

Datentypen & Aufzählungen in Faktor-IPS

Jan Ortmann
(Dokumentversion 51)

Einleitung

Dieser Artikel erläutert wie neue Datentypen in Faktor-IPS definiert werden können, die nicht zum Standardumfang von Faktor-IPS gehören. Nach der Klärung der konzeptionellen Grundlagen erläutern wir zunächst, wie einerseits vordefinierte Datentypen und andererseits existierende Java-Klassen in einem Projekt als Datentyp verwendet werden können. Im Anschluss gehen wir ausführlich auf die Definition von Aufzählungen ein.

Der Artikel ist für Softwareentwickler mit Erfahrung in objektorientierter Modellierung geschrieben. Um die Beispiele im Detail nachvollziehen zu können, sollte zudem die Faktor-IPS Einführungstutorials durchgearbeitet haben.

Konzeptionelle Grundlagen

Was ist ein Datentyp in Faktor-IPS?

Programmiersprachen bieten i.d.R. eine Menge an vordefinierten Datentypen, beispielsweise für ganze Zahlen, Gleitkommazahlen und Zeichenketten. Die Definition eines Datentyps in einer Programmiersprache besteht neben dem Namen des Datentyps aus der Festlegung der erlaubten Werte und der zulässigen Operationen¹.

Java bietet als vordefinierte Datentypen die primitiven Typen

- boolean,
- die „Integral Types“ byte, short, int, long, char,
- sowie die „Floating Point Types“ float und double.

Alle anderen Typen werden in Java als Reference Types bezeichnet. Gemäß der Java Spezifikation sind damit sowohl die Wrapper-Klassen Boolean, Integer, etc. als auch Klassen wie String und Date Reference Types².

In der OO-Programmierung werden Klassen (bzw. deren Instanzen) häufig in Value Objects und Reference Objects unterteilt. Bei Value Objects ist die Gleichheit zwischen zwei Objekten über die Wertgleichheit ihrer Eigenschaften definiert, die Objekte haben keine Identität³. Ein typisches Beispiel ist eine Klasse Money zur Abbildung von Geldbeträgen. Zwei Money-Objekte sind gleich, wenn Währung und Betrag gleich sein. Dagegen gelten zum Beispiel zwei Personen-Objekte bei denen Name, Vorname und Geburtsdatum gleich sind, nicht automatisch als gleich, da es sich trotzdem um unterschiedliche Personen handeln kann.

¹ <http://de.wikipedia.org/wiki/Datentyp>

² http://java.sun.com/docs/books/jls/third_edition/html/typesValues.html

³ s. Fowler, Patterns of Enterprise Application Architecture, S. 486

In der UML wird der Begriff Datentyp im Sinne von Value Object verwendet⁴. Analog verwenden wir den Begriff in Faktor-IPS. Bezogen auf Java können damit alle primitiven Typen und alle Klassen, die eine Anwendung des Value Object Pattern sind, als Datentyp in Faktor-IPS verwendet werden.

Aufzählungen

Eine spezielle Art Datentyp sind Aufzählungen (Enumerationen/Enums)⁵. Bei Aufzählung wird der Wertebereich – wie der Name schon sagt – durch Aufzählen der einzelnen Werte definiert. Zum Beispiel kann man die möglichen Zahlweisen aufzählen:

- Monatlich
- Quartalsweise
- Halbjährlich
- Jährlich
- Einmalzahlung

Häufig werden die einzelne Werte von Aufzählungen im Programmcode referenziert. Möchte man zum Beispiel aus dem Beitrag gemäß Zahlweise den Jahresbeitrag ermitteln, könnte dies wie folgt aussehen:

```
public Double berechneJahresbeitrag() {  
    if (zahlweise==Zahlweise.EINMALZAHLUNG) {  
        return null;  
    }  
    if (zahlweise==Zahlweise.JAEHRLICH) {  
        return beitragZw;  
    }  
    if (zahlweise==Zahlweise.MONATLICH) {  
        return beitragZw * 12;  
    }  
    throw new RuntimeException("Unbekannte Zahlweise");  
}
```

Dieser Programmcode ist abhängig von den konkreten Werte der Aufzählung⁶. Das hat zur Folge, dass das Löschen oder Hinzufügen von Werten mit Änderungen am Programmcode verbunden ist.

Bei diesem Beispiel kann man die Abhängigkeit des Programmcodes von den konkreten Werten vermeiden, indem man zur Zahlweise ein Attribut *anzahlZahlungenProJahr* hinzufügt. Die obige Methode lässt sich dann wie folgt implementieren:

4 „A data type is a type whose instances are identified only by their value. A DataType may contain attributes to support the modeling of structured data types. A typical use of data types would be to represent programming language primitive types or CORBA basic types. For example, integer and string types are often treated as data types.“. UML Infrastructure Specification, v2.1.1, Seite 134.

5 Manche mögen nun behaupten, dass die Überschrift des Artikels nicht 100% richtig ist. Sie ist so gewählt, da die Beschreibung der Definition von Aufzählungen den größten Teil des Artikels ausmacht :-)

6 Im Falle von numerischen Werten spricht man auch von „Magic Numbers“.

```
public Double berechneJahresbeitrag() {  
    if (zahlweise.getAnzahlZahlungenProJahr() == null) {  
        return null;  
    }  
    return beitragZw * zahlweise.getAnzahlZahlungenProJahr();  
}
```

Der Programmcode ist nun unabhängig von den konkreten Werten. Man kann eine wöchentliche Zahlweise ohne Änderung des Programmcodes hinzufügen, und genauso die monatliche Zahlweise entfernen.

Verwendung vordefinierter Datentypen

Faktor-IPS enthält standardmäßig Datentypen für die meisten primitiven Typen von Java und deren Wrapperklassen sowie für `String`, `Date` und `GregorianCalendar`. Darüber hinaus gibt es einen Datentyp `Money` zum Abbilden von Geldbeträgen und eine spezielle `Decimal`-Implementierung.

Welche dieser Datentypen in einem Projekt verwendet werden können, wird in der „ipsproject“-Datei festgelegt. Bevor wir einen Blick in die Datei werfen können, benötigen wir ein neues Faktor-IPS Projekt. Legen Sie hierzu zunächst ein Java-Projekt an und fügen dann im Package-Explorer die Faktor-IPS Nature hinzu⁷. Öffnen Sie nun die „ipsproject“-Datei. Wenn Sie diese nicht direkt sehen, liegt das daran, dass der Package-Explorer standardmäßig keine Dateien anzeigt, die mit einem Punkt beginnen.

Im unteren Teil des XMLs finden sie das Element `<Datatypes>`, mit dem die im Projekt erlaubten Datentypen festgelegt werden. Bei der Anlage eines neuen Projekten werden zunächst alle vordefinierten Datentypen als im Projekt verwendbar eingetragen. Möchte man bestimmte Datentypen ausschließen, zum Beispiel weil man bei Berechnungen Fließkommazahlen nicht zulassen möchte, dann kann man einfach den entsprechenden Eintrag entfernen.

```
<Datatypes>  
  <UsedPredefinedDatatypes>  
    <Datatype id="Double"/>  
    <Datatype id="Long"/>  
    <Datatype id="Money"/>  
    <Datatype id="Decimal"/>  
    <Datatype id="Date"/>  
    <Datatype id="String"/>  
    ...  
  </UsedPredefinedDatatypes>  
  <DatatypeDefinitions/>  
</Datatypes>
```

Um das zu veranschaulichen, legen Sie zunächst eine neue Vertragsklasse an und ein neues Attribut mit dem Namen *beitrag*. Wenn Sie nun im Dialog zur Bearbeitung des Attributes auf den Browse... Button neben dem Datatype Feld klicken, sehen Sie die verfügbaren Datentypen.

