

November 2004**ed 1.8.1****Ursprung:****SMIMMS 2001, Projektgruppe 7**

Michael Hass, Andreas Krümmberg, Björn Steinbring, Christian Wilms; Leitung Georg Kubitz

Mitarbeit

Stefan Moll (Schwerpunkt: Klassendokumentationen)

**Kontakt und verantwortlich: Georg Kubitz,
Hannah-Arendt Gymnasium,
49525 Lengerich
georg.Kubitz@t-online.de**

Inhaltsverzeichnis

VORWORT:	3
VORBEMERKUNG: WIE ARBEITE ICH MIT UMLED:	4
Das Vorgehen beim Entwurf eines neuen Projektes	4
Das Vorgehen bei der Dokumentation eines vorhandenen Projektes	4
WIE ARBEITE ICH MIT DER BEDIENUNGSANLEITUNG?	4
1. ERSTE SCHRITTE MIT UMLED	5
1.1. Anlegen eines Delphi-Projektes	5
1.2. Anmerkungen zum Anwendungsmuster	6
Öffnen eines UML-Diagrammes	7
1.4. Entwurf einer Klasse:	8
Attribute hinzufügen:	10
Schreib- und Lese-Methoden eines Attributes:	10
Methoden hinzufügen:	12
1.5. Klasse speichern und Code erzeugen	13
1.6. Einbinden und Ergänzen des Codes unter Delphi	15
Aktualisieren des Klassendiagramms:	18
2. DAS ERSTE BEZIEHUNGSDIAGRAMM	18
2.1. Erstellen einer Beziehung	19
2.2. Bearbeiten von Beziehungen	21
Beziehungen löschen:	21
Texte ändern:	21
Schrifttyp ändern:	21
Linienzug wählen:	21
Farbe ändern:	21
Attribut wählen:	22
Anfangs und Endpunkt von Beziehungen verschieben:	22
2.3. Eine wirklich neue Beziehung:	22
3. DAS ERSTE BOTSCHAFTSDIAGRAMM	25
3.1. Erstellen eines neuen Unterdiagramms	25
3.2. Hinzufügen eines Auftrages	27
3.3. Hinzufügen einer Anfrage	28
Wir vervollständigen nun das Diagramm weiter:	30
3.4. Hauptdiagramm anpassen und Code der Klassen erzeugen	32
4. EINBINDEN VON CODE UNTER DELPHI UND AKTUALISIEREN DER UML-DIAGRAMME	33
Code unter Delphi einbinden und ergänzen:	33
Die Diagramme aktualisieren:	34
5. ORGANISIEREN UND SYNCHRONISIEREN DER ARBEIT MIT UMLED UND DELPHI:	35
ANHANG1: UML-DIAGRAMME	36
Klassendiagramme:	36
Botschafts-Diagramme (Kollaborationsdiagramme)	37
Beziehungsdiagramme:	38
IST-Beziehung (Verfeinerung bzw. umgekehrt Verallgemeinerung, Vererbung)	38
Die Kennt-Beziehung (Verbindung, Assoziation)	38
Die Hat-Beziehung (Zerlegung, Aggregation)	38

Vorwort:

Kinder sind schon in sehr jungem Alter in der Lage, ihre Umwelt in Klassen einzuteilen: (*Papa - ein Wau-Wau!!*⁶) . Sie können auch zwischen der Klasse der Hunde (Wau-Waus) und dem konkreten Objekt, nämlich dem Hund Fiffi unterscheiden.

Kein Wunder also, dass eine auf Objekten, Objekt-Klassen (Wau-Wau) und Klassenbeziehungen (WauWau beißt Briefträger) aufbauende Softwareerstellung einen weiteren wesentlichen Schritt der Anpassung von Softwareerstellung an den Menschen darstellt. Es ist das Kunstwerk objektorientierter Analyse (OOA) und objektorientierten Designs (OOD), herauszufinden , welche Objekte zu Klassen zusammengefasst werden sollten und welche Beziehungen diese Klassen untereinander haben.

In der Industrie hat sich die Unified Modeling Language (UML) als Standard-Sprache zur objektorientierten Entwicklung von Softwaresystemen durchgesetzt. UML wurde von der Object Management Group (www.omg.org) im November 1997 als Standard festgelegt und setzt sich zunehmend auch an Universitäten und Schulen als Entwurfs- und Dokumentationswerkzeug durch. **UML ist keine Programmiersprache sondern besteht im wesentlichen aus Vereinbarungen zur graphischen Darstellung von Objektklassen und deren Beziehungen untereinander.**

Weitere Hinweise finden Sie bei learn-line unter:

<http://www.learn-line.de/angebote/oop/index.html>

bzw genauer unter:

<http://www.learn-line.de/angebote/oop/medio/theorie/design/index.html>

Das Oberstufenzentrum Handel I in Berlin liefert zum Thema OOP und UML eine sehr gute Seite:

<http://www.oszhdl.be.schule.de/gymnasium/faecher/informatik/oop/index.htm>

Ein gutes UML-Tutorial auf Universitätsniveau findet sich bei der UNI-Magdeburg:

<http://ivs.cs.uni-magdeburg.de/~dumke/UML/index.htm>

DIE UML-Seite in Deutschland : <http://www.jeckle.de/unified.htm>

UMLed ist ein für die Schule entwickeltes Werkzeug. Es dient dazu, Klassendiagramme, Beziehungsdiagramme und Botschaftsdiagramme (Kollaborationsdiagramme) am Bildschirm entwerfen zu können.

UMLed kann ebenso zur Dokumentation fertiger Projekte eingesetzt werden.

UMLed unterstützt die schulische Arbeit ferner durch **eine ausgezeichnete Verbindung zu Delphi-Quellcodedateien. (Import und Export)**. Hiermit ist es möglich, objektorientierte Programmierung fortlaufend durch **UMLed** zu begleiten.

UMLed ist natürlich **nicht** für die kommerzielle Softwareentwicklung gedacht. Hierfür gibt es mächtige Entwicklungsumgebungen, auf die ggf. zurückgegriffen werden sollte.

Vorbemerkung: Wie arbeite ich mit *UMLed*:

UMLed kennt vier wesentliche Objekte, mit denen es arbeitet: Klassen, Beziehungen zwischen Klassen, Botschaften zwischen Objekten und entsprechend Beziehungs- und Botschaftsdiagramme. Was das im einzelnen genau ist, erfahren Sie weiter unten oder im Anhang.

Das Vorgehen beim Entwurf eines neuen Projektes

Für das User-Interface nutzen Sie wie bisher die graphische Entwicklungsumgebung von Delphi. Für den Entwurf und die Entwicklung der darunter liegenden Klassenstrukturen benutzen Sie **UMLed**.

Entwerfen Sie zuerst die benötigten Klassen. Das Erstellen von Beziehungs- und Botschaftsdiagrammen kann hierbei eine große Hilfe sein. **Speichern Sie die Diagramme im gleichen Verzeichnis wie die Delphi-Quelldateien.** So haben Sie stets alle Entwurfs-, Dokumentations- und Codierungsdateien zusammen.

Während oder nach dem Erstellen der Diagramme können Sie von **UMLed** den Rahmen des Pascal-Codes der entsprechenden Units erzeugen und später sogar verändern. Die Verbindung zwischen Delphi und **UMLed** ist dabei sehr einfach zu handhaben. Näheres finden Sie weiter hinten im Kapitel 5.

Das Vorgehen bei der Dokumentation eines vorhandenen Projektes

Dieses Vorgehen unterscheidet sich nicht sehr von dem obigen. Sie können die beteiligten Klassen einfach aus den Pascal-Units importieren. Nach Ergänzen der Dokumentation wichtiger Attribute und Dienste speichern Sie die Klassen wieder ab und erstellen entsprechende Beziehungsdiagramme.

Wie arbeite ich mit der Bedienungsanleitung?

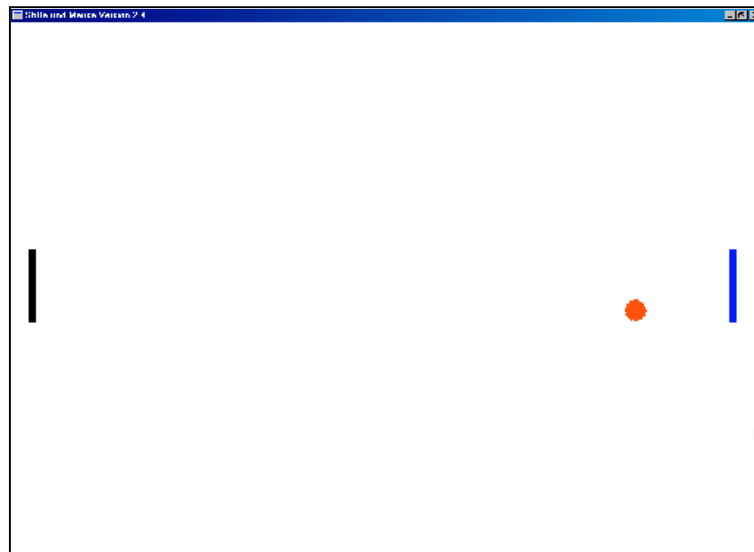
Mir erschien es am sinnvollsten, Ihnen die Arbeit mit Hilfe eines sehr einfachen Beispiels zu erläutern. Arbeiten Sie dieses durch, und es sollten eigentlich keine Fragen mehr übrig sein.

Für Anregungen und Kritik bezüglich des Programmes **UMLed** und dieser Bedienungsanleitung ist der Autor immer dankbar. Melden Sie sich bei:
georg.kubitz@t-online.de

1. Erste Schritte mit UMLed

Die Arbeit mit **UMLed** soll an einem Beispiel aus den Unterrichtsreihen des Projektes Objektorientierte Programmierung des Landesbildungsservers NRW www.learn-line.de erfolgen. Es handelt sich um die Unterrichtsreihe ‚Billard‘. Die hierfür benötigten Stift und Maus-Umgebung befindet sich im Projektverzeichnis ‚Anwendungsmuster‘ das Sie zusammen mit dieser Anleitung in einer gezippten Datei erhalten haben. Sie finden es auch auf meiner Homepage www.kubitz-online.de unter dem Punkt OOP-Fortbildung.

Ziel des Projektes soll eine Art Tischtennisspiel sein:



Ein Ball fliegt dabei schräg über den Bildschirm. Er wird von den Wänden und den Schlägern reflektiert. Das Spiel ist beendet, wenn einer der Spieler den Ball verpasst hat und er auf die dahinter liegende Wand trifft.

Wir beginnen in einem ersten Schritt mit folgender Aufgabe:

Eine Kugel soll sich über den Bildschirm bewegen und von den Rändern des Bildschirmes, die als Wände wirken, reflektiert werden. Das Programm soll mit einem Maustastendruck beendet werden.

Die vollständige Realisierung des obigen Projektes finden Sie im Verzeichnis TischTennis_fertig, dass dieser Anleitung ebenfalls beiliegt.

1.1. Anlegen eines Delphi-Projektes

Da wir Ihnen hier das Zusammenspiel von Delphi und **UMLed** zeigen wollen, beginnen wir mit der Delphi-Entwicklungsumgebung.

Bei 'normalen' Delphi- Programmen entwerfen Sie mit der Delphi-GUI die Oberfläche und speichern das Projekt und die FormularUnit in einem eigenen Verzeichnis ab.

In unserem Fall aber arbeiten wir mit der Stift und Maus-Umgebung. Erstellen Sie einfach eine Kopie des Verzeichnisses 'Anwendungsmuster'. (Dieses Musterprojekt wurde mit dem Handbuch mitgeliefert). Nennen Sie das kopierte Verzeichnis 'TischTennis1'.

1.2. Anmerkungen zum Anwendungsmuster

Das Anwendungsmuster ist auf die Programmierumgebung 'Stifte und Mäuse' der Lehrerfortbildung zur OOP NRW abgestimmt. Die Projektdatei ist keine Delphi-Standarddatei, sondern hat folgenden Code:

```
program Projektdatei;  
  
uses  
  mSuM in 'mSuM.pas';  
  mTAnwendung in 'mTAnwendung.pas';  
  
var  
  Anwendung : TAnwendung;  
  
begin  
  Anwendung := TAnwendung.erzeuge;  
  Anwendung.laufeAb;  
  Anwendung.gibFrei  
end.
```

Wie Sie sehen, wird statt der in Delphi üblichen 'Application' eine 'Anwendung' erzeugt. Werfen wir ebenso einen kurzen Blick in die Anwendungseinheit:

```
UNIT mTAnwendung;  
  
interface  
  
uses mSuM;  
  
type  
  TAnwendung = CLASS  
  
    // weitere Attribute  
  private  
    derBildschirm : Bildschirm;  
    dieMaus : Maus;  
    meinStift : Stift;  
    dieTastatur : Tastatur;  
  
    // weitere Methoden  
  public  
    constructor erzeuge;  
    procedure gibFrei;  
    procedure laufeAb;  
  
  end;  
  
implementation  
  
  //----- erzeuge (public) -----  
constructor TAnwendung.erzeuge;  
begin  
  derBildschirm := Bildschirm.init;  
  meinStift := Stift.init;  
  dieTastatur := Tastatur.init;  
  dieMaus := Maus.init;  
end;
```

```
//----- gibFrei (public) -----
procedure TAnwendung.gibFrei;
begin
    meinStift.gibFrei;
    dieMaus.gibFrei;
    dieTastatur.gibFrei;
    derBildschirm.gibFrei;
end;

//----- laufeAb (public) -----
procedure TAnwendung.laufeAb;
begin
    // hier Code einfügen
end;

end.
```

Sie erkennen, dass die Anwendung das Grundgerüst einer Stift und Maus Anwendung darstellt. Auf diesem Grundgerüst bauen wir die Entwicklung unserer Tischtennis-Anwendung auf. Die Zusammenhänge der Klasse TAnwendung mit den Stift und Maus Klassen wird durch ein im Projektverzeichnis TischTennis1 vorhandenes UML-Diagramm verdeutlicht.

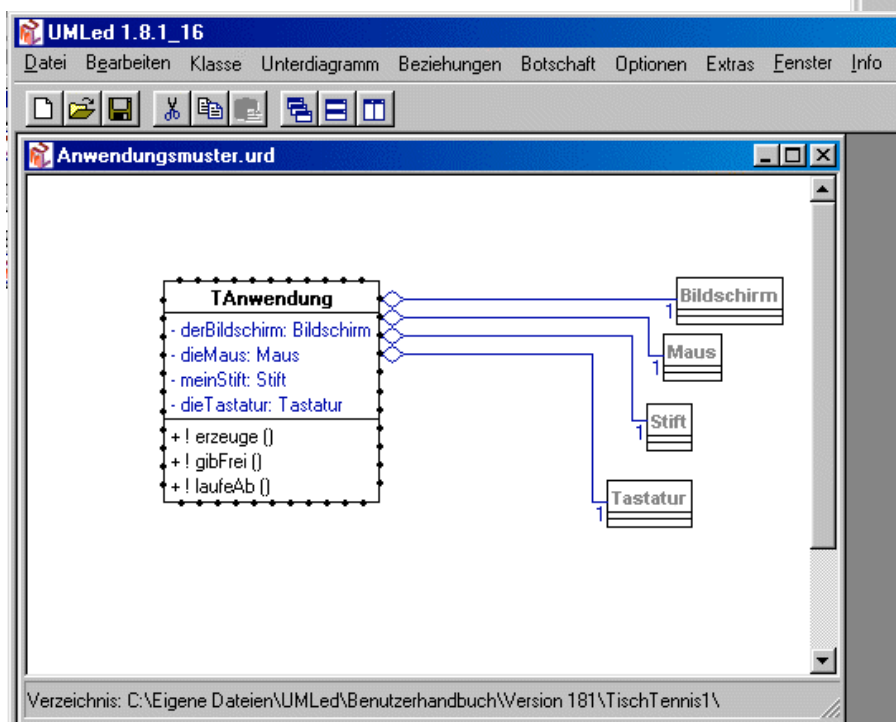
1.3. Öffnen eines UML-Diagrammes

Starten Sie **UMLed**. Es erwartet Sie ein Eingangsbildschirm, der im rechten Fenster ein Menü anbietet.

