Siegfried Spolwig

Karel D. Robot

- Der Delphi-Karel 2.3.2 -
Tutorial

Ein spielerischer Weg, um OOP mit Delphi zu lernen

Inhaltsverzeichnis

1. Delphi - die ersten Schritte	
1.0 Was Sie mindestens über Delphi wissen müssen	3
1.1 Projekt öffnen	5
1.2 Roboter einbauen und Button zur Steuerung aktivieren	6
1.3 Buttons in das Formular einfügen	7
1.4 Buttons and more	8
1.5 Interne Methode im Formular schreiben und mit Button aufrufen	9
2. Dem Roboter korrektes Benehmen beibringen (Kontrollstrukturen)	
2.1 Sequenzen	10
2.2 Auswahl, einseitig - Bedingungen	12
2.3 Auswahl, zweiseitig	13
2.4 Wiederholungen – Iteration	14
2.5 Rekursion	16
3. Einen besseren Roboter bauen (Klassenbeziehungen I)	
3.1 Wie das alles zusammenhängt	17
3.2 Vererbung	18
4. Schöne neue Welt (Klassenbeziehungen II)	
4.1 Assoziation	21
4.2 Aggregation	22
5. Anhang	
5.1 Was ist denn nun der Unterschied zwischen ...	25
5.2 Lösungen	26

Dieses Tutorial soll die ersten Schritte in die Welt der objektorientierten Programmierung unterstützen und mehr als Anregung für eigene Experimente und Fragestellungen dienen denn Vollständigkeit bieten. Deshalb ist es auch nicht als Aufgabensammlung angelegt und auch nicht als weitgehende Einführung in die Programmiersprache. Zur Vertiefung stehen am rechten Rand Links auf weiterführenden Stoff.

Die einzelnen Übungen stellen einfache Beispiele zur Lösung typischer Probleme dar. Sie sind auf die Welt 'Trainigscamp' und einen Roboter vom Typ TRobot zugeschnitten, nicht auf TKarel, der bereits komplexere Methoden besitzt. Das Tutorial ist als Durchgang vom Anfang bis Ende strukturiert, d. h. die nachfolgenden Teile stützen sich auf die vorangegangenen, sowohl im didaktischen Aufbau als auch bei der Implementierung.

Für die Bedienung der Entwicklungsumgebung sei auf die Delphi-Hilfe und auf die Webseiten am OSZ Handel hingewiesen, die die wichtigsten Dinge für Anfänger erläutern (s. Delphi-Karel-Startseite).
Und - kein Buch kann Erfahrung ersetzen.

OSZ Handel I Informatik	Karel D. Robot **Delphi - IDE**	S. Spolwig 1.0

Ziele:
Was Sie mindestens über Delphi wissen müssen ..

Informationen zur Vertiefung

Delphi ist ein Entwicklungssystem von Profis für Profis gemacht. Deshalb sagen viele, daß es für den Unterricht nicht brauchbar ist. Das ist Unsinn. Unsere Schule benutzt es seit 1995 und in dieser Zeit ist daran noch kein Schüler verzweifelt oder Lehrer deswegen vorzeitig in Pension gegangen und die guten Ergebnisse in Unterricht beweisen es. Wenn man sich als Anfänger verständig damit beschäftigt und die wichtigsten Regeln beherzigt, macht es keinen Ärger und liefert Programme, die sich vorzeigen lassen.

6 Todsünden

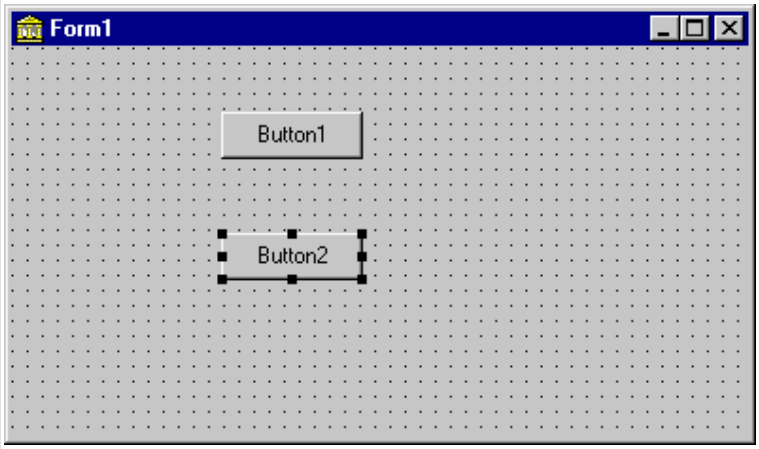
Programmsteuerung in der Symbolleiste

Projekt öffnen
Alles speichern
Starten
Umschalten
u. a. **...**

Die Symbolleisten dienen zur schnellen Ausführung von Programmfunktionen, die in Hinweisblasen gezeigt werden:

Delphi-Dateien

Formular-Fenster

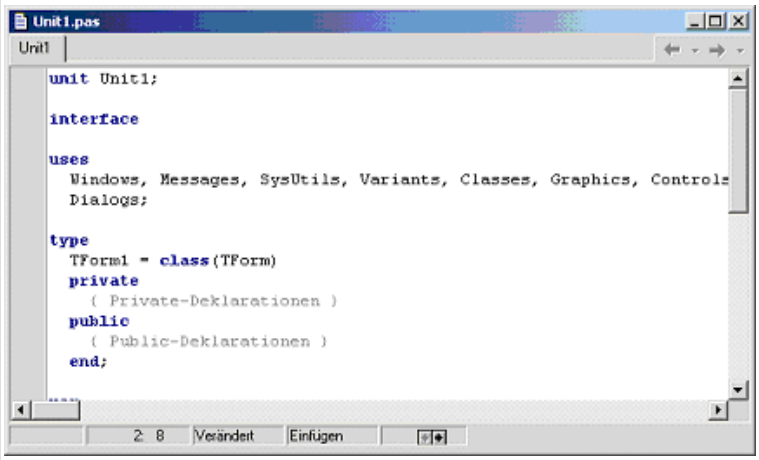


Das Formular ist das Fenster, in dem Ihr Anwendungsprogramm später unter Windows läuft.

Hier entwerfen Sie am Bildschirm die Benutzungsoberfläche des Programms. Das Formular nimmt alle GUI-Komponenten auf, die zur Ein- und Ausgabe und zur Programmsteuerung benötigt werden.

Unit-Fenster

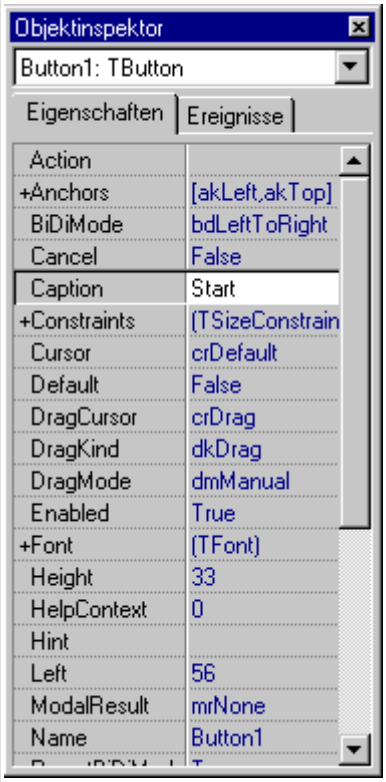
Mit  schalten Sie um auf



Hinter dem Formular-Fenster liegt das dazugehörige Fenster mit dem Delphi-Quellcode für die Komponenten des Fensters und die Objekte des Programms, die hier zum Leben erweckt werden.

Beide zusammen bilden die Schnittstelle (GUI) zwischen dem Programm und dem Menschen.

