Datenabstraktion

- **Prozeß- und Datenabstraktion**
- **Objektmodule (Datenkapselung)**
- **Abstrakte Datentypen**

Programmieren im Großen

- **Programmieren im Kleinen**
  - befaßt sich mit der Konstruktion eines ***Programms***. Wir haben bisher die dazu notwendigen Konzepte kennengelernt:
    - Daten- und Kontrollstrukturen,
    - Typen,
    - Prozeduren und Funktionen.
- **Programmieren im Großen**
  - für die Entwicklung großer Softwaresysteme müssen weitere Konzepte hinzukommen. Das vorrangige Problem ist, die ***Komplexität*** großer Softwaresysteme zu beherrschen.
  - Die gewählte Lösung heißt ***Abstraktion***.
- **Für den modularen Softwareentwurf sind zwei Abstraktionskonzepte entscheidend:**
  - ***Prozeßabstraktion***,
  - ***Datenabstraktion***.

Prozeß- und Datenabstraktion

Abstraktion: Prozeß, Daten

- **Prozeßabstraktion (auch algorith. Abstraktion):**
  - Funktionen und Prozeduren werden zu ***abstrakten*** Konzepten.
  - Module können gleichartige zusammenfassen: funktionaler Modul.
  - Bekannt sind nur die ***Eingabe***- und ***Ausgabegrößen***, aber nicht die Implementation.
  - z.B.: Für eine mathematische Funktion ist nur wesentlich, daß sie ein korrektes Ergebnis liefert, aber nicht wie dies geschieht
- **Datenabstraktion:**
  - Die ***Details*** der verwendeten Daten werden verborgen.
  - Schließt konzeptionell die Prozeßabstraktion für die Zugriffsoperationen ein.
  - Beispiel:
    - Für eine Liste ist wichtig, daß sie ein Behälter für Elemente ist und daß bestimmte Zugriffsoperationen erlaubt sind,
    - aber nicht, wie die Elemente der Liste gespeichert und bearbeitet werden.
  - Eine weitere von Datenabstraktion abstrahiert auch von der ***Art der Elemente***, die in der Liste gespeichert werden.

Prozeß- und Datenabstraktion

Information Hiding

- **Prinzip:**
  - Es werden nur die Informationen zur Verfügung gestellt, die ***absolut notwendig*** sind!
  - Alle anderen, insbesondere die ***wichtigen Informationen*** werden ***versteckt***!
  - Der Zugriff auf diese Informationen geschieht über "***Vermittler***".
- **Beispiel: "Offene Bank"**
  - Funktioniert nicht, weil
    - die Buchhaltung nicht klappt (Änderungen werden nicht jedem bekannt sein)
    - doch gestohlen wird (was mißbraucht werden kann, wird mißbraucht)
- **Deshalb:**
  - Das wertvolle wird versteckt (Geld -Tresor)
  - Es werden Vermittler eingesetzt (Mitarbeiter, Automaten)

