W e r n i g e r o d e

Fachbereich Automatisierung und Informatik

Vorlesung „Betriebssysteme“

Kapitel Java Native Interface

JNI

Version 26. Juni 2006

Dipl.-Inf., Dipl.-Ing. (FH) Michael Wilhelm
Friedrichstraße 57 - 59
38855 Wernigerode

Raum: 2.202
Tel.: 03943/659-338
Fax: 03943/659-399
Email: mwilhlem@hs-harz.de

Inhaltsverzeichnis

1 Abbildungsverzeichnis

2 Quelltextverzeichnis

3 Abkürzungs- und Dateierweiterungsverzeichnis

JNI Java Native Interface

API	Application Programm Interface
GUI	Grafik-User-Interface
*.JAVA	Javadatei, enthält den Quellcode
*.CLASS	Übersetzte Javadatei, Quellcode wurde in Zwischencode umgewandelt. Dieser wird von einem Interpreter „JVM“ ausgeführt
*.H	Headerdatei, dient der Definition von Funktionen und Methoden
*.CPP	C++ Datei, enthält den Quellcode einer DLL
*.OBJ	Übersetzte C++ Datei, enthält die Maschinenbefehle
*.DLL	Dynamik Linked Library. Diese Datei stellt Funktionen anderen Programmen zur Verfügung. Sie hat **keine** main-Methode, kann also nicht direkt aufgerufen werden.
*.SO	Modul-Datei unter Unix. Entspricht dem einer DLL. Diese Datei stellt Funktionen anderen Programmen zur Verfügung. Sie hat **keine** main-Methode, kann also nicht direkt aufgerufen werden.

4 Java Native Interface

4.1 Einleitung

Java ist durch seine verschiedenen API's[1] eine sehr gute Programmiersprache, mit der sehr leistungsfähige Programme in vielen Bereichen erstellt werden können. Von einfachen Dialogfenster bis hin zu MDI-fähigen Programmen, von einfachen Datenbankprogrammen bis hin zu verteilten Programmen über das Internet. Sie ist standardisiert und läuft somit auf allen Betriebssystemen die eine Javaumgebung zur Verfügung stellen.

In einigen Fällen ist es aber notwendig Programmteile in einer anderen Sprache zu implementieren:

- Zugriff auf spezielle Funktionen des Betriebsystems.
- Zugriff auf Hardware-Ressourcen.
- Zugriff auf Code, der nicht in Java geschrieben wurde.
- Implementation zeitkritischer Module. Siehe dazu Beispiel 5

Diese Kurzanleitung beschreibt die Möglichkeiten, mit Hilfe des Java Native Interface (JNI) Zugriff auf eine Dynamic Link Library (DLL) zu bekommen. Dabei werden im diesem Kapitel die theoretischen Grundlagen angesprochen. Danach folgen mehrere ausgetestete Beispiele. Es wird vorausgesetzt, dass Grundlagen in Java und in C bzw. C++ bestehen.

4.2 Theorie des JNI

Das Java Native Interface ist eine Schnittstelle, um nativen Code in Java einzubinden. Der native Code wird in eine DLL gekapselt. Die Kommunikation zwischen der Javaklasse und dem nativen Code erfolgt über das JNI.

Folgende Sprachen können über eine DLL und JNI mit Java verknüpft werden:

- Assembler
- C / C++
- COBOL
- Delphi
- FORTRAN

Nativer Code in den Sprachen Assembler, C und C++ können mit jedem handelsüblichen Compiler in eine DLL umgewandelt werden. Die Sprache FORTRAN kann über das „Visual Studio“ und der Programmierumgebung „Digital Visual Fortran“ in eine DLL eingebettet

[1] awt, net, rmi, sql, zip, sound, swing, swt

werden. Für „Delphi“ existiert eine eigene Umsetzung der JNI-Headerdatei in eine für Delphi konforme Unit. Die Anpassung an die Sprache COBOL ist unter [8] zu finden.

Möglichkeiten des Java Native Interface:

- Aufruf von Funktionen des nativen Codes.
- Übergabe von Parametern:
  - Skalare Parameter (String, char, int, long, float, double, boolean)
  - Arrays
  - Strukturen
  - Klassen
- Threads
- Exception Verwaltung.
- Aufruf von statischen und dynamischen Methoden.
- Adressierung von Feldern.
- Referenzierung von Objekten und Objektoperationen.

Vorgehensweise:

Die nächsten Kapitel beschreiben detailliert die Vorgehensweise zur Erzeugung einer DLL, die aus einem Javaprogramm aufgerufen werden kann. Abbildung 4 zeigt diese Vorgehensweise in einer Grafik.

4.2.1 Java Klasse / Programm erzeugen

Der erste Schritt besteht in der Erstellung des Javasourcecodes. Dieser kann mit Hilfe eines Texteditors bzw. einer Programmierumgebung erstellt werden. Das untere Beispiel zeigt den minimalen Sourcecode für ein Javaprogramm.

```
public class Bsp1 {

   public static void main(String[] args) {
     Bsp1 app  = new Bsp1();
   }

} // Bsp1
```

Quelltext 1 Javasourcecode Beispiel1

4.2.2 Deklaration der nativen Methode

Der Quelltext 1 wird nun so erweitert, dass über das JNI ein Zugriff auf die DLL „dll1“ erfolgen kann. Folgende Elemente werden in den Sourcecode eingetragen:

Verweis auf die DLL:

```
System.loadLibrary("dll1");  // lädt dll1.dll
```

Prototyp auf die Methode in der DLL:
public native void displayHelloHSHarz() ();

Aufruf der Methode in der DLL:
App.displayHelloHSHarz();

```
import java.io.*;

public class Bsp1 {

   // native method in the DLL
   public native void displayHelloHSHarz();

   static {
      System.loadLibrary("dll1");
   }

  private static void pause() {
   try {
     System.in.read();
   }
   catch (IOException e) {
   }
 }  // pause

   public static void main(String[] args) {
     Bsp1 app  = new Bsp1();
     app.displayHelloHSHarz();
     System.out.println("Hier wieder in Java");
     pause();
   }

} // Bsp1.java
```

Quelltext 2 Javasourcecode Beispiel1

In der „main-Routine“ wird ein Objekt der Klasse „Bsp1“ erzeugt und die Methode aus der DLL aufgerufen.

Die Methode „pause“ dient nur dazu, das „Dos-Fenster“ anzuzeigen. Mit Betätigen der Taste „Return“ bzw. „Enter“, verschwindet das Fenster. Diese Anweisung kann aber auch, besonders unter Dos oder Eclipse, weggelassen werden.

4.2.3 Übersetzen der Javadatei

Der Quellcode „Bsp1.java“ wird mit dem Kompiler „javac“ übersetzt und liefert die Datei „Bsp1.class“. Dabei muss ein Pfad zu den Javaprogrammen eingetragen sein. Andernfalls kann man den Pfad direkt angeben.

Notation für die DOS-Ebene:

javac Beispiel1.java

bzw. mit Pfadangabe:

c:\jbuilder4\jdk1.4\bin\javac Beispiel1.java

Als Ergebnis erhält man die gewünschte Datei Beispiel1.class.

4.2.4 Erzeugen der Headerdatei

Mit der Klassendatei und dem Hilfsprogramm „javah“ wird die Headerdatei für die DLL erzeugt. Diese Headerdatei definiert den „Rumpf“ der nativen-Methoden. Insbesondere werden die Headerdatei „jni.h“ und „jni_md.h“ eingelesen. Diese sorgen für die Anbindung an die JVM.

