

## KAPITEL 8

## VERBUNDE - Der Datentyp RECORD

## 8.1. Nicht erweiterbare Verbunde

In diesem Kapitel begegnen Sie einem der vielseitigsten Component Pascal Datentypen, dem Verbund. Verbunde sind ebenso wie Felder (Datentyp *ARRAY*) strukturierte Datentypen, der Benutzer kann in einer Verbunddeklaration wie bei einem *ARRAY*-Typ eine frei wählbare Zahl von Einzelvariablen zusammenfassen. Anders jedoch als bei einem *ARRAY*-Typ können in einem Verbund Einzelvariablen unterschiedlicher Typen erscheinen. Diese Einzelvariablen - die oft auch (etwas verwirrend) Felder des Verbundtyps genannt werden - müssen im Gegensatz zu den Einzelvariablen in einem *ARRAY*-Typ Namen erhalten, das heißt, innerhalb eines Verbundtyps wird jedes Feld so aufgeführt, als ob es sich bei ihm um eine selbständige Variable handelte, es setzt sich zusammen aus einem Bezeichner und dem durch einen Doppelpunkt separierten zugeordneten Datentyp.

Sie finden Beispiele von Verbundtypen im Programm *Verbund.odc*. Ein Verbund wird durch die Schlüsselwörter *RECORD* (<Basistyp>) ... *END* deklariert, wobei an Stelle der Punkte die Felder des Verbundes stehen. Die Angabe des Basistyps ist nur erforderlich bei einem Erweiterungstyp, in diesem Fall muß er dem Schlüsselwort *RECORD* in runden Klammern angefügt werden (genaueres dazu erfahren Sie im Abschnitt 8.3). Wie bei dem Datentyp *ARRAY* sind beim Datentyp *RECORD* sowohl explizite als auch anonyme Typdeklarationen möglich, beide Varianten erscheinen im Programm *Verbund.odc*, einmal als expliziter Typ *Eigenschaften*, zum anderen als anonymer Typ der Variablen *Nichts*. An der Deklaration der Variablen *Nichts* sehen Sie neben der Tatsache, daß Verbundtypen anonym sein können, daß ein Verbundtyp auch dann syntaktisch korrekt deklariert ist, wenn er keine Einzelfelder enthält. Eine Variable wie *Nichts* benötigt natürlich keinerlei Speicherplatz, allerdings kann sie auch keine Daten aufnehmen, sie ist anscheinend sinnlos. Sie werden in späteren Kapiteln im Zusammenhang mit dem Prinzip der Vererbung jedoch sehen, daß leere Verbunde sinnvoll sein können.

Als zweite Verbundvariable finden Sie im Programm die Variable *Kind*: *Eigenschaften*. Diese ist komplexer gebaut, sie zeigt, daß ein strukturierter Datentyp wie *ARRAY* sowohl einfache als auch strukturierte Grundtypen enthalten darf, wobei diese Grundtypen nicht nur - wie hier - *RECORD*-Typen, sondern auch *ARRAY*- oder andere strukturierte Typen sein können.

An dieser Stelle möchte ich auf eine sehr wichtige Tatsache hinweisen, die Sie im Typ *Eigenschaften* durch das fünfte, in Kommentarklammern stehende Feld *Unmöglich*: *Eigenschaften* illustriert sehen. Syntaktisch ist es durchaus zulässig, in einem Verbundtyp ein Feld zu deklarieren, dem als Typ der Typ des

Verbundes selbst zugewiesen wird. Es ist jedoch unmittelbar verständlich, daß eine solche Deklaration nicht erlaubt sein kann, da sie ins Endlose führen würde. Eine Variable des Typs *Eigenschaften* müßte neben allen anderen Feldern eine vollständige Ausgabe ihrer selbst enthalten, die ihrerseits erneut sich selbst enthielte, die erneut ... , der Arbeitsspeicher jedes Computers liefe binnen kürzester Zeit über. Wenn Sie die Kommentarklammern um die Deklaration des Feldes *Unmöglich* entfernen und danach das Programm kompilieren lassen, werden Sie bemerken, daß solche rekursiven Typdeklarationen bereits vom Compiler erkannt und mit einer Fehlermeldung quittiert werden. Die Unmöglichkeit rekursiver Typdeklarationen besteht nicht nur für den Datentyp *RECORD*, sondern auch für den Typ *ARRAY* und generell für alle explizit deklarierbaren Typen, die syntaktisch rekursive Definitionen gestatten. Von dieser Regel gibt es eine einzige - allerdings wesentliche - Ausnahme, die Sie im 10. Kapitel im Zusammenhang mit den Zeigertypen kennenlernen werden.

Jedem Einzelfeld einer Verbundvariablen kann und muß wie einer einfachen Variablen gesondert ein Wert zugewiesen worden sein, bevor dieser im weiteren Verlauf des Programms verwendet wird. Da Variablen eines Verbundtyps (nur) eine Zusammenfassung von Einzelvariablen - der Felder des Typs - sind, muß man jede dieser Einzelvariablen gesondert adressieren können, der Anweisungsteil der Prozedur *Start* zeigt Ihnen, auf welche Weise Verbundvariablen und ihre Felder zu behandeln sind. Jedes Feld einer Verbundvariablen wird dadurch adressiert, daß man den Bezeichner des gewünschten Feldes hinter den Variablennamen schreibt und die Teile durch einen Punkt zusammenfügt. Sie finden Beispiele dafür bei den Wertzuweisungen an die Felder der einzelnen Verbunde des *ARRAYs Kind* sowie bei der das Programm abschließenden *FOR*-Schleife, die die Inhalte jedes Verbundes von *Kind* ausgibt.

Verbunde wie *Eigenschaften* werden final oder nicht erweiterbar genannt (was mit diesen Bezeichnungen gemeint ist, lernen Sie im Abschnitt 8.3 kennen). Bei finalen Verbundvariablen gibt es wie bei Variablen des Typs *ARRAY* eine einzige globale Operation, die Zuweisung, durch die der gesamte Inhalt einer Variablen eines Verbundtyps einer zweiten Variablen, die von dem selben (expliziten) Typ ist, global zugewiesen werden kann, Sie sehen dies in der Zeile *Kind[2] := Kind[0]* illustriert. Alle anderen Operationen mit *RECORD*-Variablen müssen für jeden konkreten Einzelfall individuell programmiert werden, der folgende Abschnitt wird Ihnen zeigen, welche Möglichkeiten dafür bei Verbundtypen und -variablen bestehen.

### 8.2. Daten und Verbunde - Speichernutzung statisch

Wie schon erwähnt, können die einzelnen Datenfelder eines Verbundes von unterschiedlichem Typ sein, wobei nicht nur einfache Grundtypen als Typen der Felder zugelassen sind, sondern auch strukturierte Typen. Als Beispiel finden Sie im Deklarationsteil des Programms *DatRec.odc* zwei Verbundtypen, *Datum*

und *Anschrift*. Während die Felder des Typs *Datum* alle Ganzzahltypen haben, sind zwei der Felder des Typs *Anschrift* selbst vom Typ *RECORD*, das Feld *GanzerName*, dessen Typdeklaration anonym ist, und das Feld *Tag* mit dem explizit deklarierten Typ *Datum*.

