Seminar Report

Vertiefungseminar
Cobol - 99 Bottles of Beer

Stefan Hötzl
stefan.hoetzl@uibk.ac.at

19 February 2011

Supervisor: Priv.-Doz. Dr. Georg Moser

Abstract

This article gives an overview, about cobol, one of the oldest higher programming language. It includes the history, the usage in past and nowadays, and a basic introduction to cobol.

Contents

1 Einleitung

Cobol ist eine der ältesten und weit verbreitetsten höheren prozeduralen Programmiersprachen weltweit und findet heute noch große Anwendung. Diese Arbeit soll, dem Leser die wichtigsten Grundlagen, wie die Struktur [5], Datentypen [6] und Kontrollstrukturen [7] von Cobol erläutern. Auch die Geschichte [2] von Cobol und wofür [3] es entwickelt wurde sind Gegenstand dieser Arbeit. Es werden Gründe angeführt, warum Cobol eine so große Popularität [4] erlangte, dass es nach 50 Jahren immernoch so häufig im Einsatz ist, wie es heutzutage der Fall ist. Anhand eines Beispielprogrammens [8], wird der Ablauf eines Cobol Programmes erklärt und auf wichtige Punkte eingegangen. Im Kapitel Cobol Standard 2002 [9], findet sich eine Zusammenfassung über die wichtigsten Erneuerungen gegenüber den vorhergehenden Standards. Es findet sich eine Gegenüberstellung [10] von Cobol und Java, inder die wesentlichen Unterschiede aufgezeigt werden. Abschließend wird auf die heutige Verwendung [11] von Cobol eingegangen und eine kurze Zusammenfassung [12] über diesen Artikel gegeben.

2 Geschichte

Cobol[1] ist eine der ältesten, höheren, prozeduralen Programmiersprachen, die in den 1950iger Jahren entwickelt wurde und heute noch Verwendung findet. 1959 wurde Cobol von einer Gruppe namens "Short Range Comitee", das aus EDV Herstellern und Hochschulabsolventen bestand, entwickelt. Diese Gruppe wurde, im Auftrag des amerikanischen Verteidigungsministerium gegründet, um eine Spezifikation für eine hardwareunabhängige, standardisierte, problemorientierte Sprache, zur Erstellung von Programmen für den betriebswirtschaftlichen Bereich, zu entwickeln. Diese erste Spezifikation, wurde von der Programmiersprache "Flow Matic", die von Grace Hopper entwickelt wurde, sehr geprägt, deshalb wird Grace Hopper als "die Mutter" von Cobol bezeichnet. Dieser erste Standard von Cobol, wurde von verschiedenen Organisation weiterentwickelt, da diese Standards untereinander inkompatibel waren, wurde 1968 der erste Standard von ANS (American National Standard) festgelegt. 1974 und 1985 veröffentlichte ANSI (American National Standards Institute) eine überarbeite Version, die jeweils eine Reihe neuer Features beinhaltete. Der aktuelle Standard Cobol 2002 beinhaltet moderne Aspekte wie z.b. objektorientierte Programmierung und wird bis heute weiterentwickelt.

2.1 Grace Hopper

Grace Hopper[2] war eine amerikansche Informatikerin und Pionierin im Bereich der Informatik. Sie beendete 1930 ihr Studium der Mathematik und Physik an der Universität Yale mit Auszeichnung. 1952 entwickelte sie den ersten Compiler (A-0) und 1957 die Programmiersprache "Flow Matic".

Figure 1: Grace Hopper

[1] Cobol `http://de.wikipedia.org/wiki/Cobol` Zugriff: 03.Jänner 2011
[2] Grace Hopper `http://de.wikipedia.org/wiki/Grace_Hopper` Zugriff: 03.Jänner 2011

3 Wofür wurde Cobol entwickelt

Der Haupteinsatzzweck von Cobol liegt, wie im Namen **CO**mmon **B**usiness **O**riented **L**anguage schon angedeutet, im kaufmännischen Geschäftsbereich. Die Sprache ist speziell für die Verabeitung von großen strukturierten Datenmengen im Batchbetrieb, auf Mainframesystemen geeignet. Im Bereich dieser Systeme ist Cobol nach wie vor die dominierede Sprache. Cobol findet in vielen Unternehmen, aber vorallem bei Versicherungen und Banken, die große Datenmengen zu verarbeiten haben, Anwendung. Wenn man heutzutage z.b. eine Geldbehebung am Bankomaten[3] tätigt, wird diese Transaktion meist durch Cobol verarbeitet.