Objektinspektor



Im Objektinspektor können Sie die Eigenschaften der GUI-Komponenten bearbeiten und die Ereignisse bestimmen, auf die die Objekte reagieren sollen.

Die meisten werden vom System automatisch eingetragen; einige müssen Sie aber per Hand ändern, z.B. *Name* des Objekts.
(Warnung: *Ändern Sie aber nichts, wenn Sie nicht genau wissen, welche Wirkung das hat!)*

Bedienung:

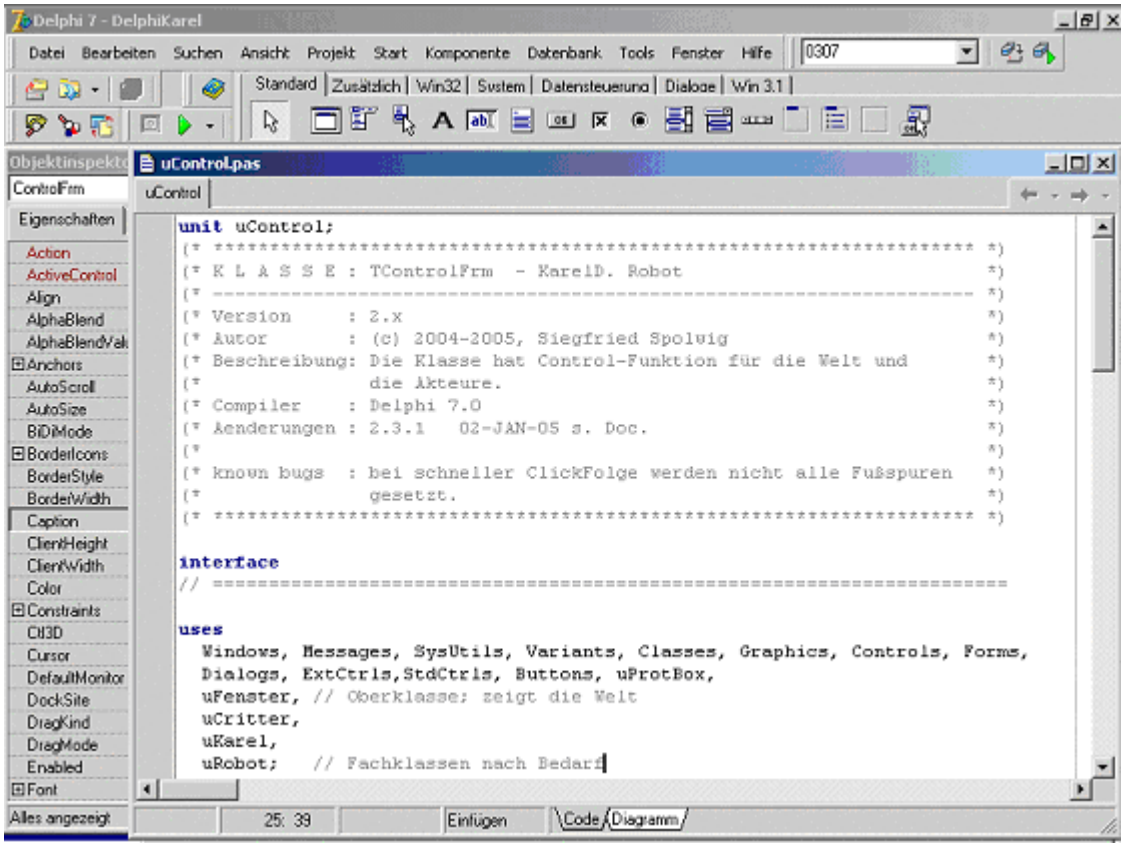
Aktivieren Sie die Komponente, die Sie bearbeiten wollen im Formular. Damit wird der Objektinspektor auf dieses Objekt eingestellt, was sie am Namen im oberen Feld sehen können.

Eigenschaften
Sie klicken auf die Eigenschaft, die Sie ändern wollen und tragen im rechten weißen Feld den entsprechenden Wert ein.

Ereignisse
Die Registerkarte Ereignisse bietet alle Ereignisse an, auf die die Komponente reagieren kann.
Sie wählen links das Ereignis und erzeugen im rechten weißen Feld mit einem Doppelklick eine neue *Ereignismethode* (oder wählen ev. eine der angebotenen aus).

Vorhandenes Projekt öffnen	
1. Delphi aus dem Menü starten	***Startleiste-Delphi***
2. Projekt öffnen	***Symbolleiste-Projekt öffnen***
3. Datei DelphiKarel.**dpr** suchen	***Dialogfenster–DelphiKarel.dpr anklicken***
4. Ev. Umschalten zwischen Formular/Code	***Symbolleiste-Ansicht - Formular / Unit***

Es öffnen sich die Projektfenster. Wenn sich mehr als hier gezeigt öffnen, können Sie die alle schließen. Sie brauchen nur das Editorfenster, das Formularfenster und den Objektinspektor.



Wenn Sie später eigene Units schreiben benutzen Sie bitte die Projektverwaltung.

Dient zur Übersicht, welche Units einem Projekt enthalten sind und um sie schnell zu öffnen.

OSZ Handel I Informatik	Karel D. Robot **Manuelle Steuerung**	S. Spolwig 1.2

Ziele:
Objekt erzeugen und aus der GUI (Graphical User Interface) mit Botschaften steuern

Informationen zur Vertiefung

Wenn Sie das Projekt geladen und gestartet haben, sehen Sie das Trainingscamp, eine schachbrettartige Welt, darauf ein paar Bäume und die Handsteuerung 'Vor' für den Roboter. Das wichtigste Objekt, der Roboter RD1, fehlt noch, weil er noch nicht erzeugt wurde. Das geschieht so:

Objekte

1. Aufgabe:
Ein Objekt erzeugen und initialisieren

a) Beenden Sie Delphi-Karel und

schalten Sie zur Unit uControl mit:  **Menü-Ansicht-Umschalten Formular/Unit**

Sie sehen den Quelltext der Unit, die die Steuerbefehle an die Objekte übermittelt.

b) Roboter RD1 deklarieren im Interface-Teil unter

```
var
 ControlFrm : TControlFrm;
 RD1 : TRobot;        // neuen Roboter deklarieren
```

c) Roboter RD1 erzeugen. Ergänzen Sie die fehlenden Zeilen in der

Methoden

```
procedure TControlFrm.ItemsErzeugen;
// ---------------------------------------------------------------------
// Zweck : Erzeugen und initialisieren aller (neuen) Items in einer Welt
// vorher : -
// nachher: Item ist erzeugt
// ---------------------------------------------------------------------
begin
  RD1 := TRobot.Create;   // erzeugt RD1
  RD1.SetPos('C',1);      // meldet RD1 mit Anfangsposition in der Welt an
  Welt.AllesZeigen;       // in der letzten Zeile, auch wenn nichts gesetzt ist
end;
```

2. Aufgabe:
Einen Button zur Hand-Steuerung aktivieren

Botschaften

Aus der GUI werden dem Roboter die Anweisungen erteilt ("Botschaften übermittelt"), die er ausführen soll. Nimmt dazu Buttons, hat man praktisch so etwas wie eine Fernbedienung. Im Programm ist bereits der erste Button (VorBtn) implementiert und vorbereitet.

```
procedure TControlFrm.VorBtnClick(Sender: TObject);
// ------------------------------------------------
begin
  inherited;   // ist vom System eingefügt
  RD1.Vor;     // Aufruf der Roboter-Methode 'Vor'
end;
```

Fügen Sie in der 2. Zeile RD1.Vor; ein. Der Button sendet die Botschaft 'Vor' an RD1, der dann weiß, daß er einen Schritt weitergehen soll.

Starten Sie das Programm und lassen Sie RD1 marschieren!

