# Tag

## GPS-vorstellen

## Einführung

* Sinnhaftigkeit Pakete => **W i e d e r v e r w e n d b a r k e i t** und Datenkapselung

## Pakete (Tafel)

* **Benutzerseite** und **Paketschreiber** darstellen (siehe Aufzeichnung)
* Pakete **ohne** Rumpf
* Pakete mit Rumpf (Deklarationsteil)
* Sichtbarkeiten von Paketen: Spezifikation/Body
* Übersetzen der einzelnen Einheiten zu einem ausführbaren Programm

### SWPÄ – nur Paket-Spezifikation

* Paket: Ada.Calendar vorstellen ( im doc-Ordner PDF erstellt)
* Rechnung-Programm: Kapitel 3.7 Programmteile wiederverwenden mit Paketen
  + Übung mit nur Spezifikation
  + Konstanten auslagern in Paketspezifikation
  + Datumsfunktionen hinzugefügt Ada.Calendar
* Lösung: im Ordner SWPAE-> main\_rechnung\_7.adb und im Ordner Globale\_Daten\Glob\_Rechnung-> liegen die eingebundenen Pakete.

### Ergänzung

* Folien: bis Seite 36
* Paket-Rumpf: **begin** (optional)
  + zur Variablen Initialisierung wird nur einmalig ausgeführt! (Anweisungsteil)
* Sichtbarkeiten von Paketen mit eingebetteten Paketen (nicht auf **private** eingehen)
  + Template: ***template\_sichtbarkeiten.adb*** mit Aussen\_Pkg und integriertem Innen\_Pkg Paket.
* Sichtbarkeiten von Paketen in Paket-Hierarchien (nicht auf **private** eingehen)
  + Template: ***template\_sichtbarkeiten.adb*** mit Aussen\_Pkg und Kind-Paket: Aussen\_Pkg.Child.
* Ausnahmebehandlung
  + Definition von Ausnahmen
  + Bearbeiten von Ausnahmen in den Unterprogrammen (wie bereits bekannt)
  + Bearbeiten von Ausnahmen im Anweisungsteil des Paket-Rumpfes
* Übungen: Aufgaben 1 und 2

### SWPÄ – mit Paket-Body

* Rechnung-Programm: Kapitel 3.8 Auslagerung von Vereinbarungen in Pakete mit Pakethierarchie

# Tag

## Wiederholung

* Aufgaben / SWPÄ nachbesprechen

## Fortsetzung

* Sinnhaftigkeit Pakete => Wiederverwendbarkeit und **D a t e n k a p s e l u n g**
* Sichtbarkeiten von Paketen: Spezifikation/Spezifikation mit **private**/Body
  + Template: ***template\_sichtbarkeiten.adb*** mit aussen\_pkg und Kind-Paket: Aussen\_Pkg.Child. Zugriff auf privaten Bereich (Spezifikation) des Vater-Pakets eingehen.
  + Skript: siehe Kapitel 3.1 Pakethierarchien

## Anwenden als ADO und ADT

Basis: **main\_keller** (im Vorgaben-Ordner)

### Schritt 1: Keller auslagern in Paket

* Wiederverwendbarkeit erfüllt, PROBLEMATIK: Manipulation
* ***main\_stack\_paket*** mit ***stack\_paket*** (Glob\_Daten\Glob\_Pakete)

### Schritt 2.1: mehrere Datenobjekte werden benötigt (ADT)

* Paket stellt Datentyp für Nutzer zur Verfügung (ADT)
* Datentyp soll **n i c h t** manipulierbar sein
  + Sichtbarkeit nach außen einschränken mit **private /** limited private
* Umbau ***Stack\_Paket*** nach ADT:
  + Neue Syntax erläutern für die Datenkapselung und von LT umsetzen lassen
  + ***main\_stack\_paket\_adt*** mit ***stack\_paket\_adt*** (Glob\_Daten\Glob\_Pakete)

### Schritt 2.2: ein Datenobjekt wird benötigt (ADO) siehe Aufgabe 3

* Paket verbirgt den Datentyp vor dem Nutzer, da dieser nicht bekannt sein muss.
* Paket legt ein Datenobjekt/-variable an (ADO).
* Sichtbarkeit von Datentyp und Datenobjekt ist nach außen nicht gegeben.
* Umbau nach ADO:
  + ***main\_stack\_paket\_ado*** mit ***stack\_paket\_ado*** (Glob\_Daten\Glob\_Pakete)