Notation für die DOS-Ebene:

javah -jni Beispiel1

Der Quelltext 3 zeigt die erzeugte Headerdatei.

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class Bsp1 */

#ifndef _Included_Bsp1
#define _Included_Bsp1
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     Bsp1
 * Method:    displayHelloHSHarz
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_Bsp1_displayHelloHSHarz
  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

Quelltext 3 Von javah erzeugter Dateiheader

In der Headerdatei wird die native Methode definiert. Insbesondere wird durch die Signatur die Methode definiert. Das obige Beispiel hat die Signatur ()V. Das bedeutet, das die Methode keine Parameter erwartet und keinen Wert zurückgibt.

Hinweis: Falls in der Javadatei ein Packagename angegeben wurde, erwartet das Programm „javah“ diesen auch als Parameter im Dateinamen. Des Weiteren muss die Klassendatei entsprechend des Package-Angabe gespeichert sein.

Beispiel:

```
package mypackage;

class myclass {


} // myclass
```

Aufruf

```
javah –jni mypackage.myclass
```

Die Konvention des Namens für die Methode im nativen Code sind von Java fest vorgegeben. Jeder Teil wird durch einen Underline (_) getrennt. Der erste Prefix lautet immer „Java“. Danach werden – falls vorhanden – die Package-Angaben, jeweils getrennt durch das Zeichen „_“ eingetragen. Danach kommt der Klassenname der Javadatei und am Schluss der Name der nativen Methode in der DLL:

4.2.5 Frame für eine DLL erzeugen

Falls ein C-Entwicklungssystem – Kompiler und Linker – auf dem Rechner vorhanden ist, kann man damit automatisch eine leere DLL erzeugen. Die nächsten beiden Quelltexte wurden mit unterschiedlichen Systemen erzeugt und zeigen jeweils eine DLL mit der „main-Routine“.

```
#include "stdafx.h"

BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                        )
{
   return TRUE;
}
```

Quelltext 4 Dynamic Link Libary mit Visual Studio

```
#include <vcl.h>
#pragma hdrstop

int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void*)
{
   return 1;
}
```

Quelltext 5 Dynamic Link Libary mit CBuilder

Die DLL bzw. das Projekt muss unter dem Namen „Beispiel1“ abgespeichert werden.

4.2.6 Erzeugen einer DLL mitttels Visual Studio

- Starten des Visual Studios
- Aufruf eines neuen Projektes mit Strg+N
- Eintragen des Pfades
- Eintragen des Namens der Dll: „dll1“

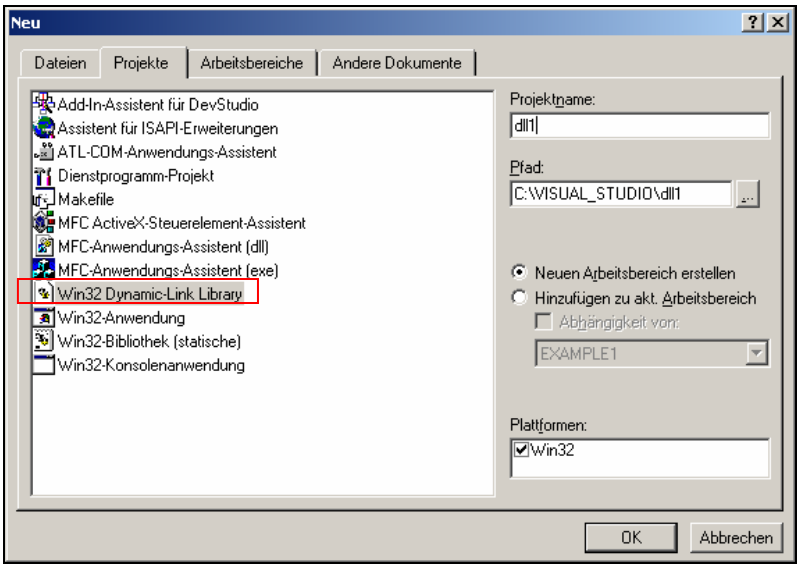


Abbildung 1 Erzeugen einer DLL mitt Visual Studio

Im nächsten Fenster wird der Typ der DLLausgewählt.

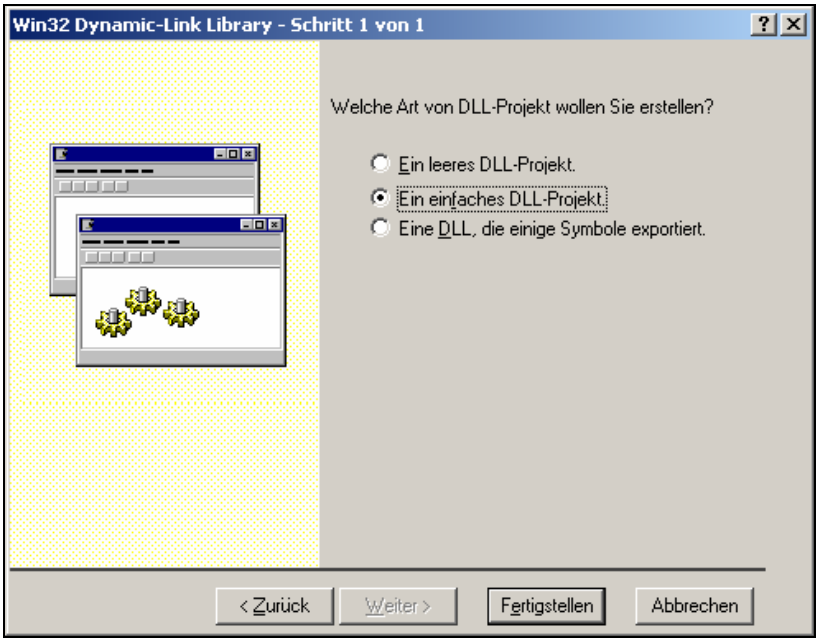


Abbildung 2 Typ der DLL

Mit diese Schritten hat eine DLL definiert. Mit der Taste „F7“ kann man die Dll erzeugen. Im Register „Datei“ wird der Quellcode angezeigt.

4.2.7 Den Dateiheader einbinden und die Prozedur erzeugen

In der DLL wird der erzeugte Header über eine Include-Anweisung eingebunden (Dateiname ist Hochkommas). Danach wird die dazugehörige Prozedur in die DLL eingetragen. Benutzt man dazu die Headerdatei, darf man das Eintragen der Parameter „env“ und „obj“ etc. nicht vergessen.

Hinweis;
Die erzeugte Headerdatei „Bsp1.h“ muss aus dem Java-Verzeichnis in das C-Verzeichnis kopiert werden.

```
#include <vcl.h>
#pragma hdrstop


#include "Bsp1.h"

int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void*)
{
   return 1;
}
//---------------------------------------------------------------------------


JNIEXPORT void JNICALL Java_Bsp1_displayHelloHSHarz (JNIEnv *env, jobject obj)
{
  puts("Hallo World");
}
```

Quelltext 6 Einfügen des Headers und Erzeugen der neuen Methode (C++ Builder)

Hinweis: Die Headerdatei benötigt die beiden Dateien „jni.h“ und „jni_md.h“. Beide Dateien sind im jeweiligen JDK zu finden. Ist ein Pfad zu den Dateien gesetzt braucht man nichts mehr zu unternehmen. Andernfalls kann man im Visual Studio den Pfad einfügen.
Die zweite Möglichkeit besteht darin, die Headerdatei in das aktuelle C-Verzeichnis zu kopieren. Danach ändert man in der Headerdatei „Bsp1.h“ den Verweis auf die Datei jni.h in eingeschlossene doppelte Hochkommas. Eventuell muss man auch noch in der Datei „jni.h“ die eckigen Klammer ersetzen.