OSZ Handel I Informatik	Karel D. Robot Manuelle Steuerung **- GUI-Komponenten**	S. Spolwig 1.3

Ziele:
Buttons in das Formular einfügen

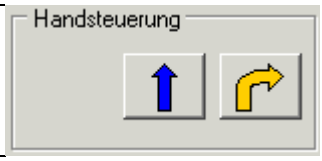
Informationen zur Vertiefung

Ein Roboter, der nur geradeaus laufen kann, dürfte unverkäuflich sein. Wenn er sich drehen kann, sieht es schon besser aus.

1. Aufgabe:
Lesen Sie nach, ob der RD1 schon eine Methode hat, mit der er sich drehen läßt.

Die Roboter

2. Aufgabe:
Neuen Button auf das Formular ziehen



RechtsButton

Gute Bezeichner

a) Ziehen Sie aus der Registerkarte 'Zusätzlich' einen Speedbutton  auf das Feld 'Handsteuerung' im ControlFrm.

b) Wechseln Sie auf den Objektinspektor, der jetzt 'SpeedButton1' anzeigt
- neuen Namen eintragen unter Eigenschaften-Name: **RechtsBtn**
- Bild einfügen unter Eigenschaften-Glyph: Anklicken und im Editor aus dem Ordner Bilder die Datei '**Rechts.bmp**' laden.

c) Die RechtsBtnClick-Methode erzeugen mit Doppelklick auf den Button und darin eintragen: **RD1.RechtsDrehen;**

Starten und testen!

3. Aufgabe
Steuern Sie den RD1 einmal um den Baum auf B,2
- rechts herum und
- links herum.

a) Notieren Sie dabei untereinander, welche Steueranweisungen dazu nötig sind!

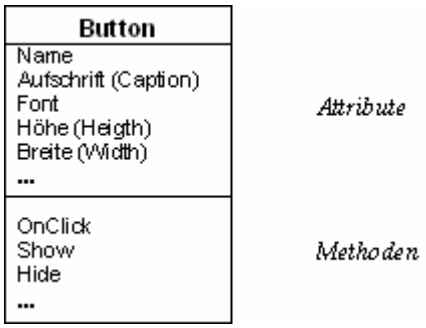
b) Schlagen Sie vor, wie man mit einem Links-Button die Steuerung verbessern kann!

OSZ Handel I Informatik	Karel D. Robot **Die GUI-Objekte**	S. Spolwig 1.4

Ziele:
GUI-Komponenten und Ereignisse verstehen

Informationen zur Vertiefung

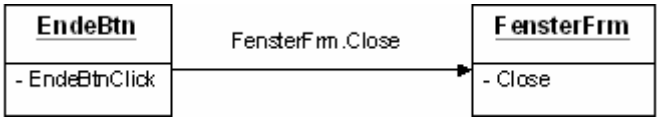
Alle GUI-**Objekte** haben bestimmte **Eigenschaften** (Attribute) und dazu **Methoden**, die sie ausführen können, z. B.



- Bildschirmobjekte können auf bestimmte **Ereignisse** reagieren, z. B. OnClick (Mausklick). Damit wird eine dazugehörige **Ereignismethode** in Gang gesetzt, in der festgelegt werden kann, was das Objekt nach dem Klick tun soll.

- d. h. es werden einer oder mehrere **Aufträge** an andere Objekte geschickt, die für diese verantwortlich sind und sie dann ausführen:

Beispiel:
Auftrag: Der EndeBtn schickt an FensterFrm den Auftrag: *"FensterFrm schließe Dich!"*



Das Zusammenspiel der GUI-Objekte und der Objekte der Fachklassen (Anwendung mit Robots, Bäumen usw.) ergibt dann den gesamten Programmablauf.

*) **GUI** (Graphical User Interface) ist die grafische Bedienungsoberfläche, auf der der Benutzer mit dem Computer kommuniziert.

Merke: **GUI-Objekte sind solche, die das Betriebssystem (und Delphi) fertig zur Verfügung stellt.**

In unserem Beispiel sind das bis jetzt

```
UserPnl     : TPanel;
EndeBtn     : TBitBtn;
GroupBox1 : TGroupBox;
VorBtn      : TSpeedButton;
RechtsBtn  : TSpeedButton;
LinksBtn    : TSpeedButton;
```

wobei FensterFrm und ControlFrm die anderen Objekte als Komponenten-Attribute enthalten.

GUI (Graphical User Interface) ist die grafische Bedienungsoberfläche, auf der der Benutzer mit dem Computer kommuniziert

OSZ Handel I Informatik	Karel D. Robot **Manuelle Steuerung**	S. Spolwig 1.5

Ziele:
Button und separate Methode

Informationen zur Vertiefung

Da RD1 keine eigene Methode zum links drehen hat, lösen wir zunächst das Problem dadurch, daß wir im ControlFrm eine Formular-Methode dafür schreiben und sie dann mit einem neuen Button aufrufen.

1. Aufgabe:
GUI-Methode im Formular schreiben

a) im INTERFACE-Teil die neue Prozedur deklarieren bei

Aufbau einer Unit

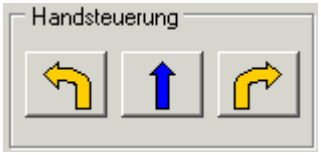
```
TControlFrm = class(TFensterFrm)
  UserPnl : TPanel;
  ...
  ...
  procedure NachLinksDrehen;
  procedure ItemsErzeugen; // darf nicht gelöscht werden!
end;
```

b) im IMPLEMENTATION-Teil unten den Code

```
procedure TControlFrm.NachLinksDrehen;
// -----------------------------------------------------------------
begin
  RD1.RechtsDrehen;
  RD1.RechtsDrehen;
  RD1.RechtsDrehen;
end;
```

2. Aufgabe:
Neuen Button auf das Formular ziehen

LinksButton



Gehen Sie genauso vor wie beim RechtsButton!

Starten und testen!

Merke: Häufig lassen sich bereits **vorhandene Methoden** für andere Aufgaben **wiederverwenden**!

OSZ Handel I Informatik	Karel D. Robot **Sequenzen -** Automatische Robotersteuerung	S. Spolwig 2.1

Ziele:
Sequenz als Folge von Anweisungen verstehen, die nach einander ausgeführt werden.

Informationen zur Vertiefung

Roboter können eine „harte Programmierung“ haben, d. h. Sie als Programmierer legen aus Kenntnis der Situation jede einzelne Aktion fest oder der Roboter benutzt seine Sensoren und entscheidet unterwegs selbständig, was zu tun ist.

Die Roboter

In diesem Abschnitt lernen Sie die drei grundsätzlichen Steueranweisungen, die in der Programmierung verwendet werden: Sequenz, Auswahl, Wiederholungen. Diese Kontrollstrukturen stehen in kleinen Programmstückchen, mit denen der Roboter gesteuert wird.

Algorithmen I

Wir beginnen mit der einfachsten: Sequenz

Für die nachfolgenden Aktionen sollten Sie den Roboter beauftragen, Spuren zu machen, damit man besser kontrollieren kann, wo er lang gegangen ist.

```
procedure TControlFrm.ItemsErzeugen;
// ------------------------------------------------------------
begin
  RD1 := TRobot.Create;
  RD1.SetPos('C',1);
  RD1.MacheSpur;
  Welt.AllesZeigen;
end;
```

---

1. Aufgabe
Dirigieren Sie RD1 von der Ausgangsposition mit der Handsteuerung nach **C,3** und von dort zwischen den Bäumen durch bis **K,3** und notieren Sie die Anweisungen!

Schreibt man diese Anweisungen in eine neue Steuerprozedur, dann kann man den Roboter diesen Weg immer automatisch gehen lassen.

Allgemein ausgedrückt	Übersetzt in die Programmiersprache
gehe eins vor gehe eins vor gehe eins vor nach links drehen ...	RD1.Vor; RD1.Vor; RD1.Vor; NachLinksDrehen; ...