4 Warum wurde Cobol populär

Cobol ist eine prozedurale Programmiersprache, die aufgrund ihrer wortreichen und stark an die natürliche Sprache angelehnte Syntax, leicht zu verstehen und von Personen ohne Programmierkenntnisse, leicht zu erlernen ist. Zu der Zeit als Cobol entwickelt wurde, gab es im Bereich, der Business Lösungen, nur Programme, die in Assembler implementiert und sehr schwer zu warten, verändern und weiterzuentwicklen waren. Der Softwareentwickler musste sich nicht mehr mit Pointern, Adressen und Computer Instruktionen beschäftigen und konnte sich auf das programmieren ansich konzentrieren. Dadurch sparten Firmen bei Entwicklungskosten große Summen an Geld.

5 Aufbau eines Cobol Programmes

Da Cobol zu einer Zeit entwickelt wurde, wo Computer mittels sogenannten Lochkarten programmiert wurden, entspricht eine Zeile Cobol Code genau einer Lochkarte mit 80 Spalten. Diese Spalten sind wie folgt definiert.

- Spalten 1-6 Zeilennummer (wird vom Compiler ignoriert)
- Spalte 7 Kennzeichnung einer Kommentar-, Fortsetzungs-, Debuggingzeile
- Spalten 8-11 Area A (Start von Division Names, Section Names, Paragraph Names, 01 Level Numbers)
- Spalten 12-72 Area B (Start von z.B. Expressions)
- Spalten 73-80 Versionsnummer

[3] Cobol mit 50 Jahren noch kein bisschen müde `http://www.zdnet.de/it_business_technik_cobol_mit_50_jahren_noch_kein_bisschen_muede_story-11000009-41004923-1.htm` Zugriff: 03.Jänner 2011

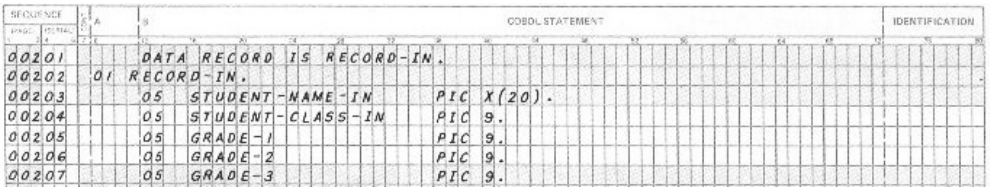

Figure 2: Cobol Programmierkarte

Jedes Cobol Programm hat eine festgelegte Struktur. Diese besteht aus den vier `DIVISIONS`:

- `IDENTIFICATION DIVISION`
- `ENVIRONMENT DIVISION`
- `DATA DIVISION`
- `PROCEDURE DIVISION`

Diese Reihenfolge ist einzuhalten. In der `IDENTIFICATION DIVISION`, stehen Informationen für den Entwickler und Compiler wie (z.b. `PROGRAM-ID`, Author, Erstellungsdatum etc.), wobei eine `PROGRAM-ID` zwingend angegeben werden muss, die anderen Prameter sind optional. Die `ENVIRONMENT DIVISION` wird unterteilt in die `CONFIGURATION SECTION` und die `INPUT-OUTPUT SECTION` und enthält Informationen über verwendete Hardware sowie externe Ressourcen (z.B. Dateien, Drucker etc.). Hier wird auch festgelegt auf welchem System, das Programm kompilliert wurde und lauffähig ist. Die vom Programm verwendeten Daten werden in der `DATA DIVISION` festgelegt und wird in die `FILE SECTION` und `WORKING-STORAGE SECTION` unterteilt. Die `FILE SECTION` beschreibt die benutzte Peripherie des Programmes. In der `WORKING-STORAGE SECTION`, erfolgt die Deklaration von Variablen, die verwendet werden. Die Parameter der `ENVIRONMENT DIVISION` und der `DATA DIVISION` sind optional. Die `PROCEDURE DIVISION` beinhaltet die eigentliche Programmlogik. Dies beinhaltet alle Anweisungen, die zur Ein- und Ausgabe, sowie zur Verarbeitung der Daten, verarbeitet werden.