Für Variablen strukturierter Datentypen kann es, wie erwähnt, keine vordefinierten Ausgabeanweisungen geben, daher findet sich im Anschluss an die Variablendeklarationen die Prozedur *Schreib*. Sie faßt eine Reihe von Anweisungen zusammen, die die aktuellen Werte der Felder des Parameters *Es* vom Typ *Anschrift* auf den Bildschirm bringen. Beachten Sie bitte, daß *Es* als *IN*-Parameter übergeben wird, Variablen eines Verbundtyps sollten wegen ihres Speicherverbrauchs wie alle Variablen strukturierter Datentypen nur in zwingenden Fällen nicht als *IN*-Parameter übergeben werden.

Auf zwei Punkte möchte ich im Zusammenhang mit der Prozedur *Schreib* aufmerksam machen. Zum einen sehen Sie, daß bei *RECORD*-Typen eine - im Prinzip - unendliche Staffeltiefe der geschachtelten Verbunde möglich ist, da jede nächsttiefere Stufe im Variablenaufruf einfach durch einen weiteren Punkt von ihrer Vorgängerin getrennt wird. Zum zweiten zeigt die Zeile *Out.Int(Es.Tag.Tag, 3)*, daß bei Variablen eines *RECORD*-Typs die spezielle Notation mit einem separierenden Punkt vor jedem Feldbezeichner es erlaubt, den Feldern unterschiedlicher Hierarchiestufen identische Namen zu geben, da diese für den Compiler, ähnlich wie bei den Variablenhierarchien in geschachtelten Prozeduren, durch die Abfolge der einzelnen Hierarchiestufen innerhalb einer Verbundvariablen voneinander verschieden sind (selbstverständlich läßt sich bei der Namenswahl in der Regel eine Doppelung der Bezeichner - wie hier *Tag* - vermeiden, sie wurde absichtlich gewählt, um die Möglichkeit der Namensgleichheit zu demonstrieren).

Das Prozedur *Start* enthält keine Neuigkeiten, sie demonstriert einerseits die Art, in der die Werte der verschiedenen Variablenfelder verändert werden können, andererseits verdeutlicht sie die Tatsache, daß Verbundvariablen global zugewiesen (kopiert) werden können, sofern die bereits im Zusammenhang mit *ARRAY*-Variablen genannten Bedingungen (s. Kap. 7.3) erfüllt, die Variablen also für den Compiler erkennbar vom selben Typ sind. Allerdings gilt diese Kopierbarkeit auch unter den genannten Bedingungen nur solange, wie die Typen der Variablen final sind. Genaueres dazu erfahren Sie im folgenden Abschnitt im Zusammenhang mit erweiterbaren Verbunden.

### 8.3. Vererbung - Erweiterbare Verbunde

Ein zentraler Begriff aktueller Programmierverfahren, den Sie vielleicht schon kennen oder zumindest gehört haben, ist die "objektorientierte Programmierung" (OOP). Als Konzept ist objektorientierte Programmierung bereits in den sechziger Jahren des 20. Jahrhunderts mit der Sprache *Simula* entstanden, es dauerte allerdings ungefähr zwanzig Jahre, bis sich diese Idee mit der Entwicklung der Sprache *Smalltalk*

verbreitete. Seitdem sind praktisch alle älteren Programmiersprachen mit OOP-Fähigkeiten nachgerüstet worden, außerdem wurde eine Reihe neuer Sprachen entwickelt, bei deren Entwurf die Verwirklichung der objektorientierten Programmierung von Anfang an im Mittelpunkt stand.

Während in den meisten Sprachen objektorientierte Programmierung über einen eigenen Datentyp - die Klasse - durchgeführt wird, benutzt Component Pascal dafür den vorhandenen Typ *RECORD*. Wichtigstes Merkmal der OOP ist die Vererbung oder Typerweiterung, die Sie in diesem Kapitel kennenlernen. Zum besseren Verständnis der Typerweiterung ist es sinnvoll, die Begriffe Datentyp und Variable genauer zu analysieren. Aus der Sicht der Verwendung einer Variablen innerhalb eines Programms ist eigentlich nur der aktuell in ihr gespeicherte Wert interessant. Prinzipiell wären deshalb nicht einmal Variablen als besonderes Konstrukt erforderlich, man könnte die jeweils gewünschten Daten direkt im Arbeitsspeicher adressieren, was in den Anfängen der Verwendung von Computern auch geschah. Allerdings sind die daraus resultierenden Risiken einer Fehladressierung so groß, daß man es als sinnvoll angesehen hat, die Speicheradressen nicht direkt, sondern auf dem Umweg über eine Variablenliste anzusteuern, in der die Adressen aller aktuellen Daten unter Namen zusammengefaßt sind. Da alle Daten in einem Computer als Zahlen "verschlüsselt" werden, war es außerdem nötig, zusätzlich zu den Variablennamen Datentypen für die Variablenwerte einzuführen, die eine korrekte "Entschlüsselung" der Daten ermöglichen. Genau genommen besteht eine Datenvariable also aus drei Einzelteilen: ihrem aktuellen Wert, ihrem Namen und dem zugehörigen Datentyp.

Im Gegensatz zur Verwendung einer Variablen im Programm, bei der es auf den aktuell gespeicherten Wert ankommt, ist aus der Sicht der Datenstruktur das Wesentliche nicht der Variablenwert, sondern der Variablentyp, den man als "Konstruktionsplan" der Variablen bezeichnen könnte. Bei einem skalaren Datentyp, beispielsweise *INTEGER*, erscheint die Bezeichnung Konstruktionsplan vielleicht ein wenig hochgegriffen, nehmen Sie aber statt dessen den Typ *RECORD* wird schnell klar, daß der Typ als abstraktes Konstruktionsmuster, als Bauplan aller Variablen dieses Typs angesehen werden kann, mit seiner Erstellung sind sämtliche Eigenschaften der von ihm abgeleiteten Variablen definiert und festgelegt.

Der enge und nicht auflösbare Zusammenhang von Variabler (genau genommen Variablenname), Wert und Typ hat auf der einen Seite eine deutliche (und unverzichtbare) Verbesserung der Programmsicherheit und Programmstabilität bewirkt, auf der anderen Seite hat sich jedoch gezeigt, daß die Möglichkeiten der Programmgestaltung durch dies starre Schema allzu sehr eingeengt wurden. Um die Einengung zu überwinden, aber dennoch auf die Stabilitätseigenschaften typisierter Variablen nicht wieder zu verzichten, sind zwei zusätzliche Konstruktionsprinzipien eingeführt worden, zum einen die oben erwähnte Vererbung, die in diesem Abschnitt dargestellt wird, zum anderen die Zeigertypen, die Sie im 10. Kapitel kennenlernen.

Vererbung ermöglicht es, den ursprünglich statischen Typ *RECORD* durch Erweiterung zu verändern, während sich mit Zeigern der feste Zusammenhang zwischen einem Variablenwert und dem Namen der

Variablen lockern läßt; bei Verwendung von Zeigern kann ein und der selbe Variableninhalt seinen Namen wechseln oder mehr als einen Namen erhalten. Die Verbindung beider Konzepte, Zeiger und Vererbung, ergibt die wesentlichste Anwendungsform der objektorientierten Programmierung, die sogenannte Polymorphie, deren Erläuterung Sie im 11. und 12. Kapitel finden.

Sehen Sie sich zum Kennenlernen der Vererbung das Modul *TutErbRec1* an. Im Deklarationsteil finden Sie drei Verbundtypen, *Rikscha*, *Fahrzeugart* und *Boot*. Der Typ *Rikscha* ist ein Verbundtyp, wie Sie ihn in den beiden vorherigen Programmen kennengelernt haben. Er ist final, das heißt, er kann nicht erweitert werden und er ist allozierbar, es können Variablen dieses Typs deklariert werden.

