Ungarisch für Anfänger

Man kommt nicht drumrum, wenn man Coding von anderen Entwicklern auditiert, ändert oder einfach nur verstehen muss: lt_ verfolgt dich. Überall. Im (prozeduralen) Coding der SAP. In ABAP-Coding von Kunden und sogar in deren festgeschriebenen Programmierrichtlinien. In SAP-PRESS-Büchern vom Rheinwerk-Verlag, der auch die offiziellen Programmierrichtlinien der SAP verlegt (in denen davon abgeraten wird). Ich halte es für eine schlechte Idee und stehe mit dieser Meinung nicht alleine. Warum, zeige ich in diesem Artikel auf.

Viele nennen es „Ungarische Notation“ und rechtfertigen sich damit, dass immerhin Microsoft das „erfunden“ habe. Damit fangen die Missverständnisse schon an.

Für ABAP-Unkundige nehmen wir jetzt einfach mal den beliebten Tabellennamen lt_ekko. Hierbei steht das l für den Sichtbarkeitsbereich, das t weist darauf hin, dass es sich um eine Tabelle handelt und ekko beschreibt die Struktur der Tabelle. Das ist aus einer Vielzahl von Gründen ein wirklich schlecht gewählter Name:

1. Der Sichtbarkeitsbereich

Was global ist und was lokal, hängt vom Kontext ab. Eine globale Klasse im SAP ist eine Klasse, die persistent im System hinterlegt und aus dem ganzen System heraus aufrufbar ist. Im Gegensatz dazu ist eine lokale Klasse nur im Rahmen eines bestimmten Programmes aufrufbar; eine lokale Klasse ist also *programmglobal*.

Bei Variablen soll das nun anders sein: Eine globale Variable ist nicht etwa eine im ganzen System global vorhanden (dann wären nur die Daten auf der Datenbank wirklich global), sondern lediglich (wie die lokale (!) Klasse) programmglobal. Oder programmlokal? Warum eine Variable global und eine im gleichen Kontext definierte Klasse lokal sein soll, entzieht sich meinen kognitiven Fähigkeiten. Warum heißen Variablen in Methoden dann nicht g….? Sie sind immerhin methodenglobal! Ja, in den Augen tut das Wort „methodenglobal“ weh, aber genau das will dieses Beispiel aufzeigen.

Hinzu kommt: Wenn ich eine globale Variable (etwa: „gv_test“) definiere und eine lokale (!) Variable gleichen Namens verwende, erhalte ich eine Variable im lokalen Kontext, die die globale Variable verschattet. Das ist in bestimmten Kontexten ein gewollter Effekt, der aber verbietet, den Sichtbarkeitsbereich in den Namen einer Variable zu schreiben.

Abgesehen davon verbietet sich das Verwenden von programmglobalen Variablen in einer modernen und gekapselten Architektur ohnehin von selbst (abgesehen von den Stellen, wo man darauf aus technischen Gründen schwerlich verzichten kann, z. B. bei Dynprofeldern, und selbst dort ist ein Dynprofeld auch ’nur‘ ein Attribut des Dynpros, was man in einer entsprechenden Klasse hinterlegen kann). Demzufolge müssten alle Variablen in Methoden mit einem l beginnen. Wenn das aber eh für alle gilt, kann man es auch weglassen.

2. Tabelle? Struktur? Feld? Machen wir es komplexer!

Fortgeschrittene Entwickler verwenden sehr häufig dynamische oder generische Mittel, indem sie erst zur Laufzeit ermitteln, welchen Datentyp eine Variable hat. Spätestens dann bekommt man Probleme mit der Benennung. Eine Variable, die mal eine Tabelle, mal ein Feld und mal eine Struktur sein kann, kann ich nicht nach obigen Mechanismus benennen.