Wir wählen den Punkt ‚Datei‘ und erhalten:

Über den Menüpunkt 'Diagramm Öffnen' gelangen Sie zum üblichen Dateidialog. Kämpfen Sie sich bis zum Verzeichnis TischTennis1 durch und öffnen Sie das Diagramm Anwendungsmuster.urd.

Sie erhalten dann das folgende UML-Diagramm:



Die Linien mit den kleinen Rauten stellen HAT-Beziehungen dar. Näheres siehe unter <http://www.learn-line.de/angebote/oop/index.html> oder im [Anhang](#).

Das Diagramm macht deutlich, dass eine Anwendung über einen Bildschirm, eine Maus, einen Stift und eine Tastatur verfügt, das heißt, dass es diese erzeugt und am Ende auch wieder löscht.

Hoffentlich ist Ihnen aufgefallen, dass nun neue Hauptmenuepunkte hinzugekommen sind, die erst bei einem geöffneten Diagramm Sinn machen.

Nach diesen Anmerkungen wenden wir uns dem Entwurf unseres Balles zu:

1.4. Entwurf einer Klasse:

Vorüberlegungen: Ein Ball muss sich an einer bestimmten Position des Bildschirms aufhalten. Diese Angaben sind in den Zustandsvariablen `zXPos` und `zYPos` gespeichert. das `z` steht hier für Zustand. Die Größe des Balles wird in `zRadius` gespeichert. Da er sich bewegen soll, benötigt er Zustandsvariablen für seine Geschwindigkeit. Wir geben einfach die Geschwindigkeit in X und in Y-Richtung an und speichern sie in den Zustandsvariablen `zVx` und `zVy`. Der Ball soll sich ferner selbst zeichnen können, er benötigt also einen Dienst `zeichne`. Außerdem muss sich der Ball um ein kleines Stück weiterbewegen können, also benötigt er den Dienst `bewege`. Damit er beim Bewegen sein altes Bild löschen kann, gönnen wir ihm den Dienst `lösche`. Damit er die Zeichenoperationen weitgehend selbständig durchführen kann, gönnen wir dem Ball einen eigenen Stift. Der Ball hat also einen Stift, den wir deshalb mit `hatStift` bezeichnen.

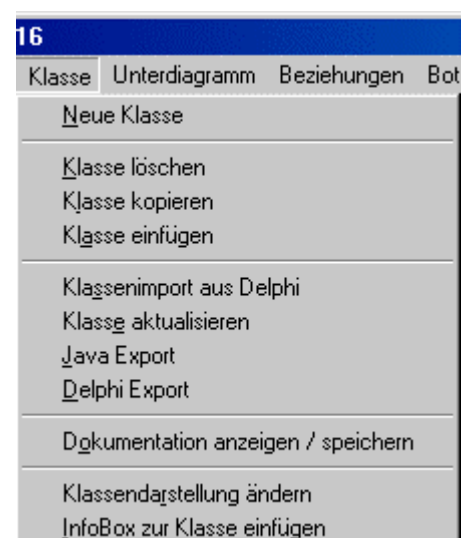
Ferner besitzt der Ball die Dienste `init` als Konstruktor, und `gibFrei` als Destruktor. Die Kugel soll sich schon bei der Initialisierung (`init`) ihre Anfangsposition und ihre Anfangsgeschwindigkeit merken.

Wir übersetzen diese Vorüberlegungen mit Hilfe von **UMLed** nun in ein Klassendiagramm:


Wir arbeiten mit dem Diagramm Anwendungsmuster.urd weiter. Da wir dieses nun ändern werden, speichern wir es erst unter dem Namen `TischTennis1` natürlich im Verzeichnis `TischTennis1` ab.

Wichtig: Das Delphi-Projekt und das Zugehörige UML-Diagramm sollten sich immer im selben Verzeichnis befinden. Das Verzeichnis wird daher stets in der Info-Zeile am unteren Rand der Arbeitsfläche von UMLed angegeben.

Da wir eine neue Klasse entwerfen wollen, wählen wir den Menüpunkt ‚Klasse‘. Studieren Sie bitte die Titel aller hier angebotenen Unterpunkte, dann wissen Sie schon eine Menge über die Möglichkeiten von **UMLed**.

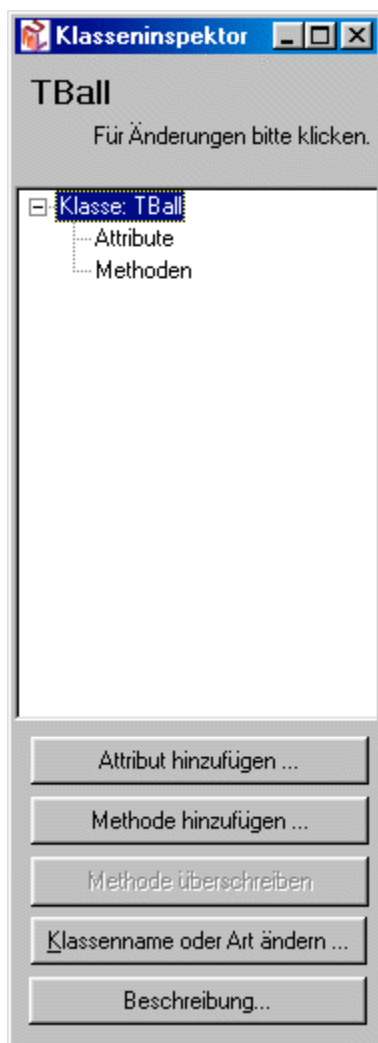


Sie können zum Anlegen einer neuen Klasse auch mit der rechten Maustaste auf die leere Arbeitsfläche klicken. Dann erhalten Sie ein Kontextmenü, das auch den Punkt ‚Neue Klasse‘ enthält:

Wählen Sie den Punkt ‚Neue Klasse‘ und geben als Namen ‚TBall‘ (und nicht Ball) ein, Sie erhalten auf der Zeichenfläche ein kleines Rechteck mit diesem Namen. Wenn Ihnen der Ort des Rechtecks mit TBall nicht gefällt, bewegen Sie die Maus in das Rechteck. Der Mauszeiger ändert sich dann zu einem Doppelkreuz .

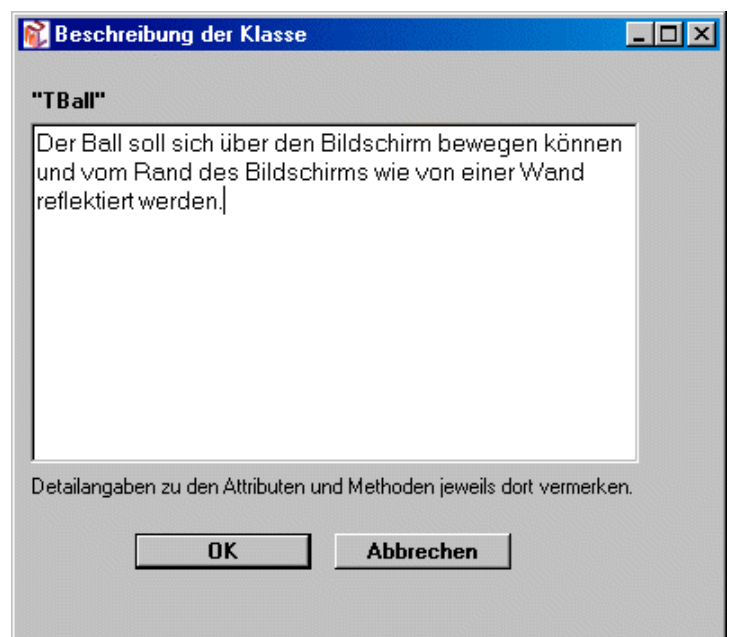
Wenn Sie die Maus nun drücken, können Sie die Klasse bei gedrückter Maustaste an einen anderen Ort verschieben.

Ferner hat sich das linke Fenster geändert!



UMLed besitzt einen sog. Klasseninspektor, der dem Objektinspektor von Delphi ähnelt. Klassen werden über den Klasseninspektor oder über das Kontextmenue bei einem rechten Mausklick in die Klasse geändert. Der Klasseninspektor ist immer sichtbar. Sollten Sie ihn einmal gelöscht haben, lässt er sich über den Menüpunkt 'Optionen' wiederholen.

Unten im Klasseninspektor finden Sie den Punkt ‚Beschreibung‘. Wählen Sie diesen Punkt aus, geben Sie eine kurze Beschreibung eines Balles ein und übernehmen mit ‚OK‘



Attribute hinzufügen:

Nun kommen wir zum Hinzufügen der Attribute. Dies geschieht über den Button ‚Attribut hinzufügen‘ des Klasseninspektors oder nach rechtem Mausklick in die Klasse über das Kontextmenü der Klasse. Es öffnet sich der Attributedialog.

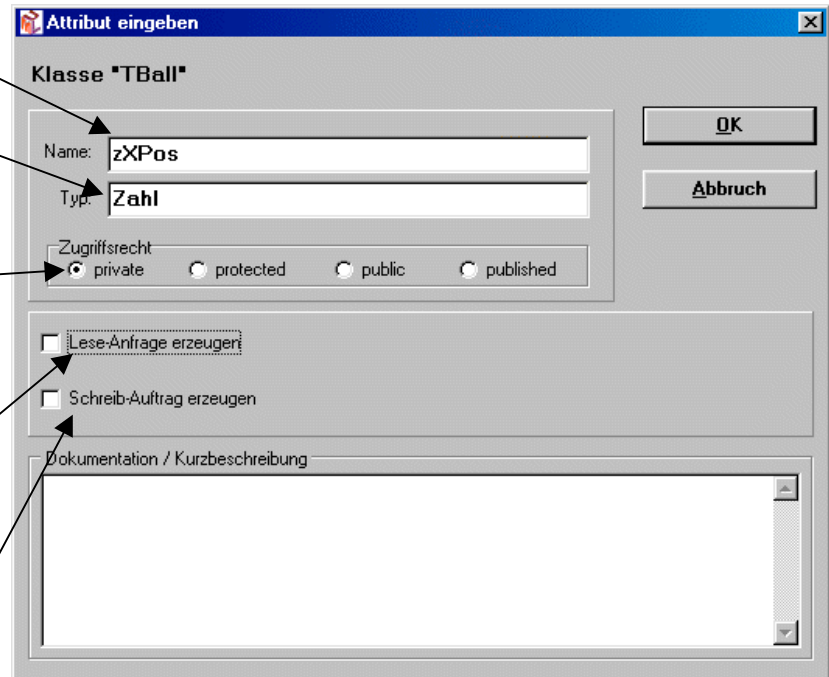
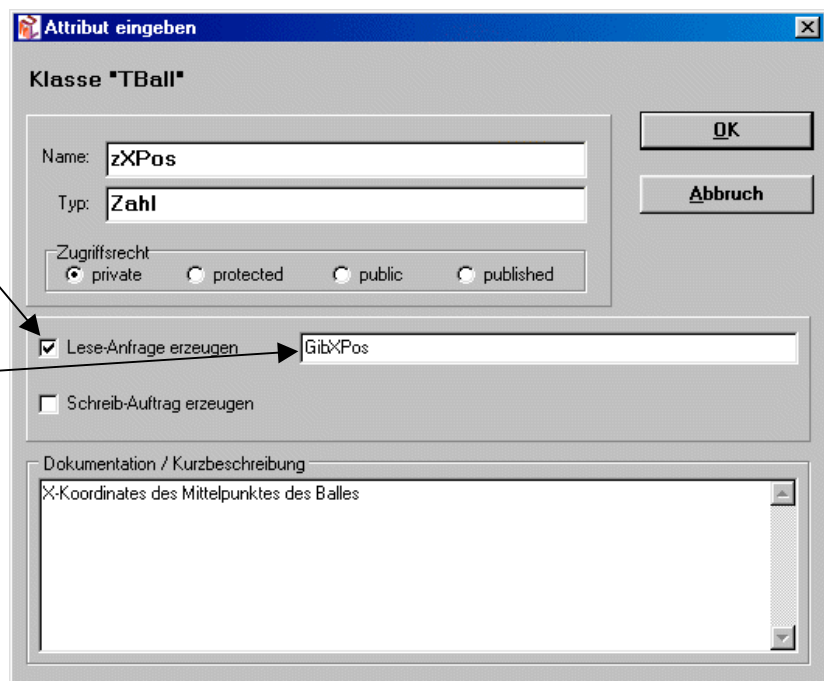
Geben Sie den Namen und den Typ des Attributes ein. **UMLed** prüft dabei nicht, ob es den Typ überhaupt gibt. Das Zugriffsrecht des Attributes belassen Sie bei ‚private‘, auch wenn andere Zugriffsrechte möglich sind.

Schreib- und Lese-Methoden eines Attributes:

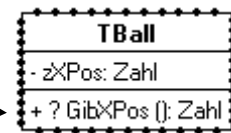
Sie erkennen, dass Sie mit Hilfe des Attribute-Dialoges auch Dienste zum Schreiben und Lesen des Attributes festlegen können. Einen Schreib-Auftrag benötigen wir nicht, da dieser Wert ja bei der Initialisierung eines Balles übergeben werden soll. Da später festgestellt werden soll, ob der Ball den rechten oder den linken Bildschirmrand erreicht hat, sollten wir aber eine Lese-Anfrage für die X-Position vorsehen. Setzen Sie ein entsprechendes Häkchen, dann erscheint auf dem AttributFormular ein Vorschlag für den Namen der Anfrage:

Ändern Sie GibZXPos in GibXPos.

