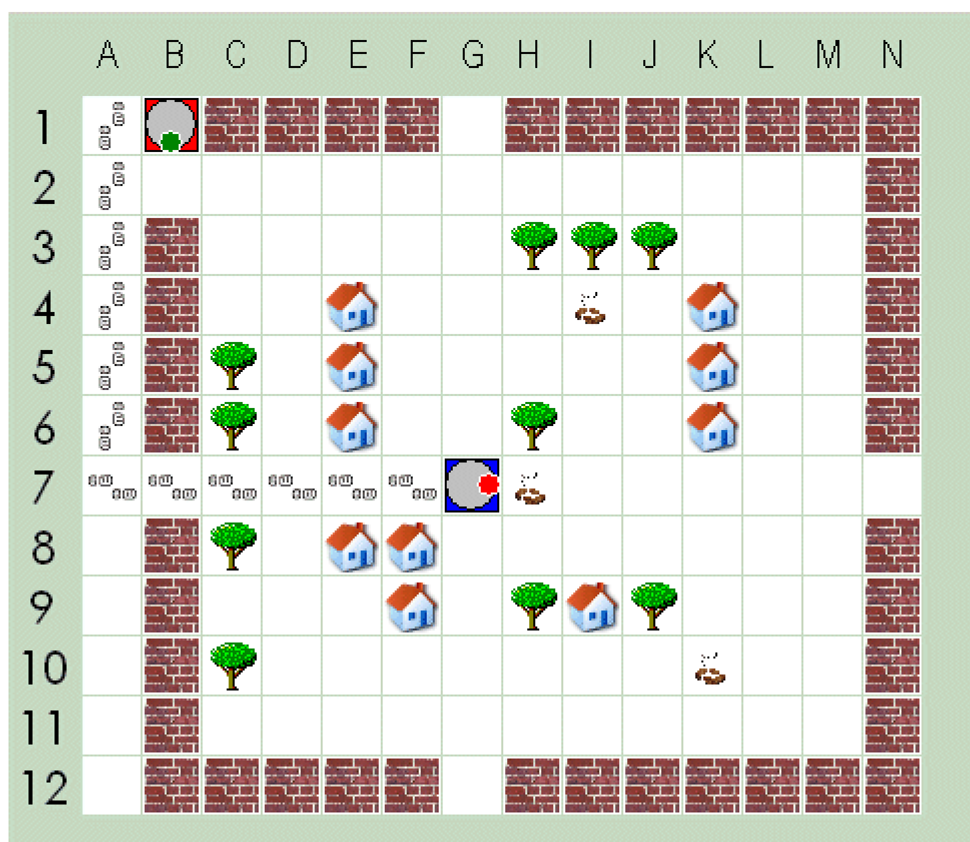
Siegfried Spolwig

# Karel D. Robot

- Der Delphi-Karel 2.4 -




*Ein spielerischer Weg, um OOP mit Delphi zu lernen*

© 2004-2005 Siegfried Spolwig. Alle Rechte sind vorbehalten. Das Werk und seine Teile sind urheberechtlich geschützt. Jede Verwertung in anderen als den gesetzlich zugelassenen Fällen bedarf deshalb der vorherigen schriftlichen Einwilligung des Verfassers.

## Inhaltsverzeichnis


| | | |
|---|---|---|
| **Vorwort** | | **3** |
| | | |
| **1. Allgemeine Beschreibung** | | **4** |
| 1.1 | Die Welten | 5 |
| 1.2 | Die Critters | 7 |
| 1.3 | Die Roboter | 8 |
| 1.4 | Die Sachen | 10 |
| | | |
| **2. Didaktische Konzeption** | | **11** |
| 2.1 | Unterrichtseinsatz | 12 |
| 2.2 | Links | 14 |
| | | |
| **3. Dokumentation** | | **15** |
| 3.1 | Klassendiagramme | 15 |
| 3.2 | Klassen | 16 |
| | - TBaum | 16 |
| | - TBlackhole | 17 |
| | - TCleanCity | 18 |
| | - TCritter | 19 |
| | - TGras | 21 |
| | - THaus | 22 |
| | - TItem | 23 |
| | - TKarel | 25 |
| | - TLabyrinth | 27 |
| | - TLagerhaus | 28 |
| | - TMatter | 30 |
| | - TMemory | 30 |
| | - TMuell | 31 |
| | - TMyWorld | 32 |
| | - TRobot | 33 |
| | - TShadow | 36 |
| | - TSpur | 37 |
| | - TStack | 38 |
| | - TStein | 40 |
| | - TTrainingscamp | 41 |
| | - TWelt | 42 |
| | - TControlFrm | 45 |
| | - TFensterFrm | 46 |
| | - TProtokollFrm | 48 |
| | | |
| **4. Architektur und Technisches** | | **50** |
| 4.1 | Installation | 52 |
| 4.2 | Programmiertips | 55 |
| 4.3 | F.A.Q. | 60 |


letzte Änderung: 28-APR-05

# Vorwort

Man möchte meinen, daß, wenn einer mehr als zehn Jahre lang Schülern, Studenten und Lehrern objektorientierte Programmierung beigebracht hat, er nun endlich wissen müßte, wie das geht. Im Prinzip schon. Aber bereits die Fülle der verschiedenen Ansätze und Versuche zeigt, daß es offensichtlich keinen Königsweg gibt, nur bessere oder weniger erfolgreiche Wege.

Auch in unserer Schule, die auf diesem Gebiet ziemlich rührig und produktiv ist, wurden immer wieder neue Ansätze probiert, verworfen, verbessert, meist jedoch in didaktischen Detailfragen, während in den übergeordneten Zielen und Methoden sich ein ordentlicher Stil etabliert hat.

Einigkeit herrscht in der Überzeugung, daß die objektorientierte Denkweise für Schüler kein Problem ist, jedoch die handwerkliche Umsetzung ('das Programmieren') wegen der fehlenden ausreichenden Übungszeit, ohne die es nicht geht, eine große Hürde ist. Dieses wird noch dadurch verschärft, daß sich die Schule für DELPHI als Entwicklungsumgebung entschieden hat. Mich verbindet mit Delphi geradezu eine Hassliebe, wobei ich dennoch glaube, daß am Ende die Vorteile für den Unterricht überwiegen. Das ist an anderer Stelle hinreichend diskutiert worden.

Objektorientierte Programmierung in der Schule ist also weniger ein intellektuelles als ein methodisches Problem.

Ein Programmierstil, dessen Nukleus das Objekt ist, fordert geradezu heraus, die Objekte, die in einem Programm bearbeitet werden, auch auf dem Bildschirm sichtbar zu machen, um eine direkte visuelle Kontrolle der Effekte zu haben, die der Schüler in seinen Anweisungen programmiert. Zu diesem Zweck haben wir die Unterrichtsequenz mit den Grafikobjekten (uGrafik) entwickelt und damit gute Erfolge verzeichnet. Da die Aufgabenstellungen häufig einen mehr oder weniger 'dekorativen' Charakter haben, kommt das Element der Algorithmisierung etwas zu kurz.

Dieses zeigt sich auch häufig bei Programmen, die Aufgabenstellungen aus dem wirtschaftlich-verwaltenden Bereich haben. Registrierende Systeme kommen oft genug mit Set- und Get-Methoden aus.

Diese Ausgangslage und reichlich Freizeit haben mich bewogen, eine Karel-Lernumgebung zu entwickeln, die mehr ermöglicht als die klassischen Versionen, die mit Mini-Languages im Wesentlichen in Algorithmen und imperative Programmierung einführen, auch wenn Sie in OOP-Sprache und Dot.Notation daher kommen. Einige dieser Lernumgebungen waren mir nur oberflächlich bekannt (weil ich damit nichts im Sinn hatte) und ich habe bewußt auch auf ein genaues Studium dieser Vorlagen verzichtet, um nicht den Blick zu verstellen für das, was IMHO für eine zeitgemäße Lernumgebung wichtig ist. Gleichzeitig ging es mir noch einmal darum, an einem Beispiel zu zeigen, wie mit den Mitteln der objektorientierten Programmierung (Vererbung, Polymorphie) sparsamer Code zustande kommt und Kapselung die Sicherheit erhöht und die Fehlersuche erleichtert. Die Programmierung war insofern eher eine leichte Anstrengung. Das Ergebnis wird hier für die Besichtigung und hoffentlich auch zur Benutzung vorgestellt.

Meinen Kollegen Hartmut Härtl, Johann Penon und Christian Steinbrucker danke ich für Anregungen und Kritik. Richard Pattis (Carnegie Mellon University) gebührt der Dank für die freundliche Genehmigung, den Namen 'Karel' verwenden zu dürfen und Werner Hartmann (ETH Zürich) für richtungweisenden Rat.

# 1. Allgemeine Beschreibung

Karel D. Robot ist ...

Was in Berlin nicht klappt - in Clean City werden Hundehäufchen und Müll per Roboter entsorgt!

- ein entfernter neuer Verwandter in der Karel-Familie, aber keine 1:1-Portierung nach Delphi, sondern ein offener Ansatz
- ist ein spielerischer Weg für den objektorientierten Einstieg in die Programmierung ('OOP von Anfang an').
- gedacht für den Anfangsunterricht à la Karel the Robot
- geeignet für den (Berliner) Einstieg über die Analyse komplexer Systeme
- entwickelt für den Anfangsunterricht auf der Ebene einer Mini-Language und den gleitenden Übergang in eine echte Programmiersprache
- mit einer integrierten Hilfe, Dokumentation und Tutorial ausgestattet
- objektorientiert (Architektur und Dot.Notation)
- erweiterbar / veränderbar - auch ohne Quellcodes
  Hier liegen die eigentlich interessanten Möglichkeiten: andere Wesen, Sachen, Welten und Methoden erfinden und entwickeln für den anspruchsvollen, fortgeschrittenen Unterricht

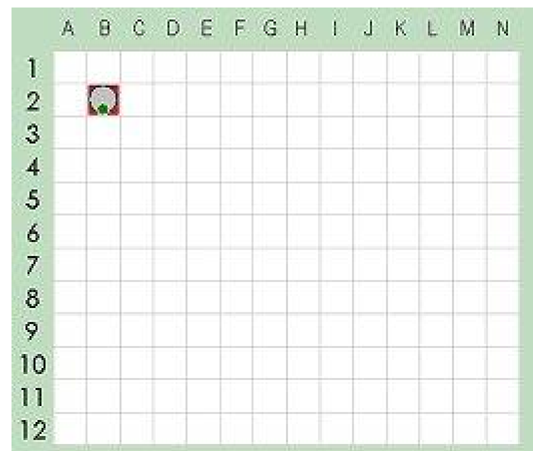
## 1.1 Die Welten

**Welt**

Karel D. Robots Welt ist klein und übersichtlich. Sie besteht aus 12 X 14 Feldern, die gleichzeitig die Position der Objekte bestimmen. Jedes Objekt kennt seine momentane Position: z. B. weiß der Roboter, daß er auf B,2 steht.

Die 'Welt' selbst hat den Überblick auf alle und weiß jederzeit, wer wo steht und was leer ist.

'Welt' ist verantwortlich für das gesamte Anzeigen, Löschen und Entfernen aller Objekte.

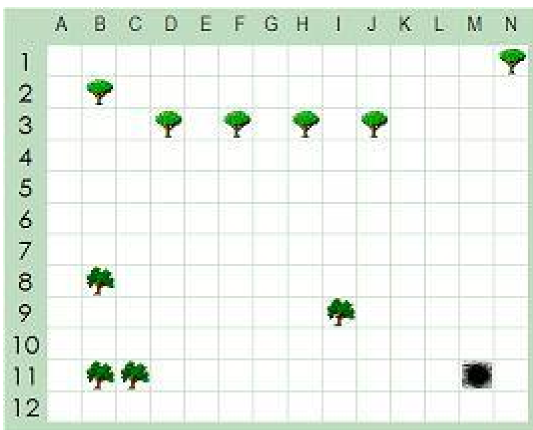



**Vorhandene Welten**

1. Trainingscamp

Der Platz für die ersten Bedienungs- und Programmierübungen. Es gibt ein paar Bäume, um die der Roboter herumkurven kann und ein schwarzes Loch, dem man besser fernbleibt.

Mögliche Aufgaben.:
- Die Handsteuerung erweitern.
- Die Welt bebauen und verschönern.
- Dem Roboter neue Methoden beibringen



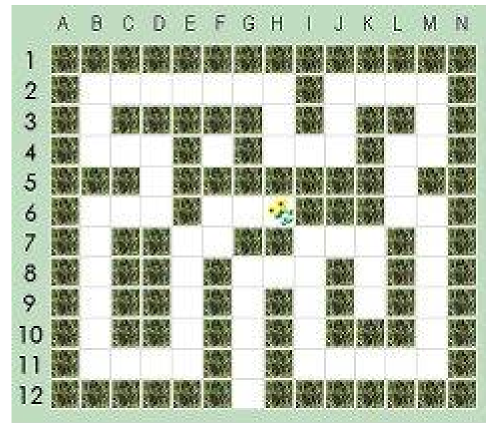

2. Clean City

Was in Berlin nicht klappt - in Clean City aber!
Ein schmuckes Dörfchen mit bösartigen Hundebesitzern, die nachts heimlich ihre Hunde irgendwo ihre Häufchen machen lassen.

Die Gemeinde hat einen Putzroboter RD1 angeschafft, der das wegmachen soll. Leider rennt der Roboter alles um, was ihm im Wege steht. Es müssen ihm also noch Verhaltensregeln beigebracht werden ...

3. Labyrinth

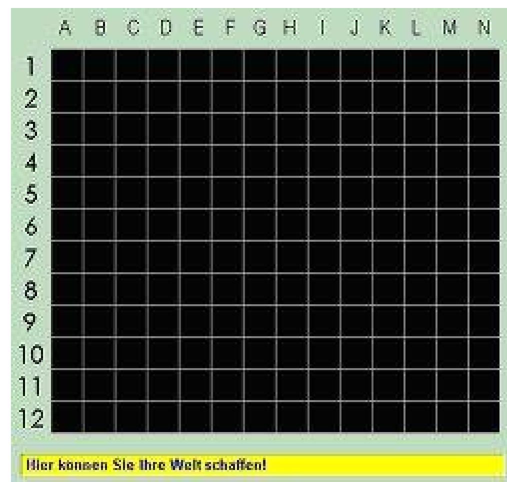
Am Ende ist die gelbe Blume. Karel soll sie finden und mitbringen.




4. My World

" Am Anfang war die Erde wüst und leer, und es war finster auf der Tiefe ..." (1. Buch Mose, 1.2.).

Hier können Sie Ihre eigene Welt erschaffen mit eigenen oder vorhandenen Figuren und eigenen Regeln.




**Programmierhinweis**:
Einfügen eines neuen Objekts in beliebige Welten:

- Objekt erzeugen z.B. Auto und
- an die gewünschte Stelle setzen mit **Auto.SetPos('G',2)**

## 1.2 Die Critters

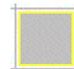
sind die beweglichen Ur-Wesen in der Welt. Sie haben die aktiven Methoden

- **Vor** d.h. gehe einen Schritt in der aktuellen Richtung vorwärts unabhängig davon, ob es frei ist oder besetzt. Wenn da was war, wird es entfernt, sie fressen quasi alles auf.

- **RechtsDrehen** d. h. auf der aktuellen Position um 90 Grad um die eigene Achse drehen.

- **OffLimits** **w**enn er sich außerhalb der Welt befindet, fällt er sozusagen vom Tisch und ist zerstört

Standardmäßig haben sie wie alle Items einen rechteckigen Grundriß, der in ein Feld paßt. Ihre Farbe ist grau mit einer gelben Umrandung.



Alle Sachen (und Items) können ein Bild tragen, das dann den Grundriß überdeckt. Damit läßt sich eine vielgestaltige Welt machen.

Die Bilder müssen vom bmp-Format sein und sollten eine Größe von 28 x 28 pix haben.
Zweckmäßigerweise werden alle neuen Bilder in das Verzeichnis 'Bilder' innerhalb das Projektordners kopiert.

**Programmierhinweis:**
Bild einfügen z. B. ein Monster

```
Critter.SetBild ('.\bilder\monster.bmp');
```

## 1.3 Die Roboter

Bevor Sie Ihren ersten Roboter bauen, studieren Sie bitte 'Asimov's Three Laws of Robotics':

1. A robot may not injure a human being, or, through inaction, allow a human being to come to harm.

2. A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.

3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

Der nachweislich erste Roboter war Robby the Robot, konstruiert von Doctor Edward Morbius nach einer Technik der Krell auf Altair 4. Die Begegnung und die Umstände wurden von einen Filmteam der MGM 1956 dokumentiert. Leider sind die Baupläne verschollen.

Man unterscheidet drei Arten von Robotern: Extern gesteuerte, halbautonome und autonome.

Ein Beispiel für eine Aufgabe, die der Roboter tun soll: Auf Feld C,4 liegt Müll, den der Roboter aufsammeln soll.

- Extern gesteuert
  bedeutet, jede Aktion kommt als Anweisung: *Vor, Vor, RechtsDrehen, ..., AufNehmen*

- Halbautonom
  bedeutet, der Roboter kann bestimmte komplexe Aktionen selbst ausführen kann.
  Die Anweisung lautet: *Gehe zu C,4, AufNehmen.*
  Er findet also selbstständig einen Weg nach C,4.

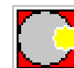
- Autonom
  bedeutet, daß der Roboter völlig selbständig eine allgemein beschriebene Aufgabe löst: *Beseitige allen Müll.*
  Er sucht alle Plätze ab, kann Müll von anderem unterscheiden und beseitigen

---

### Die Sprache

Unsere Roboter verstehen eine eigene Sprache. Das sind die Wörter für Aufträge oder Anfragen, die der Roboter ausführen kann. Sie sind im folgenden Abschnitt beschrieben. Daneben versteht er Schlüsselwörter aus Object Pascal (Delphi), die seinen eigenen Sprachumfang ergänzen. Das sind im wesentlichen solche zur allgemeinen Programmsteuerung, die Sie nach und nach bei Bedarf kennen lernen werden.

