Gliederung des Kapitels 1.4

1.4/1.5 Programmierung (und die Sprache Ada 95)

~~1.4.1 Blöcke, Deklarationen und Ausnahmen~~
~~1.4.2 Prozeduren und Funktionen~~
~~1.4.3 Moduln~~
~~1.4.4 Polymorphie~~
1.4.5 Vererbung
1.4.6 Abstrakte und konkrete Datentypen
1.4.7 Objekte
1.4.8 Grundprinzipien, Paradigmen der Programmierung

1.4.5 Vererbung

Die Welt ist voller Hierarchien und Beziehungen. In einer Modellierung soll sich dies widerspiegeln.
Im Bereich der Datentypen (und der Objekte, siehe später) verwendet man hierfür die Vererbung. Die Idee ist einfach:

1. Hat man einen Datentyp deklariert, so kann man durch Hinzufügen weiterer Komponenten hieraus weitere Datentypen ableiten ("Spezialisierung", Unterdatentypen).
2. Liegen mehrere Datentypen vor, die gewisse Komponenten gemeinsam haben, so kann man diese Gemeinsamkeiten als einen eigenen Datentyp herausziehen ("Generalisierung", Oberdatentyp) und die gegebenen Datentypen zu gemeinsamen Unterdatentypen des neuen Oberdatentyps machen.

Nun braucht man nur noch Sprachelemente, um die Erweiterung oder Einschränkung von Typen zu beschreiben.

1.4.5.1 Typerweiterung oder Spezialisierung: Wir formulieren dies sofort in Ada. Dort muss ein Datentyp, der später erweitert werden soll, mit dem Zusatz "tagged" deklariert werden (engl. tag = Etikett, Zusatz). Wir betrachten einen Typ zur Beschreibung einiger Attribute eines Fahrzeugs:

```
type Fahrzeug is tagged record
        Hersteller: array (1..30) of character;
        Höchstgeschwindigkeit: positive;
        Neupreis: delta 0.01 range 0.0 .. 50_000_000.0;
end record;
```


Will man nun Attribute für einen Bus zusammenfassen, so reicht es, obige Deklaration einfach zu erweitern mittels

```
type Bus is new Fahrzeug with record
        Sitzplätze, Stehplätze: positive;
        Wendekreis: delta 0.001 range 0.0 .. 40.0;
        Achsenzahl: positive;
end record;
```

Dies entspricht der folgenden Deklaration:

```
type Bus is record
        Sitzplätze, Stehplätze: positive;
        Wendekreis: delta 0.001 range 0.0 .. 40.0;
        Achsenzahl: positive;
        Hersteller: array (1..30) of character;
        Höchstgeschwindigkeit: positive;
        Neupreis: delta 0.01 range 0.0 .. 50_000_000.0;
end record;
```


Sprechweisen:

Man sagt, "Bus" ist ein aus "Fahrzeug" abgeleiteter Typ.

Man sagt, die Komponenten "Hersteller", "Neupreis" und "Höchstgeschwindigkeit" wurden vom Typ "Fahrzeug" auf oder an den Typ "Bus" vererbt.

Man nennt diesen Vorgang, Eigenschaften an andere Einheiten weiterzureichen, "Vererbung" (engl. inheritance).

Man sagt, "Bus" ist "Spezialisierung" oder "Erweiterung" oder Untertyp von oder zu "Fahrzeug".

Man sagt, "Fahrzeug" ist "Generalisierung" oder Obertyp von oder zu "Bus".

Man nennt die Obertypen auch (direkte) "Eltern", die Untertypen (direkte) "Kinder". Setzt man dies transitiv fort, so spricht man von "Vorfahren" bzw. "Nachkommen".

Es werden also alle Deklarationen des Typs Fahrzeug auf den Typ Bus vererbt.

(Hinweis: Diese Vererbung wird bei Paketen und Objekten, wo nicht nur Erweiterungen, sondern auch Umdefinitionen erfolgen, besonders nützlich.)

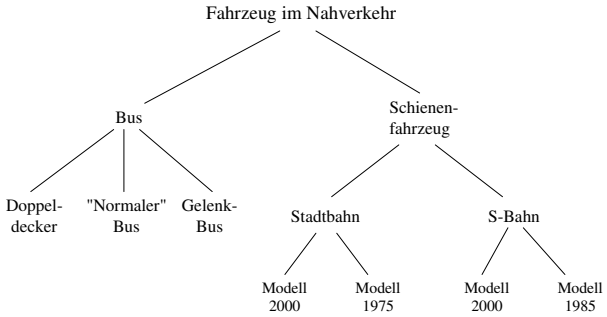
Das Gleiche kann man für eine S-Bahn tun:

```
type SBahn is new Fahrzeug with record
        Sitzplätze, Stehplätze: positive;
        Waggonlänge: delta 0.001 range 0.0 .. 400.0;
        Achsenzahl: positive;
end record;
```

Hieraus kann man weitere Typen ableiten (in Ada muss man explizit angeben, wenn man nichts hinzufügen will):

```
type Normaler_Bus is new Bus with null record;

type Doppeldecker is new Bus with record
        Höhe: float;
end record;

type Gelenkbus is new Bus with record
        Länge: float;
end record;

type Stadtbahn is new Fahrzeug with record
        Sitzplätze, Stehplätze: positive;
        AnzahlKartenEntwerter: positive;
end record;
```

Hinweise: Die Eigenschaft "tagged" vererbt sich automatisch auf alle abgeleiteten Typen. Ableitbare Typen erkennt man an "tagged" oder "with" in ihrer Definition.



