

ABAP für Fortgeschrittene

**Teil 2: Spezielle Werkzeuge und Konzepte für die
objektorientierte Programmierung**

Copyright

- *Das vorliegende Skriptum baut zu großen Teilen auf den Publikationen zum TAW11- und TAW12-Kurs – das Copyright dieser Teile liegt bei der SAP AG.*
- *Die in diesem Kurs verwendeten Abbildungen wurden – falls nicht anders gekennzeichnet – in Anlehnung zum TAW11- und TAW12-Kurs erstellt. Das Copyright dieser Teile liegt bei der SAP AG.*
- *Für alle Screenshots im Skriptum, auch wenn diese nur verkürzt oder auszugsweise gezeigt werden, gilt der Hinweis: Copyright SAP AG*
- *Die Weitergabe und Vervielfältigung dieser Publikation oder von Teilen daraus sind, zu welchem Zweck und in welcher Form auch immer, ohne die schriftliche Genehmigung von Prof. Dr. Heimo H. Adelsberger, Dipl.-Wirt.-Inf. Pouyan Khatami und Dipl.-Wirt.-Inf. Taymaz Khatami nicht gestattet..*

Inhaltsverzeichnis

| | |
|---|----------|
| COPYRIGHT | 2 |
| INHALTSVERZEICHNIS | 3 |
| ABBILDUNGSVERZEICHNIS | 4 |
| 5 SPEZIELLE WERKZEUGE UND KONZEPTE FÜR DIE OBJEKTORIENTIERTE PROGRAMMIERUNG | 6 |
| 5.1 DER CLASS BUILDER | 6 |
| 5.1.1 Praxis: Erstellen einer globalen Mitarbeiterklasse | 6 |
| 5.1.2 Praxis: Testen mit dem Class Builder | 9 |
| 5.1.3 Praxis: Erstellen einer abgeleiteten globalen Klasse | 12 |
| 5.1.4 Praxis: Import lokaler Klassen | 14 |
| 5.1.5 Praxis: Globale Klassen im Object Navigator | 17 |
| 5.1.6 Praxis: Der Refactoring-Assistent | 19 |
| 5.1.7 Praxis: Erstellen eines globalen Interfaces | 22 |
| 5.1.8 Praxis: Weitere hilfreiche Funktionen | 23 |
| 5.1.9 Kontrollfragen | 24 |
| 5.1.10 Antworten | 25 |
| 5.2 SPEZIELLE OBJEKTORIENTIERTE PROGRAMMIERKONZEPTE | 25 |
| 5.2.1 Das Singleton-Pattern | 25 |
| 5.2.2 Praxis: Anwendung des Singleton-Patterns | 25 |
| 5.2.3 Freundschaft | 29 |
| 5.2.4 Praxis: Anwendung von Freundschaft | 30 |
| 5.2.5 Kontrollfragen | 31 |
| 5.2.6 Antworten | 33 |
| 5.3 SHARED OBJECTS | 33 |
| 5.3.1 Praxis: Anlegen eines Gebietes | 35 |
| 5.3.2 Praxis: Aufbau einer Gebietsinstanz | 37 |
| 5.3.3 Praxis: Verwenden einer Gebietsinstanz | 42 |
| 5.3.4 Versionen von Gebietsinstanzen | 44 |
| 5.3.5 Kontrollfragen | 48 |
| 5.3.6 Antworten | 49 |
| 5.4 KLASSENBASIERTE AUSNAHMEN | 49 |
| 5.4.1 Praxis: Behandlung vordefinierter Ausnahmen | 51 |
| 5.4.2 Praxis: Definition und Verwendung eigener globaler Ausnahmeklassen | 53 |
| 5.4.3 Das RETRY-Statement | 57 |
| 5.4.4 Das RESUME-Statement | 57 |
| 5.4.5 Propagierung und Hierarchie von Ausnahmen | 58 |
| 5.4.6 Kontrollfragen | 63 |
| 5.4.7 Antworten | 64 |
| 5.5 KAPITELABSCHLUSS | 65 |

Abbildungsverzeichnis

| | |
|---|----|
| Abbildung 1: Anlegen einer globalen Klasse: SAP-System-Screenshot..... | 6 |
| Abbildung 2: Einstellungen beim Anlegen einer Klasse: SAP-System-Screenshot..... | 7 |
| Abbildung 3: Methodenansicht im Class Builder: SAP-System-Screenshot | 7 |
| Abbildung 4: Methodendefinition im Class Builder: SAP-System-Screenshot | 8 |
| Abbildung 5: Parameterliste zu einer Methode: SAP-System-Screenshot..... | 8 |
| Abbildung 6: Fertiger Returning-Parameter: SAP-System-Screenshot | 8 |
| Abbildung 7: Attributliste im Class Builder: SAP-System-Screenshot | 8 |
| Abbildung 8: Attribut in der Attributliste: SAP-System-Screenshot | 9 |
| Abbildung 9: Signaturanzeige: SAP-System-Screenshot..... | 9 |
| Abbildung 10: Implementierung einer Methode: SAP-System-Screenshot | 9 |
| Abbildung 11: Konstruktor in der Methodenliste: SAP-System-Screenshot | 9 |
| Abbildung 12: Aufruf des Konstruktors beim Testen einer Klasse: SAP-System-Screenshot | 10 |
| Abbildung 13: Testinstanz der Klasse: SAP-System-Screenshot..... | 10 |
| Abbildung 14: Testen einer Methode: SAP-System-Screenshot..... | 11 |
| Abbildung 15: Rückgabewert einer Methode: SAP-System-Screenshot | 11 |
| Abbildung 16: Ausgabe der PRINT-Methode: SAP-System-Screenshot | 11 |
| Abbildung 17: Anlegen einer abgeleiteten Klasse: SAP-System-Screenshot | 12 |
| Abbildung 18: Geerbte Methoden: SAP-System-Screenshot..... | 13 |
| Abbildung 19: Vorgegebenes Aufrufgerüst für den Superkonstruktor: SAP-System-Screenshot | 13 |
| Abbildung 20: Einstiegsbild des Class Builders: SAP-System-Screenshot | 14 |
| Abbildung 21: Importmaske für Programmlokale Klassen: SAP-System-Screenshot..... | 15 |
| Abbildung 22: Ausgewählte Klasse: SAP-System-Screenshot | 15 |
| Abbildung 23: Import-Bestätigung: SAP-System-Screenshot | 16 |
| Abbildung 24: Fehlermeldung beim prüfen der importierten Klasse: SAP-System-Screenshot..... | 16 |
| Abbildung 25: Das Attribut it_employees: SAP-System-Screenshot | 16 |
| Abbildung 26: Vererbung im Class Builder: SAP-System-Screenshot..... | 17 |
| Abbildung 27: Fehlermeldung: SAP-System-Screenshot | 17 |
| Abbildung 28: Klassen im Navigationsbaum: SAP-System-Screenshot..... | 18 |
| Abbildung 29: Codegerüst für den Konstruktoraufruf: SAP-System-Screenshot | 18 |
| Abbildung 30: Methoden im Navigationsbaum: SAP-System-Screenshot | 18 |
| Abbildung 31: Methoden-Aufrufgerüst: SAP-System-Screenshot | 19 |
| Abbildung 32: Ausgabe des Programms: SAP-System-Screenshot..... | 19 |
| Abbildung 33: Anlegen einer Klasse: SAP-System-Screenshot | 19 |
| Abbildung 34: Fehler durch die Vererbungsbeziehung: SAP-System-Screenshot | 20 |
| Abbildung 35: Der Refactoring-Assistent: SAP-System-Screenshot..... | 21 |
| Abbildung 36: Verschobene Komponenten: SAP-System-Screenshot | 21 |
| Abbildung 37: Inhalt der internen Tabelle im Debugger: SAP-System-Screenshot | 22 |
| Abbildung 38: Interface im Testwerkzeug: SAP-System-Screenshot..... | 23 |
| Abbildung 39: Gegliederte Anzeige von Methoden: SAP-System-Screenshot..... | 24 |
| Abbildung 40: Anlegen der Klasse: SAP-System-Screenshot | 26 |
| Abbildung 41: Konstruktorsichtbarkeit bei lokalen Klassen: SAP-System-Screenshot..... | 27 |
| Abbildung 42: Finale Klassen und Methoden | 27 |
| Abbildung 43: Statischer Konstruktor in der Methodenliste: SAP-System-Screenshot..... | 28 |
| Abbildung 44: Freundschaftsbeziehung zwischen zwei Klassen | 29 |
| Abbildung 45: Herstellen einer Freundschaftsbeziehung: SAP-System-Screenshot | 30 |
| Abbildung 46: Ausführen der Methode: SAP-System-Screenshot | 31 |
| Abbildung 47: Ergebnis des Methodenaufrufs: SAP-System-Screenshot..... | 31 |
| Abbildung 48: Einordnung des Shared Objects Memory..... | 33 |
| Abbildung 49: Zugriff auf Shared Objects | 34 |
| Abbildung 50: Gebiete und Gebietsinstanzen | 34 |
| Abbildung 51: Gebiet, Gebietsklasse und Gebietswurzelklasse | 35 |

| | |
|--|----|
| Abbildung 52: Eigenschaften der Wurzelklasse: SAP-System-Screenshot | 36 |
| Abbildung 53: Anlegen des Gebiets: SAP-System-Screenshot | 37 |
| Abbildung 54: Erzeugen der Gebietsinstanz | 38 |
| Abbildung 55: Anlegen von Objekten im Shared Memory | 39 |
| Abbildung 56: Setzen der Referenzen | 40 |
| Abbildung 57: Lösen der Schreibsperre | 41 |
| Abbildung 58: Gebiet in der Transaktion SHMM: SAP-System-Screenshot | 41 |
| Abbildung 59: Gebietsinstanz: SAP-System-Screenshot | 42 |
| Abbildung 60: Schreibsperre: SAP-System-Screenshot | 42 |
| Abbildung 61: Lesender Zugriff auf die Gebietsinstanz | 43 |
| Abbildung 62: Ausgabe des Programms: SAP-System-Screenshot | 44 |
| Abbildung 63: Gewöhnlicher Zustand mit Lesesperre auf aktiver Version | 45 |
| Abbildung 64: Koexistenz von aktiver Version und Version im Aufbau | 45 |
| Abbildung 65: Vorhandene Lesesperre auf nun veraltete Version | 46 |
| Abbildung 66: Zeitgleiches Lesen von unterschiedlichen Versionen | 47 |
| Abbildung 67: Verfallene Version | 48 |
| Abbildung 68: Beispiel für eine klassische Ausnahmebehandlung | 49 |
| Abbildung 69: Klassenbasiertes Ausnahmekonzept: Überblick | 50 |
| Abbildung 70: Hierarchie der Ausnahmeklassen | 50 |
| Abbildung 71: Syntax des TRY-Blocks | 51 |
| Abbildung 72: Laufzeitfehler bei Nulldivision: SAP-System-Screenshot | 52 |
| Abbildung 73: Selbstdefinierte Fehlermeldung: SAP-System-Screenshot | 52 |
| Abbildung 74: Fehlermeldung mit Text aus einem Ausnahmeobjekt: SAP-System-Screenshot | 53 |
| Abbildung 75: Aufruf der Methode: SAP-System-Screenshot | 53 |
| Abbildung 76: Nachrichtenklasse für die Ausnahmeklasse: SAP-System-Screenshot | 54 |
| Abbildung 77: Auswahl einer Nachricht: SAP-System-Screenshot | 55 |
| Abbildung 78: Aufrufgerüst mit TRY-Block: SAP-System-Screenshot | 56 |
| Abbildung 79: Ausgabe des Programms: SAP-System-Screenshot | 56 |
| Abbildung 80: Das RETRY-Statement | 57 |
| Abbildung 81: Das RESUME-Statement | 58 |
| Abbildung 82: Angabe von Ausnahmen im Class Builder: SAP-System-Screenshot | 58 |
| Abbildung 83: Propagierung einer Ausnahme | 59 |
| Abbildung 84: Verkettung von Ausnahmen | 60 |
| Abbildung 85: Codeausschnitte zu abgebildeten Ausnahmen | 61 |
| Abbildung 86: Standard-Ausnahmen in der Hierarchie der Ausnahmeklassen | 62 |
| Abbildung 87: Warnung bei statischer Prüfung: SAP-System-Screenshot | 62 |

5 Spezielle Werkzeuge und Konzepte für die objektorientierte Programmierung

5.1 Der Class Builder

Im Kurs „Einführung in ABAP“ haben Sie zunächst lokale Typdefinitionen mit dem `TYPES-`Befehl verwendet. Später kamen dann globale Typen im Dictionary hinzu. Einen ähnlichen Dualismus gibt es auch in Bezug auf objektorientierte Typen: Bislang haben Sie nur lokale Klassen definiert, deren Sichtbarkeit sich auf den Kontext ihres Programmes beschränkte. In diesem Abschnitt werden Sie hingegen globale Klassen kennen lernen, die als eigene Repository-Objekte existieren und von beliebigen Programmen verwendet werden können. Sie unterstützen die entsprechenden Konzepte für Repository-Objekte wie Versionierung, Namensräume, Transportwesen usw.

Zur Definition von globalen Klassen wird ein spezielles Werkzeug, der **Class Builder**, verwendet. Dieses Werkzeug ist weitgehend in die ABAP Workbench integriert. Ein direkter Aufruf des Class Builders ist über die Transaktion **SE24** möglich.

5.1.1 Praxis: Erstellen einer globalen Mitarbeiterklasse

Um den Class Builder kennen zu lernen, werden Sie hier direkt eine globale Klasse erstellen. Öffnen Sie dazu Ihr Paket im Object Navigator, und klicken Sie im Navigationsbaum auf der linken Seite mit der rechten Maustaste auf das Paket.

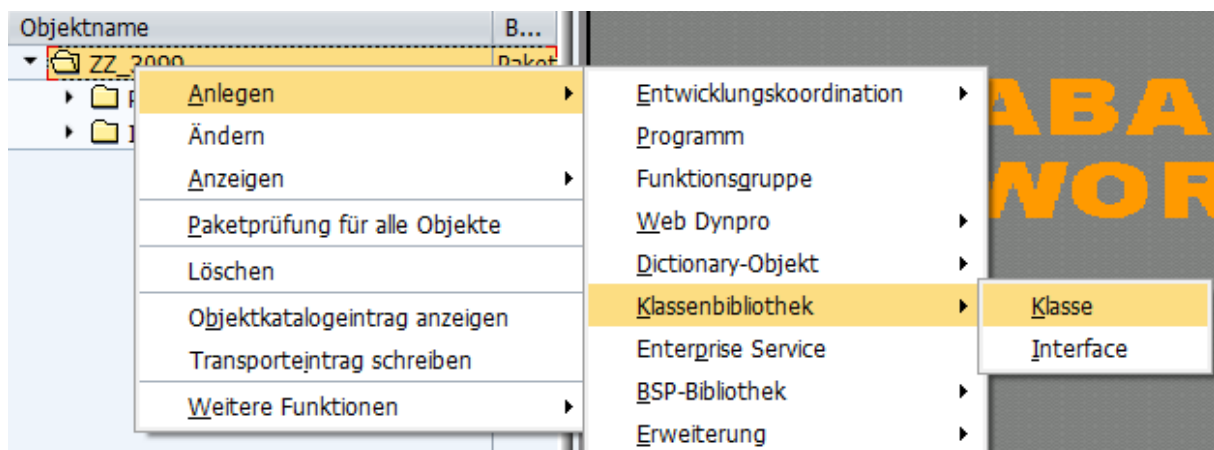


Abbildung 1: Anlegen einer globalen Klasse: SAP-System-Screenshot

Wählen Sie aus dem Kontextmenü **Anlegen->Klassenbibliothek->Klasse** aus. Es erscheint das folgende Fenster:

Abbildung 2: Einstellungen beim Anlegen einer Klasse: SAP-System-Screenshot

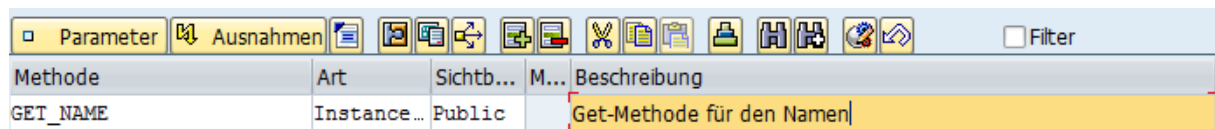
Der Name unterliegt nun den bekannten Namensraumrestriktionen. Benennen Sie Ihre Klasse daher **ZCL_####_EMPLOYEE**. Deselektieren Sie **final** und pflegen Sie eine passende Beschreibung. Bestätigen Sie und geben Sie wie gewohnt Paket und Transportauftrag ein bzw. bestätigen Sie die Nachfragen.

Der Class Builder empfängt Sie mit einer Tab-basierten Oberfläche, auf der der Tab **Methoden** ausgewählt sein sollte:

| Methode | Art | Sic... | M... | Beschreibung |
|---------|-----|--------|------|--------------|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

Abbildung 3: Methodenansicht im Class Builder: SAP-System-Screenshot

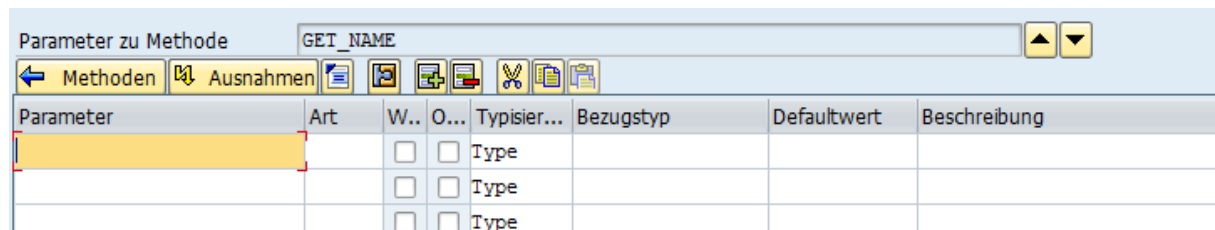
In der hier angezeigten Liste können die Methoden angelegt bzw. bearbeitet werden, über die die Klasse verfügen soll. Legen Sie hier zunächst die Methode **get_name** an, indem Sie den Namen in der ersten Zeile in der Spalte **Methode** angeben. In der Spalte **Art** können Sie wählen, ob es sich um eine statische oder um eine Instanzmethode handelt. Wählen Sie dort entsprechend **Instance**. Die Sichtbarkeit in der nächsten Spalte soll **Public** sein. Beschreiben Sie die Methode auch in der Beschreibungsspalte.



| Methode | Art | Sichtb... | M... | Beschreibung |
|----------|-------------|-----------|------|---------------------------|
| GET_NAME | Instance... | Public | | Get-Methode für den Namen |

Abbildung 4: Methodendefinition im Class Builder: SAP-System-Screenshot

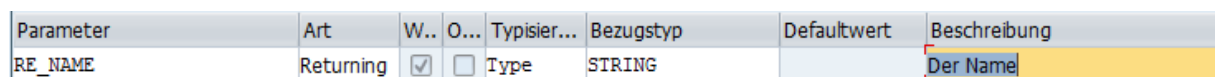
Um die Parameter der Methode zu definieren, wählen Sie die Schaltfläche **Parameter**, die Sie in der obigen Abbildung links oben sehen. Sie führt zu einer Parameterliste:



| Parameter | Art | W... | O... | Typisier... | Bezugstyp | Defaultwert | Beschreibung |
|-----------|-----|--------------------------|--------------------------|-------------|-----------|-------------|--------------|
| | | <input type="checkbox"/> | <input type="checkbox"/> | Type | | | |
| | | <input type="checkbox"/> | <input type="checkbox"/> | Type | | | |
| | | <input type="checkbox"/> | <input type="checkbox"/> | Type | | | |

Abbildung 5: Parameterliste zu einer Methode: SAP-System-Screenshot

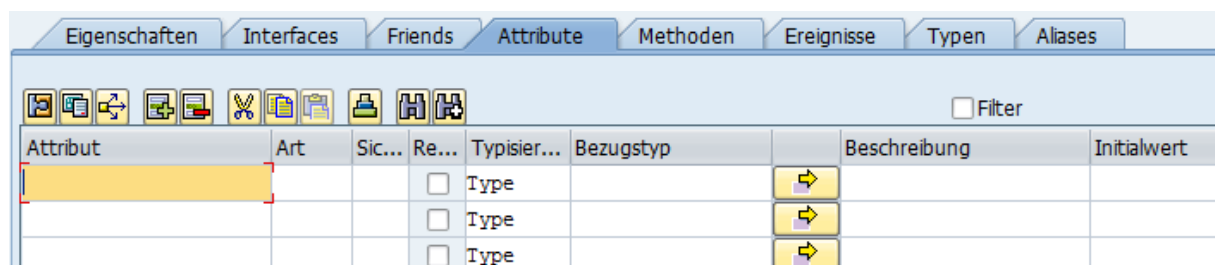
Die Methode soll einen Returning-Parameter `re_name` besitzen. Geben Sie den Namen **re_name** in die erste Spalte ein und wählen Sie in der Spalte **Art** **Returning** aus. Die dritte Spalte gibt an, ob eine Wertübergabe stattfinden soll, die vierte ob es sich um einen optionalen Parameter handelt. Übergehen Sie diese Spalten zunächst. Die folgende Spalte gibt die Art der Typisierung an. Belassen Sie hier die Angabe **Type** und geben Sie den Bezugstyp **string** in die folgende Zelle ein. Geben Sie eine Beschreibung ein und bestätigen Sie mit der Entertaste. Sie sehen, dass sich daraufhin die vorhin übersprungenen Spalten verändert haben: Da ein Returning-Parameter nicht optional sein darf und nur per Wertübergabe arbeitet, sind diese Einstellungen vorgegeben.



| Parameter | Art | W... | O... | Typisier... | Bezugstyp | Defaultwert | Beschreibung |
|-----------|-----------|-------------------------------------|--------------------------|-------------|-----------|-------------|--------------|
| RE_NAME | Returning | <input checked="" type="checkbox"/> | <input type="checkbox"/> | Type | STRING | | Der Name |

Abbildung 6: Fertiger Returning-Parameter: SAP-System-Screenshot

Mit der Schaltfläche **Methoden** kehren Sie zum **Methoden**-Tab zurück. Damit die Methode implementiert werden kann, wird das entsprechende Attribut benötigt. Wechseln Sie daher zum Tab **Attribute**.



| Attribut | Art | Sic... | Re... | Typisier... | Bezugstyp | Beschreibung | Initialwert |
|----------|-----|--------|--------------------------|-------------|-----------|--------------|-------------|
| | | | <input type="checkbox"/> | Type | | | |
| | | | <input type="checkbox"/> | Type | | | |
| | | | <input type="checkbox"/> | Type | | | |

Abbildung 7: Attributliste im Class Builder: SAP-System-Screenshot

Geben Sie in der ersten Spalte, dem Namen des Attributs, **name** ein. Da es sich um ein Instanzattribut handelt, wählen Sie in der folgenden Spalte **Instance Attribute**. Die Sichtbarkeit (nächste Spalte) soll **Private** sein. Das Häkchen für Read-Only soll nicht gesetzt und die Typisierung bei **Type** belassen werden. Geben Sie den Bezugstyp **string** und eine passende Beschreibung an, und bestätigen Sie mit Enter.



| Attribut | Art | Sic... | Re... | Typisier... | Bezugstyp | | Beschreibung | Initialwert |
|----------|-----------|----------|--------------------------|-------------|-----------|---|--------------|-------------|
| NAME | Instan... | Priva... | <input type="checkbox"/> | Type | STRING |  | Der Name | |

Abbildung 8: Attribut in der Attributliste: SAP-System-Screenshot

Wechseln Sie nun zurück zum Tab **Methoden**. Positionieren Sie den Cursor in der Zeile, in der die Methode definiert wird. Sie können nun entweder mit der Schaltfläche Quelltext , oder durch Doppelklick auf den Methodennamen zur Implementierung wechseln. Bestätigen Sie die Nachfrage, ob die Klasse gesichert werden soll. Sie gelangen nun zum ABAP Editor, in dem Sie die Methode Implementieren können. Als Hilfsmittel können Sie sich die Signatur der Methode anzeigen lassen. Klicken Sie dazu auf die Schaltfläche **Signatur**.


| Art | Parameter | Typisierung | Beschreibu... |
|---|------------------|-------------|---------------|
|  | VALUE(RE_NAME) | TYPE STRING | Der Name |

Abbildung 9: Signaturanzeige: SAP-System-Screenshot

Fügen Sie im Code die erforderliche Zeile ein:



| Methode | GET_NAME | inaktiv |
|---------|--|---------|
| 1 |  method GET_NAME. | |
| 2 | re_name = name. | |
| 3 | endmethod. | |

Abbildung 10: Implementierung einer Methode: SAP-System-Screenshot

Speichern und prüfen Sie die Methode und kehren Sie mit Zurück oder F3 wieder zum Class Builder zurück.

Implementieren Sie analog auch eine Methode **set_name** zum Setzen des Namens. Fügen Sie anschließend die privaten Attribute **dob** (Geburtsdatum), **address** (Adresse) und **base_salary** (Grundgehalt) mit entsprechenden öffentlichen get- und set-Methoden hinzu. Implementieren Sie anschließend eine Methode **get_salary**, die analog zur zuvor implementierten lokalen Mitarbeiterklasse einen altersbezogenen Aufschlag auf das Grundgehalt zur Berechnung des Gehalts verwendet. Implementieren Sie dann auch eine **print**-Methode zur Ausgabe.

Die Klasse soll nun auch einen Konstruktor erhalten. Klicken Sie dazu auf die Schaltfläche

 **Konstruktor**. Es wird dadurch automatisch ein Instanzkonstruktor in die Methodenliste eingefügt:



| Methode | Art | Sicht... | M... | Beschreibung |
|-------------|------------|----------|---|--------------|
| CONSTRUCTOR | Instanc... | Public |  | CONSTRUCTOR |

Abbildung 11: Konstruktor in der Methodenliste: SAP-System-Screenshot

Fügen Sie dem Konstruktor Parameter zur Angabe von Name, Adresse, Geburtsdatum und Grundgehalt hinzu, und implementieren Sie ihn entsprechend.

Speichern und prüfen Sie die Klasse und aktivieren Sie alle inaktiven Repository-Objekte.

5.1.2 Praxis: Testen mit dem Class Builder

Bislang gibt es kein Programm, das die Klasse `ZCL_####_EMPLOYEE` verwendet, das zum Testen der Klasse verwendet werden könnte. Der Class Builder verfügt jedoch auch über ein eigenes Werkzeug zum Testen von Klassen, das Sie wie bei einem Programm über die Schaltfläche zum Testen  erreichen. Klicken Sie daher auf die Schaltfläche.

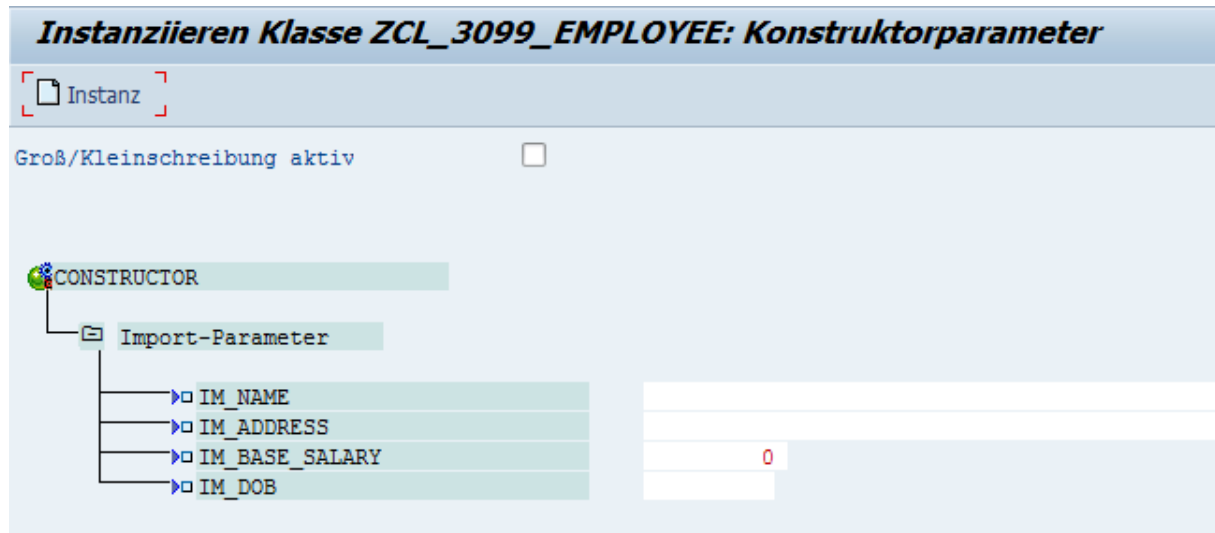



Abbildung 12: Aufruf des Konstruktors beim Testen einer Klasse: SAP-System-Screenshot

Mit dem Testwerkzeug kann nun eine Instanz der Klasse erzeugt werden. Dazu wird der Konstruktor aufgerufen und muss von Ihnen mit Parameterwerten versorgt werden. Geben Sie entsprechende Beispieldaten ein (*Hinweis: Datum im Format „ddmmyyyy“*) und klicken Sie auf .