`ENVIRONMENT DIVISION`, `DATA DIVISION`, `PROCEDURE DIVISION` lassen sich in Kapitel (`SECTION`) unterteilen, welche sich wiederum in Paragraphen (`PARAGRAPH`) unterteilen lassen. Paragraphen beinhalten Sätze (`SENTENCE`), die die eigentlichen ausführbaren Anweisungen beinhalten, in Figure 3 ist diese Hirarchie zum besseren Verständnis grafisch dargestellt.

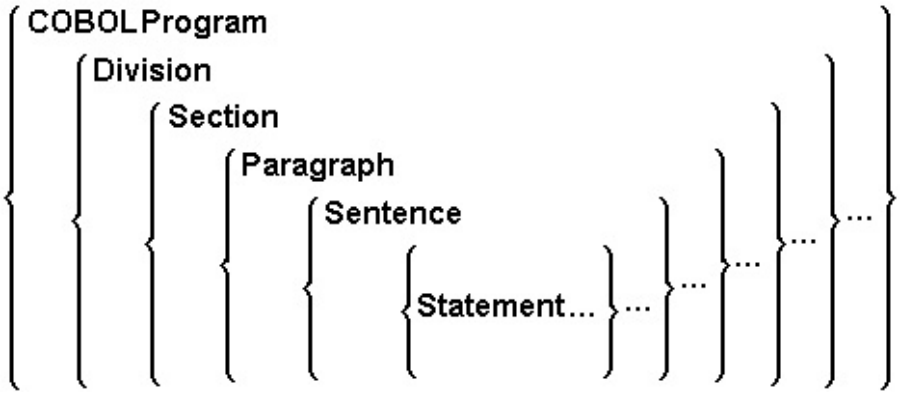

Figure 3: Cobol Hirarchie

6 Datentypen

Datentypen wie Integer oder Double, wie man z.b. aus C++ kennt gibt es in Cobol nicht. In Cobol kommt ein schwaches Typisierungskonzept zum Einsatz. Es wird nur zwischen Numerischen- bzw. Alphanumerischen Datentypen unterschieden. Numerische Datentypen können die Ziffern 0-9, einen Dezimalpunkt, ein Vorzeichen enthalten und dürfen maximal 18 Ziffern lang sein, wohingegen der Alphanumerische Datentyp, auch Groß- bzw. Kleinbuchstaben und Sonderzeichen enthalten kann. Die Deklaration der Variablen erfolgt, wie in Kapitel 4 bereits beschrieben, in der `WORKING-STORAGE SECTION` und der `DATA DIVISION`. Die Deklaration sieht folgendermaßen aus:

```
Stufennummer Variablenname PICTURE Datentyp
```

Das Schlüsselwort `PICTURE`, definiert den Datentyp und die Länge (Anzahl der Zeichen) einer Variable. Dazu gibt es 2 Maskenzeichen, "9" steht für einen Numerischen- und "X" steht für einen Alphanumerischen Datentyp. Die Länge einer Variable wird z.b. mit 999 angegben, dies würde bedeuten, dass es sich um eine numerische Variable mit der Länge 3 handelt, es gibt aber auch die Vereinfachung, die Länge dem Maskenzeichen als Parameter zu übergeben. Zwei Beispiele zur Verdeutlichung dieses Schemas:

```
01 StudentenId PIC 9999999
01 StudentenId PIC 9(7)
```

Soll eine Variable auch negative oder nicht-ganzzahlige Werte beinhalten, muss dies bei der Definition mitangegeben werden. Dafür bietet Cobol die beiden Maskenzeichen "S" bzw. "V". Wird das Zeichen "S" als Präfix an die Datendefinition anghängt, so wird das Vorzeichen mitabgespeichert. Das Zeichen "V" bestimmt die Position des Dezimalpunktes. Die folgende Deklaration definiert die Variable Temperatur mit Vorzeichen und einer Nachkommastelle.

```
01 Temperatur S9(3)V9
```

