdukt.getDefaultValueZahlweise());
        System.out.println("Erlaubte Zahlweisen: " +
kompaktProdukt.getAllowedValuesForZahlweise(null));
        System.out.println("Default Vs: " +
kompaktProdukt.getDefaultValueVersSumme());
        System.out.println("Bereich Vs: " +
kompaktProdukt.getRangeForVersSumme(null));
        System.out.println("Default Wohnflaeche: " +
kompaktProdukt.getDefaultValueWohnflaeche());
        System.out.println("Bereich Wohnflaeche: " +
kompaktProdukt.getRangeForWohnflaeche(null));
    }
}
```

Wenn Sie den Test nun ausführen, sollte er folgendes ausgeben:

Produktname: Hausrat Kompakt
Vorschlag Vs pro 1qm: 600.00 EUR
Default Zahlweise: 1
Erlaubte Zahlweisen: [1, 2]
Default Vs: MoneyNull
Bereich Vs: 10000.00 EUR-2000000.00 EUR
Default Wohnflaeche: null
Bereich Wohnflaeche: 0-1000

Damit haben wir einen Einblick in die Modellierung mit Faktor -IPS bekommen und haben erste, einfache Produkte angelegt und zur Laufzeit auf Produktdaten zugegriffen.

In Teil 2 werden wir in die Verwendung von Tabellen & Formeln einführen. Diese werden wir nutzen, um das Hausratmodell so zu erweitern, dass den Produkten flexibel Zusatzdeckungen durch Anwender aus der Fachabteilung hinzu konfiguriert werden können, ohne dass das Modell erweitert werden muss.

Teil 2: Verwendung von Tabellen und Formeln

Überblick

Im ersten Teil wurde die Modellierung mit Faktor-IPS und die Konfiguration von Produkten anhand einer einfachen Hausratversicherung erläutert. Im zweiten Teil werden wir die Verwendung von Tabellen und Formeln erklären. Dazu erweitern wir das in Teil 1 erstellte Hausratmodell.

Die Kapitel gliedern sich wie folgt:

- **Verwendung von Tabellen**
  In dem Kapitel wird das Modell zunächst um eine Tabelle zur Ermittlung der Tarifzone erweitert und der Zugriff auf den Tabelleninhalt realisiert. In einem zweiten Schritt werden produktspezifische Beitragstabellen definiert und der Zusammenhang zu den Produkten modelliert.
- **Implementierung der Beitragsberechnung**
  In diesem Abschnitt wird die Beitragsberechnung implementiert und mit Hilfe eines JUnit-Tests getestet. Innerhalb der Beitragsberechnung wird auf die produktspezifischen Beitragstabellen zugegriffen.
- **Verwendung von Formeln**
  In diesem Kapitel wird das Hausratmodell um Zusatzdeckungen erweitert. Die Modellierung erlaubt es der Fachabteilung flexibel neue Zusatzdeckungen z.B. gegen Fahrraddiebstahl oder Überspannungsschäden zu definieren, ohne dass jedes Mal das Modell erweitert werden muss. Erreicht wird dies durch den Einsatz von durch die Fachabteilung definierbaren Formeln.

Verwendung von Tabellen

In diesem Kapitel erweitern wir das Modell um Tabellen zur Abbildung der Tarifzonen und Beitragssätze und programmieren die Ermittlung der für einen Vertrag gültigen Tarifzone.

Tarifzonentabelle

Aufgrund der in Deutschland regional unterschiedlichen Schadenswahrscheinlichkeit durch Einbruchdiebstahl unterscheiden Versicherungsunternehmen in der Hausratversicherung zwischen unterschiedlichen Tarifzonen. Hierzu verwenden Versicherer eine Tabelle, mit der einem Postleitzahlenbereich eine Tarifzone zugeordnet ist. Im Tutorial nutzen wir folgende Tabelle:

Plz-von	Plz-bis	Tarifzone
17235	17237	II
30159	45549	III
59174	59199	IV
47051	47279	V
63065	63075	VI
…	…	…

Tarifzonentabelle

Für alle Postleitzahlen, die in keinen der Bereiche fallen, gilt die Tarifzone I.

Faktor-IPS unterscheidet zwischen der Definition der Tabellenstruktur und dem Tabelleninhalt. Die Tabellenstruktur wird als Teil des Modells angelegt. Der Tabelleninhalt kann abhängig vom Inhalt und der Verantwortung für die Pflege der Daten sowohl als Teil des Modells oder als Teil der Produktdefinition verwaltet werden. Zu einer Tabellenstruktur kann es dabei mehrere Tabelleninhalte geben. Dies entspricht dem Konzept von Tabellenpartitionen in relationalen Datenbankmanagementsystemen.

Legen wir zunächst die Tabellenstruktur für die Ermittlung der Tarifzonen an. Hierzu wechseln Sie zunächst zurück in die Java-Perspektive. Im Projekt Hausratmodell unter dem Ordner „model“ markieren Sie den Ordner „hausrat“ und klicken dann in der Toolbar auf . Die Tabellenstruktur nennen Sie „Tarifzonentabelle“ und klicken Finish.

Als Tabellentyp wählen Sie den Typ Single Content, da es für diese Struktur nur einen Inhalt geben soll. Nun legen wir zunächst die Spalten der Tabelle an. Alle drei Spalten (plzVon, plzBis, tarifzone) sind vom Datentyp String.

Interessant wird es jetzt bei der Definition des Postleitzahlenbereiches: Die von uns angelegte Tabellenstruktur dient uns letztendlich dazu, die Funktion (im mathematischen Sinne) tarifzone → plz abzubilden. Allein mit der Spaltendefinition und einem möglichen UniqueKey ist diese Semantik allerdings nicht abbildbar. In Faktor-IPS gibt es aus diesem Grund die Möglichkeit zu modellieren, dass die Spalten (oder eine Spalte) einen Bereich darstellen. Legen Sie jetzt einen neuen Bereich an (Bereiche ► Neu). Da die Tabelle Von- und Bis-Spalten enthält, wählen Sie als Typ Two Column Range. Als „Parameter Name“ geben Sie jetzt „plz“ ein und ordnen noch die beiden Spalten plzVon und plzBis zu.

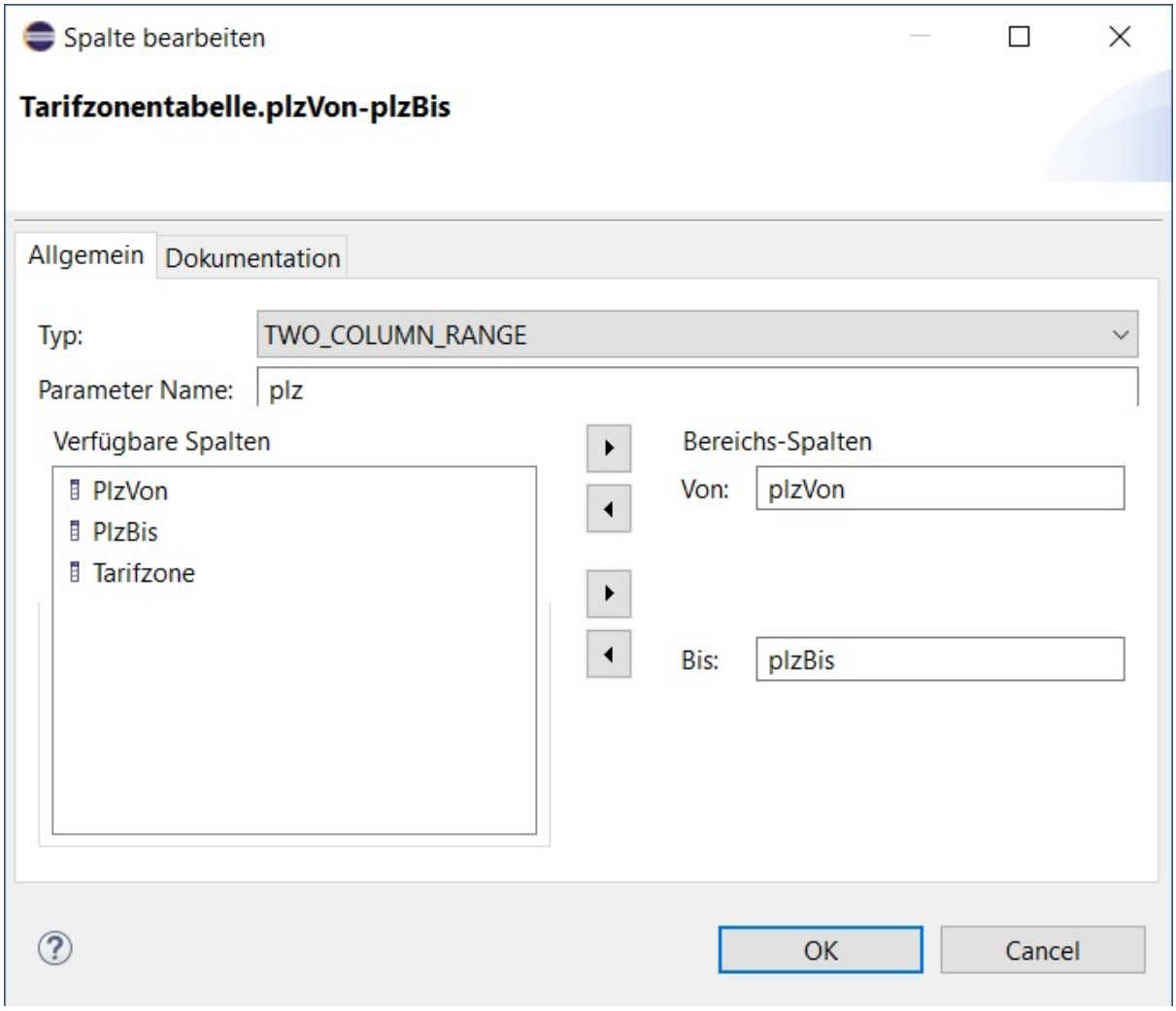

Figure 35. Bereich der Tabellenstruktur anlegen

Jetzt legen Sie noch einen neuen UniqueKey an (Indices ► Neu). Dem UniqueKey ordnen Sie jetzt **nicht** die einzelnen Spalten plzVon und plzBis zu, sondern den Bereich und speichern dann die Strukturbeschreibung.

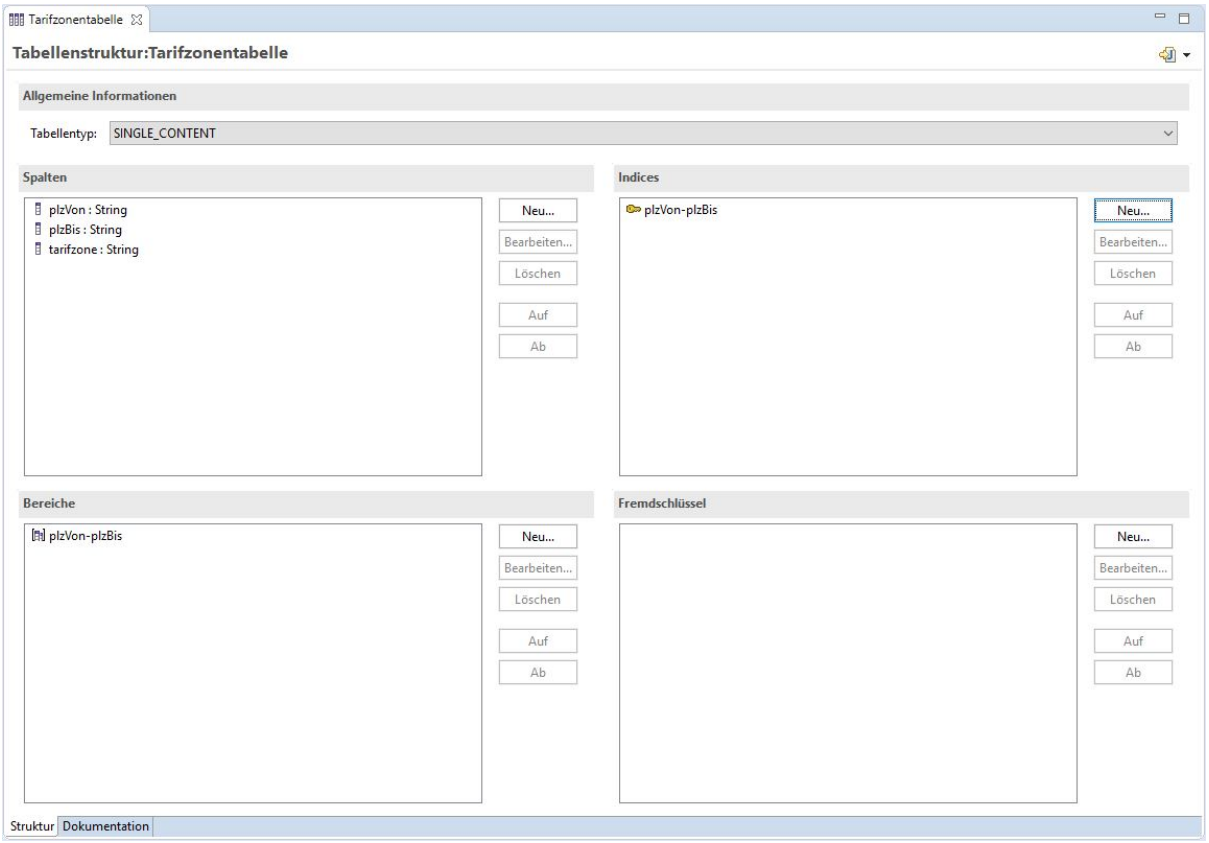

Figure 36. Tabellenstruktur Tarifzonentabelle

Faktor-IPS hat nun für die Tabellenstruktur zwei neue Klassen im Source folder „`src`“ unter dem Package `org.faktorips.tutorial.model.hausrat` generiert.

Die Klasse `TarifzonentabelleRow` repräsentiert eine Zeile der Tabelle und enthält für jede Spalte eine Membervariable mit entsprechenden Zugriffsmethoden. Die Klasse `Tarifzonentabelle` repräsentiert den Tabelleninhalt. Neben Methoden, um den Tabelleninhalt aus XML zu initialisieren, wurde aus dem UniqueKey eine Methode zum Suchen einer Zeile generiert:

```
public TarifzonentabelleRow findRow(String plz) {
    // Details der Implementierung sind hier ausgelassen
}
```

Nutzen wir jetzt diese Klasse, um die Ermittlung der Tarifzone für den Hausratvertrag zu implementieren. Die Tarifzone ist eine abgeleitete Eigenschaft des Hausratvertrags und in der Klasse `HausratVertrag` gibt es somit die Methode `getTarifzone()`. Diese hatten wir bereits wie folgt implementiert:


```
public String getTarifzone() {
    // begin-user-code
    // TODO wird spaeter anhand einer Tarifzonentabelle ermittelt
    return "I";
    // end-user-code
}
```

Nun ermitteln wir die Tarifzonen anhand der Postleitzahl aus der gerade angelegten Tabelle wie folgt:

```
public String getTarifzone() {
    // begin-user-code
    if (plz==null) {
        return null;
    }
    IRuntimeRepository repository = getHausratProdukt().getRepository();
    Tarifzonentabelle tabelle = Tarifzonentabelle.getInstance(repository);
    TarifzonentabelleRow row = tabelle.findRow(plz);
    if (row==null) {
        return "I";
    }
    return row.getTarifzone();
    // end-user-code
}
```

Es bedarf an dieser Stelle noch einer Erläuterung, wie man an die Instanz der Tabelle herankommt. Da es zur Tarifzonentabelle nur einen Inhalt gibt, hat die Klasse `Tarifzonentabelle` eine `getInstance`() Methode, die diesen Inhalt zurück liefert. Als Parameter bekommt diese Methode das RuntimeRepository, welches zur Laufzeit Zugriff auf die Produktdaten inklusive der Tabelleninhalte gibt. An dieses kommen wir leicht über das Produkt, auf dem der Vertrag basiert [2].

[2] Das Übergeben des RuntimeRepositories in die Methode `getInstance()` hat den Vorteil, dass das konkrete Repository in Testfällen leicht ausgetauscht werden kann.

Jetzt legen wir noch den Tabelleninhalt an. Die Zuordnung der Postleitzahlen zu den Tarifzonen soll von der Fachabteilung gepflegt werden. Zur Strukturierung fügen Sie noch ein neues IPS Package `Tabellen` in dem Projekt Hausratprodukte ein. Danach markieren Sie das neue Package und klicken in der Toolbar auf  . Wählen Sie in dem Dialog die Tarifzonentabelle als Struktur aus. Als Namen für den Tabelleninhalt übernehmen Sie den Namen Tarifzonentabelle und klicken Finish. In dem Editor können Sie nun die oben beispielhaft aufgeführten Zeilen erfassen. Die Projektstruktur im Produktdefinitions-Explorer sollte danach wie folgt aussehen:

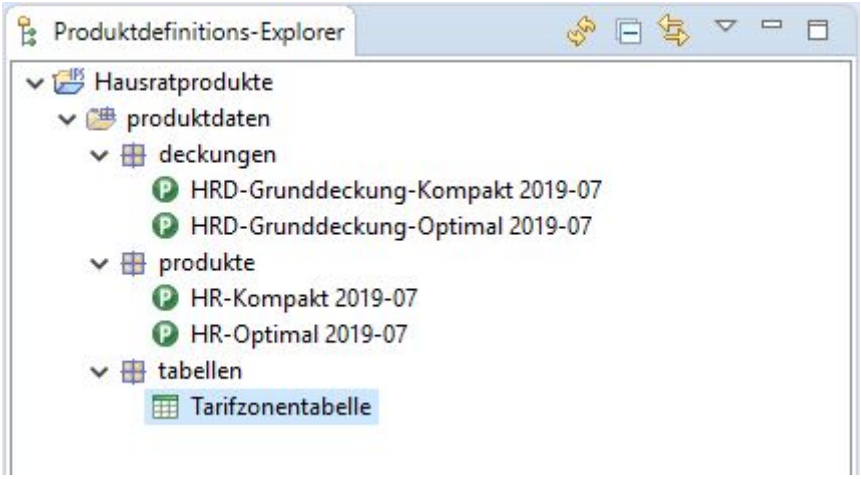

Figure 37. Projektstruktur der Produktdefinition

Zum Schluss testen wir noch die Ermittlung der Tarifzone. Hierzu erweitern wir den im ersten Teil des Tutorials angelegten JUnit Test „TutorialTest“ um die folgende Testmethode [3].

[3] Im Tutorial *Softwaretests mit Faktor-IPS* wird u.a. beschrieben, wie man diesen Test mit Faktor-IPS Hilfsmitteln komfortabel erzeugen und durchführen kann.