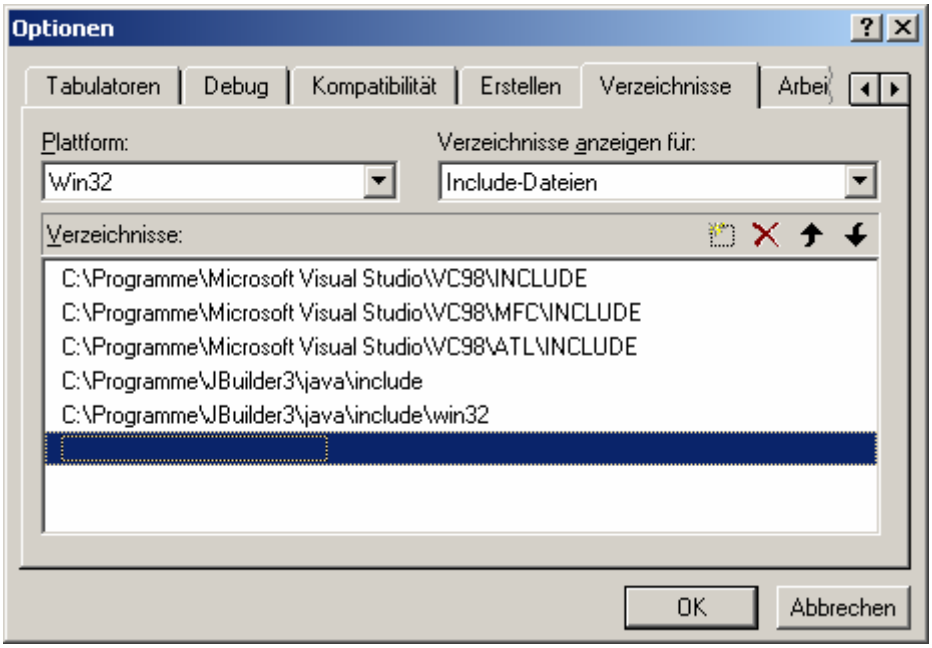
Eintragen der Headerdateien in Visial Studio:

- Aufruf Extras / Optionen
- Auswahl des Registers Verzeichnisse
- Auswahl der ComboBox „Include-Dateien“

Eintragen der beiden Verzeichnisse:

- C:\Programme\JBuilder3\java\include
- C:\Programme\JBuilder3\java\include\win32

Eingetragen in das Dialogfenster ergibt sich folgendes Bild:



Includepfade

Im nächsten Schritt werden aus der Headerdatei Bsp1.h die Methoden, hier nur eine, in die DLL-Quelldatei kopiert. Der zusätzliche Quellcode wird in die Methoden eingetragen (puts).

```
// dll1.cpp : Definiert den Einsprungpunkt für die DLL-Anwendung.
//

#include "stdafx.h"
#include "Bsp1.h"

BOOL APIENTRY DllMain( HANDLE hModule,  DWORD  ul_reason_for_call,  LPVOID lpReserved )
{
    return TRUE;
}

JNIEXPORT void JNICALL Java_Bsp1_displayHelloHSHarz JNIEnv *env, jobject obj)
{
  puts("Hallo World");
}
```

Quelltext 7 Einfügen des Headers und Erzeugen der neuen Methode (Visual Studio)

4.2.8 DLL übersetzen

Im nächsten Schritt wird die DLL übersetzt und erzeugt. Unter dem CBuilder mit der Taste „STRG+F9“, unter Visual Studio mit der Taste „F7“.

4.2.9 Das Java-Programm starten

Die DLL wird in das Java-Verzeichnis kopiert. Mit dem Befehl „java“ wird das Programm gestartet.

DOS-Ebene:

java Beispiel1

Ausgabe:

Hello World

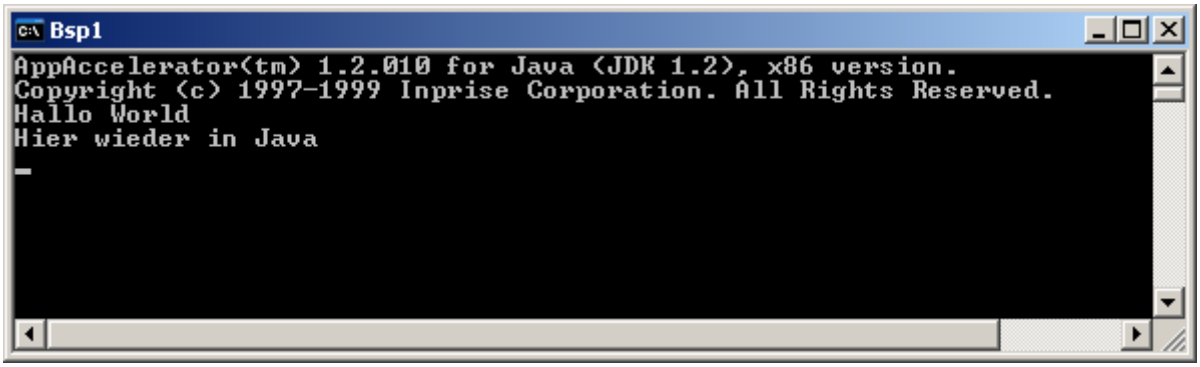


Abbildung 3 Aufruf des ersten Beispiels

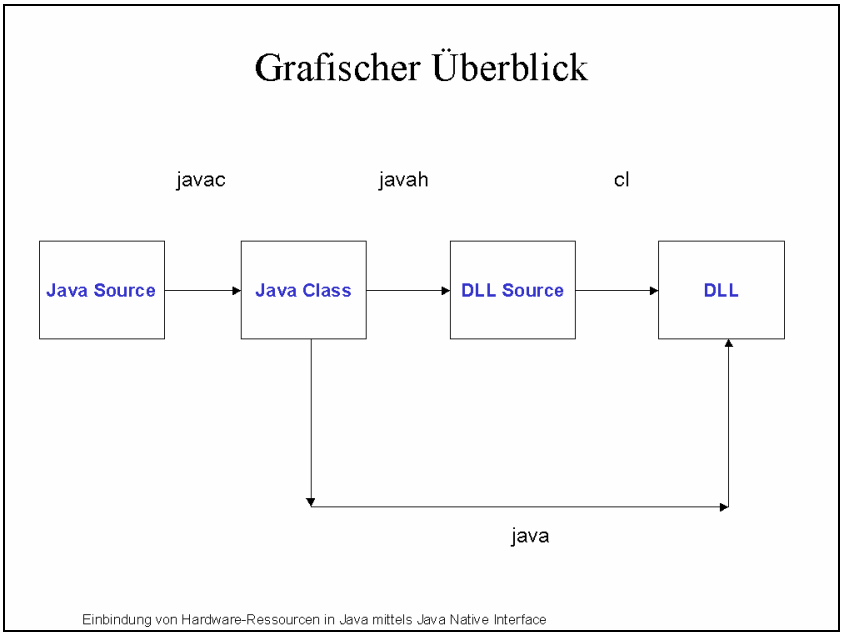


Abbildung 4 Grafischer Überblick über das Java Native Interface

4.3 Signaturen

Die Typen der Parameter einer C-Methode werden durch Abkürzungen gekennzeichnet. Für zwei Parameter „int“ und „double“ gib man die Kennzeichen „ID“ hintereinander in der Klammer an. Der Buchstabe „V“ zeigt an, dass die Methode keinen Wert zurückgibt.