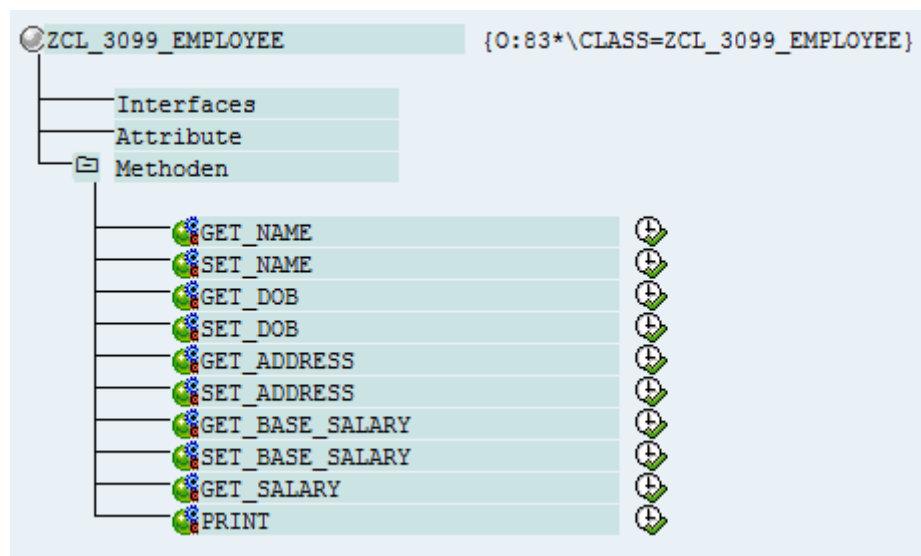



Abbildung 13: Testinstanz der Klasse: SAP-System-Screenshot

Sie sehen dort nun die öffentlichen Komponenten der Instanz. Da alle Attribute als private Attribute definiert wurden, sehen Sie hier nur Methoden. Klicken Sie auf das Symbol  neben dem Methodennamen `SET_BASE_SALARY`. Dadurch wird die Methode aufgerufen, und Sie können einen neuen Grundgehalt angeben.

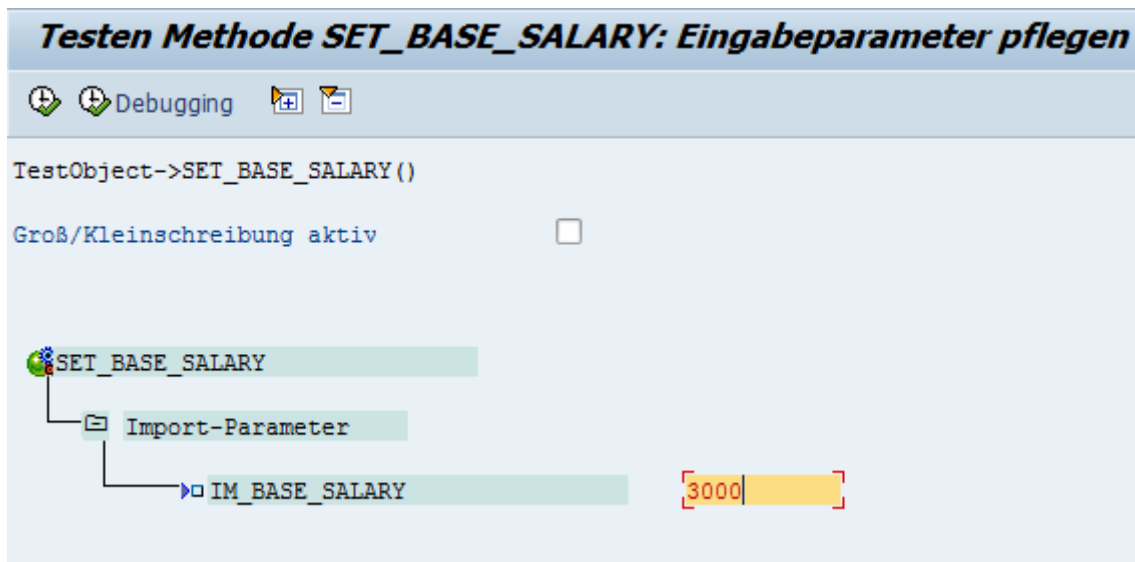



Abbildung 14: Testen einer Methode: SAP-System-Screenshot

Geben Sie einen Wert für den Importparameter an und Bestätigen Sie mit F8 oder . Rufen Sie anschließend auf dieselbe Weise die Methode get_base_salary auf. Das System zeigt Ihnen daraufhin den Wert des Parameters:

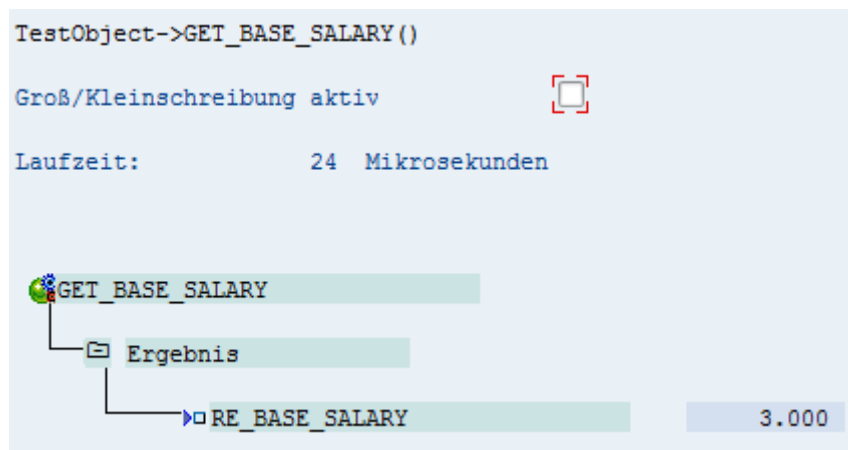


Abbildung 15: Rückgabewert einer Methode: SAP-System-Screenshot

Kehren Sie mit der Zurücktaste bzw. F3 zurück. Auch das Aufrufen von Methoden mit Ausgabefunktion ist mit dem Testtool möglich. Führen Sie zur Demonstration die print-Methode aus:

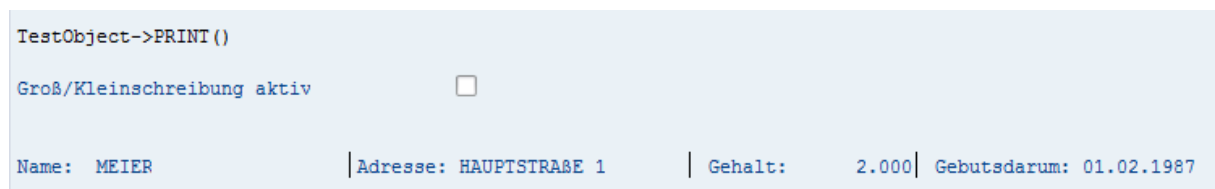



Abbildung 16: Ausgabe der PRINT-Methode: SAP-System-Screenshot

Der Class Builder bietet also sehr praktische Funktionen, um vor der Verwendung einer Klasse deren Funktionalität untersuchen zu können, ohne extra ein Programm dafür schreiben zu müssen. Das Testen ist direkt aus der Entwicklungsumgebung möglich.

5.1.3 Praxis: Erstellen einer abgeleiteten globalen Klasse

Als nächstes soll eine globale Klasse für Innendienstmitarbeiter angelegt werden. Geben Sie dazu auf dem Einstiegsbildschirm den Namen **ZCL_####_OFFICE_EMPLOYEE** ein und wählen Sie **Anlegen**.

Die Klasse soll von der vorhin angelegten Mitarbeiterklasse erben. Klicken Sie dazu auf das Symbol  (Vererbung Anlegen) neben dem Klassennamen. Es erscheint nun ein zusätzliches Feld, in das Sie den Namen der Oberklasse **ZCL_####_EMPLOYEE** eintragen:

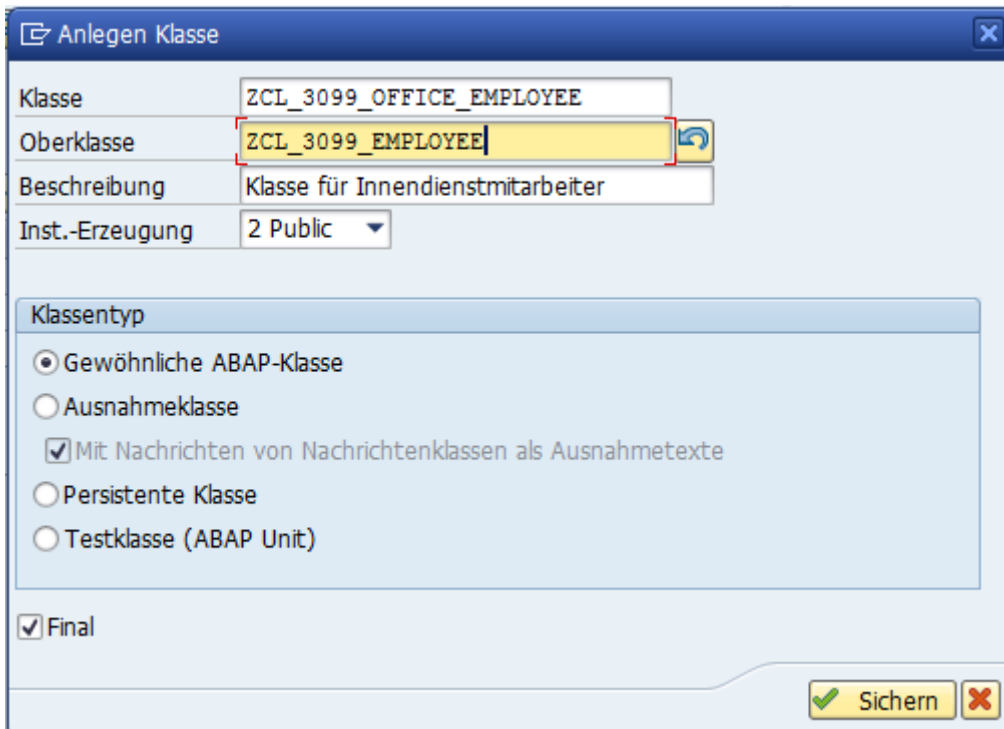


Abbildung 17: Anlegen einer abgeleiteten Klasse: SAP-System-Screenshot

In diesem Fall lassen Sie das Häkchen **Final** stehen. Eine finale Klasse kann keine Unterklassen haben. Sichern Sie und bestätigen Sie wie immer Paket und Transportauftrag.

| Methode | Art | Sic... | M... | Beschreibung |
|-----------------|----------|--------|------|-----------------------------|
| CONSTRUCTOR | Insta... | Pub... | | CONSTRUCTOR |
| GET_NAME | Insta... | Pub... | | Get-Methode für den Namen |
| SET_NAME | Insta... | Pub... | | Set-Methode für den Namen |
| GET_DOB | Insta... | Pub... | | Get-Methode für DOB |
| SET_DOB | Insta... | Pub... | | Set-Methode für DOB |
| GET_ADDRESS | Insta... | Pub... | | Set-Methode für Adresse |
| SET_ADDRESS | Insta... | Pub... | | Get-Methode für Adresse |
| GET_BASE_SALARY | Insta... | Pub... | | Get-Methode für Grundgehalt |
| SET_BASE_SALARY | Insta... | Pub... | | Set-Methode Grundgehalt |
| GET_SALARY | Insta... | Pub... | | Get-Methode für Gehalt |
| PRINT | Insta... | Pub... | | Drucken |

Abbildung 18: Geerbte Methoden: SAP-System-Screenshot

Sie sehen im Tab **Methoden** nun bereits die von der Oberklasse geerbten Methoden. Fügen Sie zunächst auf dem Tab **Attribute** das private Attribut **office** (Büro) hinzu. Implementieren Sie dann für das Attribut die get- und set-Methoden.

Als nächstes soll der Konstruktor erstellt werden. Klicken Sie dazu auf **Konstruktor**. Bestätigen Sie die Nachfrage, ob die Signatur des Oberklassenkonstruktors übernommen werden soll. So vereinfachen Sie sich die Arbeit: Sie müssen nur noch das Büro als Parameter hinzufügen. Nachdem Sie dies erledigt haben, implementieren Sie den Konstruktor. Sie sehen dort, dass der Aufruf des Superklassenkonstruktors bereits kommentiert vorhanden ist und so erneut Schreibarbeit gespart werden kann:

```

1  method CONSTRUCTOR.
2  *CALL METHOD SUPER->CONSTRUCTOR
3  *   EXPORTING
4  *       IM_NAME      =
5  *       IM_ADDRESS   =
6  *       IM_BASE_SALARY =
7  *       IM_DOB       =
8  *
9  endmethod.

```

Abbildung 19: Vorgegebenes Aufrufgerüst für den Superkonstruktor: SAP-System-Screenshot

Kehren Sie nach Implementierung des Konstruktors zurück zum Methoden-Tab. Als nächstes soll die Methode **print** redefiniert werden. Positionieren Sie dazu den Cursor auf dieser Methode (eventuell müssen Sie dafür hochscrollen) und klicken Sie auf die Schaltfläche (Redefinieren).

Hinweis: Eine Redefinition kann mit der daneben befindlichen Schaltfläche wieder aufgehoben werden.

Auch hier ist Ihnen der Aufruf der Methode aus der Oberklasse bereits als Kommentar vorgegeben und kann einfach eingebunden werden. Geben Sie zusätzlich noch das Büro aus

und speichern Sie. Speichern und prüfen Sie die Klasse und aktivieren Sie alle zu aktivierenden Elemente.

Testen Sie die print-Methode über die Testfunktionalität des Class Builders.

5.1.4 Praxis: Import lokaler Klassen

Das Anlegen einer globalen Klasse ist mit etwas Aufwand verbunden. Das liegt grundsätzlich in der Natur der Sache, ist aber dann ärgerlich, wenn dieselbe Klasse bereits als lokale Klasse vorliegt und so Dinge implementiert werden, die eigentlich schon im System vorhanden sind, nur nicht in der gewünschten Form. Um hier Abhilfe zu schaffen, bietet der Class Builder eine Importfunktion für lokale Klassen.

Starten Sie den Class Builder über die Transaktion **SE24** oder aus dem Easy-Access-Menü durch den Menüpfad **Werkzeuge -> ABAP Workbench -> Entwicklung -> Class Builder**. Ein Aufruf der Importfunktionalität direkt aus der ABAP Workbench ist **nicht** möglich.

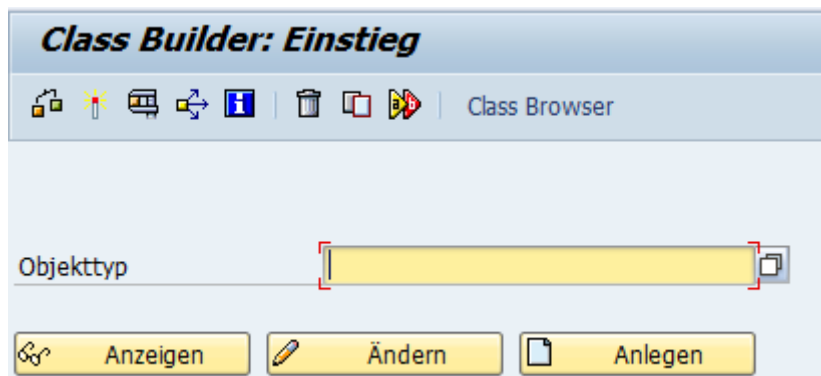


Abbildung 20: Einstiegsbild des Class Builders: SAP-System-Screenshot

Wählen Sie aus dem Menü den Pfad **Objektyp -> Importieren -> Programmlokale Klassen**.

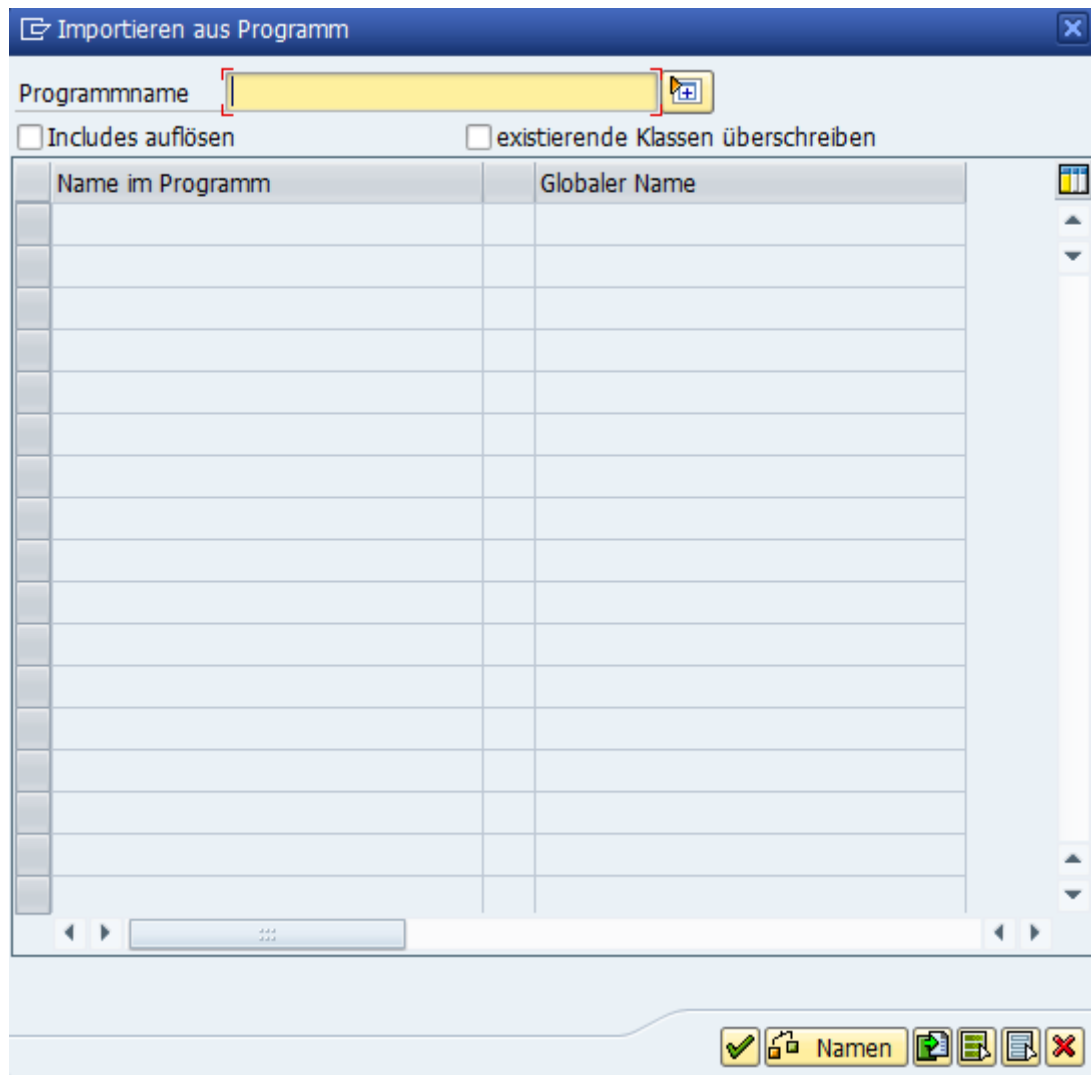


Abbildung 21: Importmaske für Programmlokale Klassen: SAP-System-Screenshot


Sie werden nun nach dem Namen des Programms gefragt, aus dem eine Klasse importiert werden soll. In unserem Fall soll die Funktionalität anhand der Klasse für Abteilungen demonstriert werden, die im Include `ZZ_####_COMPANY_2_CLASSES` definiert wurde. Geben Sie diesen Include-Namen daher in das Feld **Programmname** ein und klicken Sie auf . Es erscheinen daraufhin die lokalen Klassen und Interfaces, die im Include definiert sind. In der Liste taucht auch die Klasse `lcl_department` auf. In der rechten Spalte wird der Name angegeben, den die globale Klasse erhalten soll. Die Vorgabe liegt im SAP-Namensraum und muss daher angepasst werden. Tragen Sie dort **ZCL_####_DEPARTMENT** ein. Markieren Sie ferner die Zeile, indem Sie auf die leere Schaltfläche am Zeilenbeginn klicken:




Abbildung 22: Ausgewählte Klasse: SAP-System-Screenshot

Benennen Sie analog die Klasse `lcl_field_staff` als **ZCL_####_FIELD_STAFF** und markieren Sie diese Zeile ebenfalls.

Die anderen Klassen sollten Sie ebenfalls in dieser Maske analog umbenennen (`ZCL_####_...`), auch wenn Sie nicht importiert werden. Die Angabe des Namens wird vom

System dennoch verwendet, um Typisierungen anzupassen, wenngleich dabei nicht alle Typisierungen erfasst werden.

Stellen Sie sicher, dass die Abteilungs- und Außendienstmitarbeiter-Klasse selektiert sind, nicht aber die anderen Klassen denen lediglich der Name zugewiesen wurde.

Starten Sie dann den Import durch Klick auf  oder Drücken der F5-Taste. Geben Sie auf Nachfrage Paket und Transportauftrag an bzw. bestätigen Sie diese. Sie erhalten daraufhin eine Bestätigung:

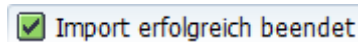


Abbildung 23: Import-Bestätigung: SAP-System-Screenshot

Geben Sie nun in der Einstiegsmaske des Class Builders unter **Objekttyp** den Namen ihrer soeben importierten Klasse, also **ZCL_####_DEPARTMENT** ein und klicken Sie auf **Ändern**. Die Klasse wird daraufhin im Änderungsmodus geöffnet.

Prüfen Sie nun die Klasse. Es erscheint eine Fehlermeldung:

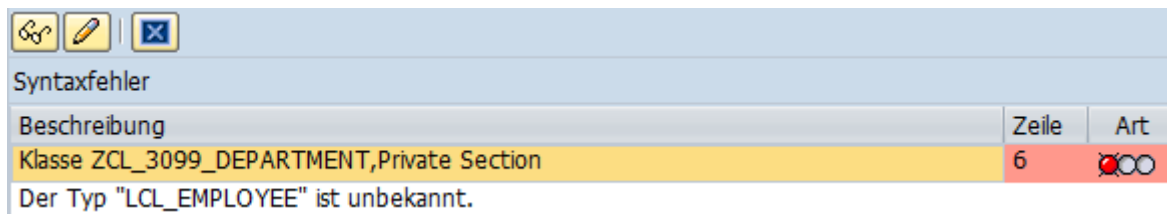


Abbildung 24: Fehlermeldung beim prüfen der importierten Klasse: SAP-System-Screenshot

Die Meldung entsteht dadurch, dass zur Typisierung (z. B. der Mitarbeiterliste) in der Klasse die bisherige lokale Klasse `lcl_employee` verwendet wird, die im globalen Kontext aber nicht zur Verfügung steht. Wechseln Sie daher zunächst zum Tab **Attribute**. Dort sind für das Attribut `it_employees` jedoch zahlreiche Felder inaktiv.

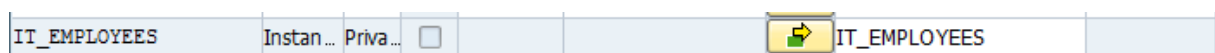



Abbildung 25: Das Attribut `it_employees`: SAP-System-Screenshot

Der Typ dieser internen Tabelle mit Referenzen ist nicht über die Felder abbildbar, sondern wird „direkt“ eingegeben. Klicken Sie auf  (Direkte Typeingabe). Korrigieren Sie im erscheinenden Quelltext die zur Typisierung verwendete Klasse auf den Namen ihrer globalen Mitarbeiterklasse, speichern Sie, und kehren Sie wieder zurück.

Stellen Sie nun sicher, dass auch der Parameter der **add_employee**-Methode korrekt typisiert ist. Wenn Sie beim Import auch die nichtimportierten Klassen „umbenannt“ haben, sollte das System den Parameter bereits korrekt auf die globale Klasse typisiert haben.

Öffnen Sie dann die Implementierung der **print**-Methode. Korrigieren Sie dort die Typisierung der Referenzvariablen, so dass Sie sich auf die globale Klasse bezieht.

In der Implementierung der Methode **get_avg_percentage** müssen ebenfalls die Typen angepasst werden. Ersetzen Sie dort `lcl_employee` und `lcl_field_staff` durch die Namen der entsprechenden globalen Pendanten. Speichern Sie die Methode anschließend, und klicken Sie doppelt im Code auf die dort nun zur Typisierung verwendete globale Außendienst-Mitarbeiterklasse. Wechseln Sie ggf. in den Änderungsmodus und öffnen Sie

den Tab **Eigenschaften**. Sie sehen, dass bei **Erbt von** die Klasse `ZCL_####_EMPLOYEE` eingetragen wurde (auch das ist eine Folge der Umbenennung beim Import; die importierte Klasse hatte ja zuvor eine lokale Klasse als Oberklasse). Speichern Sie die Klasse.

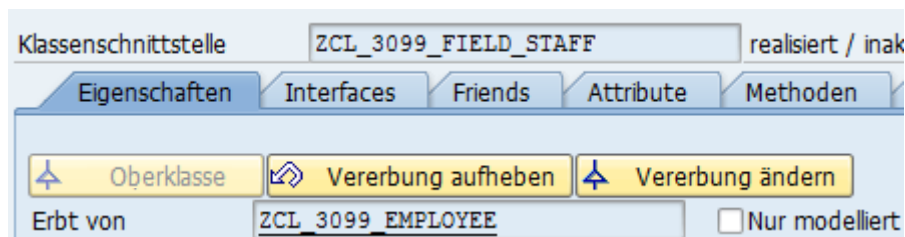


Abbildung 26: Vererbung im Class Builder: SAP-System-Screenshot

Prüfen Sie nun die Klasse. Da zur Vereinfachung bei der Klasse `ZCL_####_EMPLOYEE` kein Attribut `firstname` angelegt wurde, müssen der Konstruktor und der Superkonstruktoraufwurf korrigiert werden:

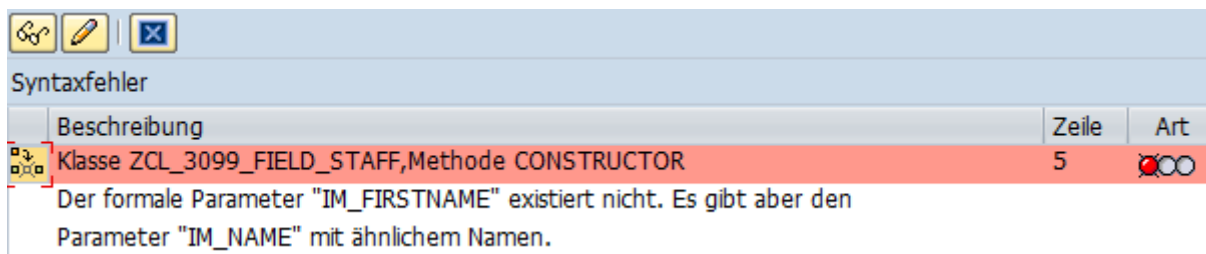



Abbildung 27: Fehlermeldung: SAP-System-Screenshot

Ein Doppelklick auf die Fehlermeldung führt Sie direkt zur Implementierung des Konstruktors der Klasse für Außendienstmitarbeiter. Entfernen Sie dort die Übergabe von `im_firstname`. Doppelklicken Sie dann in der Parameter-Information oberhalb des Quellcodes auf den Parameter `im_firstname`, um zur Parameterdefinition zu gelangen.

Entfernen Sie den Parameter dort mit . Bestätigen Sie die Nachfrage. Da der Oberklassenkonstruktor der globalen Klasse auch die Angabe der Adresse fordert, sollte der Unterklassenkonstruktor diese auch entgegennehmen, um Sie sinnvoll setzen zu können. Fügen Sie daher einen Import-Parameter `im_address` in die Parameterliste ein, und übergeben Sie diesen in der Implementierung des Konstruktors an den Oberklassenkonstruktor. Speichern Sie den Konstruktor und kehren Sie zur Klasse zurück.

Speichern und prüfen Sie die Klasse und aktivieren Sie anschließend alle zu aktivierenden Elemente. Öffnen Sie dann wieder die Department-Klasse und aktivieren Sie auch diese.

5.1.5 Praxis: Globale Klassen im Object Navigator

Verlassen Sie nun den Class Builder und öffnen Sie den Object Navigator (Transaktion **SE80**). Legen Sie ein neues Programm `ZZ_####_COMPANY_GLOBAL` an, das erneut kein TOP-Include besitzt, und bestätigen Sie Paket und Transportauftrag.

Klappen Sie den Navigationsbaum auf der linken Seite aus, so dass Sie Ihre globalen Klassen sehen können:

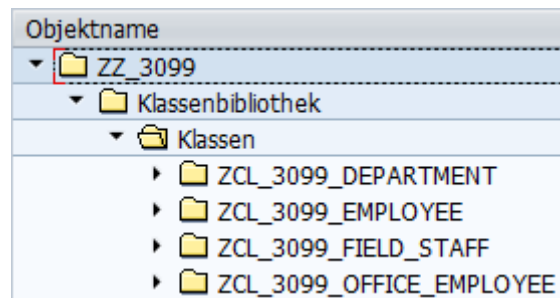


Abbildung 28: Klassen im Navigationsbaum: SAP-System-Screenshot

Erstellen Sie im Programmcode drei Referenzvariablen, die Sie mit den globalen Klassen für Abteilung, Außendienst- und Innendienstmitarbeiter typisieren. Lassen Sie danach mindestens eine Leerzeile, damit Sie mit den folgenden Codebausteinen nicht versehentlich in Ihre Definition geraten.