Mithilfe der Stufennummer können zusammengestzte Datetypen wie z.b. ein struct in C, definiert werden. Im folgenden Beispiel wird der Datentyp StudentenDetails definiert der sich aus einer StudentenId und einem wiederum zusammengestzten Datentyp StudentenName zusammensetzt. Auf diese Weise können Hirarchien in Datenstrukturen abgebildet werden.

```
01 StudentenDetails
   02 StudentenId      PIC 9(7)
   02 StudentenName
      03 Vorname       PIC X(10)
      03 Nachname      PIC X(15)
```

7 Kontrollstrukturen

In Cobol gibt es, so wie in anderen Programmiersprachen, Kontrollstrukturen. Im folgenden Abschnitt, werden ausgewählte behandelt.

7.1 IF Syntax

Neben einfachen IF Statements, biete Cobol auch die Möglichkeit, der verschachtelten IF Statements.

```
IF VarA > VarD THEN
   Statement1
ELSE
   Statement2
END-IF
```

```
IF VarA > 10 THEN
   IF VarA < 15 THEN
      DISPLAY "lower than 15"
   END-IF
ELSE
   DISPLAY "lower than 10"
END-IF
```

7.2 Switch Anweisung (EVALUATE)

Die Switch Anweisung, wie man sie z.B. aus Java oder C kennt, wird in Cobol mit dem Schlüsselwort EVALUATE eingeleitet. EVALUATE[4] ist mächtiger als eine Switch Anweisung, es besteht nämlich die Möglichkeit mithilfe des Schlüsselwortes ALSO, zwei Werte auf einmal zu prüfen, wie die folgenden Beispiele zeigen.

[4] Cobol Tutorium `http://www.csis.ul.ie/cobol/course/cobolintro.htm` Zugriff: 03.Jänner 2011

```
EVALUATE Geschlecht ALSO FamStand
   WHEN "w" ALSO "s"
      DISPLAY "Frau, ledig"
   WHEN "w" ALSO "v"
      DISPLAY "Frau verheiratet"
   WHEN "m" ALSO "s"
      DISPLAY "Mann ledig"
   WHEN "m" ALSO "v"
      DISPLAY "Mann verheiratet"
   WHEN OTHER
      DISPLAY "Fehler"
END-EVALUATE.
```

```
EVALUATE Geschlecht
   WHEN "w"
      DISPLAY "Frau"
   WHEN "m"
      DISPLAY "Mann"
   WHEN OTHER
      DISPLAY "Fehler"
```

7.3 Beziehungs Bedingung (Relation Conditions)

Beziehungs Bedingungen testen ob ein Wert, größer, gleich groß oder kleiner ist, als ein anderer Wert. Die Auswertung einer Bedingung gibt immer TRUE oder FALSE zurück, und nicht so wie z.b. in C 0 oder 1. Zu beachten ist, dass nur Werte des gleichen Typs verwendet werden dürfen. Folgendes Beispiel wäre ungültig und würde zu einem Fehler führen.

```
IF "243" IS GREATER THAN 3445 THEN ...
```

Beispiele für gültige Bedingungen:

```
IF "Haus" IS EQUAL TO "Auto" THEN ...
IF 243 IS GREATER THAN 3445 THEN ...
IF 456 IS LESS THAN 564 THEN...
```

7.4 Vorzeichen Bedingung (Sign Conditions)

Vorzeichen Bedingungen prüfen ob ein Wert positive, negativ oder Null ist.

```
0 IS POSITIVE
4 IS NEGATIVE
Var1 IS NULL
```