Dazu ist nötig:

a) im INTERFACE-Teil die neue Prozedur deklarieren bei

```
TControlFrm = class(TFensterFrm)
  UserPnl : TPanel;
  ...
  ...
  procedure UmDieBaeume;
  procedure ItemsErzeugen; // darf nicht gelöscht werden!
end;
```

iim IMPLEMENTATION-Teil unten den Code

```
procedure TControlFrm.UmDieBaeume;
// ---------------------------------------------------------------
begin
  RD1.Vor;
  RD1.Vor;
  RD1.Vor;
  NachLinksDrehen;
    ...
end;
```

Zwischenfrage: *Warum kann man nicht RD1.LinksDrehen schreiben?*

Nun fehlt nur noch ein Button, um diese Prozedur aufzurufen.
c) Ziehen Sie aus der Standardpalette einen Button auf das UsrPanel.

d) Wechseln Sie auf den Objektinspektor;
- neuen Namen eintragen unter Eigenschaften-Name: **GoBtn**
- Beschriftung einfügen unter Eigenschaften-Caption: GO!

e) Die ButtonClick-Methode erzeugen und darin eintragen: **UmDieBaeume;**

(Diese GoBtnClick-Prozedur wird in Zukunft die Testprozedur sein für alle neuen Entwicklungen, die wir noch ausprobieren müssen. Wir schreiben hier dann nur die aktuellen Anweisungen hinein. Das spart eine Menge Text im Programm.)

Starten und testen!

Merke: **Sequenzen sind eine Folge von Anweisungen**, die nacheinander ausgeführt werden.

OSZ Handel I Informatik	Karel D. Robot **Auswahl -** Automatische Robotersteuerung	S. Spolwig 2.2

Ziele:
Einseitige Auswahl oder Entscheidung: wenn - dann / if - then

Informationen zur Vertiefung

Algorithmen I

RD1 ist dumm, aber stark. Er wälzt jeden Baum nieder, deshalb werden wir die Steuerung so umschreiben, daß er nur unter der Bedingung vorwärts gehen darf, wenn das Feld vor ihm frei ist. Bei seinen Sensoren ist entsprechendes dabei. Wir können also sagen

Allgemein ausgedrückt	Übersetzt in die Programmiersprache
wenn VorneFrei ist **dann** gehe eins vor	**if** RD1.VorneFrei **then** RD1.Vor;

1. Aufgabe
Schreiben Sie eine neue Prozedur Vorwaerts2

a)
```
procedure TControlFrm.Vorwaerts2 ;
// -------------------------------------------------------------
begin
  if RD1.VorneFrei
  then RD1.Vor;
end;
```

b) Im INTERFACE-Teil deklarieren: **procedure Vorwaerts2;**

c) Mit dem GoBtn aufrufen

```
procedure TControlFrm.GoBtnClick(Sender: TObject);
// -------------------------------------------------------------
begin
  inherited;
  // UmDieBaeume;  // auskommentieren oder loeschen
  Vorwaerts2;
end;
```

Starten und testen!

Merke: **In der einseitigen Auswahl wird eine Bedingung geprüft, ob sie wahr oder falsch ist**
und dann die entsprechende Anweisung ausgeführt.

OSZ Handel I Informatik	Karel D. Robot **Auswahl -** Automatische Robotersteuerung	S. Spolwig 2.3

Ziele:
Zweiseitige Auswahl oder Entscheidung: wenn - dann - sonst / if - then - else

Informationen zur Vertiefung

Schon ganz gut, aber leider bleibt RD1 vor einem Baum stehen und ruht sich aus. Besser wäre es, er würde an dem Baum vorbei gehen. Wir können also sagen

Allgemein ausgedrückt	Übersetzt in die Programmiersprache

wenn VorneFrei ist
dann gehe eins vor
sonst
drehe nach rechts
gehe eins vor
drehe nach links
gehe eins vor
gehe eins vor
drehe nach links
gehe eins vor
drehe nach rechts
	if RD1.VorneFrei
	then RD1.Vor
	else
	begin
	RD1.RechtsDrehen;
	RD1.Vor;
	NachLinksDrehen;
	RD1.Vor;
	RD1.Vor;
	NachLinksDrehen;
	RD1.Vor;
	RD1.RechtsDrehen
	end

Algorithmen II

In einem Auswahlzweig können auch mehrere Anweisungen als ein Block stehen. Dieser Block wird zwischen begin und end eingeschlossen. Vergessen Sie das begin nach else, würde nur die erste Anweisung RD1.RechtsDrehen als Alternative ausgeführt werden und nicht alle Anweisungen des Blocks.

1. Aufgabe
Schreiben Sie eine neue **procedure TControlFrm.Vorwaerts3**
mit diesen Anweisungen. (wie Vorwaerts2).

Starten und testen!

2. Aufgabe
Der RD1 soll bei Programmstart auf Feld **A,3** stehen.
Mit dem GoButton wird er auf der Zeile 3 gestartet und soll dann um die Bäume herum in der Zeile bis **N,3** gehen.

Schreiben Sie eine neue **procedure TControlFrm.Vorwaerts4.**
Tip: Wiederverwendung!

Die Startposition ändern Sie direkt in **SetPos()** in der
procedure TControlFrm.ItemsErzeugen.

Merke: **In der zweiseitigen Auswahl wird eine Bedingung geprüft, ob sie wahr oder falsch ist**
und dann entweder die eine oder die andere Anweisung ausgeführt.

OSZ Handel I Informatik	Karel D. Robot **Wiederholungen -** Automatische Robotersteuerung	S. Spolwig 2.4

Ziele:
Schleifen (Iterationen) als Konstrukt für Aktionen, die mehrfach wiederholt werden.

Informationen zur Vertiefung

Situation

RD1 soll von **C,1** genau 8 Schritte vor gehen. Die Primitivlösung dafür wäre, 8 mal 'Vor' als Sequenz untereinander in eine Prozedur zu schreiben. Für Aktionen, die öfter wiederholt werden sollen, gibt es praktische verschiedene Sprachkonstrukte: Schleifen.

Algorithmen II

1. Wiederholung mit Zählschleife


Anwendung: **Wenn vorher zahlenmäßig feststeht, wie oft wiederholt wird.**

Allgemein ausgedrückt	Übersetzt in die Programmiersprache

mit Zähler von 1 **bis** 8 wiederhole
Vorwärts
	for zaehler := 1 **to** 8 **do**
	begin
	RD1.Vor;
	end;

1. Aufgabe
Schreiben Sie eine neue Prozedur ZumBaum1

a) Deklarieren Sie ZumBaum1;

b) Schreiben Sie den Code

```
procedure TControlFrm.ZumBaum1;
var zaehler : integer;     // Zählvariable deklarieren
begin
  for zaehler := 1 to 8 do // bei 1 anfangen und weiterzählen bis 8
  begin                    // was im Block steht, wird wiederholt
    RD1.Vor;
  end;
end;
```

c) Rufen Sie die Prozedur im Testbutton auf.

Starten und testen!

2. Wiederholung mit selbststeuernder (vorprüfender) Schleife

Situation
RD1 soll solange im Kreis herum laufen wie die Batterie reicht.


Anwendung: **Wenn vorher NICHT feststeht, wie of wiederholt wird.**

Die Aktion wird solange wiederholt, wie die Bedingung erfüllt ist.
Die Bedingung wird vor der Aktion geprüft, d.h. die Aktion wird möglicherweise nie ausgeführt.