⁷ Die Details sind im Einführungstutorial beschrieben.

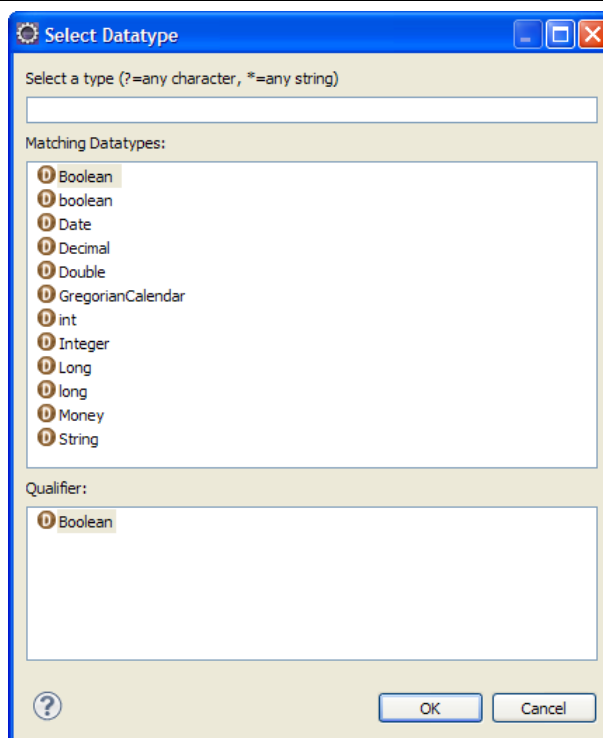


Abbildung 1: Dialog zur Auswahl eines Datentyps

Wählen Sie *Double* aus, schließen die Dialoge und speichern die Klasse. Löschen Sie nun in der „ipsproject“-Datei die Zeile, die die Verwendung des vordefinierten Datentyps *Double* im Projekt erlaubt. Speichern Sie die „ipsproject“-Datei. Das Attribut *beitrag* wird nun als fehlerhaft markiert. Wenn Sie den Dialog mit den Datentypen erneut öffnen, wird dort *Double* nicht mehr aufgeführt.

In diesem Kapitel haben wir bisher immer von den in Faktor-IPS vordefinierten Datentypen gesprochen. Dies ist nur die halbe Wahrheit. Tatsächlich verwendet Faktor-IPS den Eclipse Extension Point Mechanismus zur Definition von Datentypen. Das heißt andere PlugIns können ebenfalls (Faktor-IPS) Datentypen definieren. Hat ein Entwickler neben Faktor-IPS auch solch ein weiteres PlugIn installiert, so kann er auch die von diesem PlugIn definierten Datentypen verwenden. Aus Sicht des Entwicklers gibt es dann letztendlich keinen Unterschied zwischen einem von Faktor-IPS vordefinierten Datentyp und einem Datentyp, welcher durch ein anderes PlugIn zur Verfügung gestellt wird. Die Verwendung dieses Mechanismus setzt natürlich Kenntnisse über die Entwicklung von Eclipse-PlugIns voraus. Da die Einführung eines neuen Datentyps in diesem Szenario immer mit dem Rollout des PlugIns verbunden ist, lohnt es sich nur, wenn es sich um grundlegende Datentypen handelt, die unternehmensweit in mehreren Projekten eingesetzt werden oder wenn Faktor-IPS zur Entwicklung von Standardsoftware eingesetzt wird⁸. In einem einzelnen Projekt geht es einfacher mit dem im folgenden Kapitel beschriebenen Verfahren.

⁸ Die genaue Beschreibung würde den Rahmen dieses Artikels sprengen. Da die Faktor-IPS eigenen Datentypen ebenfalls über den Mechanismus registriert werden, können sich Entwickler mit Erfahrung in der Entwicklung von PlugIns die Definition der entsprechenden Extensions im PlugIn „org.faktorips.devtools.core“ ansehen. Der Extension Point hat die ID „datatypeDefinition“.

Verwendung von Java-Klassen als Datentyp

Neben den vordefinierten Datentypen können auch Java-Klassen zur Verwendung als Datentyp registriert werden. Als Beispiel soll uns dabei eine Klasse `IsoDate` dienen, die ein Datum auf Basis des im ISO-Standard 8601 definierten Formats (YYYY-MM-DD) darstellt. Im Gegensatz zu den Klassen `Date` und `GregorianCalendar` soll die Klasse wirklich ein Datum und keinen Zeitpunkt darstellen. Der folgenden Sourcecode stellt eine minimale Implementierung dieser Idee dar⁹.

Wenn Sie das Beispiel im Detail durchführen wollen, dann erzeugen Sie zunächst ein neues Java-Projekt (es braucht kein Faktor-IPS Projekt zu sein!) und legen die Klasse `IsoDate` im Package `enums` an. Nehmen Sie dieses neue Projekt in den Java Buildpath Ihres ersten Projektes auf. Die Begründung wieso die Klasse nicht im selben Projekt liegen darf, erfolgt weiter unten.

```
public class IsoDate {

    private int year;
    private int month;
    private int day;

    public IsoDate(int year, int month, int day) {
        this.year = year;
        this.month = month;
        this.day = day;
    }

    public int getYear() {
        return year;
    }

    public int getMonth() {
        return month;
    }

    public int getDay() {
        return day;
    }

    public String toString() {
        String m = month < 10 ? "0" + month : "" + month;
        String d = day < 10 ? "0" + day : "" + day;
        return year + '-' + m + '-' + d;
    }

}
```

Bevor wir uns damit beschäftigen wie wir `IsoDate` als Datentyp registrieren können, betrachten wir, was für Anforderungen Faktor-IPS an einen Datentyp hat. Öffnen wir hierzu noch einmal den Dialog für das Attribut *beitrag*. Da wir `Double` aus der Liste der Datentypen entfernt haben, wählen

⁹ Damit die Klasse `IsoDate` auch wirklich das Value Object Pattern realisiert, müssen natürlich noch die Methoden `equals()` und `hashCode()` entsprechend überschrieben werden. Auf die Darstellung wird hier verzichtet. Mit Eclipse können diese Methoden aber über das Menü **Source** ► **Generate hashCode() and equals()...** leicht generiert werden.

wir jetzt *Money* als Datentyp. Wechseln Sie nun auf die zweite Tabseite und geben als Defaultwert 42 ein. Faktor-IPS zeigt eine Fehlermeldung an, dass 42 kein *Money* ist.

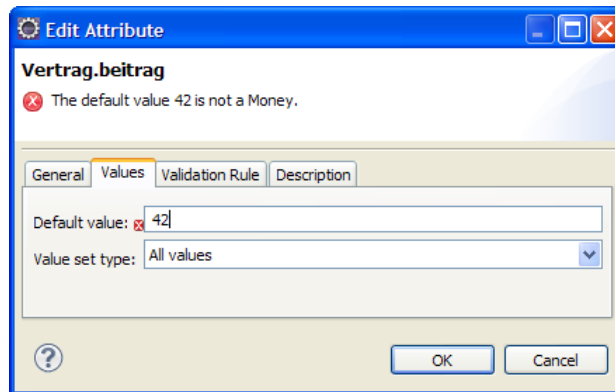


Abbildung 2: Gültigkeitsprüfung eines Wertes gegen den Datentyp im Dialog für Attribute

Faktor-IPS muss also für einen Datentyp prüfen können, ob ein gegebener String einen gültigen Wert des Datentyps darstellt. Korrigieren Sie nun den Defaultwert auf 42EUR und speichern die Klasse. Der Codegenerator erzeugt folgenden Code für das Attribut:

```
import org.faktorips.values.Money;

private Money beitrag = Money.valueOf("42EUR");

public Money getBeitrag() {
    return beitrag;
}

public void setBeitrag(Money newValue) {
    this.beitrag = newValue;
}
```

Neben dem qualifizierten Klassennamen, muss der Codegenerator also wissen, wie er aus der String-Repräsentation 42EUR eine Instanz der Klasse erzeugt. Methoden, die dies leisten, heißen im JDK meistens `valueOf(...)`, z. B. bei `Integer` und `Double`.