Hilfe liefert dazu auch das Hilfsprogramm „javap“. Dieses generiert aus der Javadatei Methodensignaturen.

```
C:\HSHarz>javap -s -p Example4
Compiled from Example4.java
public class Example4 extends java.lang.Object {
    public Example4();
        /*   ()V   */
    public native void callJavaMethode();
        /*   ()V   */
    public native void callJavaMethode2(int, double);
        /*   (ID)V   */
    public void printNumber(int, double);
        /*   (ID)V   */
        /*   ()V   */
}

C:\HSHarz>
```

Quelltext 8 Aufruf des Hilfsprogramms javap

Die Signaturen können auch folgender Tabelle entnommen werden:

Signature	**Java Typ**	**Bemerkung**
Z	boolean	
B	byte	
C	char	
S	short	
I	int	
J	long	
F	float	
D	double	
LvollAdressierteKlasse;	VollAdressierteKlasse	
[type	Type []	Array
(arg-types) Returntyp	Returntyp	

4.4 Optionen des Programms javah

Im folgenden finden Sie eine Liste, die einige der Optionen aufführt, die mit javah verwendet werden können:

Optionen	**Beschreibung**
-jni	Erstellt eine JNI-Header-Datei
-verbose	Zeigt Statusinformationen an
-version	Zeigt die javah-Version an

-o directoryName	Gibt die .h-Datei im angegebenen Ordner aus
-classpath path	Überschreibt den Standardpfad für Klassen

Die von javahgenerierte .h-Datei enthält alle Funktionsprototypen für die nativen Methoden der Klasse. Die Prototypen werden so verändert, daß das Java-Laufzeitsystem die nativen Methoden sucht und aufruft. Im Grunde genommen wird der Name der Methode gemäß einer Namenskonvention für den Aufruf von nativen Methoden verändert. Dem Klassen- und Methodennamen wird das Präfix Java_ vorangestellt. Bei einer nativen Methode namens nativeMethod in einer Klasse mit dem Namen myClass wird folgender Name in der Datei myClass.h angezeigt: Java_myClass_nativeMethod.

Das nächste Kapitel beschreibt weitere Beispiele mit unterschiedlichen Aspekten.

5 Beispiele mit JNI

5.1 Liste der Beispiele

- Bsp1 Einfache Ausgabe eines Textes
- Bsp2 Aufruf einer double-Funktion
- Bsp3 Übergabe eines Javas-Strings an eine C-Methode
- Bsp4 Berechnen einer Wertetabelle in C, Ausgabe in Java
- Bsp5 Aufruf einer C-Methode mit Übergabe eines Arrays

5.2 Aufruf einer Berechnungsmethode

Das Beispiel zeigt, wie man eine einfache Funktion aus Java aufrufen kann. Der Quellcode ist in Quelltext 9 dargestellt.

```
import java.io.*;

public class Bsp2 {

   private static void pause() {
    try {
      System.in.read();
    }
    catch (IOException e) {
    }
  }  // pause

// native method in the DLL
public native double callCalc(double a, double b);

    static {
        System.loadLibrary("dll2");
    }

    public static void main(String[] args) {
      Bsp2 app  = new Bsp2();
      double d = app.callCalc(13,5);
      System.out.println("d: "+d);
      pause();
    }

}
```

Quelltext 9 Quellcode callCalc (Beispiel2)

Die native Methode soll zwei double-Zahlen addieren und die Summe zurückgeben. Dazu wird sie als native-Methode definiert. Die dazugehörige Dll heisst dll2.dll. In der main-Methode wird ein Objekt der Klasse erzeugt und die Dll-Methode aufgerufen. Das Ergebnis wird ausgegeben.

5.2.1 Erzeugen der Headerdatei

Das Java-Programm wird mit folgender Anweisung übersetzt:
javac Bsp2.java

Mit dem Programm javah wird die Headerdatei erzeugt:
javah -jni Bsp2

Diese Headerdatei wird in das neue Verzeichnis für die zweite Dll kopiert.

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include "jni.h"
/* Header for class Bsp2 */

#ifndef _Included_Bsp2
#define _Included_Bsp2
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     Bsp2
 * Method:    callCalc
 * Signature: (DD)D
 */
JNIEXPORT jdouble JNICALL Java_Bsp2_callCalc  (JNIEnv *, jobject, jdouble, jdouble);

#ifdef __cplusplus
}
#endif
#endif
```

Quelltext 10 Headerdatei Beispiel2

Nun ist die Signatur deutlich anders. Die Methode hat zwei Parameter DD, also zwei Double-Zahlen. Als Ergebnis gibt Sie einen Double-Wert zurück.

```
// dll2.cpp : Definiert den Einsprungpunkt für die DLL-Anwendung.
//

#include "stdafx.h"
#include "Bsp2.h"

BOOL APIENTRY DllMain( HANDLE hModule, DWORD  ul_reason_for_call, LPVOID lpReserved )
{
   return TRUE;
}

JNIEXPORT jdouble JNICALL Java_Bsp2_callCalc (JNIEnv *, jobject , jdouble a, jdouble b)
{
    return a+b;
}
```

Quelltext 11 C-Quellcode Beispiel2

Mit diesem „Einstieg“ in die DLL, kann die weitere Berechnung auf andere Funktion verteilt werden.

5.3 Übergabe eines Strings an eine native Methode

Im nativen Code soll eine Methode aus Java aufgerufen werden. Als Parameter wird ein String übergeben.

5.3.1 Java Klasse erzeugen

```
import java.io.*;

public class Bsp3 {

   private static void pause() {
    try {
      System.in.read();
    }
    catch (IOException e) {
    }
  }  // pause


    // native method in the DLL
    public native void displayHelloHSHarz(String s);

    static {
        System.loadLibrary("dll3");
    }

    public static void main(String[] args) {
      Bsp3 app  = new Bsp3();
      app.displayHelloHSHarz ("hallo HS Harz, Beispiel3");
    }

}
```

Quelltext 12 Quellcode der DLL Bsp3

5.3.2 Erzeugen der Headerdatei

Die Headerdatei wird mit folgendem Befehlscode erzeugt:

javah –jni Bsp3

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class Bsp3 */
```

```
#ifndef _Included_Bsp3
#define _Included_Bsp3
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     Bsp3
 * Method:    displayHelloHSHarz
 * Signature: (Ljava/lang/String;)V
 */
JNIEXPORT void JNICALL Java_Bsp3_displayHelloHSHarz
  (JNIEnv *, jobject, jstring);

#ifdef __cplusplus
}
#endif
#endif
```

Quelltext 13 Anzeige der Headerdatei Example2.h

Die Signatur zeigt sehr deutlich die Package-Beziehung eines String in Java (Ljava/lang/String).