### Der Roboter RD1 vom Typ TRobot



ist der Urgroßvater des legendären R2D2 aus Star Wars.
Er hat ein rotes Fahrgestell und vorn eine Lampe, die die Richtung anzeigt und bei Bereitschaft grün, bei Fahrt gelb und bei Problemen rot aufleuchtet.
Er ist ziemlich dumm, aber ungeheuer stark, denn er walzt wie ein wild gewordener Bagger auf seinem Weg alles nieder, wenn er nicht richtig programmiert wird:
wo er war, ist nichts mehr; d. h. das Objekt, welches er überfahren hat, ist weg und aus der Welt entfernt.

Er wurde geplant als universelle Reinigungs- und Staubsaugermaschine, reißt leider aber auch Mauern und Häuser ab.

Er verfügt nur über die aktiven Methoden

- **Vor**
  d.h. gehe einen Schritt in der aktuellen Richtung vorwärts unabhängig davon, ob es frei ist oder besetzt. Wenn da was war, wird es entfernt.

- **RechtsDrehen**
  d. h. auf der aktuellen Position um 90 Grad um die eigene Achse drehen.

**- BatterieAufladen**
  wenn er eine bestimmten Anzahl von Schritten gegangen ist, ist die Batterie leer und kann neu geladen werden.

**- MacheSpur**
  er hinterläßt Fußabdrücke, damit man seinen Weg verfolgen kann.

**- KeineSpur**
  Fußabdrücke abschalten.

-----------------------------------------------------------------

Daneben hat er *Sensoren*, die seine Umgebung checken können.
*Diese sind bei RD1 aber noch nicht aktiviert!*

- **VorneFrei**
  d. h. er kann nachsehen, ob auf dem nächsten Feld etwas steht oder frei ist.

- **HindernisPruefen**
  d. h. wenn er an ein Hindernis kommt, geht die rote Warnlampe an und er kann feststellen, was für eine Art Hindernis es ist. Da mit kann man ihm beibringen, sich adäquat zu verhalten;
  z. B. den Müll aufsaugen, aber Bäume Stehen lassen usw.

**- OffLimits**
  wenn er sich außerhalb der Welt befindet, fällt er sozusagen vom Tisch und ist zerstört.

**- DaGewesen**
  er hat ein Gedächtnis und merkt sich, auf welchem Feld er schon war und welche Anweisungen er bekam.

Außerdem wissen sie immer:

- wo sie stehen,
- in welche Richtung sie sehen,
- wie viele Schritte sie seit Start gelaufen sind,
- wie der Batteriestand ist.

------------------------------------------------------------------

Mögliche Aufgaben:

- Dem Roboter neue Methoden beibringen
- Die Handsteuerung erweitern
- programmgesteuerte Aktionen ausführen lassen.
- Aus einem dummen extern gesteuerten Roboter einen autonomen machen.

**Der Roboter Karel vom Typ TKarel**

ist ein Exemplar der 2. Generation. Er basiert auf der Technik von RD1 und wurde darauf aufbauend weiter entwickelt. Säuberungs- und Entsorgungsaufgaben

kann er spielend erledigen. Zur Unterscheidung hat er ein blaues Fahrgestell. Sein Sprachumfang und seine Fähigkeiten sind wie bei TRobot, aber teilweise verbessert und für *Transportaufgaben* erweitert:

**- Vor**
wie oben, aber er bleibt stehen, wenn das Feld besetzt ist.

**- Aufnehmen**
Das Objekt von dem nächsten Feld in Laufrichtung wegnehmen und In seinen Container stapeln.

**- Ablegen**
Das oberste Objekt aus dem Container (LIFO) auf dem nächsten Feld in Laufrichtung absetzen.

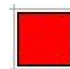
**- Einlagern**
Ein Item aus dem Container nehmen und an das Lagerhaus abgeben.

**- Auslagern**
Ein Item vom Lagerhaus abholen und in den Container laden.

**- VorDemAbgrund**
Karel kann prüfen, ob er mit dem nächsten Schritt in den Abgrund fallen wird.

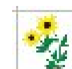
## 1.4 Sachen (Matters)

sind unbewegliche Dinge in der Welt. Sie haben normalerweise keine aktiven Methoden, sondern nur solche zu ihrer Gestaltung (Set..) und um Auskunft zu geben (Get..).



Standardmäßig haben sie wie alle Items einen rechteckigen Grundriß, der in ein Feld paßt. Hier sind die Farben gesetzt.

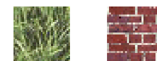
Alle Sachen (und Items) können ein Bild tragen, das dann den Grundriß überdeckt. Damit läßt sich eine vielgestaltige Welt machen.



Die Bilder müssen vom bmp-Format sein und sollten eine Größe von 28 x 28 pix haben. Zweckmäßigerweise werden alle neuen Bilder in das Verzeichnis 'Bilder' innerhalb das Projektordners kopiert.

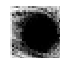
Vorhandenen Matters sind:

- Haus, Baum, Gras, Stein,



- ein Lagerhaus,
in das Objekte ein- und ausgelagert werden können; dazu steht Karel vor dem Lagerhaus und führt seine Methode *Einlagern* oder *Auslagern* aus,

- ein 'Schwarzes Loch'
in dem alle Objekte spurlos verschwinden und daher auch ge eignet für eine rückstandsfreie (kein Feinstaub;-) Müllentsor- gung.



**Programmierhinweis:**
Bild einfügen z. B. eine Blume

**Blume.SetBild ('.\bilder\blume.bmp');**

# 2. Didaktische Konzeption

**Programmierunterricht?**

Die klassischen Karels, Karas, Nikis & Co. sind für die Einführung in die IMPERATIVE Programmierung entwickelt, nicht für OOP. Es wird am Anfang ein Roboter (Computer) programmiert, aber kein Programmsystem entwickelt. Deshalb stellen sie nur das dazu erforderliche Aktionsrepertoire zur Verfügung, um operative Aufgaben (einfache und schwierige Algorithmusprobleme) lösen zu können. Insofern sind es nicht nur Mini-Languages, sondern auch geschlossene Systeme. Sie lassen, bis auf wenige Ausnahmen, keine Gestaltung der Welt im Sinne von 00-Modellierung zu.
Wer das will, sollte besser gleich den Original-Karel nehmen [1).]

**DELPHI-Karel**

- ist ein spielerischer Weg für den objektorientierten Einstieg in die Programmierung ('OOP von Anfang an'). Es werden schon in der ersten Stunde Objekte angesprochen und manipuliert mit der klassischen und leicht verständlichen OO-Notation: *Objektbezeicher.Methodenbezeichner,* z. B. *Robot.Vor.*

- soll in OO-Sichtweisen einführen und daneben auch imperative Strukturen vermitteln, aber nicht als alleinigem Schwerpunkt. Er ist deshalb auch nicht als Automat angelegt, sondern benutzt eine reale Programmiersprache.

-
Mit der offenen Architektur von DELPHI-Karel sollen Möglichkeiten der OO-Modellierung eröffnet werden:

- DELPHI-Karel fördert das Finden und Gestalten von neuen Welten und ihren Objekten (--> Critters u. Matters); es stimuliert die Phantasie der Schüler und fördert das Denken in Objektzusammenhängen.

- Es muß deshalb nicht nur unveränderbar einen Karel geben, es können auch Babies, Autos, Hunde oder Schachfiguren sein. So wird auch z. B. ein objektorientierter Karel nicht endlos beepers hinterlassen können oder nur mitzählen, wie oft er die Aktion 'Aufheben' ausgeführt hat, sondern er sollte einen Laderaum als Attribut haben, in den die aufgesammelten Objekte tatsächlich getan und wieder herausgenommen werden können.

- Der Ur-Robot RD1 hat nur ein minimales Methodenrepertoire, welches dazu auffordert, erweitert zu werden, damit dann erste operative Aufgaben besser gelöst werden können. Das kann sehr klein anfangen, aber auch sehr weit entwickelt werden, je nach Unterrichtszielen.

- Kern der objektorientierten Programmierung ist die Vererbung und es ist hier so naheliegend und einfach, sehr schnell eine Unterklasse abzuleiten, gleichermaßen eine eigene Welt mit ein paar Blumen, Bäumen und Häusern zu bepflanzen. Beziehungen werden auf eine propädeutische Weise erfahrbar und sind weniger abstrakt als z. B. die Aggregation einer Adresse in eine Personenklasse usw. Es wäre schade, diese Chance nicht zu nutzen.

In Karel D. Robot finden sich Ideen von Richard Pattis wieder aus "Karel++, A Gentle Introduction to the Art of Object-Orientated Programming", die Neuauflage von Karel the Robot aus 1981. Es ist aber nicht die 1:1 Portierung nach Delphi/Object Pascal, sondern eine eigenständige Entwicklung mit teilweise anderen weitergehenden Möglichkeiten. Der Schüler kann nicht nur unmittelbar visuell überprüfen, was er gerade programmiert hat, sondern der Roboter gibt in Konfliktsituationen Meldungen aus, um auf unzureichende Anweisungen aufmerksam zu machen.

- **Zur Programmierumgebung**

**Mini-Language vs. Objektpascal / Delphi**

Eine echte Mini-Language verfügt über einen abgeschlossenen, stark begrenzten Umfang an Schlüsselwörtern, Delphi hat ein paar mehr.

DELPHI-Karel ist ein offenes System für den möglichst streßfreien Anfangsunterricht auf der Ebene einer Mini-Language und den gleitenden Übergang in eine professionelle Programmiersprache (Delphi-Programmierumgebung). Die Vorteile: kein Wechsel der Arbeitsumgebung, kein Wechsel der Sprache, daher kein Zeitverlust wie beim Umstieg von einer didaktisch reduzierten auf eine 'richtige' Programmiersprache und kein Bruch in der Denkweise durch eine kontinuierliche Weiterentwicklung des Programmierstils.

Die oftmals geäußerten Vorbehalte gegen DELPHI als ein für den Schulunterricht zu mächtiges, daher unbrauchbares Entwicklungssystem können wir nicht teilen. Die Unterrichtsergebnisse bestätigen das.

Nach dem ersten Starten von DELPHI-Karel ist ein lauffähiges ansprechendes Programm vorhanden, das zunächst nur erkundet und leicht modifiziert wird. Der Schüler muß also nicht wie üblich erst (proprietären Delphi-) Code schreiben, um dann ein meist bescheidenes Progrämmchen zu sehen. Mit wenigen Zeilen kann ein Roboter erzeugt und bewegt werden.

Der vorhandene Karel verfügt nur über die ersten operativen Methoden (neben den üblichen Set- und Get-Methoden, die man aber zu Beginn nicht unbedingt braucht). Diese operativen Methoden bilden die Mini-Language von Delphi-Karel, die nach und nach erweitert wird durch selbst entwickelte Karel-Methoden und die Pascal-Schlüsselwörter zur Ablaufsteuerung (if, while, NOT usw.). Es wird hier in der Selbstbeschränkung des Lehrers liegen, sich hier mit speziellen Delphi-Konstrukten zurück zu halten.

## 2.1 Unterrichtseinsatz

- **Einführung in die (objektorientierte) Programmierung**

Möglicher Verlauf in 4 Phasen:

1. Anfangsunterricht in der Welt 'Trainingscamp'

   - Bedienung des Compilers
   - Erkunden von Karels Welt - die erste OO-Sicht: Was gibt es? Was passiert?
   - Editieren im Controlformular
     (zusätzliche Buttons für manuelle Steuerung einbauen, Objekte manipulieren, erzeugen, die ersten Methoden zur Bewegung des Roboters entwickeln)

2. komplexere Methoden bauen in 'Clean City',
   um operative Aufgaben (reinigen, transportieren, selbständig Wege finden usw.) zu erledigen.

3. die neuen Methoden einer eigenen Roboterklasse (Unterklasse MyRobot , Vererbung) hinzufügen.

4. Programmierpraktikum.
   Eine eigene Welt bauen mit einigen aktiven und passiven Objekten.

- mit Schwerpunkt auf objektorientierter Modellierung mit *Delphi-Karel Light*

Die vollständige Karel D. Robot-Version ist inzwischen mit einer Fülle von Klassen und Funktionen ausgestattet, so daß viele reizvolle Möglichkeiten schon vergeben sind. Will man den Unterrichtsschwerpunkt mehr auf Modellierung legen, so steht dafür eine reduzierte Light-Version zur Verfügung, die nur die Klassen und Funktionen enthält, die unbedingt notwendig sind. Die Dokumentation ist entsprechend angepaßt. Alles weitere kann im Unterricht selbst entwickelt werden.

- **Wahlmöglichkeiten …**

bei den Robotern

  - RD1 vom Typ TRobot ist die Rohversion, die fast nichts kann, aber daher das größere Entwicklungspotential für Modellierung hat - die beste Wahl für mutige Lehrer und Schüler.

  - Karel vom Typ TKarel ist weitgehend entwickelt und abgesichert und daher eher für den klassischen Algorithmikteil ohne große Umwege geeignet.

bei der Editierung - wo arbeitet der Schüler?

- entweder zunächst im ControlFrm: Nachteil - es werden Robotermethoden in der falschen Lokalität entwickelt, ist aber zu Beginn einfacher als mit einer zusätzlichen Fachklasse gleichzeitig oder
- NICHT im ControlFrm, sondern alle neuen Roboter-Methoden werden sofort in einer leeren, aber bereits vorhandenen Unterklasse 'MyRobot' entwickelt und im ControlFrm nur aufgerufen. Das wäre der korrekte Ansatz, der unauffällig zum Fachkonzept-Denken hinführt.

**.. oder Sie folgen dem Tutorial.**

- **Einstieg über komplexe Systeme**

Der alte, als auch der kommende Berliner Rahmenlehrplan sehen einen Einstieg in die Informatik über die Analyse und Modifizierung eines mäßig komplexen Systems vor. Karel D. Robot ist mit seiner klaren Struktur wie geschaffen dafür.

- **Für den fortgeschrittenen Informatikunterricht**

Es sei an dieser Stelle noch einmal betont, daß Karel D. Robot für die Einführung in die Programmierung - und zwar objektorientiert von Anfang an - entwickelt wurde. Es läßt sich aber nicht verheimlichen, daß die bereits implementierten Features erlauben, das Programm für andere weitergehende und klassische Probleme der Informatik zu 'mißbrauchen'.

- Das Gedächtnis der Roboter und die Zugriffsmöglichkeit auf die Welt kann für Backtracking-Verfahren und kürzeste Wege und ihrer Visualisierung genutzt werden
- Das Actions-Protokoll kann für einen Teach-In-Modus eingesetzt werden
- Die Welt und die Items bilden eine solide einsatzfähige Grundlage für Brettspiele usw.
- Die vollständige Dokumentation und der vorhandene Quellcode bieten eine interessante Möglichkeit für Re-Engineering Softwareprojekte (z. B. andere Grafik-Technik, Nebenläufigkeit) oder als Basis für andere Themen.

Zum Schluß eine Bitte.

Call for papers

Liebe Kolleginnen und Kollegen,

der Nutzen und der Erfolg eines solchen Lernprogrammes hängt nicht zuletzt von der Verfügbarkeit und Qualität von Beispielen, Übungsaufgaben und Unterrichtssequenzen ab. Jetzt liegen die ersten Erfahrungen vor und sollten bald allen zugänglich gemacht werden.

Wenn Sie Ideen haben für Aufgaben, neue Welten oder vielleicht sogar eine fertige Unterrichtseinheit, sollten Sie nicht zögern, es mir mit einer kurzen Erläuterung per attachment zuzusenden. Es wird eine derartige Sammlung geben. Dabei kommt es nicht auf "Druckreife" an. Wenn Sie es wünschen, werden die Beiträge unter Ihrem Namen im OSZ-Handel-Web, einer der attraktivsten Informatik-Sites mit durchschnittlich 300.000 Zugriffen pro Monat, veröffentlicht. Bei Interesse können Sie auch in eine Mailingliste aufgenommen werden zum Austausch von Erfahrungen.