7.5 For-Schleife (Perform...Times)

Perform...Times hat eine Besonderheit, die es so in keiner mir bekannten, anderen Programmiersprache gibt. Es besteht die Möglichkeit einen Codeblock "In-line" oder "Out-of-line" auszuführen. "In-line" bedeutet, dass die auszuführenden Anweisungen inner halb des Perform...Times Block steht, wohingegen "Out-of-line" bedeutet, dass die Anweisungen z.b. in einem Pragraph zusammengefasst werden und direkt auf aufgerufen werden. "Out-of-line" bietet somit die Möglichkeit Anweisungen, die man öfter braucht zusammenzufassen und an verschiedenen Stellen wiederzuverwenden.

```
PERFORM 10 TIMES                    PERFORM OutOfLine 10 TIMES
    DISPLAY "Inline"
END-PERFORM                         OutOfLine
                                        DISPLAY "Out-of-line"
```

8 Beispielprogramm

In diesem Kapitel wird anhand eines Beispiels der Ablauf eines Cobol Programmes erklärt. Dieses Programm benötigt 2 Zahlen und einen Operator (+/-), als Eingabe, diese wird dann berechnet und auf dem Bildschirm ausgegeben.

```
000000 IDENTIFICATION DIVISION.
000010 PROGRAM-ID.  Iteration-Calculator.
000020 DATA DIVISION.
000030 WORKING-STORAGE SECTION.
000040 01  NR1         PIC 9  VALUE ZEROS.
000050 01  NR2         PIC 9  VALUE ZEROS.
000060 01  Result      PIC 99 VALUE ZEROS.
000070 01  Operator    PIC X  VALUE SPACE.
000080 PROCEDURE DIVISION.
000090 Calculator.
000100     PERFORM 3 TIMES
000110     DISPLAY "First Number: "
000120     ACCEPT NR1
000130     DISPLAY "Second Number: "
000140     ACCEPT NR2
000150     DISPLAY "(+ or *): "
000160     ACCEPT Operator
000170     IF Operator = "+" THEN
000170         COMPUTE Result = R1 + NR2.
000200     IF Operator = "*" THEN
000210         COMPUTE Result = NR1 * NR2.
000230     DISPLAY "Result is = ", Result
000240     END-PERFORM.
000250     STOP RUN.
```

Bei Ausführung wird das Programm Zeile für Zeile abgearbeitet. In der `IDENTIFICATION DIVISION`, ist hier nur die `PROGRAM-ID` angegeben, die benötigt wird, damit das Programm kompiliert. Es könnten hier noch Author und Erstellungsdatum angegeben werden. Die `DATA DIVISION` enthält die Deklaration der vier Variablen NR1, NR2, Result und Operator. In diesem Beispiel fehlt die `ENVIRONMENT DIVISION`, dies bedeutet, dass keine Dateien bzw. spezielle Hardware verwendet werden. Die `PROCEDURE DIVISION`, beinhaltet eine Prozedur Calculator. In dieser wird eine Schleife 3 mal ausgeführt, die den Benutzer auffordert zwei Zahlen und einen Operator einzugeben und speichert

diese Eingaben in den entsprechenden Variablen ab. Dann wird je nach Operator das Ergebnis berechnet und auf dem Bildschirm ausgegeben. Ist die Schleife fertig durchlaufen, wird der Befehl STOP RUN ausgeführt, der das Programm beendet.

9 Cobol Standard 2002

Dieser Standard ist der neueste von Cobol. Hier werden die wichtigsten Erneuerungen zu den älteren Standards behandelt. Die wohl wichtigste Änderung ist, dass die Zeilenlänge von 80 Zeichen auf 255 Zeichen erweitert und auf die Formatierung, die in Kapitel 5 beschrieben wird, verzichtet wurde. Da eine Programmiersprache, die wirklich ernstgenommen werden will, ohne Objektorientierung nicht auskommt, wurden Erweiterungen dafür eingeführt. Es wurde ein XML Parser integriert, der über das Schlüsselwort XML PARSE verwendet werden kann. Ein Problem, das Entwickler oft hatten, war es, das es nur Globale Variablen gab und wenn Variablen benötigt wurden, die nur in einer Prozedur Verwendung fanden, mussten diese nachträglich in der DATA DIVISION hinzugefügt werden, deshalb ist es nun möglich lokale Variablen in Prozeduren zu definieren.

