Kundenpräsentation: Lexer und Symboltabelle
SWP Übersetzerbau Sommer 2010

von René Kijewski

Institut für Informatik

FU Berlin

Agenda

- Übernommene Aufgaben und Programmierstil
- Grundlegende Datentypen
- Verarbeiten der XML-Eingaben
- Verarbeiten der Ausdrücke
- Die Symboltabelle
- Fazit

Agenda

- Übernommene Aufgaben und Programmierstil
- Grundlegende Datentypen
- Verarbeiten der XML-Eingaben
- Verarbeiten der Ausdrücke
- Die Symboltabelle
- Fazit

Übernommene Aufgaben

- Allgemeine Programmieraufgaben
  - Nicht auf Quell- oder Zielsprache spezialisiert
  - Augenmerk auf hohe Wiederverwendbarkeit
  - Stark modularisiert
  - Strikte Trennung zwischen Interface und Implementierung
  - Somit:
    - Einzelteile einfach ersetzbar
    - Fehler einfach auffindbar

Programmierstil

- Allgemeine Programmieraufgaben

- Augenmerk auf Verwendung von Entwurfsmustern
  - Verringern der Einarbeitungszeit
  - Factory-Pattern
    - Nur ein Einstiegspunkt, jener aber wohldefiniert
    - Statisches Interface
    - Dadurch konkrete Implementierung ersetzbar
  - Iterator-Pattern
    - Eingabe wird zu einem „Strom an Daten“
    - Nur zwei Methoden müssen verwendet werden:
      - `hasNext()` – existieren weitere Daten?
      - `next()` – ein Datum einlesen
  - Beide Entwurfsmuster in der OOP weit verbreitet

Agenda

- Übernommene Aufgaben und Programmierstil
- Grundlegende Datentypen
- Verarbeiten der XML-Eingaben
- Verarbeiten der Ausdrücke
- Die Symboltabelle
- Fazit

Grundlegende Datentypen

- Grundlagentypen ziehen sich durch das gesamte Projekt hindurch

- Die „Stream-Position“
  - Konstrukte (z.B. Schleifen und Bezeichner) kennen jeweils Ort, an dem sie in der Eingabe standen
  - Realisiert durch:
    - *StreamPosition:* allgemeines Interface
    - *PositionBean:* Implementierung der *StreamPosition*
    - *PositionString:* String der seine Position „kennt“