Der Typ *Fahrzeugart* unterscheidet sich von finalen *RECORD*-Typen wie *Rikscha* durch das Component Pascal Schlüsselwort *EXTENSIBLE*, mit diesem Attribut wird ein *RECORD*-Typ als erweiterbar erklärt. Was damit gemeint ist, sehen Sie an dem dritten Typ, *Boot*, dessen Deklaration hinter dem Schlüsselwort *RECORD* in runden Klammern als Zusatz den Bezeichner des Basistyps *Fahrzeugart* enthält. Dadurch ist er für den Compiler (und für den Leser des Programms) als Erweiterung dieses Typs gekennzeichnet, eine Variable des Typs *Boot* enthält nicht nur die in diesem Typ deklarierten Felder *Riemen*, *Segel* und *Antreiber*, sondern auch alle Felder des Basistyps *Fahrzeugart*, also die Felder *Besitzer*, *Gewicht*, *Preis*, der Erweiterungstyp *Boot* "erbt" diese Felder von seinem Basistyp.

In Kommentarklammern sehen Sie die Versuche, die Typen *EinRad* und *RuderBoot* zu deklarieren und dazu die Fehlermeldung *base type is not extensible*, die Sie erhielten, falls Sie die Kommentarklammern vor der Kompilation entfernten. *RECORD*-Typen ohne explizite Attribute sind final, sie können nicht erweitert werden, daher die Unmöglichkeit, Untertypen von *Rikscha* und *Boot* zu deklarieren.

Durch Attribute, die dem Schlüsselwort *RECORD* vorangestellt sind, werden in Component Pascal die Eigenschaften von Verbundtypen geändert. Insgesamt gibt es vier Möglichkeiten für Attribute von Verbundtypen, die nachstehend zusammen mit ihren Eigenschaften dargestellt sind (siehe dazu auch Anhang B)

Attribut	erweiterbar	allozierbar	zuweisbar
-- (final)	Nein	Ja	Ja
EXTENSIBLE	Ja	Ja	Nein
ABSTRACT	Ja	Nein	Nein
LIMITED	Im definierenden Modul	Im definierenden Modul	Nein

Attribute und Eigenschaften von Verbunden

Der ersten Zeile der Übersicht können Sie entnehmen, daß ein finaler Verbundtyp "allozierbar" ist (Englisch: may be allocated), das heißt, es lassen sich Variablen des Typs deklarieren, derartige Variablen sind global zuweisbar, können also kopiert werden. Sie finden diese Tatsachen in der Kommandoprozedur *Start* durch die Deklarationen der Variablen *AsiaStandard: Rikscha* und *MolaMola, Nautilus: Boot* sowie in der Zeile *Nautilus := MolaMola* bestätigt, in der die Variable *MolaMola* in die Variable *Nautilus* kopiert wird.

Im Gegensatz zu finalen Verbunden ohne sichtbare Attribute werden Verbunde, deren Deklarationen explizit eines der Attribute *EXTENSIBLE*, *ABSTRACT*, *LIMITED* enthalten, erweiterbare Verbunde genannt. Erweiterbare Verbunde können "beerbt" werden, das heißt, es lassen sich Erweiterungstypen solcher Verbunde deklarieren, Sie haben dies bei dem Typ *Boot* gesehen, der ein Erweiterungstyp des Basistyps *Fahrzeugart* ist.

Variablen des Typs *EXTENSIBLE RECORD* sind allozierbar, wie Sie der Deklaration der Variablen *Vergangenheit, Zukunft: Fahrzeugart* in der Prozedur *Start* entnehmen können, sie sind in dieser Hinsicht identisch mit Variablen finaler Typen. Andererseits verdeutlicht die in Kommentarklammern der Zeile *Vergangenheit := Zukunft* zugefügte Fehlermeldung *incompatible assignment*, daß solche Variablen nicht global zuweisbar sind, Kopierversuche können wegen des Attributs *EXTENSIBLE* bereits vom Compiler erkannt und abgewiesen werden.

Im übrigen sollte Ihnen die Prozedur *Start* keine Schwierigkeiten bereiten, sie enthält Wertzuweisungen an die Felder einiger deklarierter Variablen und Anweisungen zur Ausgabe der Ergebnisse. Im folgenden Abschnitt werden Sie mehr über Verbunde und insbesondere über die beiden bisher nicht besprochenen Attribute *ABSTRACT* und *LIMITED* erfahren.

### Vererbung - Klassen

Laden Sie das Programm *ErbRec2.odc*, lassen Sie das Modul kompilieren und lassen Sie sich die Schnittstelle des Moduls anzeigen (*Info* → *Client Interface*). Das einzige Modul, das bisher neben den Kommandoprozeduren weitere Bezeichner exportierte, war das Modul *TutKonvert1*, aus dem mehrere Variablen und Bezeichner für die Erstellung der Dialogbox benötigt und deshalb exportiert wurden. Dagegen sieht die Schnittstelle von *TutErbRec2* umfangreicher aus, das Modul exportiert außer der Prozedur *Start* insgesamt vier Verbundtypen: *Fahrzeugart*, *Boot*, *BootZeit* und *Rikscha*

DEFINITION *TutErbRec2*;

TYPE

*Fahrzeugart* = *ABSTRACT RECORD*  
*Besitzer*: *ARRAY 25 OF CHAR*;

*Gewicht*: *INTEGER*;  
*Preis*: *REAL*  
*END*;

*Boot* = *EXTENSIBLE RECORD (Fahrzeugart)*  
*Fahrzeugsorte*: *ARRAY 15 OF CHAR*;  
*Riemen*: *INTEGER*;  
*Segel*: *BOOLEAN*;  
*Antrieb*: *ARRAY 15 OF CHAR*  
*END*;

*BootZeit* = *RECORD (Boot)*  
*Text*: *PROCEDURE*  
*END*;

*Rikscha* = *LIMITED RECORD (Fahrzeugart) END*;

*PROCEDURE Start*;

*END TutErbRec2*.

#### Schnittstelle des Moduls *TutErbRec2*

Eine Schnittstelle wie diese ist eigentlich der Regelfall, Module exportieren überwiegend Typdefinitionen und Prozeduren, die von anderen Modulen verwendet werden können; im Fall von *TutErbRec2* sind die Benutzer die Module *TutErbRec3* (Programm 8.3b) und *TutErbRec4* (Programm 10.3).

Vergleichen Sie die exportierten Typen in der Schnittstelle von *TutErbRec2* mit den im Modul stehenden Deklarationen dieser Typen, wird Ihnen auffallen, daß Component Pascal bei *RECORD*-Typen eine sehr feine Abstufung der Exporte erlaubt, es können nicht nur vollständige Verbunde exportiert werden, sondern es ist möglich (und nötig), jedes einzelne Verbundfeld gesondert zu exportieren, sofern dieses außerhalb des Moduls sichtbar sein soll. Sie sehen dies an dem Typ *BootZeit*, der nur das Feld *Text* und dem Typ *Rikscha*, der überhaupt kein Feld exportiert, dieser Typ erscheint in der Schnittstellenbeschreibung als leer, mit der Konsequenz, daß die nicht exportierten Felder nur innerhalb des definierenden Moduls verwendet werden können.

