

258321 DKE Projekt

**Gruppe 2**

**Teammitglieder:**

k01607605, Aistleithner Andrea

k01256561, Dusanic Maja

k01356577, Teuchtmann Alexander

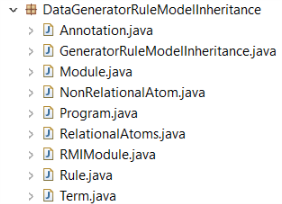
k01356229, Tomic Milos

Implementierung

# Datengenerator Rule Model Inheritance

Nach dem konzeptuellen Entwurf existiert eine (einzige) Klasse, die alle Algorithmen beinhaltet. Bei der Implementierung wurde diese eine Klasse durch eine Reihe von verschiedenen Klassen ersetzt. Der Grund für diese Entscheidung war die Erkenntnis, dass sich die Implementierung durch eine (einzige) Klasse, als zu kompliziert herausgestellt hat. Dementsprechend wurde von der Implementierung durch eine (einzige) Klasse, auf die Implementierung durch verschiedene Klassen (Objekte) umgestellt. Es wurde damit eine Klassenstruktur erstellt, die (hauptsätzlich) als ein Gerüst für die Erstellung eines Moduls dient.

* 1. **RMI Klassenstruktur**

****

**1.1.1 (Hilfs) Klasse – Generator Rule Modell Inheritance**

* **Bedeutung:**

Diese Klasse wurde zum Testen vom generierten Meta-Code im Yupiter benutzt. Sie enthält verschiedene Methoden, die einen Meta-Code ohne Logik generieren. Dementsprechend wurde damit nur die Syntax im Yupiter getestet. Es wurde die Syntax von Regeln, (nicht) relationalen Atomen, Annotationen und Terms getestet.

Rund um diese Klasse wurden im Nachhinein andere Klassen erstellt, die für den logisch basierten Meta-Code zuständig waren. Diese Klassen haben den Code einiger Methoden kopiert und für den eigenen Gebrauch entsprechend adaptiert.

**1.1.2 (Hilfs) Klasse – Non Relational Atom**

* **Bedeutung:**

Diese Klasse wurde in Verbindung mit der (obigen) Klasse Generator Rule Modell Inheritance benutzt, um einen Meta-Code ohne Logik zu generieren. Es wurde damit nur die Syntax im Yupiter getestet.

**1.1.3 Klasse – RMI Module**

* **Bedeutung**

Dies ist die zentrale Klasse des Programms. Sie vereint alle anderen Klassen, die einen Teil des Moduls darstellen, und generiert dadurch einen logisch basierten Meta-Code.

* **Zentrale Methoden**
* public String generateRMIModule(int rules, int facts, int in, int out);

Diese Methode generiert den Meta-Code vom RMI Modul basierend auf der eingegebenen Anzahl der Regeln, Fakten, Input- und Outputparameter. Das RMI Modul wird so erstellt, dass zuerst ein neues Objekt der Klasse Modul erstellt wird. Dieses wird anschließend mit Objekten der Anderen Klassen (Program, RelationalAtom, Annotation, Term) befüllt. Die angesprochenen Objekte werden mithilfe von verschiedenen Hilfsmethoden zur Verfügung gestellt.

* public String generateRMIModule(int rules, int facts, int in, int out, Module myModule)

Diese Methode generiert den Meta-Code eines geerbten RMI Moduls basierend auf der eingegebenen Anzahl der Regeln, Fakten, Input- und Outputparameter und der Superklasse. Das RMI Modul wird so erstellt, dass zuerst ein neues Objekt der Klasse Modul erstellt wird. Dieses wird anschließend mit Objekten der Anderen Klassen (Program, RelationalAtom, Annotation, Term) befüllt. Die angesprochenen Objekte werden mithilfe von verschiedenen Hilfsmethoden zur Verfügung gestellt. Im Unterschied zur oberen Methode wird hier noch zusätzlich die Verbindung zwischen der Superklasse und der Subklasse generiert.

Die Restlichen größeren Methoden in der Klasse RMI Module haben ähnliche Algorithmen deren Bedeutung aus dem Code herausgelesen werden kann.