D. Parnas, 1972

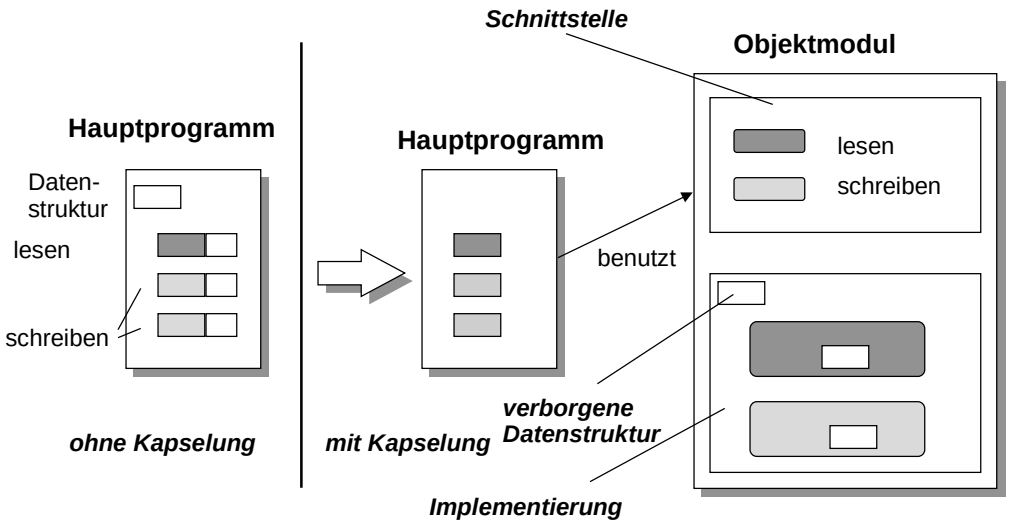
Objekt-modul

Datenkapsel

- **Die zentrale Idee:**
  - ***Trenne*** die konkrete Realisierung (i.e. Implementation) einer Datenstruktur von ihren sichtbaren Eigenschaften.
- **Merkmale:**
  - Eine Datenstruktur wird in einem Modul ***eingekapselt***.
  - An der Schnittstelle des Moduls sind nur ***Operationen sichtbar***, die den allgemeinen Umgang mit der Datenstruktur beschreiben.
  - Die Datenstruktur selbst ist ***verborgen***.
- **Jedes Objektmodul**
  - beschreibt und realisiert nur eine ***einzige sog. abstrakte Datenstruktur***.
- **Kapselung von Daten**
  - ist ein zentraler Denkansatz und ein wesentliches ***Entwurfsprinzip***

Objekt-modul

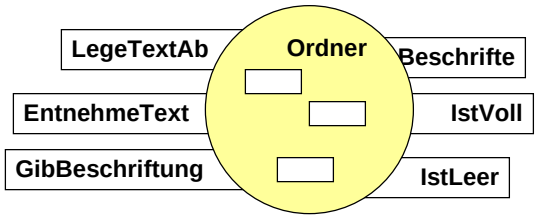
Schema: Objektmodul



Objekt-modul

Beispiel: Das Objektmodul Ordner

- **"Ordner " als Konzept**
  - ***enthält*** Texte
  - Texte können ***abgelegt*** und ***entnommen*** werden
  - ein Ordner kann ***beschriftet*** werden
  - ein Ordner kann ***leer*** oder ***voll*** sein
- **Ein Objektmodul**
  - realisiert ein ***fachliches Konzept*** (z.B. das Konzept "Ordner")
  - als ***genau eine abstrakte Datenstruktur***
  - hat ***Zustand*** (Gedächtnis)



Objekt-modul

Realisierung Objektmodul

Objektmodul Ordner ist ein Behälter für Daten des Typs TEXT

```
INTERFACE Ordner;

PROCEDURE LegeTextAb (t : TEXT);
PROCEDURE EntnehmeText () : TEXT ;
PRODEDURE IstVoll () : BOOLEAN ;
PROCEDURE IstLeer () : BOOLEAN ;
PROCEDURE Beschrifte (t : TEXT);
PROCEDURE GibBeschriftung () : TEXT;
PROCEDURE Initialisiere ();

END Ordner.
```

Abstrakte Beschreibung des fachlichen Konzepts Ordner

Das eine Objekt Ordner soll mehrfach pro Programmablauf verwendbar sein

Objekt-modul

Verwendung eines Objektmoduls 1

```
MODULE Ordner_Test EXPORTS Main;

IMPORT Ordner,  SIO;

BEGIN
  Ordner. Initialisiere();
  Ordner.Beschrifte ("Kleine Gedichte");
  Ordner.LegeTextAb ("Nicht immer sind bequeme Stuehle ...");
  Ordner.LegeTextAb ("Herr von Ribeck auf Ribeck ...");
  Ordner.LegeTextAb ("Von drauss vom Walde komm ich her ...");
  SIO.PutLine (Ordner.GibBeschriftung());
  SIO.PutLine ("--------------------------");
  SIO.PutLine (Ordner.EntnehmeText());
  SIO.PutLine (Ordner.EntnehmeText());
  SIO.PutLine (Ordner.EntnehmeText());
  Ordner.Initialisiere();
  ...
END Ordner_Test.
```

Diskussion: Wie darf ein Objekt-modul verwendet werden?