Die Deklaration des Verbundtyps *Fahrzeugart* enthält das Attribut *ABSTRACT*, der Typ ist dadurch als nicht allozierbar definiert, die in der Prozedur *Start* in Kommentarklammern zu findende Deklaration der Variablen *Nautilus: Fahrzeugart* ist nicht möglich. Dagegen ist der Typ erweiterbar, Sie sehen die Erweiterbarkeit bestätigt in den Deklarationen der Typen *Boot* und *Rikscha*, die Untertypen des Basistyps *Fahrzeugart* sind. Den Typ *Boot* mit dem Attribut *EXTENSIBLE* und den davon abgeleiteten finalen Typ *BootZeit* kennen Sie prinzipiell bereits aus dem Programm *ErbRec1.odc*, neu ist dagegen der Typ *Rikscha*

mit dem Attribut *LIMITED*. Innerhalb desjenigen Moduls, das die Typdeklaration enthält, hat ein Verbundtyp mit dem Attribut *LIMITED* die selben Eigenschaften wie der Typ *EXTENSIBLE RECORD*, er kann erweitert werden, die Deklaration des von *Rikscha* abgeleiteten (nicht exportierten) Typs *EinRad* zeigt Ihnen diese Tatsache, und es können Variablen dieses Typs alloziert werden.

An der Deklaration des Typs *EinRad* sehen Sie darüber hinaus, daß Erweiterungstypen von *LIMITED RECORD* Typen selbst das Attribut *LIMITED* erhalten müssen, sie erben das Attribut von ihrem Basistyp. Dies hängt damit zusammen, daß der Typ *LIMITED RECORD* andere Eigenschaften hat als der attributlose Typ *RECORD*, Erweiterungstypen jedoch die Eigenschaften des Basistyps bewahren müssen. Eine dieser anderen Eigenschaften wurde schon erwähnt, ein *LIMITED RECORD* Typ kann außerhalb des definierenden Moduls nicht erweitert werden, ebensowenig lassen sich dort Variablen des Typs allozieren (s. dazu das Modul *TutErbRec3*). Ein anderer Unterschied besteht darin, daß es auch im definierenden Modul nicht möglich ist, Variablen mit dem Typ *LIMITED RECORD* zu kopieren, wie Sie an der auskommentierten Zeile *TriCykel := AsiaStandard* in der Prozedur *Start* sehen.

Die Typen *BootZeit* und *Rikscha* weisen eine Besonderheit auf, eins ihrer Felder ist *Text: PROCEDURE*. Im Kapitel 6.2 haben Sie Prozedurtypen kennengelernt, die die Möglichkeit bieten, Prozedurvariablen zu deklarieren, denen dynamisch, zur Laufzeit des Programms, Prozeduren zugewiesen werden können, sofern deren Signatur mit der Typdeklaration kompatibel ist. In gleicher Weise kann einem Prozedurfeld innerhalb einer Verbundvariablen zur Laufzeit eines Programms eine entsprechende Prozedur zugewiesen werden. Der wesentliche Vorteil von Prozedurvariablen bzw. -feldern besteht darin, daß mit ihrer Deklaration nur der Prozedurtyp, nicht aber Name oder Inhalt der zuweisbaren Prozeduren festgelegt werden.

Dadurch kann bei der Programmerstellung mit Prozedurvariablen und -feldern so umgegangen werden, als ob die Prozeduren bereits existierten, auf Grund der Typdeklarationen ist der Compiler in der Lage, die Korrektheit des Programms zu überprüfen, denn den Prozedurvariablen bzw. -feldern können nur Prozeduren mit kompatibler Signatur zugewiesen werden. Diese Zuweisbarkeit besteht auch über Modulgrenzen hinweg, sofern die entsprechenden Typen exportiert werden, die Implementierung der jeweiligen Prozeduren kann daher zu viel späteren Zeitpunkten als die Deklaration des Basistyps erfolgen, auch im Zusammenhang mit Programmen, deren Inhalt dem Programmierer des Basistyps völlig unbekannt ist. Auf diese Weise ist es möglich, umfangreiche Programme als Halbfabrikate zu erzeugen, in denen späterhin von den Endnutzern nicht nur die aktuellen Daten, sondern auch die konkreten Prozeduren zu ihrer Verarbeitung ergänzt werden können.

Datentypen, die neben den Datenfeldern auch die zur Verarbeitung der Daten nötigen Prozeduren enthalten und ihre Dienste nach außen über eine Schnittstelle zur Verfügung stellen, während die Implementierung verborgen bleibt, werden als abstrakte Datentypen (ADT, zu Einzelheiten s. Kap. 11) bezeichnet.

Abstrakte Datentypen haben sich als erfolgreiche Konstruktionen erwiesen bei dem Bemühen, Programme übersichtlich und stabil zu gestalten.

Eine weitere Notwendigkeit neben diesen Aspekten war jedoch die Erhöhung der Programmflexibilität. Ein ADT ist zwar dynamisch in dem Sinn, daß die Daten und Prozeduren erst zur Laufzeit des (Anwender-) Programms bereitgestellt werden müssen, er ist jedoch statisch in dem Sinn, daß er den Anwender auf die vom Programmierer vorgegebenen Daten- und Prozedurfelder einschränkt. Die Aufhebung dieser Beschränkung durch die Einführung der Erweiterbarkeit von Datentypen stellte eine so tiefgreifende Veränderung im Verständnis von Programmerstellung dar, daß sich unter dem Begriff der objektorientierten Programmierung (OOP) eine nahezu komplett neue Terminologie herausbildete

OOP Terminologie	Component Pascal Terminologie
Klasse	erweiterbarer Verbundtyp mit dynamisch gebundenen Prozeduren (erweiterbarer ADT)
Vererbung	Typweiterung
Subklasse	Erweiterungstyp eines Basistyps
Superklasse	Basistyp eines Erweiterungstyps
Attribut <sup>9</sup>	Datenfeld eines erweiterbaren ADT (einer Klasse)
Methode	Prozedur eines erweiterbaren ADT (einer Klasse)
Objekt, Instanz	Variable eines erweiterbaren ADT (einer Klasse)

Objektorientierte Terminologie und Component Pascal Terminologie

Der wichtigste Begriff innerhalb des Konzepts der objektorientierten Programmierung ist die Klasse. Eine umfassende Darstellung der Eigenschaften von Klassen ist an dieser Stelle unmöglich, je nach verwendeter Programmiersprache und -schule variieren die Definitionen auch mehr oder minder stark, aber die beiden wichtigsten Eigenschaften von Klassen sind in allen Definitionen ähnlich

<sup>9</sup> Unglücklicherweise ist der Begriff Attribut nicht eindeutig, er wird verwendet  
 1. Als Bezeichner der Datenfelder von Klassen,  
 2. Zur Beschreibung der Eigenschaften von Verbundtypen (s. Anhang B 2.3.),  
 3. Zur Beschreibung der Eigenschaften von typgebundenen Prozeduren (s. Anhang B 3.2.2.).

- Klassen sind zu einem Typ zusammengefaßte Daten und Operationen
- Klassen können erweitert werden

Wesentliches Merkmal einer Klasse (und der Unterschied zu einem ADT) ist die Erweiterbarkeit. Bezeichnet man einen (exportierten) Verbundtyp zusammen mit den auf ihm (genauer: auf Variablen des Typs) operierenden Prozeduren als ADT, kann man sagen, daß in Component Pascal eine Klasse ein erweiterbarer ADT ist. (Einzelheiten zu Klassen und Objekten finden Sie im 10. Kapitel sowie im 11. Kapitel im Zusammenhang mit dem Begriff der Polymorphie).