**1.1.4 Klasse – Module**

* **Bedeutung**

Diese Klasse dient als Basis für das RMI Modul. Sie enthält verschiedene Attribute u.a. Listen von Regeln, relationalen Atomen und Annotationen. Diese Attribute werden durch entsprechende Setter/Getter-Methoden behandelt.

**1.1.5 Klasse – Program**

* **Bedeutung**

Diese Klasse dient nur als Behälter für den Namen vom entsprechenden Programm.

**1.1.6 Klasse – Relational Atom**

* **Bedeutung**

Diese Klasse dient zur Erstellung von relationalen Atomen, die eine Liste von Terms beinhalten. Die Objekte werden in verschiedenen Methoden der Klasse RMI Modul generiert und zum Modul hinzugefügt.

**1.1.7 Klasse – Annotation**

* **Bedeutung**

Diese Klasse dient zur Erstellung von Annotationen, die Listen von Input- und Outputpredikaten beinhalten. Die Objekte werden in verschiedenen Methoden der Klasse RMI Modul generiert und zum Modul hinzugefügt.

**1.1.8 Klasse – Term**

* **Bedeutung**

Diese Klasse dient zur Erstellung von Terms, die einen Namen und eine Serialisierung besitzten.

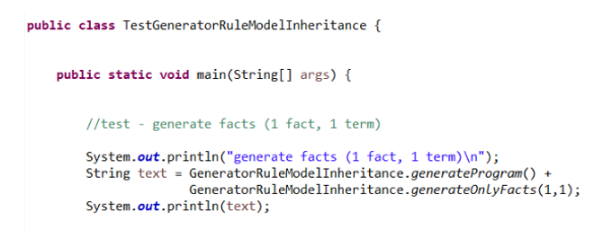
* 1. **Testen**

Im Folgenden wird erläutert wie der Meta-Code mit und ohne der Logik getestet wurde.

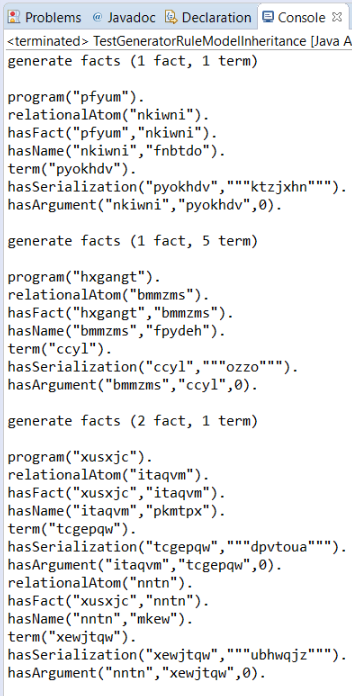
* + 1. **Testen der Syntax (ohne Logik)**

Diese Klasse „Test Generator Rule Modell Inheritance“ wurde benutzt, um die Syntax (ohne Logik) vom Meta-Code im Yupiter zu prüfen.

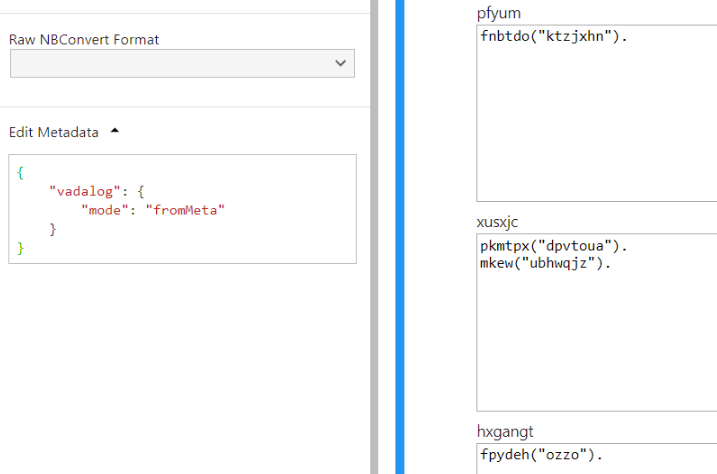
Ausschnitt aus der Klasse:



Ausschnitt aus der Konsole:

****

Ausschnitt aus Yupiter:

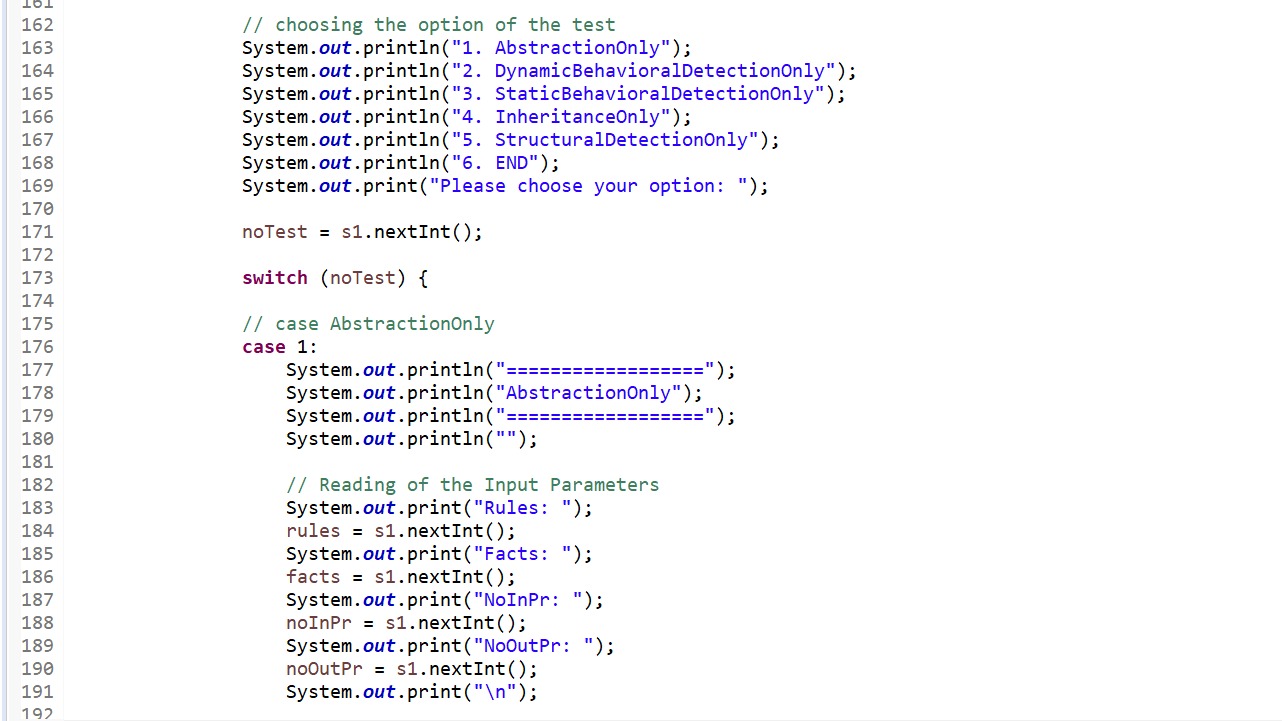


* + 1. **Testen der Syntax und der Logik**

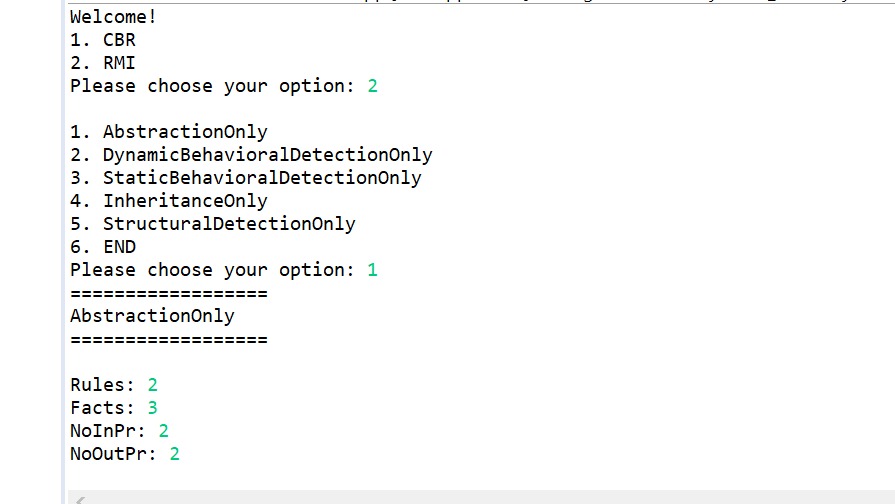
Diese Klasse „Evaluierung Framework App Test Generator Rule Modell Inheritance“ wurde benutzt, um die Syntax und die Logik vom Meta-Code im Yupiter zu prüfen.