Zur Instanziierung einer Abteilung ziehen Sie mit der Maus den Klassennamen aus dem Navigationsbaum in Ihren Quelltext. Dort erscheint daraufhin ein Codegerüst:

```

15▶ CREATE OBJECT XXXXXXXXX
16▶     EXPORTING
17▶         IM_NAME =

```

Abbildung 29: Codegerüst für den Konstruktoraufruf: SAP-System-Screenshot

Ersetzen Sie XXXXXXXXX durch den Namen Ihrer Abteilungs-Referenzvariablen und geben Sie einen Wert für den Importparameter IM_NAME an.

Erstellen Sie auf die gleiche Weise je eine Instanz der anderen beiden Referenzvariablen. Klappen Sie dann den Navigationsbaum weiter auf, so dass auch die Methoden sichtbar werden:

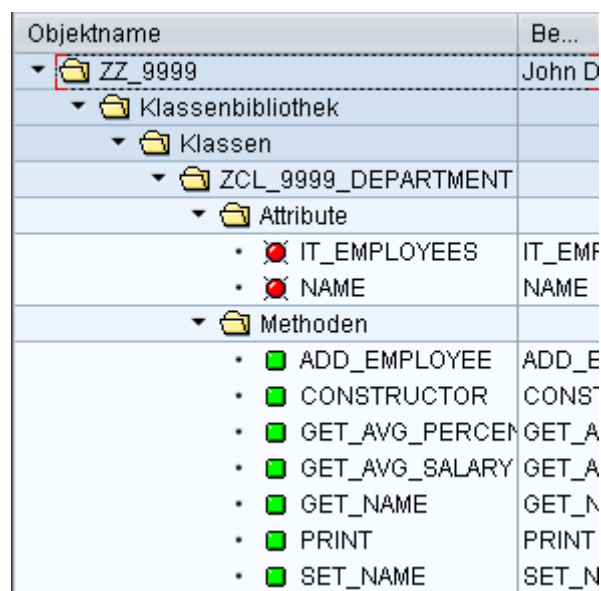


Abbildung 30: Methoden im Navigationsbaum: SAP-System-Screenshot

Ziehen Sie die Methode **ADD_EMPLOYEE** in den Quelltext. Es wird daraus ein Gerüst für den Methodenaufruf generiert:


```

35▶
36▶ CALL METHOD XXXXXXXX->ADD_EMPLOYEE
37▶     EXPORTING
38▶         IM_EMPLOYEE =
39▶         .
40▶

```

Abbildung 31: Methoden-Aufrufgerüst: SAP-System-Screenshot

Ersetzen Sie XXXXXXXX durch den Namen Ihrer Abteilungs-Referenzvariablen und geben Sie als zu importierenden Mitarbeiter die zuvor erstellte Außendienstmitarbeiter-Referenz an. Wiederholen Sie die Schritte, um auch den Innendienstmitarbeiter der Abteilung hinzuzufügen.

Rufen Sie schließlich die Ausgabemethode der Abteilung auf. Speichern, prüfen, aktivieren und testen Sie das Programm.

| | | | |
|--|-------------------------|---------------|--------------------------|
| Programm ZZ_3099_COMPANY_GLOBAL | | | |
| Abteilung: Mitarbeiter der Beispielabteilung | | | |
| Name: Meier | Adresse: Musterweg 1 | Gehalt: 5.002 | Geburtsdatum: Verkäufe: |
| Name: Müller | Adresse: Musterstraße 2 | Gehalt: 6.300 | Geburtsdatum: 10.12.1966 |

Abbildung 32: Ausgabe des Programms: SAP-System-Screenshot

5.1.6 Praxis: Der Refactoring-Assistent

Durch die Modellierung mit UML vor der Implementierung lässt sich in der Regel schon ein recht konkretes Bild vom Aufbau einer Software machen. Es kommt aber vor, dass durch nachträgliche Anforderungen, Erkenntnisse während der Implementierung o. ä. nachträgliche Änderungen wie etwa die Neuordnung von einzelnen Komponenten zu Klassen notwendig werden. Der Class Builder bietet hierfür mit dem **Refactoring-Assistenten** ein attraktives Hilfsmittel.

Als Szenario soll es zur Mitarbeiterklasse eine neue Oberklasse für Personen geben. Öffnen Sie den Object Navigator und legen Sie eine neue Klasse an, indem Sie mit der rechten Maustaste auf **Klassen** im Navigationsbaum klicken und **Anlegen** wählen.

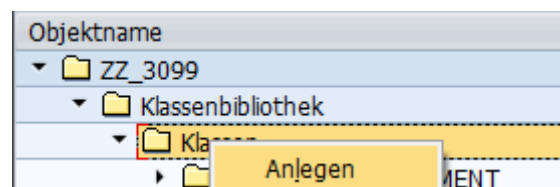


Abbildung 33: Anlegen einer Klasse: SAP-System-Screenshot

Nennen Sie die Klasse **ZCL_####_PERSON** und stellen Sie unbedingt sicher, dass das Häkchen final **nicht** gesetzt ist. Sichern Sie und bestätigen Sie Paket und Transportauftrag. Aktivieren Sie die Klasse, ohne dass Sie Attribute oder Methoden einfügen.

Öffnen Sie dann Ihre Klasse **ZCL_####_EMPLOYEE**. Stellen Sie sicher dass Sie sich im Bearbeitungsmodus befinden und öffnen Sie den Tab **Eigenschaften**. Klicken Sie dort auf **Superklasse**, um die Vererbungsbeziehung zur Personenklasse abzubilden. Es erscheint

ein neues Feld **Erbt von**. Tragen Sie dort die Klasse **zcl_####_person** ein. Speichern und prüfen Sie die Klasse. Sie erhalten eine Fehlermeldung:

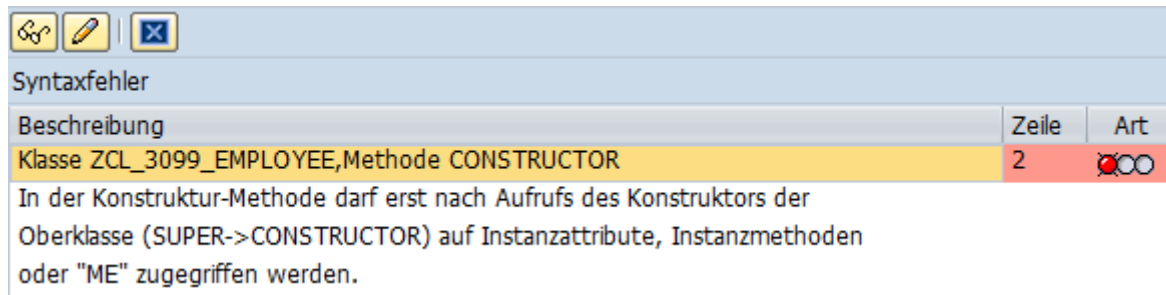



Abbildung 34: Fehler durch die Vererbungsbeziehung: SAP-System-Screenshot

Der Fehler tritt auf, weil die Klasse einen Konstruktor besitzt, der nun zunächst den Oberklassenkonstruktor rufen muss. Doppelklicken Sie auf die Fehlermeldung um zur Implementierung des Konstruktors zu gelangen, und fügen Sie dort den fehlenden Aufruf ein. Speichern, prüfen und aktivieren Sie die Klasse anschließend.

Gehen Sie nun zurück zur Klasse und wählen Sie aus dem Menü den Pfad **Hilfsmittel -> Refactoring Assistent**.

Expandieren Sie mit  alle Knoten des Baums.

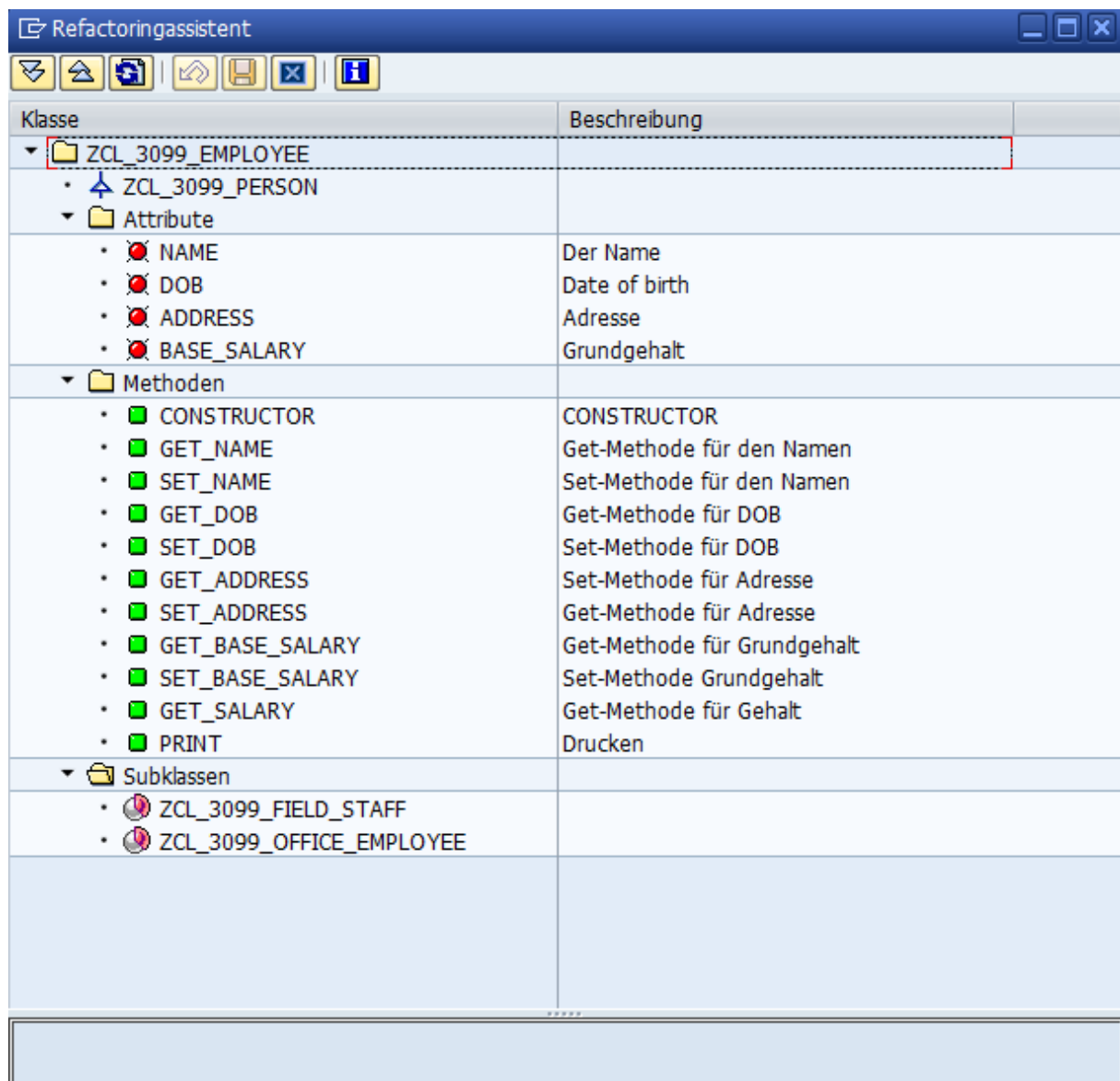


Abbildung 35: Der Refactoring-Assistent: SAP-System-Screenshot

Sie sehen nun die Oberklasse, die Attribute, die Methoden und die Unterklassen der Klasse **ZCL_####_EMPLOYEE**. Klicken und ziehen Sie nun die Attribute NAME, DOB und ADDRESS sowie deren get- und set-Methoden auf die Oberklasse.

Sie können im unteren Bereich des Fensters sehen, welche Komponenten wohin verschoben wurden:

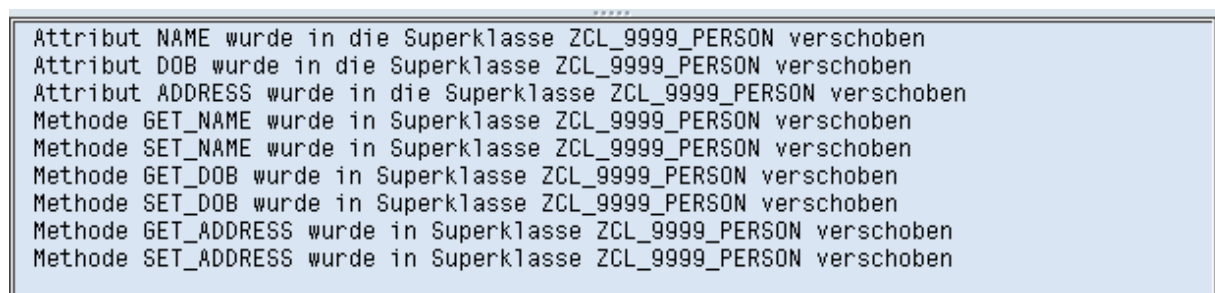


Abbildung 36: Verschobene Komponenten: SAP-System-Screenshot

Verschieben Sie außerdem den Konstruktor in die Oberklasse.

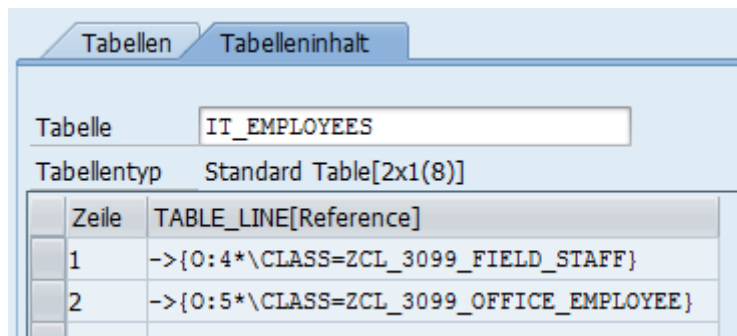
Sichern Sie nun das Refactoring mit  und beenden Sie den Assistenten.

Öffnen Sie die Klasse **ZCL_####_PERSON** und bearbeiten Sie den Konstruktor: Entfernen Sie das Grundgehalt als Parameter und im Code und entfernen Sie in letzterem den Superkonstruktoraufwurf. Speichern, prüfen und aktivieren Sie anschließend die Klasse.

Öffnen Sie nun wieder die Klasse **ZCL_####_EMPLOYEE** und legen Sie einen Konstruktor an. Bestätigen Sie die Frage, ob die Signatur des Oberklassenkonstruktors übernommen werden soll. Nun müssen Sie den Parameter für das Grundgehalt wieder einfügen und die Implementierung fertigstellen (Superkonstruktoraufwurf einkommentieren und vervollständigen und das Grundgehalt-Attribut setzen).

Wenn Sie in den Methoden **print** bzw. **get_salary** bislang ohne die entsprechenden get- und set-Methoden auf Attribute zugegriffen haben, die nun private Attribute der Oberklasse sind, müssen Sie den Code nun anpassen. Speichern, prüfen und aktivieren Sie die Klasse anschließend.

Öffnen Sie nun Ihr Programm **ZZ_####_COMPANY_GLOBAL** und vergewissern Sie sich dass es noch funktioniert. Setzen Sie einen Breakpoint auf die Zeile, in der die **CREATE OBJECT**-Anweisung zum Erstellen des Außendienstmitarbeiters steht. Testen Sie das Programm nun und schauen Sie sich im Debugger an, wie die Konstruktoren aufgerufen werden. Schauen Sie sich, nachdem beide Mitarbeiter in die Abteilung eingefügt worden sind, auch den Inhalt der internen Tabelle **it_employees** im Abteilungsobjekt an. Sie erreichen diesen durch Vorwärtsnavigation (Doppelklick). Dort können Sie zu jedem Mitarbeiter auch sehen, zu welcher Klasse er gehört, auch wenn die Tabelle über die Oberklasse **zcl_####_employee** typisiert ist.



| Zeile | TABLE_LINE[Reference] |
|-------|---|
| 1 | ->{0:4*\CLASS=ZCL_3099_FIELD_STAFF} |
| 2 | ->{0:5*\CLASS=ZCL_3099_OFFICE_EMPLOYEE} |

Abbildung 37: Inhalt der internen Tabelle im Debugger: SAP-System-Screenshot

5.1.7 Praxis: Erstellen eines globalen Interfaces

In dieser Aufgabe gehen wir davon aus, dass ein Verwender ein Interface definieren möchte, mit dem er über eine einheitliche Schnittstelle (eine Methode **print_object**) Objekte auf Reports ausgeben kann. Sie werden das Interface definieren und anschließend in Ihrer Klasse implementieren.

Öffnen Sie den Object Navigator und wählen Sie aus dem Navigationsbaum unterhalb Ihres Pakets den Knoten Klassenbibliothek aus. Klicken Sie diesen mit der rechten Maustaste an und wählen Sie **Anlegen -> Interface**. Geben Sie als Namen im Feld Interface den Wert

ZIF_####_PRINTABLE ein. Pflegen Sie eine passende Kurzbeschreibung und sichern Sie. Bestätigen Sie wie gewohnt Paket und Transportauftrag. Fügen Sie in die Liste der Methoden eine Instanzmethode **print_object** ein. Speichern, prüfen und aktivieren Sie das Interface.

Die Klasse **ZCL_####_EMPLOYEE** soll das Interface implementieren. Öffnen Sie die Klasse daher und wählen Sie die Registerkarte **Interfaces**. Fügen Sie dort in die Liste das von Ihnen definierte Interface ein. Speichern und prüfen Sie die Klasse. Sie erhalten die Meldung, dass eine Implementierung der Methode fehlt.

Wechseln Sie zur Registerkarte **Methoden**. Dort erscheint nun die Methode aus dem Interface. Erstellen Sie eine Implementierung, die die bereits vorhandene **print**-Methode aufruft. Speichern, prüfen und aktivieren Sie anschließend.

Starten Sie nun das Testwerkzeug des Class Builders und erzeugen Sie eine Instanz der Klasse. Es erscheint nun auch das Interface:

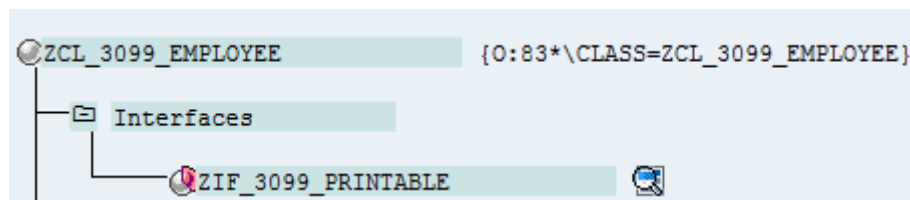




Abbildung 38: Interface im Testwerkzeug: SAP-System-Screenshot

Machen Sie den Inhalt des Interfaces durch Klick auf  sichtbar. Führen Sie dann die Interfacemethode aus und vergewissern Sie sich, dass die Ausgaben mit denen der **print**-Methode übereinstimmen.


5.1.8 Praxis: Weitere hilfreiche Funktionen

Wechseln Sie nun wieder zur Definition der Klasse **ZCL_####_OFFICE_EMPLOYEE**. Durch die nun definierte Vererbungshierarchie ist auf Anhieb nicht mehr zu sehen, in welcher Klasse die einzelnen Komponenten definiert wurden.

Um dieses Problem zu lösen, klicken Sie auf die Schaltfläche  (Einstellungen). Setzen Sie dort das Häkchen **Gruppieren nach Interfaces und Oberklassen**. Bestätigen Sie dann das Fenster. Die Darstellung der Methodenliste hat sich nun verändert: Die Methoden sind nach den Klassen gegliedert, in denen sie ursprünglich definiert wurden.

| Methode | Art | Sic... | M... | Beschreibung |
|----------------------|----------|--------|------|--------------------------------|
| <ZIF_9999_PRINTABLE> | | | | |
| PRINT_OBJECT | Insta... | Pub... | | Print the object |
| <ZCL_9999_PERSON> | | | | |
| GET_NAME | Insta... | Pub... | | Get method for the name |
| SET_NAME | Insta... | Pub... | | Set method for the name |
| GET_DOB | Insta... | Pub... | | Get method for the dob |
| SET_DOB | Insta... | Pub... | | Set method for the dob |
| GET_ADDRESS | Insta... | Pub... | | Get method for the address |
| SET_ADDRESS | Insta... | Pub... | | Set method for the address |
| <ZCL_9999_EMPLOYEE> | | | | |
| GET_BASE_SALARY | Insta... | Pub... | | Get method for the base salary |
| SET_BASE_SALARY | Insta... | Pub... | | Set method for the base salary |
| GET_SALARY | Insta... | Pub... | | Get the full salary |
| PRINT | Insta... | Pub... | | print the employee |
| CONSTRUCTOR | Insta... | Pub... | | CONSTRUCTOR |
| GET_OFFICE | Insta... | Pub... | | Get method for the office |
| SET_OFFICE | Insta... | Pub... | | Set method for the office |

Abbildung 39: Gegliederte Anzeige von Methoden: SAP-System-Screenshot

Sollte Ihnen auch dies noch nicht übersichtlich genug sein, haben Sie noch die Möglichkeit die Liste zu sortieren. Klicken Sie dazu auf  (Sortieren). Sie können dort wählen, ob nach Name, Sichtbarkeit oder Scope (statisch vs. instanzbezogen) sortiert werden soll, und können auch bis zu drei Sortierkriterien kombinieren.

5.1.9 Kontrollfragen

- Wie kann eine lokale Klasse in eine globale Klasse umgewandelt werden?
- Wie kann eine Klasse instanziiert werden, ohne manuell den Befehl `CREATE OBJECT` und die Formalparameternamen eingeben zu müssen?
- Wie können Ereignisse in der Testumgebung untersucht werden?

5.1.10 Antworten

1. Durch die Importfunktion des Class Builders. Beachten Sie aber, dass die lokale Klasse erhalten bleibt, also streng genommen lediglich eine Kopie in Form einer globalen Klasse erzeugt wird.
2. Durch Klicken und Ziehen des Klassennamens aus dem Navigationsbaum des Object Navigators in den Quellcode. Alternativ kann auch die Muster-Schaltfläche benutzt werden.
3. Durch Aktivieren der Behandlerfunktion des Testwerkzeugs für das betroffene Ereignis.

5.2 Spezielle objektorientierte Programmierkonzepte

In der Softwareentwicklung spricht man häufig von sog. **Patterns**. Dabei handelt es sich um Entwurfsmuster, die sich bei bestimmten, typischen bzw. wiederkehrenden Problemen bewährt haben. Ist ein solches Muster bekannt, kann es wie eine Schablone auf vergleichbare Probleminstanzen angewendet werden. Sie helfen nicht nur, ein gutes Design für ein Problem zu finden, sondern unterstützen auch die Kommunikation. So kann die Aufforderung, ein bestimmtes Pattern zu benutzen, eine langwierige Erläuterung einer Idee zur Implementierung ersetzen. Voraussetzung dafür ist natürlich, dass allen Beteiligten das jeweilige Pattern auch bekannt ist.

Beispiele für Patterns sind etwa das Iterator-Pattern (sequentieller Zugriff auf eine Struktur, ohne deren Details zu kennen) oder das Adapter-Pattern (Vermittlung zwischen Schnittstellen). In diesem Abschnitt wird Ihnen das Singleton-Pattern vorgestellt.

5.2.1 Das Singleton-Pattern

Es werden in der Regel drei Arten von Patterns unterschieden:

1. Erzeugungsmuster, die den Erzeugungsprozess von Objekten betreffen,
2. Strukturmuster, die die Zusammensetzung von Klassen bzw. Objekten zu größeren Strukturen betreffen, sowie
3. Verhaltensmuster, die sich der Interaktion von Klassen bzw. Objekten widmen.

Beim Singleton-Pattern handelt es sich um ein Erzeugungsmuster, das dafür sorgt, dass es von einer Klasse nur eine einzige Instanz gibt und diese Instanz nach Außen verfügbar macht. Angewendet wird das Pattern beispielsweise, um zentrale Puffer bereit zu stellen oder zentrales Logging durchzuführen.

Um ein solches Pattern in ABAP zu realisieren, muss sichergestellt sein, dass das beschriebene Verhalten in jedem Fall gewährleistet ist. Es wäre schlecht, wenn die Singleton-Eigenschaft nur auf Vereinbarungen beruht, etwa dass bei jedem Zugriff erst vom Verwender geprüft werden soll, ob eine Instanz existiert. Ein solches Vorgehen wäre Fehleranfällig und würde nicht dem Gedanken des Patterns entsprechen.

5.2.2 Praxis: Anwendung des Singleton-Patterns

Die notwendigen Schritte und Techniken sollen Ihnen hier gleich praktisch demonstriert werden. Erstellen Sie daher in Ihrem Paket eine neue Klasse mit dem Namen **ZCL_####_SINGLETON**. Direkt beim Anlegen der Klasse haben Sie die Möglichkeit, die Instanzerzeugung zu kontrollieren:

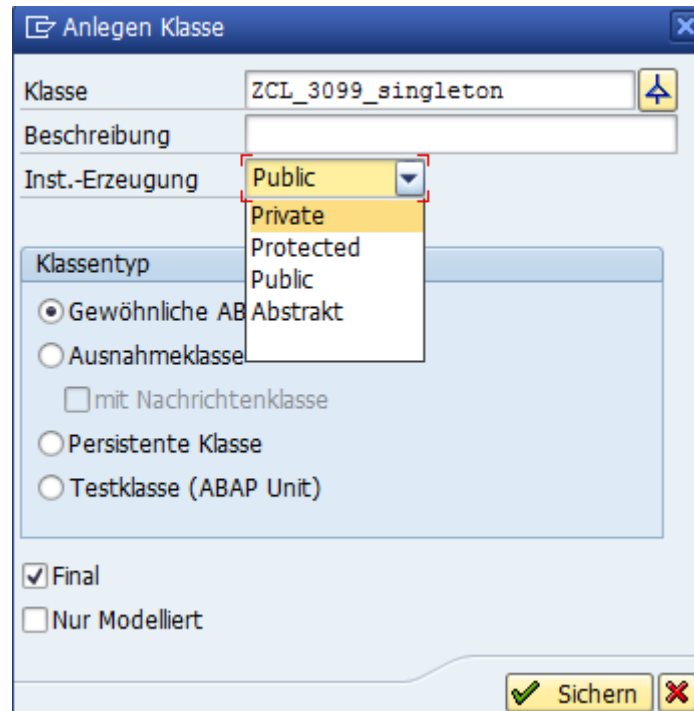


Abbildung 40: Anlegen der Klasse: SAP-System-Screenshot

Zur Auswahl stehen vier Optionen: Public, Private, Protected und Abstrakt. Die Option Abstrakt sorgt dafür, dass es sich um eine Abstrakte Klasse handelt, von der also keine Instanzen erzeugt werden können. Die anderen drei Optionen erlauben das Instanzieren aus dem jeweiligen Sichtbarkeitsbereich. Bisher haben Sie (von den abstrakten Klassen abgesehen) ausschließlich Klassen kennengelernt, bei denen von überall mittels `CREATE OBJECT` eine Instanz erzeugt werden konnte. Dies entspricht der Einstellung Public in diesem Feld. Bei der Einstellung Protected ist die Erzeugung nur in der eigenen Klasse oder in Unterklassen, bei der Einstellung Private nur in der eigenen Klasse möglich.

Diese Sichtbarkeitsangabe ist auch bei lokalen Klassen möglich. Dort ist ein Zusatz zur Klassendefinition erforderlich, um die Einschränkung vorzunehmen:

```
CLASS lcl_... DEFINITION CREATE PUBLIC.
...
ENDCLASS.
```

*Jeder Verwender kann
die Klasse instanzieren*

```
CLASS lcl_... DEFINITION CREATE PROTECTED.
...
ENDCLASS.
```

*Die Klasse kann nur durch
sich selbst oder ihre
Unterklassen instanziiert
werden*

```
CLASS lcl_... DEFINITION CREATE PRIVATE.
...
ENDCLASS.
```

*Die Klasse kann nur
durch sich selbst
instanziiert werden*

Abbildung 41: Konstruktorsichtbarkeit bei lokalen Klassen: SAP-System-Screenshot

Die Einstellung `CREATE PUBLIC` ist dabei der Defaultwert.

Die Definition des Instanzkonstruktors selbst erfolgt nur aus technischen Gründen in einem Sichtbarkeitsbereich der Klasse. Die tatsächliche Sichtbarkeit wird aber durch den gezeigten Zusatz zur Klassendefinition gesteuert. Die Definition des Konstruktors muss im Public-Sichtbarkeitsbereich stehen, es sei denn der Zusatz `CREATE PRIVATE` wurde gewählt.

Für unser Beispiel wählen Sie die Sichtbarkeit **Private**, damit nur die Klasse selbst eine Instanz erstellen kann.