Siegfried Spolwig

-----------------------

1) Sehr detailliert ausgearbeitete Aufgaben für den Programmierunterricht
s. JavaKara (Aufgaben v. H. Gierhardt), die sich aber auch für Karel D. Robot einsetzen lassen.
http://www.educeth.ch/informatik/karatojava/javakara/material/

## 2.2 Links

### Zu Karel

| | |
|---|---|
| Bergin, Stehlik, Roberts, Pattis: | Karel++ A Gentle Introduction to the Art of Object-Oriented Programming |
| EducETH: | Kara: Lernumgebungen rund ums Programmieren |
| EducETH: | Applet und Lernaufgabe zu "Java Karel - Einführung in die imperative Programmierung" |
| Freiberger, U.: | Eine Übersicht über verschiedene Entwicklungen, die auf der Idee von "Karel, the Robot" basieren. |
| Pattis, R.: | Karel the Robot: A Gentle Introduction to the Art of Programming, Second Edition |

### Zu OOP und DELPHI

Einführung mit grafischen Objekten

http://www.oszhandel.de/gymnasium/faecher/informatik/u-sequenzen/index.htm

Objektorientierte Programmierung - OOA, OOD – OOP

http://www.oszhandel.de/gymnasium/faecher/informatik/oop/index.htm

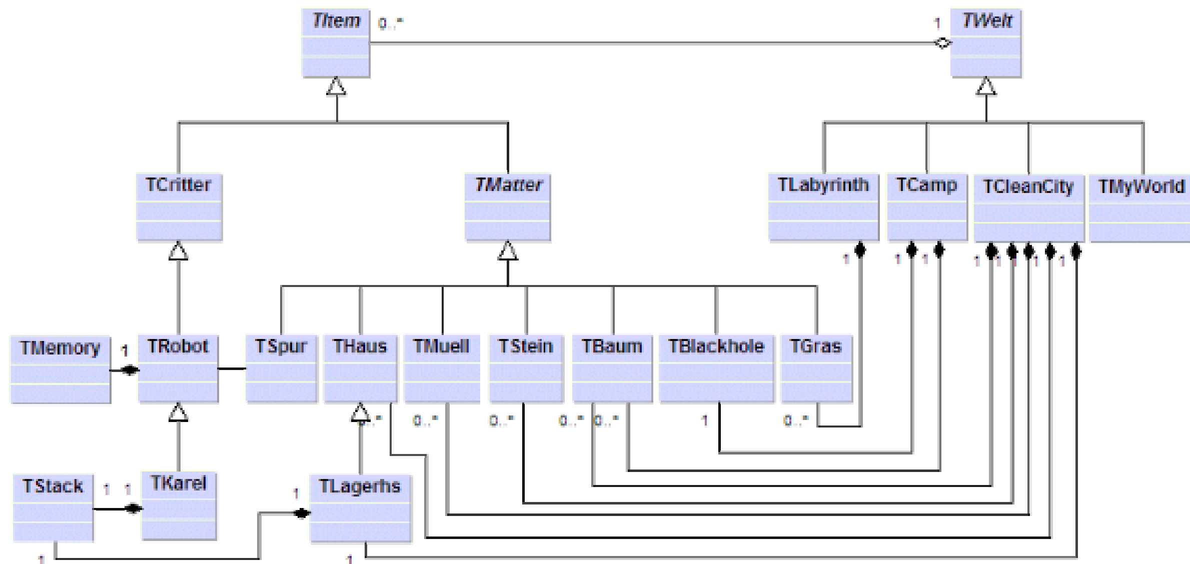
Delphi im Unterricht

http://www.oszhandel.de/gymnasium/faecher/informatik/delphi/index.htm

# 3. Dokumentation

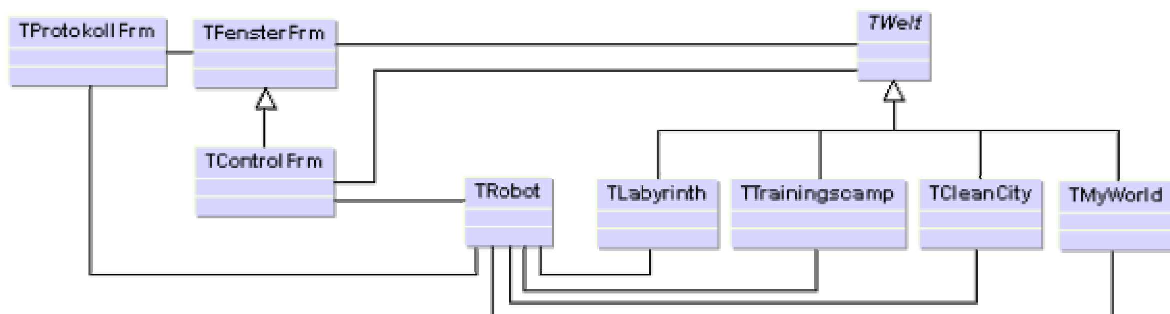
## 3.1 Klassendiagramme

### OOA-Diagramm (Fachklassen)




Bidirektionale Assoziationen zu TWelt sind wegen der Übersichtlichkeit weggelassen.

### OOD-Diagramm




Sehr stark verkürzte Form, die nur die wichtigen Beziehungen darstellen soll. Bemerkenswert ist die Aufteilung der GUI-Klassen. FensterFrm übernimmt die Darstellung, ControlFrm im Wesentlichen die Controls. ControlFrm ist das Hauptformular, in dem der Benutzer arbeitet

## 3.2 Klassen

| TItem<br>\|<br>*TMatter*<br>\|<br>TBaum | **Kurzbeschreibung** | Attribute | Methoden | Klassendiagramm |
|---|---|---|---|---|
| | Die Klasse bildet einen Baum ab. | | | |

```
UNIT uBaum;
(* ****************************************************************** *)
(* K L A S S E : TBaum                                                *)
(* ------------------------------------------------------------------ *)
(* Version     : 1.0                                                  *)
(* Autor       : (c) 2004, Siegfried Spolwig                          *)
(* Beschreibung: Die Klasse bildet einen Baum ab                      *)
(*                                                                    *)
(* Compiler    : Delphi 6                                             *)
(* Aenderungen : 0.9    01-MAR-04                                    *)
(* ****************************************************************** *)
INTERFACE
// =======================================================================
uses
  uMatter;
type
  TBaum = class(TMatter)
   private
   public
    constructor Create;       override;
    procedure   Init;
    procedure   Faellt;
    procedure   AlleeAnlegen; virtual;
  end;

(* -------------------- B e s c h r e i b u n g ------------------------
Oberklasse    :
Bezugsklassen : ..........      import:
Methoden
--------
Create
  Auftrag: Exemplar erzeugen und init.
  vorher :
  nachher: done.

Init
  Auftrag: Anfangswerte setzen
  vorher :
  nachher: Bild ist gesetzt

Set...
  Auftrag: Attribut schreiben
  vorher :
  nachher: Attribut ist gesetzt

Get...
  Auftrag: Attribut aus dem Objekt lesen
  vorher :
  nachher: -

AlleeAnlegen
  Auftrag: Setzt eine Reihe aus Bäumen
  Vorher: -
  Nachher: done
```

```
---------------------------------------------------------------------- *)
```

| TItem<br>\|<br>*TMatter*<br>\|<br>TBlackhole | **Kurzbeschreibung** | Attribute | Methoden | Klassendiagramm |
|---|---|---|---|---|
| | Die Klasse bildet ein 'Schwarzes Loch' ab, in dem Objekte spurlos verschwinden. | | | |

```
UNIT uBlackHole;
(* ******************************************************************* *)
(* K L A S S E : TBlackHole                                             *)
(* ------------------------------------------------------------------- *)
(* Version     : 0.9                                                     *)
(* Autor       : (c) 2005, Siegfried Spolwig                             *)
(* Beschreibung: Die Klasse bildet ein Schwarzes Loch ab, in dem Objekte*)
(*               spurlos verschwinden.                                    *)
(*                                                                         *)
(* Compiler    : Delphi 6                                                  *)
(* Aenderungen : 0.9   18-APR-05                                           *)
(* ******************************************************************** *)

INTERFACE
// ======================================================================
uses
  uMatter,
  uItem;

type
  TBlackHole = class(TMatter)
   public
    constructor Create;  override;
    procedure Init;
    procedure Verschlinge(it : TItem);
  end;

(* -------------------- B e s c h r e i b u n g -------------------------

Oberklasse    : TMatter
Bezugsklassen : -      import:

Methoden
--------
Create
  Auftrag: Exemplar erzeugen und init.
  vorher :
  nachher: done.

Init
  Auftrag: Anfangswerte setzen
  vorher :
  nachher: Bild gesetzt, Fuellfarbe schwarz.

procedure Verschlinge(it : TItem)
  Auftrag: Items rückstandslos entsorgen
  vorher : -
  nachher: it ist aus dem Arbeitsspeicher entfernt.
```

| TWelt<br>\|<br>TCleanCity | **Kurzbeschreibung** | Attribute | Methoden | Klassendiagramm |
|---|---|---|---|---|
| | Die Klasse enthält Steine, Häuser, Bäume, Hundehaufen, ein Lagerhaus.<br>Sie ist gedacht für die erweiterten Aktionen des Roboters. | | | |

```
UNIT uCleanCity;
(* ******************************************************************** *)
(* K L A S S E : TCleanCity                                               *)
(* -------------------------------------------------------------------- *)
(* Version      : 2.2                                                     *)
(* Autor        : (c) 2004, Siegfried Spolwig                             *)
(* Beschreibung: Die Klasse bildet die Welt 'CleanCity' ab.               *)
(*                                                                        *)
(* Compiler     : Delphi 6                                                *)
(* Aenderungen : 0.9    28-MAR-04                                        *)
(* ******************************************************************** *)

INTERFACE
// =======================================================================
uses
  uWelt,
  uStein,
  uBaum,
  uHaus,
  uLagerhaus,
  uMuell;

type
  TCleanCity = class(TWelt)
   protected
    Mauer    : TStein;
    Linde,
    Eiche    : TBaum;
    Haufen   : TMuell;
    Haus     : THaus;
    KaDeWe   : TLagerhaus;
   public
    constructor Create;   override;
    procedure   Init;
  end;

(* -------------------- B e s c h r e i b u n g -------------------------

Oberklasse    : TWelt
Bezugsklassen : -      import:


Methoden
--------

Create
  Auftrag: Exemlar erzeugen und init.
           Alle Objekte der vorigen Welt entfernen
  vorher :
  nachher: done.


Init
  Auftrag: Anfangswerte setzen
  vorher :
  nachher: Bäume, Mauern, Haufen, Häuser gesetzt
```

```
Set...
  Auftrag: Attribut schreiben
  vorher :
  nachher: Attribut ist gesetzt

Get...
  Auftrag: Attribut aus dem Objekt lesen
  vorher :
  nachher: -

---------------------------------------------------------------------- *)
```

<table>
<tr><td rowspan="2">TItem<br>|<br>TCritter</td><td>Kurzbeschreibung</td><td>Attribute</td><td>Methoden</td><td>Klassendiagramm</td></tr>
<tr><td colspan="4">TCritter ist die Oberklasse für alles, was sich bewegt und Aktionen ausführen kann (Creatures, Charaktere).<br><br>In dieser Version sind nur die Richtungen Nord, Ost, Sued, West implementiert.</td></tr>
</table>

```
UNIT uCritter;
(* ******************************************************************** *)
(* K L A S S E : TCritter                                               *)
(* -------------------------------------------------------------------- *)
(* Version     : 2.4                                                    *)
(* Autor       : (c) 2004, Siegfried Spolwig                            *)
(* Beschreibung: Die Klasse ist Oberklasse fuer alles, was sich bewegt  *)
(*               (Creatures, Charaktere).                               *)
(*                                                                      *)
(* Compiler    : Delphi 6.0                                             *)
(* Aenderungen : 0.9      06-MAR-04                                    *)
(*               2.2     20-JUL-04   s. Doc                             *)
(*               2.2.1. 14-MAR-05   TRichtung geändert                  *)
(*               2.4     18-APR-05   Vor geändert (Blackhole)           *)
(* ******************************************************************** *)

INTERFACE
// ==========================================================================
uses
  uItem;

type
  TRichtung = (Nord,Ost,Sued,West);
  TGeschwindigkeit = 1..10;

  TCritter = class(TItem)
   protected
    Richtung         : TRichtung;
    Geschwindigkeit : TGeschwindigkeit;
   public
    constructor Create;   override;
    procedure Init;
    procedure Vor;
    procedure RechtsDrehen;
    procedure SetRichtung (r: TRichtung);
    function  GetRichtung: TRichtung;
    procedure SetGeschwindigkeit (v : TGeschwindigkeit);
    function  GetGeschwindigkeit : TGeschwindigkeit;
    function  OffLimits : boolean;
   private
    procedure Step;

  end;

(* -------------------- B e s c h r e i b u n g -------------------------

Oberklasse    : TItem
```

```
Bezugsklassen : TFensterFrm - import Welt


Methoden
--------
Create
  Auftrag: Exemplar erzeugen und init.
  vorher : -
  nachher: -

Init
  Auftrag: Grundrissfarbe, Richtung setzen
  vorher :
  nachher: Grundrissfuellfarbe ist grau, Umrandung gelb, Richtung ist S,
           Geschwindigkeit ist 1.

Set...
  Auftrag: Attribut schreiben
  vorher : -
  nachher: Attribut ist gesetzt

Get...
  Auftrag: Attribut aus dem Objekt lesen
  vorher : -
  nachher: -

Vor
  Auftrag: einen Schritt in die aktuelle Richtung gehen
  vorher :
  nachher: auf folgendem Feld; in Welt sind alte Position abgemeldet und
neue
           Pos. angemeldet, wenn auf neuem Feld angekommen
           War das Feld besetzt, ist das andere Item entfernt.

RechtsDrehen
  Auftrag: Auf der aktuellen Position um 90 Grad nach rechts drehen
  vorher : -
  nachher: neue Richtung gesetzt

Offlimits
  Auftrag: Prüfen, ob sich ein Item ausserhalb der Welt befindet
  vorher : -
  nachher: true, wenn ausserhalb der Feldgrenzen, Item ist entfernt. Signal
           anzeigen.


--------------------------------------------------------------------- *)
```

<table>
<tr><td rowspan="2">TItem<br>|<br>*TMatter*<br>|<br>TGras</td><td>Kurzbeschreibung</td><td>Attribute</td><td>Methoden</td><td>Klassendiagramm</td></tr>
<tr><td colspan="4">Die Klasse bildet ein Grasstück ab.</td></tr>
</table>

```
UNIT uGras;
(* ******************************************************************** *)
(* K L A S S E : TGras                                                  *)
(* -------------------------------------------------------------------- *)
(* Version     : 1.2                                                    *)
(* Autor       : (c) 2004, Siegfried Spolwig                            *)
(* Beschreibung: Die Klasse bildet Gras ab                              *)
(*                                                                      *)
(* Compiler    : Delphi 6                                               *)
(* Aenderungen : 0.9   01-MAR-04                                        *)
(* ******************************************************************** *)
INTERFACE
// ==========================================================================
uses
  uMatter;
type
  TGras = class(TMatter)
   private
   public
    constructor Create;   override;
    procedure Init;
  end;

(* -------------------- B e s c h r e i b u n g -------------------------
Oberklasse     : TMatter
Bezugsklassen : -      import:
Methoden
--------
Create
  Auftrag: Exemplar erzeugen und initialisieren
  vorher :
  nachher: done.

Init
  Auftrag: Anfangswerte setzen
  vorher :
  nachher: Bild ist gesetzt

Set...
  Auftrag: Attribut schreiben
  vorher :
  nachher: Attribut ist gesetzt

Get...
  Auftrag: Attribut aus dem Objekt lesen
  vorher :
  nachher: -
---------------------------------------------------------------------- *)
```

<table>
<tr><td rowspan="2">TItem<br>|<br>TMatter<br>|<br>THaus</td><td>Kurzbeschreibung</td><td>Attribute</td><td>Methoden</td><td>Klassendiagramm</td></tr>
<tr><td colspan="4">Die Klasse bildet ein Haus ab.</td></tr>
</table>

```
UNIT uHaus;
(* ******************************************************************** *)
(* K L A S S E : THaus                                                  *)
(* -------------------------------------------------------------------- *)
(* Version     : 1.2                                                    *)
(* Autor       : (c) 2004, Siegfried Spolwig                            *)
(* Beschreibung: Die Klasse bildet ein Haus ab                          *)
(*                                                                      *)
(* Compiler    : Delphi 6                                               *)
(* Aenderungen : 0.9   01-MAR-04                                        *)
(* ******************************************************************** *)
INTERFACE
// ==========================================================================
uses
  uMatter;
type
  THaus = class(TMatter)
   private
   public
    constructor Create;   override;
    procedure Init;
  end;

(* -------------------- B e s c h r e i b u n g -------------------------
Oberklasse    : TMatter
Bezugsklassen : -      import:
Methoden
--------
Create
  Auftrag: Exemplar erzeugen und init.
  vorher :
  nachher: done.

Init
  Auftrag: Anfangswerte setzen
  vorher :
  nachher: Bild ist gesetzt

Set...
  Auftrag: Attribut schreiben
  vorher :
  nachher: Attribut ist gesetzt

Get...
  Auftrag: Attribut aus dem Objekt lesen
  vorher :
  nachher: -
---------------------------------------------------------------------- *)
```

| TItem | **Kurzbeschreibung** | Attribute | Methoden | Klassendiagramm |
|---|---|---|---|---|
| | TItem ist die Basisklasse für alles, was innerhalb der Welt existiert. Sie hat die Eigenschaften und Attribute, die alle abgeleiteten Klassen haben.<br><br>Von TItem werden (normalerweise) keine Exemplare erzeugt. | | | |

```
UNIT uItem;
(* ******************************************************************* *)
(* K L A S S E : TItems                                                 *)
(* -------------------------------------------------------------------- *)
(* Version     : 2.4                                                    *)
(* Autor       : (c) 2004, Siegfried Spolwig                            *)
(* Beschreibung: Die Klasse beschreibt alles, was in der Welt ist       *)
(*               Grundriss ist die (unsichtbare) Umrandung für alle     *)
(*                                                                      *)
(* Compiler    : Delphi 5.0                                             *)
(* Aenderungen : 0.9   01-MAR-04                                        *)
(*             : 2.2   20-JUL-04   s.Doc                                *)
(*               2.21  30-Nov-04   Loeschen war ohne Bild               *)
(*               2.4   19-APR-05   SetPos virtual (is doch klar!)       *)
(* ******************************************************************* *)

INTERFACE
// ==========================================================================
uses
  graphics,
  uGrafik;

type
  TItem = class (TObject)
   protected
    Grundriss    : TRechteck; // (unsichtbare) Umrandung für alle
    Bild         : TBild;
   private
    SpaltenPos   : char;
    ZeilenPos    : integer;   // Feldpos
    Farbe,
    Fuellfarbe   : TColor;
    Laenge,
    Hoehe,
    Xposition,
    Yposition    : integer;   // Pixelpos links oben
    BildGeladen : boolean;
   public
    constructor Create;  virtual;
    procedure Init;
    procedure SetPos (sp : char; zei : integer);  virtual;
    procedure SetSpaltenPos (sp : char);
    function  GetSpaltenPos : char;
    procedure SetZeilenPos(zei : integer);
    function  GetZeilenPos : integer;
    function  GetXPosition : integer;
    function  GetYPosition : integer;
    procedure SetFuellfarbe (cl : TColor);
    procedure SetFarbe (cl : TColor);
    function  GetFarbe : TColor;
    function  GetFuellFarbe : TColor;
    procedure SetBild(dateiname:string);
    function  BildIstGeladen : boolean;
    procedure Zeigen;    virtual;
    procedure Loeschen;  virtual;
```

```
    procedure Entfernen; virtual;
  end;

(* ------------------- B e s c h r e i b u n g ------------------------

Oberklasse    :
Bezugsklassen : TFensterFrm - import: Welt


Methoden
--------

Create
  Auftrag: Exemplar erzeugen und init
  vorher :
  nachher:

Init
  Auftrag: mit Anfangswerten versehen
  vorher :
  nachher:  Laenge ist Welt.Getqlaenge - 3 ; // Standardgröße festlegen
            Hoehe ist Welt.Getqlaenge - 3;
            Grundriss.Farbe ist schwarz
            Grundriss.FuellFarbe ist Zeichenblatt.Fuellfarbe;
            SpaltenPos ist ' '  // Nirwana-Geburtsort für alle
            ZeilenPos ist 0

SetPos (sp : char; zei : integer);
  Auftrag: Anmelden in Welt.
           Position im Quadranten zuweisen und daraus die Pixelpos links
           oben bestimmen und Standardgröße der Figur festlegen.
           sp in GROSSEN Buchstaben.
           SetPos soll NUR EINMAL als STARTPOSTION  verwendet werden. Mehr
           facher Aufruf führt zu unerwünschten Nebeneffekten; d.h. das Ob
           jekt wird mehrfach an verschiedenen Positionen angemeldet, ob
           wohl es nur einmal vorhanden ist!
  vorher : -
  nachher: Position übergeben und mit der Position angemeldet

Set...
  Auftrag: Attribut schreiben
  vorher : -
  nachher: Attribut ist gesetzt

Get...
  Auftrag: Attribut aus dem Objekt lesen
  vorher : -
  nachher: -

BildIstGeladen
  Anfrage: ob Bild für das Attribut geladen wurde
  vorher : -
  nachher: -

Zeigen
  Auftrag: Exemplar anzeigen
  vorher : -
  nachher: -


Loeschen
  Auftrag: Exemplar vom Bildschirm loeschen
  vorher : -
  nachher: ist unsichtbar
```

```
Entfernen
  Auftrag: Exemplar aus Speicher entfernen
  vorher :
  nachher: nicht mehr zugreifbar

---------------------------------------------------------------------- *)
```