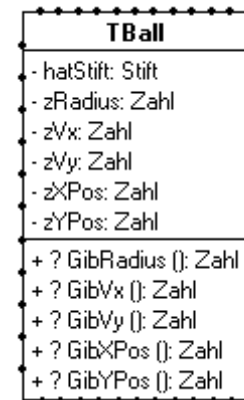
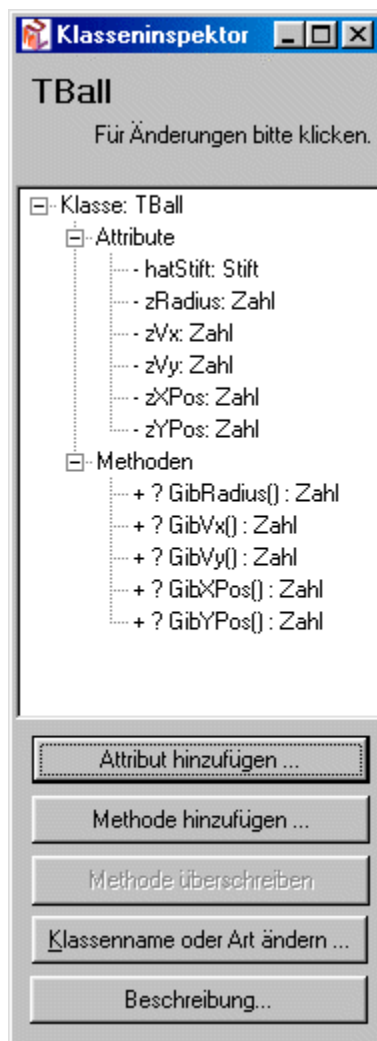
Nach Eingabe einer kurzen Beschreibung verlassen Sie den AttributeDialog über den ‚OK‘ Button

Auf der Arbeitsfläche von **UMLed** erkennen Sie neben dem neuen Attribut auch gleich eine neue Methode, nämlich die eben eingegebene Lese-Methode mit passendem Ergebnistyp.



Fügen Sie die weiteren Attribute entsprechend unseren Vorüberlegungen ein, dann sollte schliesslich die Darstellung der Klasse `TBall` auf der Arbeitsfläche und im Klasseninspektor folgendermaßen aussehen:



Beachten Sie bitte die schon hinzugefügten Lese-Methoden.

Methoden hinzufügen:

Jetzt wollen wir die weiteren Dienste eines Balles eingeben. Hierzu wählen Sie im Klasseninspektor den Punkt 'Methode hinzufügen'. Sie können auch mit der rechten Maustaste in die Klasse klicken und den Menüpunkt aus dem dann erscheinenden Kontextmenü wählen.

Sie erhalten in beiden Fällen das Methodenfenster:

Im oberen Feldern ist der Name der Methode (des Dienstes) einzugeben.

Die Parameter der Methode sind über die Schaltfläche neu einzeln einzugeben.

Wählen Sie als ‚Art der Methode‘ den Punkt ‚Constructor‘ und behalten Sie das Zugriffsrecht ‚public‘ bei.

Die Modifikatoren brauchen Sie zur Zeit noch nicht beachten. Also dort bitte nichts ankreuzen.

Wenn Sie wollen, können Sie auch den Pascal Code der Prozedur eingeben. Dies sollte aber besser später mit Delphi geschehen.

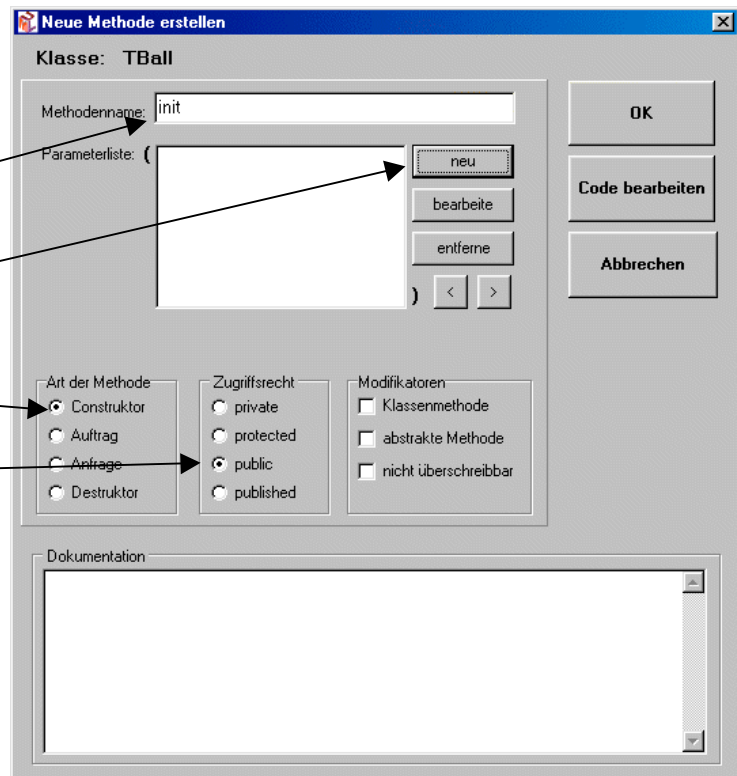
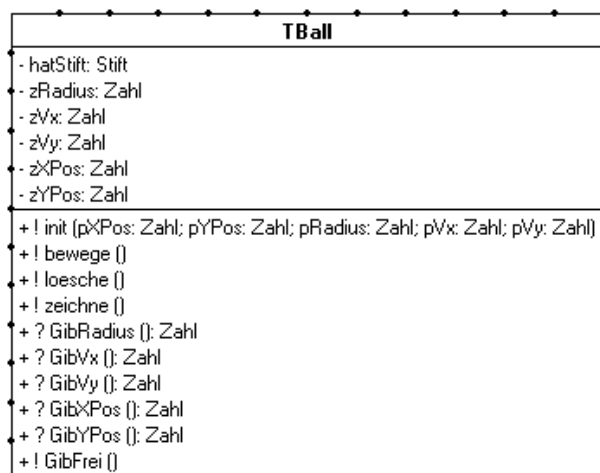
Nur die Dokumentation sollte auch hier nicht vergessen werden! Also im Feld ‚Dokumentation‘ einen kleinen Eintrag vornehmen !!

Am Schluss muss der Methodendialog natürlich über den Button ‚OK‘ verlassen werden.

Wenn Sie auf die gleiche Art und Weise die Aufträge ‚zeichne‘, ‚loesche‘ und ‚bewege‘ sowie den Destruktor ‚gibFrei‘ einfügen, ergibt sich das nebenstehende Klassendiagramm.

Beachten Sie die etwas ungewöhnliche Parameterliste, bei der jeder Parameter einzeln aufgeführt ist. Dies dient der Kompatibilität mit anderen Sprachen wie z.B. Java.

Im Moment sind alle Methoden dieser Klasse 'public' also öffentlich.

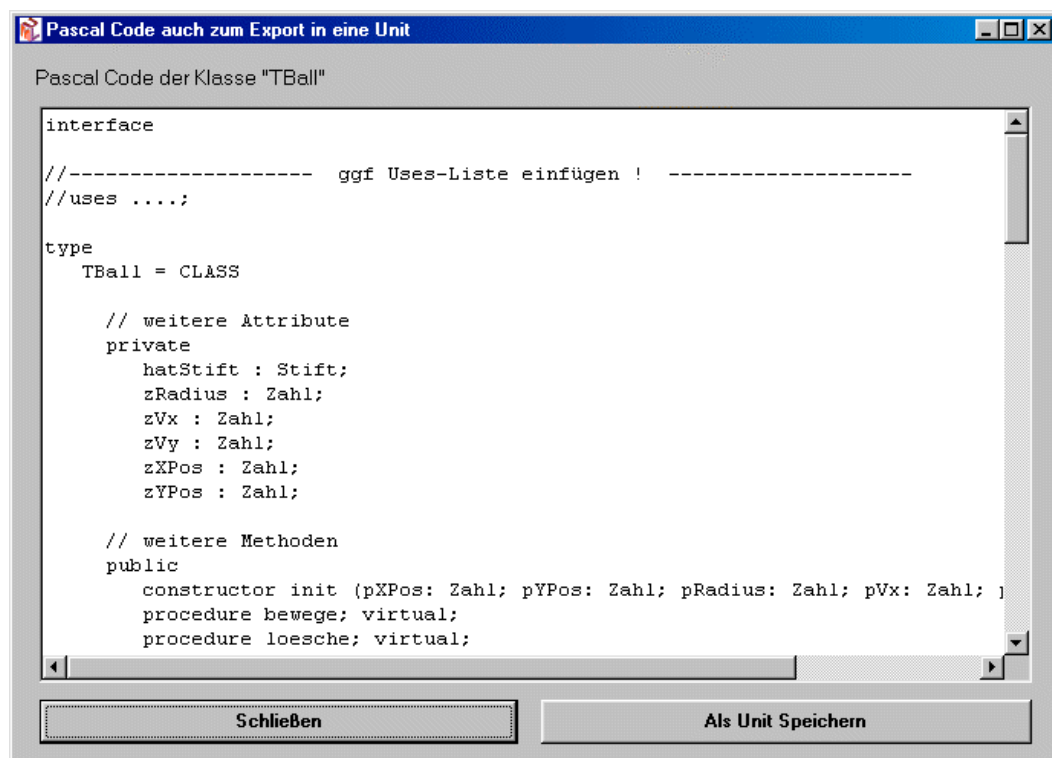
1.5. Klasse speichern und Code erzeugen

Wichtig: Für die weiteren Arbeiten gehen wir wie vorne gesagt davon aus, dass wir schon gewisse Vorarbeiten geleistet haben. Wir haben schon ein Projektverzeichnis für das Delphi-Projekt angelegt. Es hatte den Namen ,TischTennis1‘.

Nun sind zwei Dinge zu erledigen: Erstens sollte das Diagramm als Beginn einer Dokumentation gespeichert werden. Und zweitens möchten wir, dass **UMLed** uns auch bei der Programmierung Arbeit abnimmt.

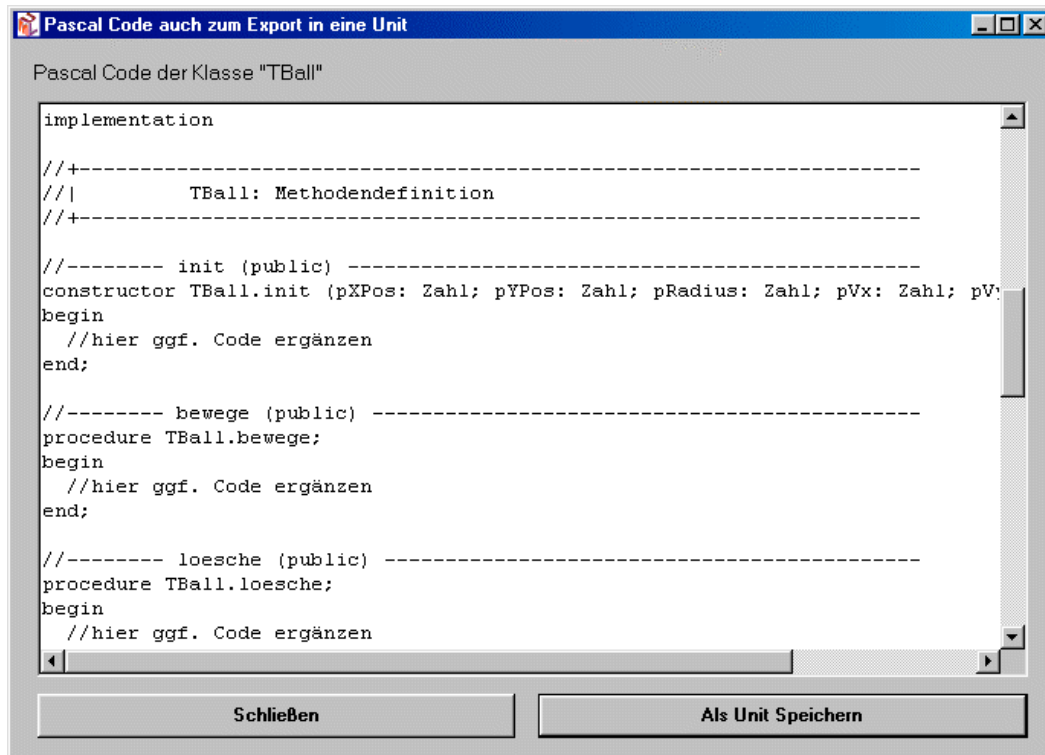
Diagramm speichern: Wir wählen in **UMLed** das Menü ,Datei‘ und daraus den Unterpunkt ,Speichern unter ...‘. Im nun folgenden üblichen Dateidialog kämpfen Sie sich zuerst zum Delphi-Projektverzeichnis durch. Notfalls könnten Sie es natürlich auch jetzt erst anlegen. In diesem speichern Sie nun das Diagramm unter dem Namen ,TischTennis1‘ ab. Das Diagramm erhält automatisch die Endung '.urd', so dass die Datei 'TischTennis1.urd' heißt.

Pascal-(bzw. Delphi)-Code erzeugen: Um einen Teil des benötigten Pascal-Codes erzeugen zu lassen, wählen Sie den Unterpunkt ,Delphi-Export‘ aus dem Menü ,Klasse‘ oder dem Klassen-Kontextmenü. Sie erhalten dann folgendes Fenster:



Alle Methoden sind als überschreibbar (virtual) deklariert. Soll eine Methode nicht überschreibbar sein, muss dies im Methodenfenster extra gekennzeichnet werden.

Scrollen Sie den Ausschnitt nach unten, dann erkennen Sie, dass auch der Rahmen des Implementationsteils vorhanden ist:



Um den Pascalcode in dieser Form zu speichern, klicken Sie auf ‚Als Unit Speichern‘. Wählen Sie als Speicherort wieder das Verzeichnis 'TischTennis1', das Sie für das Delphi-Projekt ja schon angelegt haben. **UMLed** schlägt Ihnen als Name: ‚mTKugel.pas‘ vor. ‚m‘ steht dabei für **M**odul. Behalten Sie den Namen bei und speichern Sie den Pascal-Code ab.

Jetzt haben wir den ersten Teil der Arbeit mit **UMLed** erledigt.

Was haben Sie gelernt?

- Öffnen eines vorhandenen UML-Diagrammes
- Benutzen der ersten beiden Kontextmenüs (auf leerer Fläche und in der Klasse)
- Anlegen einer neuen Klasse
- Ändern der Beschreibung der Klasse
- Hinzufügen von Attributen und von Methoden zu dieser Klasse
- Abspeichern der Klasse
- Erzeugen des Pascal-Codes und Speichern des Codes.

Wechseln Sie jetzt zur Delphi-Entwicklungsumgebung und bearbeiten den Code dort weiter. Lassen Sie **UMLed** bitte geöffnet, denn wir werden mit diesem Programm noch weiter arbeiten.

1.6. Einbinden und Ergänzen des Codes unter Delphi

Wechseln Sie wie gesagt zur Delphi-Entwicklungsumgebung. Öffnen Sie das Projekt Projektdatei.dpr im Verzeichnis TischTennis1.

Als erstes müssen Sie die soeben mit UMLed erzeugte Unit mTBall dem Projekt hinzufügen. Dies erfolgt bei allen Delphi Versionen über den Menüpunkt 'Projekt', Unterpunkt 'Dem Projekt hinzufügen'.

Nun müssen Sie den Code der verschiedenen Units ergänzen.