Objekt-modul

Verwendung eines Objektmoduls 2

```
INTERFACE Ordner;
PROCEDURE LegeTextAb (t : TEXT);
PROCEDURE EntnehmeText () : TEXT ;
PRODEDURE IstVoll () : BOOLEAN ;
PROCEDURE IstLeer () : BOOLEAN ;
PROCEDURE Beschrifte (t : TEXT);
PROCEDURE GibBeschriftung () : TEXT;
PROCEDURE Initialisiere ();
```

- **Feststellung:**
  - `LegeTextAb` und `Entnehme` sind ***nicht in jedem Zustand*** des Ordners sinnvoll:
    - Ein voller Ordner kann keine weiteren Texte aufnehmen; ein leerer keine herausgeben.
  - Solche Operationen sind nur in bestimmten Situationen (abhängig von bestimmten Bedingungen) sinnvoll.
  - Um den sicheren Umgang mit einem solchen Objekt zu gewährleisten, werden an der Schnittstelle entsprechende ***Testfunktionen*** wie `IstLeer` oder `IstVoll` zur Verfügung.

Objekt-modul

Einsatz der Testfunktionen

Prüfen der Vorbedingung

Das Objektmodul **Ordner** wird vor der ersten Verwendung initialisiert, d.h. der Inhalt wird gelöscht

```
Ordner.Initialisiere();

IF Ordner.IstVoll() THEN
  SIO.PutLine ("Ordner ist bereits voll");
ELSE
  Ordner.LegeTextAb ("Nicht immer sind bequeme Stuehle ...");
END;

IF Ordner.IstLeer() THEN
  SIO.PutLine ("Ordner ist leer");
ELSE
  t := Ordner.EntnehmeText();
END;
```

Vor Entnahme wird der Zustand des Objektmoduls Ordner geprüft und entsprechend gehandelt.

Objekt-modul

Entwurfskonzept

- **Um ein fachliches Konzept als Objektmodul zu realisieren, stellen wir *drei Arten von Operationen* zur Verfügung.**
- **Prozeduren:**
  - ***verändern*** den ***Zustand*** des Objekts. Meist sind sie von Vorbedingungen abhängig, d.h. sie können nicht in jedem Zustand ausgeführt werden.
- **Funktionen:**
  - liefern Informationen, ***ohne*** den Objektzustand nach außen sichtbar ***zu verändern***.
  - Fachliche Funktionen:
    - liefern ***fachliche Informationen*** und sind vom Objektzustand abhängig.
  - Testfunktionen:
    - ***prüfen*** den Objektzustand und werden zum Prüfen der Vorbedingungen verwendet.

Objekt-modul

Entwurfskonzept-Beispiel: Ordner

```
INTERFACE Ordner;
PROCEDURE LegeTextAb (t : TEXT);
PROCEDURE EntnehmeText () : TEXT ;
PRODEDURE IstVoll () : BOOLEAN ;
PROCEDURE IstLeer () : BOOLEAN ;
PROCEDURE Beschrifte (t : TEXT);
PROCEDURE GibBeschriftung () : TEXT;
PROCEDURE Initialisiere ();
```

- **Prozeduren:**
  - `Initialisiere`, `LegeTextAb`, `EntnehmeText`, `Beschrifte` sind verändernde Prozeduren.
- **Fachliche Funktionen:**
  - `GibBeschriftung` ist eine fachliche Funktion; sie verändert im Gegensatz zu `EntnehmeText` nicht den Ordnerzustand.
- **Textfunktionen:**
  - `IstLeer` und `IstVoll`

Objekt-modul

Ein Blick ins Innere - 1

```
MODULE Ordner EXPORTS Ordner;

CONST MaxTexte = 20;
TYPE Fassungsvermoegen = [1 .. MaxTexte];
TYPE Texte = ARRAY Fassungsvermoegen OF TEXT;
VAR ordnerInhalt : Texte;
    anzahlTexte : CARDINAL;
    beschriftung : TEXT := "";

PROCEDURE LegeTextAb (t : TEXT)=
BEGIN
  anzahlTexte := anzahlTexte + 1;
  ordnerInhalt[anzahlTexte] := t;
END LegeTextAb;

PROCEDURE EntnehmeText () : TEXT =
VAR t : TEXT;
BEGIN
  t := ordnerInhalt[anzahlTexte];
  ordnerInhalt[anzahlTexte] := "";
  anzahlTexte := anzahlTexte - 1;
  RETURN t;
END EntnehmeText;
```