<table>
<tr><td rowspan="2">TItem<br>|<br>TCritter<br>|<br>TRobot<br>|<br>TKarel</td><td>Kurzbeschreibung</td><td>Attribute</td><td>Methoden</td><td>Klassendiagramm</td></tr>
<tr><td colspan="4">Die Klasse bildet den Nachfolger von TRobot (R1D) ab, von dem er alle Attribute und Methoden geerbt hat.<br><br>Er zeigt erste Anzeichen von intelligentem Verhalten. Er prüft, ob das nächste Feld frei ist. Er kann bei der Welt anfragen, was vor ihm steht und in Kommunikation mit dem Hindernis treten, um zu entscheiden, wie er sich verhalten soll.<br>Kann Items aufladen und abladen, in das Lagerhaus bringen und abholen.<br>Bei Konflikten bleibt er stehen und zeigt eine Warnmeldung.</td></tr>
</table>

```
UNIT uKarel;
(* ******************************************************************** *)
(* K L A S S E : TKarel                                                  *)
(* -------------------------------------------------------------------- *)
(* Version      : 2.4                                                    *)
(* Autor        : (c) 2004, Siegfried Spolwig                            *)
(* Beschreibung: Die Klasse bildet den Roboter Karel der 2. Generation  *)
(*               ab.                                                     *)
(* Compiler     : Delphi 6                                               *)
(* Aenderungen : 0.9      11-MAR-04                                      *)
(*               2.3.2   14-FEB-05   VorDemAbgrund                       *)
(*               2.4     17-APR-05   Ein-, Auslagern, Ablegen erweitert  *)
(*                                   für Blackhole,GetLademenge umbenannt*)
(* ******************************************************************** *)

INTERFACE
// ==========================================================================
uses
  uRobot,
  uStack;

type
  TKarel = class(TRobot)
   protected
    Container : TStack;   // zum Aufladen von Objekten
   public
    constructor Create;   override;
    procedure Init;
    procedure Vor;
    procedure Aufnehmen;
    procedure Ablegen;
    procedure Einlagern;
    procedure Auslagern;
    function  GetLadeMenge  : integer; // Anzahl der Items im Container
    function  VorDemAbgrund : boolean;

   private
    procedure Aktualisieren;
  end;

(* -------------------- B e s c h r e i b u n g -------------------------

Oberklasse     : TRobot
```

```
Bezugsklassen : TZeit, TStack, TFensterFrm - importiert: Welt

Methoden
--------

Create
  Auftrag: Exemplar erzeugen und init.
  vorher :
  nachher:

Init
  Auftrag: Anfangswerte setzen
  vorher : -
  nachher:  Richtung ist S; Schrittzaehler ist 0; Batteriestand ist 40;
            Container ist leer; max. Lademenge ist 24; macht keine Spur.

Vor
  Auftrag: Exemplar geht 1 Feld in der aktuellen Richtung vorwärts oder
           bleibt stehen, wenn Batterie leer oder das Feld besetzt ist.
           Position in das Gedaechtnis.
  vorher : Batterie ist geladen
  nachher: neue Pos ist angemeldet, wenn neue Pos erreicht;
           vorige Pos ist abgemeldet (NIL!!). Fehlermeldung, wenn Feld be-
           setzt.

Aufnehmen
  Auftrag: Das Item von dem nächsten Feld in Laufrichtung in den Container
           stapeln.
  vorher : Container ist nicht voll.
  nachher: Item ist vom alten Feld  abgemeldet. Fehlermeldung bei vollem
           Container.

Ablegen
  Auftrag: Das oberste Item aus dem Container (LIFO) auf dem nächsten Feld
           in Laufrichtung absetzen.
  vorher : Container ist nicht leer und Feld ist frei.
  nachher: Item ist auf neuem Feld angemeldet. Fehlermeldung bei leerem
           Container.

Einlagern
  Auftrag: Item aus dem Container ins Lagerhaus abgeben
  vorher : Karel steht vor dem Lagerhaus und Container ist nicht leer, La
           Gerhaus ist nicht voll.
  nachher: Item ist dem Container entnommen und vom Lagerhaus angenommen.
           Fehlermeldung bei leerem Container.

Auslagern
  Auftrag: Item aus dem Lagerhaus nehmen und in den Container stapeln.
  vorher : Karel steht vor dem Lagerhaus, Container ist nicht voll und La
           Gerhaus ist nicht leer.
  nachher: done. Fehlermeldung bei vollem Container.


GetLadeMenge
  Anfrage: wie viele Objekte im Container sind
  vorher : -
  nachher: gibt Anzahl zurück

VorDemAbgrund
  Anfrage: ob die nächste Vor-Position OffLimits (außerhalb der Welt) ist.
  vorher : -
  nachher: true, wenn auf Aussenfeld und Richtung nach außerhalb der Welt
           zeigt

-------------------------------------------------------------------- *)
```

| TWelt<br>\|<br>TLabyrinth | **Kurzbeschreibung** | Attribute | Methoden | Klassendiagramm |
|---|---|---|---|---|
| | Die Klasse enthält nur Gras und eine Blume am Ziel. | | | |

```
UNIT uLabyrinth;
(* ******************************************************************** *)
(* K L A S S E : TLabyrint                                               *)
(* -------------------------------------------------------------------- *)
(* Version     : 2.2                                                     *)
(* Autor       : (c) 2004, Siegfried Spolwig                             *)
(* Beschreibung: Die Klasse bildet die Welt 'Labyrinth' ab.              *)
(*                                                                       *)
(* Compiler    : Delphi 6                                                *)
(* Aenderungen : 0.9   16-APR-04                                         *)
(* ******************************************************************** *)
INTERFACE
// =========================================================================
uses
  uWelt,
  uGras,
  uMatter;
type
  TLabyrinth = class(TWelt)
   protected
    Blume : TMatter;
    Gras  : TGras;
   public
    constructor Create;   override;
    procedure   Init;
  end;

(* -------------------- B e s c h r e i b u n g ------------------------
Oberklasse    : TWelt
Bezugsklassen : -      import:
Methoden
--------
Create
  Auftrag: Exemlar erzeugen und init.
           Alle Objekte der vorigen Welt entfernen
  vorher :
  nachher: done.

Init
  Auftrag: Anfangswerte setzen
  vorher :
  nachher: Gras gesetzt

Set...
  Auftrag: Attribut schreiben
  vorher :
  nachher: Attribut ist gesetzt

Get...
  Auftrag: Attribut aus dem Objekt lesen
  vorher :
  nachher: -
---------------------------------------------------------------------- *)
```

<table>
<tr><td rowspan="2">TItem<br>|<br>TMatter<br>|<br>THaus<br>|<br>TLagerhaus</td><td>Kurzbeschreibung</td><td>Attribute</td><td>Methoden</td><td>Klassendiagramm</td></tr>
<tr><td colspan="4">Die Klasse bildet ein Lagerhaus ab zum Ein- und Auslagern von Items. Die Lagermenge ist begrenzt durch die definierte Soll-Laenge.<br><br>Damit können Transportaufgaben für Karel gelöst werden. Wenn das Lagerhaus selbst von Karel an einen anderen Ort gebracht wird, bleibt der Inhalt vorhanden.</td></tr>
</table>

```
UNIT uLagerhaus;
(* ******************************************************************** *)
(* K L A S S E : TLagerhaus                                             *)
(* -------------------------------------------------------------------- *)
(* Version     : 0.9                                                    *)
(* Autor       : (c) 2005, Siegfried Spolwig                            *)
(* Beschreibung: Die Klasse bildet ein Haus ab, in dem beliebige Items  *)
(*               gelagert (LIFO) werden können.                         *)
(*                                                                      *)
(* Compiler    : Delphi 6                                               *)
(* Aenderungen : 0.9   18-APR-05                                        *)
(* ******************************************************************** *)

INTERFACE
// ==========================================================================
uses
  uHaus,
  uStack,
  uItem;

type
  TLagerhaus = class(THaus)
   protected
    Speicher : TStack;
   private
   public
    constructor Create;   override;
    procedure Init;
    procedure Annehmen(it : TItem);
    function  GetItem : TItem;
    function  GetLagermenge : integer;
    function  IstVoll : boolean;
  end;

(* -------------------- B e s c h r e i b u n g -------------------------

Oberklasse    : TMatter
Bezugsklassen : -      import:


Methoden
--------

Create
  Auftrag: Exemlar erzeugen und init.
  vorher :
  nachher: done.

Init
  Auftrag: Anfangswerte setzen
  vorher :
  nachher: Speicher ist leer, Bild ist gesetzt, Fassungsvermögen ist 25.
```

```
procedure Annehmen(it : TItem)
  Auftrag: Item in den Speicher. Ist der Speicher voll, geschieht nichts.
  vorher : Speicher ist nicht voll
  nachher: Anzahl ist um 1 erhoeht (auch wenn nichts uebergeben wurde!!)

procedure GetItem;
  Auftrag: Item aus dem Speicher entnehmen.
  vorher : Speicher ist nicht leer
  nachher: Anzahl ist um 1 vermindert. Ist der Speicher leer, geschieht
nichts.

function GetLagermenge : integer;
  Anfrage: nach der Anzahl der im Speicher gelagerten Items.
  vorher : -
  nachher: -

function IstVoll : boolean;
  Anfrage: ob der Speicher voll ist.
  vorher : -
  nachher: -

Set...
  Auftrag: Attribut schreiben
  vorher :
  nachher: Attribut ist gesetzt

Get...
  Auftrag: Attribut aus dem Objekt lesen
  vorher :
  nachher: -

-------------------------------------------------------------------- *)
```

<table>
<tr><td rowspan="2">TItem<br>|<br>TMatter</td><td>Kurzbeschreibung</td><td>Attribute</td><td>Methoden</td><td>Klassendiagramm</td></tr>
<tr><td colspan="4">TMatter ist die abstrakte Oberklasse für alle unbeweglichen Sachen.<br>Sie ist nur der allgemeinen Strukturierung wegen eingezogen.<br><br>Von TMatter sollen keine Exemplare erzeugt werden. Es ist günstiger, Unterklassen zu bilden, die dann unterschiedlich von den Objekten angesprochen werden können.</td></tr>
</table>

```
UNIT uMatter;
(* ******************************************************************** *)
(* K L A S S E : TMatter                                                  *)
(* -------------------------------------------------------------------- *)
(* Version     : 0.9.1                                                    *)
(* Autor       : (c) 2004, Siegfried Spolwig                              *)
(* Beschreibung: Die Klasse ist abstrakte Basisklasse für alle unbeweg- *)
(*               lichen Sachen.                                          *)
(*               Sie ist nur der allgemeinen Strukturierung wegen         *)
(*               eingezogen.                                              *)
(*                                                                        *)
(* Compiler    : Delphi 6.0                                               *)
(* Aenderungen : 0.9   03-MAR-04                                          *)
(* ******************************************************************** *)
INTERFACE
// ======================================================================
uses
  uItem;
type
  TMatter = class(TItem)
            end;
(* -------------------- B e s c h r e i b u n g -------------------------
Oberklasse     : TItem
Bezugsklassen : -       import:
Methoden
--------
  --
---------------------------------------------------------------------- *)
```

<table>
<tr><td rowspan="2">TListe<br>|<br>TMemory</td><td>Kurzbeschreibung</td><td>Attribute</td><td>Methoden</td><td>Klassendiagram</td></tr>
<tr><td colspan="4">Die Klasse bildet den Container ab, der Gedächtnisinhalte (shadows) aufnimmt.</td></tr>
</table>

```
UNIT uMemory;
(* ******************************************************************** *)
(* K L A S S E   : TMemory                                               *)
(* -------------------------------------------------------------------- *)
(* Version       : 1.0                                                    *)
(* Autor         : (c) S. Spolwig, OSZ-Handel I, 10997 Berlin             *)
(*                                                                        *)
(* Aufgabe       : Container zum Speichern (Merken) von Informationen,   *)
(*                 - z.B., wo Karel schon gewesen ist - die in Shadow    *)
(*                 aufgenommen sind.                                      *)
(* Compiler      : Delphi 6.0                                             *)
(*               : V.   1.0   - 23-JUN-04   Elementtyp ist TShadow        *)
(* ******************************************************************** *)
INTERFACE
```

```
(* ==================================================================== *)
uses
  uDListe,      // import: TList, Enthaelt
  uShadow;
type
  TMemory = class (TListe)
    public
     function Enthaelt (eintrag : TShadow) : boolean; virtual;
  end;

(*  ----------------- B e s c h r e i b u n g -------------------------
Oberklasse     : TListe
Bezugsklassen  : TShadow
Methoden
--------

Enthaelt (eintrag : TShadow): boolean;
  Anfrage : ob ein Element mit den mit der betreffenden Spalten-/Zeilenpos
            enthalten ist
  vorher  : S. ist nicht leer.
  nachher : true, wenn Positionen von eintrag mit denen eines Elements im
            Gedaechtnis übereinstimmen.
----------------------------------------------------------------------- *)
```

| TItem<br>\|<br>*TMatter*<br>\|<br>TMuell | **Kurzbeschreibung** | Attribute | Methoden | Klassendiagramm |
|---|---|---|---|---|
| | Die Klasse bildet einen Hundehaufen  ab. | | | |

```
UNIT uMuell;
(* ******************************************************************** *)
(* K L A S S E : TMuell                                                  *)
(* -------------------------------------------------------------------- *)
(* Version     : 1.2                                                      *)
(* Autor       : (c) 2004, Siegfried Spolwig                              *)
(* Beschreibung: Die Klasse bildet einen Hundehaufen ab                   *)
(*                                                                        *)
(* Compiler    : Delphi 6                                                 *)
(* Aenderungen : 0.9   01-MAR-04                                          *)
(* ******************************************************************** *)
INTERFACE
// ==========================================================================
uses
  uMatter;
type
  TMuell = class(TMatter)
   private
   public
    constructor Create;   override;
    procedure Init;
  end;

(* -------------------- B e s c h r e i b u n g -------------------------
Oberklasse    : TMatter
Bezugsklassen : -      import:
Methoden
--------
```

```
Create
  Auftrag: Exemplar erzeugen und init
  vorher :
  nachher:

Init
  Auftrag: mit Anfangswerten versehen
  vorher :
  nachher: Bild ist gesetzt

Set...
  Auftrag: Attribut schreiben
  vorher : -
  nachher: Attribut ist gesetzt

Get...
  Auftrag: Attribut aus dem Objekt lesen
  vorher : -
  nachher: -
---------------------------------------------------------------------- *
```

<table>
<tr><td rowspan="2">TWelt<br>|<br>TMyWorld</td><td>Kurzbeschreibung</td><td>Attribute</td><td>Methoden</td><td>Klassendiagramm</td></tr>
<tr><td colspan="4">Die Klasse ist leer (bzw. nicht vorhanden)<br><br>Sie ist gedacht für den Entwurf einer eigenen Welt mit eigenen Critters und Matters.<br><br>Viel Spaß!</td></tr>
</table>

```
UNIT uMyWorld;
(* ******************************************************************** *)
(* K L A S S E : TMyWorld                                               *)
(* -------------------------------------------------------------------- *)
(* Version     :                                                        *)
(* Autor       :                                                        *)
(* Beschreibung:                                                        *)
(*                                                                      *)
(* Compiler    : Delphi 6                                               *)
(* Aenderungen :                                                        *)
(* ******************************************************************** *)
INTERFACE
// ======================================================================

type
  TMyWorld = class(TWelt)
   private
   public
  end;
(* -------------------------------------------------------------------- *)
```