Bearbeiten Sie zuerst die gerade geöffnete Unit mTBall. Ergänzen Sie als erstes die ,uses Liste' wie folgt:

```
uses mSuM;
```

und dann den ,implementation-Teil' - und nur diesen !! - um die grünen Zeilen:

```
implementation
```

```
//+-----  
//|          TBall: Methodendefinition  
//+-----
```

```
//----- init (public) -----  
constructor TBall.init(pXpos, pYPos, pRadius, pVx, pVy : Zahl);  
begin  
    hatStift := Stift.init;  
    zXPos := pXpos;  
    zYPos := pYPos;  
    zRadius := pRadius;  
    zVx := pVx;  
    zVy := pVy;  
end;
```

```
//----- bewege (public) -----  
procedure TBall.bewege;  
begin  
    loesche;  
    zXPos := zXpos + zVx;  
    zYPos := zYPos + zVy;  
    zeichne  
end;
```

```
//----- loesche (public) -----  
procedure TBall.loesche;  
begin  
    hatStift.radiere;  
    hatStift.hoch;  
    hatStift.bewegeBis(zXPos, zYPos);  
    hatStift.zeichneKreis(zRadius);  
    hatStift.runter  
end;
```

```
//----- zeichne (public) -----
procedure TBall.zeichne;
begin
    hatStift.normal;
    hatStift.hoch;
    hatStift.bewegeBis(zXPos, zYPos);
    hatStift.zeichneKreis(zRadius);
    hatStift.runter
end;

//----- liesRadius (public) -----
function TBall.GibRadius : Zahl;
begin
    result := zRadius
end;

//----- GibVx (public) -----
function TBall.GibVx : Zahl;
begin
    result := zVx
end;

//----- GibVy (public) -----
function TBall.GibVy : Zahl;
begin
    result := zVy
end;

//----- GibXPos (public) -----
function TBall.GibXPos : Zahl;
begin
    result := zXPos
end;

//----- GibYPos (public) -----
function TBall.GibYPos : Zahl;
begin
    result := zYPos
end;

//----- gibFrei (public) -----
destructor TBall.gibFrei;
begin
    hatStift.gibFrei
end;

end.
```

Öffnen Sie dann die Unit mTAnwendung. Ergänzen Sie zuerst die Uses-Liste um den grünen Eintrag:

```
uses mSuM, mTBall;
```


Ergänzen Sie nun den restlichen Code der Anwendungsklasse um die grünen Zeilen:

```
type
    TAnwendung = CLASS

    // weitere Attribute
private
    derBildschirm : Bildschirm;
    dieMaus : Maus;
    meinStift : Stift;
    dieTastatur : Tastatur;
    hatBall : TBall;

    // weitere Methoden
public
    constructor erzeuge;
    procedure gibFrei;
    procedure laufeAb;

end;

implementation

//+-----
//|          TAnwendung: Methodendefinition
//+-----

//----- erzeuge (public) -----
constructor TAnwendung.erzeuge;
begin
    derBildschirm := Bildschirm.init;
    meinStift := Stift.init;
    dieTastatur := Tastatur.init;
    dieMaus := Maus.init;
    hatBall := TBall.init(50, 50, 20, 0.02, 0.01)
end;

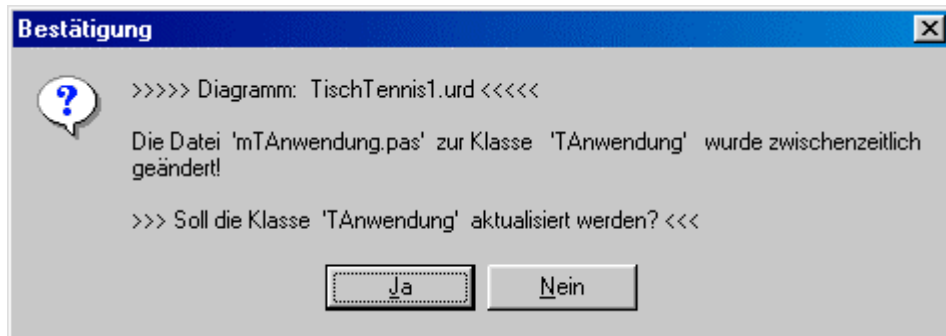
//----- gibFrei (public) -----
procedure TAnwendung.gibFrei;
begin
    hatBall.gibFrei;
    meinStift.gibFrei;
    dieMaus.gibFrei;
    dieTastatur.gibFrei;
    derBildschirm.gibFrei;
end;

//----- laufeAb (public) -----
procedure TAnwendung.laufeAb;
begin
    repeat
        hatBall.bewege
    until dieMaus.istGedrueckt
end;
end.
```

Nun sollte Ihr Delphi-Projekt funktionieren. – Aber sicherlich haben Sie irgendwo ein Semikolon vergessen! Oder etwa nicht??

Aktualisieren des Klassendiagramms:

Spätestens, wenn Ihr Projekt fehlerfrei läuft, sollten Sie das Projekt und sämtliche Units speichern! Wechseln Sie nun zu **UMLed**. Es erscheint folgende Meldung:



Bestätigen Sie die Frage mit Ja, dann werden sämtliche Änderungen der Unit auch in das Klassendiagramm übernommen. Die gleiche Meldung erhalten Sie für die Klasse `TBall`, die sie natürlich auch aktualisieren sollten. Sie erkennen, dass die Klasse `TAnwendung` nun auch ein Attribut `hatBall` besitzt.

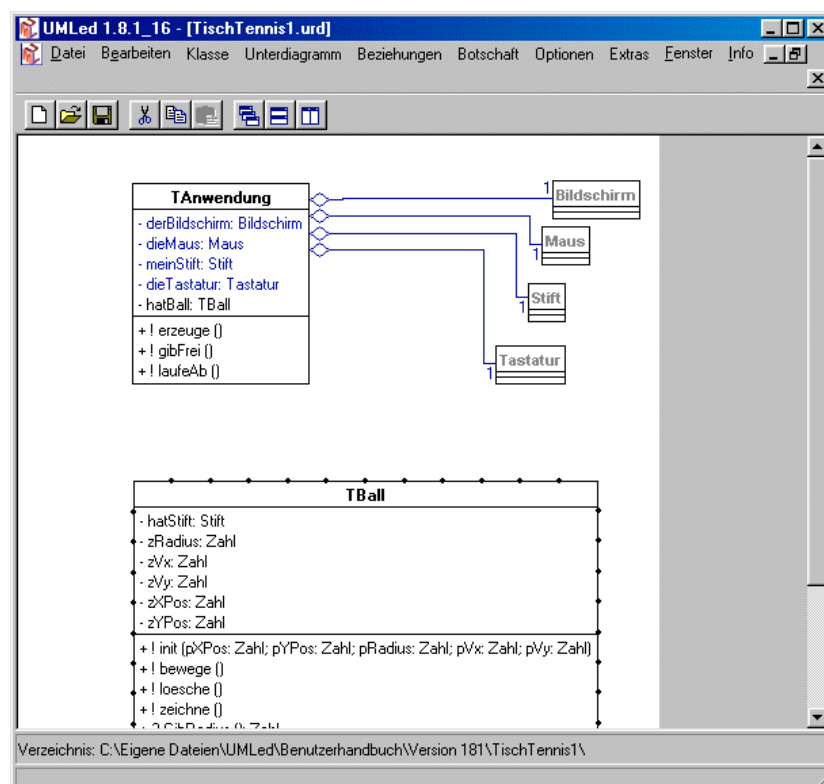
Damit ist der erste Teil unserer Arbeit erledigt. Sie können **UMLed** nun nutzen, um sich beim Entwurf eigener Klassen und deren Pascal-Codierung helfen zu lassen.

Was folgt als nächstes?

- Ein Klassendiagramm der beteiligten Klassen
- Das Hinzufügen weiterer Attribute und Methoden zur Klasse `TBall`.

2. Das erste Beziehungsdiagramm

Wenn Sie die Arbeitsfläche von **UMLed** maximieren, sollte Ihr UML-Diagramm innerhalb von UMLed nun in etwa so aussehen:



Wir wollen nun erreichen, dass der Ball von den Rändern des Bildschirms reflektiert wird. Hierbei wird uns etwas später das Beziehungsdiagramm helfen!

Vorbereitend verdeutlichen wir im Diagramm, dass die Anwendung einen Ball besitzt. Es handelt sich hierbei um eine Hat-Beziehung, so wie sie die Anwendung schon zu Bildschirm, Maus, Stift und Tastatur besitzt. Näheres zu Beziehungen siehe unter <http://www.learn-line.de/angebote/oop/index.html> oder im [Anhang](#).

2.1. Erstellen einer Beziehung

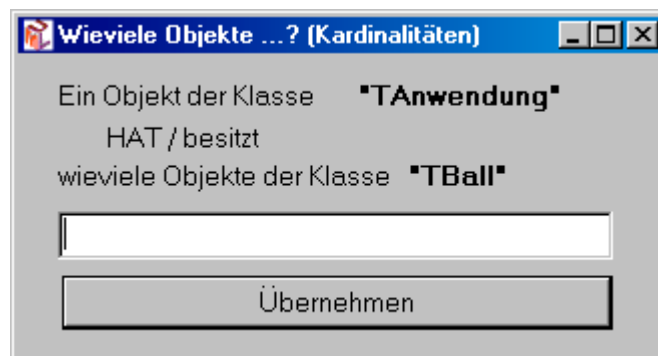
Zum Erstellen einer Beziehung zwischen Objekten (nicht zwischen den Klassen selbst) wählen Sie den Menüpunkt ‚Beziehungen‘ oder klicken Sie auf einen freien Teil der Arbeitsfläche mit der rechten Maustaste. In beiden Fällen erhalten Sie ein Menü mit dem Unterpunkt ‚HAT . Aggregation‘.



Das Erscheinungsbild der Klassen auf der Arbeitsfläche ändert sich nun. Sie erkennen lauter schwarze Punkte auf ihren Rändern.

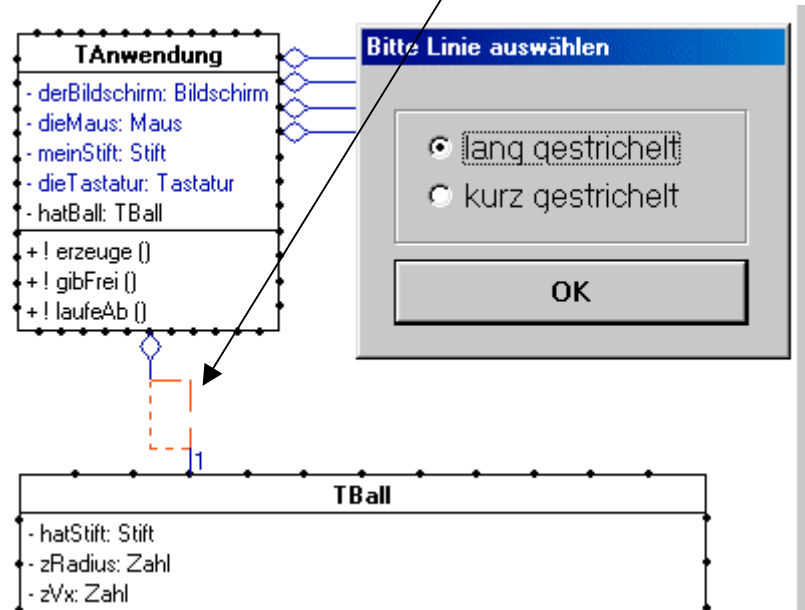
Außerdem steht unter der Arbeitsfläche ein wichtiger Hinweis: *‚Bitte Randpunkt der besitzenden Klasse mit Handsymbol anklicken!‘*

Bewegen Sie die Maus zu einem Randpunkt der Klasse TAnwendung (am besten unten in der Mitte), dann ändert sich der Mauszeiger in eine Hand. Wenn Sie nun die linke Maustaste drücken, ist dieser Punkt als Ausgangspunkt und die Klasse ‚TAnwendung‘ als Ausgangsklasse registriert. Es erscheint als weiterer Hinweis am unteren Rand von UMLed: *‚Bitte Randpunkt der zweiten Klasse mit Handsymbol anklicken‘*. Klicken Sie nun einen geeignet liegenden Punkt der Klasse ‚TBall‘ an. Damit ist festgelegt, dass Objekte der Klasse ‚TAnwendung‘ ein oder mehrere Objekte der Klasse ‚TBall‘ besitzen. Nun fragt **UMLed** Sie in einem Fenster,



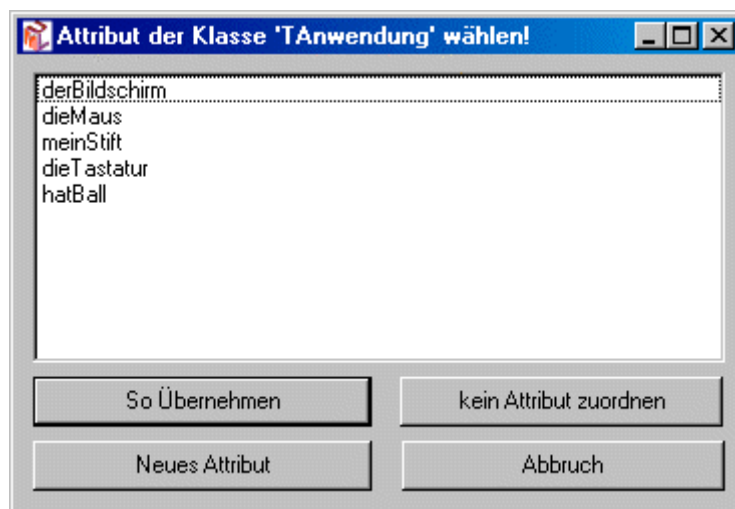
wie viele ‚Bälle‘ eine ‚Anwendung‘ besitzt. Geben Sie die Ziffer 1 ein und klicken Sie auf ‚übernehmen‘.

Es werden Ihnen nun zwei mögliche Verbindungslinien **in Rot** angeboten:



Wählen Sie einen der beiden Verbindungswege aus.

Nun folgt eine weitere wesentliche Frage von UMLed:

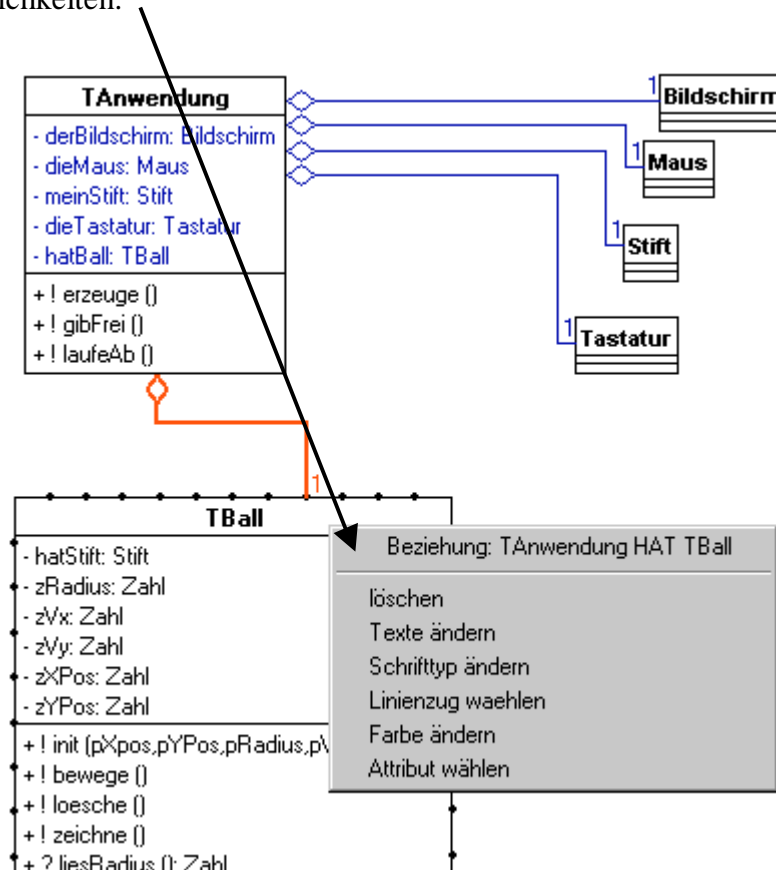


Zu jeder HAT- oder KENNT-Beziehung gehört ein Attribut sozusagen als Gedächtnis des Objektes. Wir haben hierfür schon ein Attribut vorgesehen, nämlich 'hatBall'. Klicken Sie daher dieses Attribut an und schließen Sie das Fenster mit 'so übernehmen'.