Es wird ein Array verwendet

Es wird der zuletzt abgelegte Text entnommen

Objekt-modul

Ein Blick ins Innere - 2

```
PROCEDURE IstVoll () : BOOLEAN =
BEGIN
  RETURN (anzahlTexte = MaxTexte);
END IstVoll
...

PROCEDURE Beschrifte (t : TEXT)=
BEGIN
  beschriftung := t;
END Beschrifte;
...

PROCEDURE Initialisiere () =
BEGIN
  FOR i := FIRST(Fassungsvermoegen) TO LAST(Fassungsvermoegen) DO
    ordnerInhalt[i] := "";
  END;
  anzahlTexte := 0;
END Initialisiere;

BEGIN
  Initialisiere ();
END Ordner.
```

`IstVoll` verwendet die Variable `anzahlTexte`

Objekt-modul

Zusammenfassung Objektmodul

- **Eigenschaften eines Objektmoduls:**
  - Das Modul verwaltet seine Daten ***selbst***.
  - Die interne Repräsentation der Daten ist vollständig ***verborgen***.
  - Die interne Repräsentation ist ***austauschbar***.
  - Zur Laufzeit existiert immer ***nur ein Exemplar*** eines Objektmoduls, d.h. es können z.B. nicht mehrere Ordner erzeugt werden.
  - Das Objektmodul kann von ***mehreren*** anderen "Kunden" verwendet werden.
  - Dadurch kann z.B. ein ***gemeinsamer*** Ordner verwaltet werden.

Abstrakte Datentypen

Module als Sprachelement

- **Ein Modul kann zwar**
  - ***definiert*** und zur Laufzeit von anderen Modulen ***importiert*** werden,
  - aber es ist kein ***primäres*** Sprachelement (first class), d.h.
  - ein Modul kann ***nicht*** wie ein Typ ***zur Deklaration*** von Bezeichnern verwendet werden.

  - `o1, o2 : Ordner (* geht nicht, da Ordner Modul ist *)`
- **Folge:**
  - es gibt ***nur ein Exemplar*** eines Objektmoduls.
- **Problem:**
  - Es werden oft mehrere Exemplare eines durch ein Objektmodul realisierten Objekts gebraucht.
- **Lösungsansatz:**
  - Wir formulieren ***einen Typ*** für die im Modul beschriebenen Objekte.

Abstrakte Datentypen

Konzept Abstrakter Datentyp

- **Kann als *Verallgemeinerung* des Objektmoduls (Datenkapsel) betrachtet werden.**
- **Anstatt eines Objektes wird ein Typ (für diese Objekte) definiert.**
- **Betrachten wir den ADT als (formale) Spezifikation eines Typs,**
  - dann entwerfen wir ihn durch Angabe von
  - ***Typnamen***
  - ***Signaturen*** (Operationen)
    - zum Erzeugen von Objekten, zum Verändern etc.
  - ***Axiome***
    - formulieren den semantischen Zusammenhang der Operationen.
  - ***Vorbedingungen***
    - geben an, in welchem Zustand welche Operationen gültig sind.

Abstrakte Datentypen

Beispiel 1: ADT Bool

```
TYPE BOOL

FUNCTIONS

  true:  -> BOOL
  false: -> BOOL
  not:   BOOL -> BOOL
  and:   BOOL x BOOL -> BOOL
  or:    BOOL x BOOL -> BOOL

AXIOMS

  not(true) = false
  not(false) = true

  For any x: BOOL

    not(not(x)) = x

    or(true, x) = true     and(false, x) = false
    or(x, true) = true     and(x, false) = false
    or(false, x) = x       and(true, x)  = x
    or(x, false) = x       and(x, true)  = x
```