Hinzu kommt, dass es „tiefe“ Strukturen und Tabellen gibt. Diese Variablen haben Komponenten, die ihrerseits wieder Komponenten haben. Als Beispiel nehmen wir eine Tabelle lt_ekko für Bestellköpfe, die eine Komponente (ein Feld) besitzen könnte, in der in Tabellenform die Bestellpositionen stehen. Nach obigem Mechanismus müsste ich dieses Feld lt_ekko-lt_ekpo benennen. Macht natürlich kein Mensch, denn es ist ja ein Feld. Aber auch eine Tabelle. Je nach Kontext, in dem man diese Komponente sieht.

A propos Feldsymbole: Das Schlimmste, was mir bezüglich Namensregeln je untergekommen ist, war die Regel:

Namen von Feldsymbolen beginnen mit „fs_“. Für Nicht-SAPler: Feldsymbole haben eine eigene Syntax, sie werden in spitze Klammern geschrieben, weshalb der Sinn eines feldsymbol-spezifischen Präfix zweifelhaft ist. Auch einen Sichtbarkeitsbereich braucht man dort nicht hineinzuschreiben, weil kein SAP-Entwickler mit Verstand globale Feldsymbole verwendet. Und für Feldsymbole gilt, wie beschrieben, ganz besonderes, dass ihre Struktur und ihre Tiefe oft erst aus dem Kontext bzw. zur Laufzeit hervorgehen.

In dieses Horn bläst auch Bjarne Stroustrup, den Entwickler von C++ in der C++ Style and Technique FAQ:

> “No I don’t recommend ‚Hungarian‘. I regard ‚Hungarian‘ (embedding an abbreviated version of a type in a variable name) a technique that can be useful in untyped languages, but is completely unsuitable for a language that supports generic programming and object-oriented programming“

3. Deklaration im Variablennamen

Technische Details in Variablennamen zu codieren, hat eine Menge Nachteile, aber keinen Vorteil. In der Entwicklungsumgebung im SAP, der Transaktion SE80, genügt ein Doppelklick, um zur Deklaration zu springen. Sie ist also vergleichsweise leicht zu ermitteln. Was viel schwerer zu ermitteln ist: Was steht in der Variable denn drin? Wenn ich aus der Tabelle EKKO (Tabelle der Einkaufsbelege) selektiere, dann sicherlich nicht die ganze Tabelle mit Kontrakten, Rahmenverträgen, Bestellungen, etc., sondern eine Untermenge, zum Beispiel Bestellungen. Oder eine Untermenge der Bestellungen, zum Beispiel Cross-Company-Bestellungen. Das ist vergleichsweise schwierig zu ermitteln, weil ich erst die Selektion finden und die Nachbearbeitung der Daten nachvollziehen muss, um festzustellen, dass in der Tabelle lt_ekko dann doch nur die Cross-Company-Bestellungen drin sind. Und gerade für Cross-Company-Bestellungen gilt: Man muss nicht nur das Entwicklungsobjekt, sondern auch das Unternehmen kennen, um CC-Bestellungen als solche zu identifizieren. Auch ein Kommentar hilft hier nicht weiter, weil der Kommentar nur an einer Stelle steht, die auch erstmal gefunden werden will. Darum ist der Name ein viel passenderer Ort, denn der steht an jeder Verwendungsstelle.

Wir haben also eine redundante Information („Tabelle der Struktur EKKO“) auf der einen Seite und eine praktisch nicht vorhandene („Was ist in der Tabelle gespeichert?“) auf der anderen Seite. Da bietet es sich an, das EKKO im Namen zu streichen und einen sprechenden Namen zu verwenden, der den Inhalt beschreibt: ccbest, normbest, retbest (bzw. ihre englischen Äquivalente), um auf einen Blick erfassbar zu machen: Hier handelt es sich um eine Tabelle mit Cross-Company-Bestellungen, mit Normalbestellungen oder mit Retourenbestellungen.

Das passt übrigens auch zur ursprünglichen Idee: In seinem Artikel zum Thema schreibt Charles Simonyi selbst, was er mit seinem Standard für die Benennung von Variablen und Prozeduren meint. Nur, wenn man „type“ als „data type“ interpretiert, kommt das heraus, was heute als „Ungarische Notation“ verwendet wird. Gemeint war es aber keineswegs, sondern er wollte eine Beschreibung der Variablen im Kontext ihrer Verwendung in den Namen codieren. Also genau das, was hier in diesem Artikel beschrieben wird.