Sie erkennen, dass die Beziehung mit der kleinen Raute gezeichnet wurde, dass am Ballende eine kleine '1' steht, und dass das zugehörige Attribut 'hatBall' nun auch die Farbe der Beziehung, nämlich blau erhalten hat.

2.2. Bearbeiten von Beziehungen

Um Beziehungen zu bearbeiten müssen Sie auf einen Randpunkt der Beziehung im Regelfall mit der rechten Maustaste klicken. Als Hilfe wandelt sich der Mauszeiger in eine Hand, wenn Sie sich über dem Randpunkt einer Beziehung befinden. Nach dem Klick mit der rechten Maustaste wird die Beziehung rot dargestellt und es öffnet sich ein Kontextmenü mit folgenden Möglichkeiten:



Beziehungen löschen:

klar: Die Beziehung wird nach einer Sicherheitsabfrage gelöscht.

Texte ändern:

Hiermit können Sie die Texte an den Enden der Beziehung ändern.

Schrifttyp ändern:

Hiermit können Sie den Schrifttyp für die Texte am Ende der Beziehung ändern.

Linienzug wählen:

Das haben Sie schon getan. Sie können hiermit wiederum zwischen den beiden Verbindungswegen für eine Beziehung wählen.

Farbe ändern:

Dies ist eine interessante Option. Sie ändern damit die Farbe der Linie, der Schrift und auch des zugehörigen Attributes in der besitzenden Klasse. Probieren Sie es aus und geben der Beziehung ein kräftigeres Pink !!!

Attribut wählen:

Auch dies haben Sie schon durchgeführt. Sie können hiermit nachträglich ein anderes Attribut wählen oder auch ein neues anlegen lassen.

Anfangs und Endpunkt von Beziehungen verschieben:

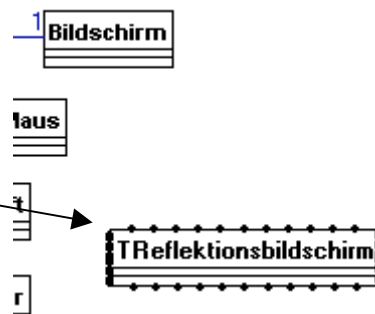
Diese Möglichkeit ist nicht über das Kontextmenü zu erreichen. Sie müssen vielmehr einen Endpunkt der Beziehung mit der linken Maustaste anklicken. Dann erscheint die Beziehung und die Randpunkte der zugehörigen Klasse in rot. Wenn Sie nun einen der roten Randpunkte anklicken, wird dieser zum Verbindungspunkt der Beziehung.

2.3. Eine wirklich neue Beziehung:

Wir wenden uns nun dem Problem der Reflexion an den Wänden zu. Es stellt sich wie immer die Frage: "Wer ist für das Reflektieren eines Balles verantwortlich?" Die Anwendung (zentrale Lösung), der Ball (liegt recht nahe) oder die Ränder des Bildschirms, also der Bildschirm selbst? Die letzte Lösung entspricht dem Sprachgebrauch 'Der Ball wird von der Wand reflektiert' und wir entscheiden uns daher für diese Lösung !!

Wir benötigen also eine neue Fähigkeit für den Bildschirm, nämlich die, einen Ball zu reflektieren. Da die Standard-Bildschirme der Stift und Maus- Umgebung dies nicht können, benötigen wir eine neue Klasse. Nennen wir sie 'TReflektionsbildschirm'. Sie ist natürlich Nachfolgeklasse der Klasse TBildschirm.

Die ersten Schritt müssten Sie beherrschen: Erstellen Sie eine neue Klasse mit dem Namen TReflektionsbildschirm und schieben Sie sie unter die Klasse TBildschirm:



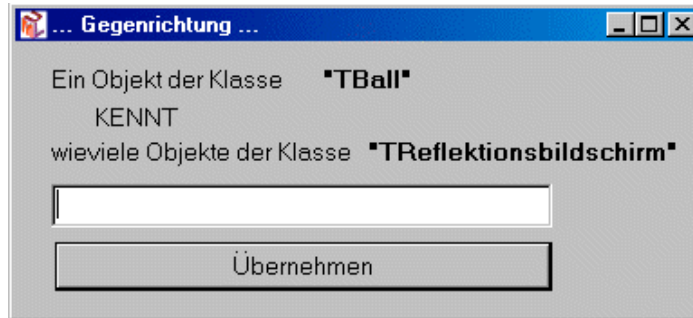
Als nächstes erstellen Sie eine IST-Beziehung zwischen der Klasse TReflektionsbildschirm und der Klasse Bildschirm.. (Menüpunkt Beziehungen, IST-Beziehung, Randpunkte wählen und dabei auf die Hilfetexte am unteren Rand der Arbeitsfläche achten). Sie erhalten dann einen roten Pfeil als Kennzeichen dieser Beziehung.

Nun wird es spannender. Um einen Ball reflektieren zu können, muss ein Reflektionsbildschirm den Ball kennen! Erstellen Sie also eine KENNT-Beziehung zwischen der Klasse TReflektionsbildschirm (als erstem, also kennendem Objekt) und der Klasse TBall (als zweitem, also gekanntem Objekt).

Sie beantworten die Frage, wie viele Bälle ein Reflektionsbildschirm kennt, natürlich mit '1'.



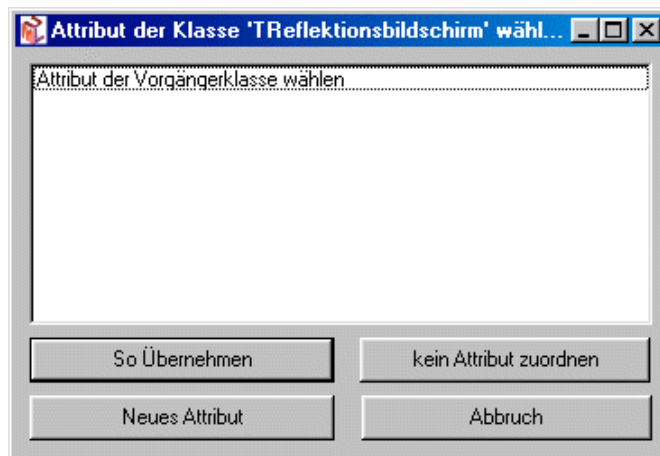
Dann erhalten Sie allerdings eine weitere Frage:



Ein Objekt der Klasse **"TBall"**
KENNT
wieviele Objekte der Klasse **"TReflektionsbildschirm"**

Da der Ball aber den Bildschirm nicht kennen muss, lassen Sie das Eingabefeld frei und drücken nur auf 'übernehmen'.

Nach der Wahl des Weges, kommt die Frage nach dem zugehörigen Attribut:

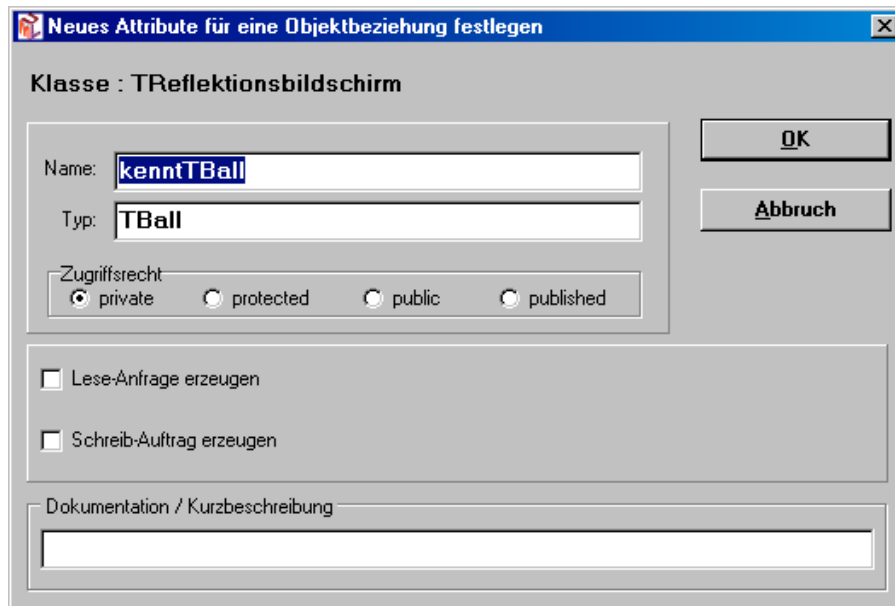


Attribut der Vorgängerklasse wählen

So Übernehmen	kein Attribut zuordnen
Neues Attribut	Abbruch

Da der Reflektionsbildschirm kein passendes Attribut besitzt (er besitzt ja bisher gar kein Attribut) klicken Sie auf 'Neues Attribut'! Sie gelangen in den Attribute-Dialog mit schon (halbwegs) passenden Vorschlägen:

ändern Sie den Namen von 'kenntTBall' in 'kenntBall' und klicken Sie auf OK.



Klasse : TReflektionsbildschirm

Name:

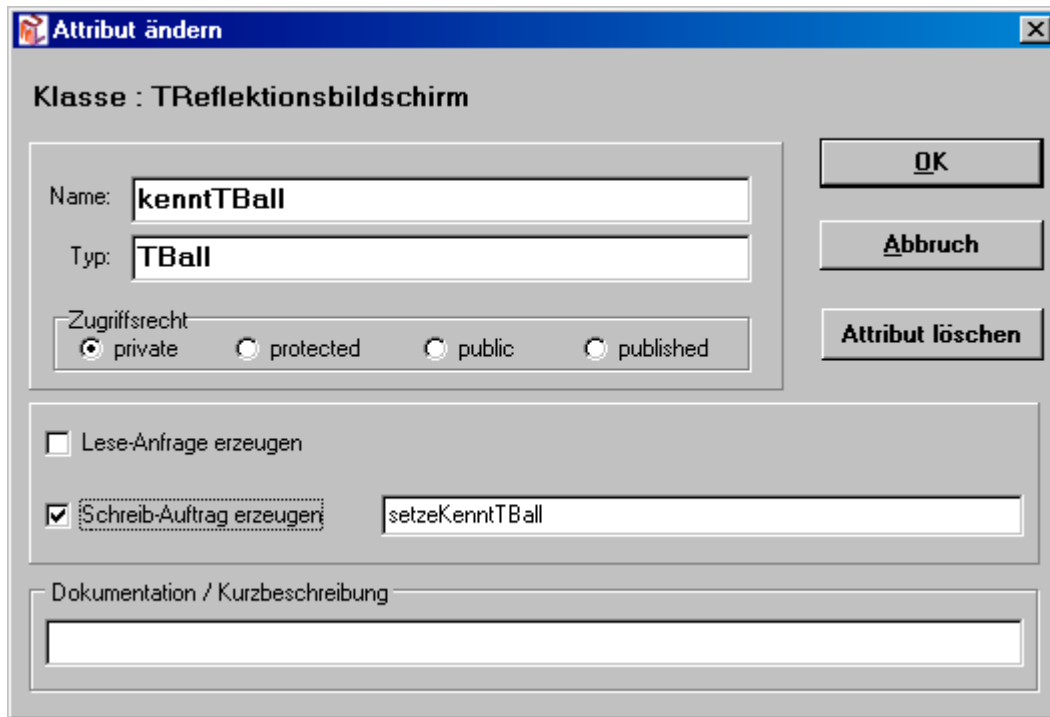
Typ:

Zugriffsrecht
☒ private ☐ protected ☐ public ☐ published

☐ Lese-Anfrage erzeugen
☐ Schreib-Auftrag erzeugen

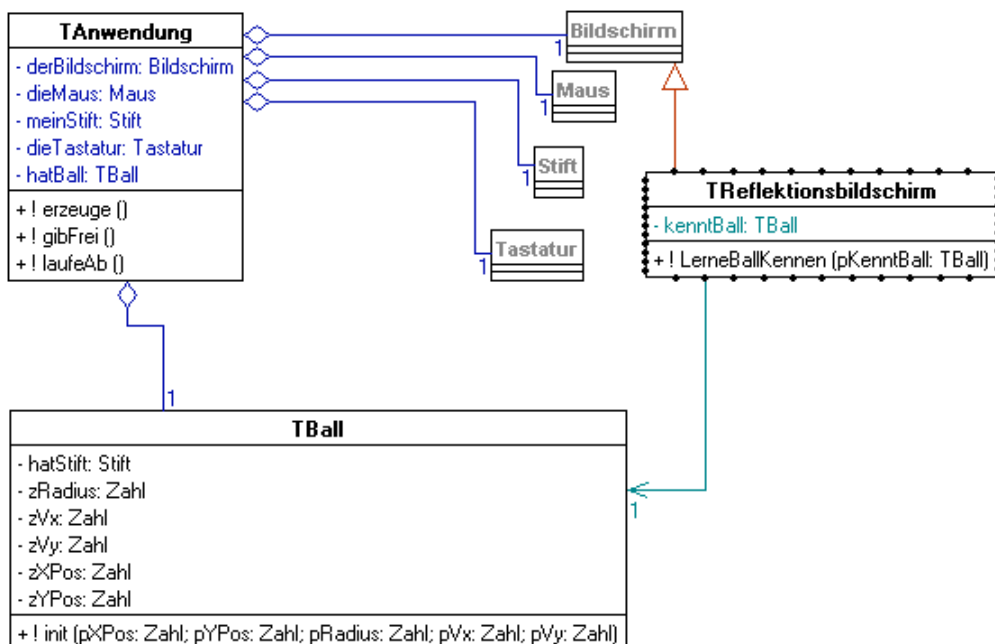
Dokumentation / Kurzbeschreibung

Natürlich muss der Bildschirm den Ball kennen lernen, bevor er ihn ggf. reflektieren kann. Wir kreuzen daher im Attribute-Dialog das Häkchen für 'Schreib-Auftrag erzeugen' an.



Der Name 'setzeKenntBall' gefällt Ihnen hoffentlich nicht. Ersetzen Sie ihn durch 'lerneBallKennen' und schliessen Sie den Dialog mit OK. Dann ist eine entsprechende Methode erzeugt worden, die sogar schon über den passenden Delphi-Code verfügt.

DAS NUN ENTSTANDENE DIAGRAMM ZEIGT DAS FOLGENDE BILD:



Machen Sie sich noch einmal die drei Schritte klar, die hier beim Herstellen einer Kennt Beziehung durchgeführt worden sind:

1. Erstellen der Beziehung (klar)
2. Neu Anlegen eines Attributes bei der kennenden Klasse.
Möglichkeit wird bei jeder Beziehung angeboten
3. Neu Anlegen eines Schreib-Auftrages für dieses neue Attribut.
Möglichkeit wird bei jedem Anlegen eines Attributes angeboten.