```
@Test
public void testGetTarifzone() {
    // Erzeugen eines Hausratvertrags mit der Factorymethode des Produktes
    HausratVertrag vertrag = kompaktProdukt.createHausratVertrag();
    vertrag.setPlz("45525");
    assertEquals("III", vertrag.getTarifzone());
}
```

Beitragstabelle

Der Beitragssatz für die Grunddeckung der Hausratversicherung soll anhand der Tarifzone aus einer Tariftabelle ermittelt werden. Dabei sollen für die beiden Produkte unterschiedliche Beitragssätze gemäß den folgenden Tabellen gelten.

Tarifzone	**Beitragssatz**
I	0.80
II	1.00
III	1.44
IV	1.70
V	2.00
VI	2.20

Table 4. Beitragstabelle HR-Optimal

Tarifzone	Beitragssatz
I	0.60
II	0.80
III	1.21
IV	1.50
V	1.80
VI	2.00

Beitragstabelle HR-Kompakt

Die Daten für die unterschiedlichen Produkte werden häufig in einer Tabelle zusammengefasst, in der es dann noch eine Spalte „ProduktID“ gibt. In Faktor-IPS kann man aber auch zu einer Tabellenstruktur mehrere Inhalte anlegen und den Zusammenhang zu den Produktbausteinen explizit modellieren!

Legen Sie hierfür eine Tabellenstruktur *TariftabelleHausrat* mit den beiden Spalten Tarifzone (String) und Beitragssatz (Decimal) an. Definieren Sie einen UnqiueKey auf die Spalte Tarifzone. Als Tabellentyp wählen Sie diesmal Multiple Contents aus, da wir für jedes Produkt einen eigenen Tabelleninhalt anlegen wollen.

Erzeugen Sie nun für die Produkte *HR-Optimal* und *HR-Kompakt* (bzw. genauer für deren Grunddeckungstypen) jeweils einen Tabelleninhalt mit dem Namen „Tariftabelle Optimal 2019-07“ und „Tariftabelle Kompakt 2019-07“ [4].

[4] Die Endung „2019-07“ sollten Sie dabei entsprechend des von Ihnen verwendeten Wirksamkeitsdatums anpassen.

Das folgende Diagramm zeigt den Zusammenhang zwischen der Klasse *Grunddeckungstyp* und der Tabellenstruktur *TariftabelleHausrat* und die entsprechenden Objektinstanzen.

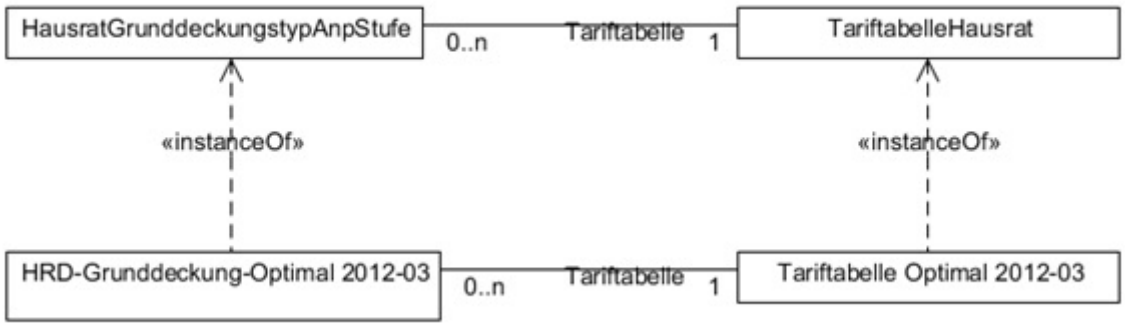

Figure 38. Zusammenhang Produktbausteine & Tabellen

Um den Zusammenhang zwischen Tabellen und Produkten in Faktor-IPS zu definieren,

öffnen Sie die Klasse *HausratGrunddeckungstyp*. Auf der zweiten Seite „Verhalten“ im Editor [5] im Abschnitt *Table Usages (Verwendete Tabellen)* können Sie eine neue Tabellenverwendung definieren. Hierzu klicken Sie auf den *New Button* im Abschnitt „Verwendete Tabellen“. Geben Sie „tariftabelle“ als Rollennamen ein, wählen Sie „Tabelleninhalt erforderlich“ aus und fügen Sie „Tabellenstruktur“ *TariftabelleHausrat* hinzu, laut nachfolgender Abbildung:

[5] Vorausgesetzt, Sie haben in den Preferences eingestellt, dass die Editoren zwei Abschnitte pro Seite verwenden sollen.

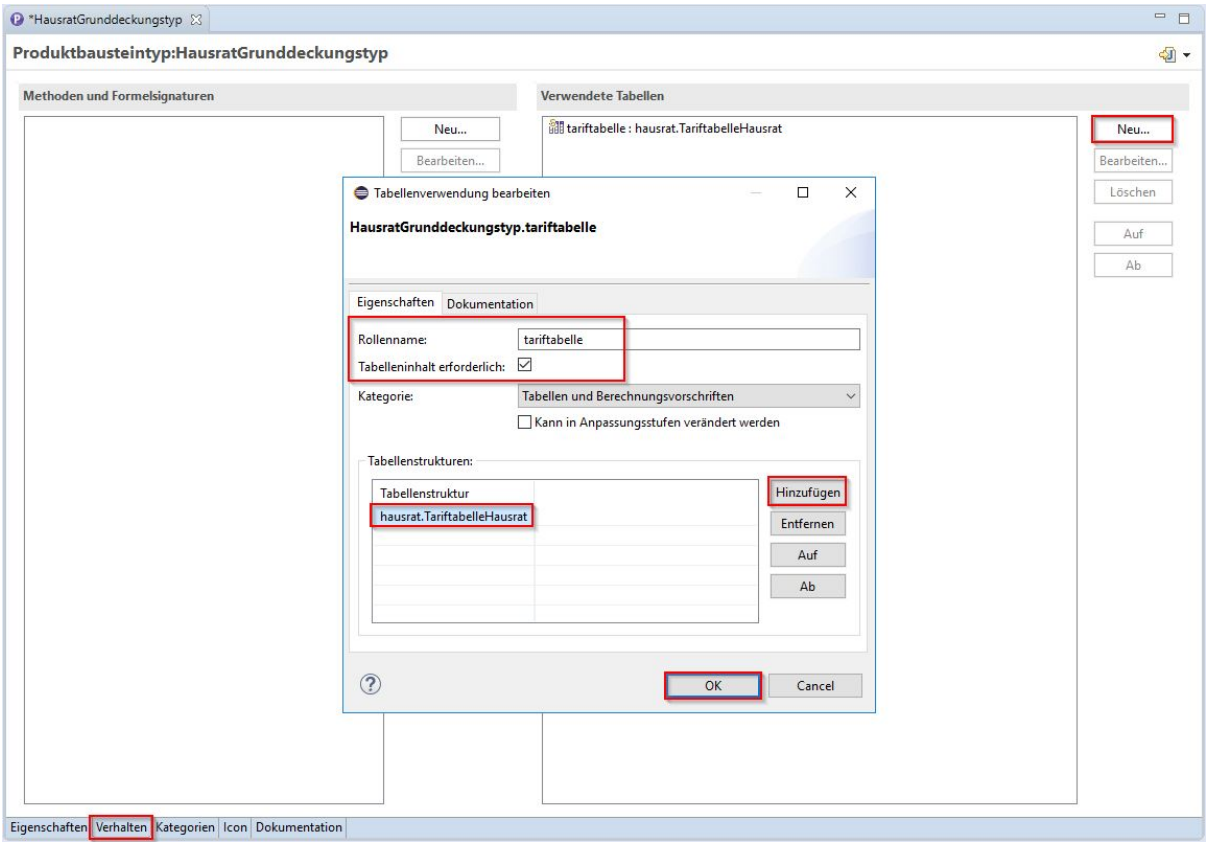

Figure 39. Modellierung der Verwendung von Tabellen

An dieser Stelle können Sie auch mehrere Tabellenstrukturen zuordnen, da im Laufe der Zeit z.B. neue Tarifierungsmerkmale hinzukommen und so unter der Rolle Tariftabelle unterschiedliche Tabellenstrukturen möglich sein können. Haken Sie noch die Checkbox *Table content required* an, da für jeden Grunddeckungstypen eine Tariftabelle angegeben werden muss, dann schließen Sie den Dialog und speichern.

Nun können wir die Tabelleninhalte den Grunddeckungstypen zuordnen. Öffnen Sie zunächst *HRD-Grunddeckung-Kompakt 2019-07*. Den Dialog, der Sie darauf hinweist, dass die Tariftabelle noch nicht zugeordnet ist, bestätigen Sie mit *Fix*. In dem Abschnitt *Tabellen und Berechnungsvorschriften* können Sie nun die Tariftabelle für HR-Kompakt zuordnen und dann speichern.

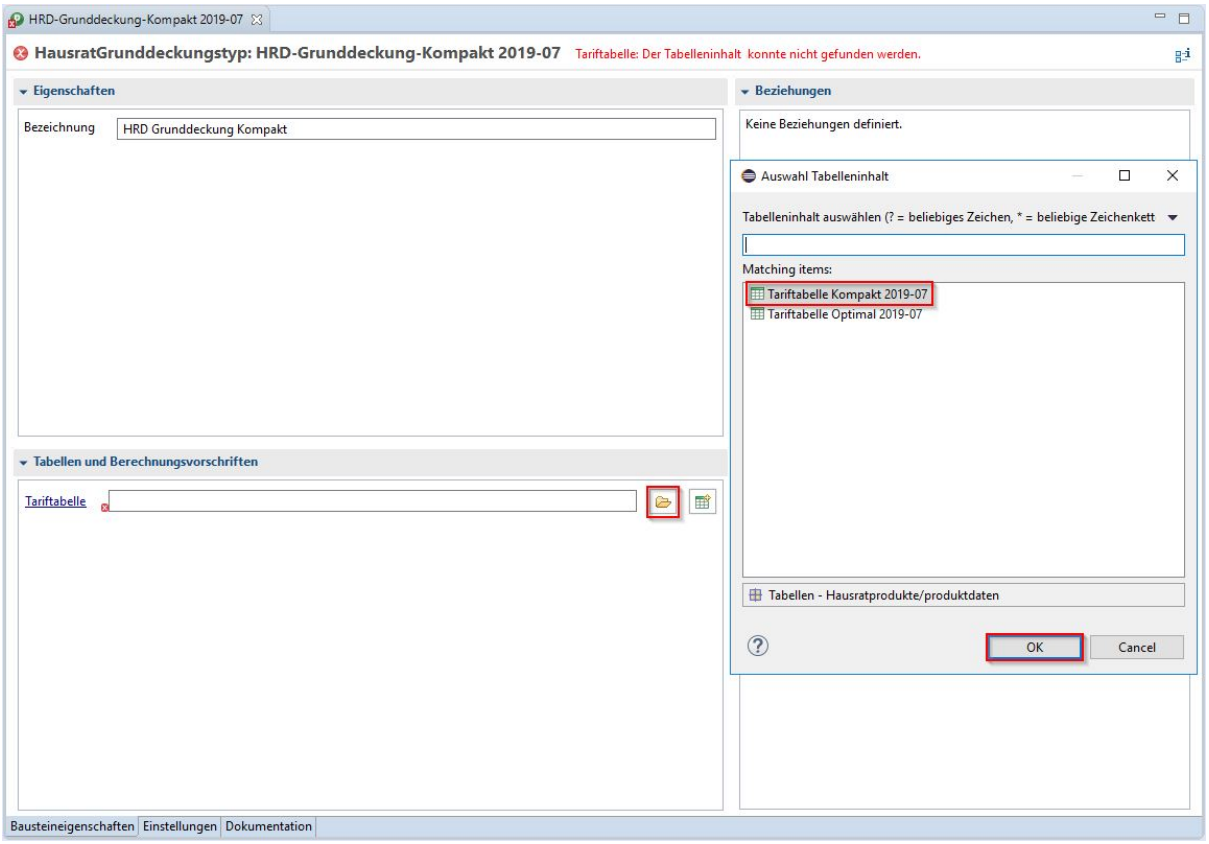

Figure 40. Zuordnung der Tabelleninhalte

Analog verfahren Sie für HR-Optimal.

Zum Schluss dieses Kapitels werfen wir noch einen Blick auf den generierten Sourcecode. In der Klasse *HausratGrunddeckungstyp* gibt es eine Methode, um den zugeordneten Tabelleninhalt zu erhalten:

```
public TariftabelleHausrat getTariftabelle() {
    if (tariftabelleName == null) {
        return null;
    }
    return (TariftabelleHausrat) getRepository().getTable(tariftabelleName);
}
```

Da auch die Findermethoden an der Tabelle generiert sind, lässt sich so mit sehr wenigen Zeilen Sourcecode ein effizienter Zugriff auf eine Tabelle realisieren.

Implementieren der Beitragsberechnung

In diesem Kapitel werden wir nun die Beitragsberechnung für unsere Hausratprodukte implementieren.

Wir erweitern unser Modell um die Methoden gemäß der folgenden Abbildung.

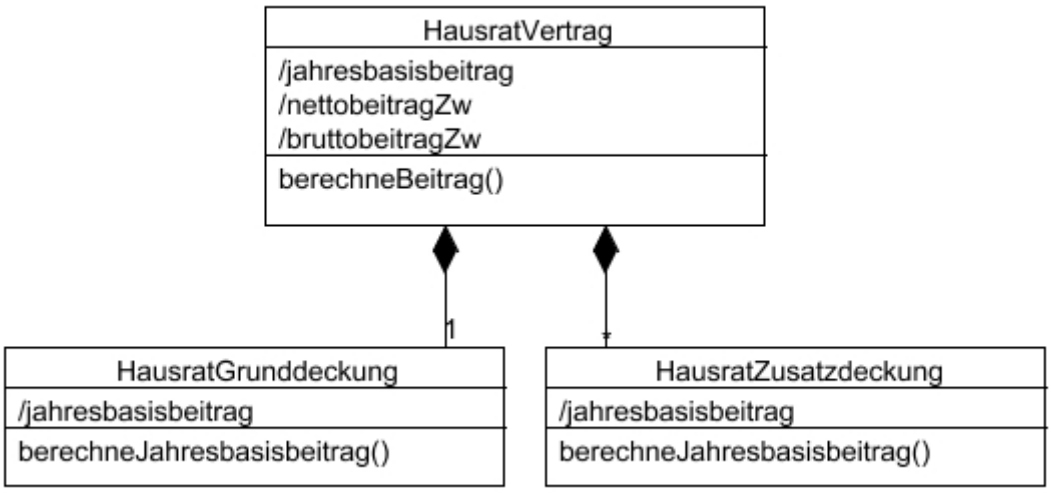

Figure 41. Berücksichtigung der Beitragsberechnung im Modell

Jede Deckung hat einen Jahresbasisbeitrag (/jahresbasisbeitrag). Der Jahresbasisbeitrag des Vertrags ist die Summe über die Jahresbasisbeiträge seiner Deckungen.

Der Nettobeitrag je Zahlungsperiode (/nettobeitragZw) ist der vom Beitragszahler zu zahlende Beitrag ohne Versicherungssteuer. Er ergibt sich aus dem Jahresbasisbeitrag dividiert durch Anzahl der Zahlungsperioden und multipliziert mit 1+Höhe des Ratenzahlungszuschlages.

Der Bruttobeitrag je Zahlungsweise (/bruttobeitragZw) ist der vom Beitragszahler zu zahlende Bruttobeitrag pro Zahlung. Er ergibt sich aus der Multiplikation des NettobeitragZw mit 1+Versicherungssteuersatz. In dem Tutorial verwenden wir Ratenzahlungszuschlag in Höhe von 3% und Versicherungssteuer von 19%.

Legen Sie nun die neuen Attribute in den Klassen *HausratVertrag* und *HausratGrunddeckung* an. Um die *HausratZusatzdeckungen* kümmern wir uns im nächsten Kapitel. Alle Attribute sind abgeleitet (cached, Berechnung durch expliziten Methodenaufruf) und vom Datentyp Money. Da es sich um ein gecachedes, abgeleitetes Attribut handelt, generiert Faktor-IPS je eine Membervariable und eine Gettermethode.

Die Berechnung aller Beitragsattribute erfolgt durch die Methode `berechneBeitrag()` der Klasse HausratVertrag. Die Methode berechnet dabei alle Beitragsattribute des Vertrags und auch den Jahresbasisbeitrag der Deckungen. Hierzu verwendet sie natürlich die Methode `berechneJahresbasisbeitrag()` der Deckungen.

Als nächstes legen wir diese Methoden an. Öffnen Sie die Klasse „HausratVertrag“, im Editor wechseln Sie auf die zweite Seite „Verhalten“ (analog wie in Abbildung 5). Im Bereich „Methoden“ drücken Sie auf Button „Neu“, geben den Namen für die Methode ein und drücken auf Button „OK“. Die folgende Abbildung zeigt den Dialog zum Bearbeiten einer Methodensignatur.

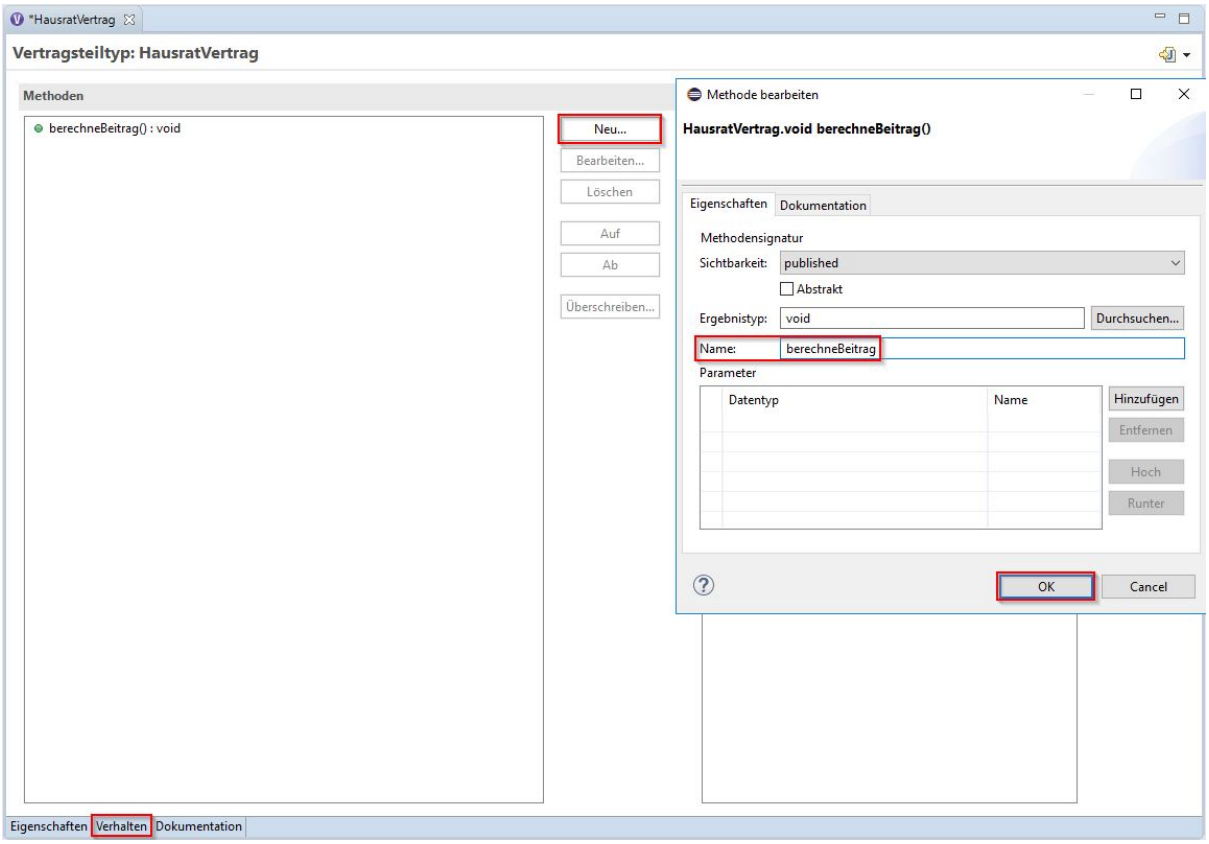

Figure 42. Dialog zum Bearbeiten einer Methodensignatur

> *Im Vergleich zur Modellierung von Beziehungen und Attributen bietet die Codegenerierung für Methoden weniger Vorteile. Methoden können daher natürlich auch direkt im Sourcecode definiert werden.*

Der folgende Sourcecodeausschnitt zeigt die Implementierung der spartenübergreifenden Beitragsberechnung in der Klasse `HausratVertrag`. Aus Übersichtlichkeitsgründen implementieren wir die Berechnung von Jahresbasisbeitrag und NettobeitragZw in zwei eigenen privaten Methoden, die wir direkt in den Sourcecode schreiben, ohne sie ins Modell aufzunehmen.

```
/**
 * @generated NOT
 */
public void berechneBeitrag() {
    berechneJahresbasisbeitrag();
    berechneNettobeitragZw();
    Decimal versSteuerFaktor = Decimal.valueOf(119, 2);
    // 1+Versicherungssteuersatz=1.19 (119 Prozent)
    bruttobeitragZw = nettobeitragZw.multiply(versSteuerFaktor,
RoundingMode.HALF_UP);
}
```



```
/**
 * @generated NOT
 */
private void berechneJahresbasisbeitrag() {
    jahresbasisbeitrag = Money.euro(0, 0);
    HausratGrunddeckung hausratGrunddeckung = getHausratGrunddeckung();

    hausratGrunddeckung.berechneJahresbasisbeitrag();
    jahresbasisbeitrag =
jahresbasisbeitrag.add(hausratGrunddeckung.getJahresbasisbeitrag());
    /*
     * TODO: wenn das Modell um Zusatzdeckungen erweitert wird, muss deren
     * Beitrag an dieser Stelle natürlich auch hinzuaddiert werden
     */
}

/**
 * @generated NOT
 */
private void berechneNettobeitragZw() {
    if (zahlweise == null) {
        nettobeitragZw = Money.NULL;
        return;
    }
    if (zahlweise.intValue() == 1) {
        nettobeitragZw = jahresbasisbeitrag;
    } else {
        Decimal rzFaktor = Decimal.valueOf(103, 2);
        // 1+ratenzahlungszuschlag=1.03 (103 Prozent)
        nettobeitragZw = jahresbasisbeitrag.multiply(rzFaktor,
RoundingMode.HALF_UP);
    }
    nettobeitragZw = nettobeitragZw.divide(zahlweise.intValue(),
RoundingMode.HALF_UP);
}
```

Legen Sie zudem ein neues Attribut mit dem Namen `wirksamAb` vom Typ GregorianCalender in dem Editor der Klasse `HausratVertrag` an. Der generierte Sourcecode der Setter- und Gettermethode sollte wie folgt aussehen.


```
/**
 * Gibt den Wert des Attributs WirksamAb zurueck.
 *
 * @generated
 */
@IpsAttribute(name = "WirksamAb", kind = AttributeKind.CHANGEABLE,
valueSetKind = ValueSetKind.AllValues)
public GregorianCalendar getWirksamAb() {
    return wirksamAb == null ? null : (GregorianCalendar) wirksamAb.clone();
}

/**
 * Setzt den Wert des Attributs WirksamAb.
 *
 * @generated
 */
@IpsAttributeSetter("WirksamAb")
public void setWirksamAb(GregorianCalendar newValue) {
    this.wirksamAb = newValue == null ? null : (GregorianCalendar)
newValue.clone();
}
```

Außerdem sollte die generierte Methode getEffectiveFromAsCalendar() das Wirksamkeitsdatum zurück geben. Hier sehen wir den generierten Sourcecode.

```
/**
 * {@inheritDoc}
 *
 * @generated
 */
@Override
public Calendar getEffectiveFromAsCalendar() {
    // TODO Implementieren des Zugriffs auf das Wirksamkeitsdatum (wird
benoetigt um
    // auf die gueltigen Produktdaten zuzugreifen).
    // Damit diese Methode bei erneutem Generieren nicht neu ueberschrieben
wird,
    // muss im Javadoc ein NOT hinter @generated geschrieben werden!
    return null;
}
```

Ändern Sie die Methode wie folgt.


