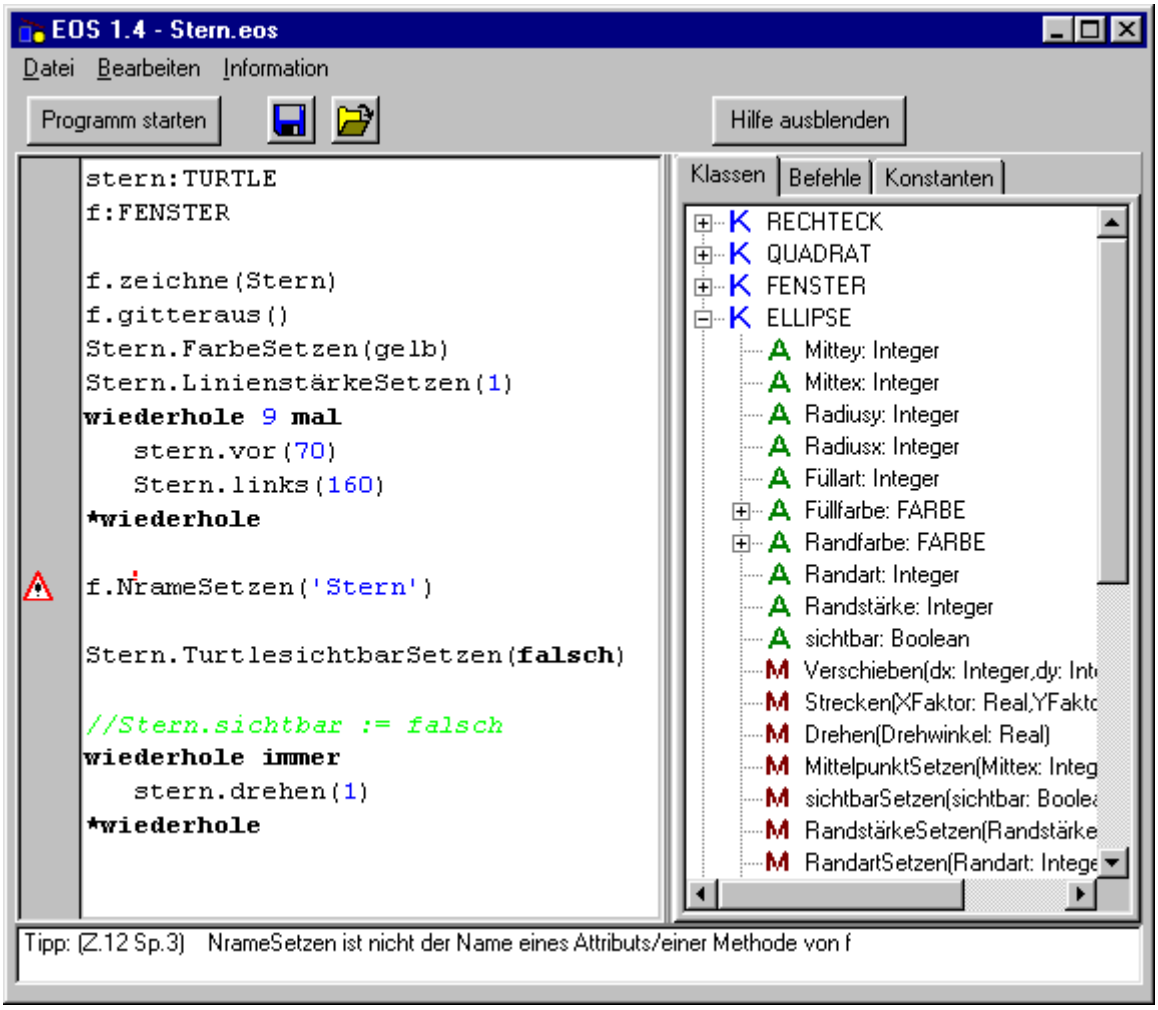
# EOS – Einfache Objektorientierte Sprache

EOS ist eine IDE mit Interpreter für eine bewusst einfach gehaltene objektorientierte Sprache, die es Schülern ab der Jahrgangsstufe 6 ermöglichen soll, erste Einblicke in die objektorientierte Programmierung zu gewinnen.

