Kurze Beschreibungen

Zur Lösung der Aufgaben von CA3 wurden folgende Klassen erstellt bzw. angepasst. Wie genau, wird nachfolgend kurz erklärt:

- Symbol,
- Symboltable,
- YAPLException,
- Scope,
- SymbolImpl,
- SymbolTableImpl,
- SymbolKind (enum),
- Type

Änderungen

- Die private static final integers im Symbol interface, welche die möglichen Typen des Symbols repräsentieren, wurden entfernt. Stattdessen wurde das enum SymbolKind eingeführt. Bei den Methoden setKind und getKind, wurde entsprechend der Typ des Rückgabewerts bzw. des Parameters von „int“ auf „SymbolKind“ geändert. Dementsprechende Änderungen wurden auch in abhängigen Klassen (Symboltable) vorgenommen.
- Symboltable bekommt eine weitere Methode, nämlich „containsIdentifierInCurrentScope“. Während „lookup“ auf den nächsthöheren Scope zurückgreift, wenn ein Symbol im aktuellen Scope nicht auffindbar ist, sucht „containsIdentifierInCurrentScope“ tatsächlich nur im aktuellen Scope. Dies ist wichtig für die Behandlung von Errorcode „SymbolExists“.
- Die Klasse Scope hat als Hauptkomponente eine Map von String auf Symbol, also die Symboltabelle für diese Scopeebene. Weitere Member sind „isGlobal“, „parentSymbol“ und ein Verweis auf den übergeordneten „Parentscope“.
- SymbolTableImpl ist die konkrete Klasse zum Interface Symboltable. Sie beinhaltet einen Stack von Scopes. Im Konstruktor wird der erste Scope – jener der Predefined Procedures – geöffnet, und die in der YAPL-Syntax definierten Procedures hinzugefügt, um Aufgabe 3.3 zu erfüllen.
- SymbolImpl ist die konkrete Klasse zum Interface Symbol.
- YAPLException implementiert das CompilerError Interface. Der Konstruktor benötigt die Fehlernummer, das fehlerverursachende Token und – wenn vorhanden (sonst null) – das zugehörige Symbol. Je nach Fehlercode wird über getMessage() die jeweilige Errormessage generiert.

Die einzige weitere Änderung ist die Anpassung von src/yapl/compiler/CA2_3.jj
Am build.xml hat sich im Vergleich zur Abgabe von CA2 nur geändert, dass version nun auf „symbolcheck“ gesetzt wurde.

Das target zur Ausführung des Parsers mit allen Testfiles einer Compilerversion ist „eval-all“, welches aber eh als default target definiert ist. Deshalb reicht „ant“.

Sonstiges:
Ausführung mittels **ant**.
Der default-Wert in built.xml wurde auf eval-all gesetzt.
Es müssen eigentlich keine ant-properties überschrieben werden, da alle Dependencies im resources-folder mitgeliefert werden. Ausführen von „ant“ reicht.