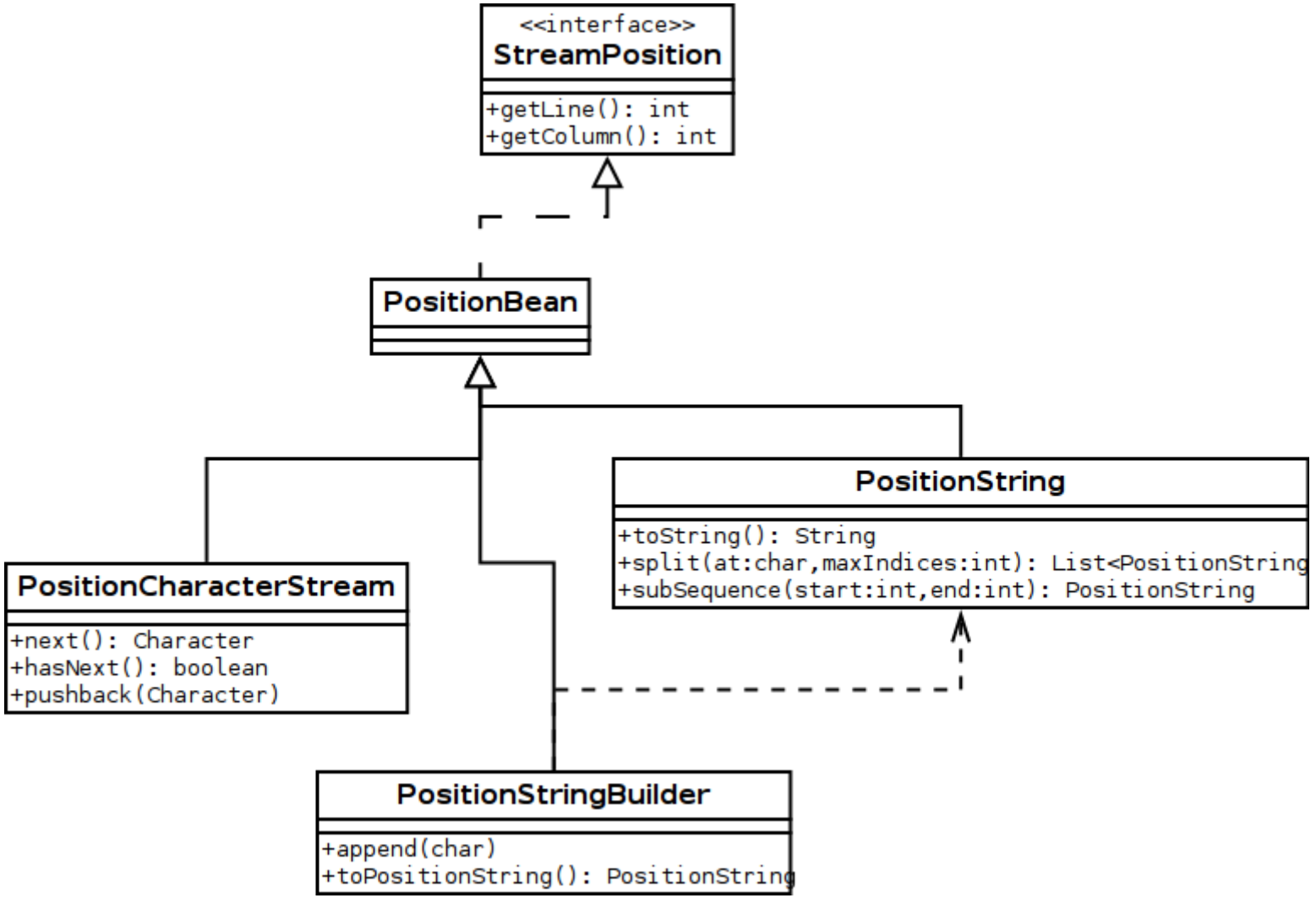
Grundlegende Datentypen

- Grundlagentypen ziehen sich durch das gesamte Projekt hindurch

- Die „Stream-Position“
  - Konstrukte (z.B. Schleifen und Bezeichner) kennen jeweils Ort, an dem sie in der Eingabe standen
  - Realisiert durch:
    - *StreamPosition:* allgemeines Interface
    - *PositionBean:* Implementierung der *StreamPosition*
    - *PositionString:* String der seine Position „kennt“

    - So wie weitere Helferklassen
    - Und alle Klassen, die das Interface benutzen
      - Hervorstehend: Die Fehlerausgabe

- Klassenstruktur



Grundlegende Datentypen

- Entstandene Probleme
  - StreamPosition noch nicht allgemein genug
- Stand der Umsetzung
  - Noch nicht von allen Klassen benutzt

Agenda

- Übernommene Aufgaben und Programmierstil
- Grundlegende Datentypen
- Verarbeiten der XML-Eingaben
- Verarbeiten der Ausdrücke
- Die Symboltabelle
- Fazit

Lexen der XML-Eingaben

- Der Begriff *lexen*
  - „Lexer“, kurz für „lexikalischer Scanner“
  - Das Zerteilen eines Datenstromes in syntaktische Teilstücke (Tokens)
  - Aufbereitung der Daten für die semantische Analyse