## 1. Beschreibung der IDE




Im Bild oben ist der **Editor** zu sehen. Er besitzt die übliche Funktionalität. Die Funktionen **Kopieren, Ausschneiden, Einfügen** sind über die Tastenkombinationen <Strg>+Einfg, <Shift>+Entf, <Shift>+Einfg zu erreichen. Dateien können **geöffnet** (gelbes Ordnersymbol bzw. Datei->Öffnen) und **gespeichert** (blaue Diskette bzw. Datei->Speichern) werden.

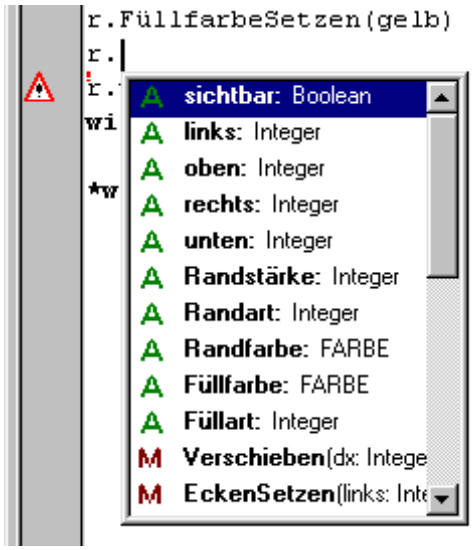
Während der Programmeingabe wird die **Syntax laufend überprüft**. Syntaxfehler werden am unteren Rand des Fensters angezeigt. Ihre genaue Position im Programmtext wird durch ein **Warndreieck** und ein kleines rotes Ausrufezeichen kenntlich gemacht. Die Schüler werden so laufend darüber informiert, ob ihr Programm korrekt ist und müssen nicht mühselige „Programm starten – Fehler suchen – Programm verbessern“ – Zyklen durchlaufen.

Mit „ // “ kann eine **Programmzeile auskommentiert** werden. Kommentare über mehrere Programmzeilen hinweg sind mit **geschweiften Klammern** möglich.

Um die Lesbarkeit der Programme zu erhöhen, verfügt der Editor über **Syntax-Highlighting**. Außerdem wird der Programmtext bei Wiederholungen und bedingten Anweisungen **automatisch eingerückt**. Das Programm hat somit zu jedem Zeitpunkt eine ansprechende Form, die das Durchschauen der Programmstruktur für die Schüler vereinfacht.
Über den Button **„Hilfe zeigen“** (im Bild hat er seine Aufschrift bereits zu „Hilfe ausblenden“ geändert) kann jederzeit (auch während des Programmlaufs – siehe später) eine Hilfe eingeblendet werden, die **Auskunft über die Klassenstruktur, die Befehle von EOS und die verfügbaren Konstanten** (z.B. vordefinierte Farben, Füllarten, Linienarten) gibt. Diese Hilfe ist zwar knapp gehalten, dafür aber so übersichtlich, so dass sich die Schüler schnell die Informationen finden, die sie brauchen. Überdies gibt es beim Aufruf der Hilfefunktion kein Fensterwirrwarr. Die Hilfe ist immer dort, wo sie gebraucht wird und verdeckt nie den Programmtext.