Wir fügen so eine Methode zu unserer `IsoDate` Klasse hinzu. Falls der String nicht geparsed werden kann, wirft unsere Methode analog zu den Methoden im JDK eine `Exception`.

```
public static final IsoDate valueOf(String s) {
    if (s==null || s.equals("")) {
        return null;
    }
    StringTokenizer tokenizer = new StringTokenizer(s, "-");
    int year = Integer.parseInt(tokenizer.nextToken());
    int month = Integer.parseInt(tokenizer.nextToken());
    int date = Integer.parseInt(tokenizer.nextToken());
    return new IsoDate(year, month, date);
}
```

Das reicht uns bereits um die Java-Klasse `IsoDate` in Faktor-IPS als Datentyp zu verwenden. In der „ipsproject“-Datei definieren wir den Datentyp im Element `<DatatypeDefinitions>` wie folgt:

```
<Datatypes>
  <UsedPredefinedDatatypes>
    ... unverändert
  </UsedPredefinedDatatypes>
  <DatatypeDefinitions>
    <Datatype
      id="IsoDate"
      javaClass="enums.IsoDate"
      valueOfMethod="valueOf"/>
    </DatatypeDefinitions>
</Datatypes>
```

Beim Eingeben muss man darauf achten, das `<DatatypeDefinitions>` Element richtig zu öffnen und abzuschließen, da vor dem Einfügen des Datentyps die Kurzform `<DatatypeDefinitions/>` verwendet wird!

Die Bedeutung der Attribute ist wie folgt:

- **ID**
Die in Faktor-IPS verwendete Identifikation des Datentyps.
- **javaClass**
Der qualifizierte Java-Klassenname.
- **valueOfMethod**
Der Name der Methode, die eine String-Repräsentation in eine Instanz der Java-Klasse parsed.

Speichern Sie die „ipsproject“-Datei. Sollte ein Fehler in der Definition vorliegen, wird ein entsprechender Problemmarker angezeigt. Dies ist zum Beispiel der Fall, wenn die Klasse nicht gefunden wird und liegt entweder daran, dass der Java Buildpath nicht richtig definiert ist, oder der Klassenname falsch geschrieben wurde.

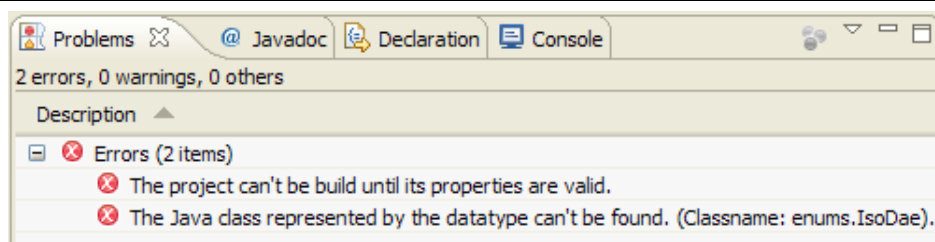


Abbildung 3: Problemview falls die Javaklasse des Datentyps nicht gefunden wird

Ist die Definition korrekt, kann der Datentyp sofort im Modell verwendet werden. Legen Sie an der Vertragsklasse ein weiteres Attribut *versicherungsbeginn* an. In der Liste der Datentypen muss jetzt auch *IsoDate* erscheinen. Wenn Sie einen ungültigen Defaultwert wie zum Beispiel 42 eingeben erhalten Sie eine Fehlermeldung während Werte wie 2010-01-01 akzeptiert werden.

Nach dem Speichern der Vertragsklasse sollte in der Implementierungsklasse folgender zusätzlicher Sourcecode generiert worden sein.

```
import enums.IsoDate;

private IsoDate versicherungsbeginn = IsoDate.valueOf("2010-01-01");

public IsoDate getVersicherungsbeginn() {
    return versicherungsbeginn;
}

public void setVersicherungsbeginn(IsoDate newValue) {
    this.versicherungsbeginn = newValue;
}
```

Faktor-IPS prüft die Gültigkeit der eingegebenen Werte, indem die Java-Klasse `IsoDate` geladen und per Reflection die `valueOf(...)` Methode aufgerufen wird. Aus diesem Grund muss die Klasse im Java-Buildpath des Projektes verfügbar sein. Das ist zwar gegeben, wenn der Sourcecode direkt im Modellprojekt liegt, der Clean-Build von Eclipse bereitet allerdings ein Problem. Beim Clean-Build werden von Eclipse zunächst alle Java Classfiles des Projektes gelöscht, insbesondere natürlich auch das File „`IsoDate.class`“. Wenn danach der Codegenerator den Sourcecode erzeugt, wird auch der Datentyp `IsoDate` benötigt. Dieser ist in diesem Augenblick aber nicht valide, da die Java-Klasse („`IsoDate.class`“) nicht mehr gefunden werden kann. Aus diesem Grund müssen Klassen, die als Datentyp verwendet werden, immer in eigenen Projekten verwaltet werden oder in Jar-Files zur Verfügung gestellt werden.

Bevor wir beschreiben wie mit Faktor-IPS Aufzählungen definiert werden können, erläutern wir im Rest des Kapitels noch einige Details bzgl. der Verwendung von Java-Klassen als Datentyp. Dabei handelt es sich aber um weiterführende Informationen, die auch gerne übersprungen werden können.

Zur Prüfung ob ein String einen Wert des Datentyps darstellt, kann zusätzlich eine Methode angegeben werden, die `true` zurück liefert, wenn der String geparsed werden kann und ansonsten

false. Das XML-Attribut zur Angabe des Methodennamen heißt `isParsableMethod`.

Java-Klassen, die eine Aufzählung definieren, können ebenfalls über den Mechanismus als Datentyp verwendet werden. Hierzu muss das XML-Attribut `isEnumType="true"` gesetzt werden und die folgenden zusätzlichen Informationen angegeben werden.

XML-Attribut	Erläuterung
<code>getAllValuesMethod</code>	Der Name der statischen Methode, die alle Werte der Aufzählung zurückgeben kann. Die zurückgegebenen Werte sind Instanzen der Klasse, die als Datentyp registriert werden soll.
<code>isSupportingNames</code>	true, falls es eine Methode gibt, die zu einem Wert eine für den Benutzer verständliche Bezeichnung zurückgeben kann. Dies sollte bei Aufzählungen eigentlich immer der Fall sein. Ansonsten false.
<code>getNameMethod</code>	Der Name der -nicht statischen- Methode, die für den Wert (this) die Bezeichnung zurückgeben kann.

Da mit der Version 2.3 die Möglichkeit besteht, Aufzählungen über Editoren zu definieren, gehen wir auf diesen Aspekt nicht weiter ein.