## Wiederholung: Sichtbarkeiten und Nutzung von Daten in Paketen

* Foliensatz: bis Seite 55
  + private und **limited private**
    - Vorgabe: Beispiel **main\_limited\_private.adb** (siehe Skript) mit Paket\_1 und Paket\_2.
  + **offene Konstantendeklaration** bis Seite 59, Skript Seite 30

## Übungen

* Aufgabe 4 und 5.1 und 5.2
* Problematik bei ADT-Bus: Initialisierung der Sitzplätze? Kein Datenobjekt im Paket vorhanden!
  + Programmeinschränkung: Jeden Bus nacheinander buchen nur erlauben.
  + Zu Beginn des Buchungsvorgang, den Bus initialisieren (findet im Paket statt)
  + Öffentliche Schnittstelle zum Initialisieren des Busses aller Tage anbieten (Spezifikation-anpassen)
  + Offene Konstanten-Deklaration hinzufügen bei Implementierung als internes Array.
  + Array in Verbundstyp kapseln und die interne Komponente des Verbundes vorinitialisieren (others => Plaetze).

# Tag

## Wiederholung

* Pakete, Sichtbarkeiten, Ado, Adt bis 9:00 Uhr
  + Put in ADT/ADO-Paket implementieren sowie in die beiden HP

### SWPÄ – ADO und ADT

* Rechnung-Programm: Kapitel 3.9 Datenkapselung in Pakete

## Generische Programmeinheiten

### generische Unterprogramme

* Basis: ***template\_gen\_procedure\_vorlage.adb***
  + Beispiel: Überladung umwandeln in generische Prozedur (generischen Typ)
* Basis: ***template\_gen\_reihe***
  + Beispiel: Min-Ermittlung von Werten (Generischer: Element, Index, Array)
  + Basis: Übung von LT: mit konkreten Datentypen machen lassen und dann Umbauen an Tafel in generische Funktion
    - Reihung über TAGE\_T und TEMPERATUREN\_T verwalten und TAGE\_T vom geringsten Wert zurückgeben. function Min (Reihe : in TEMPERATUR\_TA) return TAGE\_T;
    - Reihung über MONATE\_T und STUNDEN\_T verwalten und MONATE\_T zurückgeben. Function Min (Reihe : in MONATSSTUNDEN\_TA) return MONATE\_T;
    - Erweiterung: in Reihung Person mit Gehalt in Personen verwalten! (Komponente => is private)

bis Mittag

### Übungen

* Aufgabe: 7, 8, 9, 10

# Tag

## Wiederholung

* generische Paketspezifikation
  + Sichtbarkeiten nochmals aufgreifen (generische Parameter siehe Seite 51 Skript)

## Generische Pakete

Basis: Keller als ADO main\_Stack\_Paket\_ado und Stack\_Paket\_Ado

Umbau zu generisches Paket (Schritte erklären)

1. Schritt: Paket generisch machen: Vorteil: mehrere Pakete (Keller) sind auszuprägen! ***stack\_paket\_ado\_gen***

2. Schritt: Hauptprogramm anpassen: ***main\_stack\_paket\_ado\_gen***

2.1 generisches Paket einbinden

2.2 generisches Paket ausprägen

2.3 Variablen und Aufrufe anpassen

2.4 Prozedur : Ausgabe\_Komponente generisch machen

2.5 2. Keller ausprägen

2.6 Ausgabe\_Komponente für 2. Keller ausprägen und aufrufen

3. Schritt: aufzunehmende **Komponente** in den Keller generisch machen   
(anpassen: ***stack\_paket\_ado\_gen\_1***)

3.1 Zuerst mit ganzzahligen Datentypen, danach  
 3.2 soll auch ein Verbund für die Aufnahme möglich sein (**private**)

4. Schritt: Hauptprogramm nochmals anpassen ***main\_stack\_paket\_ado\_gen\_1***

4.1 Ausgabe\_Keller\_1 generisch machen (Komponente, Ist\_Leer, Pop und Ausgabe\_Komponente übergeben) **oder** ein generisches Paket übergeben   
with package Keller is new Stack\_Ado\_Gen\_1(ELEMENT\_T)

Problematik: Initialisierung des Kellerobjekts, mit der 0 ist **n i c h t** mehr möglich!

Abhilfe-Möglichkeiten: generische Konstantendeklaration, für den Fall, dass der Komponenten\_T –Typ **private** ist oder bei skalaren/diskreten Datentypen mit KOMPONENTEN\_T'First.