5.3.3 Erstellen der DLL

Folgender Quelltext muss erzeugt werden.

```
// dll3.cpp : Definiert den Einsprungpunkt für die DLL-Anwendung.
//

#include "stdafx.h"
#include "Bsp3.h"


BOOL APIENTRY DllMain( HANDLE hModule,  DWORD  ul_reason_for_call,  LPVOID lpReserved )
{
   return TRUE;
}

JNIEXPORT void JNICALL Java_Bsp3_displayHelloHSHarz
    (JNIEnv *env, jobject obj, jstring s) {
  // Transform the Java-String to C++ String
 const char* msg=env->GetStringUTFChars(s,0);
 puts(msg);
}
```

Quelltext 14 DLL Sourcecode des Bsp3

Hinweis:
Diese Datei wird unter dem Namen „dll3.dll“ abgespeichert. Aus dem Javaprogramm wird die Methode displayHelloHSHarz mit einem Parameter aufgerufen. Normale Standardtypen können einfach gekastet werden. Strings müssen mit Hilfe des JNIEnvironment „env“ umge-

wandelt werden. Ein wichtiger Punkt ist di eUmwandlung des Objektes „String“ in ein dynamisches Charakterfeld. Das geschieht mittels der Methode GetStringUTFChars.

5.3.4 Ausführen des Javaprogramms

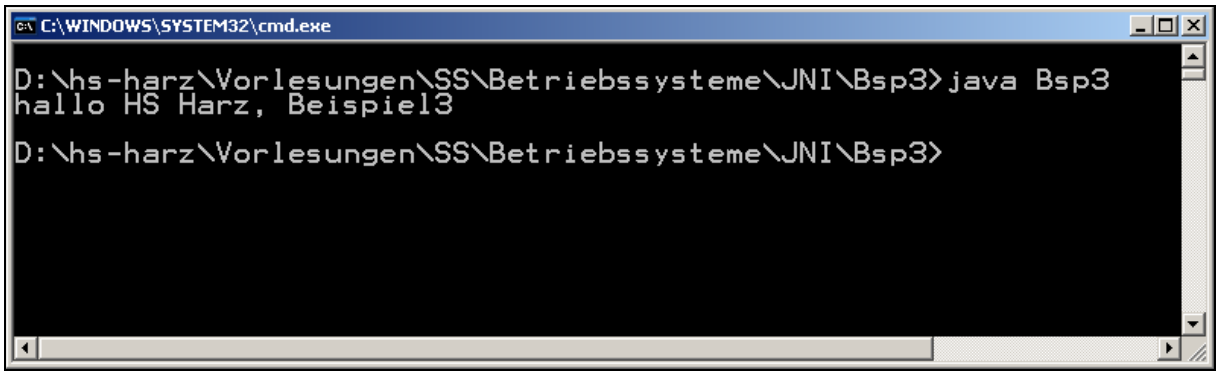


Abbildung 5 Aufruf des dritten Beispiels

5.4 Aufruf einer Java-Methode

Dieses Beispiel ruft aus dem nativen Code eine Javamethode des aufrufenden Objektes auf. Generell können mit diesem Verfahren alle Objekte bzw. Methoden erreicht werden.

Das Programm soll in der DLL eine Wertetabelle berechnen. Die Ausgabe erfolgt aber in java durch eine Ausgabemethode. Dabei werden die Daten nur als Text ausgegeben. Sinnvoller wäre zum Beispiel die Anzeige als Diagramm. Damit hätte man die komplexe Berechnung in C, die Anzeige aber plattformunabhängig in Java.

5.4.1 Java-Quellcode

```
import java.io.*;
import java.lang.System;

public class Bsp4 {

     // Aufruf der Startroutine
    public native void callWerteTabelle(double xa, double xsw, double xe);

    public void printXY(double x, double y) {
      System.err.println(" Java:  x:"+x+"   y:"+y);
   }

   public void start() {
     callWerteTabelle(1.0, 0.5, 10.0);
   }
```

```
   static {
      System.loadLibrary("dll4");
   }

   public static void main(String[] args) {
     Bsp4 app  = new Bsp4();
     app.start();
   }

}
```

Quelltext 15 Quellcode Bsp4

Die native Methode soll überträgt drei double Zahlen. Die erste ist der Anfangswert, die zweite ist die Schrittweite, der dritte Wert ist der Endwert. Diese C-Funktion berechnet damit dann eine Wertetabelle und ruft für die Anzeige eine Java-Methode auf. Die dazugehörige Dll heisst dll4.dll. In der Java main-Methode wird ein Objekt der Klasse erzeugt und die Dll-Methode aufgerufen. Das Ergebnis wird ausgegeben.

5.4.2 Erzeugen der Headerdatei

Das Java-Programm wird mit folgender Anweisung übersetzt:

```
javac  Bsp4.java
```

Mit dem Programm javah wird die Headerdatei erzeugt:

```
javah  -jni  Bsp4
```

Diese Headerdatei wird in das neue Verzeichnis für die vierte Dll kopiert.

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class Bsp4 */

#ifndef _Included_Bsp4
#define _Included_Bsp4
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     Bsp4
 * Method:    callWerteTabelle
 * Signature: (DDD)V
 */
JNIEXPORT void JNICALL Java_Bsp4_callWerteTabelle
  (JNIEnv *, jobject, jdouble, jdouble, jdouble);

#ifdef __cplusplus
}
#endif
#endif
```

Quelltext 16 Headerdatei Beispiel4

Die Methode hat drei Parameter DDD, also drei Double-Zahlen. Sie gibt kein Ergebnis zurück.

Die berechnetete Funktion lautet:

```
double fx(double x) {
    return x*x*x*x - 3.89*x*x*x*x  + 5.66*x*x*x  -3.6511*x + 0.8811;
}
```

Der C-Quellcode lautet folgendermaßen:

```
// dll4.cpp : Definiert den Einsprungpunkt für die DLL-Anwendung.
//

#include "stdafx.h"
#include "Bsp4.h"

BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                        )
{
   return TRUE;
}

double fx(double x) {
 return x*x*x*x - 3.89*x*x*x*x  + 5.66*x*x*x  -3.6511*x + 0.8811;
}

// Funktion erstellt eine Wertetabelle
// Anfangswert:  xa
// Schrittweite: xsw
// Endewert:     xe
JNIEXPORT void JNICALL Java_Bsp4_callWerteTabelle
(JNIEnv *env, jobject obj, jdouble xa, jdouble xsw, jdouble xe) {
    // Pointer to a java method
  jmethodID method_printXY;
    // Pointer to a java class
  jclass clss;
     double x, y;  // Hilfsvariablen
  puts("Anfangen der Berechnung in der DLL");
    // get the class
  clss = env->GetObjectClass(obj);
  method_printXY = env->GetMethodID(clss,"printXY", "(DD)V");
  x=xa;  // Anfangswert
     while (x<xe) {
          // Funktionswert holen
         y = fx(x);
          // Aufruf der Java-Methode
    env->CallVoidMethod(obj,method_printXY, x,y);
           // X-Wert erhöhen
         x+=xsw;
  };
  puts("Ende der Berechnung");