Neben der Definition von Datentypen im eigentlichen Sinne, können an dieser Stelle (konzeptionell vielleicht etwas unsauber) auch Java-Klassen registrieren werden, die nicht als Datentyp (für Attribute) verwendet werden, sondern als Typ für Parameter und Rückgabetypen von Methoden. So könnte man zum Beispiel die Faktor-IPS Klasse `MessageList` wie folgt zur Verwendung in Methoden zulassen.

```
<Datatype
  id="MessageList"
  javaClass="org.faktorips.runtime.MessageList"
  valueObject="false"/>
```

Definition von Aufzählungen

Bei der Definition von Aufzählungen in Faktor-IPS unterscheiden wir explizit zwischen

- Aufzählungen, bei denen die Werte Teil des Modells bzw. des Programms sind und
- Aufzählungen, bei denen die Werte Teil der Konfiguration sind und das Modell/Programm unabhängig von den einzelnen Werten sind.

Umgesetzt ist dies durch die Trennung in Aufzählungstypen (Enumeration Types) und den Inhalt von Aufzählungen (Enumeration Contents). Im Aufzählungstyp werden die Attribute festgelegt. Optional kann der Aufzählungstyp auch die Werte enthalten. In diesem Fall sind die Werte Teil des Modells und können im Programmcode referenziert werden. Der Codegenerator generiert hierfür dann – je nach Java Version und Einstellungen – einen entsprechenden Java-Enum mit einem Literal pro Wert der Aufzählung.

Die konkreten Werte der Ausprägung können aber auch separat in einem eigenen Aufzählungsinhalt verwaltet werden. In diesem Fall generiert Faktor-IPS lediglich eine Klassen mit den entsprechenden Zugriffsmethoden für die Attribute. Zur Laufzeit können natürlich die Werte eines Aufzählungstyps abgefragt werden, es gibt aber keine generierten Literale (bzw. Konstanten). Im Programmcode sollte es keine Vergleiche auf konkrete einzelne Werte geben.

In einer zukünftigen Version von Faktor-IPS wird es zusätzlich die Möglichkeit geben, Werte, die Teil des Modells sind direkt im Aufzählungstyp zu definieren und diese in der Konfiguration über einen Aufzählungsinhalt zu erweitern.

Definition von Aufzählungstypen mit Werten

Bei den Sourcecode-Beispielen in diesem Artikel gehen wir von der Verwendung eines JRE 5 oder höher aus, so dass der Codegenerator standardmäßig Java Enums verwendet. Wer dies nicht möchte kann dies in den Projekteigenschaften unter Faktor-IPS Code Generator ausschalten.

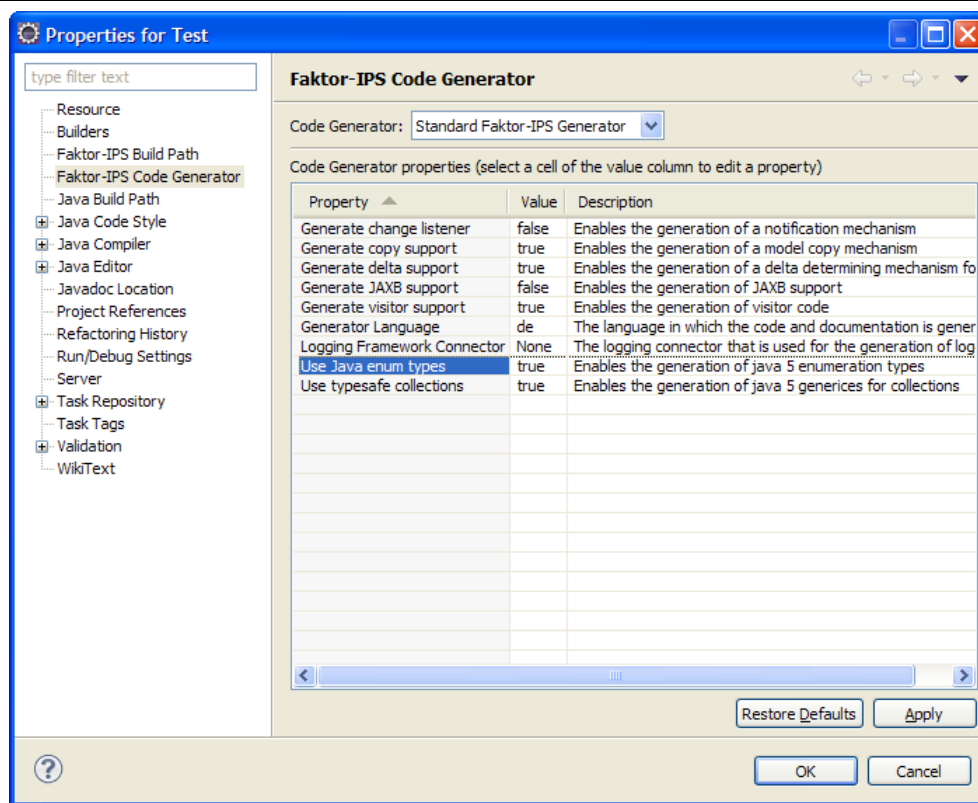


Abbildung 4: Codegenerator-Optionen


Als ersten Aufzählungstyp werden wir die Zahlweise wie folgt definieren:

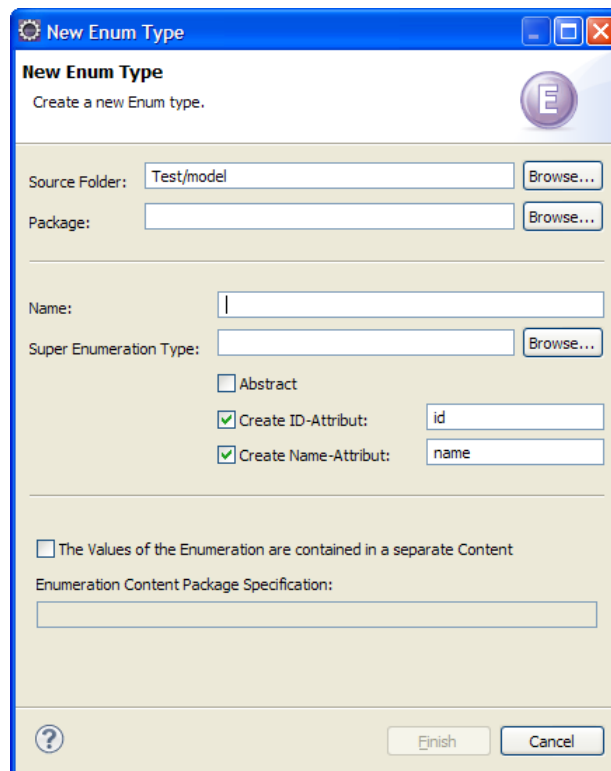
ID	Name	Anzahl Zahlungen Pro Jahr
m	Monatlich	12
q	Quartalsweise	4
h	Halbjährlich	2
j	Jährlich	1
e	Einmalzahlung	<null>

Tabelle 1: Zahlweisen

Aufzählungen in Faktor-IPS können beliebig viele Attribute haben. Eines der Attribute muss als (default) ID gekennzeichnet werden. Die ID wird Faktor-IPS intern zur Identifikation verwendet. Darüber hinaus kann sie an anderen Stellen zur Identifikation verwendet werden, zum Beispiel bei der Persistierung in Datenbanken. Da hierfür u.U. ein anderes Attribut als Identifier verwendet werden soll, bezeichnen wir das von Faktor-IPS verwendete Attribut als default Identifier.

Darüber hinaus muss eines der Attribute als Name (Bezeichnung) gekennzeichnet werden. Dieses Attribut wird in Faktor-IPS zur Anzeige der Werte verwendet, z. B. bei der Festlegung von Defaultwerten von Attributen oder im Editor für Produktbausteine.

Markieren Sie nun im Package-Explorer den Modellordner des Projektes und klicken auf  in der Toolbar. Es öffnet sich der folgende Wizard.