Allgemein ausgedrückt	Übersetzt in die Programmiersprache

solange Batterie NICHT leer wiederhole
Vorwärts
Nach links drehen
Vorwärts
Nach links drehen
Vorwärts
Nach links drehen
Vorwärts
Nach links drehen
	while NOT RD1.BatterieLeer **do**
	begin
	RD1.Vor;
	NachLinksDrehen;
	RD1.Vor;
	NachLinksDrehen;
	RD1.Vor;
	NachLinksDrehen;
	RD1.Vor;
	NachLinksDrehen;
	end;

2. Aufgabe
Schreiben Sie eine neue Prozedur ZumBaum2;

b) Deklarieren **procedure ZumBaum2;**

c) Schreiben Sie den Code

```
procedure TControlFrm.ZumBaum2;
// -------------------------------------------------------
begin
  RD1.SetGeschwindigkeit(5);    // etwas mehr Tempo bitte!

  while NOT RD1.BatterieLeer do // Bedingung prüfen, ob wahr
  begin
    RD1.Vor;
    NachLinksDrehen;
    RD1.Vor;
    NachLinksDrehen;
    RD1.Vor;
    NachLinksDrehen;
    RD1.Vor;
    NachLinksDrehen;
  end;
end;
```

d) Rufen Sie die Prozedur im Testbutton auf.

Starten und testen!

OSZ Handel I Informatik	Karel D. Robot **Rekursion -** Automatische Robotersteuerung	S. Spolwig 2.5

Ziele:
Rekursion als Wiederholung durch Schachtelung und als Konstrukt für Aktionen, die mehrfach wiederholt werden, obwohl es keine Schleife ist.

Informationen zur Vertiefung

Situation

Wie vorher. RD1 soll von **C,1** geradeaus gehen bis er einen Baum gefunden hat.

Algorithmen II

3. Wiederholung durch Rekursion

Allgemein ausgedrückt	Übersetzt in die Programmiersprache

Suchen
wenn Du einen Baum gefunden hast
dann bist du fertig
sonst
gehe 1 Vor
Suchen (das ganze noch mal von vorne)
	procedure BaumSuchen;
	begin
	if NOT RD1.VorneFrei
	then
	else // tue nichts mehr
	begin
	RD1.Vor;
	BaumSuchen;
	end;
	end;

Bei einer direkten Rekursion ruft sich die Prozedur selbst auf. Das kann eine Endlosschleife werden, wenn die Abbruchbedingung nicht erfüllt wird. Deshalb gelten Rekursionen zwar als elegant und oft als effizient, aber auch als gefährlich. (Dieses Beispiel ist gefährlich! Unter welchen Umständen?)

1. Aufgabe
Schreiben Sie eine neue Prozedur BaumSuchen

a) Deklarieren Sie **procedure BaumSuchen;**

b) Schreiben den Code

```
procedure TControlFrm.BaumSuchen;
// ------------------------------------------------------------
begin
  if NOT RD1.VorneFrei // er hat was gefunden
  then // tue nichts mehr
  else
    begin
      RD1.Vor;
      BaumSuchen; // die ganze Sache wieder von vorne
    end;
 end;
```

c) Rufen Sie die Prozedur im Testbutton auf.

Starten und testen!


Merke: Am Anfang muß eine erfüllbare Abbruchbedingung stehen,
sonst ruft sich die Prozedur endlos auf bis der Arbeitsspeicher nicht mehr ausreicht und der Computer abstürzt.

OSZ Handel I Informatik	Karel D. Robot **Klassen und Beziehungen**	S. Spolwig 3.1

Ziele:
Klassen als Konstruktionsvorlage für Objekte (Exemplare); Klassenbeziehungen

Informationen zur Vertiefung

Um die nächsten Kapitel zu verstehen, müssen wir uns mit etwas Hintergrundwissen über die Objekte vertraut machen, was bisher beiseite gelassen wurde. Als wir den Roboter RD1 in der 1. Übung mit Create erzeugt haben, konnte er sofort eine ganze Menge. Woher?

Klassen

Klasse

Objekte

Objekte

Für jedes Objekt gibt es einen 'Bauplan', die Beschreibung seiner Klasse. Dort ist festgelegt, welche Attribute (Farbe, Länge, Breite usw.) und welche Methoden es hat. Man kann natürlich auch mehrere Objekte von einer Klasse ableiten (Probieren Sie es aus!), die dann alle völlig gleich in ihrer Art sind, aber jedes unterschiedliche Attributwerte haben und selbständig agieren kann.

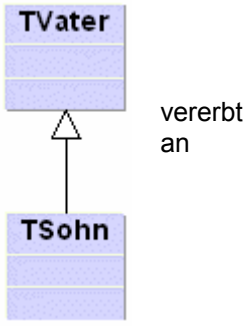
Wenn in einem Programm mehrere Objekte auftauchen, müssen Sie einerseits irgendwie miteinander kommunizieren können. Das tun Sie über ihre Methoden.

Botschaften

Andererseits können Objekte mit anderen auch bestimmte Beziehungen haben, familiäre wie Eltern zu Kindern (Vererbung) oder nur Bekanntschaften (Assoziation) oder sie enthalten eine andere Klasse als komplexes Attribut (Aggregation).

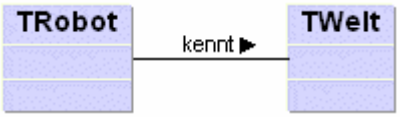
In der grafischen Darstellung sieht das so aus

Beziehungen I

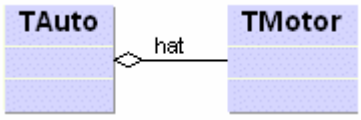


vererbt an

Vererbung



Assoziation



Aggregation

Mit diesen drei Beziehungsarten kann man sehr komplexe Ausschnitte der Welt in einem Modell realitätsnah abbilden und als Software auf einem Rechner implementieren.

OSZ Handel I Informatik	Karel D. Robot **Vererbung**	S. Spolwig 3.2

Ziele:
Neue Klasse einführen als Unterklasse; Vererbungsbeziehung

Informationen zur Vertiefung

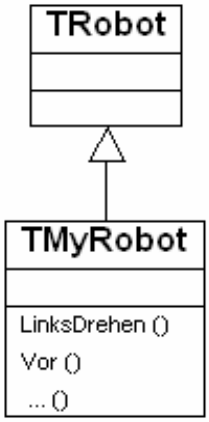
Wenn Mercedes-Benz ein neues Modell plant, fängt man auch nicht wieder von vorn auf dem Stand von 1886 an, sondern entwickelt eine laufende Baureihe weiter mit neuen Eigenschaften, Teilen und Fahrverhalten. Genauso wird im Prinzip Software entwikkelt.

Um einen besseren Roboter zu bauen, nehmen wir als Basis die Klasse TRobot, in der RD1 beschrieben ist, und spezifizieren eine neue Klasse TMyRobot als Unterklasse. Die Unterklasse erbt alles von der Oberklasse und ergänzt nur noch die neuen Attribute und Methoden. Dabei kann man auch vorhandene Methoden durch neue ersetzen (überschreiben).

Vererbung

in der grafischen Darstellung sieht das so aus:

Alles, was TRobot an Attributen und Methoden hat, ist praktisch auch in TMyRobot enthalten als sei es dort eingetragen.



TRobot

Neue Attribute sind z. Z. nicht unbedingt nötig.

Als neue Methoden sollten wir zunächst die aus den ersten Übungen implementieren, die im ControlFrm ausprobiert wurden und nun als robot-eigene Methoden übernommen werden können.

Wir brauchen also eine neue Unit, in der TMyRobot geschrieben wird. Dazu kann man eine völlige leere Unit von Delphi nehmen oder man benutzt eine halbfertige Vorlage. Das werden wir hier tun.

1. Aufgabe:
Die neue Klasse TMyRobot implementieren

a) Unit hinzufügen mit
Delphi-Menü-Projekt-Dem Projekt hinzufügen .... uMyRobot.pas
Prüfen Sie mit Ansicht-Projektverwaltung, ob die Datei unter DelphiKarel.exe aufgeführt wird.