Lexen der XML-Eingaben

- Der Begriff *lexen*

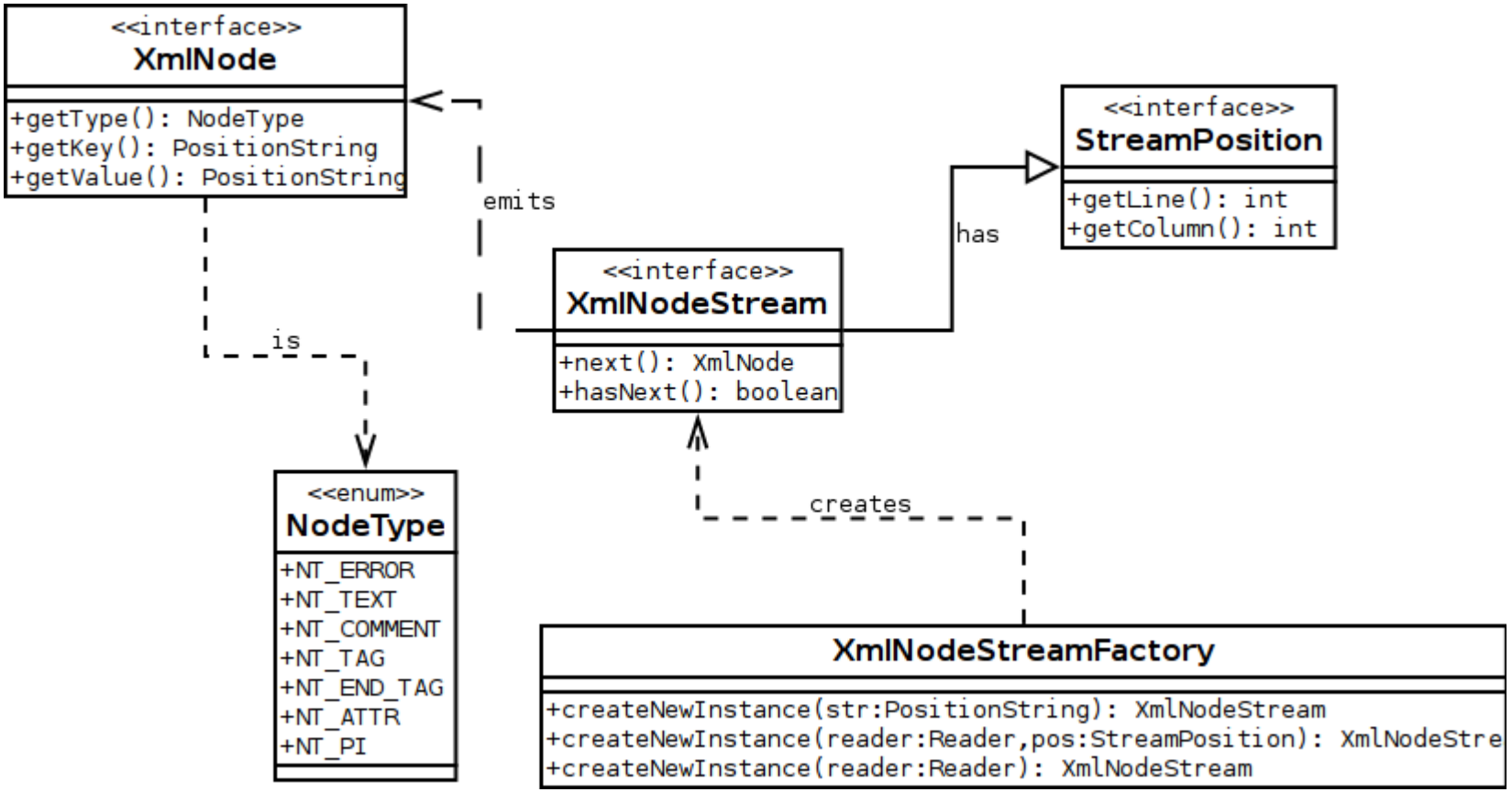
- Die Struktur der Programmiersprache hat einen XML-artigen Aufbau
  - mit geringfügigen Vereinfachungen für den Anwender
    - U.a. „<“ und „>“ in Attributen erlaubt
  - Eingabe als Rohtext im Format
    - ```
      <module name="Wert">
          abc
      </module>
      ```
  - Verstanden als
    - Öffnendes Tag „`module`“
    - Attribute „`name`“ = „`Wert`“
    - Text „`abc`“
    - Schließendes Tag „`module`“
      ```

Lexen der XML-Eingaben

- Der Begriff *lexen*
- Die Struktur der Programmiersprache hat einen XML-artigen Aufbau
- Einlesen erfolgt in Anlehnung an *StAX*
  - *Streaming API for XML*
    - Quasistandard zum Parsen von XML-Eingaben in Java
  - Schrittweises Erkennen einzelner Lexeme
  - Vereinfacht Aufbau eines DOM-Baumes

Lexen der XML-Eingaben

- Klassenstruktur



Lexen der XML-Eingaben

- Entstandene Probleme
  - „Gewissensfragen“: Nähe zu XML-Standard oder mehr Vereinfachungen?