## Zusammenfassung

* Foliensatz bis Seite 111
* Übungen: Aufgabe 5.3, 5.4, 11, 12

# Tag

## Wiederholung

* Übungen nachbesprechen und weiter Übungen machen bis 9:00 Uhr

## Generische Schleife

Überleitung geschieht durch Erkennen, dass die Prozedur: Ausgabe\_Keller nicht zum HP sondern zum Paket gehört. Ruft ausschließlich Paket-Unterprogramme auf. Überführung in Paket. **Keine** abgewandelte Implementierung.

### Einführung

* Tafel: ***Stack\_paket\_ado\_gen*** an Tafel skizzieren, für Nutzer und Paketschreiber
* Aufgabe: Alle Elemente des Kellers ausgeben, ohne den Zustand des Kellers zu verändern?
  + Problematik erarbeiten!
    - Schleife: **Wo,** im Paket oder beim Nutzer implementieren?
      * Kennt seine interne Struktur zum Verwalten der Daten. Muss deswegen nicht den Keller leeren um zum Ziel zu gelangen.
    - Aktion: **Wo**, im Paket oder beim Nutzer definieren?
      * Kennt den Datentyp und kann die Ausgabe nach seinen Anforderungen gestalten.
    - Signatur der Aktion?
  + Lösung an der Tafel erarbeiten

#### Komponent\_T für ganzzahlige Datentypen definieren

Ausgabe aller Inhalte im Paket lösen, da im HP dies nur mit dem Zerstören der Inhalte des Kellers einhergeht.

Nachteil: Testausgaben (Konstanten) müssen vom HP so akzeptiert werden!

#### Komponent\_T für private Datentypen definieren

Paket weiß von dem Komponenten-Typ nichts, kann keine Ausgabe generieren.

Fakten:

Paket kennt die Verwaltungsstruktur der Daten und Nutzer kennt die Daten.

Nutzer schreibt eine Ausgabe-Prozedur der Daten (PERSON\_TR) und im Paket werden die Daten bereitgestellt.

1. Plan: generische Prozedur dem Paket übergeben. Nachteil : Immer die gleiche Ausgabe der Daten möglich.
2. Plan: mehr Flexibilität, wenn die Prozedur (Fuer\_alle\_Elemente), die generische Prozedur (Aktion) erhält.

### Umsetzung

5. Schritt: Generische Schleife, Aktion über alle Elemente durchführen

generische Schleife implementieren: ***stack\_paket\_ado\_gen\_schleife***

5.1 Spezifikation  
5.2 Body

6. Schritt: Hauptprogramm entsprechend anpassen ***main\_stack\_paket\_ado\_gen\_schleife***

6.1 generische Schleife **ausprägen,** mit Hilfenahme der generischen Prozedur: Ausgabe\_Komponente.

Vorteil: Elemente werden **n i c h t** gelöscht, sondern sind weiterhin im Keller vorhanden, da das Unterprogramm **Push** nicht aufgerufen werden muss.6.2 **Aufrufen** der generischen Schleife

6.3 Prozedur: **Anzahl\_Elemente\_Keller\_1** schreiben (Spezifikation, Vorgabe) und Body. Es ist eine globale Variablen (HP) erforderlich! (Soll von LT erkannt werden!)

6.4 **Ausprägen** der generischen Schleife

6.5 **Aufrufen** der generischen Schleife

Problematik: Wenn die generische Schleife öfters aufgerufen wird, dann ist ein zurücksetzen der globalen Variablen (HP) erforderlich!

6.6 Die Summe über alle Werte bilden (optional), als weitere Aktion vom HP.

### Zusammenfassung

* Foliensatz bis 114 (Ende)
* Übung: Aufgabe 13

## Synonymvereinbarung (Renaming)

* Foliensatz: Seite 60 bis Seite 70
* template: ***Umbenennung*** erklären! Umbenennung von Objekten
  + Reihungen und Verbunde
  + Exception
  + Prozeduren und Funktionen
* Übung: Aufgabe 6

# Ergänzung (NiceToKnow)

ab Ada 2005 (Ergänzung, nicht so wichtig)

* Kontext-Klausel: **with**-Klausel
  + **private** with [Paket]
    - für die Nutzung nur im privaten Bereich des eingebundenen Pakets.

Anmerkung: Paket als private deklarieren (siehe Sichtbarkeiten),  
erlaubt den Zugriff auf private Vaterklassen-Daten und Nutzung  
nur in der eigenen Paket-Hierarchie.

* + **limited with** [Paket] unvollständige Paketeinbindung
    - für Nutzung bei Ringschluss, bei gegenseitigem Einbinden