Klassendiagramm

b) Entfernen Sie im Editor alle geschweiften Kommentarklammern **{ }**,
mit denen Textteile ausgeblendet sind.

c) In der Methode **Init** können Sie Attributwerte festlegen, die MyRobot beim Programmstart haben soll: SetPos(...) , SetFarbe(..), SetFuellfarbe(..)
---> s. Spezifikationen

```
procedure TMyRobot.Init;
// ------------------------------------------------------------
begin
  SetFarbe(clYellow);

end;
```

d) Neue Methoden 'Vor', 'LinksDrehen' usw. schreiben

Im INTERFACE-Teil

```
procedure Vor; // die geerbte Methode wird 'überschrieben'
```

Im IMPEMENTATION-Teil

```
procedure TMyRobot.Vor;
// ------------------------------------------------------------
begin
  if VorneFrei
  then inherited Vor; //dann nimm die geerbte (inherited) Meth.
end;
```

Weil in der Klasse die Methoden für alle späteren Exemplare (Robot-Objekte) geschrieben werden, fällt hier der Namensaufruf RD1 weg.

---

2. Aufgabe:
Die neue Klasse im ControlFrm verwenden

Vorbereitung

Für den neuen Roboter sollten wir uControl.pas freiräumen.
a) Legen Sie von uControl.pas unter Windows eine Kopie zum Nachlesen im Ordner Backup an.

b) Löschen Sie im Interface und in Implementation die

```
procedure NachLinksDrehen;
procedure UmDieBaeume;
procedure Vorwaerts2;
procedure Vorwaerts3;
procedure Vorwaerts4;
procedure ZumBaum1;
procedure ZumBaum2;
procedure BaumSuchen;
```

(Die folgenden Schritte gelten für alle neuen Units entsprechend!)

Im INTERFACE-Teil

a) Neue Unit einbinden
unter uses einfügen: **uMyRobot;**

b) Neue Variable deklarieren
unter var einfügen: **Robby : TMyRobot;**

Im IMPEMENTATION-Teil

c) Robby erzeugen in TControlFrm.ItemsErzeugen und Position setzen, wie gehabt.

Ersetzen Sie die Zeile RD1 := TRobot.Create;
durch Robby := TMyRobot.Create;

d) Menü-Suchen-Ersetzen Sie in den Prozeduren alle **RD1**.xxx durch **Robby**.xxx

Starten Sie das Programm und lassen Sie Robby marschieren!

3. Aufgabe
Ändern Sie TControlFrm.LinksBtnClick(Sender: TObject)!

4. Aufgabe
Große Wiederholung und Übung:

a) Versuchen Sie, die Prozeduren, die wir entwickelt haben, mit Robby neu zu implementieren. Vorsicht - sie dürften etwas anders laufen!

b) Entwerfen und implementieren Sie weitere Methoden, die ein gut konstruierter Robot haben sollte!

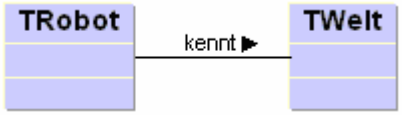
Nicht vergessen:
Objekte können erst dann angesprochen werden, wenn sie schon erzeugt sind!

OSZ Handel I Informatik	Karel D. Robot **Assoziation**	S. Spolwig 4.1

Ziele:
Assoziation als lose 'Kennt'-Beziehung

Informationen zur Vertiefung

Die Welt kann ganz gut ohne Roboter leben und man kann einen Roboter entwickeln, der etwas anderes tun soll als in der Welt herum laufen. Will sagen, beide Objekte können unabhängig von einander existieren.



Wenn sich der Roboter wie hier in der Welt orientieren soll, muß er die Welt kennen. Diese Beziehung wird Assoziation genannt. Es ist eine lose Beziehung ('kennt'), die nach Bedarf aufgebaut und wieder abgebrochen werden kann. Das läßt sich entsprechend implementieren.

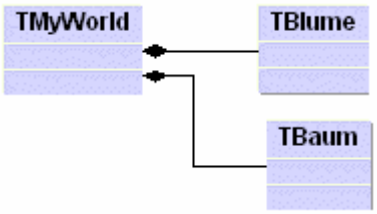
Typische Assoziationen sind die Beziehungen zwischen den Delphi-Formularen und den Fachklassen (die Klassen, die das eigentliche Anwendungsproblem modellieren).

Assoziation

OSZ Handel I Informatik	Karel D. Robot **Aggregation**	S. Spolwig 4.2

Ziele:
Neue Klasse definieren; Aggregation als Enthaltensein-Beziehung verstehen

Informationen zur Vertiefung



Die Welt wird erst schön durch die Dinge, die sie hat, aus der sie besteht. Man kann sagen, die Blumen, die Bäume sind ein Teil der Welt.

Diese Beziehung wird Aggregation genannt. Es ist eine festere Beziehung, bei der eine Klasse ein Teil einer anderen Klasse ist, also in ihr enthalten ist. Das läßt sich dadurch implementieren, daß beim Erzeugen eines Exemplars das andere enthaltene miterzeugt wird.

Aggregation

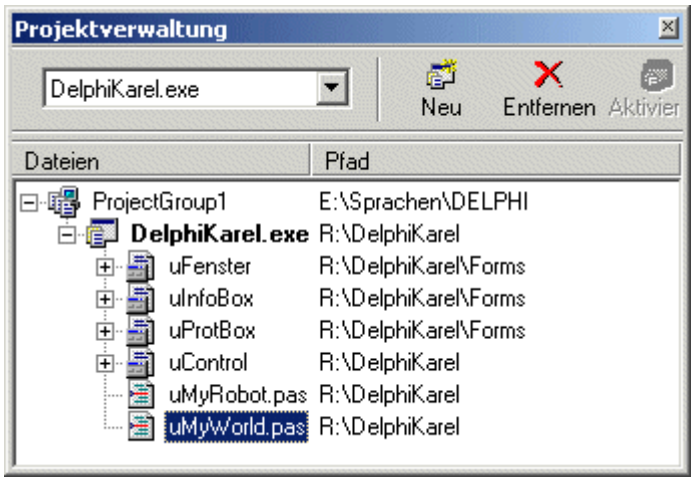
Wir brauchen also eine neue Unit, in der TMyWorld geschrieben wird. Dazu werden wir eine völlige leere Unit von Delphi nehmen (statt der, die bereits im Projekt vorhanden ist).

1. Aufgabe
Entwerfen und schreiben Sie eine neue Klasse TMWorld

a) Neue Unit einführen - **Datei-Neu-Unit**

b) Neue Unit1 umbenennen mit - **Datei-Speichern unter... uMyWorld.pas** im Hauptverzeichnis.
Damit wird die Datei gleichzeitig dem Projekt hinzugefügt und man braucht es nicht extra zu tun.

Prüfen mit
Ansicht-Projektverwaltung.