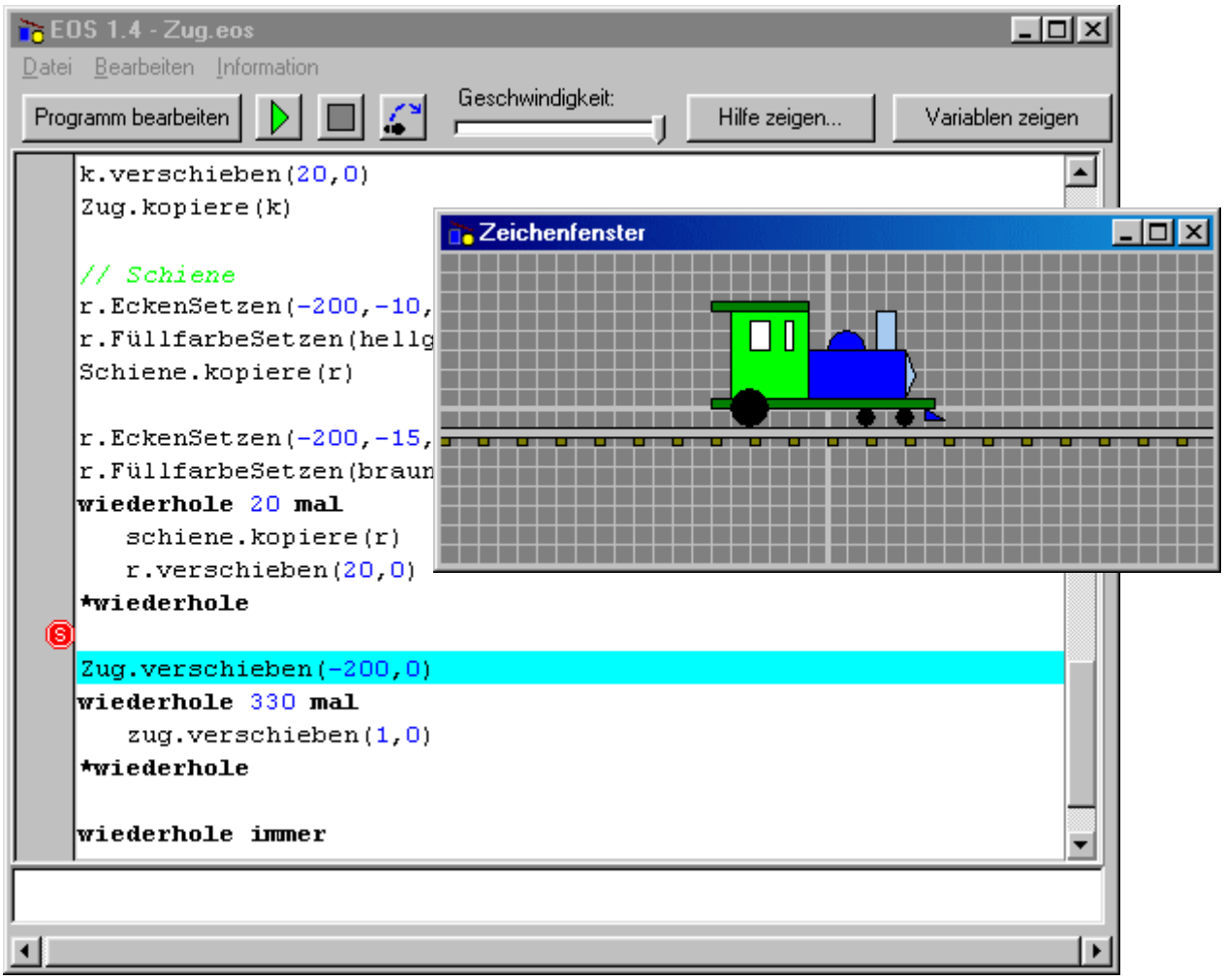
Sowohl während der Programmeingabe als auch während des Programmlaufs können **Haltepunkte** gesetzt werden, indem in das graue Feld links neben der entsprechenden Programmzeile geklickt wird. Die Haltepunkte werden durch ein kleines Stopschild (rot, mit eine S drin) angedeutet.

Der Interpreter verfügt über **Code-Vervollständigungsfunktionen**. Sobald man einen Objektbezeichner gefolgt von einem Punkt eingetippt hat, öffnet sich ein Fenster, in dem alle möglichen Attribute und Methoden des betreffenden Objekts zur Auswahl angeboten werden. Auch bei der Variablendeklaration hilft die Code-Vervollständigung.




Über den Button **„Programm starten“** gelangt man zum Interpreter (und wieder zurück).