3. Das erste Botschaftsdiagramm

Botschaftsdiagramme - in UML eigentlich Kollaborationsdiagramme genannt – dienen dazu, den zeitlichen Ablauf von Aufträgen und Anfragen bei speziellen Situationen zu verdeutlichen. Hierbei können mit Hilfe von **UMLed** die beteiligten Klassen weiter entwickelt werden.

Ihnen ist sicher aufgefallen, dass wir den Delphi-Code der neue Klasse TReflektionsbildschirm noch nicht erzeugt und gespeichert haben. Dies hat seinen Sinn, denn der Entwurf dieser Klasse wird im folgenden noch weiter vervollständigt werden.

Wir analysieren dazu den Botschaftsaustausch zwischen den beteiligten Objekten bei der Reflektion eines Balles durch den Rand des Bildschirms.

3.1. Erstellen eines neuen Unterdiagramms.

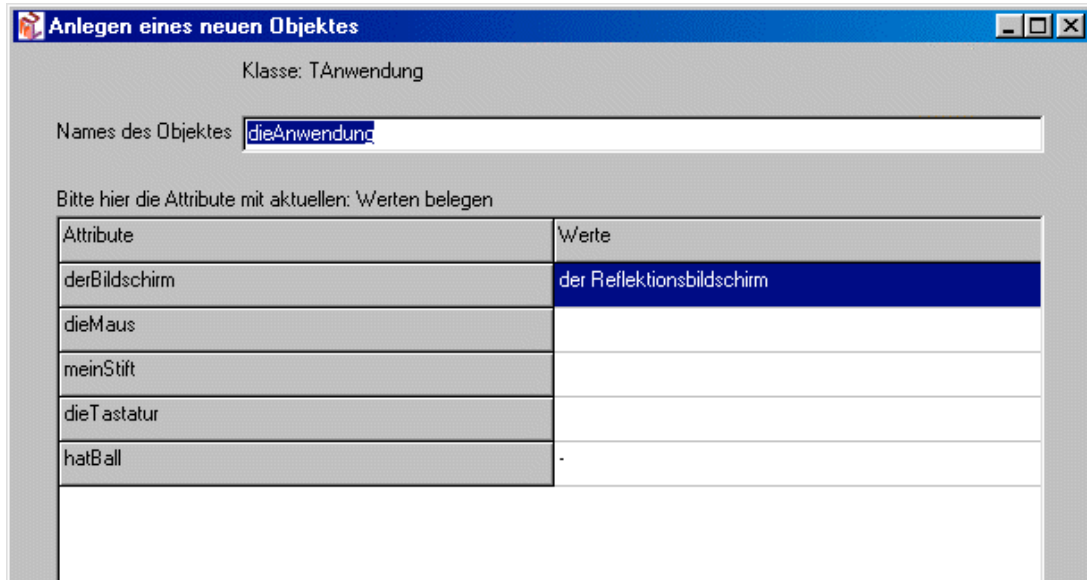
Hierfür erstellen wir eines neues Diagramm, das mit Objekten der bisher entwickelten Klassen arbeiten soll. Wählen Sie den Menüpunkt 'Unterdiagramm' von UMLed, dann den Unterpunkt 'Neues Unterdiagramm' und geben Sie als Namen 'Reflektion eines Balles durch den Rand des Bildschirms' ein.

Es entsteht eine neue noch leere Arbeitsfläche in UMLed. Übrigens ohne den Menüpunkt 'Unterdiagramm'.

Hierhinein müssen Sie nun als erstes die beteiligten Klassen (Objekte) aus dem ursprünglichen Diagramm einfügen. Dies geschieht durch den Menüpunkt Objekt/Klasse oder durch Rechtsklick auf die freie Arbeitsfläche. Im letzteren Fall erscheint das nebenstehende Kontextmenü, aus dem sie ‚Objekt einfügen‘ wählen.



Wählen Sie aus den angebotenen Klassen für das Objekt die Klasse TAnwendung. Es erscheint der Dialog zum Anlegen eines Objektes. Ändern Sie den vorgeschlagenen Namen in ‚dieAnwendung‘. Klicken Sie mit der Maus in das Feld neben dem Attribut ‚derBildschirm‘ und tragen Sie ‚der Reflektionsbildschirm‘ ein. Löschen Sie danach die Bindestriche in den Eingabefeldern neben ‚dieMaus‘, ‚meinStift‘ und ‚dieTastatur‘. Beenden Sie Ihre Eingaben mit dem Knopf ‚so übernehmen‘.

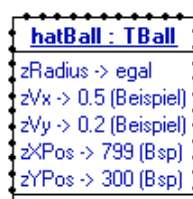


Attribute	Werte
derBildschirm	der Reflektionsbildschirm
dieMaus	
meinStift	
dieTastatur	
hatBall	-

Sie erhalten auf der Arbeitsfläche folgendes Objekt:



Sie erkennen, dass im Diagramm nur die Attribute eines Objektes angezeigt, die mit Werten belegt wurden. Verfahren Sie ebenso, um ein Objekt der Klasse ‚TReflektionsbildschirm‘ und ein weiteres der Klasse ‚TBall‘ einzufügen..



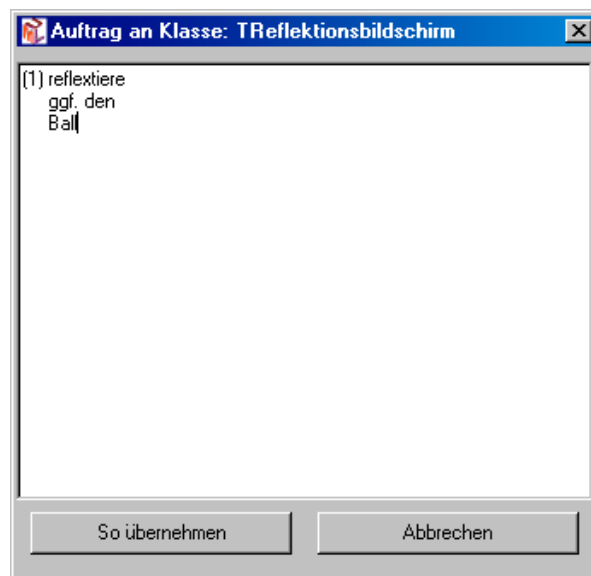
3.2. Hinzufügen eines Auftrages

Im Rumpf der Arbeitsschleife der Anwendung muss stets zuerst der Ball den Auftrag erhalten, sich zu bewegen und dann der Reflexionsbildschirm den Auftrag erhalten, gegebenenfalls den Ball zu reflektieren. Wir beschäftigen uns im Folgenden nur mit dieser Relektion.

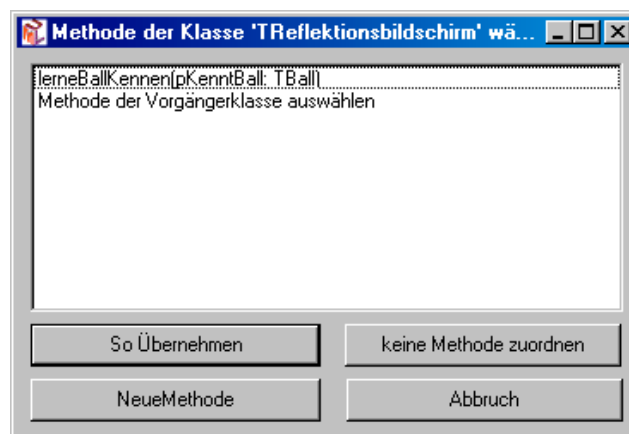
Die erste Botschaft ist also von der Anwendung an den Reflexionsbildschirm gerichtet und lautet einfach: 'Reflektiere ggf. den Ball'. Wie realisieren wir dies mit **UMLed**?

Wählen Sie den Menüpunkt 'Botschaft, Auftrag' oder machen Sie einen Klick mit der rechten Maustaste in die freie Arbeitsfläche des Unterdigramms und wählen Sie aus dem Kontextmenü den Punkt 'Auftrag' aus. Wiederum werden die Randpunkte der Klassen sichtbar. Klicken Sie zuerst auf einen Randpunkt (rechts mittig) der Anwendung und dann auf einen Randpunkt (links) des Reflexionsbildschirmes. Es erscheint drauf eine Eingabebox zur Formulierung des Auftrages.

Füllen Sie den Text wie angegeben aus und übernehmen Sie ihn.



Es folgt nun die Nachfrage nach einer passenden Methode der beauftragten Klasse. :

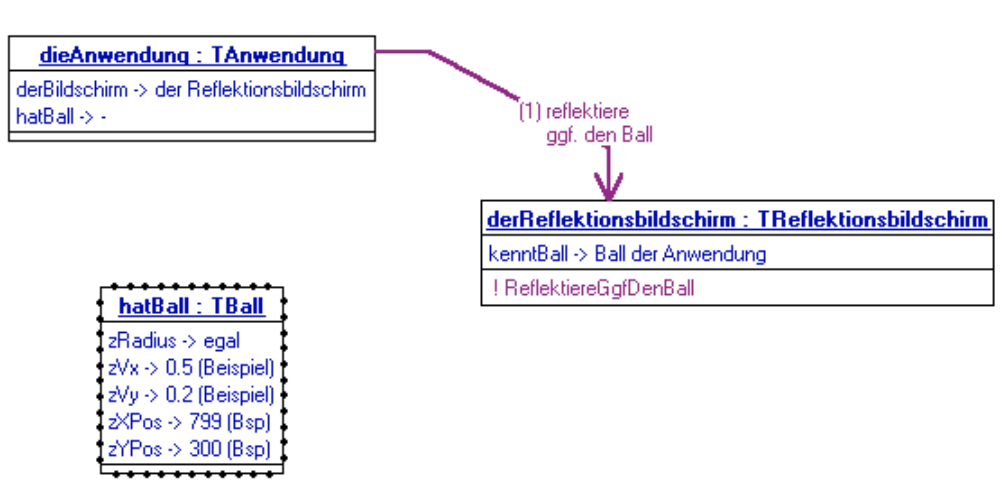


Da die Klasse TRelektionsbildschirm noch keine eigene Methode besitzt, um diesen Auftrag erfüllen zu können, klicken Sie auf 'neue Methode'

Sie gelangen in den Methodendialog, bei dem die erste Zeile des vorher formulierten Auftrages als Methodenname steht. Ändern Sie diese Zeile sinnvoll, z.B. in 'reflektiereGgfDenBall' und schließen Sie mit 'ÖK'.



Im Diagramm ist nun eine Botschaftspfeil zu erkennen (nicht zu verwechseln mit einer IST-Beziehung !!) und die entsprechende neue Methode der Klasse TReflektionsBildschirm:



3.3. Hinzufügen einer Anfrage

Um entscheiden zu können, ob der Ball reflektiert werden muss, benötigt der ReflektionsBildschirm Informationen über die horizontale und vertikale Position des Balles sowie über seinen Radius.

Wählen Sie den Punkt 'ANFRAGE' aus dem Menü 'Botschaft' oder dem Kontextmenü der freien Zeichenfläche. Es erscheinen die bekannten Punkte an allen Klassen. Wählen Sie zuerst

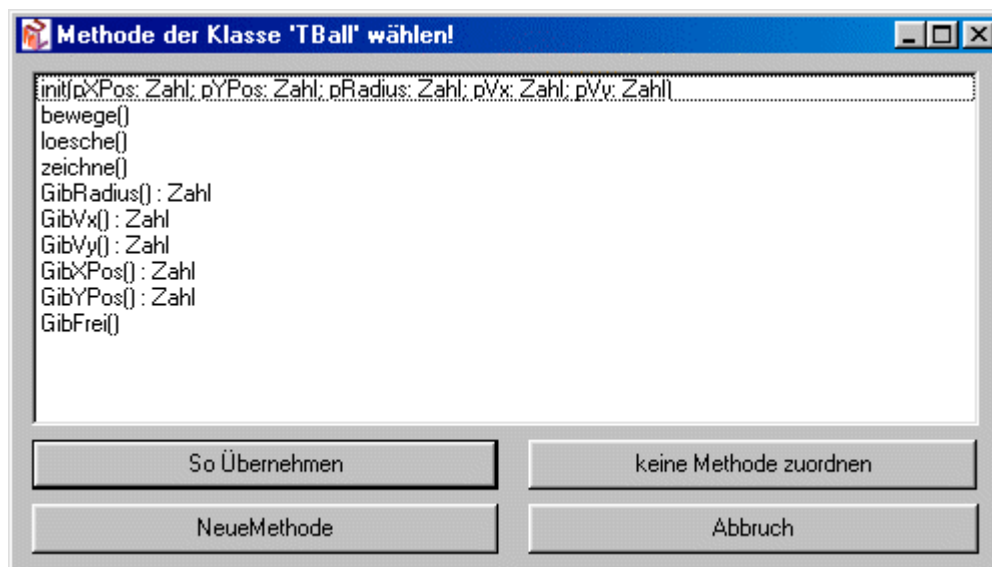
einen passend liegenden Punkt der fragenden Klasse, also des ReflektionsBildschirms und dann einen passend liegenden Punkt der antwortenden Klasse, also des Balles.
Es erscheint eine Eingabebox für die Anfrage:



In welche Sie 'Wie lautet Deine horizontale Position' eingeben können. Als Datentyp der Antwort geben Sie dann 'Zahl' ein.

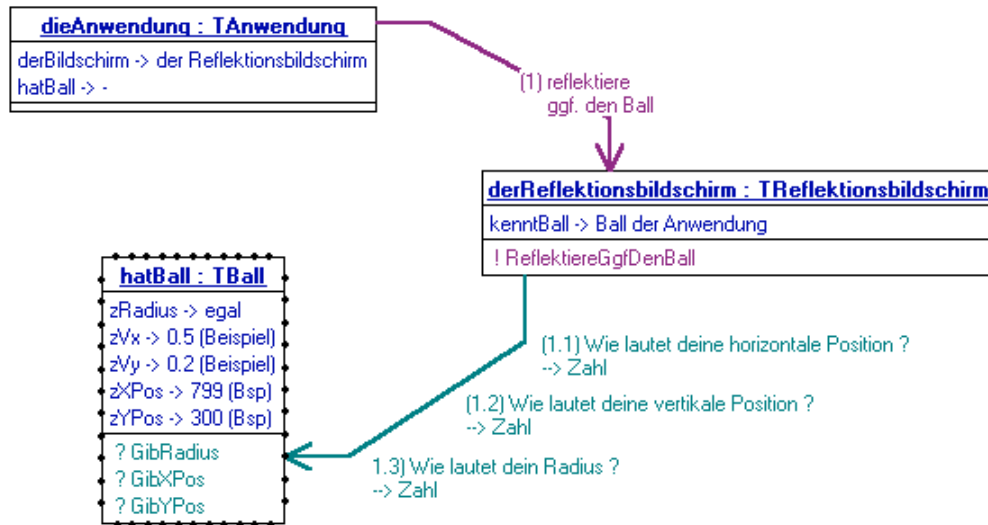


In der nun folgenden Methodenauswahl finden Sie schon eine geeignete Methode, nämlich GibXPos, die Sie anklicken und dann ,so übernehmen‘.