Im unteren Bereich des Fensters finden Sie die Angabe, ob die Klasse final sein soll. Von einer als final gekennzeichneten Klasse dürfen keine Unterklassen abgeleitet werden. Auch hierbei handelt es sich nicht um ein Konzept, das allein globalen Klassen vorbehalten ist. Auch lokale Klassen können durch den Zusatz `FINAL` in der Definition der Klasse als final gekennzeichnet werden. Ebenso können Methoden final sein, was bedeutet, dass diese nicht redefiniert werden dürfen. Eine finale Klasse hat implizit nur finale Methoden, weshalb diese nicht extra als final markiert werden **dürfen**.

```
CLASS lcl_... DEFINITION FINAL
    [ INHERITING FROM ... ].
...
ENDCLASS.
```

*Klasse kann keine
Unterklassen haben*

```
CLASS lcl_... DEFINITION.
...
METHODS ... FINAL ...
...
ENDCLASS.
```


*Methode kann nicht
Redefiniert werden*

Abbildung 42: Finale Klassen und Methoden

Unsere Klasse `ZCL_####_SINGLETON` soll keine Unterklassen haben. Stellen Sie daher sicher dass das Häkchen **Final** gesetzt ist. **Sichern** Sie und bestätigen Sie wie gewohnt Paket und Transportauftrag.

Die Instanz der Klasse muss durch ein statisches Attribut festgehalten werden. Erstellen Sie daher zunächst ein **privates, statisches** Attribut `r_instance`, das als Referenz auf ein Objekt der Klasse selbst typisiert ist.

Als nächstes sollte für die Erzeugung der Instanz gesorgt werden. Sie werden sich möglicherweise gefragt haben, wie denn eine Klasse mit einem nur Privat erreichbaren Konstruktor instanziiert werden kann. Die Lösung ist hier eine statische Methode. Diese ist Teil der Klasse, hat also Zugriff auf die privaten Komponenten der Klasse, und ist bei entsprechender Sichtbarkeit, ohne dass eine Instanz existieren muss, von außen erreichbar. Es bietet sich an, für den Zweck der Instanziierung einen statischen Konstruktor zu verwenden. Wie bereits an anderer Stelle erläutert, wird der statische Konstruktor vor der ersten Verwendung der Klasse ausgeführt. So ist gewährleistet, dass die Instanz beim ersten Zugriff vorhanden ist, und eine Prüfung entfällt, ob die Instanz noch erzeugt werden muss.

Klicken Sie daher auf  **Klassenkonstruktor** um einen statischen Konstruktor anzulegen. Der statische Konstruktor erscheint daraufhin in der Liste der Methoden.


| Methode | Art | Sic... | M... | Beschreibung |
|--------------------------|----------|--------|--|--------------------------|
| CLASS_CONSTRUCTOR | Stati... | Pub... |  | CLASS_CONSTRUCTOR |

Abbildung 43: Statischer Konstruktor in der Methodenliste: SAP-System-Screenshot

Öffnen Sie durch Doppelklick auf den Methodennamen die Implementierung des Konstruktors. Bestätigen Sie die Nachfrage, ob die Klasse gesichert werden soll.

Instanziiieren Sie im Code des statischen Konstruktors die Klasse über das Attribut `r_instance`. Speichern Sie und kehren Sie zurück zur Methodenliste.

Es fehlt nun noch eine Methode, um die Instanz von außen erreichen zu können. Erstellen Sie dafür eine statische Methode **get_singleton**. Definieren Sie für diese Methode einen Rückgabeparameter `re_singleton` und typisieren Sie diesen als entsprechende Referenz. Implementieren Sie dann den Code der Methode, der lediglich aus einer Zuweisung der Instanzreferenz zum Rückgabeparameter besteht. Kehren Sie dann zur Klasse zurück und speichern, prüfen und aktivieren Sie diese.

Legen Sie als nächstes im Object Navigator ein Programm `ZZ_####_SINGLETON` an; auch dieses soll kein TOP-Include besitzen. Bestätigen Sie die gewohnten Nachfragen nach Paket und Transportauftrag.

Definieren Sie im Programm zwei Referenzvariable vom Typ der Singleton-Klasse, und instanziiieren Sie beide durch Aufruf der statischen Methode **get_singleton** der Singleton-Klasse. **Achtung:** Sie dürfen diese Methode nicht über die Referenzvariable und den Operator `->` aufrufen. Dies würde zu einem Absturz führen, da die Referenzvariable noch auf kein Objekt zeigt. Stattdessen müssen Sie die Syntax für einen statischen Methodenaufruf (Klassenname und Operator `=>`) verwenden, und den Rückgabewert der Referenzvariable zuweisen.

Setzen Sie dann einen Breakpoint auf die erste Instanziierung und testen Sie das Programm. Nutzen Sie den Einzelschrittmodus des Debuggers, um das Verhalten zu beobachten. Vor der ersten Instanziierung wird der statische Konstruktor ausgeführt. Bei der zweiten

Instanziierung wird hingegen direkt in die get-Methode gesprungen. Beide Referenzvariablen zeigen anschließend auf dasselbe, einzige Objekt der Singleton-Klasse.

5.2.3 Freundschaft

Es ist vorstellbar, dass eine Singleton-Klasse, die eine Pufferfunktion für Daten wahrnimmt, diese Daten mehreren anderen Klassen zur Verfügung stellen muss, da diese Klassen mit den Daten arbeiten. Der Hintergrund könnte eine Aufteilung der Funktionalität in mehrere Klassen und Pakete sein. Damit die Klassen auf die Daten der Singleton-Klasse zugreifen können, müsste diese die Daten öffentlich bereit stellen, was aber nicht unbedingt erwünscht ist.

Für solche und ähnliche Fälle gibt es in ABAP das Konzept der **Freundschaftsbeziehung** zwischen Klassen. Durch eine solche Beziehung kann eine Klasse auf private Komponenten einer anderen Klasse zugreifen.

Die Beziehung ist gerichtet: Gewährt eine Klasse A einer anderen Klasse B Freundschaft, so kann B auf die privaten Komponenten von A zugreifen, für den umgekehrten Fall müsste erst auch die Klasse B der Klasse A Freundschaft gewähren.

Wenn die Klasse A eine Unterklasse A2 und die Klasse B eine Unterklasse B2 besitzt, könnte bei weiterhin nur von A an B gewährter Freundschaft auch die Klasse B2 auf die privaten Komponenten von A zugreifen. Hier ist also bei Klassen mit vielen Unterklassen ggf. Vorsicht geboten. Die Klassen B und B2 können jedoch nicht auf private Komponenten von A2 zugreifen: Die gewährte Freundschaft wird vererbt (von B an B2), das gewähren von Freundschaft (von A an A2) hingegen nicht.

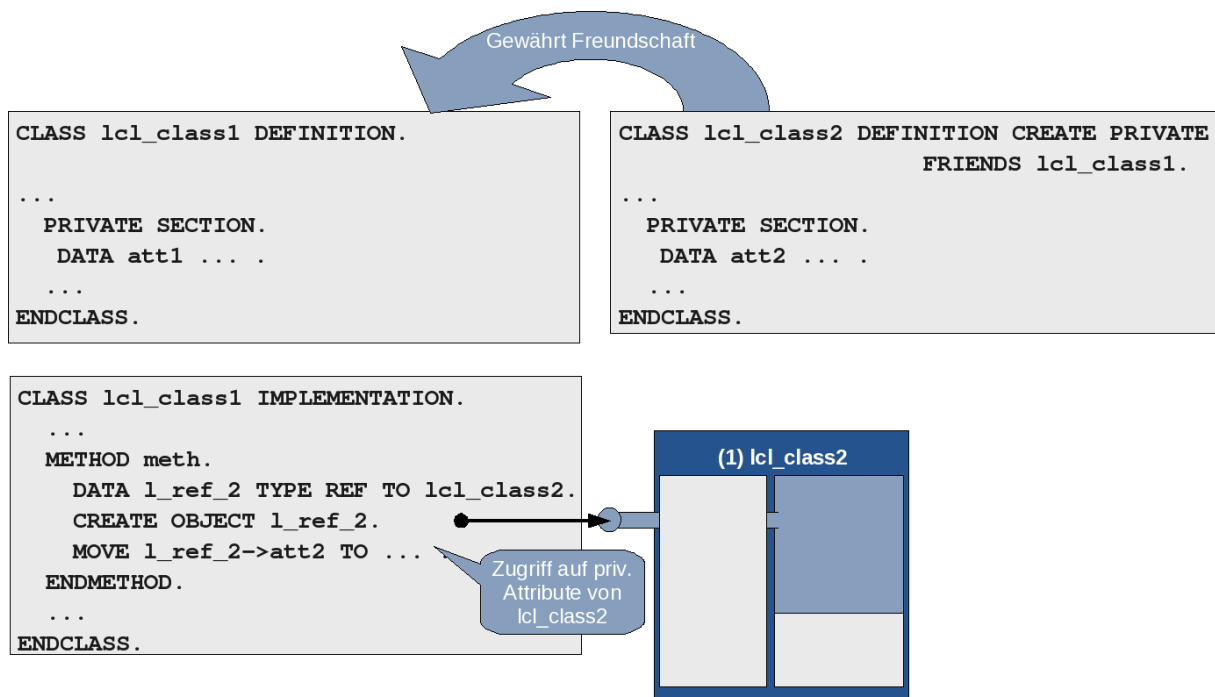



Abbildung 44: Freundschaftsbeziehung zwischen zwei Klassen

Auch Interfaces kann Freundschaft gewährt werden. Dies hat zur Folge, dass allen Klassen, die das Interface implementieren (auch über zusammengesetzte Interfaces), Freundschaft gewährt wird.

5.2.4 Praxis: Anwendung von Freundschaft

Als Beispiel sollen in der Singleton-Klasse Flugverbindungen gepuffert werden, die einer befreundeten Klasse zur Verfügung gestellt werden.

Erstellen Sie eine Klasse mit dem Namen **ZCL_####_FLINFO** unter Angabe von Paket und Transportauftrag. Erstellen Sie eine öffentliche, statische Methode **get_n_o_connections**. Diese soll eine Fluggesellschaft als Import-Parameter **im_carrid** erhalten (Bezugstyp **S_CARR_ID**) und die Anzahl der Verbindungen als Returning-Parameter **re_n_o_connections** zurückliefern. Definieren Sie diese Parameter, und speichern, prüfen und aktivieren Sie danach zunächst die Klasse **ZCL_####_FLINFO**. Damit die Methode implementiert werden kann, müssen zunächst von der Singleton-Klasse die entsprechenden Daten bereitgestellt werden.

Öffnen Sie die Singleton-Klasse. Um den Puffer zu definieren, wird ein geeigneter Typ benötigt. Wechseln Sie zur Registerkarte **Typen**. Hier können Sie Typen zur Verwendung in der Klasse definieren. Geben Sie dem neuen Typen den Namen **TY_CONNECTION_LIST** und die Sichtbarkeit **Private**. Klicken Sie dann in derselben Zeile auf  (Direkte Typeingabe). Bestätigen Sie, dass die Klasse gespeichert werden soll. Im angezeigten Code finden Sie nun die Zeile `types TY_CONNECTION_LIST`. Vervollständigen Sie diese, so dass der Typ ein Tabellentyp mit Zeilentyp **SPFLI** und den Schlüsselfeldern **carrid** und **connid** ist. Speichern Sie und kehren Sie zur Klasse zurück. Wechseln Sie zur Registerkarte **Attribute** und definieren Sie ein statisches, privates Attribut **CONNECTION_LIST**, das Sie mit dem soeben angelegten Typen typisieren.

Wechseln Sie nun zur Registerkarte **Methoden** und passen Sie den statischen Konstruktor so an, dass er die Flugverbindungen in die interne Tabelle **CONNECTION_LIST** einliest.

Als nächstes soll die Freundschaftsbeziehung hergestellt werden. Öffnen Sie dazu die Registerkarte **Friends**. Tragen Sie in die erste Zeile der Tabelle Ihre Klasse **ZCL_####_FLINFO** ein und bestätigen Sie mit Enter. Es erscheint daraufhin rechts die Beschreibung der Klasse.

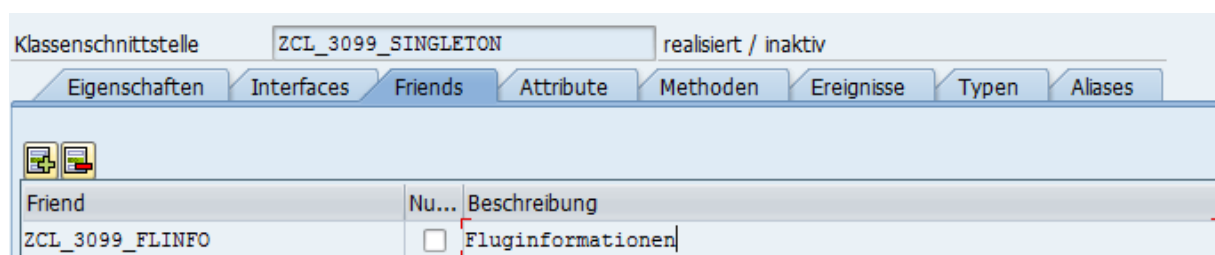


Abbildung 45: Herstellen einer Freundschaftsbeziehung: SAP-System-Screenshot

Kehren Sie zur Klasse zurück, speichern, prüfen und aktivieren Sie diese. Durch die Definition der Freundschaftsbeziehung kann die Klasse **ZCL_####_FLINFO** nun auf die private Liste mit den Flugverbindungen aus der Singleton-Klasse zugreifen.

Öffnen Sie wieder die Klasse **ZCL_####_FLINFO**. Implementieren Sie nun die Methode **get_n_o_connections**. Durchlaufen Sie dazu die interne Tabelle aus der Singleton-Klasse, zählen Sie die Anzahl und geben diese über den Returning-Parameter zurück. Speichern, prüfen und aktivieren Sie die Klasse und testen Sie diese mit den Bordmitteln des Class

Builders. Wählen Sie die statische Methode **get_n_o_connections** zum Ausführen aus und geben Sie das Kürzel einer Fluggesellschaft als Parameterwert ein, z. B. „AA“ (American Airlines):

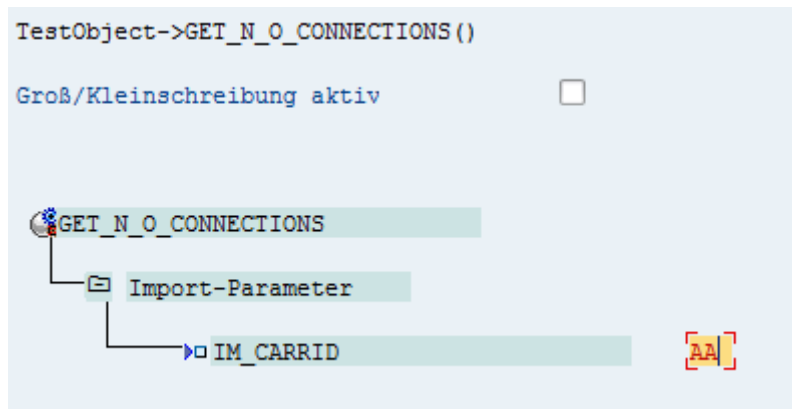



Abbildung 46: Ausführen der Methode: SAP-System-Screenshot

Klicken Sie dann auf , um die Methode auszuführen. Sie sollten daraufhin das korrekte Ergebnis sehen:

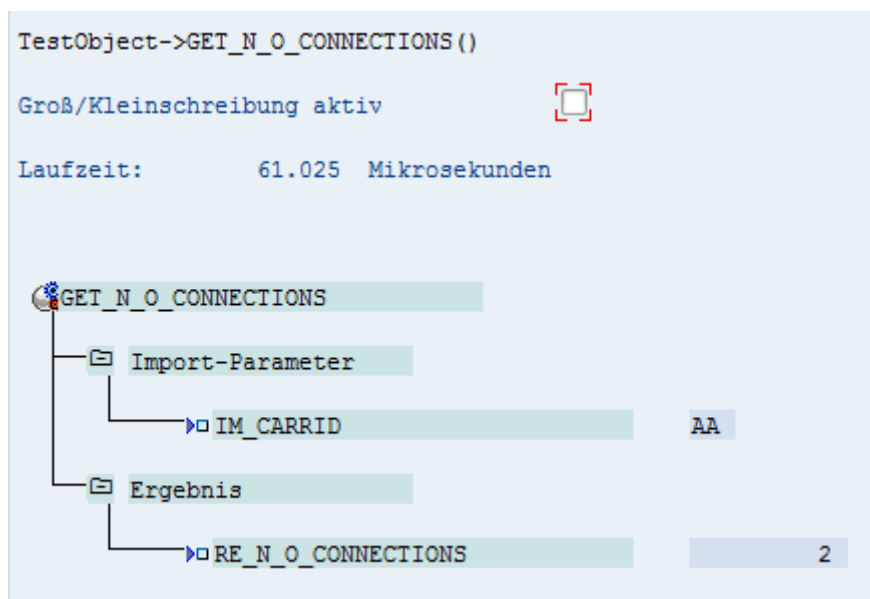


Abbildung 47: Ergebnis des Methodenaufrufs: SAP-System-Screenshot

Sie haben nun erfolgreich eine Methode implementiert, die auf eine private Komponente einer anderen Klasse zugreift. Dies war nur durch die Gewährung der Freundschaft durch diese Klasse möglich.

5.2.5 Kontrollfragen

1. Was sind finale Klassen bzw. finale Methoden?
2. Kann eine finale Klasse nicht-finale Methoden enthalten?
3. Kann eine nicht-finale Klasse finale Methoden enthalten?
4. Was versteht man unter einer Freundschaftsbeziehung?
5. Wie verhält sich die Freundschaftsbeziehung bei Vererbung?

6. Wie kann der Zugriff auf den Instanzkonstruktor einer lokalen Klasse eingeschränkt werden?
7. Kann in einer globalen Klasse ein lokaler Typ definiert werden?

Die Antworten finden Sie auf der nächsten Seite.

5.2.6 Antworten

1. Finale Klassen: Keine Bildung von Unterklassen möglich. Finale Methoden: Keine Redefinition möglich. Finale Klassen enthalten implizit nur finale Methoden.
2. Nein, finale Klassen enthalten implizit nur finale Methoden.
3. Ja.
4. Durch das Gewähren von Freundschaft erlaubt eine Klasse einer anderen Klasse den Zugriff auf ihre privaten Komponenten.
5. Erben einer Klasse, der Freundschaft gewährt wurde, wird auch Freundschaft gewährt. Unterklassen einer Freundschaft gewährenden Klasse erben das gewähren von Freundschaft jedoch nicht.
6. Durch den Zusatz `CREATE PROTECTED` bzw. `CREATE PRIVATE` bei der Klassendefinition.
7. Ja (siehe Übung zum Singleton-Pattern).

5.3 Shared Objects

Seit SAP NetWeaver 2004 gibt es die Möglichkeit, Objekte im Shared Memory abzulegen. Diese sind dann unabhängig von einem bestimmten Programm und auch unabhängig von der Benutzersitzung verfügbar. Diese Technik bietet sich zum Puffern von Daten an, die häufig gelesen, aber nur selten geschrieben werden, da so zeitintensive Datenbankzugriffe eingespart werden können, die gerade bei Anwendungen mit vielen Nutzern eine erhebliche Last darstellen können.

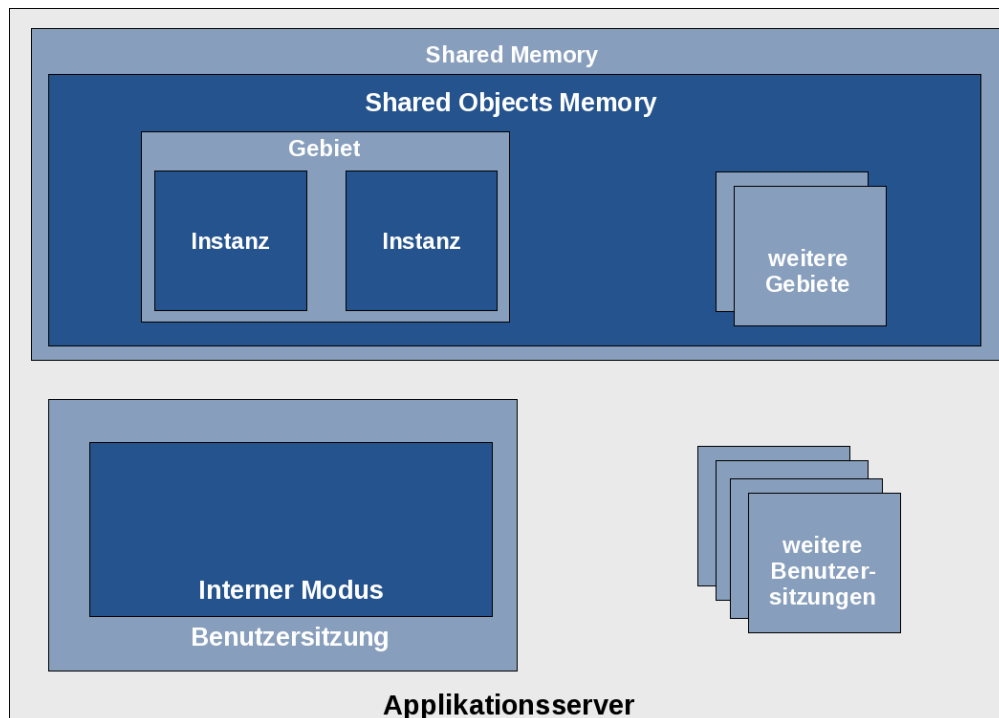


Abbildung 48: Einordnung des Shared Objects Memory

Für Shared Objects wird das Shared Objects Memory als Teil des Shared Memory verwendet. Hierbei handelt es sich um einen Speicherbereich, der sich auf dem Anwendungsserver befindet und gemeinsam von allen ABAP-Programmen des Anwendungsservers verwendet wird. Das Shared Objects Memory besteht aus Gebieten bzw. Gebietsinstanzen. Ein Gebiet

stellt eine Vorlage dar, von der mehrere benannte Instanzen angelegt werden können. Zu jedem Gebiet gibt es eine Default-Instanz.

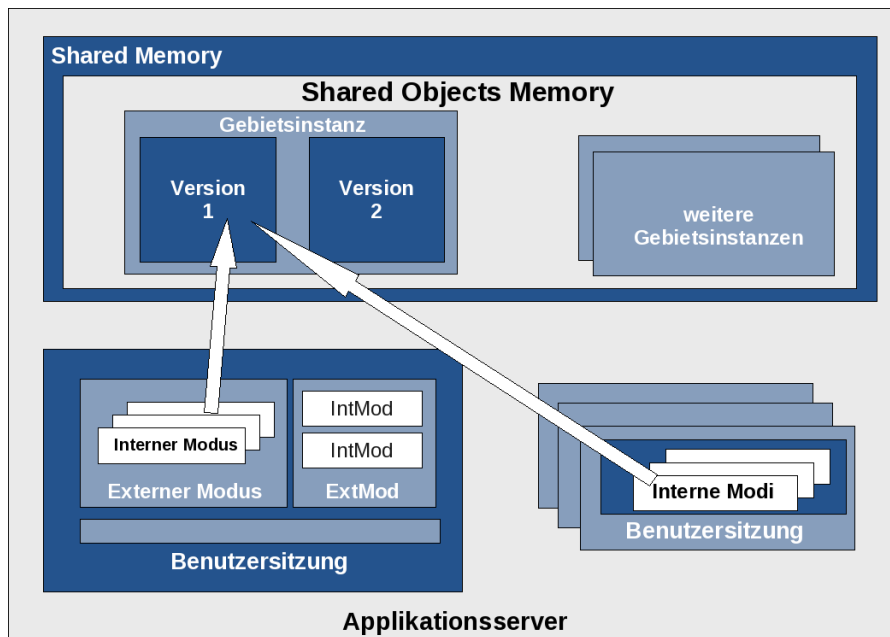


Abbildung 49: Zugriff auf Shared Objects

Von jeder Instanz kann es mehrere Versionen geben. Dies wird später noch erläutert. Auf die im Shared Objects Memory hinterlegten Objekte kann programm-, sitzungs- und benutzerübergreifend zugegriffen werden. Der Zugriff wird durch einen Sperrmechanismus koordiniert. Die Geschwindigkeitsvorteile durch die Verwendung des Shared Memory sind enorm, so gibt SAP an, durch die Verwendung in der ABAP Workbench selbst die Navigation beim ersten Zugriff um den Faktor 100 zu beschleunigen.

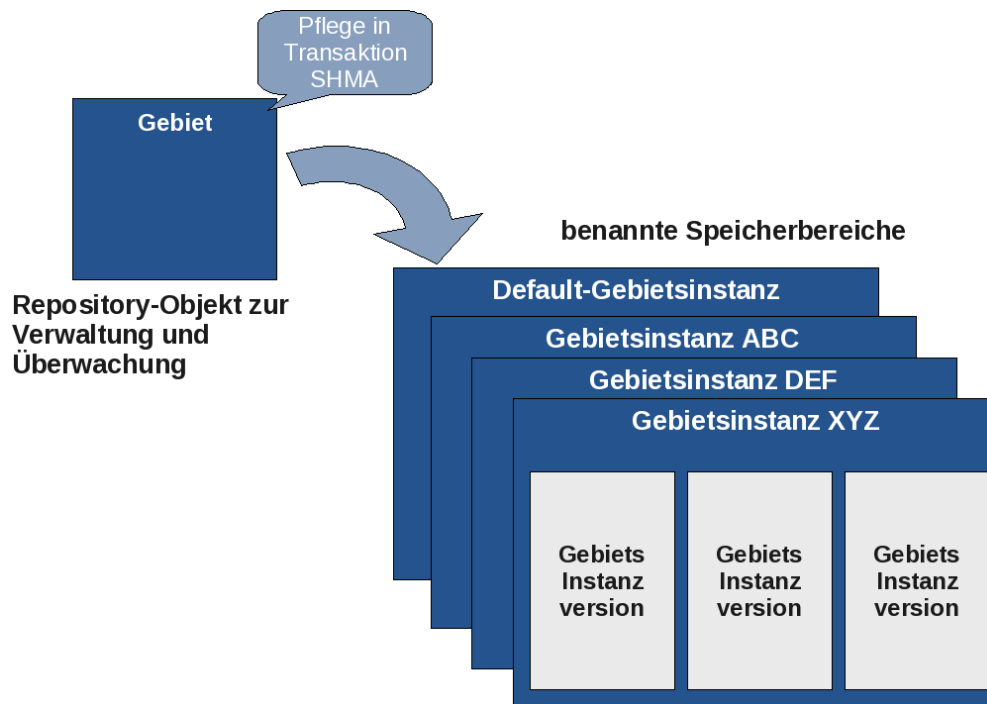


Abbildung 50: Gebiete und Gebietsinstanzen

Beim Anlegen eines Gebietes wird automatisch eine **Gebietsklasse** angelegt. Über diese Gebietsklasse erfolgen alle Zugriffe auf das Gebiet. Die Gebietsklasse ist eine globale, finale Unterklasse der Klasse CL_SHM_AREA und besitzt denselben Namen wie das Gebiet. Daher empfiehlt es sich, das Gebiet mit einem Namen der zur Namenskonvention für globale Klassen im Kundennamensraum zu benennen.

Die Gebietsklasse stellt eine Referenz auf eine sog. **Gebietswurzelklasse** bereit. Die Gebietswurzelklasse wird von Ihnen angelegt und enthält die eigentlichen Daten, die im Shared Memory liegen sollen, oder referenziert weitere Objekte im Shared Memory. Wichtig ist dabei, dass die entsprechenden Objekte Klassen angehören, die als **Shared Memory-fähig** gekennzeichnet wurden. Dies geschieht mit dem Zusatz SHARED MEMORY ENABLED der CLASS-Anweisung bzw. im Class Builder mit dem Häkchen Shared Memory fähig auf der Eigenschaften-Registrierkarte.

Die folgende Abbildung veranschaulicht den Zusammenhang.

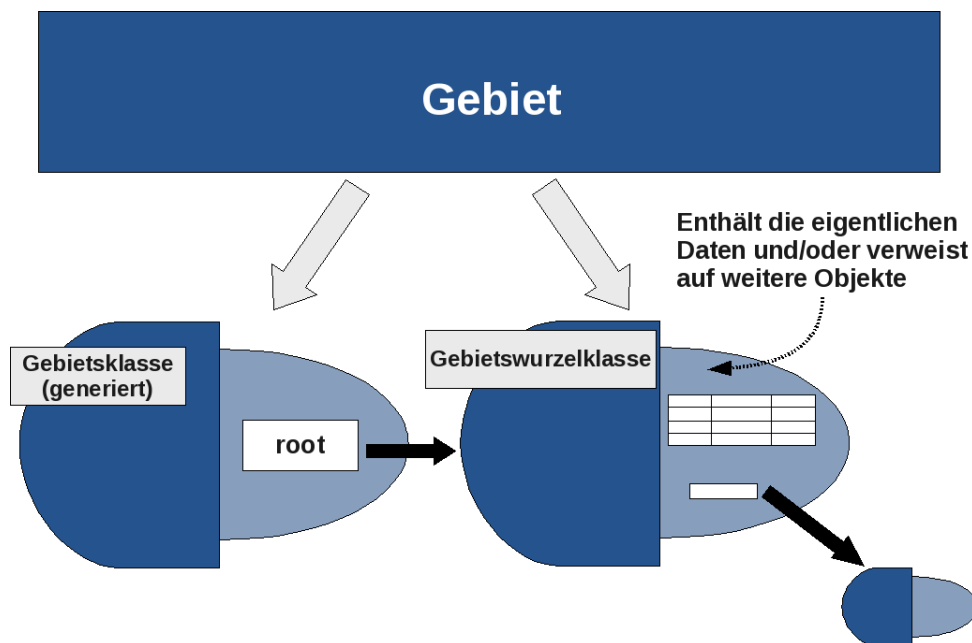


Abbildung 51: Gebiet, Gebietsklasse und Gebietswurzelklasse

Die Gebietsklasse stellt statische Methoden bereit, mit denen der interne Modus eines ABAP-Programms an eine Gebietsinstanz angebunden werden kann. Dabei wird ein sog. Gebietshandle erzeugt, über das später auf das Gebiet zugegriffen werden kann.