Aufbau einer Vererbungs-Hierarchie

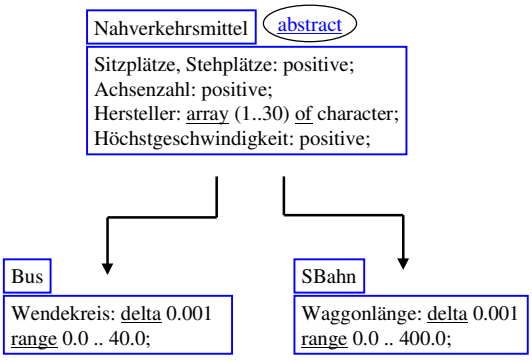
1.4.5.2 Zusammenfassen oder Generalisierung: Wir formulieren dies ebenfalls sofort in Ada. Gegeben seien:

```
type Bus is record
        Sitzplätze, Stehplätze: positive;
        Wendekreis: delta 0.001 range 0.0 .. 40.0;
        Achsenzahl: positive;
        Hersteller: array (1..30) of character;
        Höchstgeschwindigkeit: positive;
end record;

type SBahn is new Fahrzeug with record
        Hersteller: array (1..30) of character;
        Höchstgeschwindigkeit: positive;
        Sitzplätze, Stehplätze: positive;
        Waggonlänge: delta 0.001 range 0.0 .. 400.0;
        Achsenzahl: positive;
end record;
```

In einen Obertyp herausziehen

Wir deklarieren also einen neuen Typ "Nahverkehrsmittel", aus dem wir die Typen Bus und SBahn ableiten können:

```
type Nahverkehrsmittel is abstract tagged record
        Sitzplätze, Stehplätze: positive;
        Achsenzahl: positive;
        Hersteller: array (1..30) of character;
        Höchstgeschwindigkeit: positive;
end record;

type Bus is new Nahverkehrsmittel with record
        Wendekreis: delta 0.001 range 0.0 .. 40.0;
end record;

type SBahn is new Nahverkehrsmittel with record
        Waggonlänge: delta 0.001 range 0.0 .. 400.0;
end record;
```

Hier tritt das Sprachelement "abstract" auf. Es bedeutet, dass man einen Datentyp deklariert hat, den man nur für die Vererbung verwendet, zu dem man aber keine Variablen oder formalen Parameter deklarieren darf. Bei dem "abstract"-Typ "Nahverkehrsmittel" sind also Deklarationen folgender Art *verboten*

```
X: Nahverkehrsmittel;
function F (A: in Nahverkehrsmittel) return Boolean; ...
```

In unserem Beispiel ist der Zusatz "abstract" nicht notwendig gewesen, doch sollte man ihn stets verwenden, wenn man den jeweiligen Typ nur in der "Vererbungs-Hierarchie" einsetzt.

1.4.5.3 Umdefinitionen (redefinition)

Bei der Vererbung kann man verebte Komponeten neu definieren. Die vererbten Komponenten sind dann wegen der Sichtbarkeitsregel automatisch ausgeblendet (können aber bei allgemeinen Strukturen wie packages über Qualifizierung mit Punkt-Notation dennoch angesprochen werden). Beispiel:

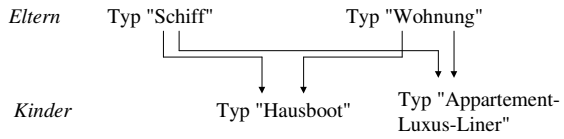
```
type wenig is 1..20;

type Kleinbus is new Bus with record
        Sitzplätze: wenig;
end record;
```

Die ererbte Komponente "Sitzplätze" wird durch diese neue Komponente "Sitzplätze" überschrieben (engl.: overridden).

1.4.5.4 Mehrfachvererbung

Nach dem bisherigen Konzept kann ein Datentyp höchstens einen (direkten) Obertyp besitzen. Man spricht von Einfach-Vererbung (single inheritance).

Manchmal sollen aber Eigenschaften mehrerer Datentypen an einen Datentyp weitergereicht werden. Diesen Vorgang nennt man Mehrfach-Vererbung (multiple inheritance).



In Ada ist Mehrfachvererbung nicht zugelassen. Die Gründe hierfür liegen zum einen in Problemen der Implementierung, zum anderen in der Fehleranfälligkeit bei der Verwendung mehrfacher Ableitungen, vor allem wenn sie erst zur Laufzeit nachvollzogen werden können.

Die Programmierer sind hier selbst verantwortlich, wenn durch Vererbung gleicher Namen, die aber verschiedene Typen in den unterschiedlichen Eltern bezeichnen, Namens-konflikte oder andere Mehrdeutigkeiten entstehen. In Java ist Mehrfachvererbung erlaubt.
Generell sollte man die Mehrfachvererbung sparsam und "sehr kontrolliert" verwenden.

Diese Idee der Vererbung ist nicht auf Datentypen beschränkt. Wir können sie auch für Moduln einsetzen:

Wenn wir einen Modul "Listenverarbeitung" mit der Definition des Datentyps "Liste" und hierauf zugelassenen Operationen und Prozeduren eingeführt haben, so können wir hieraus einen Datentyp "Stack" durch Erweiterung gewinnen, indem wir weitere Prozeduren und Funktionen hinzunehmen (empty und isempty können wir übernehmen, top, pop und push fügen wir unter Verwendung der bereits definierten Listen-Operatioenn hinzu; ggf. definieren wir auch "einfügen" zu "push" um).

Aber dies haben wir bereits getan mit dem Sprachelement with, mit dem man die sichtbaren Teile eines Pakets in eine andere Programmeinheit übernehmen kann.