Im Sinn der vorstehenden Definitionen sind die Typen *BootZeit* und *Rikscha* Klassen, sie sind erweiterbar und enthalten mit *Text*: *PROCEDURE* Prozedurfelder, denen an anderer Stelle deklarierte Prozeduren zugewiesen werden können, Sie sehen Beispiele dafür in diesem und dem folgenden Programm *ErbRec3.odc*.

Im vorliegenden Programm *ErbRec2.odc* finden Sie außer den Typdeklarationen die Deklarationen mehrerer globaler Variablen und die übliche Kommandoprozedur *Start* sowie drei weitere Prozeduren, *RikschaText*, *BootText* und *BootZeitText*. Die Namen verweisen augenscheinlich auf die entsprechenden Verbundtypen, sie werden auch in der Prozedur *Start* im Zusammenhang mit Variablen der jeweiligen Typen verwendet. Während aber die Prozeduren *RikschaText* und *BootZeitText* den *Text*-Feldern der Verbundvariablen *AsiaStandard* bzw. *MolaMola* zugewiesen und als solche aufgerufen werden, kann die Prozedur *BootText* nicht in gleicher Weise mit der Variablen *Zukunft* verknüpft werden, da deren Typ *Boot* kein entsprechendes Feld besitzt, *BootText* kann inhaltlich auch im Zusammenhang mit Variablen des Typs *Boot* nur über einen gewöhnlichen Prozeduraufruf verwendet werden.

Machen Sie sich bitte noch einmal deutlich, daß es sich bei allen Prozeduren dieses Moduls um die bisher üblichen Prozeduren handelt, die sowohl Prozedurfeldern in Verbundvariablen bzw. selbständigen Prozedurvariablen zugewiesen als auch unabhängig davon an jeder Stelle des Programms aktiviert werden können. Im folgenden Programm *ErbRec3.odc* werden Sie erfahren, daß es neben diesen Prozeduren eine zweite, nur im Zusammenhang mit Verbunden deklarierbare Prozedurart gibt, die typgebundenen Prozeduren.

Zwei Einzelheiten seien noch erwähnt. In Bezug auf die Eigenschaften von Verbundtypen zeigt Ihnen die Kommandoprozedur *Start* in der auskommentierten Zeile *Zukunft := MolaMola*, daß für den Compiler ein Basistyp und ein von ihm abgeleiteter Untertyp als verschieden gelten, Variablen solcher Typen sind nicht zuweisungskompatibel.

Der zweite Punkt betrifft eine mögliche Unklarheit. Die der auskommentierten Variablendeklaration *Nautilus*: *Fahrzeugart* beigefügte Fehlermeldung des Compilers *abstract or limited records may not be allocated* steht anscheinend im Widerspruch zu der unmittelbar anschließenden Deklaration der Variablen *AsiaStandard*, *TriCykel*: *Rikscha*; entsprechend der Fehlermeldung dürften diese Variablen nicht allozier-

bar sein. Wie Ihnen die Zusammenstellung der Attribute von Verbunden im vorigen Abschnitt und der Kommentar hinter den Deklarationen zeigen, gilt die Nichtallozierbarkeit für *LIMITED RECORD* Variablen jedoch nur außerhalb des definierenden Moduls, innerhalb des die Typdeklaration enthaltenden Moduls können Variablen solcher Typen uneingeschränkt deklariert werden. Die Formulierung der Fehlermeldung beruht auf der Tatsache, daß die weitaus überwiegende Zahl von Allokationen bzw. Allokationsversuchen in externen Modulen stattfindet, von denen *LIMITED RECORD* Typen importiert werden. Das anschließende Programm *ErbRec3.odc* wird Ihnen zeigen, daß Variablen dieses Typs tatsächlich nicht außerhalb des definierenden Moduls deklarierbar sind.

### Vererbung - Typgebundene Prozeduren

Die Beispielprogramme des 1. bis 7. und auch die ersten drei Programme des 8. Kapitels, in denen Sie die grundlegenden Bausteine der Programmiersprache Component Pascal kennengelernt haben, sind weitgehend in sich abgeschlossene Einheiten, anders verhält es sich jedoch mit dem Modul *TutErbRec2*. Zwar exportiert dieses mit *Start* ebenfalls die übliche Kommandoprozedur, darüber hinaus aber auch vier der fünf deklarierten Verbundtypen. Im vorigen Abschnitt habe ich darauf hingewiesen, daß Modularexporte hauptsächlich dazu dienen, von anderen Modulen verwendet zu werden. Eines der Module, die *TutErbRec2* benutzen, ist das Modul *TutErbRec3*. In dessen Importdeklaration finden Sie in den Zeilen *IMPORT.. ER2 := TutErbRec2* eine Neuigkeit, ein importiertes Modul kann unter Verwendung des Zuweisungszeichens einen eigenen, modulinternen Alias-Namen erhalten. Dies ist bei einer eventuellen Namensänderung des importierten Moduls, das auch als Servermodul (oder kurz Server) bezeichnet wird, eine praktische Möglichkeit, diesen Namen im importierenden Modul, auch Client module (oder nur Client) des importierten Moduls genannt, nicht an vielen verschiedenen Stellen, sondern nur in der Importdeklaration ändern zu müssen.

Alle drei in *TutErbRec3* deklarierten Verbundtypen sind von Basistypen abgeleitete Untertypen. Bei den ersten beiden, *Rikscha* und *Boot* sehen Sie, daß der Basistyp in dem Fall, daß er aus einem Servermodul importiert wird, genauso wie jeder andere Bezeichner mit dem Namen (bzw. dem Alias) dieses Moduls qualifiziert werden muß.

Die versuchten Deklarationen *Rikscha = RECORD (ER2.Rikscha) END* und *BootZeit = RECORD (ER2.BootZeit) END* sind nicht möglich, wie Sie herausfinden werden, wenn Sie die Kommentarklammern um die Deklarationen entfernen und das Modul kompilieren lassen. Bei *BootZeit* liegt dies daran, daß der Basistyp *TutErbRec2.BootZeit* final ist, dagegen weist *TutErbRec2.Rikscha*, der Basistyp von *Rikscha*, das Attribut *LIMITED* auf, er ist dadurch als außerhalb des deklarierenden Moduls nicht erweiterbar gekennzeichnet. Wegen des Attributs *LIMITED* wird der Compiler auch den in der Prozedur *Start* als Kommentar zu findenden Deklarationsversuch *AsiaStandard*: *ER2.Rikscha* abweisen, *LIMITED RECORD* Typen sind,

wie im vorigen Abschnitt erwähnt, außerhalb des deklarierenden Moduls nicht allozierbar. Alle übrigen Variablen Deklarationen sind dagegen auf Grund der Attribute ihrer jeweiligen Typen zugelassen.