Abstrakte Datentypen

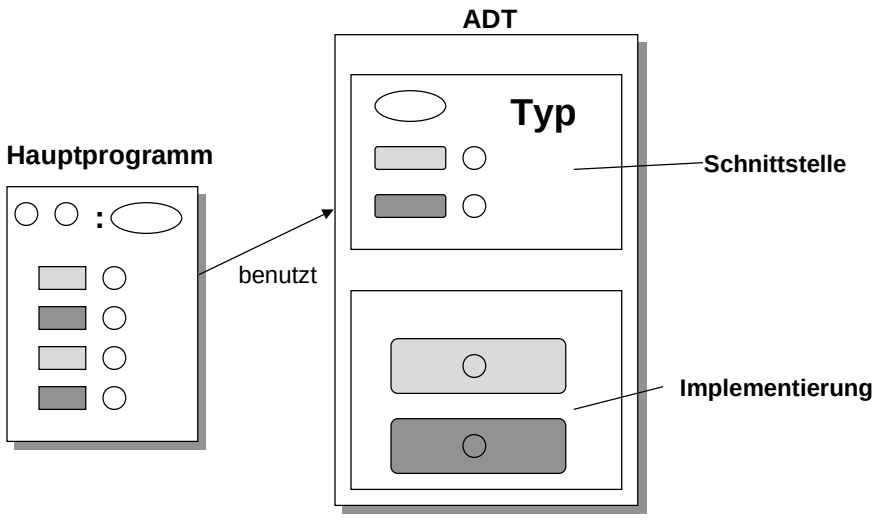
Beispiel 2: ADT NAT

```
TYPE   NAT

FUNCTIONS

  zero: -> NAT
  succ: NAT -> NAT
  iszero: NAT -> BOOL
  pred: NAT -> NAT

  add, mult, sub : NAT x NAT -> NAT

AXIOMS                                  PRECONDITIONS

  For any x, y: NAT                     pred(x: NAT)
                                        requires iszero(x) = false

    pred(succ(x))    = x
    iszero(zero)     = true
    iszero(succ(x))  = false

    add(zero, x)     = x
    add(succ(x), y)  = succ(add(x, y))
    sub(x, zero)     = x
    sub(x, succ(y))  = pred(sub(x, y))
    mult(x, zero)    = zero
    mult(x, succ(y)) = add(mult(x, y), x)
```

Realisierung von ADTs durch Module

- **Die zentrale Idee:**
  - ***Trenne*** die konkrete Realisierung (i.e. Implementation) einer Menge gleichartiger Datenstrukturen von ihren allgemeinen Eigenschaften.
- **Merkmale:**
  - Die Beschreibung der gleichartigen Exemplare einer Datenstruktur wird in einem sog. ADT-Modul ***eingekapselt***.
  - An der Schnittstelle des Moduls sind nur ***Operationen*** sichtbar, die den allgemeinen Umgang mit ***jedem Exemplar*** der Datenstruktur beschreiben.
  - Ein ***Bezeichner für den Typ*** der Datenstruktur wird exportiert.
  - Die Implementierung der Datenstruktur selbst ist ***verborgen***.
- **Jedes ADT-Modul beschreibt und realisiert eine Menge von Exemplaren der abstrakten Datenstruktur.**
  - Die Exemplare werden von ***ihren Kunden*** verwaltet. Ihre Bearbeitung geschieht nur mit Hilfe der exportierten Operationen des ADT-Moduls.

Realisierungsschema ADT



Abstrakte Datentypen

Realisierung eines ADT in Modula-3

- **Forderung:**
  - In der Schnittstelle darf die Typstruktur eines ADT ***nicht sichtbar*** sein.
  - Lediglich der ***Name*** soll bekannt gemacht werden.
- **Realisierung im Modula-3**
  - In der Schnittstelle wird ein ***opakter Typ*** (verdeckter Typ) deklariert
  - Dies geschieht, indem der Typ als ***Untertyp*** zum vordefinierten Typ REFANY deklariert wird.
  - ***Obertyp*** eines opakten Typs muß ein ***Referenztyp*** sein.
  - In der Implementierung des ADTs wird der opake Typ ***offengelegt*** ("enthüllt")
  - Bsp.:

```
TYPE Ordner <: REFANY;
```

Mithilfe des Subtyp-Konzepts und mit opaken Typen können die Forderungen umgesetzt werden

Abstrakte Datentypen

Schnittstelle des ADT-Moduls Ordner

Nur der Name des ADT ist sichtbar!

Alle Operationen erhalten als ersten Parameter das jeweilige Ordnerobjekt

```
INTERFACE OrdnerADT;

TYPE Ordner <: REFANY;

PROCEDURE LegeTextAb (VAR o: Ordner; t : TEXT);
PROCEDURE EntnehmeText (VAR o: Ordner; ) : TEXT ;
PROCEDURE IstVoll ( o: Ordner; ) : BOOLEAN ;
PROCEDURE IstLeer ( o: Ordner; ) : BOOLEAN ;
PROCEDURE Beschrifte (VAR o: Ordner; t : TEXT);
PROCEDURE GibBeschriftung ( o: Ordner; ) : TEXT;
PROCEDURE Anlegen ( ) : Ordner;

END OrdnerADT.
```