5.3.1 Praxis: Anlegen eines Gebietes

Um Ihnen die Funktionsweise an einem konkreten Beispiel zeigen zu können, werden Sie nun ein eigenes Gebiet anlegen. Dafür benötigen Sie zuvor aber eine Wurzelklasse.

Legen Sie daher zunächst in Ihrem Paket eine globale Klasse **ZCL_####_ROOT** an.

Wechseln Sie zur Registerkarte **Eigenschaften** und setzen Sie dort das Häkchen **Shared Memory fähig**.

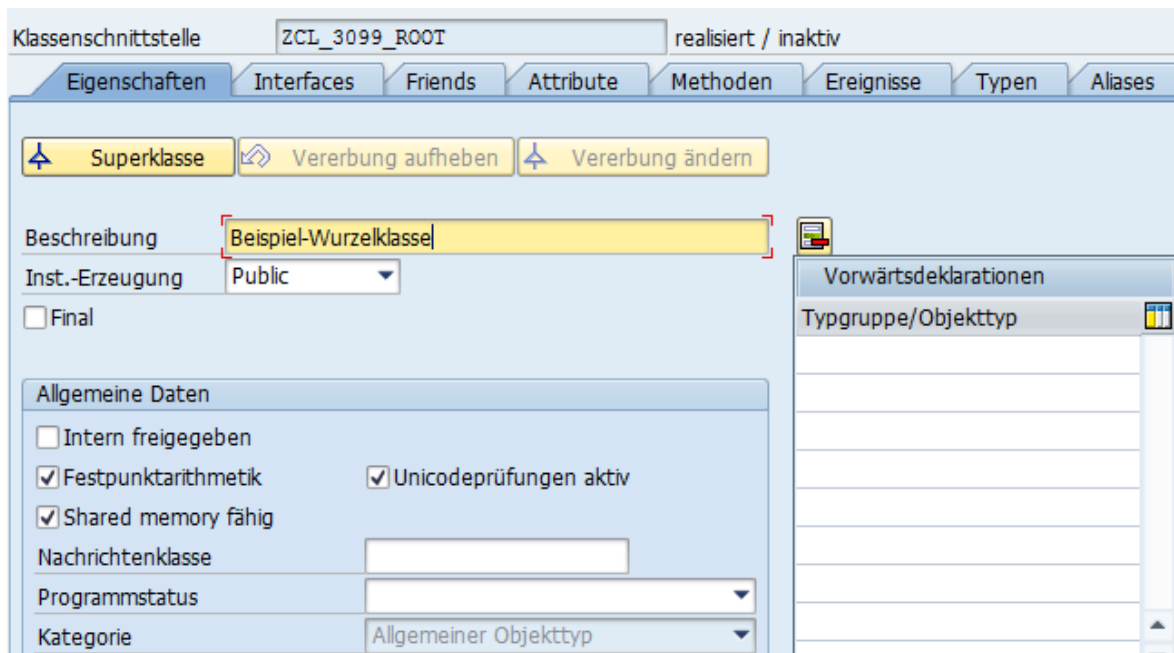


Abbildung 52: Eigenschaften der Wurzelklasse: SAP-System-Screenshot

Im Beispiel soll das Shared Memory benutzt werden, um die Flugverbindungen (vergleichbar mit dem Beispiel zur Freundschaft von Klassen) für einen Katalog zu puffern. Erstellen Sie dafür eine weitere Klasse mit dem Namen **ZCL_####_CATALOGUE**. Aktivieren Sie auch für diese Klasse die **Shared-Memory-Fähigkeit** und erstellen Sie ein (Instanz-)Attribut **connections** mit der Sichtbarkeit **Public**. Typisieren Sie dieses als Liste von Flugverbindungen indem sie auf der **Typen**-Registerkarte einen öffentlichen Typ definieren, der aufgebaut ist wie im Beispiel zur Freundschaftsbeziehung. Speichern, prüfen und aktivieren Sie die Klasse.

Öffnen Sie dann wieder die Klasse **ZCL_####_ROOT** und fügen Sie ihr ein öffentliches (Instanz-)Attribut **catalogue** hinzu, welches als Referenz auf die Klasse **ZCL_####_CATALOGUE** typisiert ist. Speichern, prüfen und aktivieren Sie auch diese Klasse.

Hinweis: Die Tabelle mit den Flugverbindungen hätte technisch auch direkt in der Wurzelklasse untergebracht werden können. Die zusätzliche Katalogklasse wurde aus didaktischen Gründen hinzugefügt, um Ihnen das arbeiten mit mehreren Objekten im Shared Memory zu demonstrieren.

Starten Sie dann die Transaktion **SHMA**. Mit dieser Transaktion werden Sie Ihr Gebiet anlegen. Geben Sie als Gebietsnamen **ZCL_####_AREA** an. Wie schon zuvor erläutert wird zum Gebiet eine Gebietsklasse gleichen Namens angelegt, weshalb dieser mit **ZCL_** beginnen sollte. Klicken Sie auf **Anlegen** und pflegen Sie eine passende Kurzbeschreibung. Setzen Sie im Bereich Grundeigenschaften das Häkchen **Mandantenabhängiges Gebiet**. Ein mandantenabhängiges Gebiet bezieht die Methoden zum Zugriff standardmäßig auf den aktuellen Mandanten. Wäre dieses Häkchen nicht gesetzt, würden das Gebiet und damit auch die Objekte darin mandantenunabhängig arbeiten. Stellen Sie ferner sicher dass das Häkchen **Transaktionales Gebiet** hier **nicht** gesetzt ist und geben Sie ihre Wurzelklasse an.

The screenshot shows the SAP 'Gebiet' (Area) configuration screen. The 'Name' field contains 'ZCL_3099_AREA' and the 'Beschreibung' (Description) field contains 'Beispiel für ein Gebiet'. The 'Attribute' tab is active. In the 'Grundeigenschaften' (Basic Properties) section, 'Wurzelklasse' (Root Class) is set to 'ZCL_3099_ROOT', 'Mandantenabhängiges Gebiet' (Client-dependent Area) is checked, and 'Automatischer Aufbau' (Automatic Structure) and 'Transaktionales Gebiet' (Transactional Area) are unchecked. In the 'Feste Eigenschaften' (Fixed Properties) section, 'Mit Versionierung' (With Versioning) is checked. In the 'Dynamische Eigenschaften' (Dynamic Properties) section, 'Konstruktorklasse' (Constructor Class) is empty, and 'Verdrängungsart' (Displacement Type) is set to 'Nicht verdrängbar' (Not Displaceable).

Abbildung 53: Anlegen des Gebiets: SAP-System-Screenshot

Sichern Sie unter Angabe Ihres Pakets und Transportauftrags und verlassen Sie das Gebiet.

5.3.2 Praxis: Aufbau einer Gebietsinstanz

Als nächstes soll vom definierten Gebiet nun eine Instanz erzeugt und Daten hinterlegt werden. Legen Sie dazu ein neues Programm **ZZ_####_SO_WRITER** an. Wählen Sie wie gewohnt kein TOP-Include und bestätigen Sie ihr Paket und Ihren Transportauftrag.

Definieren Sie in Ihrem Programm eine Referenzvariable **r_handle**, die mit Ihrer Gebietsklasse **ZCL_####_AREA** typisiert ist.

Weisen Sie nun der Referenzvariable den Rückgabewert der statischen Methode `attach_for_write` Ihrer Gebietsklasse zu (siehe auch folgende Abbildung).

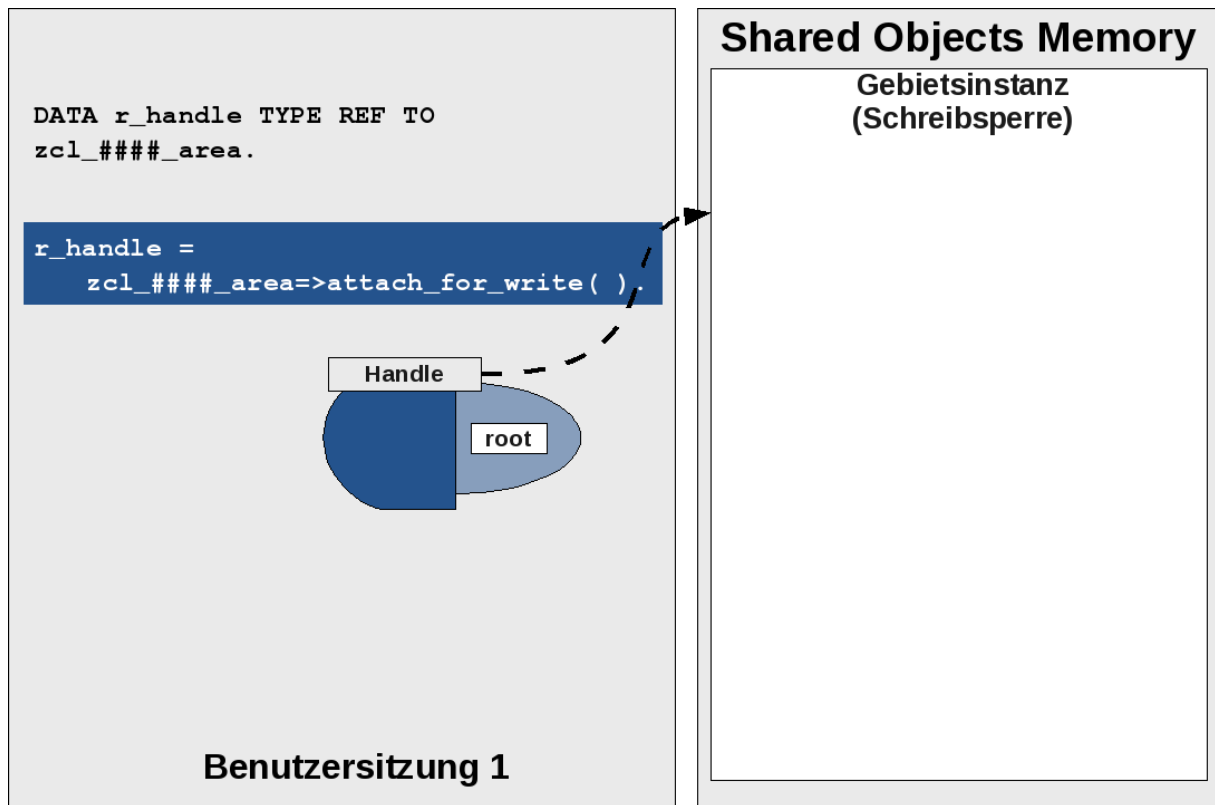


Abbildung 54: Erzeugen der Gebietsinstanz

Sie haben hierdurch nun eine Instanz ihres Gebiets erzeugt, auf die über das Handle zugegriffen werden kann. Alle folgenden Zugriffe erfolgen auf diesem Weg.

Als nächstes können Sie Objekte in ihrer Gebietsinstanz erzeugen. Definieren Sie dafür eine Referenzvariable **r_root** vom Typ Ihrer Wurzelklasse und eine Referenzvariable **r_catalogue** vom Typ Ihrer Katalogklasse. Um die Objekte im Shared Memory in Ihrer Gebietsinstanz zu erzeugen, verwenden Sie beim `CREATE OBJECT`-Befehl den Zusatz `AREA HANDLE r_handle`. Die Objekte werden daraufhin in der durch Ihr Handle gegebenen Shared Memory-Gebietsinstanz angelegt:

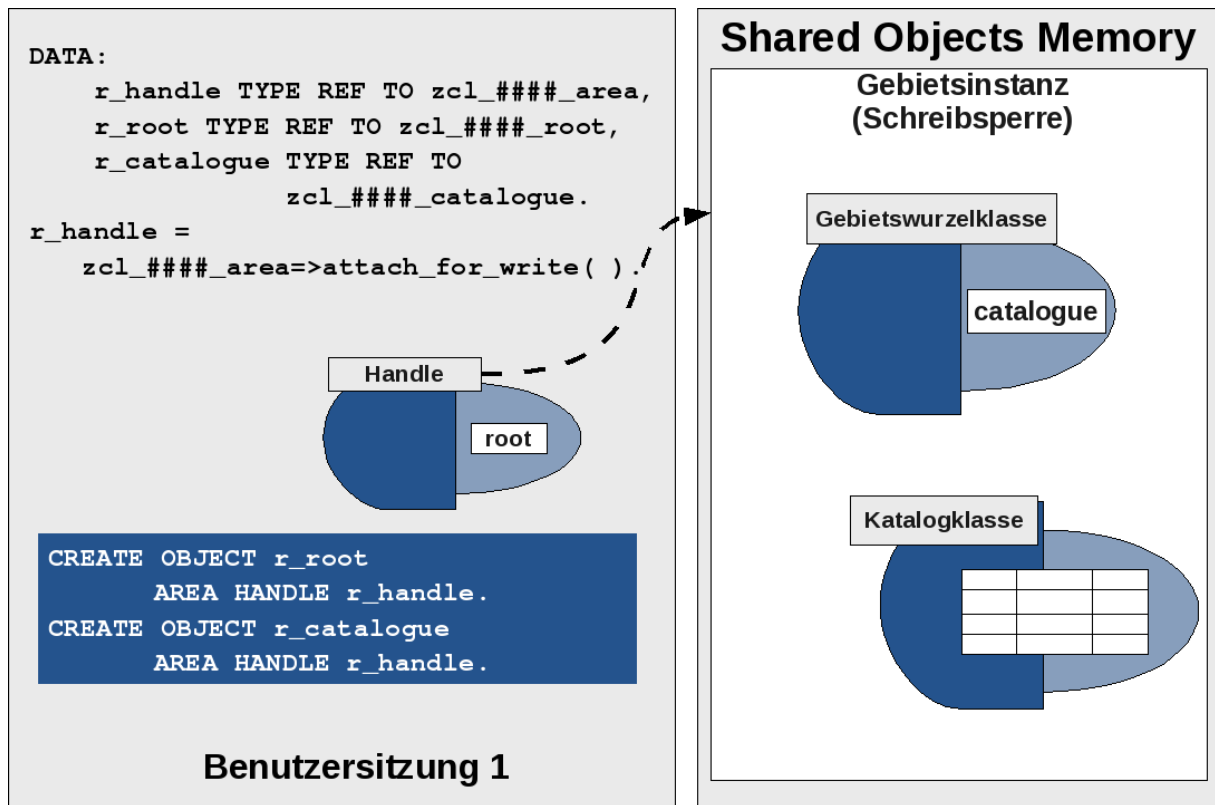


Abbildung 55: Anlegen von Objekten im Shared Memory

Die Objekte sind nun vorhanden, es fehlen jedoch noch Referenzen: Zum einen muss der Katalog noch von der Wurzelklasseninstanz (Attribut **catalogue**) referenziert werden, zum anderen muss die Wurzelklasseninstanz noch als Wurzelobjekt gesetzt werden. Nur so können die Objekte später wieder erreicht werden. Ein anderer Verwender muss sich dann nur eine Referenz auf die Gebietsinstanz besorgen, und kann dann über die Wurzelreferenz des Handle und die Katalogreferenz des Wurzelobjekts zum Katalogobjekt navigieren.

Die Katalogreferenz können Sie wie gewohnt zuweisen. Um das Wurzelobjekt zu setzen, verwenden Sie die Instanzmethode `set_root` des Handle.

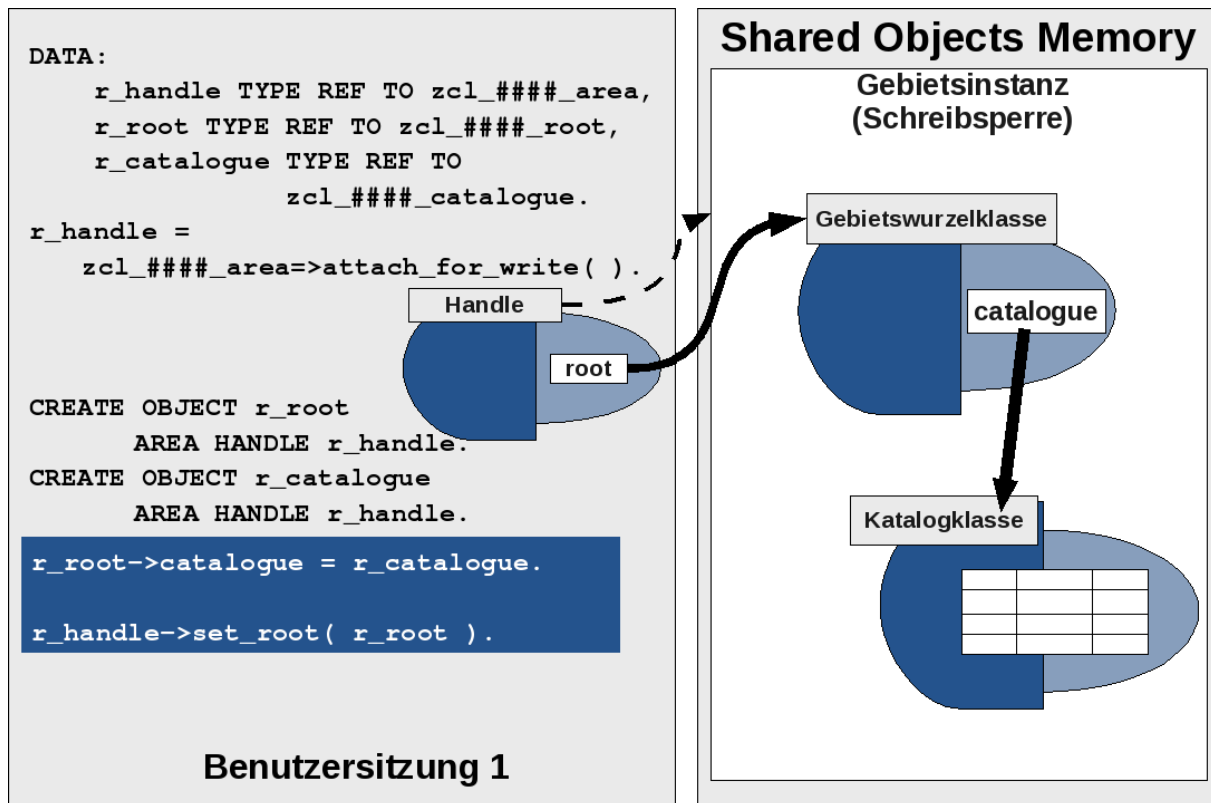


Abbildung 56: Setzen der Referenzen

Alternativ hätte das Katalogobjekt auch in einem Konstruktor der Wurzelklasse erzeugt werden können. Dann wäre dessen Erzeugung und Referenzsetzung hier nicht mehr erforderlich gewesen.

Als nächstes kann der Datenbankzugriff erfolgen. Lesen Sie mit einer geeigneten `SELECT`-Anfrage alle Flugverbindungen aus der Datenbanktabelle **SPFLI** in die interne Tabelle im Katalogobjekt ein.

Seit der Erzeugung der Gebietsinstanz war diese durch den `attach_for_write`-Befehl automatisch mit einer Schreibsperre gesperrt. Diese Sperre muss aufgehoben werden, da sonst keine lesenden Zugriffe auf die Gebietsinstanz möglich sind. Dies geschieht über die Methode **detach_commit** des Handles. Diese stammt aus der Klasse **CL_SHM_AREA** und wird von dieser an die Gebietsklasse vererbt.

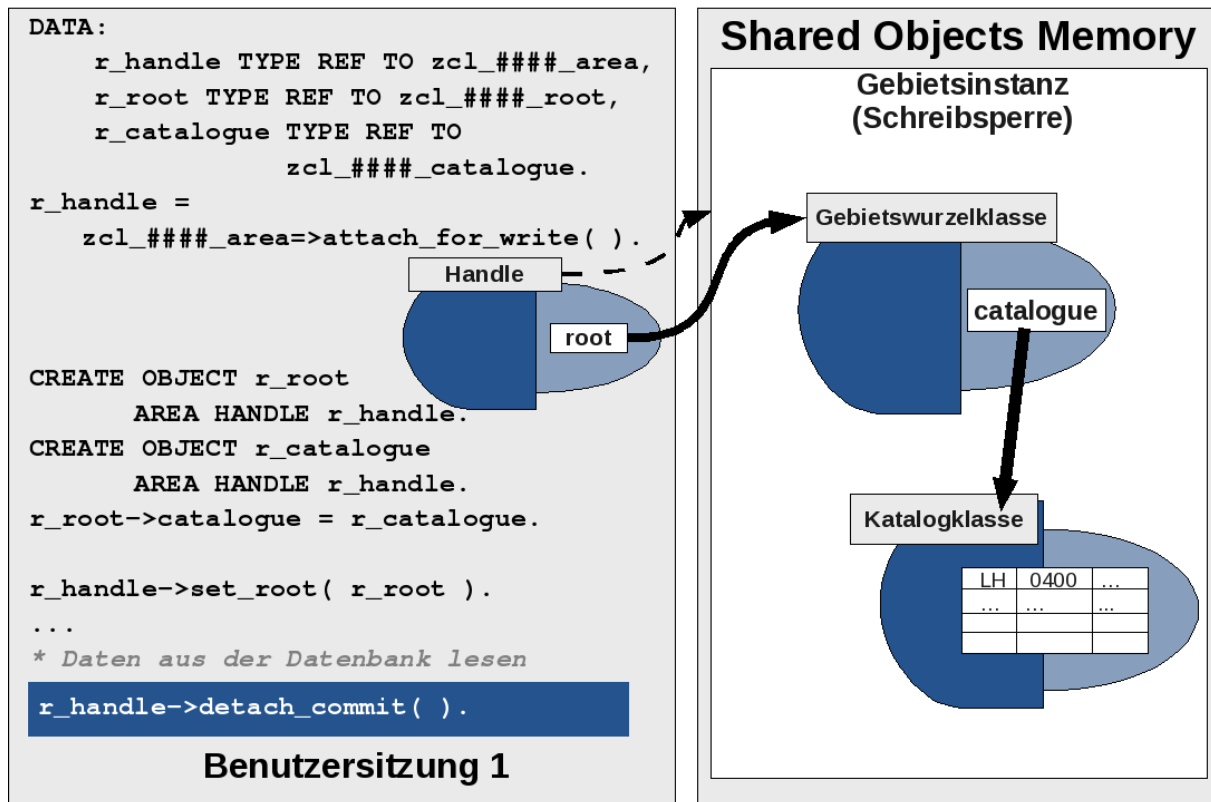


Abbildung 57: Lösen der Schreibsperre

Speichern, prüfen und aktivieren Sie Ihr Programm. Setzen Sie einen **Breakpoint** auf die Zeile, in der das Handle initialisiert wird und öffnen Sie in einem zweiten Modus die Transaktion **SHMM**. In dieser Transaktion können Sie das Shared Memory beobachten. Testen Sie nun Ihr Programm. Nach der `attach_for_write`-Anweisung erscheint Ihr Gebiet im zweiten Modus (klicken Sie zum Aktualisieren auf):

| SHM: Gebiete | | | | | | |
|---|-----------|-----------|---|---|---|--|
| Verwaltung Trace-Verwaltung | | | | | | |
| <div style="display: flex; justify-content: space-between;"> Gebiete Shared Objects Memory </div> | | | | | | |
| <div style="display: flex; justify-content: space-between; align-items: center;"> <div> </div> <div>Sicht: Übersicht</div> </div> | | | | | | |
| Gebiet | Instanzen | Versionen | | | | |
| CL_SANA_SHM_AREA | 2 | 2 | 0 | 0 | 2 | |
| CL_ICF_SHM_AREA | 1 | 1 | 0 | 0 | 1 | |
| ZCL_3099_AREA | 1 | 1 | 1 | 0 | 0 | |

Abbildung 58: Gebiet in der Transaktion SHMM: SAP-System-Screenshot

Doppelklicken Sie hier auf das Gebiet. Sie sehen dort Ihre Gebietsinstanz:

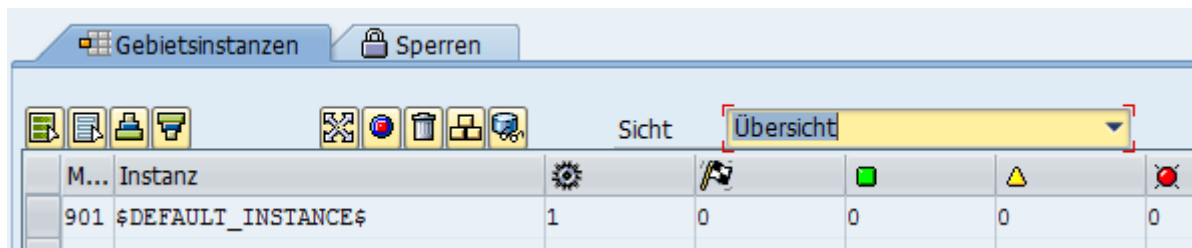



Abbildung 59: Gebietsinstanz: SAP-System-Screenshot

Öffnen Sie die Registerkarte **Sperren**. Sie sehen dort die Schreibsperre, die automatisch durch das System angelegt wurde und unerwünschte Zugriffe auf das Gebiet verhindert. Dass es sich um eine Schreibsperre handelt, können Sie der Spalte Typ entnehmen, indem Sie den Mauszeiger über dem Symbol  ruhen lassen.

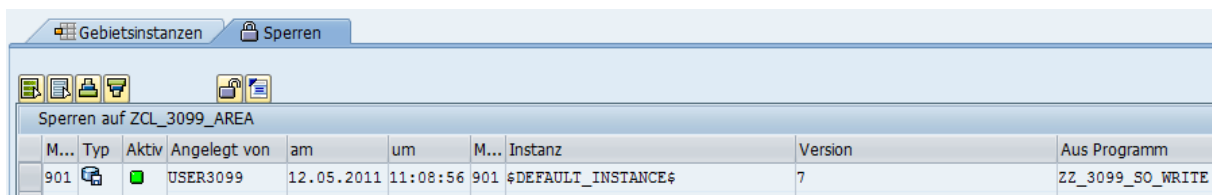


Abbildung 60: Schreibsperre: SAP-System-Screenshot

Fahren Sie im Debugging-Modus fort. Nach dem `detach_commit`-Befehl wird die Sperre wieder gelöscht und verschwindet beim Aktualisieren aus der Liste. Kehren Sie nun zum Object Navigator zurück.

5.3.3 Praxis: Verwenden einer Gebietsinstanz

Nachdem die Daten in der Gebietsinstanz hinterlegt wurden, können diese von anderen Programmen und anderen Benutzern verwendet werden.

Erstellen Sie ein neues Programm **ZZ_####_SO_READER**, ohne TOP-Include und unter Verwendung Ihres gewohnten Pakets und Transportauftrags. In diesem Programm benötigen Sie zunächst ebenfalls eine Referenzvariable `r_handle` vom Typ ihrer Gebietsklasse. In diesem Programm soll lesend auf die Gebietsinstanz zugegriffen werden. Hierfür verwenden Sie nicht die Methode `attach_for_write`, sondern die Methode `attach_for_read`:

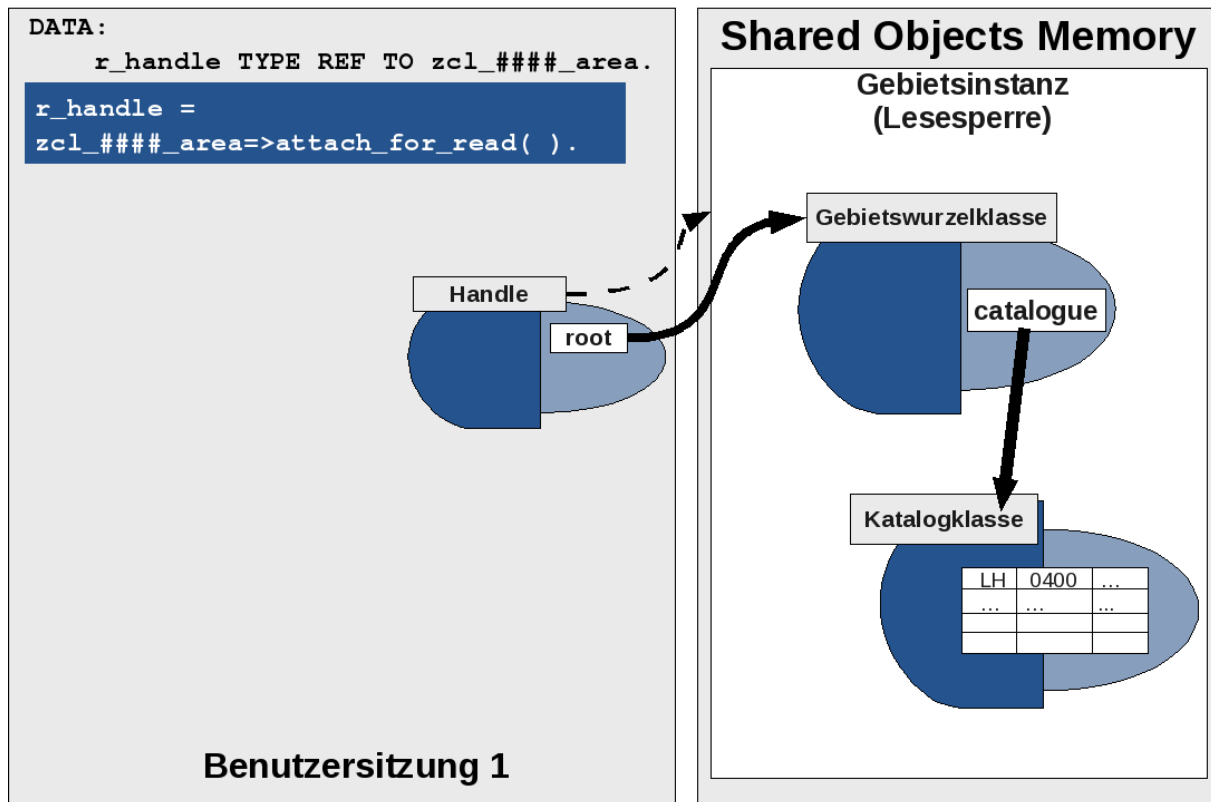


Abbildung 61: Lesender Zugriff auf die Gebietsinstanz

Zum Ausgeben der Flugverbindungen können Sie nun direkt über das Attribut `root` des `Handle` zum Katalogobjekt und dessen Flugverbindungstabelle navigieren (durch mehrfaches verwenden des Operators `->`). Möglicherweise fragen Sie sich, warum vorhin für den Zugriff auf `root` eine `set`-Methode erforderlich war. Es handelt sich hier um ein öffentliches Attribut mit **Read-Only**-Status. Somit sind lesende Zugriffe direkt möglich.