Lexen der XML-Eingaben

- Entstandene Probleme

- Zusammenfassung der Umsetzung
  - Kann allgemeine XML-Daten lesen
  - Implementierung des Iterator-Interfaces
  - Nichtüberprüfen auf Wohlgeformtheit
    - „<a>…“ ist kein Fehler
    - Somit auch als Lexer für durchschnittliche HTML-Seiten verwendbar

Agenda

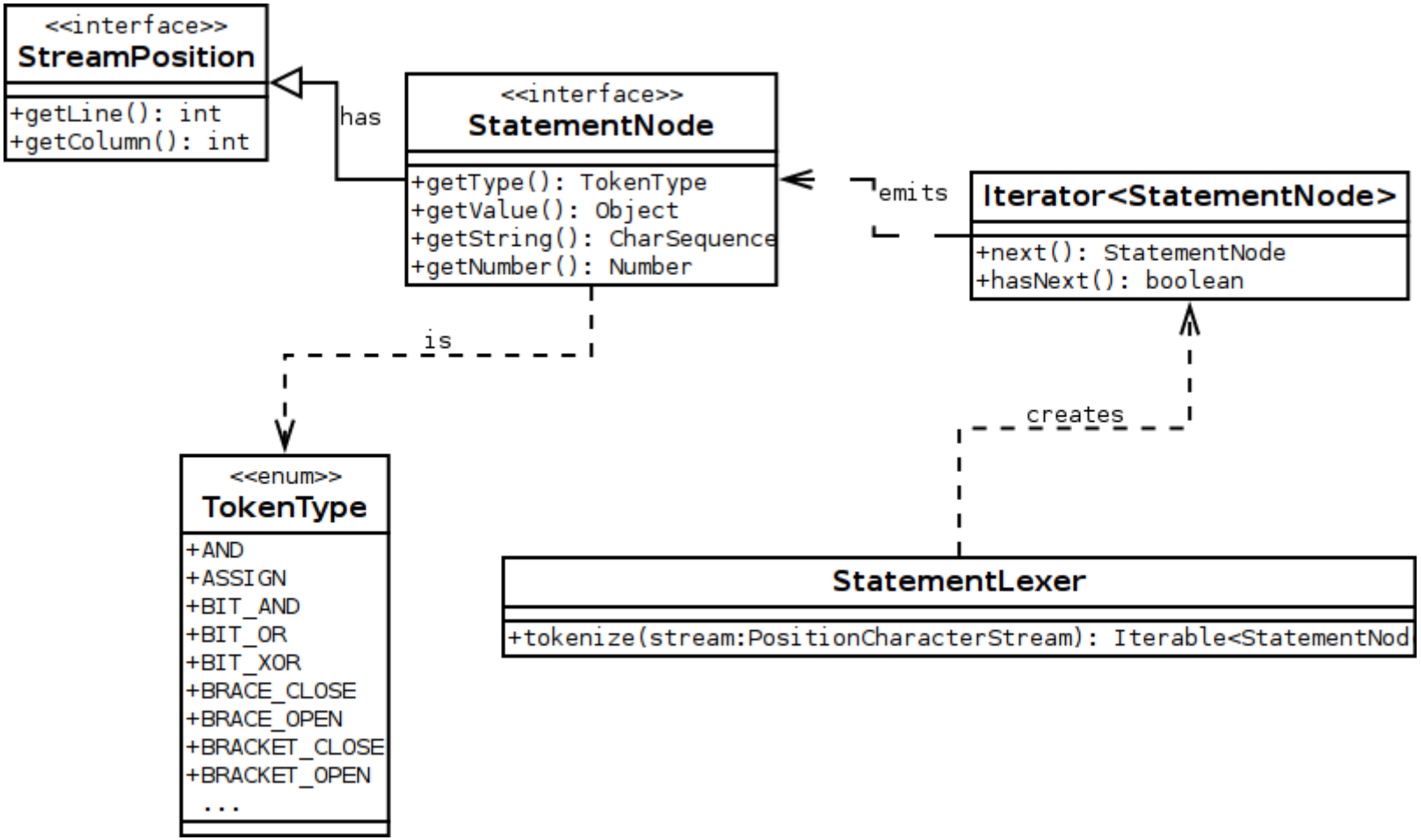
- Übernommene Aufgaben und Programmierstil
- Grundlegende Datentypen
- Verarbeiten der XML-Eingaben
- Verarbeiten der Ausdrücke
- Die Symboltabelle
- Fazit