10 Vergleich Cobol Java

Cobol basierd auf einem Prozeduralen Programmiermuster, wohingegen Java auf einem Objektorientierten Programmiermuster basierd. Java ist streng typisiert und kennt verschiedenste Datentypen, wie z.B. Integer, Float, Double etc.. Cobol ist schwach typisiert und unterscheidet nur zwischen 2 Datentypen, dem Numerischen- und Alphanumerischen-Typ. Cobol erlaubt die Variablen Deklaration nur in der dafür vorgesehenen DIVISION, zudem sind alle Variablen global. In Java können Variablen fast überall deklariert werden und haben einen gewissen Gültigkeitsbereich.
Da Cobol darauf spezialisiert ist, große Datenmengen und viele Transaktionen zu verarbeiten, eignet es sich weniger um komplexe Algorithmen zu implementieren, wohingegen Java keinerlei Probleme damit hat komplexe Algorithmen abzubilden. Wenn man aber mit Java große Datenmangen, vernünftig verarbeiten will, wird man um eine Datenbank nicht herum kommen, dies beteutet wiederum einen großen Mehraufwand in der Entwicklung. Die folgende Implementierung des Beispielprogrammes aus Kapitel 8, in beiden Sprachen soll die Unterschiede verdeutlichen.

Cobol	Java

000000 IDENTIFICATION DIVISION.
000010 PROGRAM-ID. Iteration-Calculator.
000020 DATA DIVISION.
000030 WORKING-STORAGE SECTION.
000040 01 Num1 PIC 9 VALUE ZEROS.
000050 01 Num2 PIC 9 VALUE ZEROS.
000060 01 Result PIC 99 VALUE ZEROS.
000070 01 Operator PIC X VALUE SPACE.
000080 PROCEDURE DIVISION.
000090 Calculator.
000100 PERFORM 3 TIMES
000110 DISPLAY "Enter First Number: "
000120 ACCEPT Num1
000130 DISPLAY "Enter Second Number: "
000140 ACCEPT Num2
000150 DISPLAY "Enter operator (+ or *):
000160 ACCEPT Operator
000170 IF Operator = "+" THEN
000170 COMPUTE Result = Num1 + Num2
000190 END-IF
000200 IF Operator = "*" THEN
000210 COMPUTE Result = Num1 * Num2
000220 END-IF
000230 DISPLAY "Result is = ", Result
000240 END-PERFORM.
000250 STOP RUN.
	public class IterationCalculator {
	private int Num1;
	private int Num2;
	private int Result;
	private String Operator;
	
	public void calculate() {
	Scanner sc = new Scanner(System.in);
	for(int i=0; i<3; i++) {
	System.out.println("First Number:");
	Num1 = sc.nextInt();
	System.out.println("Second Number:");
	Num2 = sc.nextInt();
	System.out.println("Operator (+ or *)");
	Operator = sc.next();
	if(Operator.equals("+"))
	Result = Num1 + Num2;
	if(Operator.equals("*"))
	Result = Num1 * Num2;
	System.out.println("Result = " + Result);
	}
	}
	public static void main(String[] args) {
	new IterationCalculator().calculate();
	}
	}}

11 Cobol Heute

Cobol ist nach wie vor die dominierende Sprache auf Mainframesystemen für Businessanwendungen. Datamonitor[5] schrieb in einem Bericht, "Cobol-Continuing to Drive Value in the 21st Century", vom November 2008 folgendes:

- Weltweit sind 200 Milliarden Zeilen Cobolcode aktiv im Einsatz.
- 75% der anfallenden Daten in Businessanwendungen und 90% der weltweiten Finanztransaktion, werden von Cobolprogrammen verarbeitet.
- Jedes Jahr werden 5 Milliarden neue Zeilen Cobolcode, implementiert.

Ein weiterer Grund dafür, dass Cobol in naher Zukunft, nicht abgelöst werden wird, ist der, dass viele Programme die teilweise schon über 30 Jahre alt sind und über diese Zeit sehr gewachsen sind, immernoch in Verwendung sind und die Kosten für eine Neuentwicklung, meist um ein vielfaches teurer sind, als die Wartung und weiterentwicklung der bestehenden Lösungen. Auch die stetige Weiterentwicklung der Sprache und Features, wie z.b. die Einführung objektorientierter Programmierung, oder die Integration in andere Programmiersprachen (z.b. .NET), verhelfen Cobol dazu, nicht an popularität zu verlieren. Das folgende Beispiel soll verdeutlichen, wie sich Cobol in .NET integrieren lässt. Diese Erweiterung wird durch NetCobol[6] der Firma Fujitsu[7] bereitgestellt.