Definieren Sie einen Arbeitsbereich vom Typ `SPFLI`, und durchlaufen Sie damit die Flugverbindungen. Geben Sie zu jeder Verbindung die Komponenten `CARRID`, `CONNID`, `CITYFROM` und `CITYTO` aus.

Statt der `detach_commit`-Methode beim Schreibzugriff kann beim lesenden Zugriff mit der `detach`-Methode die Sperre explizit aufgehoben werden.

Speichern, prüfen, aktivieren und testen Sie Ihr Programm. Die Ausgabe sollte etwa wie folgt aussehen:

| Report ZZ_3099_SO_READER | | | |
|---------------------------------|------|-----------|---------------|
| Report ZZ_3099_SO_READER | | | |
| AZ | 0555 | ROME | FRANKFURT |
| AZ | 0790 | ROME | OSAKA |
| AZ | 0788 | ROME | TOKYO |
| LH | 0400 | FRANKFURT | NEW YORK |
| LH | 0402 | FRANKFURT | NEW YORK |
| UA | 3517 | FRANKFURT | NEW YORK |
| JL | 0408 | FRANKFURT | TOKYO |
| UA | 0941 | FRANKFURT | SAN FRANCISCO |
| QF | 0006 | FRANKFURT | SINGAPORE |
| LH | 2402 | FRANKFURT | BERLIN |
| DL | 0106 | NEW YORK | FRANKFURT |
| LH | 0401 | NEW YORK | FRANKFURT |

Abbildung 62: Ausgabe des Programms: SAP-System-Screenshot

Sollte stattdessen ein Kurzdump erscheinen, der auf die Ausnahme CX_SHM_NO_ACTIVE_VERSION hinweist, wurde das Programm ZZ_####_SO_WRITER nicht ausgeführt oder ist fehlerhaft. Sie können diese Ausnahme auch in einem TRY-Block abfangen.

Sie haben nun Flugverbindungen aus dem Shared Memory gelesen, die zuvor von einem anderen Programm dort hinterlegt wurden. Für den lesenden Zugriff war kein Datenbankzugriff erforderlich.

5.3.4 Versionen von Gebietsinstanzen

Zu einer Gebietsinstanz können mehrere Versionen existieren, sofern diese Möglichkeit aktiviert ist. Die entsprechende Konfiguration ist Ihnen bereits beim Anlegen des Gebietes begegnet, weiterhin werden in der Transaktion **SHMM**, die Sie beim Debugging benutzt haben, die Versionen dargestellt.

Die Versionierung spielt dann eine Rolle, wenn auf einer Gebietsinstanz geschrieben wird. Wie Sie bereits wissen, wird während des Schreibvorgangs automatisch eine Schreibsperre auf die Gebietsinstanz gesetzt. Gäbe es keine Versionen, könnten in dieser Zeit keine anderen Zugriffe auf die Gebietsinstanz erfolgen. Darüber hinaus ist je nach Frequentierung der Anwendung(en) davon auszugehen, dass zu einem Zeitpunkt relativ viele Nutzer Lesesperren auf die Instanz gesetzt haben.

Im Normalfall gibt es nur genau eine Version. Diese ist die aktive Version und wird von den lesenden Anwendern verwendet und mit Lesesperren belegt:

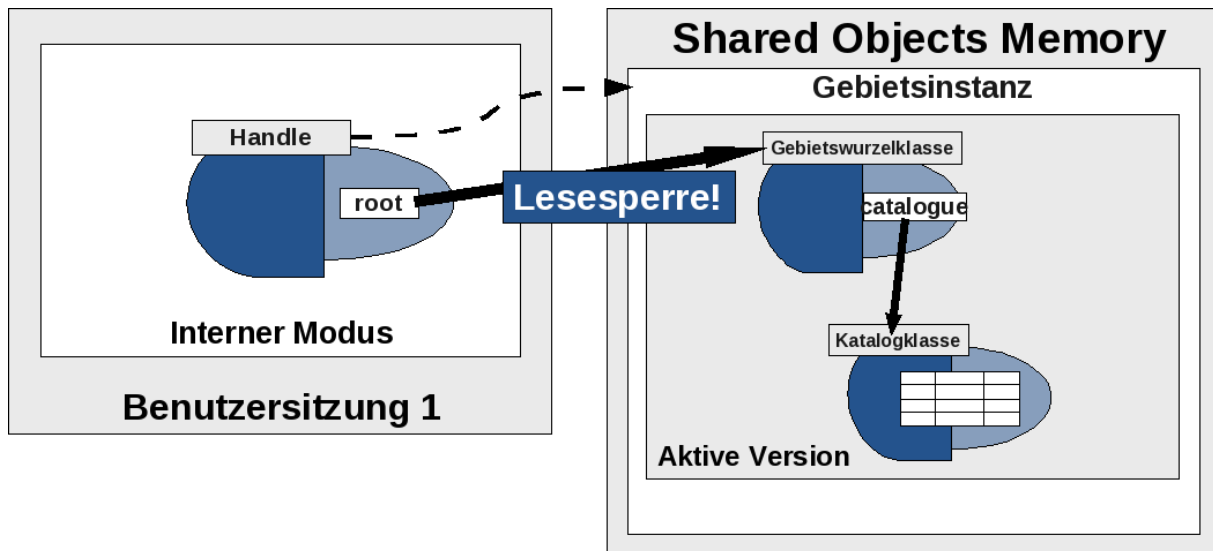


Abbildung 63: Gewöhnlicher Zustand mit Lesesperre auf aktiver Version

Neben der aktiven Version können, wenn die konfigurierte Versionsanzahl dies zulässt, mehrere weitere Versionen existieren. Beim Setzen einer Schreibsperre auf die Gebietsinstanz durch eine Anwendung wird eine zusätzliche Version (Version im Aufbau) angelegt. Diese existiert parallel zur aktiven Version, die weiterhin verwendet werden kann.

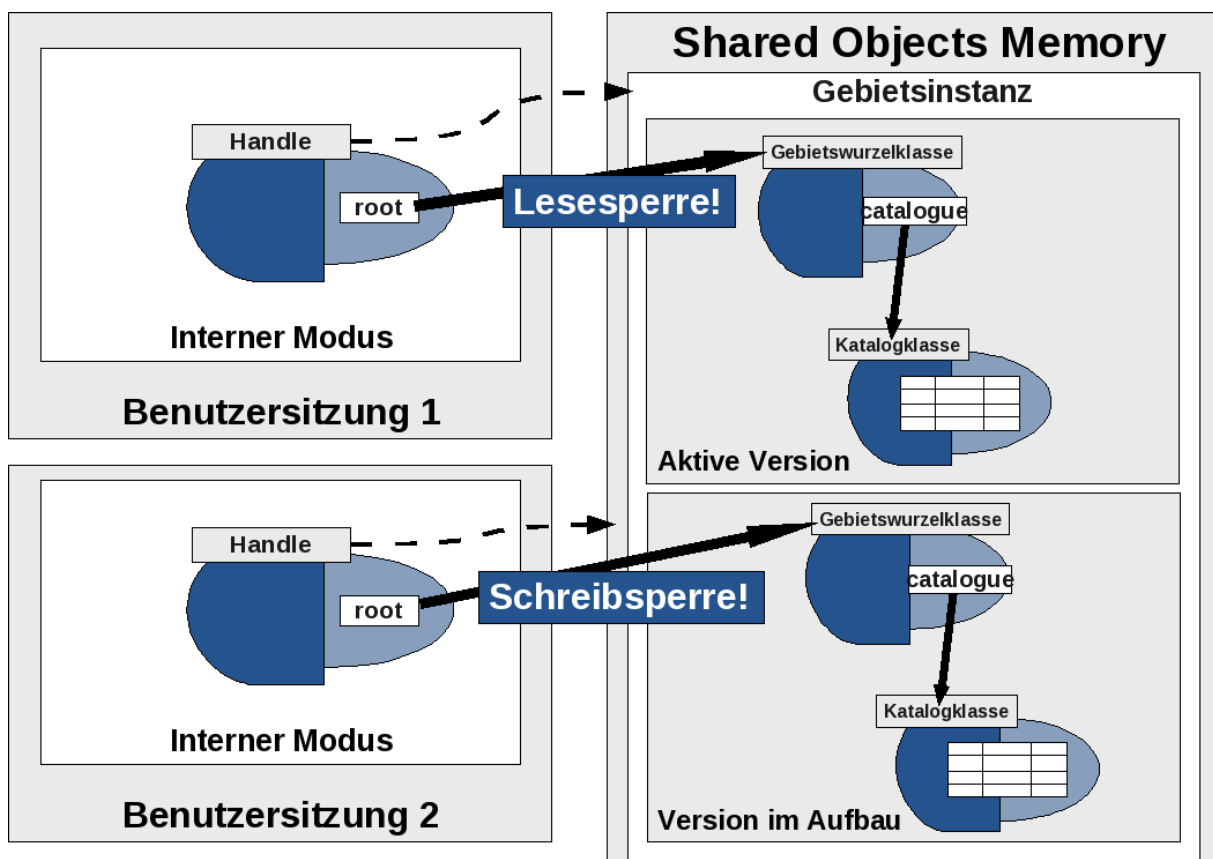


Abbildung 64: Koexistenz von aktiver Version und Version im Aufbau

Wenn die Bearbeitung durch `detach_commit` beendet wird, wird die Version im Aufbau zur neuen aktiven Version. Die bisherige aktive Version wird als veraltet gekennzeichnet. Die existierenden Sperren auf diese Version werden nicht beeinflusst und die zugehörigen Anwendungen arbeiten bis zum Lösen der Sperre weiter mit der jetzt veralteten Version.

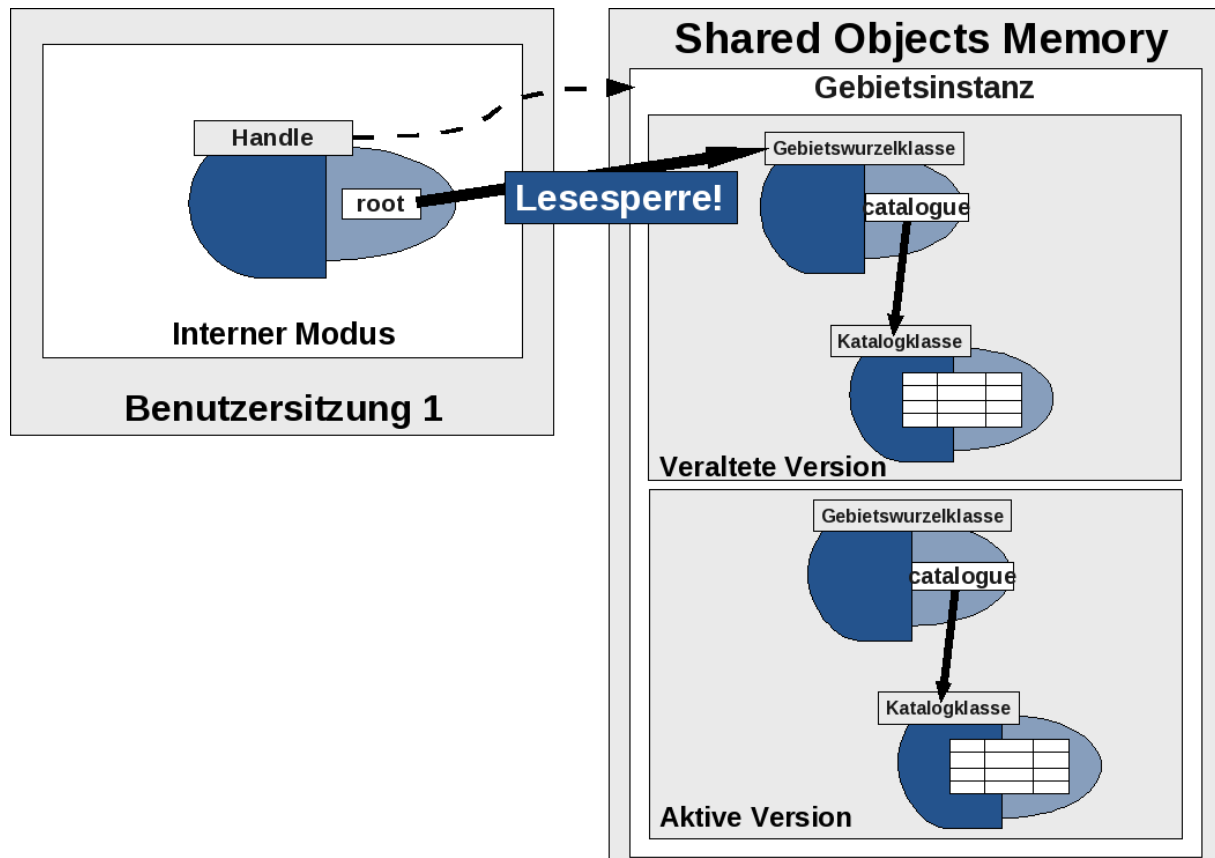


Abbildung 65: Vorhandene Lesesperre auf nun veraltete Version

Neue Lesesperren werden hingegen ausschließlich auf die aktive Version gesetzt. So kommt es, dass zu einem Zeitpunkt zwei Benutzer unterschiedliche Daten lesen, je nachdem auf welcher Version Sie arbeiten.

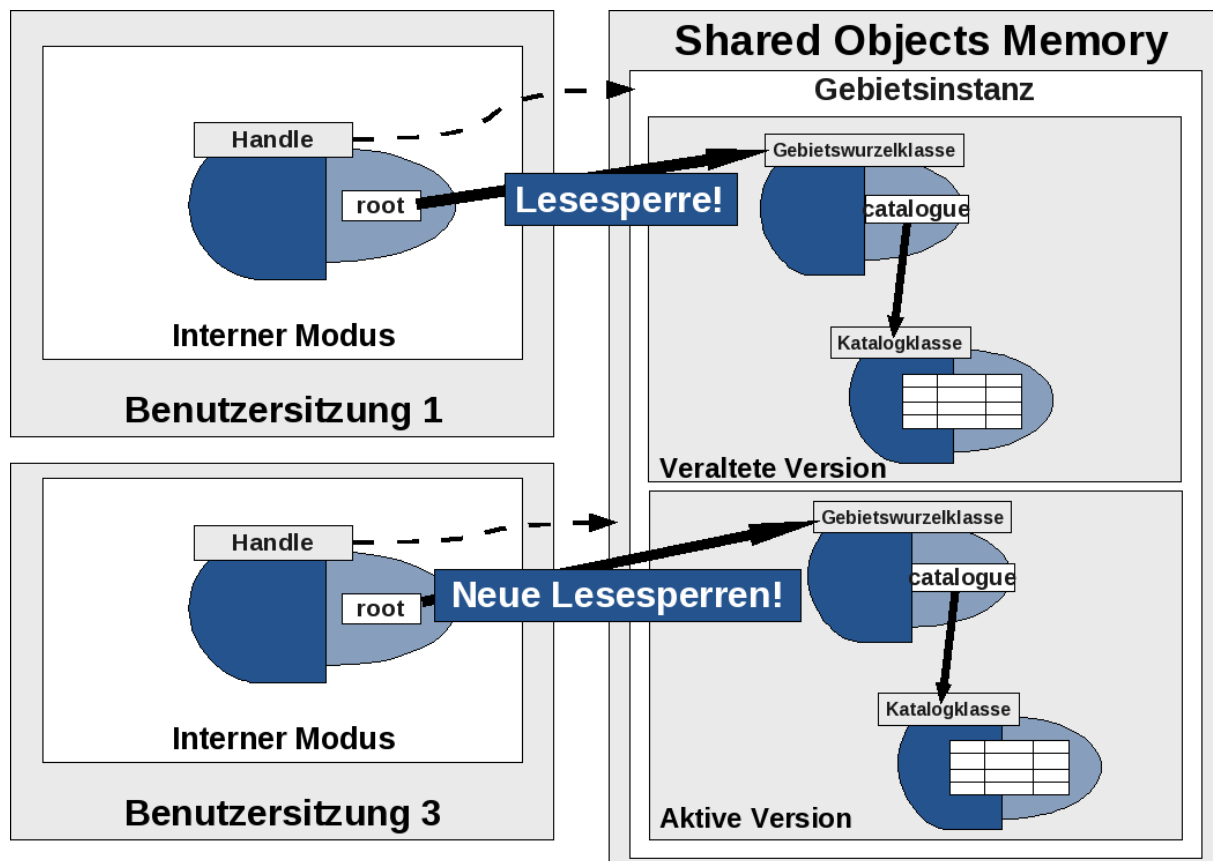


Abbildung 66: Zeitgleiches Lesen von unterschiedlichen Versionen

Nachdem alle Sperren auf die veraltete Version durch den jeweiligen Aufrufer gelöst wurden, wird die Version als verfallen markiert. Der Garbage-Collector entfernt verfallene Versionen aus dem Shared Memory. Bei Konfiguration einer maximalen Anzahl von Versionen werden diese Versionen nicht mehr zur aktuellen Versionsanzahl hinzugezählt.

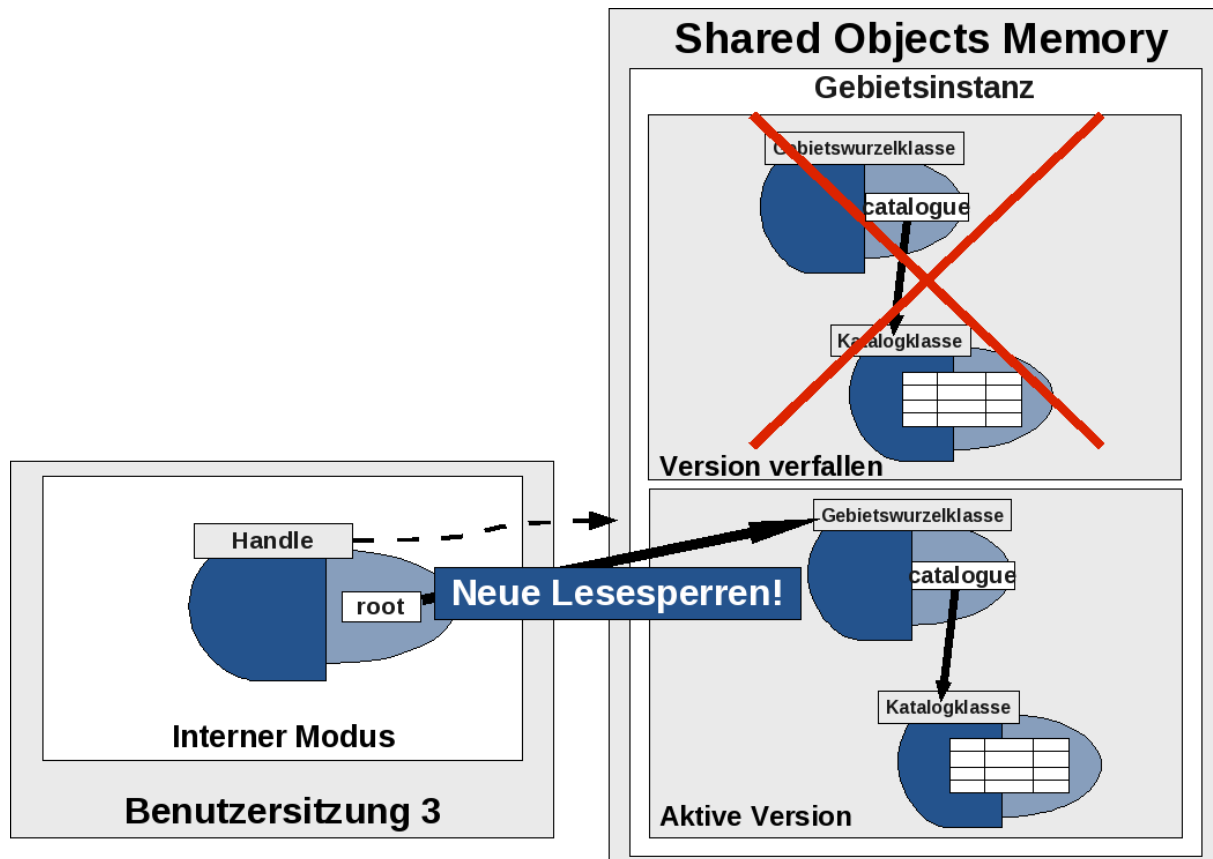


Abbildung 67: Verfallene Version

Alle neuen Sperren werden auf der jeweils gerade aktiven Version gesetzt. Zu jedem Zeitpunkt gibt es nur eine aktive Version. Wenn die Konfiguration des Gebiets es zulässt, können aber mehrere veraltete Versionen existieren, auf die es noch Lesesperren gibt und die deshalb noch nicht verfallen sind.

Als Vorteil der Versionierung lässt sich festhalten, dass Konflikte durch Sperren deutlich verringert werden. Dem steht der Nachteil gegenüber, dass unterschiedliche Verwender zum selben Zeitpunkt unterschiedliche Daten erhalten können.

5.3.5 Kontrollfragen

1. Was ist der Unterschied zwischen Gebiet und Gebietsinstanz?
2. Muss zu jedem Gebiet eine Gebietsklasse angelegt werden?
3. Was sind mögliche Einsatzgebiete für Shared Objects?

5.3.6 Antworten

1. Das Gebiet beschreibt eine Art Bauplan, der dann in Form einer Gebietsinstanz instanziiert wird.
2. Nein, die Gebietsklasse wird automatisch generiert.
3. Einsatzgebiete in denen selten geschrieben, aber häufig gelesen wird, wie etwa Kataloge.

5.4 Klassenbasierte Ausnahmen

Unter **Ausnahmen** versteht man Situationen, in denen ein normales Fortfahren im Programmablauf nicht sinnvoll ist. Bereits aus dem Kurs „Einführung in ABAP“ kennen Sie das klassische Konzept von ABAP zur Verarbeitung von Ausnahmen. Dabei wurde das Systemfeld sy-subrc mit einem Wert belegt, der anschließend ausgewertet werden konnte. Abhängig vom ermittelten Wert wurde dann auf die Ausnahme reagiert, indem etwa eine Fehlermeldung ausgegeben wurde.

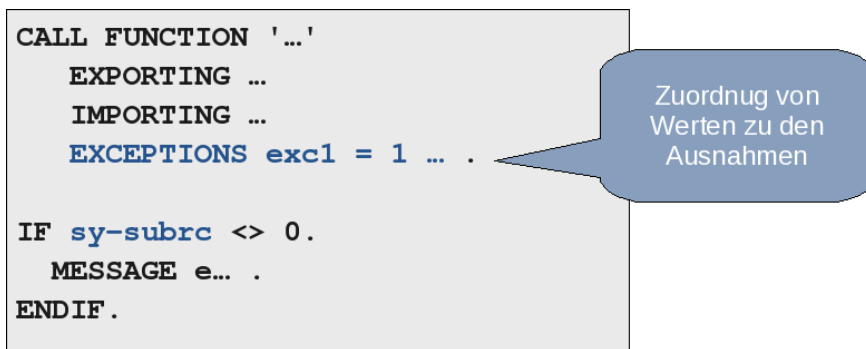


Abbildung 68: Beispiel für eine klassische Ausnahmebehandlung

Ab dem Release 6.10 steht nun ein neues, **klassenbasiertes** Konzept zur Behandlung von Ausnahmen zur Verfügung. Das neue Ausnahmekonzept erlaubt es, an beliebigen Stellen im Programmablauf Ausnahmen auszulösen. Bisher war nur innerhalb von Funktionsbausteinen und Methoden ein Auslösen mittels RAISE möglich.

Beide Konzepte – das klassische Konzept mit sy-subrc und das neue, klassenbasierte Ausnahmekonzept – existieren parallel. Bei Systemausnahmen wie Nulldivisionen sowie bei Ausnahmen von Methoden und Funktionsbausteinen ist eine Verwendung des sy-subrc oder des neuen klassenbasierten Ausnahmekonzepts möglich. Der Entwickler hat also die Auswahl, welches Ausnahmekonzept er verwenden möchte. Es darf allerdings **keine Mischung** der Konzepte stattfinden: Ein Funktionsbaustein oder eine Methode darf nicht sowohl klassische als auch klassenbasierte Ausnahmen liefern. Das Auslösen von Ausnahmen aus beliebigen Verarbeitungsblocken ist nur mit dem neuen klassenbasierten Konzept möglich.

Im klassenbasierten Ausnahmekonzept gehört jede Ausnahme zu einer Ausnahmeklasse. Beim Auslösen der Ausnahme wird ein Objekt der Ausnahmeklasse instanziiert, auf welches dann bei der Behandlung zugegriffen werden kann. In den Attributen des Ausnahmeobjekts sind Informationen zur aufgetretenen Ausnahme hinterlegt, die dann bei der Behandlung verwendet werden können.

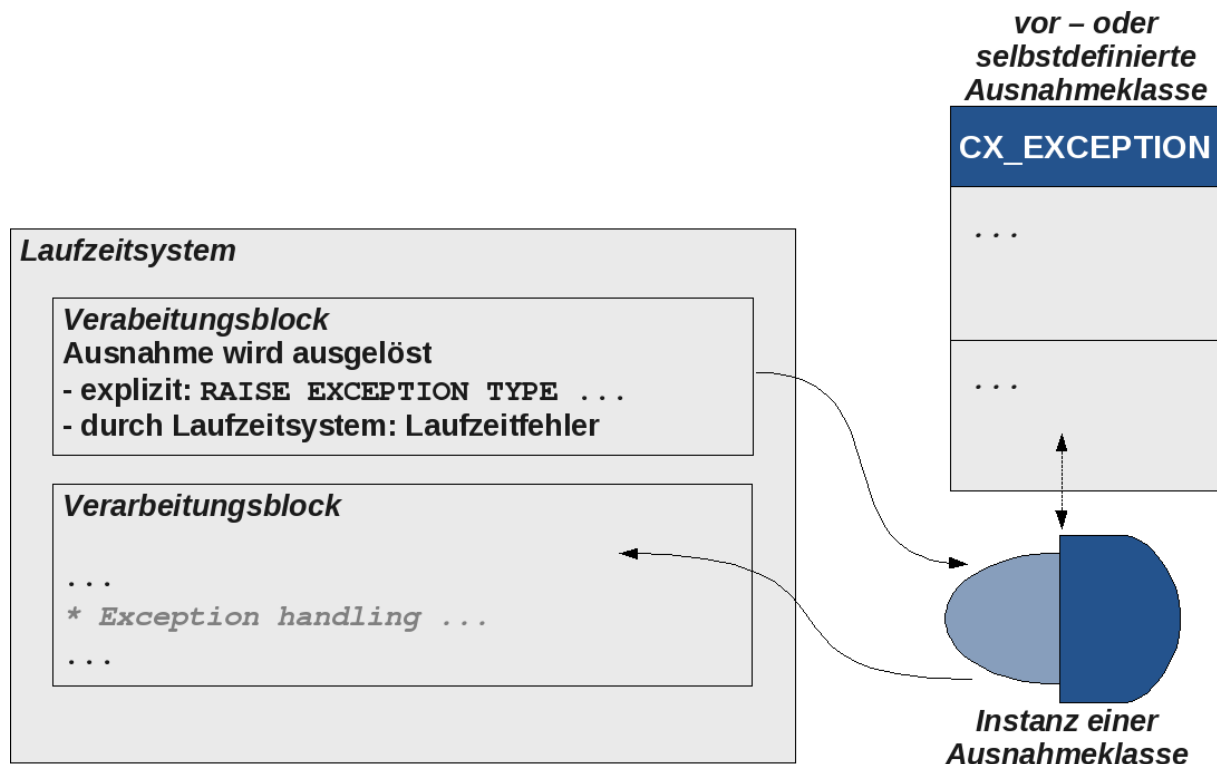


Abbildung 69: Klassenbasiertes Ausnahmekonzept: Überblick

Alle Ausnahmeklassen bilden eine Vererbungshierarchie, an deren Spitze die Klasse CX_ROOT steht. Diese Klasse hat die drei Unterklassen CX_NO_CHECK, CX_DYNAMIC_CHECK und CX_STATIC_CHECK. Alle eigenen Ausnahmeklassen müssen sich in der Vererbungshierarchie unter einer dieser drei Klassen einordnen.