The image shows a 'New Enum Type' wizard dialog box. It has a title bar with a gear icon and the text 'New Enum Type'. Below the title bar, it says 'New Enum Type' and 'Create a new Enum type.' with an enum icon. The dialog contains several input fields and checkboxes. 'Source Folder' is set to 'Test/model' with a 'Browse...' button. 'Package' is empty with a 'Browse...' button. 'Name' is empty. 'Super Enumeration Type' is empty with a 'Browse...' button. There are three checkboxes: 'Abstract' (unchecked), 'Create ID-Attribut' (checked), and 'Create Name-Attribut' (checked). The 'Create ID-Attribut' checkbox has an input field with 'id'. The 'Create Name-Attribut' checkbox has an input field with 'name'. At the bottom, there is a checkbox 'The Values of the Enumeration are contained in a separate Content' (unchecked) and a text field 'Enumeration Content Package Specification:'. At the very bottom are 'Finish' and 'Cancel' buttons.

Abbildung 5: Wizard zum Anlegen eines neuen Aufzählungstyps

Geben Sie nun den Namen „Zahlweise“ ein. Die Felder Super Type und abstract betrachten wir später. Die beiden Checkboxes Create ID-Attribute und Create Name-Attribute sind standardmäßig angehakt, so dass gleich die beiden benötigten Attribute (s.o.) erzeugt werden.

Die einzelnen Zahlweisen sollen im ersten Schritt direkt im Aufzählungstyp definiert werden. Aus diesem Grund bleibt die Checkbox The Values of the Enumeration are contained in a separate Content abgehakt.

Klicken Sie nun auf Finish, um den Aufzählungstyp zu erzeugen. Es öffnet sich der Editor zur Bearbeitung des Aufzählungstyps.

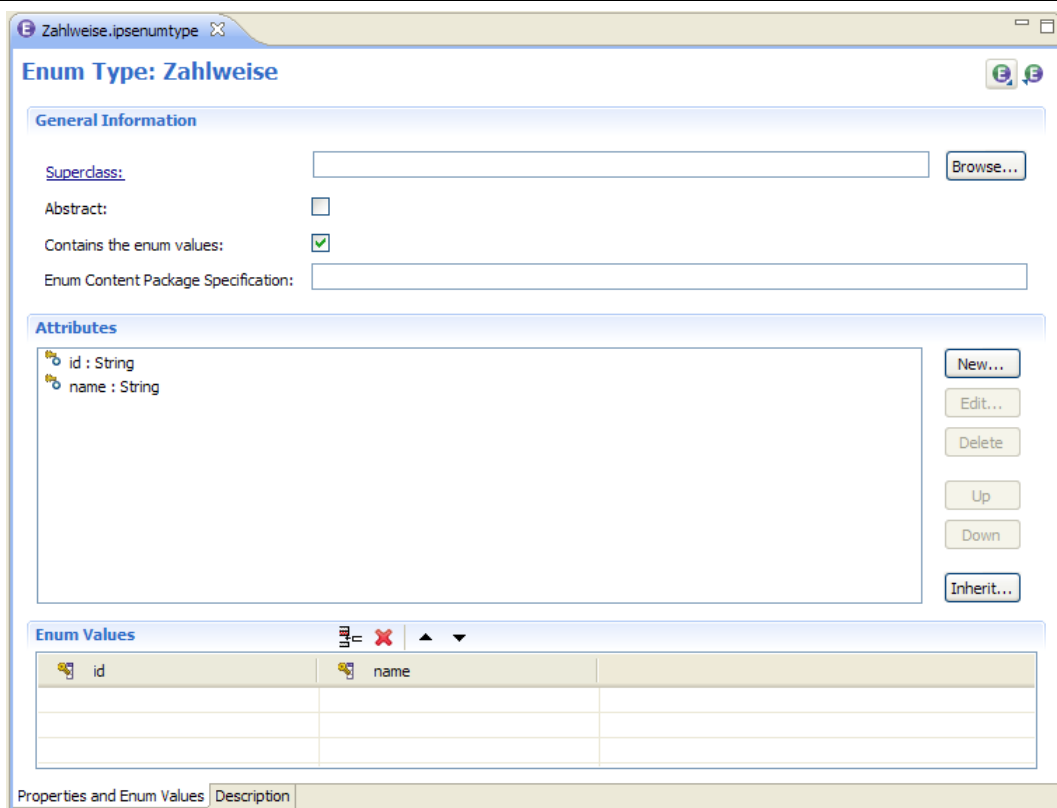


Abbildung 6: Editor für Aufzählungstypen

In dem Abschnitt General Information stehen die Informationen aus dem Wizard. In dem Abschnitt Attributes die Attribute des Aufzählungstypen. Der Wizard hat hier die beiden Attribute *id* und *name* angelegt. Als erstes fügen wir nun das Attribut *anzahlZahlungenProJahr* hinzu. Nach Klicken auf New... öffnet sich der folgende Dialog.

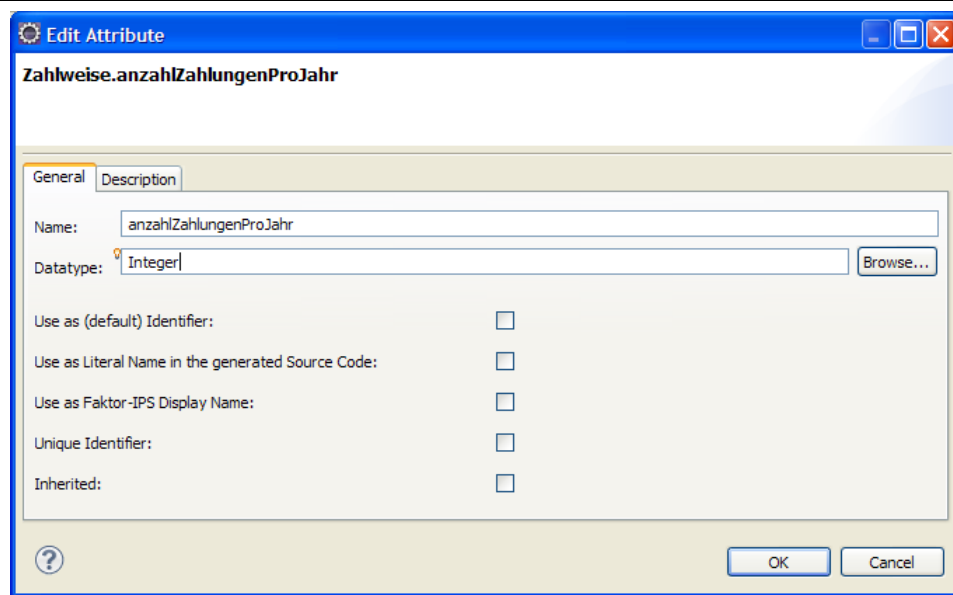



Abbildung 7: Dialog zum Bearbeiten eines Attributes

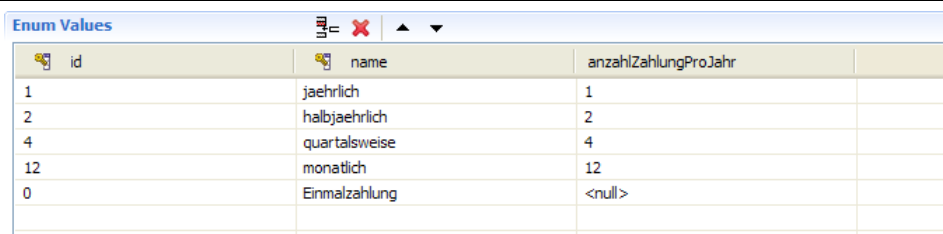
Geben Sie den Namen und den Datentyp des Attributs gemäß der Abbildung ein. Die folgende Tabelle beschreibt die Bedeutung der Checkboxes:

Use as (default) Identifier	Das Attribut wird als Identifier intern in Faktor-IPS verwendet. Verwendet man die Option, dass der Sourcecode JAXB Annotationen generiert, so wird bei Aufzählungstypen mit separatem Aufzählungsinhalt diese Ids im XML abgelegt.
Use as Literal Name in the generated Source Code	Das Attribut wird als Name der Literale bzw. Konstanten im generierte Sourcecode verwendet.
Use as Faktor-IPS Display Name	Das Attribut wird zur Anzeige der Werte in Faktor-IPS verwendet.
Unique Identifier	Die Werte dieses Attribut sind eindeutig über alle Werte der Aufzählung. Der Wert kann hierdurch also identifiziert werden. Der Codegenerator generiert für solche Attribute eine entsprechende Zugriffsmethode <code>getValueByAttributeName(...)</code> .
Inherited	Kennzeichnet das Attribut als vom Supertype geerbt. Dazu später mehr.