## 2. Der Interpreter




Im Interpreter kann das Programm „in einem Rutsch“ (Button mit dem grünen Dreieck) oder **schrittweise** ausgeführt werden (Button mit dem angedeuteten Fußstapfen). Die **aktuelle Position des Programmzeigers** wird im Editorfenster hellblau hinterlegt.
Sobald ein **Objekt vom Typ Fenster** deklariert wurde, wird dieses Fenster auf dem Bildschirm angezeigt. Repräsentanten aller Objekte, die mit der Methode add dem Fenster hinzugefügt wurden, sind darin zu sehen. Wird ein **Attribut** eines dieser Objekte **geändert**, so **ändert sich automatisch das Aussehen** des Repräsentanten im Fenster. Selbstverständlich können auch mehrere Fenster geöffnet werden. Die Größe der Ausgabefenster kann mit der Maus oder durch entsprechende Befehle im Programm (z.B. f.unten := 500) geändert werden.

Mit dem **Reset-Button** (graues Quadrat) wird das Programm angehalten und in seinen Ausgangszustand versetzt.

Mit dem Pause-Button (  , erscheint nach Drücken auf  ) kann das Programm während des Programmlaufs angehalten werden, ohne einen Programmreset durchzuführen.

Der Interpreter ist interrupt-gesteuert, d.h. im ununterbrochenen Programmablauf (  ) werden alle n Millisekunden (n ist einstellbar) so viele Befehle ausgeführt, bis die Ausgabefenster neu gezeichnet werden müssen (höchstens aber 10). Dann wird die Kontrolle wieder an den Hauptthread des Programms übergeben. Dadurch wird das System auch bei einer Endlosschleife im EOS-Programm niemals ganz blockiert. Insbesondere sollte ein Systemabsturz durch einen Programmfehler nicht möglich sein.
(Falls doch mal einer auftreten sollte: Bitte mailen sie mir das entsprechende EOS-Programm!)

## 3. Die Syntax der Sprache EOS

**Kurzübersicht:**

- EOS ist **case-insensitive**, Klein- und Großschreibung ist also ohne Bedeutung.
- **Variablen** werden durch `Name1,Name2,...:Typ` **deklariert**, also z.B.
  `r1,r2 : Rechteck`
- Der **Punkt** hat die übliche Bedeutung als Operator zur Beschreibung von Methoden/Attributen eines Objekts, z.B.
  `r1.Farbe`
- **Anweisungen** werden durch Leerzeichen oder Zeilenumbruch voneinander **getrennt**. Der Einstieg in die Programmierung soll den Schülern nicht durch Strichpunkte verleidet werden.
- **Zuweisungen** geschehen mit `:=` , also z.B.
  `i := 10`
- **Zuweisungen zu Objektattributen**, also z.B.
  `r1.links := -20`
  sind **verboten**. Dieses Verbot kann in den Programmeinstellungen **aufgehoben** werden.
- Zur **Kennzeichnung von Programmblöcken** haben die Strukturbefehle `wiederhole, wenn, solange, für` entsprechende Befehle `*wiederhole, *wenn, *solange, *für` zur Kennzeichnung des Blockendes (wie in Robot-Karol). `begin,end` (bzw. geschweifte Klammern) zur Blockkennzeichnung sind also nicht nötig.
- Einen Befehl `Ende` oder dergleichen zur Kennzeichnung des Programmendes gibt es nicht. Textende = Programmende.
- **Eigene Methoden** (auch mit Parametern) können durch
  ```
  Methode <Name> (Parameterliste)
     <Sequenz>
  Ende
  ```
  an jeder Stelle des Programms (außerhalb von Wiederholungen oder bedingten Anweisungen) deklariert und mit `<Name>( )` aufgerufen werden. Die **Deklaration der Methode kann auch nach dem ersten Aufruf** erfolgen!
- Die **benutzbaren Klassen** sind
  `Fenster, Rechteck, Quadrat, Ellipse, Kreis, Strecke, Textfeld, Turtle` (LOGO-Schildkröte), `Gruppe, Farbe`
  Die Basisklasse aller zeichenbaren Figuren ist `Figur`. Direkt davon abgeleitet ist `GefüllteFigur` , `Strecke und Gruppe`. Von `GefüllteFigur` abgeleitet sind `Rechteck, Quadrat, Ellips, Kreis,`

`Textfeld`.
In abgeleiteten Klassen sind nicht immer alle Attribute/Methoden der Elternklasse sichtbar, so hat die Klasse `Kreis` z.B. kein sichtbares Attribut `links`.