Der Anweisungsteil von *Start* dürfte problemlos zu verstehen sein, ich beschränke mich auf die für Sie neuen Prozedurdeklarationen. Betrachten Sie die drei Signaturen *PROCEDURE (VAR r: Rikscha) Text*, *NEW* sowie *PROCEDURE (IN b: Boot) Text*, *NEW*, *EXTENSIBLE* und *PROCEDURE (IN bz: BootZeit) Text*. Als erstes sollte Ihnen die ungewöhnliche Position der Parameterlisten auffallen, sie stehen nicht wie bei den bisherigen Prozeduren hinter den Prozedurnamen, sondern vor diesen unmittelbar hinter dem Schlüsselwort *PROCEDURE*. In Component Pascal wird ein so positionierter Parameter Empfängerparameter (oder auch nur Empfänger; englisch: receiver parameter oder receiver) genannt. Mit dieser Bezeichnung verbindet sich folgender Sachverhalt. Über den Empfängerparameter sind die Prozeduren mit dessen Typ verbunden, sie entsprechen in dieser Hinsicht Prozedurfeldern in Verbundtypen. Während aber einem Prozedurfeld in jedem Einzelfall explizit eine Prozedur zugewiesen werden muß, sind Prozeduren mit Empfängerparametern fest an den Typ des Empfängers gebunden und die Implementierung des auszuführenden Codes ist unmittelbar in ihnen enthalten, derartige Prozeduren heißen deshalb in Component Pascal Terminologie typgebundene Prozeduren. In OOP-Terminologie werden die einem erweiterbaren ADT (einer Klasse) zugeordneten Prozeduren Methoden genannt. Wie Sie gesehen haben, bietet Component Pascal verschiedene Möglichkeiten, Methoden zu implementieren, dagegen kennen klassenbasierte Programmiersprachen nur typgebundene Prozeduren, die Bezeichnungen Methode und typgebundene Prozedur sind in solchen Sprachen Synonyme.

Im Gegensatz zu Prozedurfeldern werden typgebundene Prozeduren innerhalb der Deklaration eines Verbundes nicht explizit aufgeführt, die Zugehörigkeit zu der entsprechenden Klasse ist über den Typ des Empfängerparameters festgelegt. Dagegen enthält die Schnittstelle des Moduls die Signaturen der exportierten typgebundenen Prozeduren ähnlich wie Prozedurfelder innerhalb der Klassen, auf diese Weise ist für den Leser der Schnittstelle der Zusammenhang zwischen einer Klasse und ihren typgebundenen Prozeduren eindeutig erkennbar

*Definition TutErbRec3;*

```
IMPORT
  TutErbRec2;
```

```
TYPE
  Boot = LIMITED RECORD (TutErbRec2.Boot)
    Name: ARRAY 25 OF CHAR;
    (IN b: Boot) Text, NEW, EXTENSIBLE;
  END;
```

```
  BootZeit = LIMITED RECORD (Boot)
    Zeitpunkt: ARRAY 8 OF CHAR;
    (IN bz: BootZeit) Text
  END;
```

```
  Rikscha = RECORD (TutErbRec2.Fahrzeugart)
    Radzahl: INTEGER;
    Antrieb: ARRAY 15 OF CHAR;
    (VAR r: Rikscha) Text, NEW
  END;
```

```
  PROCEDURE Start;
  PROCEDURE Text;
```

```
END TutErbRec3.
```

#### Schnittstelle des Moduls *TutErbRec3*

Den Deklarationen können Sie entnehmen, daß zu unterschiedlichen Klassen gehörende typgebundene Prozeduren gleiche Namen haben können, wegen der Empfängerparameter sind sie ebenso lokal zu ihren Klassen wie Datenfelder der Klassen, die innerhalb der Typdeklarationen stehen und namensgleich mit Feldern anderer Klassen sein dürfen. Außerdem sehen Sie an der Prozedur *Text*, deren Name mit den Namen der drei typgebundenen Prozeduren übereinstimmt, daß der Name einer typgebundenen Prozedur und der Name einer global deklarierten Prozedur identisch sein können, wegen der Lokalität typgebundener Prozeduren zu ihren Klassen besitzen beide Namen unterschiedliche Gültigkeitsbereiche. Dagegen wird der Versuch, innerhalb einer Klasse ein (gewöhnliches) Prozedurfeld mit dem Namen einer existierenden zur Klasse gehörenden typgebundenen Prozedur zu deklarieren, vom Compiler mit einer entsprechenden Fehlermeldung abgewiesen. Die in Kommentarklammern stehende Zeile *Text: PROCEDURE* in der Deklaration der Klasse *Rikscha* macht deutlich, daß der Prozedurfeldname und der Name der typgebundenen Prozedur hier innerhalb des selben Gültigkeitsbereiches kollidieren, typgebundene Prozeduren werden als Felder der Klasse gewertet.

Auch in einem weiteren Punkt werden typgebundene Prozeduren wie Prozedurfelder behandelt, sie können nur von einer Variablen des Empfängertyps aktiviert werden. Ihr Aufruf erfolgt in der Weise, daß dem Namen der jeweils aktivierenden Verbundvariablen wie bei einem Prozedurfeld der Name der typgebundenen Prozedur mit separierendem Punkt angefügt wird, wobei der Compiler die formalen Empfängerparameter (in *TutErbRec3* die Parameter *r* bzw. *b* bzw. *bz*) bei Aufruf einer typgebundenen Prozedur genauso wie Parameter in einem "normalen" Prozeduraufruf durch die Namen der aktuellen (aufrufenden) Variablen ersetzt. Beispielsweise wird in der Kommando-prozedur *Start* das Objekt *AsiaStandard* als Variable der Klasse *Rikscha* deklariert; die gegen Ende der Prozedur zu findende Zeile *AsiaStandard.Text*

aktiviert die typgebundene Prozedur (Rikscha-) *Text*. Bei diesem Aufruf ersetzt der Compiler in (Rikscha-) *Text* alle Vorkommen des formalen Empfängerparameters *r* durch den aktuellen Parameter *AsiaStandard*, genauso wie dies bei anderen Prozedurparametern geschieht. (Typgebundene Prozeduren können übrigens nicht nur wie in diesem Programm einen Empfängerparameter aufweisen, sie können wie andere Prozeduren eine dem Namen folgende Parameterliste besitzen, für die alle Regeln gelten, die Sie im 5. Kapitel im Zusammenhang mit Prozeduren kennengelernt haben; Beispiele derartiger typgebundener Prozeduren finden Sie im 11. Kapitel).

Der Vorteil eines expliziten Empfängerparameters gegenüber der in vielen Programmiersprachen üblichen Anonymität des Empfängers besteht darin, daß auch bei flüchtigem Blick auf den Text der Implementation einer typgebundenen Prozedur der Empfänger in jedem Fall eindeutig identifizierbar ist, es sind keine Verwechslungen zu befürchten. Ein Bezeichner wie *Besitzer* in der ersten Zeile *Out.String(r.Besitzer)* der typgebundenen Prozedur (Rikscha-) *Text* ist durch den vor dem Punkt stehenden Empfänger *r* eindeutig als Feld der Klasse *Rikscha* zu erkennen; ein (im Programm nicht existierender) weiterer, außerhalb der Klasse *Rikscha* deklarierter Bezeichner *Besitzer* gehörte nicht zu dem selben Gültigkeitsbereich wie das Feld *Besitzer*, er könnte deshalb auch innerhalb der Prozedur (Rikscha-) *Text* unabhängig vom Feld *Besitzer* verwendet werden.

Blicken Sie jetzt noch einmal auf die Schnittstelle des Moduls *TutErbRec3*, wird Ihnen auffallen, daß die Klasse *TutErbRec3.Rikscha* das Feld *Besitzer* (anscheinend) nicht enthält. Da *TutErbRec3.Rikscha* durch den Zusatz (*TutErbRec2.Fahrzeugart*) in der Deklaration als Unterklasse der Klasse *TutErbRec2.Fahrzeugart* zu erkennen ist, zeigt Ihnen ein weiterer Blick auf die Schnittstelle des Moduls *TutErbRec2*, daß *Besitzer* bereits als exportiertes Feld in der Basisklasse *TutErbRec2.Fahrzeugart* deklariert wurde, deshalb ohne weiteres Bestandteil der Klasse *TutErbRec3.Rikscha* ist und als Feld dieser Klasse verwendet werden kann.