Für das neu angelegt Attribut *anzahlAnZahlungenProJahr* bleiben die Checkboxes alle abgehakt. Standardmäßig hat der Wizard das Attribute *id* als default Identifier markiert und das Attribute

name als Literal Name und Display Name (nicht sehr überraschend).

Nach dem Schließen des Dialogs mit OK sind nun in dem Abschnitt Enum Values drei Spalten mit den Namen der Attribute zu sehen. Durch Drücken auf  wird eine Zeile für den ersten Aufzählungswert angelegt. Die weiteren Zeilen werden beim Tabben automatisch angelegt. Geben Sie nun die Zahlweisen gemäß Tabelle 1 ein. Das Resultat sollte wie folgt aussehen.



id	name	anzahlZahlungProJahr
1	jaehrlich	1
2	halbjaehrlich	2
4	quartalsweise	4
12	monatlich	12
0	Einmalzahlung	<null>

Abbildung 8: Aufzählungswerte für Zahlweise

Nach dem Speichern hat Faktor-IPS den folgenden Sourcecode generiert. Der Übersichtlichkeit halber sind in dem folgenden Abschnitt die Javadocs entfernt und lediglich der Code für die Werte Jährlich und Einmalzahlung abgebildet.

```
public enum Zahlweise implements IEnumValue {

    JAEHRLICH("1", "jaehrlich", new Integer(1)),
    EINMALZAHLUNG("0", "Einmalzahlung", null);

    private final String id;
    private final String name;
    private final Integer anzahlZahlungenProJahr;

    private Zahlweise(String id, String name, Integer anzahlZahlungenProJahr) {
        this.id = id;
        this.name = name;
        this.anzahlZahlungenProJahr = anzahlZahlungenProJahr;
    }

    public String getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public Integer getAnzahlZahlungenProJahr() {
        return anzahlZahlungenProJahr;
    }

    @Override
    public String toString() {
        return "Zahlweise: " + name;
    }
}
```

Für jedes Attribut wurde eine entsprechende Membervariable und Gettermethode generiert. Für jeden Aufzählungswert ein entsprechendes Literal.

Darüber hinaus wird für jedes als eindeutig gekennzeichnete Attribut zwei Methoden generiert. Die erste ermittelt den Aufzählungswert zu einer ID und die zweite Methode prüft, ob eine gegebene ID einen Aufzählungswert identifiziert. Der folgende Codeabschnitt zeigt dies für das Attribut *id*. Analog wird der Code in dem Beispiel für das Attribut *name* generiert.

```
public static final Zahlweise getValueById(String id) {
    if (id == null) { return null; }
    if (id.equals("1")) { return JAEHRLICH; }
    if (id.equals("0")) { return EINMALZAHLUNG; }
    return null;
}

public static final boolean isValueById(String id) {
    return getValueById(id) != null;
}
```

Darüber hinaus wird noch die Methode `getEnumValueId()` generiert, um das Interface `IEnumValue` zu implementieren. Dies wird von der Faktor-IPS Runtime benötigt.


Separate Definition der Aufzählungswerte

Als Beispiel verwenden wir die Art des Parkplatzes. In der KFZ-Versicherung wird die Art des in der Nacht verwendeten Parkplatzes häufig als Risikomerkmakl verwendet. Die folgende Tabelle enthält die zulässigen Werte für die Parkplatzart.

ID	Name
1	Straße
2	Öffentliches Parkhaus
3	Stellplatz vor Eigenheim

Tabelle 2: Parkplatzarten

Da es nicht erforderlich ist, im Programmcode auf einzelne Parkplatzarten abzufragen und wir der Fachabteilung ermöglichen wollen, die Liste der Parkplatzarten zu pflegen, definieren wir diesmal, dass die Werte des Aufzählungstyps *Parkplatzart* in einem separaten Aufzählungsinhalt verwaltet werden.

Markieren Sie nun wieder den Modellordner im Package-Explorer und starten über  den Wizard, um einen neuen Aufzählungstyp anzulegen. In dem Dialog haken Sie diesmal die Checkbox *The Values of the Enumeration are contained in a separate Content an* und vergeben noch einen Namen für die Aufzählungsinhalt, i.d.R. bietet es sich an ihn identisch zum Aufzählungstypen zu nennen. Die folgende Abbildung zeigt den ausgefüllten Dialog.

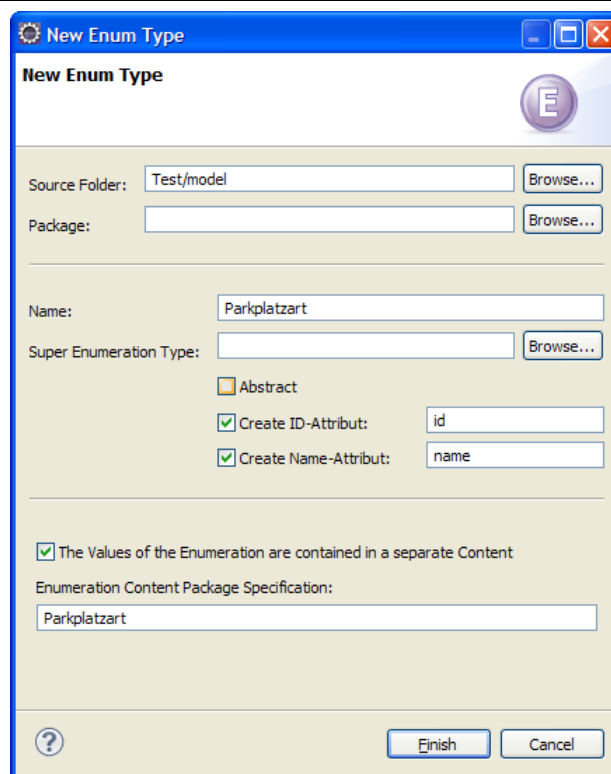


Abbildung 9: Ausgefüllter Wizard zum Anlegen der Parkplatzart

Nach dem Klicken von Finish öffnet sich wieder der Editor zur Definition des Aufzählungstyps. Der Abschnitt Enum Values ist jetzt ausgegraut, da die Werte nun nicht mehr direkt im Aufzählungstyp definiert werden. Da der Aufzählungstyp Parkplatzart keine weiteren Attribute hat, sind wir mit der Definition des Typs auch schon fertig.