[5] http://www.datamonitor.com/
[6] http://www.fujitsu.com/global/services/software/netcobol/ Zugriff: 18.2.2011
[7] http://www.fujitsu.com/

```
010 CLASS-ID. HELLO INHERITS FORM.
020 ENVIRONMENT DIVISION.
030 CONFIGURATION SECTION.
040 REPOSITORY.
050  PROPERTY WIN-TEXT AS "Text"
060  CLASS APPLICATION AS "System.WinForms.Application"
070  CLASS FORM AS "System.WinForms.Form".
080 FACTORY.
090 PROCEDURE DIVISION.
100 METHOD-ID. MAIN.
110 DATA DIVISION.
120 WORKING-STORAGE SECTION.
130  77 APP-OBJ USAGE OBJECT REFERENCE FORM.
140 PROCEDURE DIVISION.
150  INVOKE HELLO "NEW" RETURNING APP-OBJ.
160  MOVE "Hallo Cobol" TO WIN-TEXT OF APP-OBJ.
170  INVOKE APPLICATION "Run" USING BY VALUE APP-OBJ.
180 END METHOD MAIN.
190 END FACTORY.
200 END CLASS HELLO.
```

Dieses einfache Beispiel[8] öffnet ein Framefenster und gibt den String "Hallo Cobol" aus. Auf den ersten Blick sieht, es so aus als wäre, das ein ganz normales Cobol Programm, bei näherer Betrachtung sieht man aber große Unterschiede. Als erstes fällt auf, dass die Programm-ID fehlt, die ja wie in Kapitel 4 beschrieben, eigentlich zwingend notwendig ist, damit das Progamm kompiliert wird. Hier handelt es sich aber um objektorientiertes Cobol und eine Klasse. Dies ist also eine Klasse namens HELLO und erbt von der Klasse FORM. In den Zeilen 50, 60 und 70 werden benutzte Ressourcen definiert, dabei handelt es sich um .NET Klassen. Zeile 100 definiert eine Methode namens MAIN, das auch ein Konstrukt von objektorienterten Cobol ist. Interessant ist die Zeile 130, hier wird ein APP-OBJ Objekt angelegt, das auf ein Objekt der .NET Kasse FORM referenziert. Diese Objekt wird in Zeile 150, 160, 170 dazu benutzt, um das Formular anzulegen, und den Text "Hallo Cobol" auszugeben.

[8] `http://msdn.microsoft.com/de-de/library/bb978974.aspx` Zugriff: 14.2.2011

12 Zusammenfassung

In dieser Arbeit wurden die wichtigsten Eigenschaften und formalen Strukturen, der Programmiersprache Cobol erläutert. Es wurde beschrieben, wie die Sprache entstand und wie sie sich geschichtlich entwickelt hat. Der große Erfolg von Cobol ist darauf zurückzuführen, dass es wortreich und die Syntax stark an die natürliche Sprache angelehnt ist, dies hatte gegenüber der damals verwendeten Assemblersprachen, den Vorteil, dass die Sprache leicht zu verstehen und lernen war. Es konnten dadurch Entwicklungskosten verringert werden, da sich Entwickler mehr auf das programmieren als auf die Bugsuche konzentrieren konnten. All das führte dazu, dass eine mittlerweile 50 Jahre alte Sprache immernoch große Verwendung findet.

References

[1] Reimann Artur. COBOL, Language of Choice - Then and Now (January, 2001), COBOLReport.com

[2] Daniel D. McCracken. COBOL - Anleitung zur strukturierten Programmierung, 5. Aufl., Oldenbourg Verlag, 1987

[3] Wolf-Michael Kähler: Programmieren in COBOL 85, 5. Aufl., Vieweg Verlag, 1991.