```
/**
 * {@inheritDoc}
 *
 * @generated NOT
 */
@Override
public Calendar getEffectiveFromAsCalendar() {
    return getWirksamAb();
}
```

Beitragsberechnung für die Hausrat-Deckungen

Für die Hausratversicherung muss nun die Beitragsberechnung auf Deckungsebene implementiert werden.

Der Jahresbasisbeitrag für die Grunddeckung wird wie folgt berechnet:

- Ermittlung des Beitragssatzes pro 1000 Euro Versicherungssumme aus der Tariftabelle.
- Division der Versicherungssumme durch 1000 Euro und Multiplikation mit dem Beitragssatz.

Da sich diese Berechnungsvorschrift nicht ändert, implementieren wir sie direkt in der Javaklasse *HausratGrunddeckung*. Bei den Zusatzversicherungen werden wir dann der Fachabteilung erlauben den Jahresbasisbeitrag über Berechnungsformeln festzulegen.

Zunächst definieren wir die Methode *berechneJahresbasisbeitrag()* im Editor der *HausratGrunddeckung* (mit Access Modifier published).

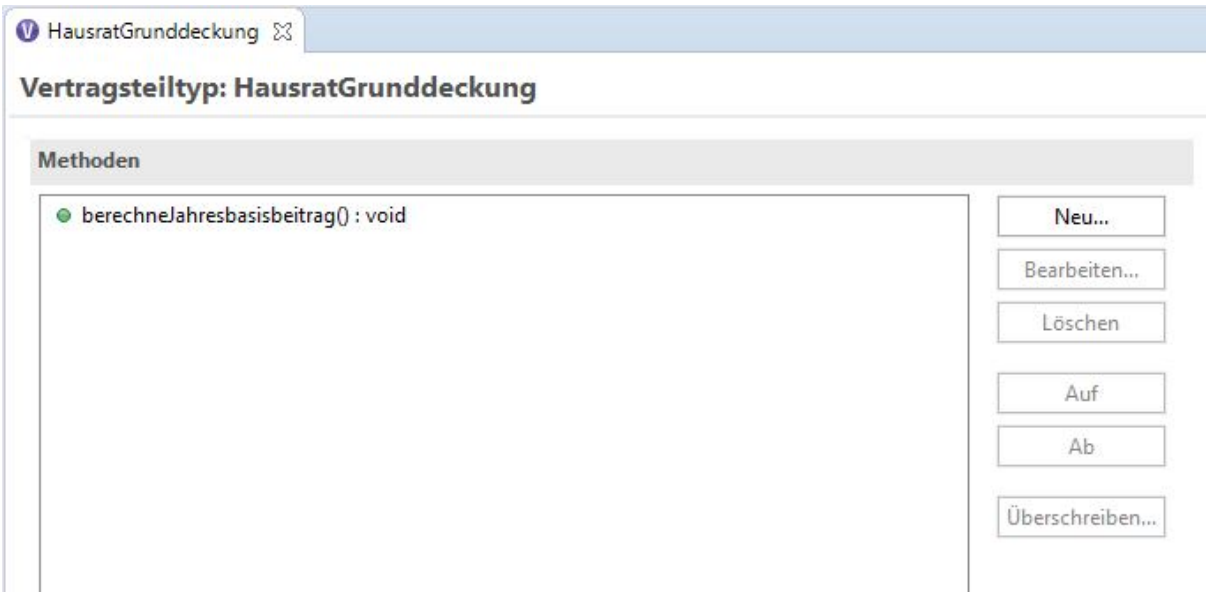

Figure 43. Anlegen der Methode berechneJahresbasisbeitrag

Öffnen Sie nun die Javaklasse im Editor und implementieren Sie die Methode wie folgt:

```
/**
 * @generated NOT
 */
public void berechneJahresbasisbeitrag() {
    TariftabelleHausrat tabelle = getTariftabelle();
    TariftabelleHausratRow row = null;
    if (tabelle != null) {
        row = tabelle.findRow(getHausratVertrag().getTarifzone());
    }
    if (row == null) {
        jahresbasisbeitrag = Money.NULL;
        return;
    }
    Money vs = getHausratVertrag().getVersSumme();
    Decimal beitragssatz = row.getBeitragssatz();
    jahresbasisbeitrag = vs.divide(1000,
RoundingMode.HALF_UP).multiply(beitragssatz, RoundingMode.HALF_UP);
}
```

Wir testen die Beitragsberechung indem wir wieder unseren JUnit-Test erweitern.

```
@Test
public void testBerechneBeitrag() {
    // Erzeugen eines hausratvertrags mit der Factorymethode des Produktes
    HausratVertrag vertrag = kompaktProdukt.createHausratVertrag();
    // Wirksamkeitsdatum des Vertrages setzen
    vertrag.setWirksamAb(new GregorianCalendar(2019, 7, 19));
    // Vertragsattribute setzen
    vertrag.setPlz("45525"); // => tarifzone 3
    vertrag.setVersSumme(Money.euro(60000));
    vertrag.setZahlweise(new Integer(2)); // halbjaehrlich
    /*
     * Grunddeckungstyp holen, der dem Produkt in der Anpassungsstufe
     * zugeordnet ist.
     */
    HausratGrunddeckungstyp deckungstyp = kompaktProdukt
            .getHausratGrunddeckungstyp();
    // Grunddeckung erzeugen und zum Vertrag hinzufuegen
    HausratGrunddeckung deckung = vertrag
            .newHausratGrunddeckung(deckungstyp);
    // Beitrag berechnen und Ergebniss pruefen
    vertrag.berechneBeitrag();

    // tarifzone 3 => beitragssatz = 1.21 jahresbasisbeitrag
    // = versicherungsumme / 1000 * beitragssatz = 60000 / 1000 * 1,21
    // = 72,60
    assertEquals(Money.euro(72, 60), deckung.getJahresbasisbeitrag());

    // Jahresbasisbeitrag vertrag = Jahresbasisbeitrag deckung
    assertEquals(Money.euro(72, 60), vertrag.getJahresbasisbeitrag());

    // NettobeitragZw = 72,60 / 2 * 1,03 (wg. Ratenzahlungszuschlag von 3%)
    // = 37,389
    // => 37,39
    assertEquals(Money.euro(37, 39), vertrag.getNettobeitragZw());