Der folgende Abschnitt zeigt den generierten Sourcecode, wieder ohne Javadoc. Da die Werte im Modell nicht bekannt sind, wird statt einem Java Enum eine Klasse generiert. Für jedes Attribut wird wieder eine Membervariable und eine Gettermethode generiert. Der protected Konstruktor wird von der Faktor-IPS Runtime (per Reflection) verwendet, der public Konstruktor kann zum Beispiel in Testfällen verwendet werden.

```

public final class Parkplatzart implements IEnumValue, Serializable {

    public static final long serialVersionUID = 1L;

    private final String id;
    private final String name;

    protected Parkplatzart(
        List<String> enumValues,
        IRuntimeRepository repository) {

        this.id = enumValues.get(0);
        this.name = enumValues.get(1);
    }

    public Parkplatzart(String id, String name) {
        this.id = id;
        this.name = name;
    }

    public String getId() {
        return id;
    }

    public String getName() {
        return name;
    }


    @Override
    public String toString() {
        return "Parkplatzart: " + id;
    }

    public Object getID() {
        return id;
    }

}

```

Bevor wir die Frage klären, wie wir programmatisch an die Werte kommen, definieren wir erst einmal die konkreten Parkplatzarten. In diesem Artikel legen wir den Aufzählungsinhalt im gleichen Projekt an wie den Aufzählungstyp. Damit eine Fachabteilung die Werte unabhängig vom Modell bearbeiten kann, würde man den Aufzählungsinhalt in der Regel in einem Projekt anlegen, in dem ausschließlich die Produktdaten verwaltet werden, und nicht direkt im Modellprojekt¹⁰.

Den Aufzählungsinhalt legen wir durch Klicken auf  in der Toolbar an. Falls der Sourcefolder nicht vorbelegt ist, wählen Sie den Sourcefolder (i.d.R. Projektname/model) aus. Ansonsten brauchen Sie lediglich noch den Aufzählungstyp auszuwählen. Das Package, in dem der Inhalt gespeichert wird, wird durch den Aufzählungstyp vorgegeben.

¹⁰ Details können dem Einführungstutorial und dem Tutorial zur Modellpartitionierung entnommen werden.

// TODO Abbildung nach Umbau einfügen

Abbildung 10: Wizard zum Anlegen eines neuen Aufzählungsinhalts

Nach dem Klicken von Finish öffnet sich der Editor zum Bearbeiten der Aufzählungswerte. Dies funktioniert genauso wie im Editor für Aufzählungstypen.

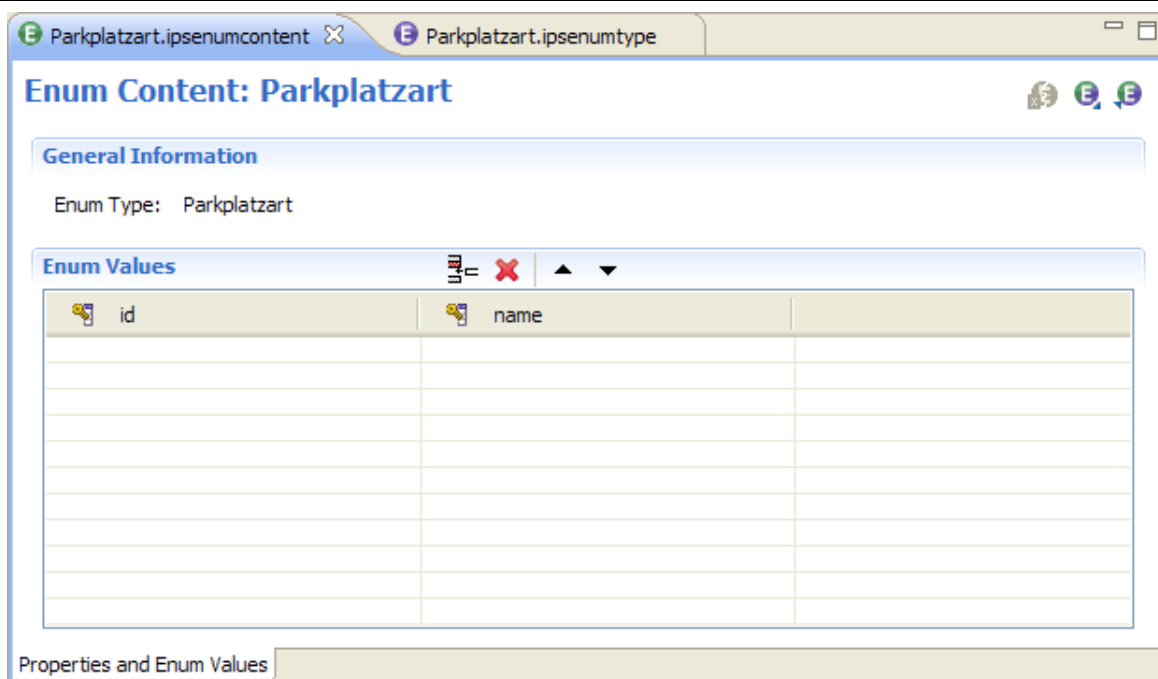


Abbildung 11: Editor für Aufzählungswerte

Geben Sie nun die Parkplatzarten gemäß Tabelle 2 ein und speichern Sie.

Für die Parkplatzarten wird nun – natürlich - kein Java Sourcecode generiert, da der Programmcode unabhängig von den konkreten Werten sein soll. Damit auf die Aufzählungswerte zur Laufzeit zugegriffen werden kann, hat der Generator statt dessen ein XML-File mit den Aufzählungswerten im Java Sourcefolder „derived“ angelegt.

Die Werte eines Aufzählungstyps kann man über das Faktor-IPS `RuntimeRepository` über die Methode `getEnumValues(Class<T> enumType)` abfragen. Der folgende Sourcecode gibt die Parkplatzarten auf der Konsole aus.

```
public static void main(String[] args) {
    // Repository erzeugen
    IRuntimeRepository repository =
        ClassloaderRuntimeRepository.create(
            "org/faktorips/sample/model/internal/faktorips-repository-toc.xml");

    // Parkplatzarten aus dem Repository abfragen ...
    List<Parkplatzart> arten = repository.getEnumValues(Parkplatzart.class);

    // ... und auf der Konsole ausgeben
    for (Parkplatzart parkplatzart : arten) {
        System.out.println(parkplatzart.toString());
    }
}
```

Die Ausgabe des Programm sieht wie folgt aus:

```
Parkplatzart: Straße(id1)
Parkplatzart: Öffentliches Parkhaus(id2)
Parkplatzart: Stellplatz vor Eigenheim(id3)
```

Intuitiver wäre die Ermittlung der Werte eines Aufzählungstyps über eine statische Methode an der generierten Klasse, die dann an das Repository delegiert, wie folgt:

```
public final static List<Parkplatzart>getAllValues(IRuntimeRepository rp) {
    return rp.getEnumValues(Parkplatzart.class);
}
```

Wenn diese Aufzählungsklassen aber zur Remote-Kommunikation über Data Transfer Objects verwendet werden, müssen Sie dazu auch auf der Clientseite bekannt sein. Das RuntimeRepository steht dagegen im Client nicht zur Verfügung. Aus diesem Grund darf es keine Compileabhängigkeit der Aufzählungsklassen zum RuntimeRepository geben.

Verwendung von Ableitungen

In Applikationen gibt es häufig Aspekte die für alle Aufzählungen gleich funktionieren, zum Beispiel die Anzeige der Werte einer Aufzählung in einer Listbox. Um dies umzusetzen ist es hilfreich, wenn alle Aufzählungen ein einheitliches Interface implementieren bzw. von einer gemeinsamen Basisklasse ableiten. Für diese Aspekte ist es dabei in der Regel nicht relevant, ob die Werte Teil des Modells oder Teil der Konfiguration sind. Beide Arten von Aufzählungen sollten gleich behandelt werden können.