}
```

Quelltext 17 C-Quellcode Beispiel4

5.4.3 Aufruf des vierten Beispiels

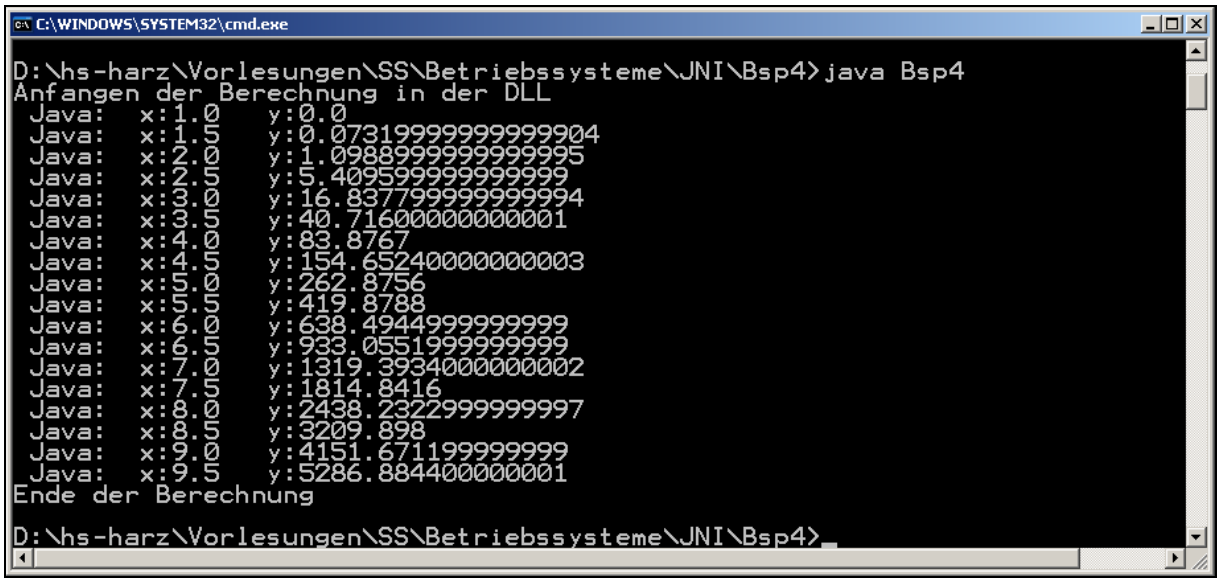


Abbildung 6 Aufruf des vierten Beispiels

Mit diesem Quellcode, ist es natülich leicht, die Daten in einen Editor zu schreiben oder in einem Graphen darzustellen (siehe Beispiel 7).

5.5 Aufruf einer C-Methode mit Übergabe eines Arrays

Das Beispiel zeigt, wie man ein Array an eine C-Methode übergeben kann.

```
import java.io.*;

public class Bsp5 {

  private static void pause() {
   try {
     System.in.read();
   }
   catch (IOException e) {
   }
 }  // pause


   private int [] array = new int[4];

   public native void displayArray(int[] a);

   public Bsp5 ()
   {
     for (int i=0; i<4; i++) {
        array[i] = i+22;
```

```
    }
  }

  static {
     System.loadLibrary("dll5");
  }

  public static void main(String[] args) {
    Bsp5 app  = new Bsp5();
    app.displayArray(app.array);
  }

}
```

Quelltext 18 Quellcode Übergabe eines Arrays (Beispiel5)

Die native Methode erhält ein Int-Array und zeigt die einzelnen Werte an.

5.5.1 Erzeugen der Headerdatei

Das Java-Programm wird mit folgender Anweisung übersetzt:
javac Bsp5.java

Mit dem Programm javah wird die Headerdatei erzeugt:
javah -jni Bsp5

Diese Headerdatei wird in das neue Verzeichnis für die fünfte Dll kopiert.

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include "jni.h"
/* Header for class Bsp5 */

#ifndef _Included_Bsp5
#define _Included_Bsp5
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     Bsp5
 * Method:    displayArray
 * Signature: ([I)V
 */
JNIEXPORT void JNICALL Java_Bsp5_displayArray
  (JNIEnv *, jobject, jintArray);

#ifdef __cplusplus
}
#endif
#endif
```

Quelltext 19 Headerdatei Beispiel5

Die Methode hat nun die Signatur [I, also ein Array von int-Zahlen. Mit Hilfe der env-Variablen kann man die Grenze bestimmen.

```
// dll5.cpp : Definiert den Einsprungpunkt für die DLL-Anwendung.
//

#include "stdafx.h"
#include "Bsp5.h"


BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                        )
{
   return TRUE;
}

JNIEXPORT void JNICALL Java_Bsp5_displayArray
(JNIEnv * env, jobject obj, jintArray intArray) {
  int i, k, n;
     // Bestimme die Länge des Feldes
  jsize len = env->GetArrayLength(intArray);
    //  Array-Objekt holen
  jint *intArray0 = env->GetIntArrayElements(intArray,0);
  n = (int) len;
  printf("size: %d\n",n);
  for (i=0; i<n; i++) {
     k =intArray0[i];
     printf("Array[ %d ] = %d\n",i,k);
  }
}
```

Quelltext 20 C-Quellcode Beispiel5

Als erstes muss man mit der Methode „GetArrayLength“ die Länge des Java-Arrays bestimmen. Danach erhält man mit der Methode „GetIntArrayElements“ einen C-Pointer, so dass man auf die einzelnen Werte zugreifen kann. Der Rest ist einfaches C.

5.5.2 Aufruf des fünften Beispiels

Die nächste Abbildung zeigt die einzelnen Werte des Arrays.