Wenn Sie diesen Namen benutzen, läßt sich die Welt aus dem Projektmenü aufrufen, sonst müssen Sie in ControlFrm die Unit importieren, deklarieren und ein Exemplar kreieren, wie bei jedem anderen Objekt auch.

c) Schreiben Sie mit dem Editor den gesamten Code in die neue Unit und fügen Sie alles ein, was Sie möchten.

```
UNIT uMyWorld;
(* ************************************************************************************** *)
(* K L A S S E : TMyWorld                                                                 *)
(* -------------------------------------------------------------------------------------- *)
(* Version          : 2.2                                                                 *)
(* Autor            :                                                                     *)
(* Beschreibung: Die Klasse bildet die Welt 'MyWorld'mit einem Haus und                   *)
(*                    einem Monster ab.                                                   *)
(* Compiler         : Delphi 6                                                            *)
(* Aenderungen : 0.9            28-MAR-04                                                 *)
(* ************************************************************************************** *)

INTERFACE
// ============================================================
uses
  uWelt,
  uCritter; // und was man man sonst gern drin hätte

type
  TMyWorld = class(TWelt) // erbt alles von TWelt
   protected
    Haus    : THaus;
    Critter  : TCritter;
   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: Haus, Critter erzeugen und init.
             Alle Objekte der vorigen Welt entfernen
  vorher :
  nachher: done.

Init
  Auftrag: Anfangswerte setzen
  vorher :
  nachher: Fuellfarbe grün, Rasterlinie Silber

---------------------------------------------------------------- *)

IMPLEMENTATION
// ================================================================
USES graphics,
  uFenster;  // importiert Welt und was gebraucht wird

constructor TMyWorld.Create;
// ------------------------------------------------------------
begin
  inherited Create;
  Welt.AlleItemsEntfernen;  // alten Kram weg
```

```
  Haus := THaus.Create;     // die aggregierten Objekte erzeugen
  Haus.SetPos ('H',2);
  Critter := TCritter.Create;
  Critter.SetPos ('A',8);
  Critter.SetBild ('.\bilder\monster.bmp');

// ..... und was Sie sonst brauchen
  Init;
  Welt.AllesZeigen;
end;


procedure TMyWorld.Init;
// -----------------------------------------------------------------
begin
  SetFuellFarbe(clgreen);
  Rasterlinie.SetFarbe(clSilver);
end;

END. // -------- UNIT -------------
```

Starten, im Menü aufrufen und testen!


Tip: Sehen Sie sich auch uCleanCity.pas u. a. als Vorlage an

Für den Anfänger gibt sich die objektorientierte Denkweise etwas verwirrend, weil für denselben Sachverhalt oft unterschiedliche Ausdrücke verwendet werden je nach dem von welchem Standpunkt man die Situation betrachtet.

Auf der Ebene der objektorientierten Analyse (OOA) wird man eine allgemeinere Notation vorziehen, weil zu diesem Zeitpunkt noch keine Gedanken an die spätere Implementation verschwendet werden sollen. Die nachstehende Tabelle gibt den jeweiligen Sachverhalt auf verschiedenen Ebenen am Beispiel einer Kugel wieder.

Äquivalente Begriffe

Realwelt	OO-Analysesicht	OO-Designsicht	Implementationssicht (Delphi)
alle Kugeln einer Art mit gleichen Eigenschaften und Verhalten	**Klasse** Kugel	**Klasse** (Modul)	**Type** type TKugel = class (in einer Unit)
eine rote Kugel davon (mit individuell ausgeprägten Eigenschaften)	**Objekt** Kugel	**Exemplar**, Instanz, Objekt	**Variable** var Kugel
Eigenschaft	**Attribut**	**Attribut**	**Variable** var Farbe
Verhalten	**Methode**	**Methode**	**procedure** Rollen **function** GetColor


Hausinterne Regel: **1 Realobjekt ==> 1 Klasse ==> 1 Unit**

OSZ Handel I Informatik	Karel D. Robot **Anhang -** Lösungen	S. Spolwig 5.2

Kap.				
	1.2 1.3 1.5	2.1 2.2 2.3	3.2	4.2

1.2 Roboter einbauen und Button zur Steuerung aktivieren

1.c)

```
procedure TControlFrm.ItemsErzeugen;
// ------------------------------------------------------------------
// Zweck : Erzeugen und initialisieren aller (neuen) Items in einer Welt
// vorher : -
// nachher: Item ist erzeugt
// ------------------------------------------------------------------
begin
  RD1 := TRobot.Create;    // erzeugt RD1
  RD1.SetPos('C',1);       // meldet RD1 mit Anfangsposition in der Welt an
  Welt.AllesZeigen;  // in der letzten Zeile, auch wenn nichts gesetzt ist
end;
```

2.

```
procedure TControlFrm.VorBtnClick(Sender: TObject);
// ------------------------------------------------
begin
  inherited;  // ist vom System eingefügt
  RD1.Vor;    // Aufruf der Roboter-Methode 'Vor'
end;
```

1.3 Buttons in das Formular einfügen

```
procedure TControlFrm.RechtsBtnClick(Sender: TObject);
// ------------------------------------------------------------------
begin
  inherited;
  RD1.RechtsDrehen;
end;
```

```
3a)
Rechts herum

Vor C 2
Vor C 3
RechtsDrehen C 3
Vor B 3
Vor A 3
RechtsDrehen A 3
Vor A 2
Vor A 1
RechtsDrehen A 1
Vor B 1
Vor C 1

Links herum

RechtsDrehen C 1
Vor B 1
Vor A 1
RechtsDrehen A 1
RechtsDrehen A 1
RechtsDrehen A 1
Vor A 2
Vor A 3
```

```
RechtsDrehen A 3
RechtsDrehen A 3
RechtsDrehen A 3
Vor B 3
Vor C 3
RechtsDrehen C 3
RechtsDrehen C 3
RechtsDrehen C 3
Vor C 2
Vor C 1
```

3b)

3 Mal nach RechtsDrehen

1.5 Interne GUI-Methode im Formular schreiben und mit Button aufrufen

1 a), b)

```
type
  TControlFrm = class(TFensterFrm)
   UserPnl   : TPanel;
   EndeBtn   : TBitBtn;
   GroupBox1: TGroupBox;
   VorBtn    : TSpeedButton;
   RechtsBtn: TSpeedButton;
   LinksBtn : TSpeedButton;

   procedure EndeBtnClick(Sender: TObject);
   procedure RechtsBtnClick(Sender: TObject);
   procedure VorBtnClick(Sender: TObject);
   procedure LinksBtnClick(Sender: TObject);

   procedure NachLinksDrehen;
   procedure ItemsErzeugen; // darf nicht gelöscht werden!
end;
```

2.1 Sequenzen

```
procedure TControlFrm.ItemsErzeugen;
// -------------------------------------------------------------
begin
  RD1 := TRobot.Create;
  RD1.SetPos('C',1);
  RD1.MacheSpur;
  Welt.AllesZeigen;
end;
```

1 b)

```
procedure TControlFrm.UmDieBaeume;
// -------------------------------------------------------------
begin
  RD1.Vor;
  RD1.Vor;
  RD1.Vor;
  NachLinksDrehen;
  RD1.Vor;
  RD1.Vor;
  NachLinksDrehen;
  RD1.Vor;
  RD1.Vor;
  Rd1.RechtsDrehen;
  RD1.Vor;
  RD1.Vor;
  RD1.RechtsDrehen;
  RD1.Vor;
  RD1.Vor;
  NachLinksDrehen;
  RD1.Vor;
```

```
  RD1.Vor;
  NachLinksDrehen;
  RD1.Vor;
  RD1.Vor;
  RD1.RechtsDrehen;
  RD1.Vor;
  RD1.Vor;
  RD1.RechtsDrehen;
  RD1.Vor;end;
end;
```

Warum kann man nicht RD1.LinskDrehen schreiben?
-- Roboter haben keine eigene Methode dafür.

e)

```
procedure TControlFrm.GoBtnClick(Sender: TObject);
// -----------------------------------------------------------------
begin
  inherited;
  UmDieBaeume;
end;
```

2.2 Auswahl, einseitig - Bedingungen

c)

```
procedure TControlFrm.GoBtnClick(Sender: TObject);
// -----------------------------------------------------------------
begin
  inherited;
  Vorwaerts2;
end;
```

2.3 Auswahl, zweiseitig

1)

```
procedure TControlFrm.Vorwaerts3;
// -----------------------------------------------------------------
begin
if RD1.VorneFrei
then RD1.Vor
else
  begin
    RD1.RechtsDrehen;
    RD1.Vor;
    NachLinksDrehen;
    RD1.Vor;
    RD1.Vor;
    NachLinksDrehen;
    RD1.Vor;
    RD1.RechtsDrehen
  end;
end;
```