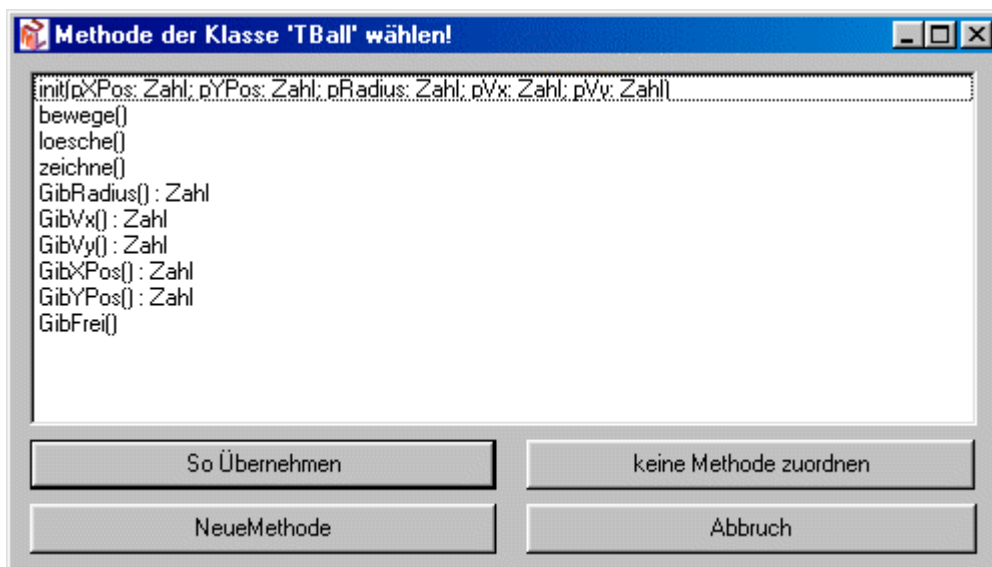
Verfahren Sie genau so mit den Anfragen für die vertikale Position und für den Radius. Wenn Sie für alle drei Anfragen den selben Anfangs- und Endpunkt wählen, bleibt Ihr Diagramm sehr übersichtlich. Sie müssen allerdings die Anfragetexte mit gedrückter linker Maustaste an die passenden Stellen ziehen.

Damit ergibt sich folgender Zwischenstand für das Diagramm:



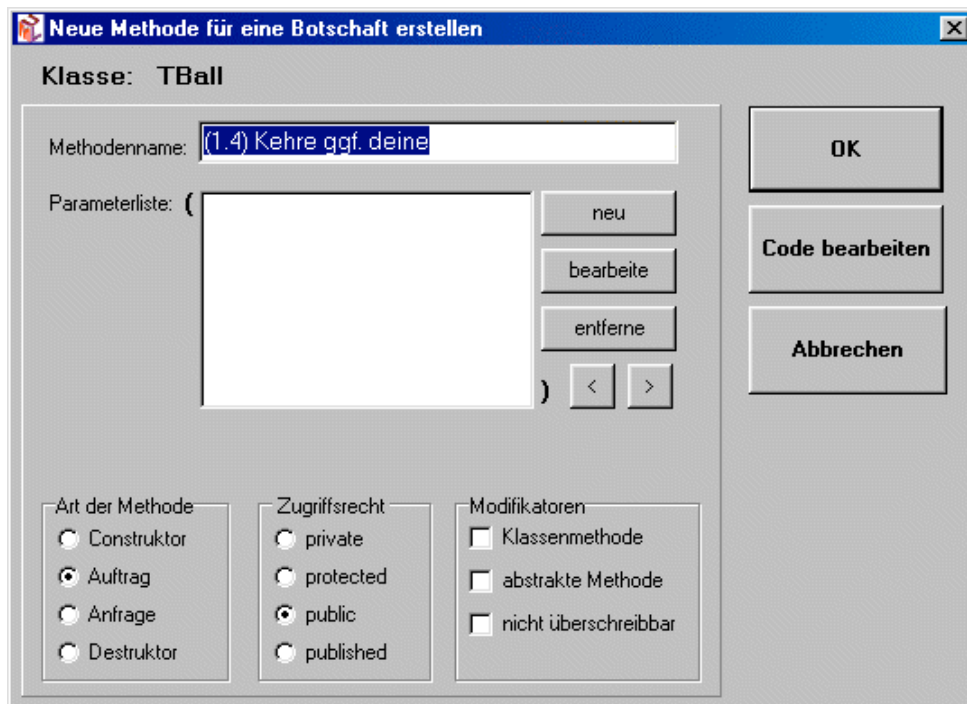
Wir vervollständigen nun das Diagramm weiter:

Je nach den Angaben, welche der Ball geliefert hat, kann es sein, dass der RelektionsBildschirm den Ball beauftragt seine Richtung zu ändern. Er erteilt dann zB. einen Auftrag : 'Kehre Deine horizontale Geschwindigkeit' um. Erstellen Sie also einen solchen Auftrag. Sie werden dann an entsprechender Stelle wiederum nach einer Methode der Klasse TBall gefragt und bemerken, dass kein passender Auftrag vorhanden ist.



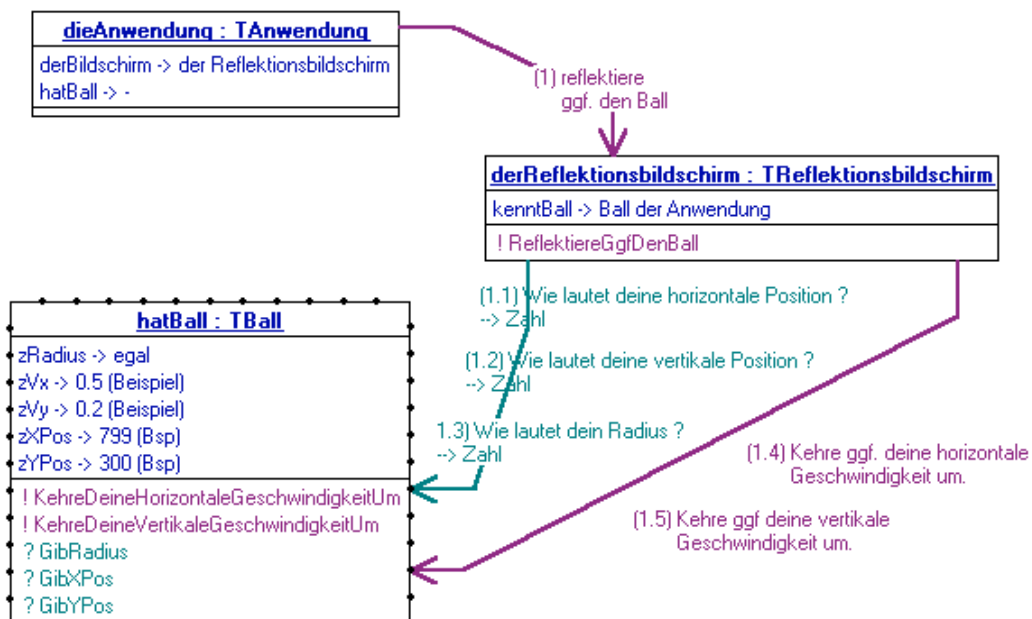
Also wählen Sie ‚Neue Methode‘.

Im folgenden Methodendialog ändern Sie den vorgeschlagenen Namen in ‚KehreDeineHorizontaleGeschwindigkeitUm‘.



Verfahren Sie nun ebenso mit dem Auftrag 'Kehre Deine vertikale Geschwindigkeit um!'

Wenn Sie mit der rechten Maustaste in die Texte der Botschaften klicken, können sie unter anderem die Texte noch ein wenig anpassen. So lässt sich sogar die zeitliche Reihenfolge der Aufträge und Anfragen festhalten:



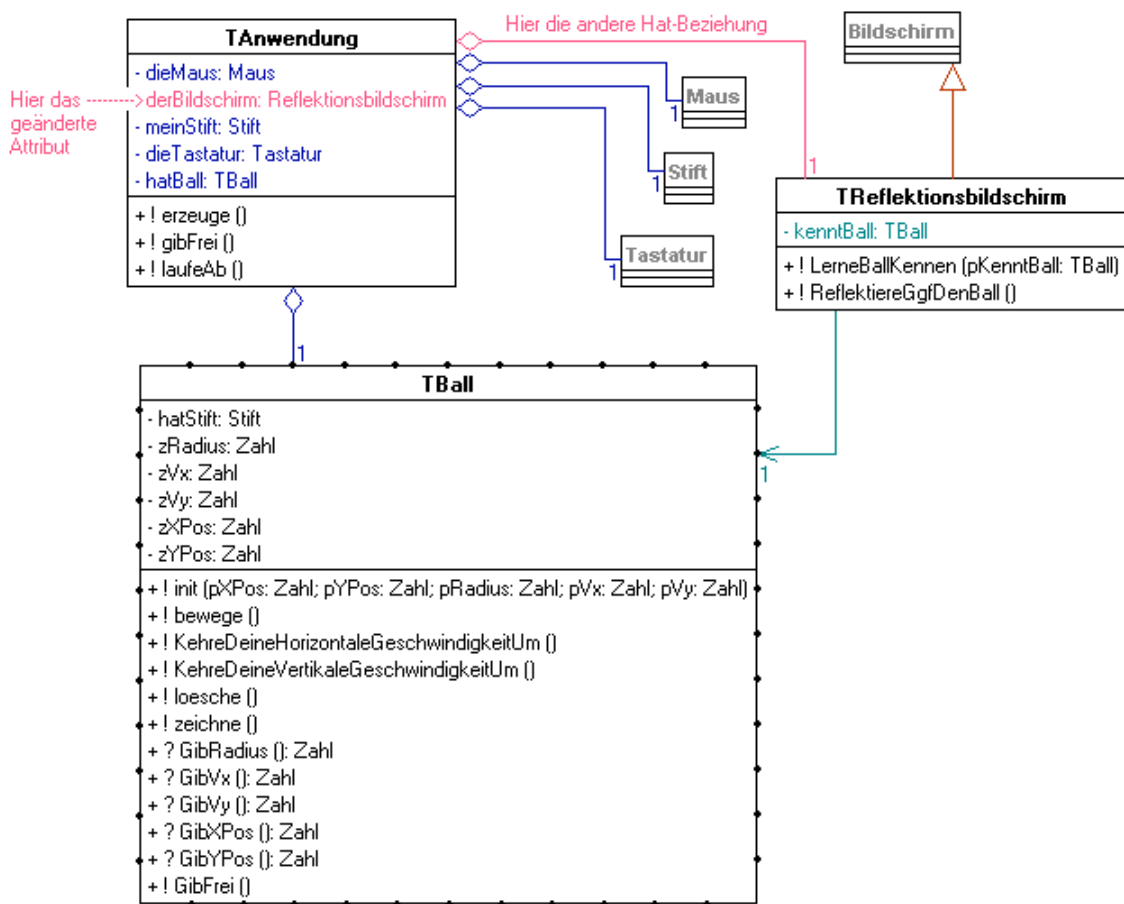
Beachten Sie, dass Aufträge violett und Anfragen grün gezeichnet werden, es sei denn, Sie wählen andere Schriftfarben aus.

3.4. Hauptdiagramm anpassen und Code der Klassen erzeugen

Kehren Sie über den Menüpunkt Fenster wieder zum Hauptdiagramm zurück. Sie werden bemerken, dass die neuen Methoden auch hier in die Klassen aufgenommen worden sind.

Als genauer Beobachter werden Sie ggf. feststellen, dass die Hat-Beziehung zwischen den Anwendung und dem Bildschirm nicht mehr korrekt ist. Die Anwendung hat ja einen ReflektionsBildschirm !

Ändern Sie dies und ändern Sie den Typ des Attributes 'derBildschirm' der Klasse TAnwendung, dann erhalten Sie folgendes UML-Hauptdiagramm (auf Maximieren gestellt):



Nun müssen Sie den Delphi-Code der Klassen TReflektionsbildschirm, TBall und TAnwendung erzeugen und wieder in das Projektverzeichnis 'Tischtennis1' exportieren.

Lassen Sie sich zuerst den Code des ReflektionsBildschirms anzeigen. (Rechtsklick in die Klasse, Menüpunkt Delphi Export) Sie stellen fest, dass die Ball-Unit sogar schon in die Uses-Liste der Klasse TReflektionsbildschirm aufgenommen wurde. Speichern Sie die Unit im Projektverzeichnis TischTennis1 unter dem vorgeschlagenen Namen ab.

Verfahren Sie ebenso mit dem Code der Klassen TBall und TAnwendung, da diese sich ja zwischenzeitlich auch geändert haben.

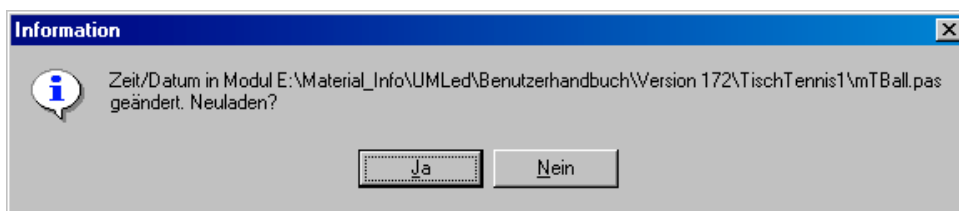
4. Einbinden von Code unter Delphi und Aktualisieren der UML-Diagramme 2.

Wir werden nun noch einmal die neuen Units in Delphi einbinden bzw. die vorhandenen in Delphi aktualisieren:

Wenn Sie so wie in dieser Anleitung beschrieben gearbeitet haben, dann müsste zum gegenwärtigen Zeitpunkt unter Delphi das Projekt 'Projektdatei.dpr' im Verzeichnis 'TischTennis1' sowie die Units 'mTAnwendung.pas' und 'mTBall.pas' geöffnet sein.

Code unter Delphi einbinden und ergänzen:

Wechseln Sie nun wieder zur Delphi-Entwicklungsumgebung. Sie erhalten sofort folgende



Frage : (oder eine entsprechende mit einer anderen schon geöffneten Datei). Beantworten Sie diese Fragen alle mit 'Ja', dann sind ihre Delphi-Dateien wieder auf dem Stand, unter dem sie unter UMLed gespeichert wurden. Fügen sie als letztes die Unit mTReflektionsbildschirm dem Projekt hinzu.

Öffnen Sie nun als erstes die Unit 'mTAnwendung'. Ändern Sie den Code der Methoden 'erzeuge' und 'laufeAb' entsprechend den grünen Zeilen:

```
constructor TAnwendung.erzeuge;
begin
    derBildschirm := TReflektionsBildschirm.init;
    meinStift := Stift.init;
    dieTastatur := Tastatur.init;
    dieMaus := Maus.init;
    hatBall := TBall.init(50, 50, 20, 0.02, 0.01);
    derBildschirm.lerneBallKennen(hatBall)
end;

procedure TAnwendung.laufeAb;
begin
    repeat
        hatBall.bewege;
        derBildschirm.reflektiereGgfDenBall;
    until dieMaus.istGedrueckt
end;
```

In der Unit 'mTBall' brauchen Sie nur den Code der folgenden Methoden ergänzen:

```
procedure TBall.KehreDeineHorizontaleGeschwindigkeitUm;
begin
    zVx := -zVx;
end;
```

```
procedure TBall.KehreDeineVertikaleGeschwindigkeitUm;
begin
    zVy := -zVy;
end;
```

Öffnen Sie nun die Unit 'mTReflektionsBildschirm'. Sie brauchen dort nur den Code der Methode 'reflektiereGgfDenBall' wie angegeben ergänzen

```
procedure TReflektionsbildschirm.reflektiereGgfDenBall;
begin
    if (kenntBall.GibXPos - kenntBall.gibRadius < 0) or
       (kenntBall.GibXPos + kenntBall.gibRadius > self.Breite)
    then kenntBall.KehreDeineHorizontaleGeschwindigkeitUm;
    if (kenntBall.GibYPos - kenntBall.gibRadius < 0) or
       (kenntBall.GibYPos + kenntBall.gibRadius > self.Hoehe)
    then kenntBall.KehreDeineVertikaleGeschwindigkeitUm;
end;
```

Wenn Sie alles richtig gemacht haben, müsste Ihre Anwendung nun korrekt ablaufen.