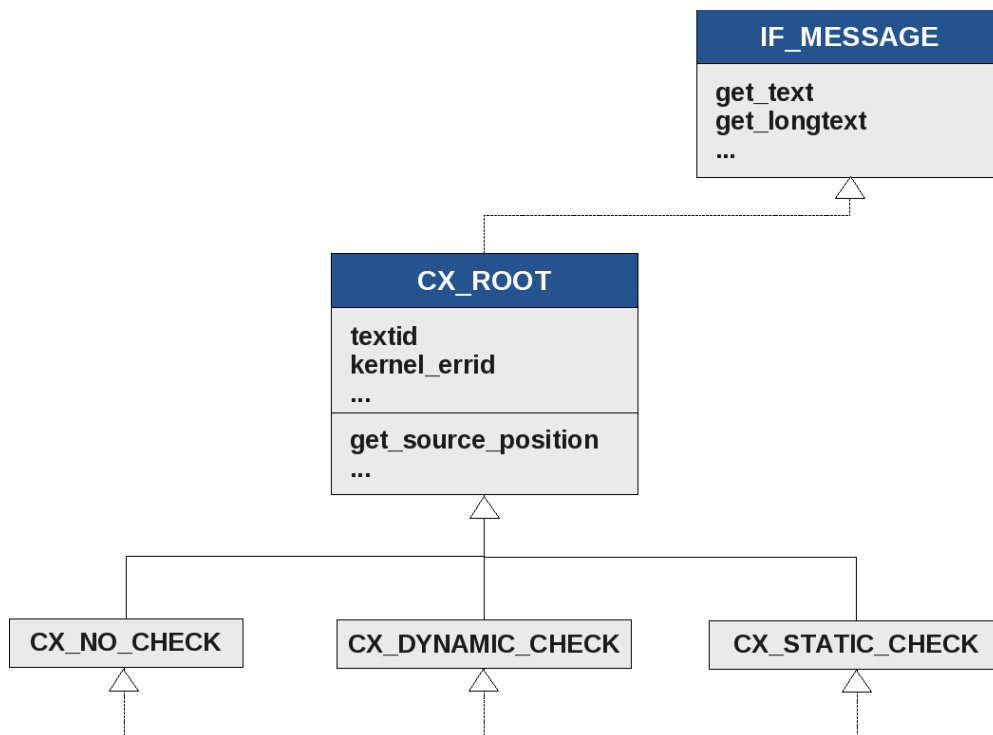


Abbildung 70: Hierarchie der Ausnahmeklassen

Die Bedeutung der einzelnen Klassen wird später noch erläutert.

Die Syntax, die im Rahmen des neuen Ausnahmekonzepts benutzt wird, haben Sie bereits im Kontext von Down-Casts kennen gelernt.

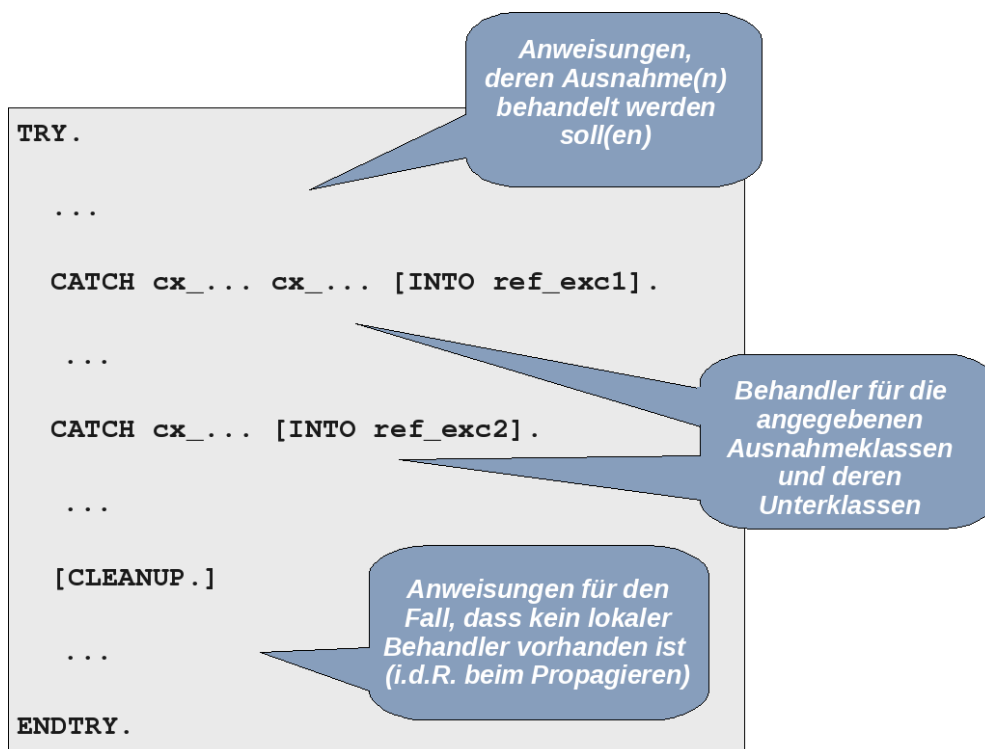


Abbildung 71: Syntax des TRY-Blocks

Der TRY-Block umschließt die Anweisungen, deren Ausnahmen behandelt werden sollen. Es folgen Behandler für bestimmte Ausnahmeklassen. Diese werden jeweils mit `CATCH` angegeben. Es können für einen Behandler mehrere Ausnahmeklassen angegeben werden. Neben diesen Klassen selbst werden auch etwaige Unterklassenausnahmen behandelt. Der `CLEANUP`-Bereich wird ausgeführt, wenn in diesem TRY-Block kein Behandler für die aufgetretene Ausnahme vorhanden ist. Das bedeutet nicht, dass die Ausnahme gar nicht behandelt wird. Es könnte etwa einen umgebenden Block geben, in dem die Behandlung erfolgt. Durch den `INTO`-Zusatz kann eine Referenzvariable angegeben werden, über die anschließend das Ausnahmeobjekt verfügbar ist. Die Variable muss so allgemein typisiert sein, dass sie Objekte aller in der `CATCH`-Anweisung angegebenen Ausnahmeklassen referenzieren kann. Der `CLEANUP`-Block kann z. B. dazu verwendet werden, blockierte Ressourcen wieder freizugeben.

5.4.1 Praxis: Behandlung vordefinierter Ausnahmen

Laufzeitfehler wie etwa eine Nulldivision können mit dem klassenbasierten Ausnahmekonzept abgefangen und behandelt werden. Hierfür existieren im System vordefinierte Ausnahmeklassen, die für den entsprechenden `CATCH`-Bereich verwendet werden können.

Legen Sie für diese Übung ein neues Programm `ZZ_####_CALC` an. Das Programm soll kein TOP-Include besitzen und wie gewohnt Ihrem Paket und Ihrem Transportauftrag zugeordnet sein.

Definieren Sie im Programm mit der `PARAMETERS`-Anweisung zwei Parameter, die Sie mit `i` typisieren, sowie eine Variable mit dem Namen **ergebnis** vom gleichen Typ. Im Programm soll der erste Parameter durch den zweiten Parameter dividiert und das Ergebnis in der Variable gespeichert werden. Diese soll anschließend mit dem `WRITE`-Befehl ausgegeben werden.

Implementieren Sie das Programm entsprechend, speichern, prüfen und aktivieren Sie es.

Rufen Sie es auf und testen Sie es mit einfachen Beispielen. Geben Sie auch als Beispiel einmal den Wert 0 für den zweiten Parameter an (aber nicht den ersten, 0/0 ist in ABAP keine Nulldivision). Daraufhin erscheint, wie nicht anders zu erwarten, ein Laufzeitfehler:

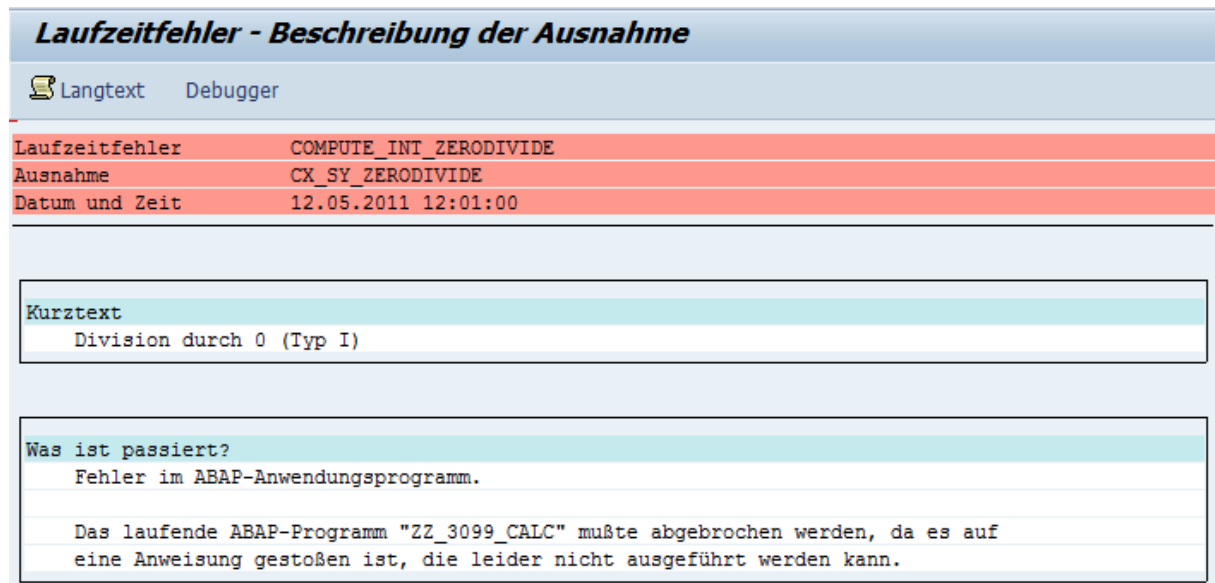


Abbildung 72: Laufzeitfehler bei Nulldivision: SAP-System-Screenshot

Sie können den Ausgaben des Systems beim Auftreten des Laufzeitfehlers die vordefinierte Ausnahmeklasse entnehmen, die für den Fehler verwendet wird. Sie lautet hier **CX_SY_ZERODIVIDE**.

Öffnen Sie wieder Ihr Programm im Object Navigator. Fassen Sie die Zeile zur Berechnung des Ergebnisses in einen `TRY`-Block ein und behandeln Sie die Ausnahme, indem Sie eine entsprechende `CATCH`-Anweisung einbauen. Geben Sie dort eine Fehlermeldung (Typ E) mit dem `MESSAGE`-Befehl aus. Führen Sie die Berechnung im Verarbeitungsblock `AT SELECTION-SCREEN` durch, damit der Nutzer seine Eingabe ggf. korrigieren kann. Die Ausgabe erfolgt hingegen im Block `START-OF-SELECTION`.

Hinweis: Diese Blöcke werden auch als „Ereignisse“ bezeichnet. Dies hat nichts mit dem Begriff des Ereignisses aus der Ereignisverarbeitung mit ABAP Objects zu tun.

Speichern, prüfen und aktivieren Sie Ihr Programm. Bei fehlerhafter Eingabe sollte nun Ihre Fehlermeldung erscheinen:

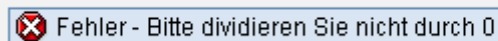


Abbildung 73: Selbstdefinierte Fehlermeldung: SAP-System-Screenshot

Wie oben beschrieben wurde, wird eine Ausnahme auch behandelt, wenn ihre Oberklasse in der CATCH-Anweisung steht. Mit dem Class Builders können Sie herausfinden, dass die Oberklasse der Ausnahme CX_SY_ARITHMETIC_ERROR heißt. Ersetzen Sie im CATCH-Befehl den Namen der Ausnahmeklasse durch den Namen dieser Oberklasse.

Da zumindest die Behandlerdefinition nun auch andere Fehler als Nulldivisionen zulässt, sollten Sie den Text der Fehlermeldung aus dem Ausnahmeobjekt der Ausnahme ermitteln. Definieren Sie dazu (vor dem TRY-Block) eine Referenzvariable vom Typ der Oberklasse und weisen Sie durch den INTO-Zusatz der CATCH-Anweisung das Ausnahmeobjekt dieser Referenzvariablen zu. Rufen Sie dann zur Ermittlung des Texts die Methode `get_text` des Ausnahmeobjekts auf und verwenden Sie den zurückgelieferten Text als Fehlermeldungstext. Speichern, prüfen und aktivieren Sie das Programm. Testen Sie es erneut mit einer unzulässigen Division. Es erscheint nun der aus dem Ausnahmeobjekt ausgelesene Text:

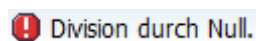


Abbildung 74: Fehlermeldung mit Text aus einem Ausnahmeobjekt: SAP-System-Screenshot

Die Methode `get_text`, die Sie zur Ermittlung des Fehlertextes verwendet haben, ist Teil des von der Klasse CX_ROOT implementierten Interfaces IF_MESSAGE. Dieses definiert auch eine Methode `get_longtext` mit einem Langtext, der jedoch nicht bei jeder Ausnahme erzeugt wird.

5.4.2 Praxis: Definition und Verwendung eigener globaler Ausnahmeklassen

Als nächstes soll die Definition eigener Ausnahmeklassen, die Auslösung von entsprechenden Ausnahmen und deren Behandlung betrachtet werden. Öffnen Sie hierfür zunächst Ihre Klasse ZCL_####_FLINFO und definieren Sie eine neue Methode `get_connection`. Diese soll zu einer Carrier-ID und Connection-Nummer die Detaildaten zur Flugverbindung zurückliefern.

Definieren Sie für die Methode die Importparameter `im_carrid` und `im_connid` und als Returning-Parameter `re_connection` eine Flugverbindung (Typisiert über `spfli`).

Implementieren Sie die Methode, indem Sie mit dem READ TABLE-Befehl auf die Verbindungsliste der befreundeten Singleton-Klasse zugreifen. Speichern Sie die Methodenimplementierung und kehren Sie zur Klasse zurück.

Speichern, prüfen und aktivieren Sie die Klasse. Testen Sie die Klasse und rufen Sie die soeben definierte Methode auf. Bei einer korrekten Parameterangabe wie etwa Fluggesellschaft LH und Flugverbindung 400 erhalten Sie die entsprechenden Detaildaten:

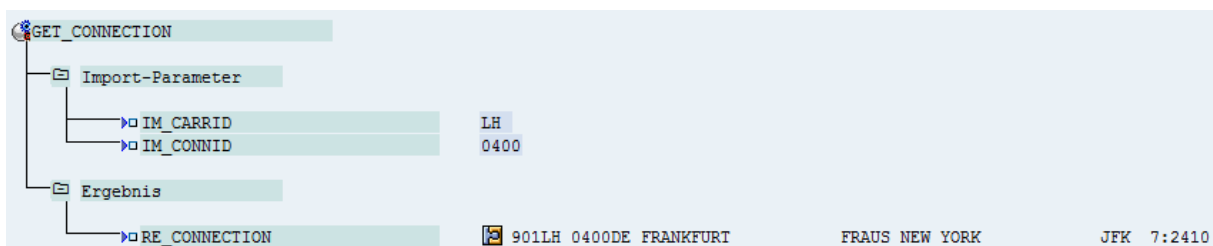


Abbildung 75: Aufruf der Methode: SAP-System-Screenshot

Geben Sie hingegen keine gültigen Parameter an, ist der Rückgabewert eine Struktur ohne gesetzte Felder. Für diesen Fall soll nun eine Ausnahmeklasse definiert, die entsprechende Ausnahme ausgelöst und behandelt werden.

Um die Ausnahmeklasse anzulegen, klicken Sie im Navigationsbaum des Object Navigators unterhalb Ihres Pakets und Klassenbibliothek mit der rechten Maustaste auf **Klassen** und wählen Sie **Anlegen**. Benennen Sie die Klasse als **ZCX_####_EXCEPTION**. Es handelt sich hier um einen vorgegebenen Namensraum: SAP-Ausnahmeklassen beginnen mit **CX_**, kundeneigene Ausnahmeklassen haben davor den Kundennamensraum, also in unserem Fall **ZCX_**.

Wählen Sie als Klassentyp **Ausnahmeklasse** und setzen Sie das Häkchen **mit Nachrichtenklasse**. So können Sie später die Fehlermeldungen, die beim Auftreten der Ausnahme ausgegeben werden sollen, in einer Nachrichtenklasse pflegen. Sichern Sie die Ausnahmeklasse auf gewohnte Weise.

Legen Sie zunächst eine Nachrichtenklasse an, indem Sie mit der rechten Maustaste auf Ihr Paket im Navigationsbaum klicken und **Anlegen -> Weitere -> Nachrichtenklasse** wählen. Nennen Sie die Klasse **ZZ_####_MSG** und sichern Sie die Klasse. Hinterlegen Sie darin als Nachricht 000 eine Meldung ohne Platzhalter, die besagt dass die Verbindung nicht existiert, und als Nachricht 001 eine Meldung mit Platzhaltern für Fluggesellschaft und Verbindungsnummer. Sichern Sie die Nachrichtenklasse.

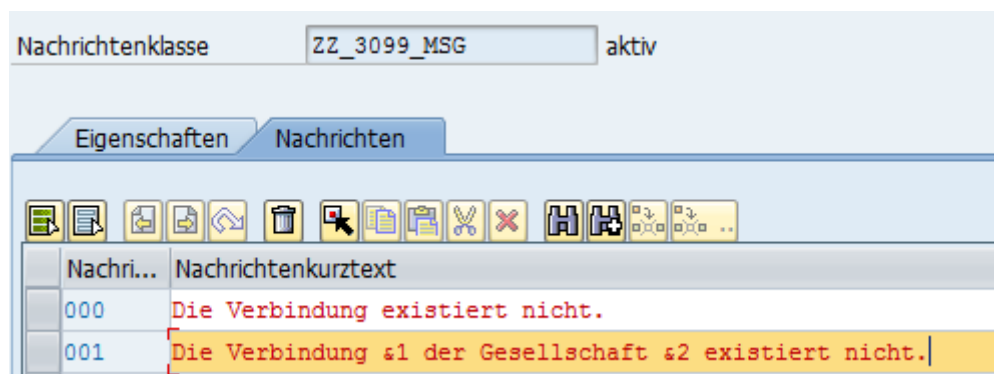


Abbildung 76: Nachrichtenklasse für die Ausnahmeklasse: SAP-System-Screenshot

Öffnen Sie nun wieder Ihre Ausnahmeklasse und wechseln Sie zum Karteireiter **Attribute**. Wie bereits zuvor erläutert, können Ausnahmeklassen Attribute besitzen. So können die Ausnahmeobjekte Informationen über die Fehlersituation aufnehmen, die dann später bei der Behandlung verwendet werden können, etwa zur Ausgabe einer genaueren Fehlermeldung. Legen Sie hier die Attribute **carrid** und **connid** an. Beide sollen öffentliche Instanzattribute sein. Die benötigten Typen lauten **S_CARR_ID** bzw. **S_CONN_ID**. Sichern Sie die Klasse. Wählen Sie nun die Registerkarte **Texte**. Dort ist bereits ein Default-Eintrag vorhanden.


Positionieren Sie den Cursor auf diesem Eintrag und klicken Sie auf  **Nachrichtentext**. Geben Sie im erscheinenden Fenster den Namen Ihrer Nachrichtenklasse und die Nachrichtennummer 000 ein. Bestätigen Sie mit Enter, um den Nachrichtentext sehen zu können.

Abbildung 77: Auswahl einer Nachricht: SAP-System-Screenshot

Bestätigen Sie mit **Ändern**. Erstellen Sie eine weitere Text-ID mit dem Namen **NO_CONNECTION**. Beziehen Sie hier den Nachrichtentext aus Nachricht 001 Ihrer Nachrichtenklasse und wählen Sie als Attribute CARRID und CONNID aus. Bestätigen Sie auch dieses Fenster, speichern, prüfen und aktivieren Sie die Ausnahmeklasse.

Als nächstes muss die Methode, in der der Fehler auftritt, angeben, dass Sie die Ausnahme erzeugt. Öffnen Sie daher die Klasse **ZCL_####_FLINFO** und dort die Registerkarte **Methoden**. Positionieren Sie den Cursor auf die Zeile mit der Methode

GET_CONNECTION und klicken Sie auf **Ausnahmen**. Stellen Sie sicher, dass der Haken **Ausnahmeklassen** gesetzt ist. Geben Sie dann Ihre Klasse **ZCX_####_EXCEPTION** ein. Würde es sich um eine lokale Klasse handeln, müssten Sie stattdessen die Methodendefinition um den Zusatz `RAISING ZCX_####_EXCEPTION` ergänzen.

Öffnen Sie die Implementierung der Methode. Prüfen Sie nach dem `READ TABLE`-Befehl, ob `sy-subrc` den Wert 4 hat. In diesem Fall wurde die Verbindung nicht gefunden und die Ausnahme muss ausgelöst werden. Dies geschieht mit dem `RAISE EXCEPTION`-Befehl, der folgende Syntax besitzt:

```
RAISE EXCEPTION {
    { TYPE cx_class
      [EXPORTING par1 = val1 par2 = val2 ... ]}
    | oref }.
```

Hierbei ist `cx_class` die Ausnahmeklasse. Hinter `EXPORTING` werden die Parameter übergeben. `oref` steht für ein bereits vorhandenes referenziertes Ausnahmeobjekt, das statt einer neuen Instanz der Ausnahmeklasse verwendet wird.

Geben Sie in Ihrem Code Ihre Ausnahmeklasse an und übergeben Sie an die Formalparameter CARRID und CONNID die Angaben aus den Parametern der Methode, die zu dem Fehler geführt haben, sowie als Parameter `textid` die vorhin definierte Text-Id `ZCX_####_EXCEPTION=>NO_CONNECTION`. Sie sehen hier, dass der Class Builder automatisch die Parameter für den Instanzkonstruktor aus den von Ihnen definierten öffentlichen Attributen generiert hat. Speichern, prüfen und aktivieren Sie die Klasse.

Erstellen Sie nun ein neues Programm **ZZ_####_EXCEPTION**, ohne TOP-Include und unter Angabe von Paket und Transportauftrag. Dieses soll die Methode `get_connection` der Klasse **ZZ_####_FLINFO** aufrufen. Ziehen Sie die Methode aus dem Navigationsbaum in Ihren Quelltext. Es entsteht wie gewohnt ein Aufrufgerüst, diesmal ist dieses jedoch mit einem TRY-Block incl. passendem CATCH-Teil versehen:

```

14▶ *TRY.
15▶
16▶ CALL METHOD ZCL_3099_FLINFO=>GET_CONNECTION
17▶ EXPORTING
18▶     IM_CARRID      =
19▶     IM_CONNID      =
20▶ RECEIVING
21▶     RE_CONNECTION =
22▶
23▶ * CATCH ZCX_3099_EXCEPTION INTO r_exception .
24▶
25▶ * ENDTRY.

```

Abbildung 78: Aufrufgerüst mit TRY-Block: SAP-System-Screenshot

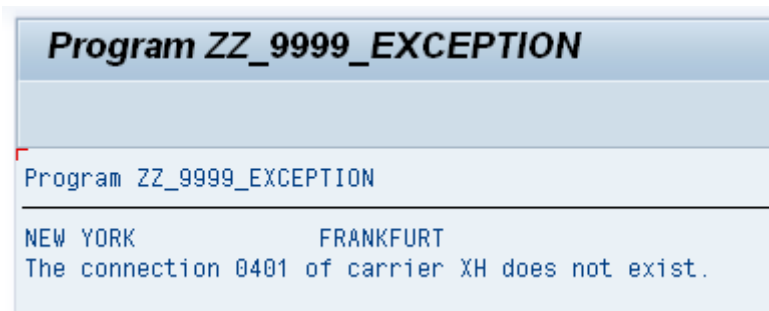
*Hinweis: Dieses Verhalten kann getrennt nach Methoden und Funktionsbausteinen unter **Hilfsmittel -> Einstellungen -> ABAP Editor -> Muster** konfiguriert werden, wie der Name sagt gilt diese Einstellung dann auch für die Muster-Schaltfläche.*

Entfernen Sie die Kommentarsterne und Übergeben Sie zunächst korrekte Werte (z. B. `'LH'` und `'0400'`) für die Parameter. Definieren Sie eine Variable vom Typ `SPFLI` zur Aufnahme des Rückgabewerts. Geben Sie zwischen Methodenaufruf und CATCH-Anweisung die Komponenten `cityfrom` und `cityto` der Flugverbindung aus.

Ergänzen Sie den CATCH-Teil um den INTO-Teil, indem Sie vor dem TRY-Block eine auf Ihre Ausnahmeklasse typisierte Referenzvariable deklarieren und diese hier entsprechend angeben. Lesen Sie im CATCH-Teil den Ausnahmetext aus dem Ausnahmeobjekt und geben Sie diesen mit der `WRITE`-Anweisung aus.

Kopieren Sie nun den gesamten TRY-Block. Ändern Sie in der Kopie die Parameterwerte für den Methodenaufruf so, dass eine ungültige Kombination entsteht.

Speichern, prüfen und aktivieren Sie Ihr Programm. Testen Sie es anschließend. Es sollte eine Ausgabe ähnlich der folgenden erscheinen:



```

Program ZZ_9999_EXCEPTION
NEW YORK          FRANKFURT
The connection 0401 of carrier XH does not exist.

```

Abbildung 79: Ausgabe des Programms: SAP-System-Screenshot

Sie haben nun erfolgreich eine eigene Ausnahmeklasse definiert, die Ausnahme ausgelöst, abgefangen und die im Ausnahmeobjekt mitgegebenen Informationen verwendet.

5.4.3 Das RETRY-Statement

Eine Neuerung des Releases SAP NetWeaver 7.0 EhP 2 ist das RETRY-Statement. Hiermit ist es möglich, in einem CATCH-Block die Ursache für eine Ausnahme zu beseitigen, und anschließend wieder zum Beginn des TRY-Blocks zu springen.

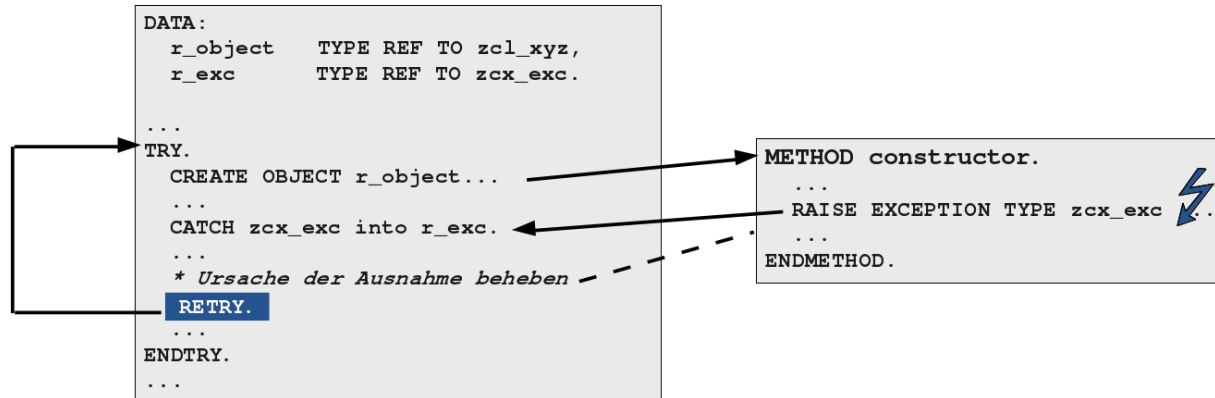


Abbildung 80: Das RETRY-Statement

Das RETRY-Statement sollte mit Vorsicht eingesetzt werden: Wird die Ursache vor dem RETRY **nicht** behoben, entsteht leicht eine Endlosschleife.

5.4.4 Das RESUME-Statement

Eine weitere Neuerung des Releases SAP NetWeaver 7.0 EhP 2 ist das RESUME-Statement. Hiermit kann ein Programm direkt nach dem Statement fortgesetzt werden, das die Ausnahme ausgelöst hat. Hierfür gelten folgende Voraussetzungen:

1. Die Ausnahme muss mit CATCH BEFORE UNWIND gefangen werden. Dies stellt sicher, dass der Kontext der Ausnahme für ein etwaiges RESUME beibehalten wird. Falls nach einem solchen CATCH kein RESUME angewendet wird, löscht das System den entsprechenden Kontext am Ende des CATCH-Blocks.
2. Die Ausnahme muss mit RAISE RESUMABLE ausgelöst werden, einer Variante des RAISE EXCEPTION-Statements. Dies ist für die Vorbereitung des Resume erforderlich.
3. Bei einer Propagierung der Ausnahme, muss diese auf allen Ebenen als fortsetzbar spezifiziert werden. Hierzu dient der RAISING RESUMABLE (...) -Zusatz mit dem Namen der Ausnahme in den Klammern. So sind für die Fortsetzung die Methoden auf allen Ebenen vorbereitet.