2)

```
procedure TControlFrm.Vorwaerts4;
// -----------------------------------------------------------------
begin
  NachLinksDrehen;
  Vorwaerts3;
  Vorwaerts3;
  Vorwaerts3;
  Vorwaerts3;
  Vorwaerts3;
  Vorwaerts3;
  Vorwaerts3;
```

```
  Vorwaerts3;
  Vorwaerts3;
end;
```

3.2 Vererbung

1.)

```
UNIT uMyRobot;
(* ************************************************************** *)
(* K L A S S E : TMyRobot                                         *)
(* -------------------------------------------------------------- *)
(* Version : 0.9                                                  *)
(* Autor :                                                        *)
(* Beschreib.: Die Klasse bildet den neuen verbesserten Roboter der *)
(*               3. Generation ab.                                *)
(* Compiler : Delphi 7                                            *)
(* Aenderungen : 0.9 05-APR-04                                    *)
(* ************************************************************** *)

INTERFACE
// ============================================================================
uses
uRobot;

type
TMyRobot = class(TRobot) // erbt alles von Ur-Roboter TRobot
 public
  constructor Create; override;
  procedure Init;
  procedure Vor;
  procedure LinksDrehen;
 private
end;

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

Methoden
--------

Create
Auftrag: Exemplar erzeugen und initilaisieren
vorher :
nachher:

Init
Auftrag: Anfangswerte setzen
vorher : -
nachher:

Vor
Auftrag: Exemplar geht 1 Feld in der aktuellen Richtung vorwärts,
         wenn das Feld frei ist

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

------------------------------------------------------------------ *)



IMPLEMENTATION
// ============================================================================
USES graphics; // importiert TColor
```

```
constructor TMyRobot.Create;
// ------------------------------------------------------------------
begin
  inherited;
  Init;
end;

procedure TMyRobot.Init;
// ------------------------------------------------------------------
begin
  SetFarbe(clYellow);
end;

procedure TMyRobot.Vor;
// ------------------------------------------------------------------
begin
  if VorneFrei
  then inherited Vor;
end;

procedure TMyRobot.LinksDrehen;
// ------------------------------------------------------------------
begin
  RechtsDrehen;
  RechtsDrehen;
  RechtsDrehen;
end;

END. // -------- UNIT -------------


2.)

unit uControl;
(* ************************************************************** *)
(* K L A S S E : TControlFrm - KarelD. Robot                      *)
(* -------------------------------------------------------------- *)
(* Version     : 2.2.1                                            *)
(* Autor       : (c) 2004, Siegfried Spolwig                      *)
(* Beschreibung: Die Klasse hat Control-Funktion für die Welt und  *)
(*               die Akteure                                      *)
(* Compiler    : Delphi 6.0                                       *)
(* Aenderungen : 2.2 21-JUL-04 s. Doc.                            *)
(* known bugs  : bei schneller ClickFolge werden nicht alle Fuß-   *)
(*               spuren gesetzt.                                  *)
(* ************************************************************** *)

interface
// ====================================================================

uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, ExtCtrls,StdCtrls, Buttons, uProtBox,
uFenster, // Oberklasse; zeigt die Welt
uKarel,
uRobot,
uMyRobot; // Fachklassen nach Bedarf

type
TControlFrm = class(TFensterFrm)
UserPnl   : TPanel;
EndeBtn   : TBitBtn;
GroupBox1 : TGroupBox;
VorBtn    : TSpeedButton;
RechtsBtn : TSpeedButton;
LinksBtn  : TSpeedButton;
GoBtn     : TButton;
procedure EndeBtnClick(Sender: TObject);
procedure RechtsBtnClick(Sender: TObject);
procedure GoBtnClick(Sender: TObject);
```

```
procedure VorBtnClick(Sender: TObject);
procedure LinksBtnClick(Sender: TObject);

procedure ItemsErzeugen; // darf nicht gelöscht werden!
end;

var
  ControlFrm : TControlFrm;
  RD1   : TRobot;
  Robby : TMyRobot;

implementation
{$R *.dfm}
// ====================================================================

procedure TControlFrm.ItemsErzeugen;
// --------------------------------------------------------------------
// Zweck : Erzeugen und initialisieren aller (neuen) Items in einer Welt
// vorher : -
// nachher: Item ist erzeugt
// --------------------------------------------------------------------
begin
  Robby := TMyRobot.Create;
  Robby.SetPos('C',1);

  Welt.AllesZeigen; // in der letzten Zeile, auch wenn nichts gesetzt ist
end;


procedure TControlFrm.EndeBtnClick(Sender: TObject);
// --------------------------------------------------------------------
// Zweck : Entfernt alle Items aus dem Speicher, speichert und schließt
// die ProtBox und zum Schluss das ControlFrm.
// nachher: Programm beendet.
// --------------------------------------------------------------------
begin
  Welt.AlleItemsEntfernen;
  ProtokollFrm.Close;
  Close;
end;

procedure TControlFrm.VorBtnClick(Sender: TObject);
// --------------------------------------------------------------------
begin
  inherited;
  Robby.Vor;
end;

procedure TControlFrm.RechtsBtnClick(Sender: TObject);
// --------------------------------------------------------------------
begin
  inherited;
  Robby.RechtsDrehen;
end;

procedure TControlFrm.LinksBtnClick(Sender: TObject);
// --------------------------------------------------------------------
begin
  inherited;
  Robby.LinksDrehen;
end;









procedure TControlFrm.GoBtnClick(Sender: TObject);
// --------------------------------------------------------------------
begin
```

```
  inherited;
// UmDieBaeume;
// Vorwaerts2;
// Vorwaerts3;
// vorwaerts4
// ZumBaum1;
// ZumBaum2;
// BaumSuchen;
end;

END. //--- Unit --
```

3 c)

Weitere neue Methoden für den Roboter könnten sein:

LinksDrehen; RechtsFrei; LinksFrei; Prüfen, ob er vor dem Abgrund (Offlimits) steht; u. a. m. ..

4.2 Aggregation

```
UNIT uMyWorld;
(* ************************************************************** *)
(* K L A S S E : TMyWorld                                         *)
(* -------------------------------------------------------------- *)
(* Version     : 2.2                                              *)
(* Autor       :                                                  *)
(* Beschreibung: Die Klasse bildet die Welt 'MyWorld'mit einem Haus *)
(*               und einem Monster                                *)
(* Compiler    : Delphi 6                                         *)
(* Aenderungen : 0.9 28-MAR-04                                    *)
(* ************************************************************** *)

INTERFACE
// ============================================================================
uses
  uWelt,
  uHaus,
  uCritter; // und was man man sonst gern drin hätte

type
  TMyWorld = class(TWelt)
   protected
    Haus : THaus;
    Critter : TCritter;
   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: Haus, Critter 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: -

------------------------------------------------------------------------ *)

IMPLEMENTATION
// ========================================================================
USES graphics,
uFenster; // importiert Welt und was gebraucht wird

constructor TMyWorld.Create;
// ------------------------------------------------------------------
begin
  inherited Create;
  Welt.AlleItemsEntfernen; // alten Kram weg

  Haus := THaus.Create; // die aggregierten Objekte erzeugen
  Haus.SetPos('H',2);
  Critter := TCritter.Create;
  Critter.SetPos('A',8);
  Critter.SetBild('.\bilder\monster.bmp');
  Init;
  Welt.AllesZeigen;
end;

procedure TMyWorld.Init;
// ------------------------------------------------------------------
begin
  SetFuellFarbe(clgreen);
  Rasterlinie.SetFarbe(clSilver);
end;

END. // -------- UNIT -------------
```