Ist es nicht erstaunlich, mit wie wenig Code hier gearbeitet werden kann? Eine gute Modellierung erspart eben Arbeit !!

Die Diagramme aktualisieren:

Wichtig für die weitere Arbeit mit UMLed ist es, dass die Delphi-Quelltexte und das UML-Diagramm immer synchronisiert werden. Wenn Sie mit Delphi Änderungen vorgenommen haben, geht dies immer in folgenden zwei Schritten vor sich:

1. Die geänderten Dateien unbedingt unter Delphi abspeichern
2. Zu UMLed wechseln. Die Klassen werden dann nach Rückfrage automatisch aktualisiert.

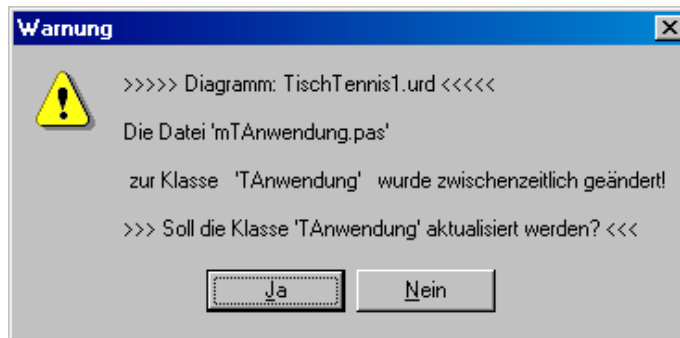
Zu 1:

Speichern Sie also unter Delphi alle Änderungen ab. Vorsicht: Es werden nur die Dateien gespeichert, die dem Projekt hinzugefügt worden sind.!!

Tipp: Um das Speichern nicht zu vergessen, sollten Sie unbedingt unter dem Menüpunkt **'Tools, Umgebungsoptionen'** auf der Karte **'Präferenzen'** unter der Rubrik **'Optionen für Autospeichern'** das Häkchen bei **'Editordateien'** setzen.

Zu 2:

Wechseln Sie zu UMLed. Sie erhalten die ihnen schon bekannten ausführlichen Fragen:



die Sie bitte wiederum alle bejahen.

Das Erscheinungsbild Ihres UML-Diagrammes hat sich nicht geändert, aber der Code der Klassen ist nun auch unter UMLed auf dem neuesten Stand!

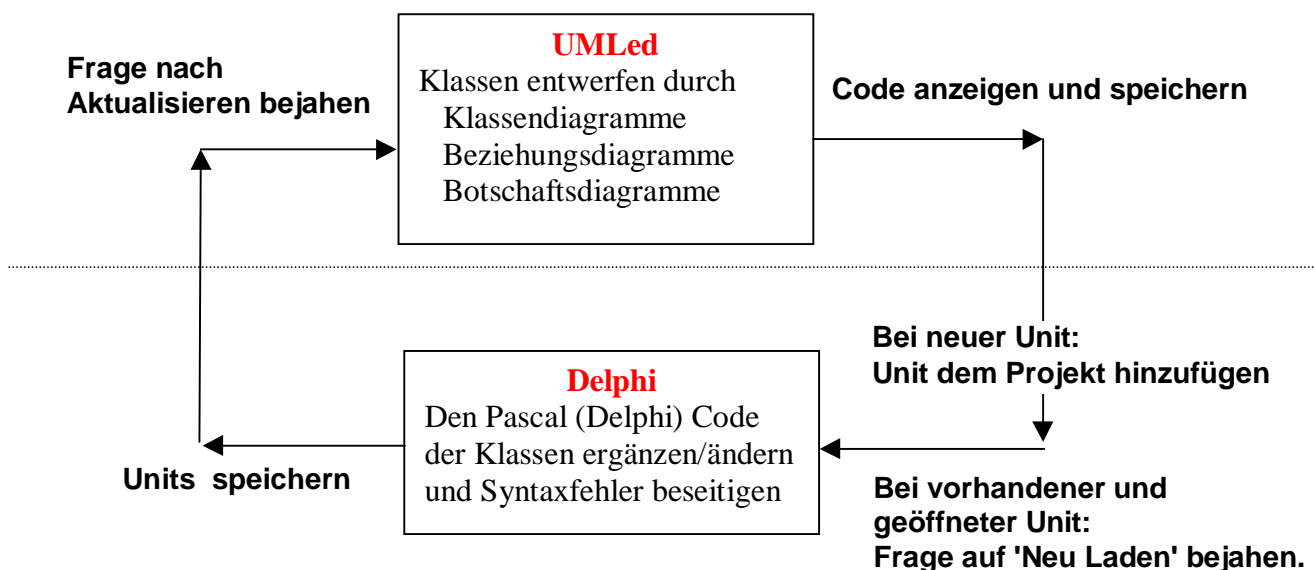
5. Organisieren und Synchronisieren der Arbeit mit UMLed und Delphi:

Voraussetzung einer guten Zusammenarbeit von UMLed und Delphi ist, dass sie erstens: Wie unter Delphi allgemein üblich alle Delphi-Dateien in einem eigenen Verzeichnis verwalten.

zweitens: Die UML-Diagramme im selben Verzeichnis abspeichern.

Die Synchronisation funktioniert dann auch, wenn Sie Delphi bzw. UMLed zwischenzeitlich schliessen oder ganze Projekt-Verzeichnisse mit Delphi und UML-Dateien kopieren oder verschieben.

Einen Überblick über das Vorgehen finden Sie hier:



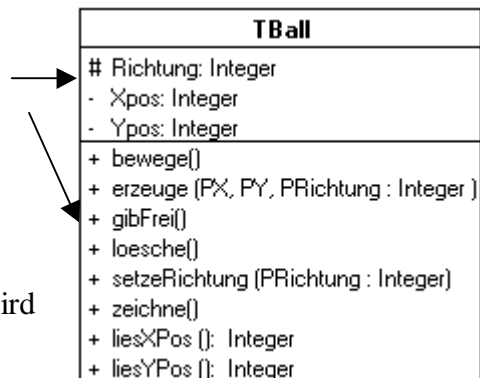
Anhang1: UML-Diagramme

Klassendiagramme:

Ein Klassendiagramm besteht aus dem Namen der Klasse sowie einem Bereich für die Attribute (Zustandsvariablen) und einem Bereich für die angebotenen Dienste (Methoden).

Falls statt einer Klasse ein Objekt dargestellt werden soll, wird der Klassenname unterstrichen.

Der Sichtbarkeitsbereich für Attribute und Dienste wird durch "+" für **public** und "-" für **private** und # für **protected** gekennzeichnet. **Private** bedeutet, dass nur diese Klasse auf die Eigenschaft zugreifen kann. **Protected** bedeutet, dass Nachfolgeklassen (und deren Nachfolgeklassen usw.) auf die entsprechende Eigenschaft zugreifen können. Andere Klassen können auf dieses Attribut nicht zugreifen. **Public** bedeutet, dass alle Klassen auf diese Eigenschaft zugreifen können. Ein Zugriff ist natürlich nur möglich, wenn überhaupt Zugriff auf ein Objekt der entsprechenden Klasse möglich ist. Unter Delphi gibt es noch das Zugriffsrecht "++" für **published**, das dem Zugriffsrecht **public** entspricht und zusätzlich die Anzeige im Objektinspektor ermöglicht.



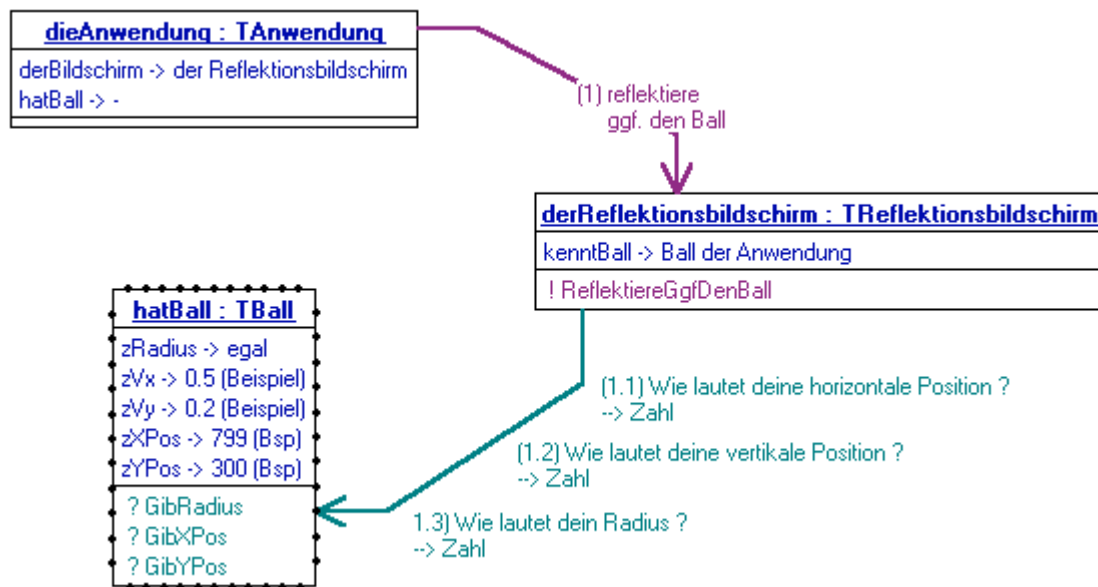
Attribute sollten immer **private** oder **protected** sein, auf sie sollte lesend und schreibend nur durch spezielle Dienste (Methoden) zugegriffen werden können.

Die Dienste/Methoden werden nach Aufträgen (realisiert durch Prozeduren) und Anfragen (realisiert durch Funktionen) unterschieden. Im Klassendiagramm sind Anfragen daran zu erkennen, dass der Parameterliste ein Resultattyp folgt. Aufträgen kann ein Rufzeichen, Anfragen ein Fragezeichen vorangestellt werden.

Dienste sind in der Regel öffentlich, also **public**. In der Entwurfsphase kann es als Vorbereitung der Programmierung jedoch durchaus sinnvoll sein, private Dienste für Objekte einer Klasse vorzusehen. Verwendet man Klassendiagramme zur Dokumentation, werden private Dienste häufig nicht aufgeführt.

Botschafts-Diagramme (Kollaborationsdiagramme)

Will ein Objekt den Dienst eines anderen Objektes nutzen, muss es ihm zur Auftragserteilung (**unten violett**) oder zum Stellen einer Anfrage (**unten grün**) eine Botschaft senden. Das Senden einer Botschaft wird durch einen Pfeil vom Auftraggeber (Client) zum Auftragnehmer (Server) dargestellt. Werden Daten versandt, könne diese mit Richtungsangabe und Typ am Botschaftspfeil notiert werden. Parameter sind Daten, die vom Auftraggeber zum Auftragnehmer fließen. Die Antworten auf eine Anfrage fließen vom Auftragnehmer zum Auftraggeber.



Diese Diagramme sollten innerhalb von **UMLed** immer als Unterdiagramme eines Hauptdiagramms realisiert werden.

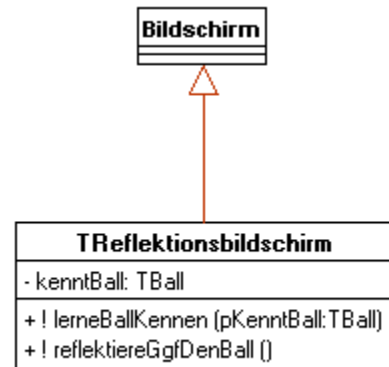
Beziehungsdigramme:

IST-Beziehung (Verfeinerung bzw. umgekehrt Verallgemeinerung, Vererbung)

Durch einen Pfeil auf die Oberklasse wird die Ist-Beziehung dargestellt.

Gemeint ist im nebenstehenden Diagramm, dass die Menge der Reflektionsbildschirme in der Menge der Bildschirme enthalten ist.

In der Unterklasse werden nur die zusätzlichen Attribute und Methoden aufgeführt.



Die Kennt-Beziehung (Verbindung, Assoziation)

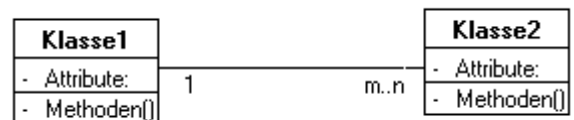
Will ein Objekt einen Dienst eines anderen Objektes in Anspruch nehmen, muss es dieses Objekt kennen.

Die Assoziation ist also eine Beziehung zwischen Objekten. Sie wird jedoch normalerweise mit Hilfe der Klassen dargestellt.

Die mögliche Anzahl der gekannten Objekte wird durch Zahlen angegeben. Ist die Anzahl nicht bekannt, verwendet man Buchstaben. So bedeutet die nebenstehende Angabe "m..n" beispielsweise, dass ein Objekt der Klasse1 mindestens **m** und höchstens **n** Objekte der Klasse2 kennt. Objekte der Klasse2 kennen die Objekte der Klasse 1 jedoch nicht !



Handelt es sich um eine wechselseitige Kennt-Beziehung, wird dies durch entsprechende Zahlenangaben (Kardinalitäten) an der Verbindung dargestellt. Im nebenstehenden Beispiel kennen die Objekte der Klasse 2 jeweils genau ein Objekt der Klasse 1.



Die Hat-Beziehung (Zerlegung, Aggregation)

ist eigentlich eine spezielle kennt Beziehung. Der Besitzer hat (besitzt) Objekte einer anderen Klasse. Im Gegensatz zur Kennt Beziehung, bei der es sich um die Kontaktaufnahme zweier autonomer Objekte handelt, ist bei der HAT-Beziehung das besitzende Objekt für die Erzeugung, Verwaltung und das Löschen des anderen Objektes zuständig

Auch die HAT-Beziehung wird meistens mit Klassen dargestellt. Die Verbindungslinie zeigt eine (eigentlich gefüllte) Raute bei der besitzenden Klasse.



Können die Objekte, die der Besitzer hat, auch ohne diesen existieren, (schwache Aggregation), dann wird die Raute nicht ausgefüllt. Bsp: Jedes Auto hat einen Motor. Ein Motor kann aber auch ohne Auto existieren. Da im Unterricht die schwache Aggregation nicht vorkommt, arbeitet UMLed immer mit unausgefüllten Rauten.

Die Anzahl der Objekte, die der Besitzer hat, kann auch hier durch Zahlenangaben dargestellt werden. Ist die Anzahl nicht bekannt, verwendet man wie bei der Assoziation Buchstaben.