Es ist zweifellos nicht sehr komfortabel, eine Klassenhierarchie in dieser Weise auch nur über zwei Stufen zurückzuerfolgen. Der Benutzer einer Klassenbibliothek mit Hierarchien, die sich eventuell über viele Stufen erstrecken, würde die Umständlichkeit wahrscheinlich hin und wieder mit einigen passenden Bemerkungen kommentieren. Daher gibt es außer den bisher verwendeten Schnittstellen, die Sie sich über den Menüpunkt *Info* → *Client Interface* anzeigen lassen können, eine zweite Möglichkeit, die "flache" oder erweiterte Schnittstellenbeschreibung. Diese können Sie sich anzeigen lassen, indem Sie unter dem Menüpunkt *Info* → *Interface...* zuerst das Kontrollfeld *Flat Interface* aktivieren und anschließend auf *Show Client Interface* klicken

*DEFINITION TutErbRec3;*

*IMPORT TutErbRec2;*

*TYPE*

```

Boot = LIMITED RECORD (TutErbRec2.Boot(TutErbRec2.Fahrzeugart))
  (* TutErbRec2.Fahrzeugart *) Besitzer: ARRAY 25 OF CHAR;
  (* TutErbRec2.Fahrzeugart *) Gewicht: INTEGER;
  (* TutErbRec2.Fahrzeugart *) Preis: REAL;
  (* TutErbRec2.Boot *) Fahrzeugsorte: ARRAY 15 OF CHAR;
  (* TutErbRec2.Boot *) Riemen: INTEGER;
  (* TutErbRec2.Boot *) Segel: BOOLEAN;
  (* TutErbRec2.Boot *) Antrieb: ARRAY 15 OF CHAR;
  Name: ARRAY 25 OF CHAR;
  (IN b: Boot) Text, NEW, EXTENSIBLE
END;

```

```

BootZeit = LIMITED RECORD (Boot(TutErbRec2.Boot(TutErbRec2.Fahrzeugart)))
  (* TutErbRec2.Fahrzeugart *) Besitzer: ARRAY 25 OF CHAR;
  (* TutErbRec2.Fahrzeugart *) Gewicht: INTEGER;
  (* TutErbRec2.Fahrzeugart *) Preis: REAL;
  (* TutErbRec2.Boot *) Fahrzeugsorte: ARRAY 15 OF CHAR;
  (* TutErbRec2.Boot *) Riemen: INTEGER;
  (* TutErbRec2.Boot *) Segel: BOOLEAN;
  (* TutErbRec2.Boot *) Antrieb: ARRAY 15 OF CHAR;
  Name: ARRAY 25 OF CHAR;
  Zeitpunkt: ARRAY 8 OF CHAR;
  (IN bz: BootZeit) Text
END;

```

```

Rikscha = RECORD (TutErbRec2.Fahrzeugart)
  (* TutErbRec2.Fahrzeugart *) Besitzer: ARRAY 25 OF CHAR;
  (* TutErbRec2.Fahrzeugart *) Gewicht: INTEGER;
  (* TutErbRec2.Fahrzeugart *) Preis: REAL;
  Radzahl: INTEGER;
  Antrieb: ARRAY 15 OF CHAR;
  (VAR r: Rikscha) Text, NEW
END;

```

*PROCEDURE Start;*  
*PROCEDURE Text;*

*END TutErbRec3.*

Erweiterte Schnittstelle des Moduls *TutErbRec3*

Diese erweiterte Schnittstelle zeigt nicht nur die direkten Exporte des Moduls an, sondern zusätzlich die gesamte Importhierarchie. Sie finden in der erweiterten Schnittstelle des Moduls *TutErbRec3* in der Klasse *Rikscha* das vermißte Feld *Besitzer* mit der qualifizierten Angabe der exportierenden Basisklasse *TutErbRec2.Fahrzeugart*. Entsprechendes gilt für alle übrigen Bezeichner, die auf Grund der Importhierarchie des Moduls in diesem zur Verfügung stehen. An den Klassen *Boot* und *BootZeit* sehen Sie außerdem, daß bestehende Klassenhierarchien bis hin zur Basisklasse sowohl innerhalb eines Moduls als auch über Modulgrenzen hinweg vollständig qualifiziert dargestellt werden.

Auf zwei wichtige Aspekte möchte ich Sie aufmerksam machen. Empfängerparameter von typgebundenen Prozeduren, die an Klassen des Typs *RECORD* gebunden sind, müssen mit den Schlüsselwörtern *VAR* oder *IN* deklariert werden. Die Schlüsselwörter *VAR* und *IN* haben auch bei Empfängerparametern die Bedeutungen, die Sie bereits früher im Zusammenhang mit strukturierten Prozedurparametern kennengelernt haben, ein formaler *VAR*-Empfängerparameter deklariert den aktuellen Parameter als veränderbar, seine Felder können nicht nur gelesen, sondern auch geschrieben werden, Sie sehen dies bestätigt in der Zeile *r.Radzahl := 3* der typgebundenen Prozedur (Rikscha-) *Text*. Dagegen können *IN*-Empfängerparameter innerhalb einer typgebundenen Prozedur nur gelesen werden, sie sind schreibgeschützt; in der typgebundenen Prozedur (Boot-) *Text* quittiert der Compiler den (auskommentierten) Versuch, in der Zeile *b.Fahrzeugsorte := "Schlitten"* ein Feld des *IN*-Empfängerparameters *b* zu verändern mit der dahinter stehenden Fehlermeldung.

Als zweites sehen Sie an der Prozedur (BootZeit-) *Text*, daß typgebundene Prozeduren überschrieben werden können. *BootZeit* ist ein von *Boot* abgeleiteter Typ, eine Unterklasse. Unterklassen erben von ihren Basisklassen nicht nur die Datenfelder, sondern auch die typgebundenen Prozeduren. Vielfach möchte man aber für die Objekte der Unterklasse außer den Werten der Datenfelder auch die Prozeduren ändern, um sie den geänderten Notwendigkeiten der Unterklasse anzupassen. Aus diesem Grund ist es möglich, in einer Unterklasse die geerbten typgebundenen Prozeduren neu zu definieren, sie zu überschreiben. Eine überschriebene typgebundene Prozedur erkennen Sie daran, daß ihr etwas wesentliches fehlt, das Component Pascal Schlüsselwort *NEW*. Dies ist eins von vier Schlüsselwörtern, die in Verbindung mit typgebundenen Prozeduren Attribute der Prozeduren genannt werden und folgende Eigenschaften haben

Attribut	Bedeutung
- (kein Attribut)	Methode kann aufgerufen, aber nicht überschrieben werden (finale Methode)
<b>EXTENSIBLE</b>	Methode kann aufgerufen und überschrieben werden
<b>ABSTRACT</b>	Methode kann nicht aufgerufen, muß überschrieben werden
<b>EMPTY</b>	Methode kann aufgerufen und überschrieben werden

Die Attribute <sup>11</sup> einer typgebundenen Prozedur stehen, durch Komma separiert, hinter der Parameterliste der Prozedur (bei einer Funktion hinter deren Rückgabotyp). Eine neu eingeführte typgebundene Prozedur muß das Attribut *NEW* erhalten, dadurch ist sie für den Compiler und für den Leser einer Modulschnittstelle eindeutig einer Klasse des aktuellen Moduls zugeordnet, beide müssen nicht innerhalb einer eventuell tief strukturierten Klassenhierarchie nach einer dort nicht vorhandenen Basisprozedur suchen. Ebenso wissen beide, daß in einer Superklasse eine Basisprozedur existieren muß, wenn einer typgebundenen Prozedur das Attribut *NEW* fehlt, in diesem Fall handelt es sich um eine redefinierte Prozedur, die Basisprozedur wurde überschrieben.