Erzeugt ein neues Ordnerobjekt und gibt es zurück

Abstrakte Datentypen

Beispiel: ADT Ordner

Angabe des Typnamens

Angabe der Operationen

```
INTERFACE OrdnerADT;

TYPE Ordner <: REFANY;

PROCEDURE LegeTextAb (VAR o: Ordner; t : TEXT);
PROCEDURE EntnehmeText (VAR o: Ordner; ) : TEXT ;
PROCEDURE IstVoll ( o: Ordner; ) : BOOLEAN ;
PROCEDURE IstLeer ( o: Ordner; ) : BOOLEAN ;
PROCEDURE Beschrifte (VAR o: Ordner; t : TEXT);
PROCEDURE GibBeschriftung ( o: Ordner; ) : TEXT;
PROCEDURE Anlegen ( ) : Ordner;

END OrdnerADT.
```

```
IMPORT OrdnerADT; (* Client *)
VAR
  ord1, ord2 : OrdnerADT.Ordner

BEGIN
  ord1 := OrdnerADT.Anlegen();
  OrdnerADT.Beschrifte (ord1, "Kleine Gedichte");
  OrdnerADT.LegeTextAb (ord1, "Nicht immer sind bequeme ...");

  ord2 := OrdnerADT.Anlegen();
  OrdnerADT.Beschrifte (ord2, "Musikstuecke");
  OrdnerADT.LegeTextAb (ord2, "Tief im Westen ...");
```


Abstrakte Datentypen

Implementierung des ADT Ordner

```
MODULE OrdnerADT EXPORTS OrdnerADT;

CONST MaxTexte = 20;
TYPE Fassungsvermoegen = [1 .. MaxTexte];
     Texte = ARRAY Fassungsvermoegen OF TEXT;

REVEAL Ordner = BRANDED REF RECORD
                  ordnerInhalt : Texte;
                  anzahlTexte  : Fassungsvermoegen;
                  beschriftung : TEXT := "";
                END;
```

- **REVEAL-Deklaration**
  - damit wird die ***interne Struktur*** des opakenTyps bekannt gegeben
  - Steht immer in der ***Implementierung*** eines ADTs (information hiding).
  - Der äußere Typ-Konstruktor ***muß*** ein mit einem ***Brandzeichen*** versehener Referenztyp sein
  - Dieses unterscheidet den Typ von anderen ***strukturell gleichen*** Typen.

Abstrakte Datentypen

Ein Blick ins Innere des ADTs

```
PROCEDURE LegeTextAb (VAR o: Ordner; t : TEXT)=
BEGIN
  o^.anzahlTexte := o^.anzahlTexte + 1;
  o^.ordnerInhalt[o^.anzahlTexte] := t;
END LegeTextAb;

PROCEDURE EntnehmeText (VAR o: Ordner) : TEXT =
VAR t : TEXT;
BEGIN
  t := o^.ordnerInhalt[o^.anzahlTexte];
  o^.ordnerInhalt[o^.anzahlTexte] := "";
  o^.anzahlTexte := o^.anzahlTexte - 1;
  RETURN t;
END EntnehmeText;


PROCEDURE Anlegen () : Ordner =
VAR o : Ordner;
BEGIN
  o := NEW(Ordner);
  FOR i := FIRST(Fassungsvermoegen) TO LAST(Fassungsvermoegen) DO
    o^.ordnerInhalt[i] := "";
  END;
  o^.anzahlTexte := 0;
  RETURN o;
END Anlegen;
```


Abstrakte Datentypen

Diskussion : ADT-Realisierung in Modula-3

- **Wir können feststellen**
  - Als Typ für einen ADT müssen wir einen opaken Referenztyp wählen.
  - Grund: So kann der Übersetzer für Objekte eines ADTs ausreichend Speicherplatz reservieren, ohne den inneren Aufbau des Typs zu kennen.
- **Konsequenz:**
  - Es können neben den Operationen der Schnittstelle des ADTs auch die Operationen
    - Zuweisung und
    - Vergleich auf Objekten des ADTs durchgeführt werden (Pointer-Zuweisung und Pointer-Vergleich).
  - Diese sollten jedoch nicht genutzt werden!
  - Das Brandzeichen verhindert, daß einem Objekt eines ADT zufälligerweise ein Wert eines strukturell gleichen Typs zugewiesen werden kann.