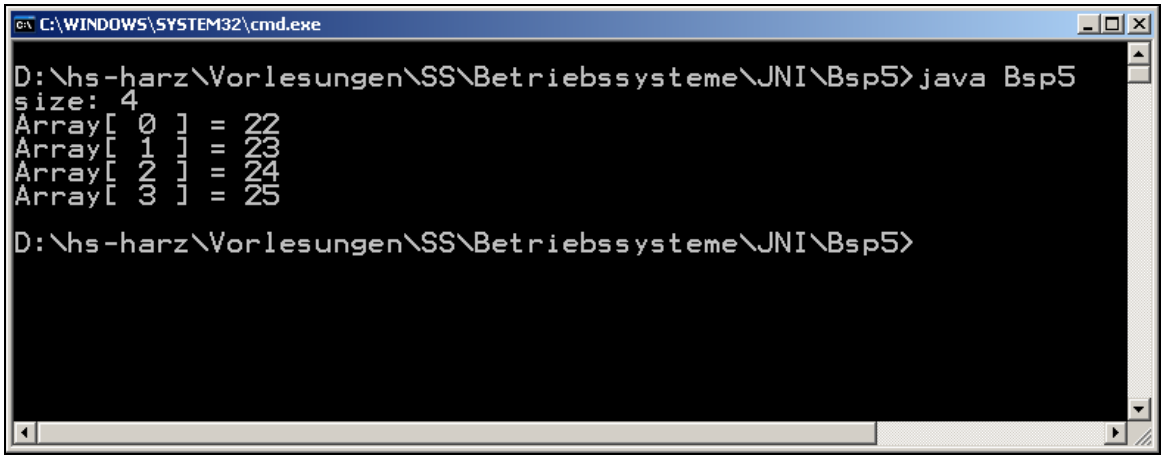


Abbildung 7 Aufruf des fünften Beispiels

5.6 Benchmarktest Primzahlbestimmung

Das Beispiel zeigt, inwieweit sich die Laufzeit eines Programms mittels Java und mittels nativen Code, C, unterscheidet. Als Beispiel wurde die Primzahlbestimmung gewählt.

```
import java.io.*;

public class Bsp6 {

   private static void pause() {
    try {
      System.in.read();
    }
    catch (IOException e) {
    }
  }  // pause


  private void callPrimzahlenJava(long max) {
       int i, j, n, k;
       double d;
       boolean bGefunden;
       for (i=2; i<max; i++) {
         d = i;
         n = (int) Math.sqrt(d);
         bGefunden=true;
         for (j=2; j<=n; j++) {
           // d = Math.sin(44.0);
           k=i / j;   // 10 div 2   ganzzahlig, oder modulo
           if ( (k*j) == i ) {
             // keine Primzahl
             bGefunden=false;
             break;
           }
         }
       } // for i
```

```
  } // callPrimzahlenJava

   public native void callPrimzahlenDLL(long max);

   static {
      System.loadLibrary("dll6");
   }

   public static void main(String[] args) {
     Bsp6 app  = new Bsp6();
      long t1, t2;
     long t3, t4;
     long n;
     n=50000000;
     System.out.println("n: "+n);
      t1 =  System.currentTimeMillis();
     app.callPrimzahlenJava(n);
      t2 =  System.currentTimeMillis();

      t3 =  System.currentTimeMillis();
     app.callPrimzahlenDLL(n);
      t4 =  System.currentTimeMillis();

     System.out.println("");
     System.out.println("");
     System.out.println("Zeit Java: "+(t2-t1) );
     System.out.println("Zeit    C: "+(t4-t3) );

     pause();
   }

}
```

Quelltext 21 Quellcode Benchmark (Beispiel6)

Die Primzahlberechnung ist relativ einfach gestaltet. Optimierungen sind natürlich gestattet. Die native Methode verzweigt auf eine Methode in der DLL, die den gleichen Quellcode der Java-Methode enthält.

5.6.1 Erzeugen der Headerdatei

Das Java-Programm wird mit folgender Anweisung übersetzt:

```
javac  Bsp6.java
```

Mit dem Programm javah wird die Headerdatei erzeugt:

```
javah  -jni  Bsp6
```

Diese Headerdatei wird in das neue Verzeichnis für die sechste Dll kopiert.

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class Bsp6 */

#ifndef _Included_Bsp6
#define _Included_Bsp6
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     Bsp6
 * Method:    callPrimzahlenDLL
 * Signature: (J)V
 */
JNIEXPORT void JNICALL Java_Bsp6_callPrimzahlenDLL  (JNIEnv *, jobject, jlong);
#ifdef __cplusplus
}
#endif
#endif
```

Quelltext 22 Headerdatei Beispiel6

Die Methode hat nun die Signatur (J)V. Sie hat also einen long-Parameter und gibt auch keine Wert zurück.

```
// dll6.cpp : Definiert den Einsprungpunkt für die DLL-Anwendung.
//

#include "stdafx.h"
#include "Bsp6.h"
#include <math.h>
#include <time.h>

BOOL APIENTRY DllMain( HANDLE hModule,  DWORD  ul_reason_for_call, LPVOID lpReserved)
{
   return TRUE;
}



JNIEXPORT void JNICALL Java_Bsp6_callPrimzahlenDLL (JNIEnv *env, jobject obj, jlong max) {
    int i, j, n, k;
    double d;
  BOOL bGefunden;

  for (i=2; i<max; i++) {
     d = i;
     n = (int) sqrt(d)+1;
     bGefunden=true;
     for (j=2; j<=n; j++) {
          // d = sin(44.0) // // damit wird es langsamer;
       k=i / j;   // 10 div 2   ganzzahlig
       if ( (k*j) == i ) {
         // keine Primzahl
         bGefunden=false;
         break;
       }
```

```
    }
    /*
        if (bGefunden) {
       printf("%u\n",i);
    }
        */
  } // for i

}
```

Quelltext 23 C-Quellcode Beispiel6 (Benchmark)

Die Bestimmung, ob eine Zahl n eine Primzahl ist, wird über eine innere Schleife ermittet. Diese muss aber nur bis zur Wurzel aus n laufen. Man dividiert die zu testende Primzahl durch die For-Schleifenzahl k. Ergibt das Produkt der ganzahlige Zahl mit der Variablen k wieder die Primzahl, so ist diese keine Primzahl.

5.6.2 Aufruf des sechsten Beispiels

Die nächste Tabelle zeigt die einzelnen Zeiten für verschiedene Werte von Max.

Anzahl	Java-Zeit [ms]	C-DLL Zeit [ms]	Faktor
1000	0	0	-
10 000	20	0	-
100 000	260	60	4,33
1 000 000	4586	1282	3,577
10 000 000	85989 00:01:25	26188 26,188 s	3,28
50 000 000	982293 00:16:22	367057 00:06:07	2,676
100 000 000	2653796 00:44:13	1039375 00:17:19	2,55

Abbildung 8 Ergebnisse des Benchmark-Tests

Die obige Tabelle zeigt die Laufzeiten auf einem Intel-Notebook mit 1,7 GHz berechnet. Die DLL muss aber im Release-Modus übersetzt werden, da sonst Debug-Informationen die Laufzeit beeinträchtigen. Die einfachen Operationen zeigen einen Laufzeitsfaktor von ca. 3. Mit größeren Operationen, wie sinus, ist ein höherer Faktor möglich. Die Zeiten sind im Format HH:MM:SS angegeben.

5.7 Wertetabelle mit einem Editor in einem Frame

Dieses Beispiel verfeinert das vierte Beispiel. Das Programm soll in der DLL eine Wertetabelle berechnen. Die Ausgabe erfolgt aber in java durch eine Ausgabemethode. Diesmal aber werden die Daten in einem Editor gespeichert.

5.7.1 Java-Quellcode

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;


public class Bsp7 extends JFrame {

 JTextArea _editor = new JTextArea("");

 public Bsp7() {
  setSize(600, 600);
  setTitle("JNI-Beispiel 7 (4) mit Editor");
  setGUI();
 }

 public void setGUI() {
  JButton BnOk, BnEsc;

  this.getContentPane().setLayout ( new BorderLayout(10,20) );  // vgap, hgap


  JScrollPane sc = new JScrollPane(_editor);
  this.getContentPane().add(sc, BorderLayout.CENTER);


  JPanel panelBn = new JPanel();
  panelBn.setLayout ( new FlowLayout() );
  BnOk = new JButton("Starten");  // Label erzeugen
  panelBn.add(BnOk);
  BnEsc = new JButton("Abbruch");  // Label erzeugen
  panelBn.add(BnEsc);

  this.getContentPane().add(panelBn, BorderLayout.SOUTH);

  BnOk.addActionListener(new java.awt.event.ActionListener() {

    public void actionPerformed(ActionEvent e) {
      BnOk_Click();
    }
  });
  BnEsc.addActionListener(new java.awt.event.ActionListener() {

    public void actionPerformed(ActionEvent e) {
      BnEsc_Click();
    }
  });

 } // setGUI

 void BnOk_Click() {
   callWerteTabelle(1.0, 0.1, 10.0);
 }
 void BnEsc_Click() {
   System.exit(0);
 }
```

```
    // Aufruf der Startroutine
   public native void callWerteTabelle(double xa, double xsw, double xe);

   public void printXY(double x, double y) {
     String sStr =" Java:  x:"+x+"\ty:"+y;
     _editor.append(sStr+"\n");
     System.err.println(sStr);
   }

   static {
       System.loadLibrary("dll7");
   }


   public static void main(String[] args) {
     Bsp7 frame  = new Bsp7();
     frame.setVisible(true);

   }

  //Überschreiben, damit das Programm bei Herunterfahren des Systems beendet werden kann
  protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if(e.getID() == WindowEvent.WINDOW_CLOSING) {
      System.exit(0);
    }
  }


}
```

Quelltext 24 Quellcode Bsp7

Die native Methode soll überträgt drei double Zahlen. Die erste ist der Anfangswert, die zweite ist die Schrittweite, der dritte Wert ist der Endwert. Diese C-Funktion berechnet damit dann eine Wertetabelle und ruft für die Anzeige eine Java-Methode auf. Die dazugehörige Dll heisst dll7.dll. In der Java main-Methode wird ein Objekt der Klasse erzeugt und die Dll-Methode aufgerufen. Das Ergebnis wird ausgegeben.