Lexen der Ausdrücke

- Syntaktisches Verstehen von Java-Ausdrücken:
  - `(1+2) * 3`
    - öffnende Klammer
    - Zahl „1“
    - Plus
    - Zahl „2“
    - …
- Ähnliche Aufgabe wie das Lexen der XML-Daten
  - Nur mit mehr Lexem-Typen

Lexen der Ausdrücke

- Klassenstruktur



Lexen der Ausdrücke

- Entstandene Probleme
  - Designfragen:
  - Abweichung von Java-Syntax, wenn sinnvoll
  - Schwergewichtige Objekte zur Vereinheitlichung

- Zusammenfassung der Umsetzung
  - Anpassungen an XML noch nicht vollständig
    - Zeichenreferenzen wie `>` werden nicht verstanden

Agenda

- Übernommene Aufgaben und Programmierstil
- Grundlegende Datentypen
- Verarbeiten der XML-Eingaben
- Verarbeiten der Ausdrücke
- Die Symboltabelle
- Fazit

Die Symboltabelle

- Schnittstelle: Plattform- und Compiler-spezifische Datentypen
  - Für Plattform Java werden die Klassen Javas geladen
  - Interface ist Plattform-unabhängig
  - Java-ähnlicher Aufbau der Plattform jedoch zwingend