    // BruttobeitragZw = 37,39 * Versicherungssteuersatz = 37,39 * 1,19
    // = 44,49
    assertEquals(Money.euro(44, 49), vertrag.getBruttobeitragZw());
}
```

Verwendung von Formeln

Unser bisheriges Hausratmodell bietet der Fachabteilung wenig Flexibilität. Ein Produkt kann genau eine Grunddeckung haben, der Beitrag wird über die Tariftabelle festgelegt. Jetzt wollen wir es der Fachabteilung erlauben, flexibel Zusatzdeckungen zu definieren,

ohne dass das Modell oder der Programmcode (durch die Anwendungsentwicklung) geändert werden muss. Für die Berechnung des Beitrags werden wir hierzu die Formelsprache von Faktor-IPS verwenden.

Zusatzdeckungen gegen Fahrraddiebstahl und Überspannungsschäden dienen uns als Beispiel:

	Fahrraddiebstahl	*Überspannung*
Versicherungssumme der Zusatzdeckung	1% der im Vertrag vereinbarten Summe, maximal 3000 Euro.	5% der im Vertrag vereinbarten Summe. Keine Deckelung.
Jahresbasisbeitrag	10% der Versicherungssumme der Fahrraddiebstahldeckung	10 Euro + 3% der Versicherungssumme der Überspannungsdeckung

Die Zusatzdeckungen haben eine eigene Versicherungssumme, die von der im Vertrag vereinbarten Versicherungssumme abhängt. Der Jahresbasisbeitrag hängt wiederum von der Versicherungssumme der Deckung ab. Um solche Zusatzdeckungen abbilden zu können, erweitern wir das Modell nun wie im folgenden Klassendiagramm abgebildet:

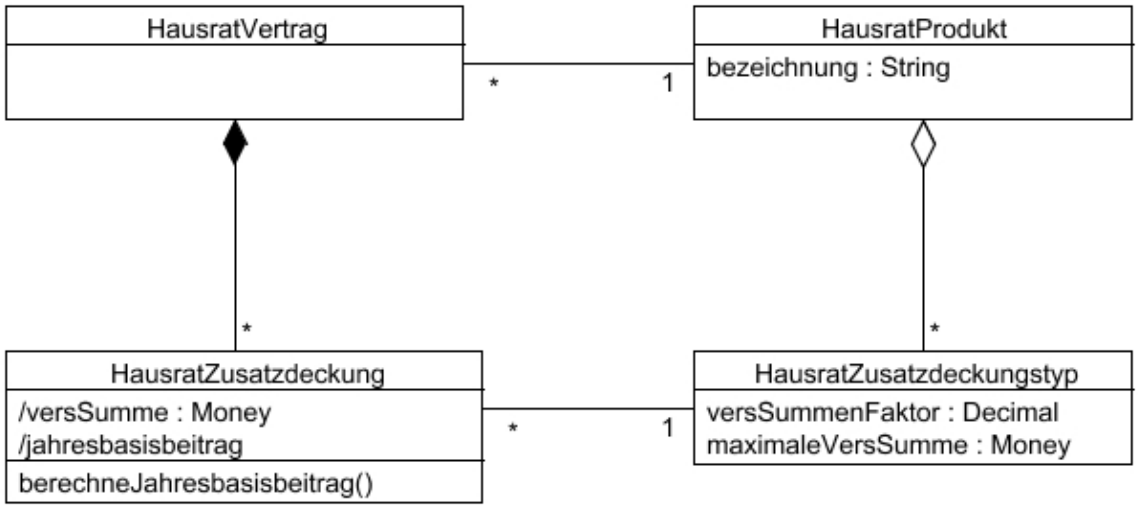

Figure 44. Modellausschnitt für Zusatzdeckungen

Ein *HausratVertrag* kann beliebig viele solcher Zusatzdeckungen enthalten. Die Konfigurationsklasse zur *HausratZusatzdeckung* nennen wir *HausratZusatzdeckungstyp*. Diese hat die Eigenschaften „versSummenFaktor“ und „maximaleVersSumme“. Die Versicherungssumme der Zusatzdeckung ergibt sich durch Multiplikation der Versicherungssumme des Vertrags mit dem Faktor und wird durch die maximale Versicherungssumme gedeckelt.

Unsere beiden Beispieldeckungen sind Instanzen der Klasse *HausratZusatzdeckungstyp*. Dies zeigt das folgende Diagramm.

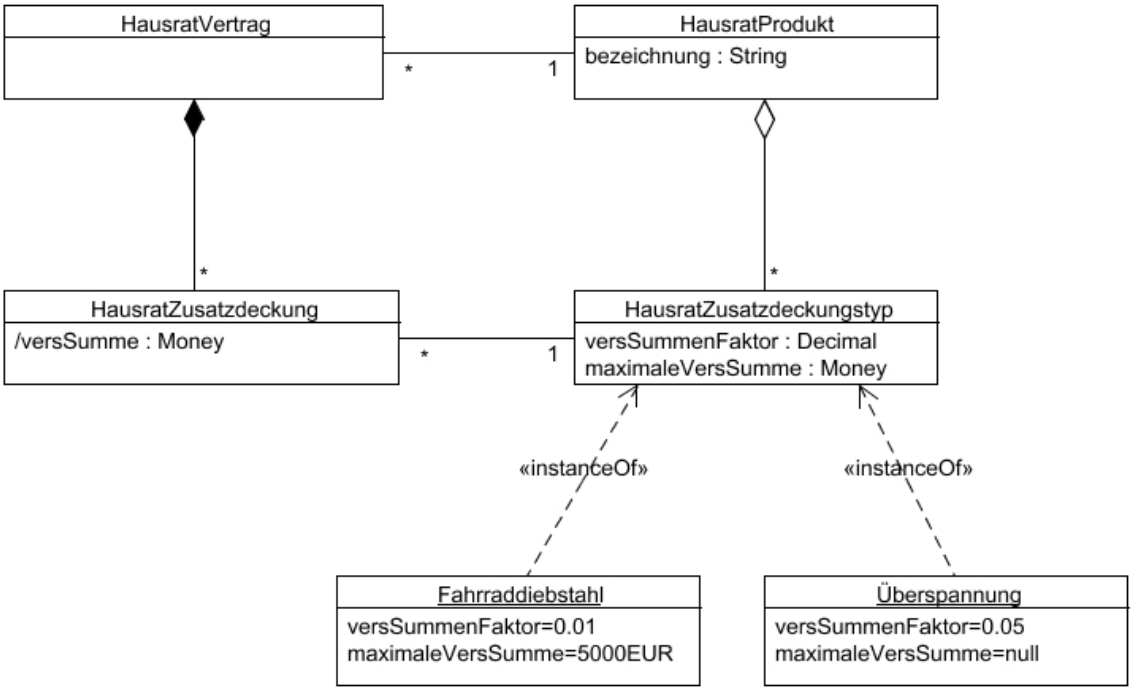

Figure 45. Modellausschnitt für Zusatzdeckungen mit Instanzen

Bevor wir zur Beitragsberechnung kommen, legen wir zunächst die beiden neuen Klassen *HausratZusatzdeckung* und *HausratZusatzdeckungstyp* an. Mit dem Assistenten zum Anlegen einer neuen Vertragsklasse kann man auch direkt die zugehörige Konfigurationsklasse mit erzeugen. Starten Sie nun den Assistenten und geben auf der ersten Seite die Daten wie im folgenden Bild zu sehen ein

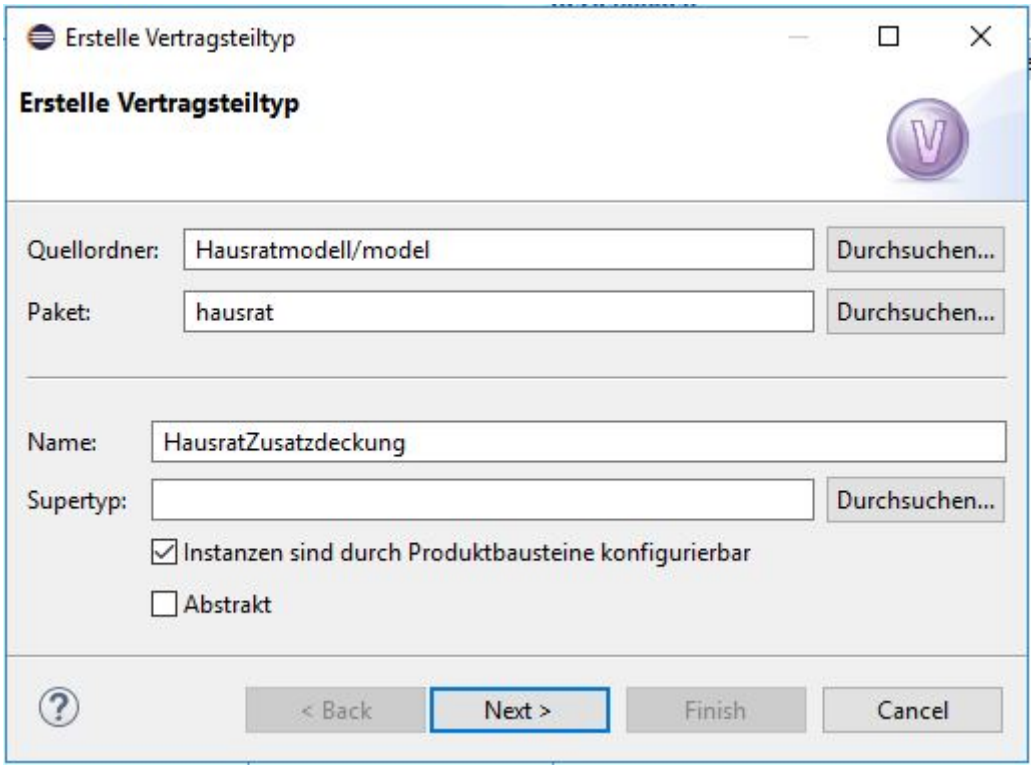

Figure 46. Assistent zum Anlegen der Vertragsklasse HausratZusatzdeckung

Auf der zweiten Seite geben Sie noch den Klassennamen HausratZusatzdeckungstyp ein und klicken *Finish*. Faktor-IPS legt nun beide Klassen an und richtet auch die Referenzen aufeinander ein.

Nun müssen wir noch die Beziehung zwischen *HausratVertrag* und *HausratZusatzdeckung* und entsprechend zwischen *HausratProdukt* und *HausratZusatzdeckungstyp* anlegen. Hierzu verwenden Sie den Assistenten zum Anlegen einer neuen Beziehung im Vertragsklasseneditor. Dieser erlaubt es Ihnen gleichzeitig auch die Beziehung auf der Produktseite mit anzulegen.

Nachdem die Beziehungen definiert sind, legen Sie an der Klasse *HausratZusatzdeckung* das abgeleitete (cached) Attribut versSumme (Money) an und an der Klasse *HausratZusatzdeckungstyp* die Attribute versSummenFaktor (Decimal), maximaleVersSumme (Money), und bezeichnung (String). Nachdem dies geschehen ist, können wir auch direkt die Ermittlung der Versicherungssumme implementieren. Hierzu implementieren wir in der Klasse `HausratZusatzdeckung` die Methode `getVersSumme()` wie folgt:


```
/**
 * Gibt den Wert des Attributs versSumme zurueck.
 *
 * @generated NOT
 */
@IpsAttribute(name = "versSumme", kind =
AttributeKind.DERIVED_BY_EXPLICIT_METHOD_CALL, valueSetKind =
ValueSetKind.AllValues)
public Money getVersSumme() {
    HausratZusatzdeckungstyp gen = getHausratZusatzdeckungstyp();
    if (gen == null) {
        return Money.NULL;
    }
    Decimal faktor = gen.getVersSummenFaktor();
    Money vsVertrag = getHausratVertrag().getVersSumme();
    Money vs = vsVertrag.multiply(faktor, RoundingMode.HALF_UP);
    if (vs.isNull()) {
        return vs;
    }
    Money maxVs = gen.getMaximaleVersSumme();
    if (vs.greaterThan(maxVs)) {
        return maxVs;
    }
    return vs;
}
```

Nun legen wir die beiden Deckungstypen für die Versicherung von Fahrraddiebstählen bzw. Überspannungsschäden an. Wechseln Sie hierzu wieder in die Produktdefinitionsperspektive und markieren im Produktdefinitionsexplorer im Projekt Hausratprodukte das Paket „deckungen“. Nun legen Sie zwei Produktbausteine mit den Namen HRD-Fahrraddiebstahl 2019-07 und HRD-Überspannung 2019-07 basierend auf der Klasse HausratZusatzdeckungstyp an und geben im Editor deren Eigenschaften ein.

	HRD-Fahrraddiebstahl 2019-07	**HRD-Überspannung 2019-07**
Bezeichnung	Fahrraddiebstahl	Überspannungsschutz
VersSummenFaktor	0.01	0.05
Maximale VersSumme	3000 EUR	<null>

Nun müssen die neuen Deckungen noch den Produkten zugeordnet werden. Dies erfolgt analog zur Zuordnung der Grunddeckungen. Bei Verträgen, die auf Basis des Produktes HR-Optimal (das im Ordner „produkte“ liegt) abgeschlossen werden, sollen die Deckungen immer enthalten sein (Art der Beziehung ist Obligatorisch), beim Produkt HR-Kompakt sollen Sie optional dazu gewählt werden können (Art der Beziehung Optional). Sie können

dies über die Art der Beziehung im Bausteineditor einstellen.

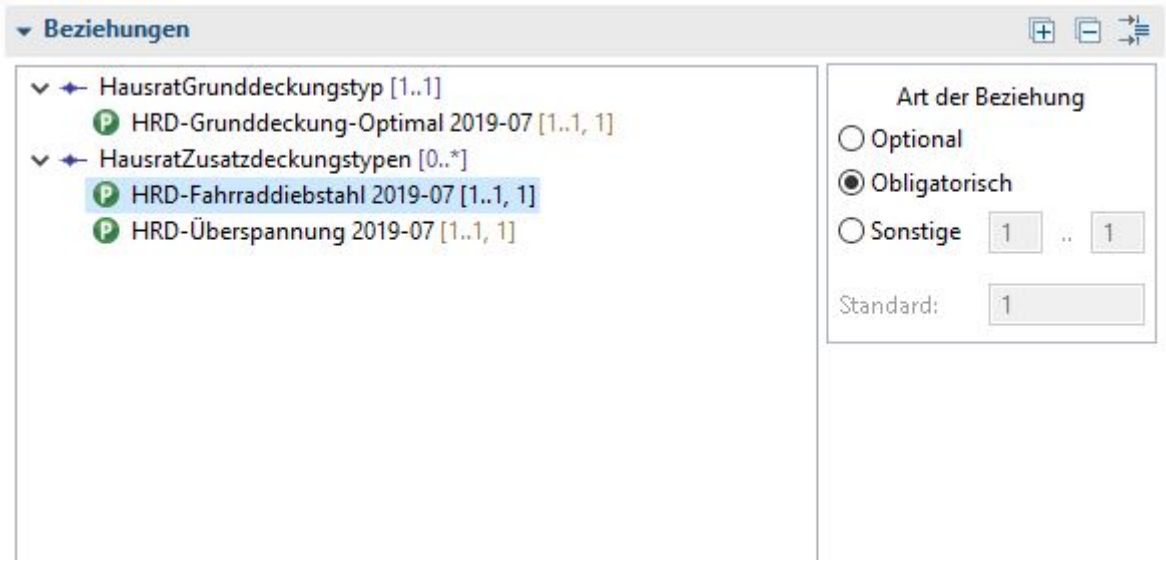

Figure 47. Ausschnitt aus dem Bausteineditor zum Bearbeiten der Beziehungen

Beitragsberechnung für die Zusatzdeckungen

Die Berechnung des Jahresbasisbeitrags soll durch die Fachabteilung durch eine Formel festgelegt werden. Der Beitrag für eine Zusatzdeckung wird i.d.R. von der Versicherungssumme und evtl.weiteren risikorelevanten Merkmalen abhängen [3]. In der Formel muss man also auf diese Eigenschaften zugreifen können. Hierzu sind prinzipiell zwei Wege denkbar:

- die Formelsprache erlaubt eine beliebige Navigation durch den Objektgraphen
- die in der Formel verwendbaren Parameter werden explizit festgelegt.

In Faktor-IPS wird die zweite Alternative verwendet. Hierfür gibt es zwei Gründe:

1. Die Syntax für die Navigation durch den Objektgraphen kann schnell komplex werden. Wie sieht zum Beispiel eine für die Fachabteilung leicht verständliche Syntax zur Ermittlung der Deckung mit der höchsten Versicherungssumme aus?
2. Aktualität von abgeleiteten Attributen

[3] In der Hausratversicherung neben der Tarifzone zum Beispiel die Art des Hauses (Ein-/Mehrfamilienhaus) oder die Bauweise.

Der zweite Aspekt lässt sich am besten an einem Beispiel erläutern. Für die Berechnung des Beitrags einer Zusatzdeckung wird die Versicherungssumme benötigt. Diese ist selbst ein abgeleitetes Attribut. Wenn es sich dabei um ein gecachtes Attribut handelt, muss vor dem Aufruf der Formel zur Beitragsberechnung sichergestellt werden, dass die Versicherungssumme berechnet wurde. Erlaubt man nun eine beliebige Navigation durch den Objektgraphen muss man für alle erreichbaren Objekte sicherstellen, dass die

abgeleiteten Attribute berechnete Werte enthalten. Da dies leicht zu Fehlern führt und auch bzgl. der Performance ungünstig ist, werden in Faktor-IPS die in einer Formel verwendbaren Parameter explizit festgelegt.

Als Formelparameter können sowohl einfache Parameter wie z. B. die Versicherungssumme definiert werden als auch ganze Objekte wie z.B. der Vertrag. Letzteres hat den Vorteil, dass die Parameterliste nicht erweitert werden muss, wenn die Fachabteilung auf bisher nicht genutzte Merkmale zugreift. Für die Berechnung des Jahresbasisbeitrags der Hausratzusatzdeckung verwenden wir die Zusatzdeckung selbst und den Hausratvertrag (zu dem die Deckung gehört) als Parameter.

Um in Zusatzdeckungen die Formel für die Beitragsberechnung hinterlegen zu können, muss zunächst in der Klasse *HausratZusatzdeckungstyp* die Formelsignatur mit den Parametern definiert werden. Öffnen Sie nun den Editor für die Klasse *HausratZusatzdeckungstyp*. Klicken Sie auf der zweiten Seite im Abschnitt *Methods and Formula Signatures* (Methoden und Formelsignaturen) auf den *New* Button, um eine Formelsignatur anzulegen und tragen Sie die Daten wie im folgenden Screenshot zu sehen ein.

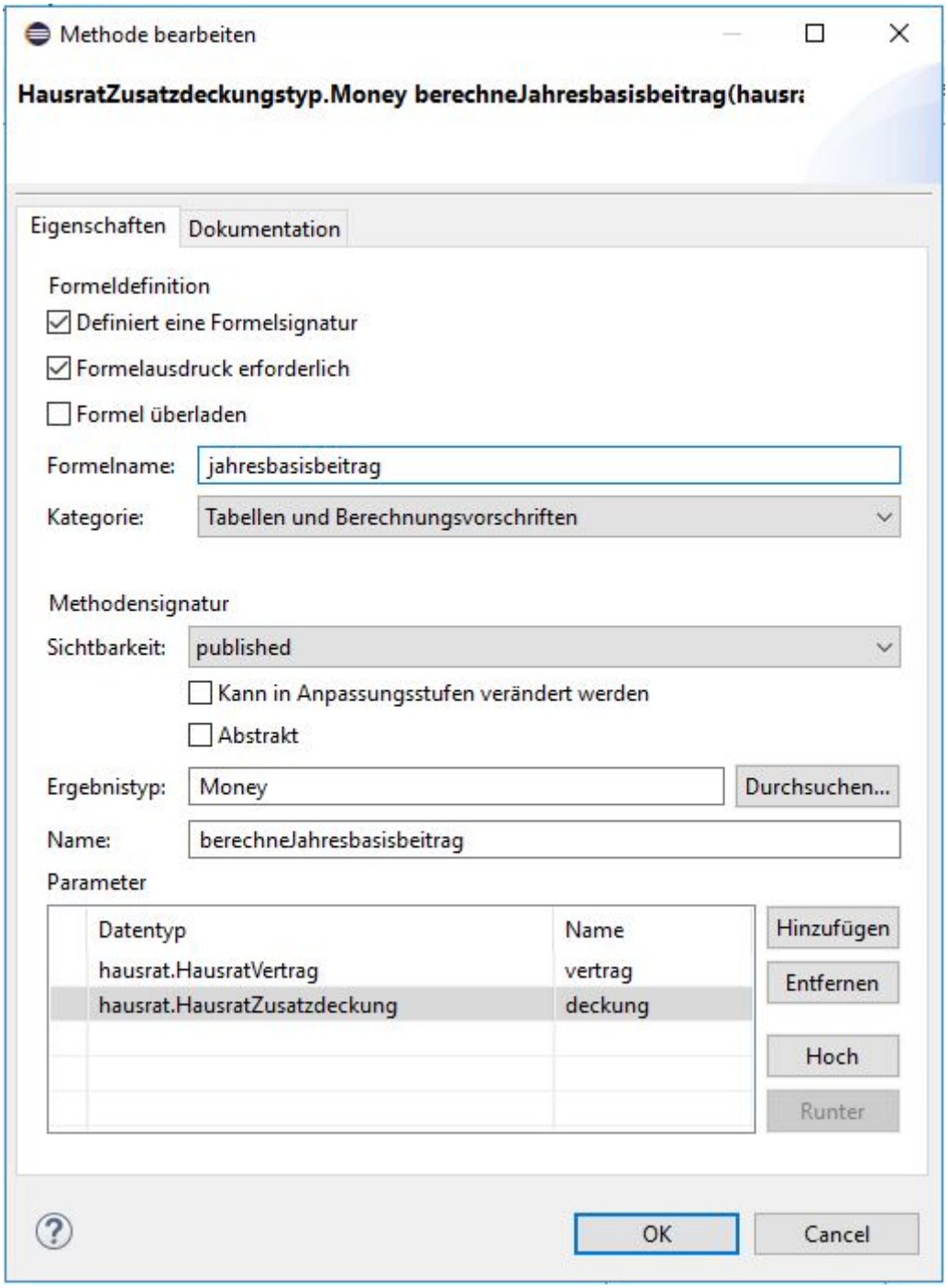

Figure 48. Dialog zur Definiton einer Formelsignatur

Schließen Sie den Dialog und speichern. In der Klasse `HausratZusatzdeckungstyp` befindet sich nun die Methode `berechneJahresbasisbeitrag`(...) zur Berechnung des Basisbeitrags.

Öffnen wir nun die Fahrraddiebstahldeckung, um die Formel für die Beitragsberechnung festzulegen. Beim Öffnen erscheint zunächst ein Dialog, in dem angezeigt wird, dass es im Modell eine neue Formel gibt, die es bisher nicht in der Produktdefinition gab.

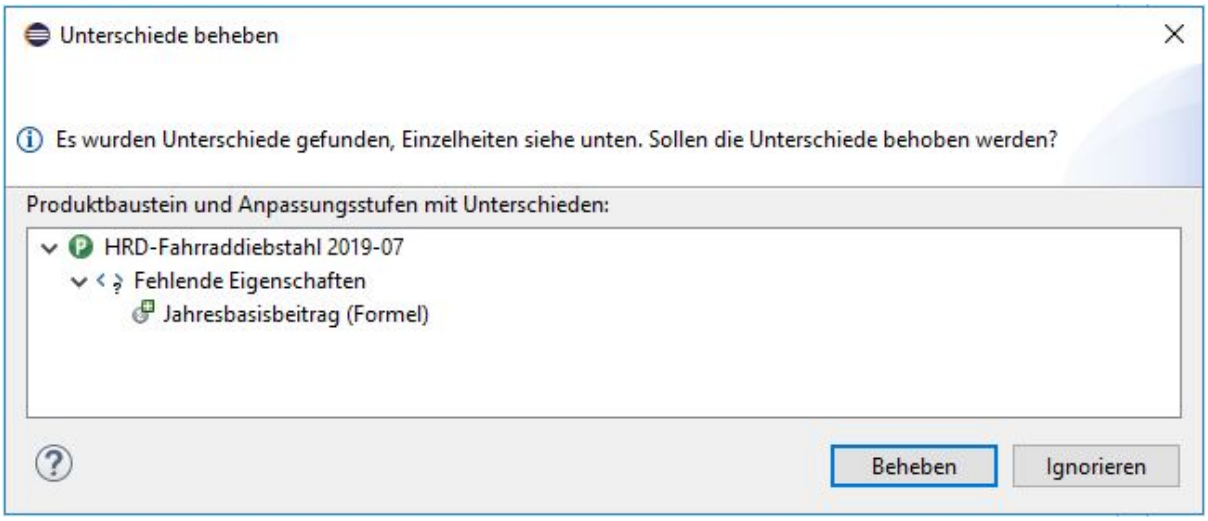

Figure 49. Dialog Unterschiede beheben

Bestätigen Sie mit Beheben, dass diese hinzugefügt werden soll. In dem Abschnitt *Tables & Formulas* (Tabellen und Berechungsvorschriften) wird nun die noch leere Formel für den Beitragssatz angezeigt. Klicken Sie auf den Button neben dem Formelfeld, um die Formel zu editieren. Es öffnet sich der folgende Dialog, in dem Sie die Formel bearbeiten können und in dem auch die verfügbaren Parameter angezeigt werden:

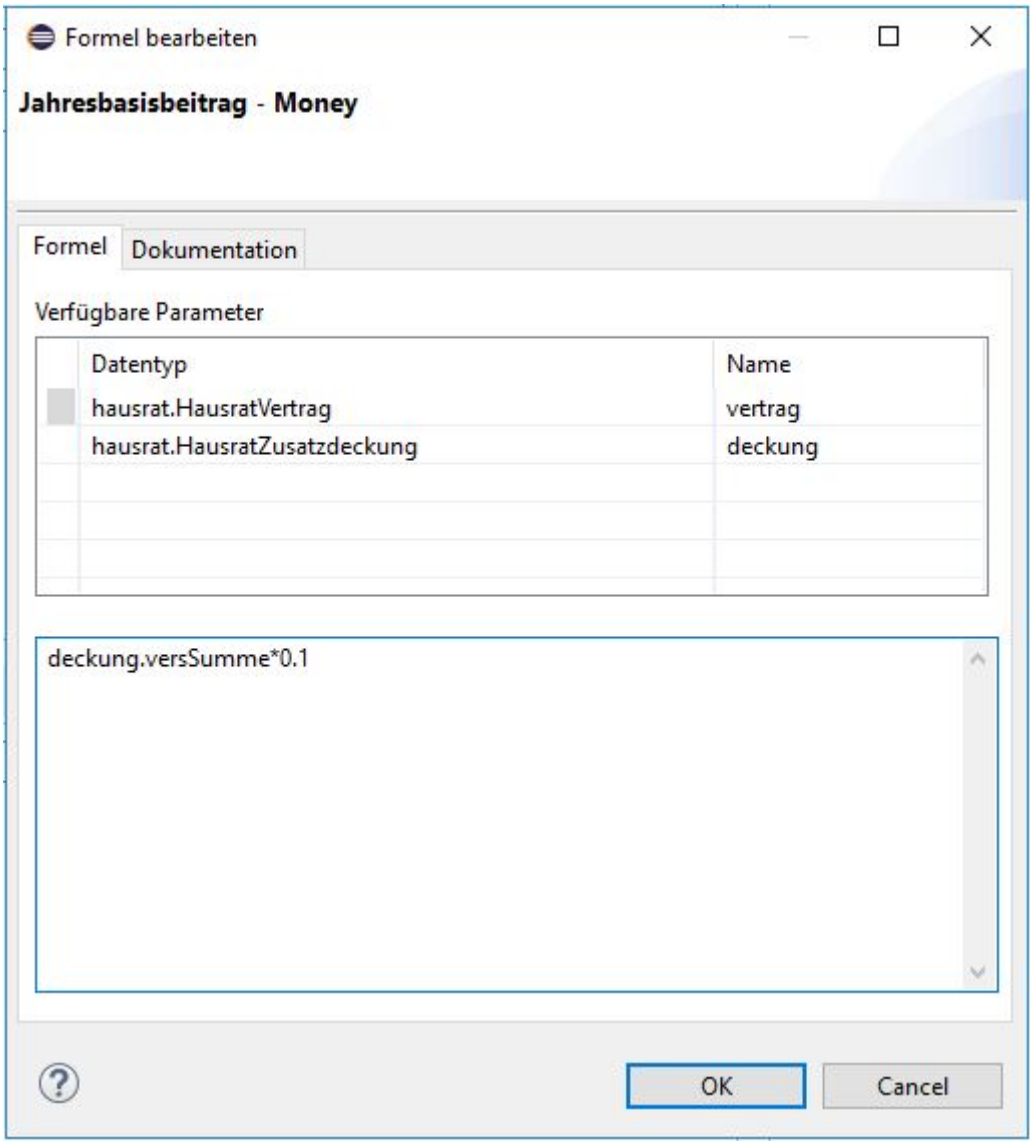

Figure 50. Anlegen einer Formel

Der Beitrag für die Fahrraddiebstahlversicherung soll 10% der Versicherungssumme der Zusatzdeckung betragen. Drücken Sie Strg-Space in der mittleren Eingabebox und Sie sehen die Parameter und Funktionen, die Sie zur Verfügung haben. Wählen Sie den Parameter „deckung“ aus und geben danach noch einen Punkt ein. Sie bekommen die Eigenschaften der Deckung zur Auswahl angeboten. Wählen Sie die „versSumme“. Nun multiplizieren Sie die Versicherungssumme noch mit 0.1.

Schließen Sie den Dialog und speichern den Baustein. Analog definieren Sie nun, dass der Beitrag für die Überspannungsdeckung 10Euro + 3% der Versicherungssumme beträgt (Formel: 10EUR + deckung.versSumme * 0.03).

Faktor-IPS hat nun die Subklassen für die beiden Produktbausteine mit der in Java Sourcecode übersetzten Formel generiert. Sie finden beide Klassen im Java-Sourcefolder „src/main/resources“ im Package `org.faktorips.tutorial.produktdaten.internal.deckungen` [4]. Der folgende Sourcecode enthält die generierte Methode

`berechneJahresbasisbeitrag`(...) für die Fahrraddiebstahldeckung.

[4] Da der Name von Produktbausteinen auch Blanks und Bindestriche enthalten kann, diese aber nicht in Java- Klassennamen erlaubt sind, wurden diese durch Unterstriche ersetzt. Konfigurieren können Sie die Ersetzung in der „.ipsproject“ Datei im Abschnitt ProductCmptNamingStrategy.