<table>
<tr><td rowspan="2">TItem<br>|<br>TCritter<br>|<br>TRobot</td><td>Kurzbeschreibung</td><td>Attribute</td><td>Methoden</td><td>Klassendiagramm</td></tr>
<tr><td colspan="4">Die Klasse bildet den Ur-Roboter RD1 (Robot Device 1) der 1. Generation ab. Er kann nur Vor-Gehen, nach rechts drehen und prüfen, was vor ihm ist.<br>Er merkt sich, wo er schon war.<br><br>Er hat Sensoren, die aber noch nicht aktiviert sind.</td></tr>
</table>

```
UNIT uRobot;
(* ****************************************************************** *)
(* K L A S S E : TRobot                                               *)
(* ------------------------------------------------------------------ *)
(* Version     : 2.4                                                  *)
(* Autor       : (c) 2004-2005, Siegfried Spolwig                     *)
(* Beschreibung: Die Klasse bildet den Ur-Roboter RD1 der 1. Generation *)
(*               ab. Er ist nur mit wenigen Features ausgestattet (s.u.)*)
(*               Felder, die er passiert hat, sind leer.              *)
(*                                                                    *)
(* Compiler    : Delphi 6                                             *)
(* Aenderungen : 0.9     11-MAR-04                                    *)
(*               2.2    20-JUL-04   s. Doc                            *)
(*               2.21   30-Nov-04   macht Spur                        *)
(*               2.2.1  16-DEC-04   keine Spur bei OFFlimits; machtSpur *)
(*                                  in Vor geändert                   *)
(*               2.4    18-APR-05   Vor geändert (Blackhole)          *)
(* ****************************************************************** *)

INTERFACE
// =======================================================================
uses
  uGrafik,
  uItem,
  uCritter,
  uMemory;

type
  TRobot = class(TCritter)
   protected
    Form,
    Lampe          : TKreis;
    Gedaechtnis    : TMemory;
    Schrittzaehler: integer;
    Batteriestand : integer;
    MachtSpur      : boolean;

   public
    constructor Create;  override;
    procedure Init;
    procedure SetPos(spalte: char; zeile: integer);  override;
    procedure Vor;
    procedure RechtsDrehen;
    function  HindernisPruefen : TItem;
    function  VorneFrei : boolean;
    function  BatterieLeer : boolean;
    procedure BatterieAufladen;
    procedure MacheSpur;
    procedure KeineSpur;
    function  DaGewesen : boolean;
    function  GetSchrittZaehler : integer;
    function  GetBatteriestand : integer;

    procedure Zeigen;     override;
```

```
    procedure Loeschen;   override;
    procedure Entfernen;  override;

    procedure InsGedaechtnis(action : string);
    function  GetGedaechtnis : TMemory;

   private
    procedure Step;
    procedure Aktualisieren;
  end;

(* ------------------- B e s c h r e i b u n g ------------------------

Oberklasse    : TCritter
Bezugsklassen : TMemory, TZeit, TFensterFrm - importiert: Welt

Methoden
--------

Create
  Auftrag: Exemplar erzeugen und init.
  vorher :
  nachher:

Init
  Auftrag: Anfangswerte setzen
  vorher : -
  nachher: Richtung ist S, Schrittzaehler ist 0, Batteriestand ist 40,
           Geschwindigkeit ist 1; Gedaechtnis ist leer.

Set...
  Auftrag: Attribut schreiben
  vorher : -
  nachher: Attribut ist gesetzt

Get...
  Auftrag: Attribut aus dem Objekt lesen
  vorher : -
  nachher: -

Vor
  Auftrag: Exemplar geht 1 Feld in der aktuellen Richtung vorwärts unabhän-
           gig davon, ob das Feld besetzt ist
  vorher : Batterie ist geladen
  nachher: Eins weiter oder Stehenbleiben, wenn HindernisPruefen oder
           Batteriestand = 0

RechtsDrehen
  Auftrag: Auf der aktuellen Position um 90 Grad nach rechts drehen
  vorher : -
  nachher: neue Richtung gesetzt. Lampe zeigt in neue Richtung.

HindernisPruefen
  Anfrage: Item auf dem nächsten vorausliegenden Feld holen, damit es
           angesprochen werden kann. Rote Warnlampe an.
  vorher : -
  nachher: liefert Item oder NIL

VorneFrei
  Anfrage: prüfen, nächstes Feld frei ist
  vorher : -
  nachher: True, wenn Feld leer (NIL) ist.
```

```
BatterieLeer
  Anfrage: ob Batterie leer ist
  vorher : -
  nachher: true, wenn Batteriestand = 0

BatterieAufladen
  Auftrag: leere Batterie laden
  vorher : -
  nachher: Ladung ist 40 Einheiten

MacheSpur
  Auftrag: Eine Spur auf dem Feld hinterlassen, das der Roboter als voriges
           betreten hat
  vorher : -
  nachher: Spur (ein Item!) ist gesetzt.

KeineSpur
  Auftrag: Spur abschalten
  vorher : -
  nachher: Machtspur ist false.

InsGedaechtnis(action : string);
  Auftrag: action,i.e. letzte Anweisung (z. Z. 'Vor', 'RechtsDrehen'),
           Spalten- u. Zeilenposition merken. Startposition wird nicht er-
           fasst.
  vorher : -
  nachher: Action, Spalte, Zeile im Gedaechtnis (Speicher, nicht Platte!)

DaGewesen
  Anfrage: ob Roboter das naechste Feld in Laufrichtung schon betreten hat-
           te.
  vorher : Das Feld war mit 'Vor' erreicht. Ausgangsposition wird nicht er
           fasst.
  nachher: true, wenn ein Feld mit der Folgeposition im Gedächtnis gefunden
           wird

--------------------------------------------------------------------- *)
```

<table>
<tr><td rowspan="2">TShadow</td><td>Kurzbeschreibung</td><td>Attribute</td><td>Methoden</td><td>Klassendiagram</td></tr>
<tr><td colspan="4">Die Klasse bildet einen Gedaechtnisinhalt ab.</td></tr>
</table>

```
UNIT uShadow;
(* ******************************************************************* *)
(* K L A S S E : TShadow                                                *)
(* ------------------------------------------------------------------- *)
(* Version     : 2.3                                                    *)
(* Autor       : (c) 2004, Siegfried Spolwig                            *)
(* Beschreibung: Die Klasse dient zum Zusammenfassen von Informationen  *)
(*               eines Items (z. B. Karel), die ins Gedaechtnis über-   *)
(*               nommen oder gespeichert werden sollen.                 *)
(* Compiler    : Delphi 6                                               *)
(*                                                                      *)
(* Aenderungen : 2.3  25-JUN-04                                         *)
(* ******************************************************************* *)
INTERFACE
// ==========================================================================
type
  TShadow = class(TObject)
   private
    SpaltenPos  : char;      // Feldpos
    ZeilenPos   : integer;
    LastAction  : string[15];
   public
    constructor Create;
    procedure Init;
    procedure SetSpaltenPos (sp : char);
    function  GetSpaltenPos : char;
    procedure SetZeilenPos (zei : integer);
    function  GetZeilenPos : integer;
    procedure SetLastAction (la : string);
    function  GetLastAction : string;
  end;

(* -------------------- B e s c h r e i b u n g -------------------------
Oberklasse    : -
Bezugsklassen : -      import:
Methoden
--------
Create
  Auftrag: Exemlar erzeugen und init.
  vorher :
  nachher: done.

Init
  Auftrag: Anfangswerte setzen
  vorher :
  nachher: SpaltenPos ist ' ', ZeilenPos ist 0, LastAction ist leer.


Set...
  Auftrag: Wert in das Attribut schreiben
  vorher :
  nachher: -

Get...
  Auftrag: Attribut aus dem Objekt lesen
  vorher :
  nachher: -
--------------------------------------------------------------------- *)
```

| TItem<br>\|<br>*TMatter*<br>\|<br>TSpur | **Kurzbeschreibung** | Attribute | Methoden | Klassendiagramm |
|---|---|---|---|---|
| | Die Klasse bildet einen Schritt ab, den Karel gegangen ist. Es ist keine rein grafische Markierung, sondern ein Item, das Karel ablegt und wieder aufnehmen kann.<br>Die Fussabdrücke zeigen jeweils in Laufrichtung. | | | |

```
UNIT uSpur;
(* ****************************************************************** *)
(* K L A S S E : TSpur                                                 *)
(* ------------------------------------------------------------------ *)
(* Version     : 1.2                                                   *)
(* Autor       : (c) 2004, Siegfried Spolwig                           *)
(* Beschreibung: Die Klasse bildet eine Fußspur ab                     *)
(*                                                                     *)
(* Compiler    : Delphi 6                                              *)
(* Aenderungen : V. 0.9    01-MAR-04                                   *)
(* ****************************************************************** *)
INTERFACE
// =====================================================================
uses
  uMatter,
  uCritter;
type
  TSpur = class(TMatter)
   private
   public
    constructor Create;       override;
    procedure   Init;
    procedure Zeigen (wohin : TRichtung);  reintroduce;
  end;

(* -------------------- B e s c h r e i b u n g -------------------------
Oberklasse    : TMatter
Bezugsklassen : TCritter - importiert : Richtung
Methoden
--------
Create
  Auftrag: Exemlar erzeugen und init.
  vorher :
  nachher: done.

Init
  Auftrag: Anfangswerte setzen
  vorher :
  nachher: Bild ist gesetzt, Spur nach Süden

Set...
  Auftrag: Attribut schreiben

  vorher :
  nachher: Attribut ist gesetzt

Get...
  Auftrag: Attribut aus dem Objekt lesen
  vorher :
  nachher: -

Zeigen (wohin: TRichtung);
  Auftrag: wohin die Reise geht
  vorher : -
```

```
  nachher: -
---------------------------------------------------------------------- *)
```

| TStack | **Kurzbeschreibung** | Attribute | Methoden | Klassendiagram |
| --- | --- | --- | --- | --- |
| | Die Klasse bildet den Container ab, den Karel und das Lagerhaus besitzen, um Items aufnehmen und wieder herausnehmen zu können.<br>TStack arbeitet nach dem LIFO-Prinzip. | | | |

```
UNIT uStack;
(* ******************************************************************** *)
(* K L A S S E    : TStack - generische Klasse                            *)
(* -------------------------------------------------------------------- *)
(* Version        : 1.1                                                  *)
(* Autor          : (c) S. Spolwig, OSZ-Handel I, 10997 Berlin           *)
(*                                                                       *)
(* Aufgabe        : Allgemeine statische Liste (Stack)zur Verwaltung     *)
(*                  beliebiger Objekte. Hinzufuegen eines Elements am    *)
(*                  Ende (Push), Entnehmen (Pop) vom Ende.               *)
(*                  In abgeleiteten Klassen sollten der Elemententyp de- *)
(*                  klariert und die abstrakten Methoden Load, Store     *)
(*                  ueberschrieben werden.                               *)
(* Compiler       : Delphi 4.0                                           *)
(* Aenderung      : V. 1.0    - 20-DEC-99                                *)
(*                     1.1    - 18-APR-05  SollLaenge, um unterschied-   *)
(*                                         liche Listenlaengen festzulegen*)
(* ******************************************************************** *)

INTERFACE
(* ==================================================================== *)

const
  MAXLAENGE = 200; // reicht für alle Felder

type
  TElement = Pointer;

  TStack   = class (TObject)
              private
               Kollektion : array [0..MAXLAENGE + 1] of TElement;
               SollLaenge,
               LiLaenge    : Word;    // Anzahl der belegten Elemente
              public
               constructor Create;                       virtual;
               procedure Init;

               procedure Push (Elem : TElement);
               function  Pop: TElement;
               procedure RemoveAll;
               procedure SetSollLaenge (sl : Word);
               function  GetLen   : integer ;
               function  IsEmpty : boolean;
               function  IsFull   : boolean;

               procedure Load (Dateiname : string);    virtual;  abstract;
               procedure Store(Dateiname : string);    virtual;  abstract;
             end;

(*  ------------------ B e s c h r e i b u n g -------------------------

Oberklasse      : TObject
```

```
Bezugsklassen  : -


Methoden
--------

Create
  Auftrag : Leere L. erzeugen
  vorher  : -
  nachher : Lilaenge ist Null.

Init
  Auftrag : L. initial.
  vorher  : L. ist vorhanden
  nachher : Lilaenge ist Null, SollLaenge ist MAXLAENGE.


Pop
  Anfrage : Zugriff auf das letzte Element zum Lesen
  vorher  : Die Liste ist init.
  nachher : Liefert das aktuelle Element. Danach ist es aus der Liste ent-
            fernt. Wenn die Liste leer war wird NIL geliefert.

Push
  Auftrag : Neues Element an das Ende der Liste anhaengen.
            Wenn die Liste leer war, ist das neue El. das erste.
  vorher  : Die Liste ist initialisiert
  nachher : Listenlaenge ist um eins erhoeht. Ist die Liste voll,
            geschieht nichts.

RemoveAll
  Auftrag: Alle Elemente aus der Liste entfernen
  vorher : -
  nachher: Die Liste ist leer.

IsEmpty
  Anfrage: ob Liste leer ist
  vorher : Die Liste ist initialisiert.
  nachher: True, wenn die Liste leer ist.

IsFull
  Anfrage: ob Liste voll ist
  vorher : Die Liste ist initialisiert.
  nachher: True, wenn LiLaenge = MAXLISTE

Set...
  Auftrag: Attribut schreiben
  vorher : -
  nachher: Attribut ist gesetzt

GetLen
  Anfrage: gibt Listenlaenge zurueck
  vorher : Liste ist initialisiert
  nachher: -


Load
  Auftrag: Liste aus externer Datei laden.
  vorher : nicht definiert
  nachher: nicht definiert.


Store
  Auftrag: Liste in externe Datei speichern
  vorher : nicht definiert.
```

```
  nachher: nicht definiert.

------------------------------------------------------------------------ *)
```

| TItem<br>\|<br>*TMatter*<br>\|<br>TStein | **Kurzbeschreibung** | Attribute | Methoden | Klassendiagramm |
|---|---|---|---|---|
| | Die Klasse bildet einen Stein ab. | | | |

```
UNIT uStein;
(* ******************************************************************** *)
(* K L A S S E : TStein                                                 *)
(* -------------------------------------------------------------------- *)
(* Version     : 1.2                                                    *)
(* Autor       : (c) 2004, Siegfried Spolwig                            *)
(* Beschreibung: Die Klasse bildet einen Ziegelstein ab                 *)
(*                                                                      *)
(* Compiler    : Delphi 6                                               *)
(* Aenderungen : 0.9   01-MAR-04                                        *)
(* ******************************************************************** *)
INTERFACE
// =========================================================================
uses
  uMatter;
type
  TStein = class(TMatter)
   private
   public
    constructor Create;       override;
    procedure   Init;
    procedure   MauerBauen;   virtual;
  end;

(* -------------------- B e s c h r e i b u n g ------------------------
Oberklasse    : TMatter
Bezugsklassen : -      import:
Methoden
--------
Create
  Auftrag: Exemlar erzeugen und init.
  vorher :
  nachher: done.

Init
  Auftrag: Anfangswerte setzen
  vorher :
  nachher: Bild ist gesetzt

Set...
  Auftrag: Attribut schreiben
  vorher :
  nachher: Attribut ist gesetzt

Get...
  Auftrag: Attribut aus dem Objekt lesen
  vorher :
  nachher: -


MauerBauen
```

```
  Auftrag: Setzt eine Reihe aus Steinen
  vorher :
  nachher: -
---------------------------------------------------------------------- *)
```

| TWelt<br>\|<br>TTrainingscamp | **Kurzbeschreibung** | Attribute | Methoden | Klassendiagramm |
|---|---|---|---|---|
| | Die Klasse enthält nur Bäume und ein gefährliches 'Schwarzes Loch'.<br>Sie ist gedacht für die ersten Tests und Aktionen mit dem Roboter | | | |

```
UNIT uTrainingscamp;
(* ******************************************************************** *)
(* K L A S S E : TTrainingscamp                                          *)
(* -------------------------------------------------------------------- *)
(* Version     : 2.2                                                     *)
(* Autor       : (c) 2004, Siegfried Spolwig                            *)
(* Beschreibung: Die Klasse bildet die Welt 'Trainingscamp' ab.         *)
(*                                                                       *)
(* Compiler    : Delphi 6                                                *)
(* Aenderungen : 0.9   28-MAR-04                                         *)
(*               2.4   18-APR-05   SchwarzesLoch eingeführt              *)
(* ******************************************************************** *)

INTERFACE
// ==========================================================================
uses
  uWelt,
  uBaum,
  uBlackhole;

type
  TTrainingscamp = class(TWelt)
   protected
    Eiche,
    Linde : TBaum;
    SchwarzesLoch : TBlackhole;
   public
    constructor Create;   override;
    procedure   Init;
  end;

(* -------------------- B e s c h r e i b u n g -------------------------

Oberklasse    : TWelt
Bezugsklassen : -       import:


Methoden
--------

Create
  Auftrag: Exemlar erzeugen und init.
           Alle Objekte der vorigen Welt entfernen
  vorher :
  nachher: done.

Init
  Auftrag: Anfangswerte setzen
  vorher :
  nachher: Bäume und Steine gesetzt
```

```
Set...
  Auftrag: Attribut schreiben
  vorher :
  nachher: Attribut ist gesetzt

Get...
  Auftrag: Attribut aus dem Objekt lesen
  vorher :
  nachher: -

---------------------------------------------------------------------- *)
```