Entsprechende Checkboxes sind im Class Builder und Function Builder verfügbar.

Der Handler einer Exception kann zur Laufzeit feststellen, ob eine Ausnahme als fortsetzbar ausgelöst wurde oder nicht. Alle Ausnahmeobjekte besitzen das öffentliche Instanzattribut IS_RESUMABLE mit den Werten , ' (Leerzeichen) oder ,X'. Der Versuch, eine nicht-fortsetzbare Ausnahme fortzusetzen, führt zum Laufzeitfehler CX_SY_ILLEGAL_HANDLER.

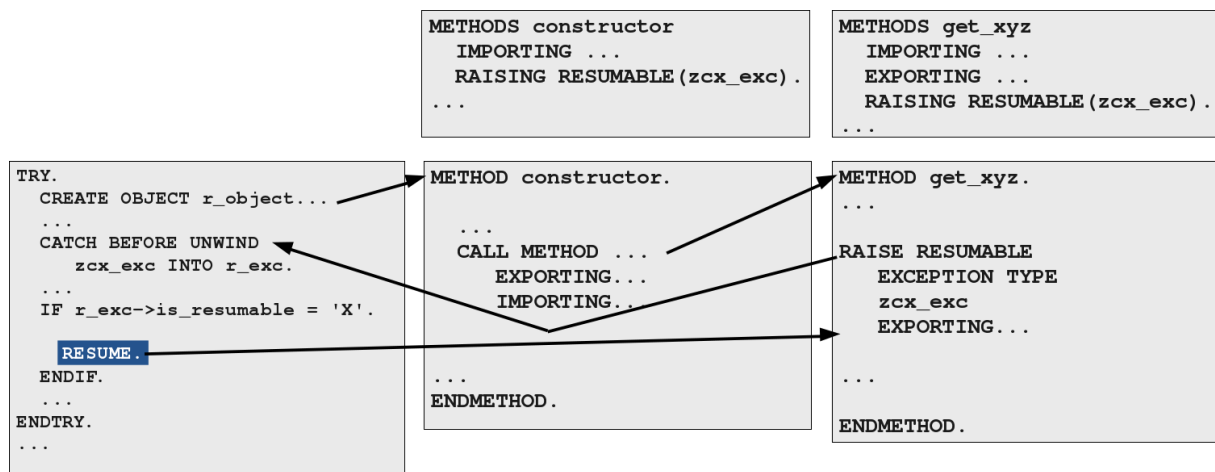


Abbildung 81: Das RESUME-Statement

Die Abbildung zeigt den Kontrollfluss (Pfeile) bei Verwendung des `RESUME`-Statements. Über die Zwischenebene hinweg wird die Ausnahme als fortsetzbar weitergereicht. Die Methode `get_xyz` kann daher nach der Ausnahme fortgesetzt werden. Ohne `BEFORE UNWIND` wäre das Objekt in dem die Ausnahme aufgetreten ist nicht mehr verfügbar.

5.4.5 Propagierung und Hierarchie von Ausnahmen

Im klassenbasierten Ausnahmekonzept ist es nicht nötig, Ausnahmen dort zu behandeln, wo Sie auftreten. Stattdessen können Sie von einer Modularisierungseinheit an deren Aufrufer weitergegeben (propagiert) und erst dort behandelt werden. Dazu muss die jeweilige Modularisierungseinheit in der Regel durch den `RAISING`-Zusatz (z. B. bei Methoden lokaler Klassen oder Unterprogrammen), bei Methoden globaler Klassen im Class Builder, angeben welche Ausnahmen von Ihr geliefert werden.

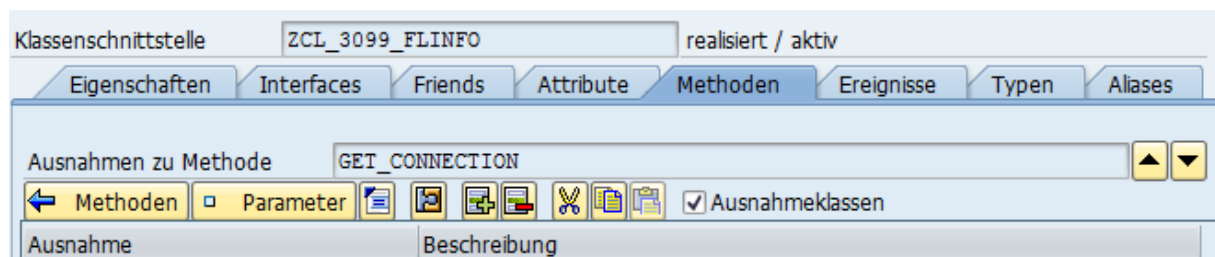


Abbildung 82: Angabe von Ausnahmen im Class Builder: SAP-System-Screenshot

Für klassenbasierte Ausnahmen muss hier das Häkchen **Ausnahmeklassen** gesetzt werden (siehe Abbildung oben). Bei Funktionsbausteinen wird die Einstellung auf der Registerkarte **Ausnahmen** vorgenommen. Hier wird deutlich, dass zwischen den beiden Ausnahmekonzepten gewählt werden muss und nicht beide gleichzeitig benutzt werden können.

Die Propagierung kann über mehrere Stufen erfolgen, spätestens auf der obersten Ebene, d. h. in Ereignisblöcken oder Dialogmodulen, müssen die Ausnahmen jedoch behandelt werden, da es ansonsten zu einem Laufzeitfehler kommt.

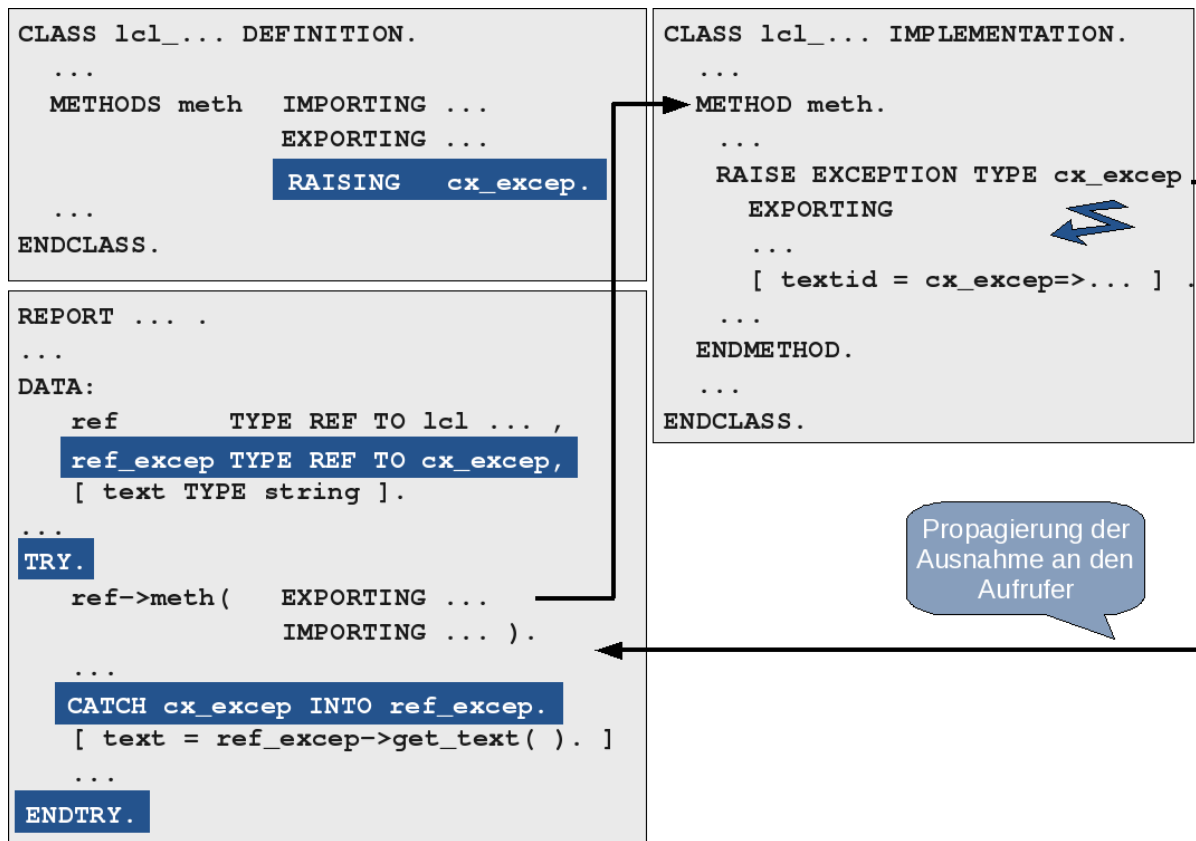


Abbildung 83: Propagierung einer Ausnahme

Die Abbildung zeigt einen Methodenaufruf, in dem mit `RAISE EXCEPTION` eine Ausnahme ausgelöst wird. In der Methode `meth` befindet sich keine Behandlung der Ausnahme. Stattdessen wurde bei der Definition der Methode die Ausnahme angegeben, und in der aufrufenden Modularisierungseinheit wird die Ausnahme behandelt. Die Angabe der `textid` im `RAISE EXCEPTION`-Befehl ist optional. In der Übung haben Sie diese Angabe vorgenommen, um die Text-Id explizit angeben zu können. Hätten Sie dies nicht getan, wäre als Default der Standardtext, der in der Registerkarte **Texte** der Ausnahmeklasse bereits vorgegeben war, verwendet worden.

In diesem Beispiel findet die Propagierung nur über eine Stufe statt. Es wäre aber auch möglich, über mehrere Stufen hinweg zu propagieren. Dabei können Ausnahmen auch miteinander **verkettet** werden: Das Ausnahmeobjekt der zunächst aufgetretenen Ausnahme kann in einem ersten `CATCH`-Block entgegengenommen werden; dort wird dann eine weitere Ausnahme ausgelöst, in deren Ausnahmeobjekt das ursprüngliche Ausnahmeobjekt verpackt wird. Hierfür besitzt jede Ausnahmeklasse das öffentliche Instanzattribut **previous**, das in der obersten Ausnahmeklasse **CX_ROOT** definiert ist.

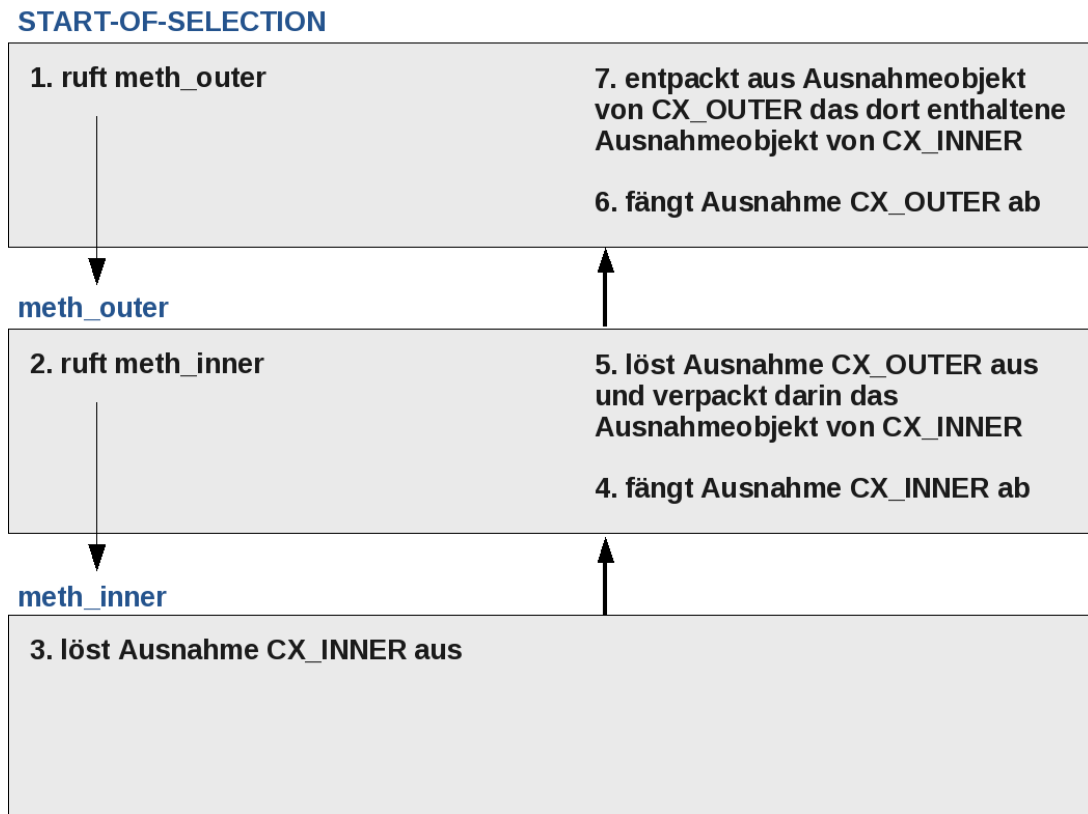


Abbildung 84: Verkettung von Ausnahmen

Die Abbildung skizziert eine solche verkettete Ausnahmebehandlung. Das Ausnahmeobjekt der inneren Ausnahme wird in der umgebenden Methode `meth_outer` in einem `CATCH`-Bereich abgefangen. Dort wird dann mit `RAISE EXCEPTION` eine neue Ausnahme ausgelöst und dabei deren Attribut **previous** das Ausnahmeobjekt der inneren Ausnahme zugewiesen. Man nennt dies auch **Abbilden** der Ausnahme. Im Code könnte dies etwa so aussehen:

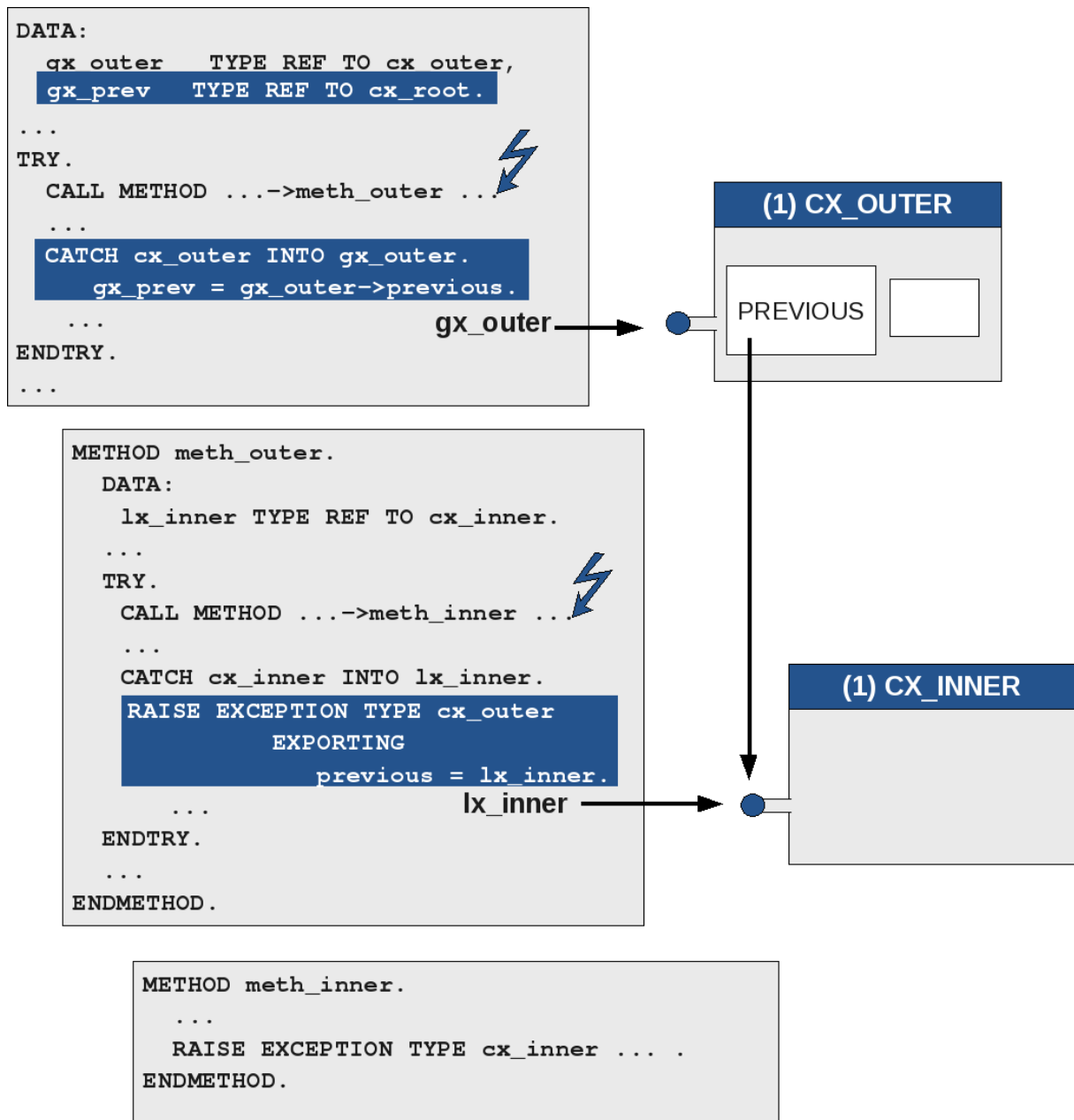


Abbildung 85: Codeausschnitte zu abgebildeten Ausnahmen

Die Referenzvariable `gx_prev` im obigen Beispiel ist mit der allgemeinsten Ausnahmeklasse `cx_root` typisiert und kann so Ausnahmen beliebiger Ausnahmeklassen referenzieren. Alle Ausnahmeklassen werden von der Klasse **CX_ROOT** abgeleitet. Das gilt auch für die Standardausnahmen des Laufzeitsystems, wie die folgende Abbildung beispielhaft zeigt.

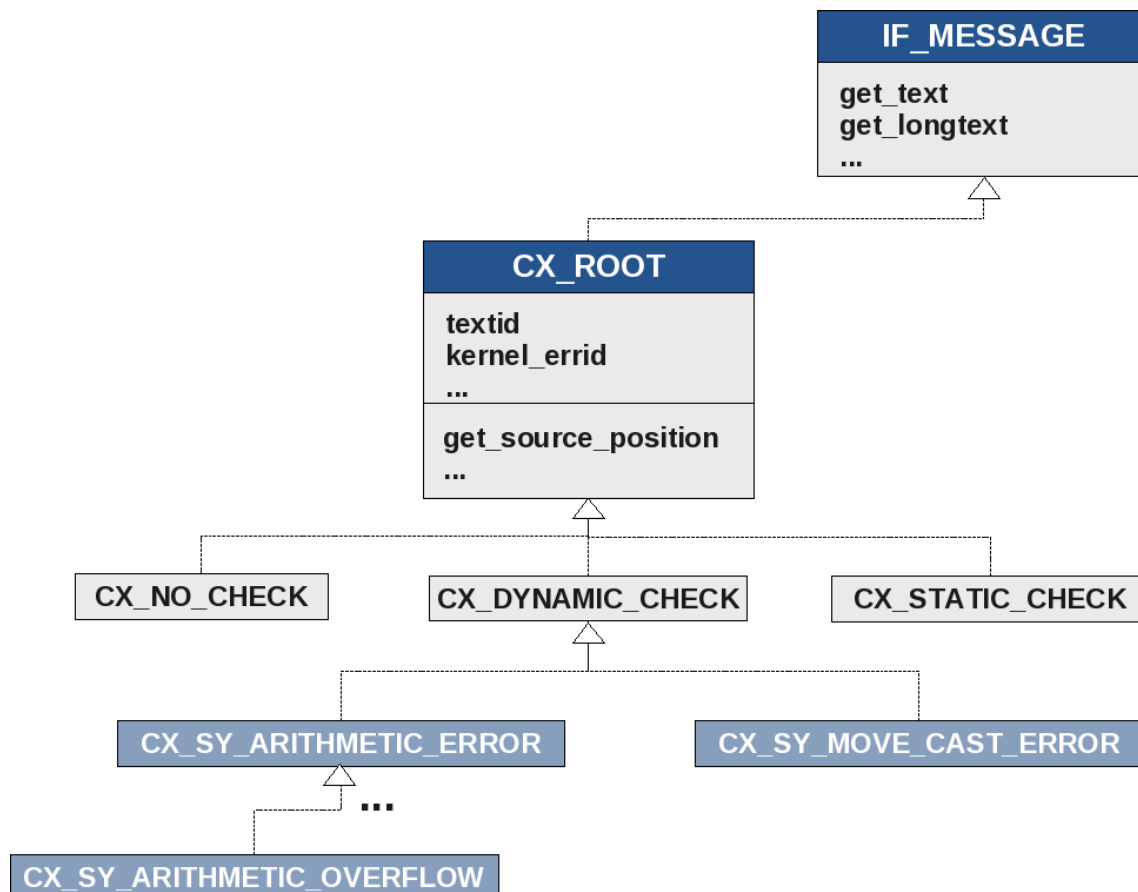


Abbildung 86: Standard-Ausnahmen in der Hierarchie der Ausnahmeklassen

Die drei Unterklassen **CX_NO_CHECK**, **CX_DYNAMIC_CHECK** und **CX_STATIC_CHECK** der Klasse **CX_ROOT** bestimmen, wie die zugehörigen Ausnahmen propagiert werden dürfen.

Ausnahmen, die zu Unterklassen von **CX_STATIC_CHECK** gehören, müssen in Prozeduren (Methoden, Funktionsbausteinen und Unterprogrammen) behandelt oder explizit propagiert werden, also in der Schnittstelle der Prozedur angegeben werden. Wird dies ausgelassen, wird bei der Syntaxprüfung eine Warnung gemeldet. Die von Ihnen definierte globale Ausnahmeklasse **ZCL_####_EXCEPTION** ist Unterklasse von **CX_STATIC_CHECK**. Würde aus der Methodendefinition von **get_connection** in der Klasse **ZCL_####_FLINFO** die Angabe der Ausnahme fehlen, würde dies also bei der Prüfung zu einer Fehlermeldung führen:

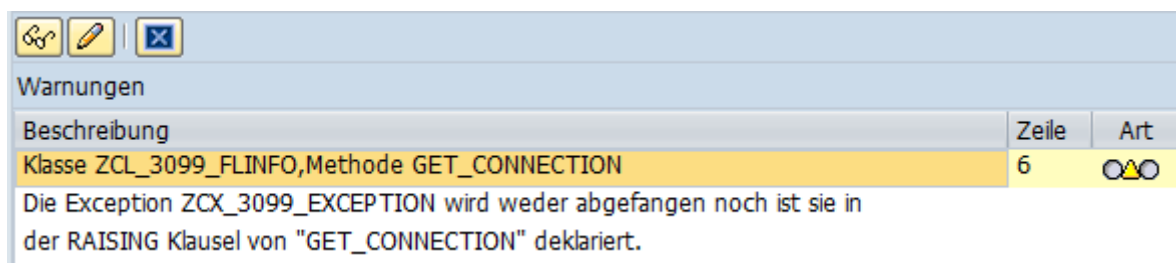


Abbildung 87: Warnung bei statischer Prüfung: SAP-System-Screenshot

Ausnahmen aus Unterklassen von **CX_DYNAMIC_CHECK** müssen ebenfalls behandelt oder explizit propagiert werden, jedoch findet hier keine statische Prüfung statt, es wird also keine Warnung ausgegeben wenn dies nicht eingehalten wurde. Stattdessen entsteht beim Auftreten ein Laufzeitfehler.

Typische Beispiele für derartige Ausnahmen sind die vordefinierten Systemausnahmen wie etwa die Klasse **CX_SY_ARITHMETIC_OVERFLOW** für einen arithmetischen Überlauf oder **CX_SY_ZERODIVIDE** für eine Nulldivision: Die entsprechenden Anweisungen führen ohne Behandler nicht zu einer Warnung beim Prüfen des Quellcodes, beim Ausführen kommt es dann aber zum Laufzeitfehler (siehe dazu Abschnitt 5.4.1).

Die dritte Möglichkeit ist die Verwendung der Oberklasse **CX_NO_CHECK**. Bei dieser Klasse **darf** die Ausnahme **nicht explizit propagiert** werden. Eine Syntaxwarnung bei fehlender Propagierung kann es deshalb hier nicht geben. Die Ausnahmen können behandelt werden, werden ansonsten aber vom System **automatisch propagiert**. Diese Ausnahmen dürfen also nicht mit dem **RAISING**-Zusatz in einer Signatur angegeben werden. Werden Sie nicht behandelt, tritt der Laufzeitfehler infolge der automatischen Propagierung auf der obersten Aufrufebene auf. Die Oberklasse **CX_NO_CHECK** wird ebenfalls von einigen Systemausnahmen verwendet.

Wenn eine Ausnahme von der Laufzeitumgebung ausgelöst wird, enthält das in der Klasse **CX_ROOT** definierte Attribut **KERNEL_ERRID** den Namen des zugehörigen Laufzeitfehlers, der bei einer nicht-Behandlung auftritt.

5.4.6 Kontrollfragen

1. Welche der folgenden Aussagen sind wahr?
 - a. Jede Ausnahme führt zu einem Absturz des Programms.
 - b. Für jede Ausnahme, die in einem TRY-Block behandelt werden soll, wird jeweils ein CATCH-Teil benötigt.
 - c. Alle selbstdefinierten Ausnahmeklassen erben von der Klasse **CX_DYNAMIC_CHECK**.
 - d. Ausnahmen können über mehrere Aufrufebenen propagiert werden.
 - e. In objektorientierten Programmen kann das klassische Ausnahmekonzept nicht mehr verwendet werden.
 - f. Ein explizites Auslösen von Ausnahmen aus beliebigen Verarbeitungsblöcken ist erst seit dem neuen, klassenbasierten Ausnahmekonzept möglich.
 - g. TRY-Blöcke sind schachtelbar.
 - h. Funktionsbausteine können keine klassenbasierten Ausnahmen besitzen.
2. Wie gelangt man an eine Referenz auf das Ausnahmeobjekt?

5.4.7 Antworten

1. Aussagen:
 - a. Falsch
 - b. Falsch, es können mehrere Klassen angegeben werden, und es werden auch Ausnahmen der jeweiligen Unterklasse behandelt
 - c. Falsch
 - d. Wahr
 - e. Falsch
 - f. Wahr
 - g. Wahr
 - h. Falsch
2. Durch den INTO-Zusatz von CATCH.

5.5 Kapitelabschluss

Sie befinden sich am Ende dieses Abschnitts. Bevor sie die im folgenden Absatz beschriebene E-Mail verfassen, beachten Sie bitte die folgenden Hinweise:

1. Prüfen Sie, ob sie wirklich **alle** Aufgaben seit dem vorangegangenen Abschluss bearbeitet haben. Diese sind mit „Praxis:“ in der Überschrift gekennzeichnet (5.1.1, 5.1.2, 5.1.3, 5.1.4, 5.1.5, 5.1.6, 5.1.7, 5.1.8, 5.2.2, 5.2.4, 5.3.1, 5.3.2, 5.3.3, 5.4.1, 5.4.2).
2. Prüfen Sie bitte noch einmal genau ob alle ihre Repository-Objekte **korrekt funktionieren**
3. Stellen Sie sicher, dass alle Repository-Objekte **aktiviert** sind. Um Objekte zu finden, die noch nicht aktiviert sind, wählen Sie aus dem Drop-Down-Menü oberhalb des Navigationsbaums im Object Navigator **Inaktive Objekte** aus. Geben Sie anschließend im darunter befindlichen Feld ihren Benutzernamen **USER#-####** ein und bestätigen Sie. Anschließend werden im Navigationsbaum die inaktiven Objekte dargestellt, die noch aktiviert werden müssen. Aktivieren Sie diese nun. Beachten Sie, dass sie die Zweige des Baums ggf. noch aufklappen müssen. Um zu ihrem Paket zurückzukehren, wählen Sie im Drop-Down-Menü wieder **Paket** aus und bestätigen Sie ihren Paketnamen.
4. Stellen Sie weiterhin sicher, dass die **Namen** ihrer Entwicklungsobjekte genau den Vorgaben im Skript entsprechen. Sollten Sie sich vertippt haben, können Sie Programme umbenennen, indem Sie diese mit der rechten Maustaste im Navigationsbaum des Object Navigators anklicken und **Umbenennen...** auswählen.

Wenn Sie den Kurs bis zu dieser Stelle bearbeitet haben, senden Sie bitte eine formlose E-Mail an die vom Kursbetreuer für diesen Kurs genannte Adresse mit dem Betreff „ABAP2: Abschluss Kapitel 5 User #####“ (die Anführungszeichen gehören nicht mit zum Betreff). Sie erhalten dann in Kürze Feedback (je nach Ergebnis **entweder** über den Fortschrittsbericht, wenn alles in Ordnung ist, **oder** per E-Mail, wenn noch Korrekturen nötig sind) und können Mängel ggf. noch nachbessern. Bitte achten Sie darauf, den Betreff genau wie angegeben zu formulieren, um eine effiziente Verarbeitung der Mail zu ermöglichen.

Sollten Sie Fragen haben, formulieren Sie diese bitte in einer **separaten E-Mail** mit aussagekräftigem Betreff, da die Kapitelabschlussmails meist nur über den Betreff verarbeitet werden!