**Übersicht in EBNF:**

**Konstante**
<const> ::= <boolescheconst>|<integerconst>|<stringconst>|<realconst>
<boolescheconst> ::= **wahr** | **falsch**
<digit> ::= 0 | 1 | … | 9
<letter> ::= a | b | … | z | ä | ö | ü | ß | _
<character> ::= <letter> | <digit>
<zeichen> ::= <character> | <sonderzeichen>
<sonderzeichen> ::= ! | “ | …
<integerconst> ::= (1 | 2| … | 9) {<digit>}
<stringconst> ::= ’{<zeichen>}’
<realconst> ::= <mant> [e <expo>]
<mant> ::= <digit>.<digit>[<digit>]
<expo> ::= (+ | - ) <digit>[<digit>]

**Bezeichner**
<id> ::= <letter>{<character>}

**Variablendeklaration**
<variablendeklaration> ::= <id>{,<id>} : <variablentyp>
<var id> ::= [<selector>.]<id>
<selector> ::= <var id>

**Methodendeklaration**
<methodendeklaration> ::= Methode <var id> <Sequenz> Ende

**Funktions/Methodenaufruf**
<call> ::= <var id> ( <exp> { , <exp> } )

**Ausdrücke**

<exp> ::= <const> | <var id> | (<exp>) | <monad op><exp>|
<exp><dyad op><exp>|<call>
<monad op> ::= - | not
<dyad op> ::= + | - | * | / | < | <= | > | >= | <> | = | und | oder

**Anweisungen**
<anweisung> ::= <variablendeklaration> | <zuweisung> | <sequenz> |
< bedingung > | <wiederholung> | <call> | <with-Block>
|<Methodendeklaration>
<zuweisung> ::= <var id> := <exp>
<sequenz> ::= <anweisung> {’ ’ <anweisung>} (durch Leerzeichen getrennt!)
<bedingung> ::= wenn <exp> dann <sequenz> [ sonst <sequenz> ] *wenn
<wiederholung> ::= <wiederhole immer> | <wiederhole n mal> | <wiederhole bis> |

<wiederhole solange1> | <wiederhole solange2> | <solange tue>
<wiederhole immer> ::= wiederhole immer <sequenz> *wiederhole
<wiederhole n mal> ::= wiederhole ( <integerconst> | ' (' <exp> ' )' ) mal <sequenz>
*wiederhole
<wiederhole bis> ::= wiederhole <sequenz> bis <exp> *wiederhole
<wiederhole solange1> ::= wiederhole solange <exp> <sequenz> *wiederhole
<wiederhole solange2> ::= wiederhole <sequenz> solange <exp> *wiederhole
<solange tue> ::= solange <exp> tue <sequenz> *solange
<with-Block> ::= für <exp> <sequenz> *für

**Programm**
<programm> ::= <sequenz>

**Nähere Angaben:**

Variablendeklarationen können überall im Programm auftauchen, wo die Verschachtelungstiefe 0 beträgt, d.h. nicht zwischen wiederhole ... *wiederhole, wenn ... *wenn, usw.

Die Anweisungen werden mit Leerzeichen (oder cr) getrennt. Strichpunkte sind nicht nötig => die Schüler sind deutlich schneller im Tippen und müssen sich nicht mit den Strichpunkten herumschlagen.

Wird dem Attribut eines Objekts ein Wert zugewiesen, so wird **automatisch** die jeweilige set-Methode des Objekts aufgerufen (wie in Delphi, c#, VBA), die evtl. ein Neuzeichnen des Objekts veranlasst.

Ein „Ende-Befehl" zum Programmende ist nicht nötig.

**Systemfunktionen:**
sin, cos, tan, round, trunc (alles wie in Pascal)

## 4. Die Klassen von EOS

... und die Sprache selbst lernt man am besten durch die mitgelieferten Beispieldateien kennen!