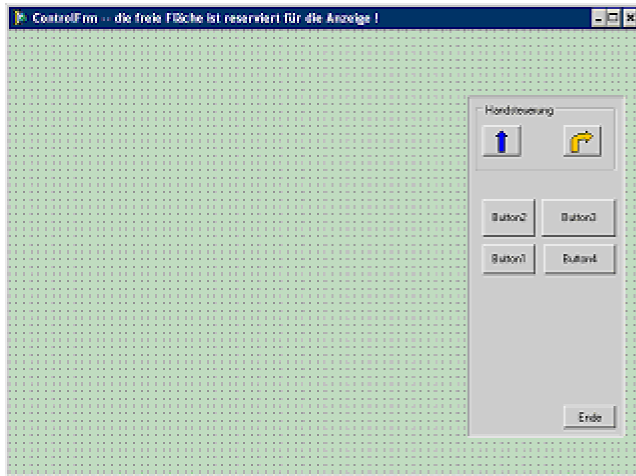
<table>
<tr><td rowspan="2">TWelt -- TItem</td><td>Kurzbeschreibung</td><td>Attribute</td><td>Methoden</td><td>Klassendiagramm</td></tr>
<tr><td colspan="4">Die Klasse ist die zentrale Registratur für alle Items in der Welt. Die Items melden sich an/ab und werden registriert.<br><br>TWelt ist die abstrakte Oberklasse für effektive Welten, die die Umgebung für die Objekte bilden.</td></tr>
</table>

```
UNIT uWelt;
(* ******************************************************************** *)
(* K L A S S E : TWelt                                                  *)
(* -------------------------------------------------------------------- *)
(* Version     : 2.2                                                    *)
(* Autor       : (c) 2004, Siegfried Spolwig                            *)
(* Beschreibung: Die Klasse ist die zentrale Registratur für alle Items.*)
(*               Die Items melden sich an/ab und werden dabei           *)
(*               registriert.                                           *)
(*               TWelt ist die quasi abstrakte Oberklasse für konkrete  *)
(*               Welten, die die Umgebung der Objekte bilden.           *)
(*                                                                      *)
(* Compiler    : Delphi 6.0                                             *)
(* Aenderungen : 0.9     01-MAR-04                                      *)
(*               2.2     20-JUL-04   s. Doc.                            *)
(*               2.2.1   03-JAN-05   SetLinienFarbe                     *)
(* ******************************************************************** *)

INTERFACE
// ==========================================================================
uses
  graphics,
  uGrafik,   // import TLinie
  uItem;

type
  TWelt = class(TObject)
   protected
    Rasterlinie : TLinie;
   private
    Feld        : array['A'..'N',1..12] of TItem;   // 14x12
    QLaenge     : integer; // Laenge eines Quadranten

   public
    constructor Create;  virtual;
    procedure Init;
    procedure ItemSetzen (It:TItem);
    procedure ItemAbmelden (It:TItem);
    procedure AlleItemsEntfernen;
    function  GetQLaenge : integer;
    function  GetItem(spalte: char; zeile:integer) : TItem;
```

```
    procedure SetFuellFarbe(f : TColor);
    procedure SetLinienFarbe(f: TColor);

    procedure Zeigen;      virtual;
    procedure AllesZeigen; virtual;
    procedure AlleLoeschen;

//    procedure Test;
  end;

(* ------------------- B e s c h r e i b u n g ------------------------

Oberklasse    : -
Bezugsklassen : TFensterFrm;  TItem, TMatter, TStein, TBaum

Methoden
--------

Create
  Auftrag: Exemplar erzeugen und init.
  vorher :
  nachher:

Init
  Auftrag: Anfangswerte setzen
  vorher : -
  nachher: Alle Felder auf leer (NIL) gesetzt. Rasterlinienfarbe ist Sil-
ver.

ItemSetzen
  Auftrag: Item meldet sich mit der aktuellen Position auf dem entsprechen-
den
           Feld an
  vorher : -
  nachher: angemeldet, unabhängig davon ob das Feld besetzt war


ItItemAbmelden
  Auftrag: Item meldet sich mit der aktuellen Position auf dem entsprechen-
den
           Feld ab.
  vorher : Item steht auf dem Feld
  nachher: Feld ist leer (NIL)

Set...
  Auftrag: Attribut schreiben
  vorher : Welt ist init.
  nachher: Attribut ist gesetzt

Get...
  Auftrag: Attribut aus dem Objekt lesen
  vorher : Welt ist init.
  nachher: -

AlleItemsEntfernen;
  Auftrag: Alle Items aus dem Speicher entfernen
  vorher : -
  nachher: done. Alle Felder NIL

GetQLaenge
  Anfrage: nach der Länge eines Feldes
  vorher :
  nachher: -

GetItem
```

```
  Anfrage: Item aus der Position holen
  vorher :
  nachher: Zeiger auf Item, das dort bleibt

Zeigen
  Auftrag: Aktuelle Welt auf dem Bildschirm anzeigen
  vorher :
  nachher: Hintergrund und Gitter

AllesZeigen
  Auftrag: Welt und alle angemeldeten Items zeigen
  vorher : -
  nachher: -

AlleLoeschen
  Auftrag: Alle angemeldeten Items vom Bildschirm loeschen
  vorher :
  nachher: Welt ist leer, Items unsichtbar

---------------------------------------------------------------------- *)
```

<table>
<tr><td rowspan="2">TFensterFrm<br>|<br>TControlFrm</td><td>Kurzbeschreibung</td><td>Attribute</td><td>Methoden</td><td>Klassendiagramm</td></tr>
<tr><td colspan="4">Die Klasse ist das Main-Formular, das der Benutzer sieht und in dem er Steuerungskomponenten und eigene Objekte in der procedure ItemsErzeugen implementieren kann.</td></tr>
</table>



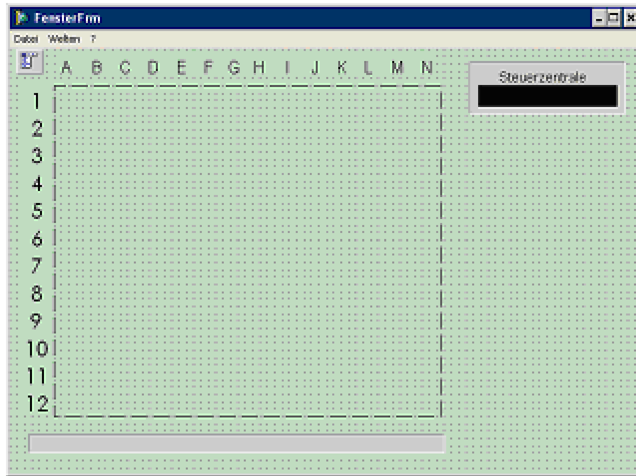

Zur Erläuterung:

Controlfrm ist das GUI-Formular, in dem zur Entwurfszeit die Eingaben des Benutzers zusammengefasst sind.

Zur Laufzeit werden alle Komponenten von Controlfrm und FensterFrm, in dem im Wesentlichen die Anzeigen (view) implementiert sind, in einem Formular dargestellt.

```
unit uControl;
(* ****************************************************************** *)
(* K L A S S E : TControlFrm  - Karel D. Robot                        *)
(* ------------------------------------------------------------------ *)
(* Version     : 2.1.3                                                *)
(* Autor       : (c) 2004, Siegfried Spolwig                          *)
(* Beschreibung: Die Klasse hat Control-Funktion für die Welt und     *)
(*               die Akteure.                                          *)
(* Compiler    : Delphi 6.0                                           *)
(* Aenderungen : 2.1.3   11-JUL-04 s. Doc.                            *)
(*                                                                     *)
(* known bugs  : bei schneller Klickfolge fehlen Fußspuren            *)
(* ****************************************************************** *)
interface
// ======================================================================
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
  Dialogs, ExtCtrls, StdCtrls, Buttons, uProtBox,
  uFenster,     // Oberklasse; zeigt die Welt
  uCritter,
  ukarel,       // Fachklassen nach Bedarf
  uRobot;
type
  TControlFrm = class(TFensterFrm)
    UserPnl    : TPanel;
    EndeBtn    : TBitBtn;
    GroupBox1  : TGroupBox;
    VorBtn     : TSpeedButton;

    procedure EndeBtnClick(Sender: TObject);
    procedure VorBtnClick(Sender: TObject);
    procedure ItemsErzeugen; // darf nicht gelöscht werden!
  end;

var
  ControlFrm : TControlFrm;
// ----------------------------------------------------------------------
```

| TFensterFrm | **Kurzbeschreibung** | Attribute | Methoden | Klassendiagramm |
|---|---|---|---|---|
| | Die Klasse realisiert die Anzeige der Welten und ist normalerweise nicht für den Benutzer zugänglich. | | | |




Zur Erläuterung:

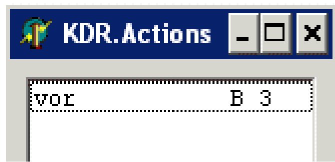
FensterFrm ist das Hauptformular, in dem zur Entwurfszeit im Wesentlichen die Anzeigen (view) implementiert sind.

Zur Laufzeit werden alle Komponenten von FensterFrm und von Controlfrm, in dem im Wesentlichen die Eingaben implementiert sind, in einem Formular zusammen dargestellt.

```
unit uFenster;
(* ******************************************************************** *)
(* K L A S S E : TFensterFrm  - Karel D. Robot                          *)
(* -------------------------------------------------------------------- *)
(* Version     : 2.2                                                    *)
(* Autor       : (c) 2004, Siegfried Spolwig                            *)
(* Beschreibung: Die Klasse hat Control/Viewfunktion für die Welt und   *)
(*               die Akteure.                                           *)
(* Compiler    : Delphi 6.0                                             *)
(* Aenderung   : 2.0    02-APR-04   User-ControlFrm ausgelagert         *)
(*                                  Der Benutzer sieht nur eigene Kompo-*)
(*                                  nenten in ControlFrm                *)
(*               2.2    20-JUL-04   s. Doc.                             *)
(*                                                                      *)
(* known bugs  : Fußspuren fehlen bei schnellen Clicks.                 *)
(* ******************************************************************** *)
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ExtCtrls, ShellApi, Menus, Buttons,
  uGrafik, uInfoBox, uProtBox,
  uWelt, uTrainingscamp, uCleanCity, uMyWorld,  uLabyrinth, uBaum, uHaus,
  uMuell, uStein;               // die Fachklassen nach Bedarf

type
  TFensterFrm = class(TForm)
    WeltImg        : TImage;
    SpaltenLbl     : TLabel;
    ZeilenLbl      : TLabel;
    MeldePnl       : TPanel;
    ZentralPnl     : TPanel;
    SteuerLbl      : TLabel;
    KontrollPnl    : TPanel;
    MainMenu1      : TMainMenu;
    Datei1         : TMenuItem;
    Welten1        : TMenuItem;
    About1         : TMenuItem;
    Training1      : TMenuItem;
    CleanCity1     : TMenuItem;
    Labyrinth1     : TMenuItem;
    MyWorld1       : TMenuItem;
    About          : TMenuItem;
```

```
    Hilfe1        : TMenuItem;
    Mauerbau1     : TMenuItem;
    Alleebaeume1  : TMenuItem;
    Protokoll1    : TMenuItem;
    ProtSpeichern1: TMenuItem;
    ProtLaden1    : TMenuItem;
    ProtLoeschen1 : TMenuItem;
    procedure FormCreate(Sender: TObject);
    procedure AboutClick(Sender: TObject);
    procedure Hilfe1Click(Sender: TObject);
    procedure Training1Click(Sender: TObject);
    procedure Labyrinth1Click(Sender: TObject);
    procedure CleanCity1Click(Sender: TObject);
    procedure MyWorld1Click(Sender: TObject);
    procedure MeldePnlClick(Sender: TObject);
    procedure Mauerbau1Click(Sender: TObject);

    procedure Alleebaeume1Click(Sender: TObject);
    procedure Protokoll1Click(Sender: TObject);
    procedure ProtSpeichern1Click(Sender: TObject);
    procedure ProtLaden1Click(Sender: TObject);
    procedure ProtLoeschen1Click(Sender: TObject);
    procedure Meldung(s : string);
   private
   public
  end;

var
  FensterFrm    : TFensterFrm;
  Zeichenblatt : TZeichenblatt;
  Welt          : TWelt;         // kind of Singleton pattern - global
  Trainingscamp: TTrainingscamp;
  CleanCity     : TCleanCity;
  Labyrinth     : TLabyrinth;
  MyWorld       : TMyWorld;

// ------------------------------------------------------------------------------------------------------------
```

<table>
<tr><td rowspan="2">TProtokollFrm</td><td>Kurzbeschreibung</td><td>Attribute</td><td>Methoden</td><td>Klassendiagramm</td></tr>
<tr><td colspan="4">Die Klasse ist das Formular, das das Gedaechtnis-Protokoll von Robots anzeigt.</td></tr>
</table>




Zur Erläuterung:

Protokollform ist ein Formular, das sich öffnet, wenn das Gedaechtnis-Protokoll von Robots angezeigt werden soll.

```
unit uProtBox;
(* ****************************************************************** *)
(* K L A S S E : TProtokollFrm  - Karel D. Robot                      *)
(* ------------------------------------------------------------------ *)
(* Version     :  2.2                                                 *)
(* Autor       : (c) 2004, Siegfried Spolwig                          *)
(* Beschreibung: Die Klasse öffnet ein Fenster zur Anzeige der Actions, *)
(*               die Karel nach dem Start ausgeführt hat.             *)
(* Compiler    : Delphi 6.0                                           *)
(* Aenderungen : s. Doc. / 25-JUL-04  TMemo statt TListBox            *)
(*                                                                    *)
(* known bugs  : -                                                    *)
(* ****************************************************************** *)
interface
// ==========================================================================
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
  Dialogs, StdCtrls,
  uRobot;
type
  TProtokollFrm = class(TForm)
    Memo1 : TMemo;

    function  IstAktiv : boolean;
    procedure Aktualisieren(rob : TRobot);
    procedure Speichern(Sender: TObject);
    procedure Laden(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  end;
var
  ProtokollFrm: TProtokollFrm;

(*  ----------------- B e s c h r e i b u n g -------------------------
Oberklasse     : -
Bezugsklassen  : TRobot
Methoden
--------
IstAktiv
  Anfrage: ob das Aktualisieren des Protokoll aufgerufen wurde
  vorher : -
  nachher: true, wenn Aktualisieren gestartet ist

Aktualisieren(rob : TRobot);
  Auftrag: Alte Einträge löschen, das Gedaechtnis von rob an das Protokoll
           übergeben
  vorher : -
  nachher: done

Laden
  Auftrag: Altes Protokoll aus externer Datei <KDRoute.dat> laden.
  vorher : Die Liste ist initialisiert.
```

```
  nachher: Das Protokoll ist in die Listbox geladen.

Speichern
  Auftrag: Aktuell gezeigtes Protokoll in externe Datei <KDRoute.dat>
           speichern.
  vorher : -
  nachher: Die Liste ist gespeichert. Ist die Liste leer, wird eine Datei
           mit der Länge 0 angelegt. Datei ist geschlossen.

FormClose
  Auftrag: Aktuelles Protokoll bei Programmende speichern
  vorher : -
  nachher: done.
----------------------------------------------------------------------- *)
```

# 4. Architektur und Technisches

**Objektorientierung** - es gibt NUR Objekte

Ein System, das vorgibt, in objektorientierte Programmierung einzuführen, muß selbst objektorientiert aufgebaut sein. Diese Forderung ist eingehalten. Alles, was in der Delphi-Karel-Welt ist, sind Objekte und als solche ansprechbar. Deshalb wurde auch z. B. die Spur, die der Roboter ziehen kann, als Objekt angelegt und kann wie jedes Objekt auch wieder entfernt werden. Ebenso werden in den Container des Robots die aufgesammelten Objekte getan und können nach dem LIFO-Prinzip (stack) wieder heraus genommen werden.

**Architektur / Design**

Die Klassenhierarchie ist klassisch angelegt mit relativ tief gestaffelten Vererbungsketten. Wo die Situationen es erfordern, sind polymorphe Zugriffe vorhanden (Darstellung, Container).

Die Struktur der Klassen in Karel D. Robot ergibt sich aus der Sachlogik des Anwendungsproblems, nämlich dem Modell einer Welt mit Sachen und Lebewesen, wie im OOA-Modell dargestellt. Üblicherweise werden die so gefundenen Klassen dann ‚Fachklassen' genannt, womit implizit unterstellt wird, daß sie kein Wissen über die Art und Weise ihrer Darstellung auf dem Bildschirm (View) haben und nicht an der Steuerung (Controller) durch die Entgegennahme von Events beteiligt sind.

Das Konzept der Trennung dieser drei Zuständigkeiten wird Model-View-Controller Pattern genannt und ist in Verbindung mit dem 4-Schichten-Modell das derzeit übliche Verfahren, um große Softwaresysteme zu strukturieren und entwerfen. DELPHI-Karel folgt diesem Prinzip teilweise nicht. Die Idee war, verhaltensvollständige Objekte zu entwerfen, die selbst in der Lage sein sollten, sich dem Benutzer auf dem Bildschirm zu präsentieren. Das ist die alte Smalltalk-Idee, die meines Erachtens bei Anwendungen mit grafischen Objekten immer noch ihre Berechtigung hat und dabei durchaus Vorteile bietet. Insofern ist der Begriff der Fachklassen hier erweitert, als daß jedes Objekt `per definitionem` (`TItem.Zeigen`) seinen spezifischen View mitbringt (und daher auf das GUI zugreifen kann) und manipulierbar ist.

Die grafische Grundlage für DELPHI-Karel ist *uGrafik*, eine Sammlung von Klassen, die eben diesem Prinzip folgen. Daher ist die Implementation des Gesamtsystems vergleichsweise einfach und elegant zu lösen.

**Zweiseitige Assoziationen**

Wenn Objekte wie hier sich selbst darstellen sollen, muß ein Zugriff auf Systemkomponenten (Canvas usw.) organisiert werden, was in Fachklassen eigentlich nichts zu suchen hat. Das wird hier mit Hilfe der Grafik-Bibliothek uGrafik realisiert.