Die Symboltabelle

- Schnittstelle: Plattform- und Compiler-spezifische Datentypen

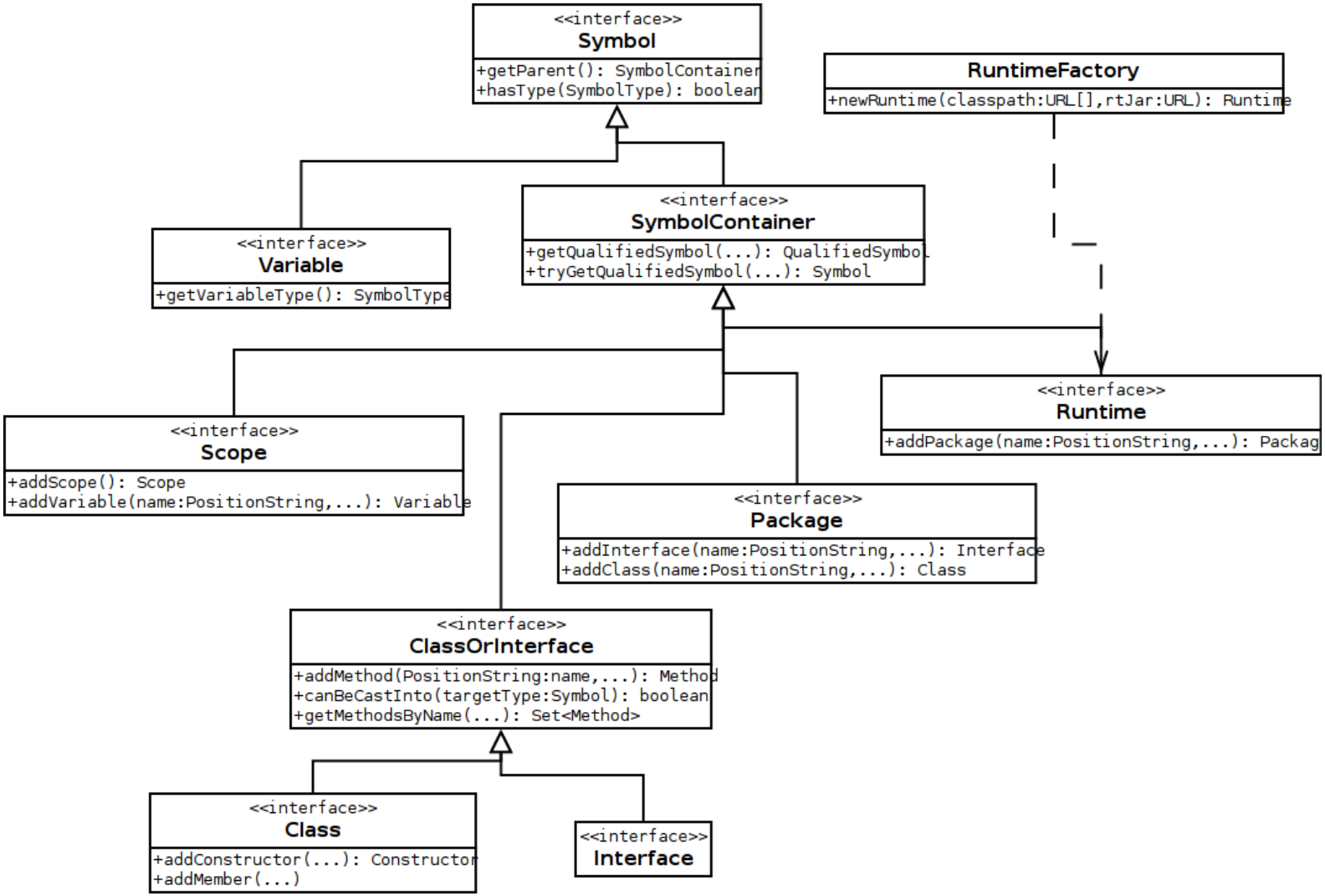
- Namensauflösung
  - Tabelle wird durch den Annotator mit Benutzerdaten befüllt
  - Verschachtelung und Sichtbarkeit durch Composite-Pattern
  - Speicherung der Referenzierungen von Symbolen
  - Namensauflösung in zwei Schritten

Die Symboltabelle

- Schnittstelle: Plattform- und Compiler-spezifische Datentypen
- Namensauflösung
- Gültigkeit von Bezeichnern
  - Abweisen von Schlüsselwörtern als Bezeichner
  - Überprüfung von Bezeichnern auf Gültigkeit
  - Dekoration ungültiger Bezeichner