Wir erweitern also unsere bisherigen Beispiele um einen Basistyp, den wir *AbstractEnumValue* nennen. Dies zeigt das folgende Diagramm.

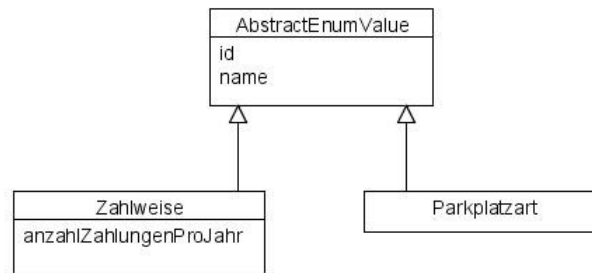
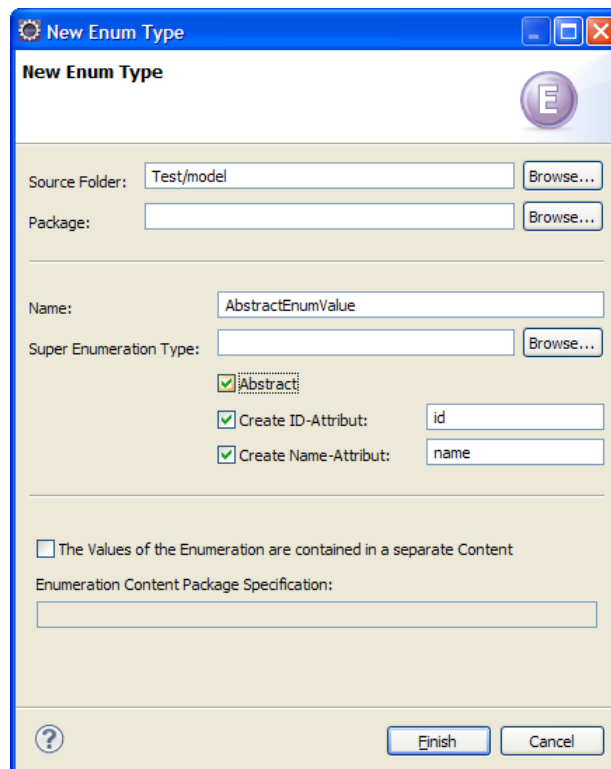


Abbildung 12: Ableitungshierarchie der Aufzählungstypen

Legen Sie nun bitte den Aufzählungstyp *AbstractEnumValue* gemäß der folgenden Abbildung an.



Achten Sie darauf direkt die Checkbox `Abstract` anzuhaken. Wenn Sie die Abstrakt-Eigenschaft nachträglich im Editor ändern, müssen Sie vorher den generierten Sourcecode für *AbstractEnumValue* löschen. Das von Faktor-IPS verwendete Framework zum Mergen des Sourcecodes (JMerge) kann mit dieser Änderung nicht umgehen.

Faktor-IPS generiert für den abstrakten Aufzählungstyp das folgende Interface:

```
public interface AbstractEnumValue extends IEnumValue {  
  
    public String getId();  
  
    public String getName();  
  
}
```

Die Generierung eines Interfaces ist erforderlich, da Java-Enums lediglich Interfaces implementieren können, aber nicht von anderen Enums abgeleitet werden können.

Nun müssen wir noch in den bestehenden Aufzählungstypen *AbstractEnumValue* als Supertyp eintragen. Öffnen Sie zunächst den Editor für die Zahlweise. Im Abschnitt General Information können Sie den Supertyp angeben. Bei den beiden Attributen *id* und *name* müssen wir nun noch markieren, dass die Definition aus dem Supertyp geerbt wird. Hierzu öffnen sie den Dialog und haken die Checkbox *inherited* an.

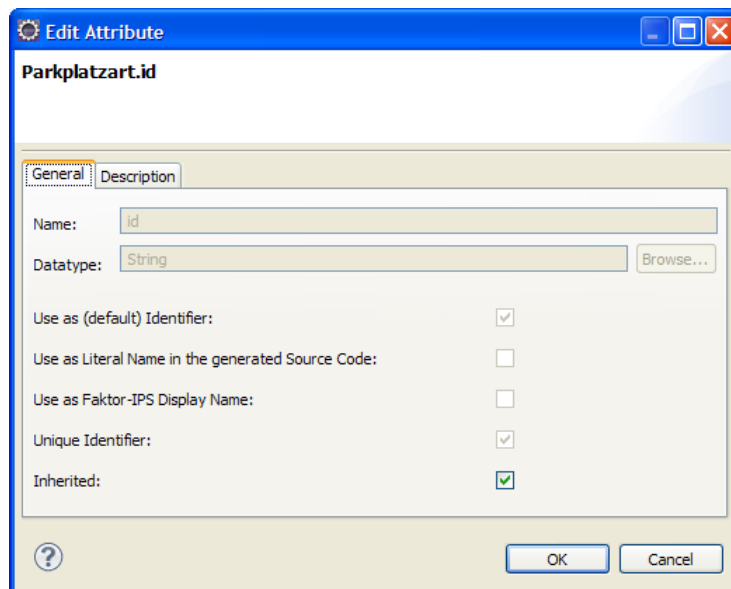


Abbildung 13: Markierung eines Attributes als geerbt

Nachdem Sie beide Attribute als geerbt markiert und gespeichert haben, implementiert der generierte Java-Enum *Zahlweise* das Interface *AbstractEnumValue*. Der Rest des Codes für *Zahlweise* bleibt unverändert.

```
public enum Zahlweise implements IEnumValue, AbstractEnumValue {  
    ... wie bisher  
}
```

Leiten Sie nun analog den Aufzählungstyp *Parkplatzart* von *AbstractEnumValue* ab. Danach

implementiert nun auch die Java-Klasse `Parkplatzart` das Interface `AbstractEnumValue`.

```
public final class Parkplatzart implements IEnumValue, AbstractEnumValue,
    Serializable {

    .. wie bisher
}
```

Nun können wir Aspekte, die für alle Aufzählungen gleich sind, unter Verwendung von `AbstractEnumValue` implementieren. Der folgende Codeabschnitt führt das Beispiel zur Ausgabe der `Parkplatzarten` fort und gibt nun auch die `Zahlweisen` auf der Konsole aus. Die Ausgabe der Werte auf der Konsole ist in eine Methode `printValues(...)` ausgelagert worden.

```
public static void main(String[] args) {
    // Repository erzeugen
    IRuntimeRepository repository =
        ClassloaderRuntimeRepository.create(
            "org/faktorips/sample/model/internal/faktorips-repository-toc.xml");

    // Parkplatzarten aus dem Repository abfragen ...
    List<Parkplatzart> arten = repository.getEnumValues(Parkplatzart.class);

    // ... und auf der Konsole ausgeben
    printValues(arten);

    // Zahlweisen auf der Konsole ausgeben
    List<Zahlweise> zahlweisen = Arrays.asList(Zahlweise.values());
    printValues(zahlweisen);
}

public static void printValues(List<? extends AbstractEnumValue> values) {
    for (AbstractEnumValue value : values) {
        System.out.println(value.toString());
    }
}
```

Das nachträgliche Einrichten der Ableitungsbeziehung ist etwas lästig, da man die Attribute nachträglich als abgeleitet kennzeichnen muss. In der Regel legt man aber weitere Aufzählungstypen auf Basis des bestehenden *AbstractEnumValue* an. Gibt man den Supertyp direkt im Wizard zum Erzeugen eines neuen Aufzählungstypen an, dann werden auch die entsprechenden Attribute aus dem Supertyp erzeugt.