Im Folgenden werden Ausschnitte aus dem Testen von 1. AbstractionOnly gezeigt.

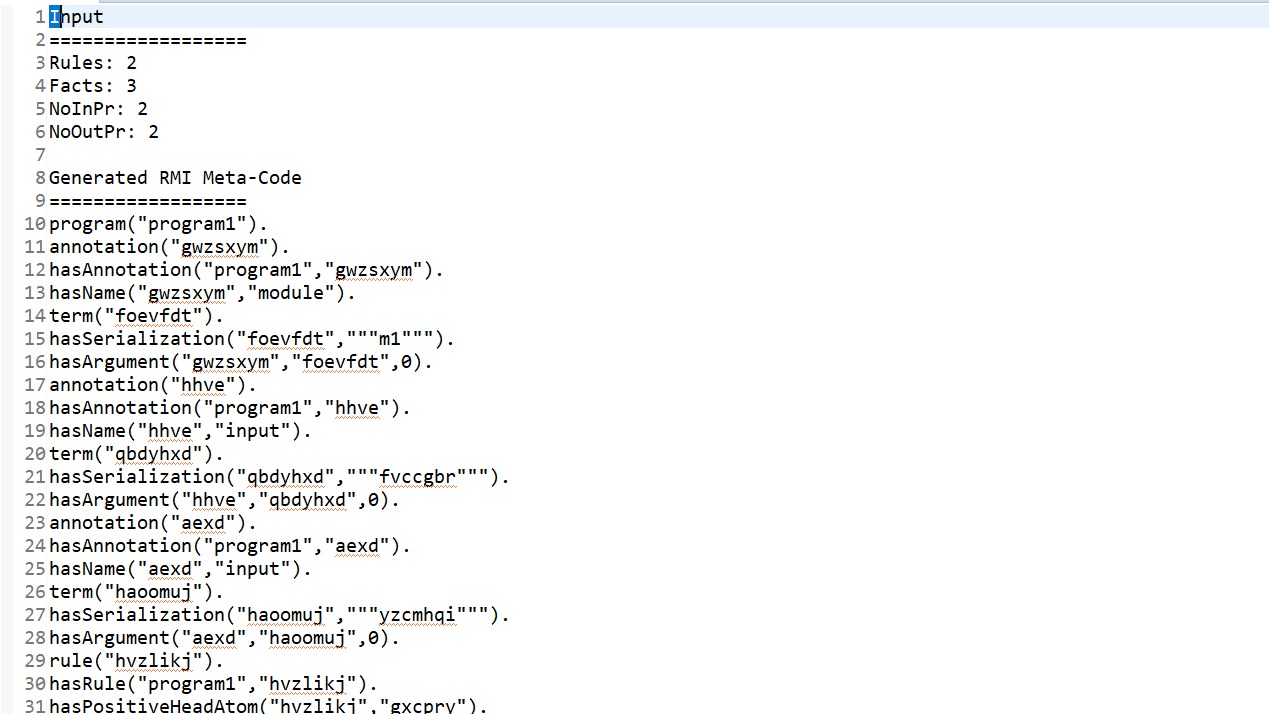
Ausschnitt aus der Klasse:



Ausschnitt aus der Konsole:



Ausschnit aus der Datei, wo der Meta-Code gespeichert wird:



Ausschnitt aus dem Yupiter:



# Abbildungsverzeichnis