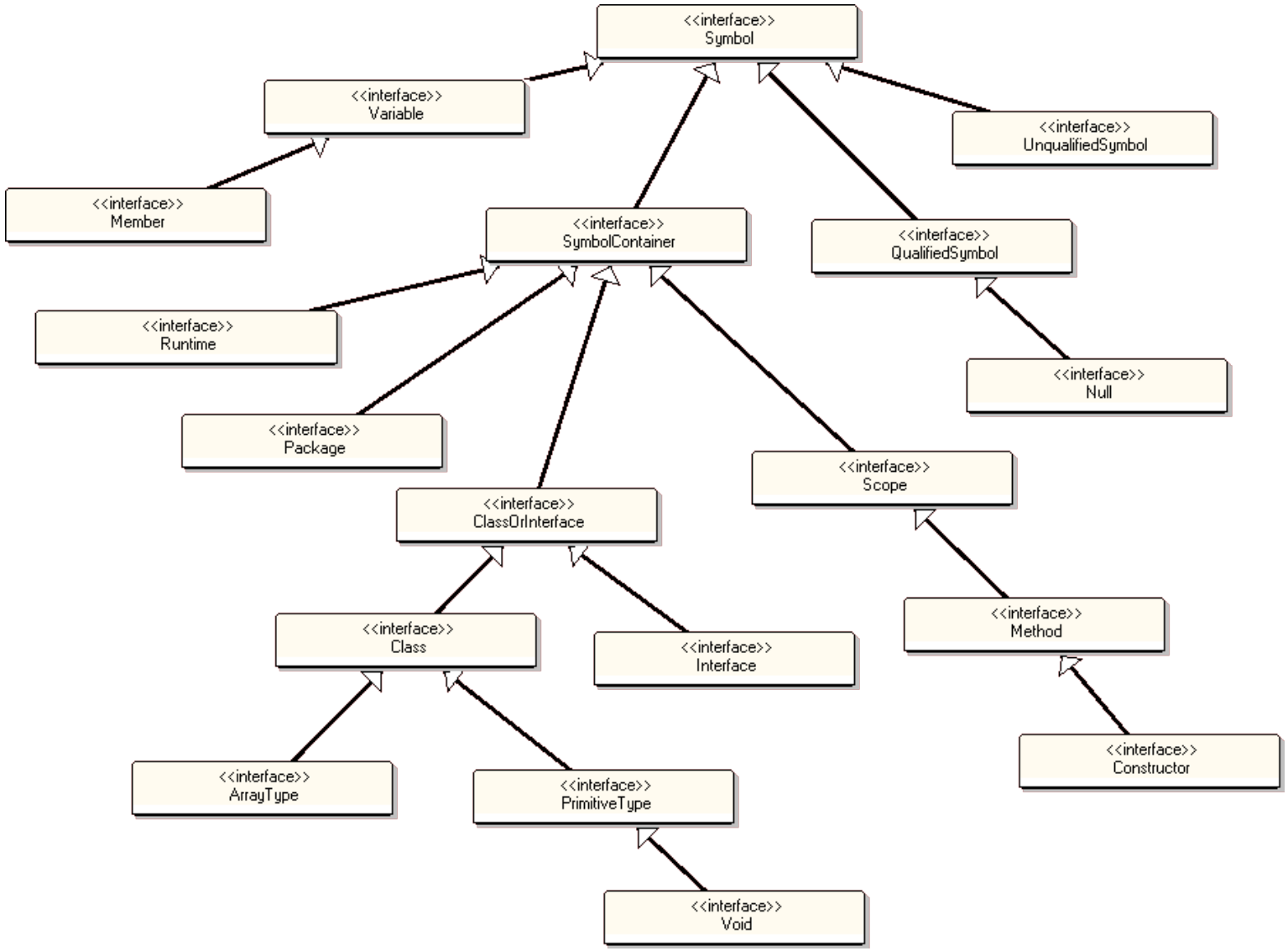
Die Symboltabelle

- Klassenstruktur (stark vereinfacht)



Die Symboltabelle

- Klassenstruktur (Vererbungshierarchie)



Die Symboltabelle

- Entstandene Probleme
  - Größe wird Problem
    - Optimieren des Ladens notwendig
    - Optimieren der Datenspeicherung notwendig
  - Zeitdruck
    - Aufgabe nicht in vier Wochen zu bewerkstelligen

Die Symboltabelle

- Entstandene Probleme

- Zusammenfassung der Umsetzung
  - Gute Grundlage zur Weiterarbeit
  - Weitere Optimierungsmöglichkeiten möglich
    - Abkehr von Javas Reflexions-API
    - Benutzung von *HashMaps* anstatt *TreeMaps*

Agenda

- Übernommene Aufgaben und Programmierstil
- Grundlegende Datentypen
- Verarbeiten der XML-Eingaben
- Verarbeiten der Ausdrücke
- Die Symboltabelle
- Fazit

Fazit

- Umsetzung größtenteils abgeschlossen
  - Unit-Tests unzureichend durchgeführt
  - Integration in andere Teilprojekte nicht abgeschlossen
  - Vereinfachungen in der Anwendung noch möglich

Fazit

- Umsetzung größtenteils abgeschlossen

- Zusammenarbeit der Teilprojekte
  - Designfragen schnell und basisdemokratisch gelöst
  - Einheitliche Programmierumgebung
  - Viele persönliche Gespräche

Agenda

- Übernommene Aufgaben und Programmierstil
- Grundlegende Datentypen
- Verarbeiten der XML-Eingaben
- Verarbeiten der Ausdrücke
- Die Symboltabelle
- Fazit

Vielen Dank!