Kompliziert wird es, wenn sich der Datentyp ändert. Sei es, dass sich während der Entwicklung einer noch nicht produktiven Anwendung oder (noch schlimmer) während der Erweiterung einer bereits produktiven Applikation der Datentyp ändert. Das ist gar nicht so selten, wie man meint. Ich habe für einen Kunden in Hamburg gearbeitet, der sehr strikte Regeln verfolgt, bei dem sogar Einzelfelder nicht als „lv_“ benannt wurden, sondern je nach Datentyp des Feldes „li_“ (lokales Integerfeld) oder „lp_“ (lokales gepackte Feld), etc. verwendet werden mussten. Zwingend, denn der Code Inspector ließ keinen Transport zu, wenn diese Regeln nicht eingehalten waren. Wenn sich nun der Typ ändert, muss in der gesamten Anwendung an jeder Verwendungsstelle umbenannt werden, zum Beispiel von „li_“ auf „ln_“, weil herauskommt, dass der neue Datentyp „numerische Zeichenfolge“ dann doch besser ist.

Oder nehmen wir den Fall, dass in einem produzeduralen Programm mit „globalen“ und „lokalen“ Variablen (nach bisherigem Stand des Artikels sind die Anführungszeichen berechtigt) eine Variable den Sichtbarkeitsbereich wechselt. „Oh, ich brauche das Feld ja doch global!“ – viel Spaß beim Umbenennen! Und nein: Suchen/Ersetzen führt meist nicht zum Erfolg, sondern macht das Coding an anderer Stelle kaputt.

Und selbst, wenn man das ausschließen will, gibt es äußere Einflüsse, die einen Wechsel des Datentypes verlangen, wie zum Beispiel die Änderung von 32 auf 64 Bit bei Windows-Programmen. In denen steht jetzt u. U. im Variablennamen ein Datentyp, den es gar nicht mehr gibt! Das ist dann der Punkt, wo zu restriktive Namensregeln eigentlich zu syntaktisch falschen Programmen führen, wenn man diese Namensregeln als Teil der Syntax begreift.

So schreibt es auch Robert Martin, immerhin geht die agile Softwareentwicklung auf ihn zurück, in „Clean Code: A Handbook of Agile Software Craftsmanship“:

> “… nowadays HN and other forms of type encoding are simply impediments. They make it harder to change the name or type of a variable, function, member or class. They make it harder to read the code. And they create the possibility that the encoding system will mislead the reader.”

wobei er mit „HN“ die „Hungarian Notation“ meint.

Auch Linus Thorvalds verbannt die Ungarische Notation in „The Linux Kernel Coding Style“ (Chapter 4, Naming):

> “Encoding the type of a function into the name (so-called Hungarian notation) is brain damaged—the compiler knows the types anyway and can check those, and it only confuses the programmer.”

Zusammengefasst lässt sich also sagen, dass sowohl der Sichtbarkeitsbereich als auch der Datentyp nichts in einem Variablennamen verloren haben. Was ein Entwickler wirklich wissen muss, steht nicht drin – und was drinsteht, ist nur einen Doppelklick (in Eclipse noch weniger) entfernt. Nur, weil es teilweise in SAP-Coding so steht oder weil *SAP-Press-Bücher* dies so verbreiten oder auch nur, weil man es immer schon so gemacht hat, wird es nicht richtiger oder besser – und wartungsfreundlicher schon gar nicht.

Und ehe jemand fragt: Ja, ich habe es früher auch so gemacht. Weil es alle so gemacht haben, insbesondere die, unter deren Mentoring ich meine ersten Programme schrieb. Aber wir lernen alle dazu, mich und sicher auch die SAP eingeschlossen. In diesem Sinne sollte man immer wieder seine Methoden, seinen Stil und eigentlich auch sich selbst in Frage stellen. Nur wer nicht in Frage stellt, lernt nicht dazu.

Ralf Wenzel