Allerdings ist nicht jede typgebundene Prozedur überschreibbar, dies ist nur möglich, wenn die Prozedur nicht final ist. Nicht finale typgebundene Prozeduren sind daran erkennbar, daß sie (gegebenenfalls zusätzlich zum Attribut *NEW*) ein weiteres Attribut haben, *EXTENSIBLE*, *ABSTRACT* oder *EMPTY*. Im Modul *TutErbRec3* finden Sie außer dem Attribut *NEW* nur das Attribut *EXTENSIBLE*, die anderen beiden Attribute bitte ich, im Augenblick zu ignorieren, sie werden in späteren Programmen erklärt werden.

Die erste der drei typgebundenen Prozeduren des Moduls hat die Signatur *PROCEDURE (VAR r: Rikscha) Text, NEW*, sie ist dadurch als neu eingeführt und final deklariert, sie kann nicht überschrieben werden. Die zweite mit der Signatur *PROCEDURE (IN b: Boot) Text, NEW, EXTENSIBLE* ist ebenfalls als neu deklariert, darüber hinaus aber als erweiterbar, sie kann in einer Unterklasse von *Boot* überschrieben werden. Die dritte typgebundene Prozedur mit der Signatur *PROCEDURE (IN bz: BootZeit) Text* besitzt keine sichtbaren Attribute, sie ist dadurch einerseits als final kenntlich, andererseits zeigt das Fehlen des Attributs *NEW* an, daß eine in der Basisklasse von *BootZeit* existierende Prozedur gleichen Namens redefiniert wird.

Die Redefinition einer typgebundenen Prozedur ist nur unter Beibehaltung des in der Basisklasse festgelegten Namens und der Parameterlisten möglich, typgebundene Prozeduren sind mit ihren vollständigen Signaturen Bestandteil der Klassendefinitionen, sie sind Prozedurkonstanten im Gegensatz zu gewöhnlichen Prozedurfeldern eines Verbundtyps, die Prozedurvariablen repräsentieren. Hierin liegt, neben der geänderten Syntax, ein wesentlicher Unterschied zwischen einer typgebundenen Prozedur und einem Prozedurfeld, das zwar ebenso wie eine typgebundene Prozedur einen festen Namen hat, dem aber jederzeit Prozeduren mit beliebigen Namen zugewiesen werden können, sofern deren Signaturen mit der Deklaration des Prozedurfeldes kompatibel sind, ich werde später genauer auf diesen Unterschied eingehen.

Die typgebundene Prozedur (BootZeit-) *Text* macht Sie mit zwei weiteren Neuigkeiten bekannt. Zum einen sehen Sie, daß der Empfängerparameter einer überschriebenen typgebundenen Prozedur exakt mit dem der Basisprozedur übereinstimmen muß, ein schreibgeschützter *IN*-Empfänger kann in einer redefi-

<sup>11</sup> Eine Zusammenstellung der Attribute typgebundener Prozeduren und ihrer Bedeutungen finden Sie im Anhang B.3.2.2.

nierten Prozedur nicht in einen VAR-Empfänger zurückverwandelt werden, ebensowenig ist es möglich, einen VAR-Empfängerparameter in einer überschreibenden Prozedur zu einem schreibgeschützten *IN*-Empfängerparameter zu machen, in beiden Fällen besteht der Compiler auf Einhaltung der ursprünglichen Deklaration der typgebundenen Prozedur.

Als zweite Neuigkeit sehen Sie im *ELSE*-Teil der *IF*-Abfrage in der Zeile *bz.Text* den Aufruf der typgebundenen Prozedur *Text*, den man als rekursiven Aufruf bezeichnen kann. Das Zeichen "*^*" im Anschluss an den Namen *Text* stellt einen "Zeiger" dar, der nicht auf die aktuelle Prozedur, sondern auf deren Basisprozedur weist, der Aufruf *bz.Text* aktiviert also die eigentlich überschriebene Prozedur der Basisklasse *Boot*, ein solcher Aufruf wird "super-call" genannt (weiteres dazu im 10. Kapitel).

Die Tatsache, daß die typgebundene Prozedur *Text* der Klasse *TutErbRec3.Boot* in der Unterklasse *TutErbRec3.BootZeit* überschrieben worden ist, findet sich in der erweiterten Schnittstelle des Moduls *TutErbRec3* wieder. Die Schnittstelle der Klasse *TutErbRec3.BootZeit* enthält außer der Darstellung der klasseeigenen typgebundenen Prozedur (*IN bz: BootZeit*) *Text* auch die Darstellung der Basisprozedur (*IN b: Boot*) *Text*, *NEW*, *EXTENSIBLE*, der Leser der Schnittstelle wird auf diese Weise informiert, daß es sich bei *Text* um eine überschriebene Prozedur handelt, die jeweiligen Empfängerparameter geben darüber hinaus eindeutig Auskunft über die Namen der die Prozeduren enthaltenden Klassen.

Zum Schluß möchte ich Sie auf einen wesentlichen Unterschied aufmerksam machen. Neben dem erwähnten Prozeduraufruf *AsiaStandard.Text* finden Sie am Ende der Prozedur *Start* drei weitere, identisch strukturierte Aufrufe, *Nautilus.Text*, *MolaMola.Text* und *Stachelrochen.Text*. Während aber die ersten beiden ebenso wie *AsiaStandard.Text* Aufrufe von typgebundenen Prozeduren sind, handelt es sich bei *Stachelrochen.Text* um einen gewöhnlichen Prozeduraufruf, denn der Typ *TutErbRec2.BootZeit* der Verbundvariablen *Stachelrochen* enthält als einziges exportiertes Feld das Prozedurfeld *Text*, dem vor der Aktivierung eine Prozedur zugewiesen werden muß. Man kann dies so interpretieren, daß (gewöhnliche) Prozeduren in Verbunden jeweils an eine einzelne Variable - ihr Objekt - gebunden sind; einem anderen Objekt der selben Klasse kann eine andere kompatible Prozedur zugewiesen werden. Dagegen sind typgebundene Prozeduren nicht an einzelne Objekte, sondern an die Klasse selbst gebunden, sie sind Bestandteile der Klasse, weshalb sie nicht initialisiert werden müssen.

Auch in diesem Programm sollten Sie, wie schon früher, die Bildschirmausgabe der Prozedur *Start* mit Papier und Graphitstift erarbeiten, bevor Sie das Kommando ausführen lassen; ebenso sollten Sie die Auswirkung jeder der beiden in Kommentarklammern stehenden Zeilen *MolaMola.Fahrzeugsorte := "Fahrrad"* bzw. *MolaMola.Fahrzeugsorte := "Geisterschiff"* auf den ausgegebenen Text selbst herauszufinden versuchen, wobei Sie insbesondere auf die unterschiedlichen Ergebnisse des Aufrufs der Methode *MolaMola.Text* in Abhängigkeit von dem aktuell in *MolaMola.Fahrzeugsorte* gespeicherten Namen achten sollten. Erst im Anschluss sollten Sie (abwechselnd) die Kommentarklammern entfernen und die Bildschirmausgabe mit Ihren eigenen Ergebnissen vergleichen.