Die Welt registriert als übergeordnete Instanz die Anwesenheit aller Objekte, gleichzeitig benötigen die Objekte Informationen von der Welt. Da Delphi keine Zirkelbezüge in der Schnittstelle zuläßt, werden die erforderlichen Assoziationen im Implementation-Teil aufgebaut. In den Schnittstellen werden nur die Klassen importiert, die auch exportiert werden müssen.

**Mehrere Roboter / Critters**

Selbstverständlich können sich mehrere Exemplare 'gleichzeitig' in der Welt bewegen. In dieser Version ist jedoch keine echte Nebenläufigkeit eingebaut. Das hat zur Folge, daß ein Objekt seine Bewegung abgeschlossen haben muß, bevor das nächste startet. Da ein Weg 30 pix beträgt, ist das natürlich deutlich sichtbar. Ein Effekt, mit dem man aber das Problem der Nebenläufigkeit schön sichtbar machen kann.

### Bildschirmauflösung - Darstellung

Unter Windows 2000 kommt es manchmal vor, daß die Zeichen 1..12 , A..N (Zeilen/Spalten) in einem zu kleinen Font dargestellt werden. --> F.A.Q

### Version-Historie

| | | |
|---|---|---|
| **2.4** | 17-APR-05 | Neu : TLagerhaus eingeführt, TKarel.Einlagern, TKarel.Auslagern.<br>Neu : TBlackhole als 'Schwarzes Loch' oder als rückstandsfreier Müllschlucker verschlingt jedes Objekt.<br>TStack.SollLaenge erlaubt jetzt unterschiedliche definierte Listenlaengen.<br>Bug in TRobot.Zeigen beseitigt, Setpos war nicht virtual. |
| 2.3.3**.** | 14-MAR-05 | TRichtung = (Nord, Ost, Sued, West). Das Zeichen A als Aufzählungstyp oder Char irritierte im Anfangsunterricht. |
| **2.3.2** | 14-FEB-05 | Welt.SetLinienFarbe, TKarel.VorDemAbgrund, redaktionelle Änderungen |
| **2.3.1** | 07-JAN-05 | Schönheitsfehler beseitigt. Demo.exe hinzugefügt. |
| 2.3 | 02-JAN-05 | Tutorial --> Menü-? |
| 2.2.1 | 30-Nov-04 | TRobot kann Spur machen.<br>Bugs in TItem beseitigt; Bild wurde nicht gelöscht. |
| **2.2** | 25-JUL-04 | Einfacher TeachIn-Mode mit Copy/Paste aus dem Protokoll<br>Interne Sicherheitsstandards erhöht |
| 2.1.3 | 20-JUL-04 | Robot speichert das Gedächtnis nicht selbst; sondern Laden und Speichern erfolgt direkt aus der Listbox --> Menü-Datei<br>Aktualisierung der Anzeige in der Listbox erfolgt durch aktiven Aufruf durch den Roboter.<br>Geschwindigkeit verbessert. |
| 2.1 | 24-JUN-04 | Robots haben ein Gedächtnis (TMemory, TShadow), wo sie sich die Actions und Felder merken, die sie seit dem Start besucht hatten (DaGewesen). Die Informationen werden als Protokoll geführt --> Menü-Datei. |
| **2.0.7** | 14-JUN-04 | Geschwindigkeit in TCritter, TRobot |
| 2.0.6 | 26-APR-04<br>08-MAY-04 | Kleine textuelle Änderungen<br>Für Delphi 4 angepaßt (Christian Steinbrucker). |
| 2.0.5 | 18-APR-04 | THaufen in TMuell umbenannt, bug in VorneFrei beseitigt |
| 2.0.4 | 15-APR-04 | TLabyrinth |
| **2.0.3** | 11-APR-04 | Roboter kann Aufnehmen, ablegen<br>Fußspuren - TSpur |
| 2.0.2 | 05-APR-04 | Programm merkt sich bei Recompilieren die gewählt Welt.<br>Automatisches Erzeugen des Roboters entfernt. Jetzt in uControl.pas. |
| 2.0.1 | 02-APR-04 | User-ControlForm ausgelagert. Der Benutzer sieht nur eigene Komponenten in ControlFrm |
| **1.3** | 28-MAR-04 | TWelt abstrakt; Unterklassen |
| 1.2 | 27-MAR-04 | Clean City, My World |
| 1.1.3 | 26-MAR-04 | Welten erzeugen in TWelt |
| 1.1.2. | 24-MAR-04 | Trainingscamp |
| 1.0 | 22-MAR-04 | Docs,<br>Robot: VorneFrei, HindernisPruefen, Rechtsdrehen statt Links |
| 0.9.3 | 18-MAR-04 | Bilder für Items |
| 0.9 | 03-MAR-04 | '*Kap der guten Hoffnung*' - TItem |

## 4.1 Installation - V. 2.4

**1. Delphi-Versionen -**

Wenn Sie Delphi 6 oder 7 benutzen, installieren Sie bitte **Delphi-Karel.zip.**

Wenn Sie Delphi 4 oder 5 benutzen, installieren Sie bitte **Delphi4-Karel.zip.**

Bei Delphi 5 erscheint beim ersten Start die Meldung

Klicken Sie OK!

**2.** Die Datei **Delphi-Karel.zip** enthält alle erforderlichen Files.

- Legen Sie einen neuen Ordner an in dem Verzeichnis, wo Sie Ihre Delphi-Projekte ablegen.
- Extrahierend Sie **Delphi-Karel.zip** in dieses Verzeichnis.

**Erste Schritte, wenn Sie gleich loslegen wollen ...**

Beim ersten Start erscheint die Welt 'Trainingscamp'. Mit nur drei Zeilen erzeugen Sie den Roboter RD1 vom Typ TRobot und können ihn mit dem eingebauten Button starten.
s. ==> Tutorial.

**3. Ordner / Dateien:**

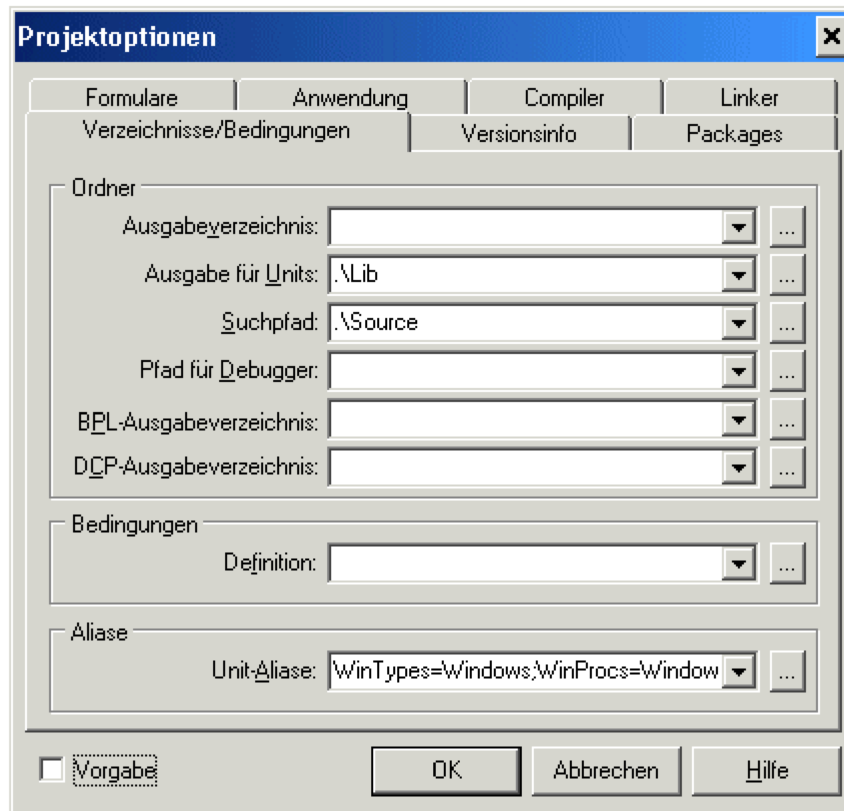
Delphi-Karel.zip enthält folgende Ordner / Dateien:

| | |
|---|---|
| \Backup | für eigene Sicherungen |
| \Bilder | .bmp-Bilder für das Projekt |
| \Docs | Dokumentationen |
| \Docs\delphi_karel\ | HTML-Seiten für Hilfe, Spezifikationen, Beschreibungen |
| \Docs\tutorial\ | Einführung in OOP |
| \Forms | Dateien von uFenster, uInfoBox , uProtBox |
| \Lib | .dcu - Dateien vom Programm, uGrafik, uZeit, uDListe |
| \Source | Quellcode von uGrafik, uZeit, uDListe und alle verborgenen Fachklassen |
| \ <br><br>DelphiKarel.dof<br>DelphiKarel.dpr<br>DelphiKarel.res<br><br>install.htm<br>KdR.ini | <br><br><br><br>Ressourcen<br><br>Diese Anleitung<br>Ini-File für die jeweils bearbeitete Welt |

| | |
|---|---|
| KdRoute.dat<br><br>Demo.exe<br><br>uControl.ddp<br>uControl.dfm<br>uControl.pas | gespeichertes Gedächtnis-Protokoll von Robot<br><br>zeigt einige Features der Roboter RD1 und Karel<br><br><br>Control-Formular |
| uMyRobot.pas | leere, vorbereitete Roboterklasse |

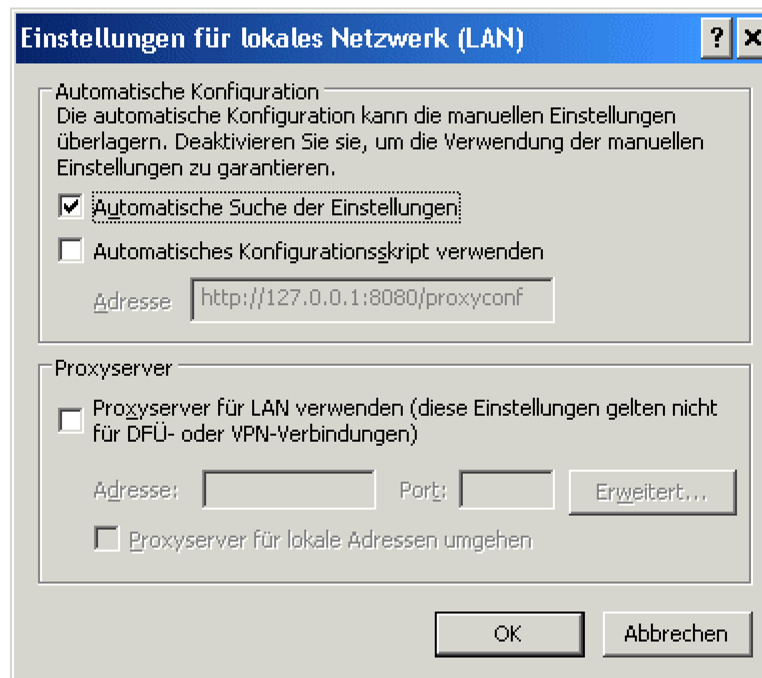
1. **Einstellungen / Pfade bei Delphi**

   Die Datei DelphiKarel.dof enthält die Pfade für das Projekt. Wenn Ihr Delphi-System beim ersten Start Probleme mit Pfaden meldet, löschen Sie diese Datei und tragen Sie von Hand bei Projekt-Optionen ein:



## 5. Einstellungen / Pfade beim Internet Explorer

Wenn die Hilfeseite nicht angezeigt wird, prüfen Sie bitte die Einstellungen des Internet Explorers.
Bei Extras-Internetoptionen-Verbindungen-LAN sollte 'Automatische Suche' aktiviert sein.



## 4.2 Programmiertips

**Wie kann ich ...**

1) meine **Übungen** strukturieren und speichern?
2) ein **neues Objekt** in das Programm einfügen?
3) die **Methoden verändern** von **TRobot** (oder anderen Klassen)
4) ein **Bild** als Attribut eines Objekts einfügen?
5) dem Roboter beibringen, ein **Objekt zu identifizieren**, damit er es z. B. aufnehmen kann?
6) eine Aktion in Karels **Gedächtnis** aufnehmen?
7) das **Protokoll** der Roboter-Actions aktivieren?
8) einen Roboter zu einem entfernten Feld **springen** lassen?
9) einen Roboter **schneller machen**?
10) eine **Meldung** ausgeben?

**Antworten**

**1. meine Übungen strukturieren und speichern?**

Zur Strukturierung

Vor 30 Jahren war man in der Schule stolz, wenn als 1. Programm auf einem ansonsten schwarzen Bildschirm in grüner Schrift erschien: HALLO WORLD

Das ganze Programm dazu so aus:

Genauso winzig war die Datei, die das Programm speicherte.

```
program hallo;
begin
  writeln('HALLO WORLD');
end.
```

Sie haben den Vorzug mit einem professionellen Programm zu arbeiten, das entsprechend umfangreich ist. Dabei operieren Sie zunächst immer nur an einem ziemlich kleinen Teil eines lebenden Organismus. Man muß die Übersicht behalten und kann nicht für jede kleine Änderung jedesmal das gesamte Projekt mit seinen rund 2 MB abspeichern. Läßt man alle Änderungen im ControlFrm stehen, wird das schnell lang und unübersichtlich.

Anfangs werden Sie wahrscheinlich im ControlFrm arbeiten. Nehmen wir an, die Aufgabe lautet RD1 soll dreimal im Kreis herumlaufen.

...

RD1 soll dreimal im Kreis herumlaufen.

**A)**

In vielen Delphi-Anfängerbüchern würden Sie einen Vorschlag wie diesen finden.

Oder noch vielleicht viel längere Prozeduren bei anderen Problemen.

```
procedure TControlFrm.Button1Click(Sender: TObject);
var i : integer;
begin
  for i := 1 to 12 do
  begin
   RD1.Vor;
   RD1.RechtsDrehen
  end;
end;
```

**So nicht !**

Ein Lichtschalter an der Wand produziert weder die Elektrizität noch erhellt er den Raum. Er schaltet nur den Strom ein. Genau das soll ein Button tun.

**B)** Die bessere Lösung ist aufgeteilt:

in die **Ereignismethode** (der Button-Click)

```
procedure TControlFrm.Button1Click(Sender: TOb-
ject);
begin
  ImKreisHerum;
end;
```

und eine **private Methode**, die das Problem bearbeitet und bei Bedarf aufgerufen wird

```
procedure TControlFrm.ImKreisHerum;
var i : integer;
begin
  for i := 1 to 12 do
  begin
    RD1.Vor;
    RD1.RechtsDrehen
  end;
end;
```

Das ist guter Programmierstil. Es sieht zunächst umständlich aus, schafft aber klar strukturierte Programme, in denen das an der Stelle steht, wo es hingehört (Lokalität). Sie brauchen auch nur *einen* Button, um verschiedene Methoden aufzurufen und zu testen.

**C)** Genau genommen haben Sie eine neue Methode für Roboter entwickelt, die in die Roboterklasse aufgenommen werden kann und dann für jeden Roboter zur Verfügung steht.

Zum Speichern von Übungen im ControlFrm

- Die pragmatische Lösung: Statt das gesamte Projekt zu speichern, können Sie einzelne Prozeduren / Lösungen aus dem ControlFrm ausschneiden und in eine separate Textdatei zur Speicherung kopieren und bei Bedarf später wieder zurückkopieren. Das funktioniert aber *nicht mit der Lösung A,* sondern nur mit B (ohne die ButtonClick-Methode).

- Die professionelle Lösung: Wenn es eine gute Methode wert ist, als neue allgemeine Roboter-Methode implementiert zu werden, gehen wie in C vor und speichern die Unit mit der Roboterklasse.

### 2. ein neues Objekt in das Programm einfügen?

```
var Auto : TCritter;    // in ControlFrm bei var deklarieren
Auto := TAuto.Create;   // Objekt erzeugen
                           z.B. Auto in procedure ItemsErzeugen
Auto.SetPos ('G',2);    // und an die gewünschte Stelle setzen
```

### 3. die Methoden von TRobot (oder anderen Klassen) verändern?

Im Prinzip in der Unit uRobot. Davon wird aber dringend abgeraten, wenn Sie nicht sehr genau wissen, was Sie tun, weil jede Änderung sich in den Unterklassen ebenfalls auswirkt.

Es ist besser, eine neue Unterklasse TMyRobot (oder so ähnlich genannt) zu erstellen und dort die geerbte Methode zu überschreiben und dann mit TMyKarel weiterarbeiten.

Beispiel:

```
Unit uMyRobot;
INTERFACE
uses uRobot;

type
  TMyRobot = class (TRobot)   // erbt alles

  procedure Vor; override;    // hier überschreiben mit der eigenen
Methode
  ...
```

### 4. ein Bild als Attribut einfügen?

Zweckmäßigerweise sollten alle Exemplare der betreffenden Klasse dasselbe Bild tragen. Dann fügen Sie in die Initialisierungsprozedur die Zeile ein:

```
procedure Txyz.Init;
begin
  ...
  Blume.SetBild ('.\bilder\blume.bmp');
end;
```

Zuvor müssen Sie das Bild blume.bmp in das Verzeichnis \bilder kopieren. Format : 28 x 28 pixel.

### 5. dem Roboter beibringen, ein Objekt zu identifizieren, damit er es z. B. aufnehmen kann?

Man kann Objekte fragen, zu welchem Typ sie gehören. Wenn Karel z. B. vor einem Baum steht, liefert seine Methode 'HindernisPruefen : TItem' die entsprechende Klasse.

```
if (RD1.HindernisPruefen is TBaum)
then ...
```

### 6. eine Aktion in Karels Gedächtnis aufnehmen?

Derzeitig werden die Methoden 'Vor' und 'RechtsDrehen' ins Gedächtnis gemerkt.
Fügen Sie die Anweisung 'InsGedaechtnis ('...'); in die gewünschte Prozedur der Roboterklasse ein.
Im Parameter steht der string, der die Aktion bezeichnet.