Abstrakte Datentypen

Zuweisung und Vergleich von ADT-Objekten

```
ordner1 := OrdnerADT.Anlegen();
OrdnerADT.Beschrifte (ordner1, "Kleine Gedichte");
OrdnerADT.LegeTextAb (ordner1, "Nicht immer sind bequeme Stuehle ...");

ordner2 := OrdnerADT.Anlegen();
OrdnerADT.Beschrifte (ordner2, "Kleine Gedichte");
OrdnerADT.LegeTextAb (ordner2, "Nicht immer sind bequeme Stuehle ...");

IF ordner1 = ordner2 THEN
  SIO.PutLine ("1 ordner sind gleich");
ELSE
  ordner2 := ordner1;
  OrdnerADT.Beschrifte (ordner1, "Romane");
END;
IF ordner1 = ordner2 THEN
  SIO.PutLine ("nach Zuweisung sind die Ordner gleich");
END;

SIO.PutLine (OrdnerADT.GibBeschriftung(ordner1));
SIO.PutLine ("--------------------------");
SIO.PutLine (OrdnerADT.GibBeschriftung(ordner2));
```

Vergleich der Referenzen (Adressen

ordner2 zeigt auf die selbe Adresse wie ordner1

Ändert ordner1 und ordner2

Abstrakte Datentypen

Modul als Entwicklungseinheit

- **Ein Modul ist charakterisiert durch eine *Entwurfs-entscheidung*.**
  - (z.B. wir wollen Ordner verarbeiten können)
- **Jedes Modul basiert auf genau einer Entwurfsentscheidung.**
- **Module *verbergen* Implementierungsentscheidungen.**
  - Jedes Modul hat sein Geheimnis.
- **Es werden immer die Entscheidungen**
  - in einem Modul gekapselt, die vielleicht ***revidiert*** werden müssen als andere.
  - z.B. Datenaufbau
- **Der Änderungsaufwand muß möglichst immer auf ein Modul begrenzt werden.**

Abstrakte Datentypen

Zusammenfassung ADT

- **In imperativen Sprachen**
  - mit einem ***Modulkonzept***
  - realisieren wir abstrakte Datentypen mit einem ADT-Modul.
- **Ein ADT-Modul zeigt folgende Eigenschaften:**
  - Das Modul liefert ***Exemplare einer Datenstruktur***, die beim Kunden aufbewahrt werden.
  - Dadurch können ***mehrere Exemplare*** eines ADT-Moduls bearbeitet werden.
  - Die Datenstruktur ist nur als ***Verweis*** auf die interne Repräsentation bekannt.
  - Die Repräsentation selbst ist ***verborgen*** und ***austauschbar***.
  - Die Datenstrukturen des ADT-Moduls können (fast) nur über die ***exportierten Operationen*** des Moduls bearbeitet werden.

Was haben wir gelernt?

- **Modularisierung**
  - Mittel, um ***handhabbare*** Programmeinheiten zu konstruieren
  - Module bestehen aus ***Schnittstelle*** und ***Implementierung***
- **Information Hiding**
  - Ziel:
    - Implementierung wesentlicher Details ist ***nicht*** nach ***außen sichtbar,*** insbesondere anwendbar auf Datenstrukturierung
  - Objektmodul:
    - Es wird in einem Modul ***ein Objekt*** gemäß Information Hiding realisiert, Datenabstraktion für ein Objekt
  - Abstrakter Datentyp:
    - Es wird ein ***geschützter Typ*** realisiert. Objekte des ADTs können nur mit den Operationen des ADTs manipuliert werden, Datenabstraktion für eine „Klasse“ von Objekten

Glossar

- **Information Hiding: Prozeß- oder funktionale Abstraktion, Datenabstraktion, funktionaler Module, Datenobjektmodul (Kapsel), abstrakter Datentyp**
- **opake Datenstruktur, opake (geschützte, abstrakte, geheime) Datentypen**
- **Schnittstellenoperationen eines Datenabstraktionsbausteins: Initialisierung (Löschung), Lesen, Verändern, Sicherheitsoperationen**
- **Realisierung opaker Datentypen durch Verweistypen**
- **Typname, Signatur, Axiome, Vorbedingungen eines ADT**
- **Subtypen in Modula-3 haben Referenzsemantik (keine Zuweisung und keine Gleichheitsabfrage nutzen!)**