```
public Money berechneJahresbasisbeitrag(final HausratVertrag vertrag, final
HausratZusatzdeckung deckung) throws FormulaExecutionException {
    try {
        return deckung.getVersSumme().multiply(Decimal.valueOf("0.1"),
RoundingMode.HALF_UP);
    } catch (Exception e) {
        StringBuffer parameterValues = new StringBuffer();
        parameterValues.append("vertrag=");
        parameterValues.append(vertrag == null ? "null" : vertrag.toString());
        parameterValues.append(", ");
        parameterValues.append("deckung=");
        parameterValues.append(deckung == null ? "null" : deckung.toString());
        throw new FormulaExecutionException(toString(),
"deckung.versSumme*0.1", parameterValues.toString(), e);
    }
}
```

Tritt beim Ausführen der in Java übersetzten Formel ein Fehler auf, wird eine RuntimeException geworfen, die den Formeltext sowie die String-Repräsentation der übergebenen Parameter enthält.

Nun müssen wir noch dafür sorgen, dass die Formel im Rahmen der Beitragsberechnung auch aufgerufen wird, indem wir in der Klasse `HausratZusatzdeckung` die Methode `berechneJahresbasisbeitrag`() implementieren. Dies geht nun einfach, indem wir an die Berechnungsmethode im Zusatzdeckungstyp delegieren und als Parameter die Zusatzdeckung (this) und den Vetrag, zu dem die Deckung gehört, übergeben. Legen Sie in der Modellklasse *HausratZusatzdeckung* die Methode *berechneJahresbasisbeitrag* mit dem Rückgabewert Money, der Sichtbarkeit published und ohne Parameter an und speichern.

Öffnen Sie nun die Java-Klasse `HausratZusatzdeckung` und implementieren Sie die Methode wie folgt.

```
public Money berechneJahresbasisbeitrag() {
    return
getHausratZusatzdeckungstyp().berechneJahresbasisbeitrag(getHausratVertrag(),
this);
}
```


Damit der Beitrag der Zusatzdeckungen auch dem Gesamtbeitrag des Hausratvertrags zugeschlagen wird, erweitern wir noch die Beitragsberechnung in der Klasse Hausratvertrag:

```
private void berechneJahresbasisbeitrag() {
    jahresbasisbeitrag = Money.euro(0, 0);
    HausratGrunddeckung hausratGrunddeckung = getHausratGrunddeckung();

    hausratGrunddeckung.berechneJahresbasisbeitrag();
    jahresbasisbeitrag =
jahresbasisbeitrag.add(hausratGrunddeckung.getJahresbasisbeitrag());

    /*
     * Über Zusatzdeckungen iterieren und jeweils den Beitrag dem
Gesamtbeitrag
     * hinzuaddieren:
     */
    List<? extends HausratZusatzdeckung> zusatzdeckungen =
getHausratZusatzdeckungen();
    for (int i = 0; i < zusatzdeckungen.size(); i++) {
        HausratZusatzdeckung hausratZusatzdeckung = zusatzdeckungen.get(i);
        jahresbasisbeitrag =
jahresbasisbeitrag.add(hausratZusatzdeckung.berechneJahresbasisbeitrag());
    }
}
```

Zum Abschluss des Kapitels testen wir die neue Funktionalität wieder durch die Erweiterung des JUnit-Tests wie folgt.


```
@Test
public void testBerechneJahresbasisbeitragFahrraddiebstahl() {
    // Erzeugen eines hausratvertrags mit der Factorymethode des Produktes
    HausratVertrag vertrag = kompaktProdukt.createHausratVertrag();

    // Wirksamkeitsdatum des Vertrages setzen, damit dieProduktAnpassungsstufe
gefunden wird!
    // Dies muss nach dem Gueltigkeitsbeginn der Anpassungsstufe liegen!
    vertrag.setWirksamAb(new GregorianCalendar(2019, 7, 10));

    // Vertragsattribute setzen
    vertrag.setVersSumme(Money.euro(60000));

    // Zusatzdeckungstyp Fahrraddiebstahl holen
    // Der Einfachheit halber, nehmen wir hier an, der erste ist
Fahrraddiebstahl
    HausratZusatzdeckungstyp deckungstyp =
kompaktProdukt.getHausratZusatzdeckungstyp(0);

    // Zusatzdeckung erzeugen
    HausratZusatzdeckung deckung =
vertrag.newHausratZusatzdeckung(deckungstyp);

    // Jahresbasisbeitrag berechnen und testen
    deckung.berechneJahresbasisbeitrag();

    // Versicherungssumme der Deckung = 1% von 60.000, max 5.000 => 600
    // Beitrag = 10% von 600 = 60
    assertEquals(Money.euro(60, 0), deckung.berechneJahresbasisbeitrag());
}
```

In diesem zweiten Teil des Tutorials haben wir gesehen, wie Tabellen in Faktor-IPS genutzt werden, haben die Beitragsberechnung implementiert und getestet und am Beispiel der Zusatzdeckungen gesehen, wie man ein Modell so gestaltet, dass es flexibel erweitert werden kann.

Ein weiteres Tutorial zeigt, wie man mit den hier erstellten Modellklassen in einer operativen Anwendung arbeitet (Tutorial Hausrat Angebotssystem).

Im Tutorial zur Modellpartitionierung wird gezeigt, wie man komplexe Modelle in sinnvolle Teile zerlegt und damit umgeht. Insbesondere die Trennung in einzelne Sparten und die Trennung von spartenspezifischen und spartenübergreifenden Aspekten wird dort beispielhaft gezeigt.

Wie man beim Testen von Faktor-IPS unterstützt wird, zeigt das Tutorial Softwaretests mit

Faktor-IPS.