Beispiel:

```
procedure TMyKarel.Rueckwaerts;
begin
   ...
   InsGedaechtnis('Rueckwaerts');
end;
```

### 7. das Protokoll der Roboter-Actions aktivieren?

Da bei der Entwicklung dieses Programms noch nicht bekannt ist, wie Ihr Roboter später vielleicht heißen wird, aktivieren Sie das Protokoll selbst. Das Protokollformular bekommt mitgeteilt, welcher Roboter die Daten zur Anzeige liefert, z. B. RD1:

ProtokollFrm.Aktualisieren(RD1);

Fügen Sie im ControlFrm diese Anweisung in die Methode ein, mit der Sie den Roboter starten.

Beispiel:

```
procedure TControlFrm.GoBtnClick(Sender: TObject);
begin
  inherited;
  while NOT RD1.BatterieLeer do
    begin
     if RD1.VorneFrei
     then RD1.Vor
     ....
     ...
     ProtokollFrm.Aktualisieren(RD1);  // jede Aktion wird ange-
zeigt
  end;
end;
```

Sie können die Anweisung auch in mehreren Prozeduren implementieren; Außer in TControlFrm.ItemsErzeugen (Absturz!)

### 8. einen Roboter zu einem entfernten Feld springen lassen?

**Am besten gar nicht!**

Angenommen beim Programmstart heißt es 'RD1.SetPos('B',1)'. Diese Anweisung registriert ihn mit dieser Position in der Welt. Mit einer Anweisung 'RD1.Vor' würde er sich sauber abmelden, mit einem Sprung z. B. nach ('D',5) aber nicht: damit ist er dann zweimal in der Welt registriert. Im unmittelbaren Programmablauf scheint das zunächst nicht zu stören, hat aber Probleme. B,1 ist besetzt, obwohl nichts zu sehen ist und spätestens beim Programmende erscheint eine Fehlermeldung, weil die Welt nicht sauber geschlossen werden kann.

Wenn Sie dennoch den Roboter 'fliegen' lassen wollen, sorgen Sie für eine korrekte Abmeldung mit der Anweisung

```
Begin
  ...
  Welt.Abmelden(RD1);
  RD1.SetPos(...);  // neue Position
  ...
```

### 9. einen Roboter schneller machen?

Die Darstellungsgeschwindigkeit hängt einerseits von uGrafik und dem darin implementierten Modell und zweitens von der Prozessorgeschwindigkeit ab.
Setzen Sie zunächst die Geschwindigkeit des Objekts herauf mit z. B. *RD1.SetGeschwindigkeit(8);* (max. 10). Wenn das nicht hilft und Sie einen langsamen Rechner haben, hilft nur noch eine Anpassung in uZeit.pas bei der

```
procedure TZeit.SetPause(millisec : longint);
* ------------------------------------------------ *)
 var i : LongInt;
 begin
   for i := 1 to 22400 * millisec do // ca. 1 mSek Bratzeit
   Application.ProcessMessages;
 end;
```

Die Schleife verheizt einfach Prozessorzeit: setzen Sie den Wert herunter bis die Pause (die in allen Critter-Bewegungen steckt) kürzer wird. Dadurch werden die Bewegungen schneller.

**10. eine Meldung ausgeben?**

Beispiel:

```
procedure TControlFrm.GoBtnClick(Sender: TObject);
// ------------------------------------------------
begin
  inherited;
  RD1.AllesEinsammeln;
  Meldung(' Fertig');
end;
```

## 4.3 F.A.Q – Frequently asked questions

| | |
|---|---|
| Units / Dateien | - Wie kann ich eine andere Unit hinzufügen (z.B. ein anderes uTrainings-camp.pas)?<br>- Wie kann ich eine Unit sichern (Backup) ? |
| Karels Tempo | - Wie kann ich die Robots schneller machen? |
| ControlFrm<br>ProtBoxFrm | - Warum darf die *procedure ItemsErzeugen* nicht gelöscht werden?<br>- Was zeigt die Memobox KDR.Actions? |
| Fehlermeldungen / Abstürze | - Warum stürzt das Programm manchmal ab, wenn ich auf einen Button klicke?<br>- Wie kann ich ein abgestürztes Programm wieder in Gang setzen?<br>- Was bedeutet Quelldatei nicht gefunden?<br>- ERangeError<br>- EExternalException |
| Bildschirm | - Wie kann ich die Zeilen-/Spaltenfonts besser anpassen? |

**Units / Dateien**

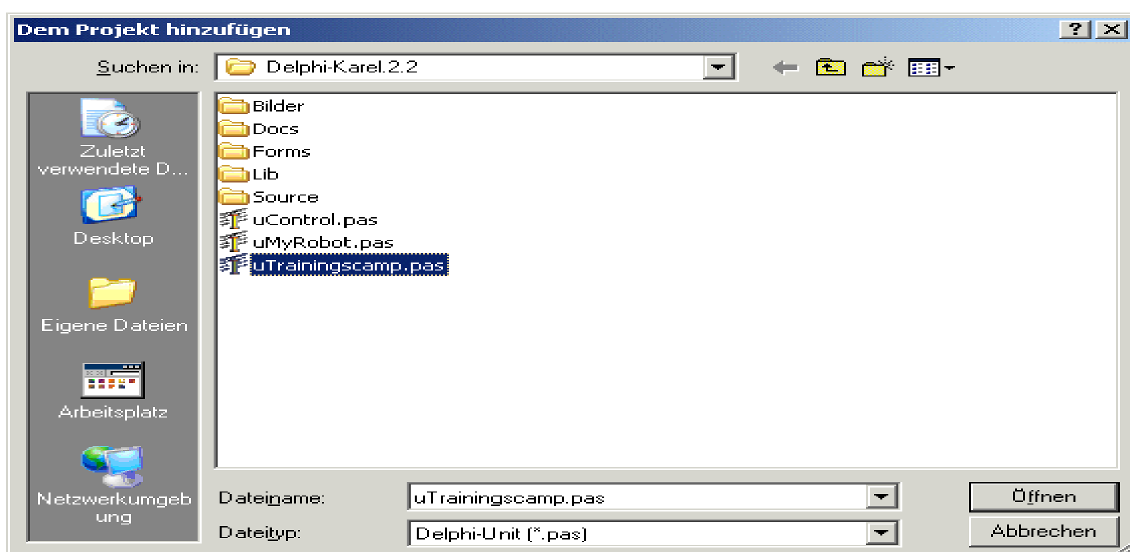
- **Wie kann ich eine andere Unit (z.B. uTrainigscamp.pas) hinzufügen?**

(Die hier beschriebenen Methoden gelten nur für Fachklassendateien! Delphi-Formulare lassen sich so nicht hin- und herkopieren, weil das System dann nicht mehr korrekt arbeitet.)

Ihr Lehrer hat Ihnen eine andere Welt mit einer neuen Aufgabenstellung zugeschickt. Die Datei sollte den Namen einer der vorgegebenen Welt haben, weil sie dann aus dem Karel-Menü aufgerufen werden kann.

Lösung:

| Andere Unit hinzufügen | |
|---|---|
| 1. Unit einkopieren | Unter Windows die Datei uTrainigscamp.pas in das Hauptverzeichnis von DelphiKarel kopieren. |
| 2. Delphi starten | ***Datei-Projekt öffnen*** ... DelphiKarel.dpr |
| 3. Neue Unit registrieren | ***Projekt-Dem Projekt hinzufügen ...*** uTrainigs-camp.pas |
| 4. Projekt neu compilieren, | ***Projekt-DelphiKarel compilieren***<br>(weil Delphi sonst intern die alte Version verwendet) |



- **Wie kann ich eine Unit sichern?**

  Sie haben z. B. einen eigenen Roboter entwickelt (z. B. uMyRobot.pas, der im Hauptverzeichnis steht) und möchten für die aktuelle Version ein Backup anlegen. Je nach Voreinstellung legt Delphi zwar ein temporäres Backup für die Projektdateien an, was aber bei jeder Änderung sofort überschrieben wird. Deshalb sollten Sie nach jeder größeren Änderung ein eigenes Backup mit einer Versionsnummer speichern.

  Lösung:

| **Unit sichern** (aus Delphi) | |
|---|---|
| 1. Unit öffnen | ***Datei-Öffnen ...*** uMyRobot.pas |
| 2. Unit speichern in Backup-Ordner | ***Datei-Speichern unter ...*** Backup\uMyRobot_2.pas<br>(Namen ändern mit Versionsnummer)<br>Delphi registriert sofort die neue Datei mit dem neuen Namen im Projekt. Deshalb: |
| 3. Backup aus der Registrierung nehmen | **Projekt-Aus dem Projekt entfernen ...'** uMyRobot_2.pas |
| 4. Original-Unit aus dem Hauptverzeichnis wieder einbinden | **Projekt-Dem Projekt hinzufügen ...'** uMyRobot.pas |

  Ein bißchen umständlich zwar, aber sicherer als aus Windows mit kopieren, umbenennen usw.

## Karels Tempo

- **Wie kann ich die Robots schneller machen?**

  Die Darstellungsgeschwindigkeit hängt einerseits von uGrafik und dem darin implementierten Modell und zweitens von der Prozessorgeschwindigkeit ab.
  Setzen Sie zunächst die Geschwindigkeit des Objekts herauf mit z. B.
  *RD1.SetGeschwindigkeit(8);*
  (max. 10). Wenn das nicht reicht und Sie einen langsamen Rechner haben, hilft nur noch eine Anpassung im Modul uZeit.pas ==> Programmiertips

## ControlFrm

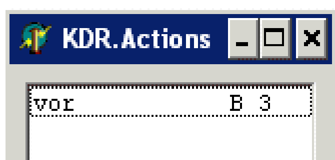
- **Warum darf die *procedure ItemsErzeugen* nicht gelöscht werden**?

  Für das Anzeigen aller Items und der kompletten Welt ist die Unit uFensterFrm verantwortlich, die sie normalerweise nie sehen. Diese Unit braucht die **procedure ItemsErzeugen** für die Anzeige der Welt und von neuen Objekten, die noch nicht da waren.

## ProtBoxFrm

- **Was zeigt die Memobox KDR.Actions?**

  Der *erste Eintrag* erfolgt, wenn Karel die erste Aktion (Vor oder Rechtsdrehen) ausgeführt hat.



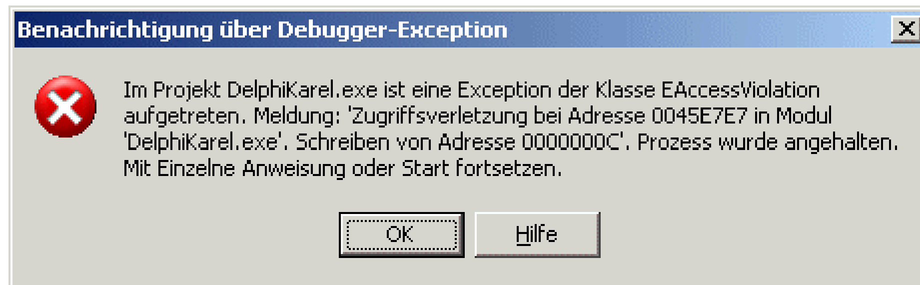

Action war 'Vor' --> danach steht Karel auf B 3

Die Startposition wird nicht aufgezeichnet.

## Fehlermeldungen / Abstürze

- **Warum stürzt das Programm manchmal ab, wenn ich auf einen Button klicke?**

**a)** Der Standard-Anfänger-Fehler.




Sie haben wahrscheinlich im Button ein neues Objekt angesprochen, was Sie noch nicht mit 'Create' erzeugt haben.

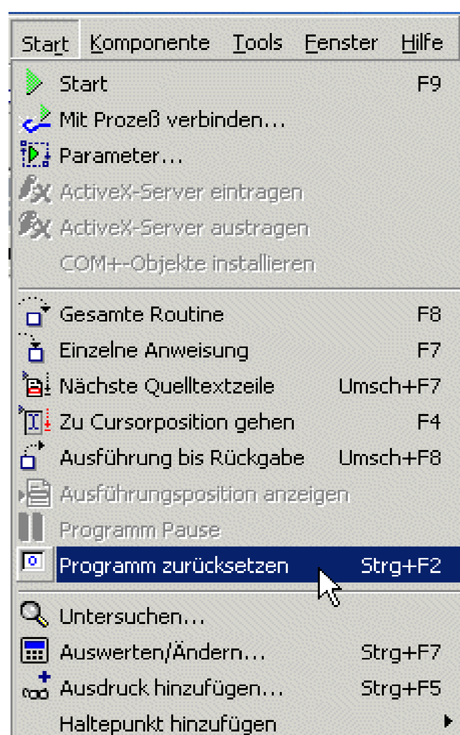
**b)** Das kommt auch dann vor, wenn Sie einen Roboter aus der Welt herauslaufen lassen. Es ist dann aus dem Speicher entfernt und jeder Versuch, ein nicht existierendes Exemplar anzusprechen, führt den Compiler zum Stillstand, weil er nicht weiß, was er in dieser Situation zu tun hat.

(Es ist pure Bosheit, daß diese Exception nicht abgefangen wird, aber nur um Sie immer wieder daran zu erinnern, daß ein Objekt zuerst mit Create erzeugt werden muß, bevor man es ansprechen kann.)

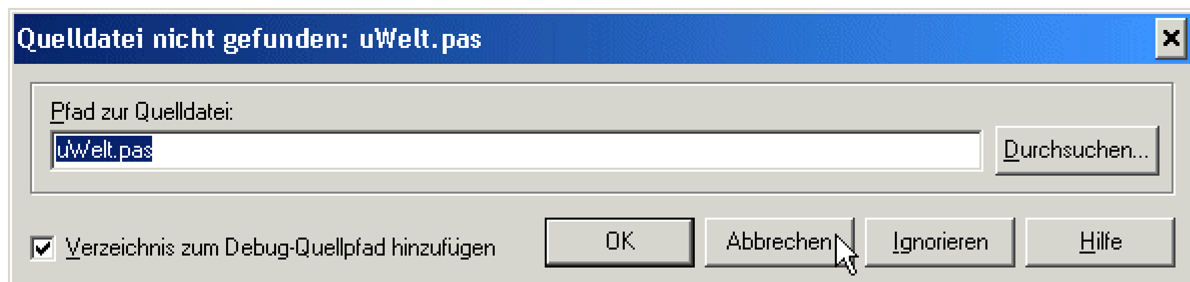
**c)** Sie haben einem Item eine Position gegeben, die OFFLIMITS ist; d. h. außerhalb von A..N oder 1..12. liegt. --> s. ERangeError

- **Wie kann ich ein abgestürztes Programm wieder in Gang setzen?**

**Lösung:**
Klicken Sie zunächst die Fehlermeldung mit OK weg und dann im DELPHI-Menü-Start-Programm zurücksetzen



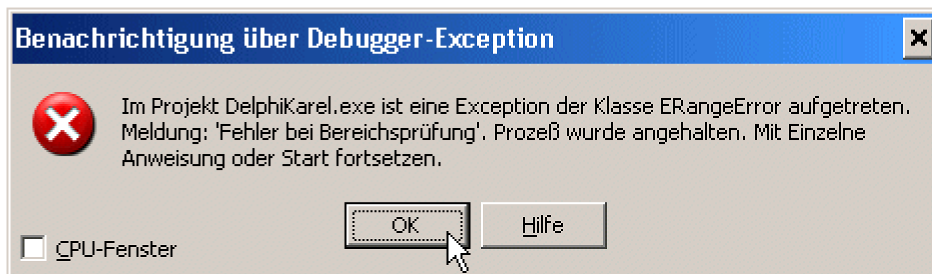
- **Was bedeutet diese Fehlermeldung?**




Völlig harmlos! - Der Compiler ist in eine Situation gekommen, die er aufgrund einer unerlaubten Anweisung nicht lösen kann. Deshalb sucht er nach Code in den Units, die im lib-Verzeichnis stehen. Der Fehler liegt aber dort nicht.

**Lösung**: Abbrechen klicken.

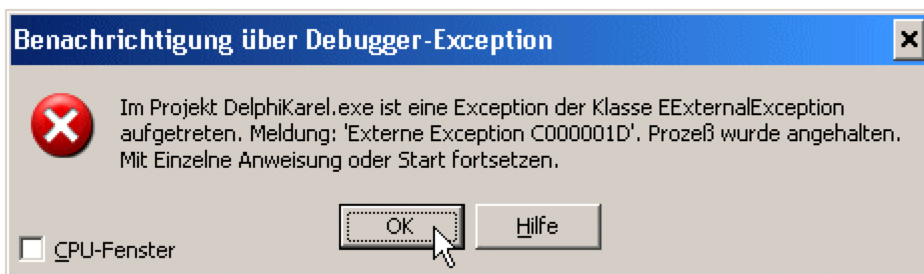
- **Was bedeutet diese Fehlermeldung? - ERangeError**




Völlig harmlos! - Sie haben einem Item eine Position gegeben, die OFFLIMITS ist; d. h. außerhalb von A..N oder 1..12. liegt.

**Lösung**: OK klicken und die Bewegung des Roboters oder Lage des (neuen) Items prüfen.

- **Was bedeutet diese Fehlermeldung? - EExternalException**




Völlig harmlos! - Sie haben wahrscheinlich nur den Roboter aus der Welt bewegt (OffLimits) s. oben. Das Objekt existiert nicht (mehr) im Speicher! Ein typischer Anfängerfehler.

**Lösung**: OK klicken und den Weg des Roboters prüfen; bzw. nachsehen, ob das Item mit Create erzeugt wurde.

- **Wie kann ich die Zeilen-/Spaltenfonts besser anpassen?**

  FensterFrm öffnen
  ZeilenLbl aktivieren
  Im Objektinspektor-Font anklicken
  Anderen Font ausprobieren

  das Gleiche für SpaltenLbl

**Dateien** des Karel D. Robot Projekts ==> install.htm

**Installation ==>** install.htm