5.7.2 Erzeugen der Headerdatei

Das Java-Programm wird mit folgender Anweisung übersetzt:

```
javac  Bsp7.java
```

Mit dem Programm javah wird die Headerdatei erzeugt:

```
javah  -jni  Bsp7
```

Diese Headerdatei wird in das neue Verzeichnis für die vierte Dll kopiert.

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
```

```
/* Header for class Bsp7 */

#ifndef _Included_Bsp7
#define _Included_Bsp7
#ifdef __cplusplus
extern "C" {

#endif
/* Inaccessible static: LOCK */
#undef Bsp7_assert
#define Bsp7_assert 0L
/* Inaccessible static: isInc */
/* Inaccessible static: incRate */
#undef Bsp7_TOP_ALIGNMENT
#define Bsp7_TOP_ALIGNMENT 0.0f
#undef Bsp7_CENTER_ALIGNMENT
#define Bsp7_CENTER_ALIGNMENT 0.5f
#undef Bsp7_BOTTOM_ALIGNMENT
#define Bsp7_BOTTOM_ALIGNMENT 1.0f
#undef Bsp7_LEFT_ALIGNMENT
#define Bsp7_LEFT_ALIGNMENT 0.0f
#undef Bsp7_RIGHT_ALIGNMENT
#define Bsp7_RIGHT_ALIGNMENT 1.0f
#undef Bsp7_serialVersionUID
#define Bsp7_serialVersionUID -7644114512714619750LL
#undef Bsp7_serialVersionUID
#define Bsp7_serialVersionUID 4613797578919906343LL
#undef Bsp7_OPENED
#define Bsp7_OPENED 1L
/* Inaccessible static: nameCounter */
#undef Bsp7_serialVersionUID
#define Bsp7_serialVersionUID 4497834738069338734LL
#undef Bsp7_DEFAULT_CURSOR
#define Bsp7_DEFAULT_CURSOR 0L
#undef Bsp7_CROSSHAIR_CURSOR
#define Bsp7_CROSSHAIR_CURSOR 1L
#undef Bsp7_TEXT_CURSOR
#define Bsp7_TEXT_CURSOR 2L
#undef Bsp7_WAIT_CURSOR
#define Bsp7_WAIT_CURSOR 3L
#undef Bsp7_SW_RESIZE_CURSOR
#define Bsp7_SW_RESIZE_CURSOR 4L
#undef Bsp7_SE_RESIZE_CURSOR
#define Bsp7_SE_RESIZE_CURSOR 5L
#undef Bsp7_NW_RESIZE_CURSOR
#define Bsp7_NW_RESIZE_CURSOR 6L
#undef Bsp7_NE_RESIZE_CURSOR
#define Bsp7_NE_RESIZE_CURSOR 7L
#undef Bsp7_N_RESIZE_CURSOR
#define Bsp7_N_RESIZE_CURSOR 8L
#undef Bsp7_S_RESIZE_CURSOR
#define Bsp7_S_RESIZE_CURSOR 9L
#undef Bsp7_W_RESIZE_CURSOR
#define Bsp7_W_RESIZE_CURSOR 10L
#undef Bsp7_E_RESIZE_CURSOR
#define Bsp7_E_RESIZE_CURSOR 11L
#undef Bsp7_HAND_CURSOR
#define Bsp7_HAND_CURSOR 12L
#undef Bsp7_MOVE_CURSOR
```

```
#define Bsp7_MOVE_CURSOR 13L
#undef Bsp7_NORMAL
#define Bsp7_NORMAL 0L
#undef Bsp7_ICONIFIED
#define Bsp7_ICONIFIED 1L
/* Inaccessible static: nameCounter */
#undef Bsp7_serialVersionUID
#define Bsp7_serialVersionUID 2673458971256075116LL
/* Inaccessible static: class_00024java_00024awt_00024Frame */
/*
 * Class:     Bsp7
 * Method:    callWerteTabelle
 * Signature: (DDD)V
 */
JNIEXPORT void JNICALL Java_Bsp7_callWerteTabelle
  (JNIEnv *, jobject, jdouble, jdouble, jdouble);

#ifdef __cplusplus
}
#endif
#endif
```

Quelltext 25 Headerdatei Beispiel7

Durch das JFrame werden mehrere Konstanten definiert. Diese können aber auskommentiert werden. Der unterer Block zeigt die „normale“ Headerdatei. Die Methode hat drei Parameter DDD, also drei Double-Zahlen. Sie gibt kein Ergebnis zurück.

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class Bsp7 */

#ifndef _Included_Bsp7
#define _Included_Bsp7
#ifdef __cplusplus
extern "C" {

#endif

/* Inaccessible static: class_00024java_00024awt_00024Frame */
/*
 * Class:     Bsp7
 * Method:    callWerteTabelle
 * Signature: (DDD)V
 */
JNIEXPORT void JNICALL Java_Bsp7_callWerteTabelle
  (JNIEnv *, jobject, jdouble, jdouble, jdouble);

#ifdef __cplusplus
}
#endif
#endif
```

Die berechnetete Funktion lautet:

```
double fx(double x) {
    return x*x*x*x - 3.89*x*x*x*x  + 5.66*x*x*x  -3.6511*x + 0.8811;
}
```

Der C-Quellcode lautet folgendermaßen:

```
// dll7.cpp : Definiert den Einsprungpunkt für die DLL-Anwendung.
//

#include "stdafx.h"

#include "Bsp7.h"

BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                        )
{
   return TRUE;
}

double fx(double x) {
 return x*x*x*x - 3.89*x*x*x*x  + 5.66*x*x*x  -3.6511*x + 0.8811;
}

// Funktion erstellt eine Wertetabelle
// Anfangswert:  xa
// Schrittweite: xsw
// Endewert:     xe
JNIEXPORT void JNICALL Java_Bsp7_callWerteTabelle
(JNIEnv *env, jobject obj, jdouble xa, jdouble xsw, jdouble xe) {
    // Pointer to a java method
  jmethodID method_printXY;
    // Pointer to a java class
  jclass clss;
    double x, y;  // Hilfsvariablen
  puts("Anfangen der Berechnung in der DLL");
    // get the class
  clss = env->GetObjectClass(obj);
  method_printXY = env->GetMethodID(clss,"printXY", "(DD)V");
  x=xa;  // Anfangswert
    while (x<xe) {
         // Funktionswert holen
        y = fx(x);
         // Aufruf der Java-Methode
        env->CallVoidMethod(obj,method_printXY, x,y);
          // X-Wert erhöhen
        x+=xsw;
  };
  puts("Ende der Berechnung");

}
```

Quelltext 26 C-Quellcode Beispiel7 ist (fast) identisch mit Beispiel4

5.7.3 Aufruf des vierten Beispiels

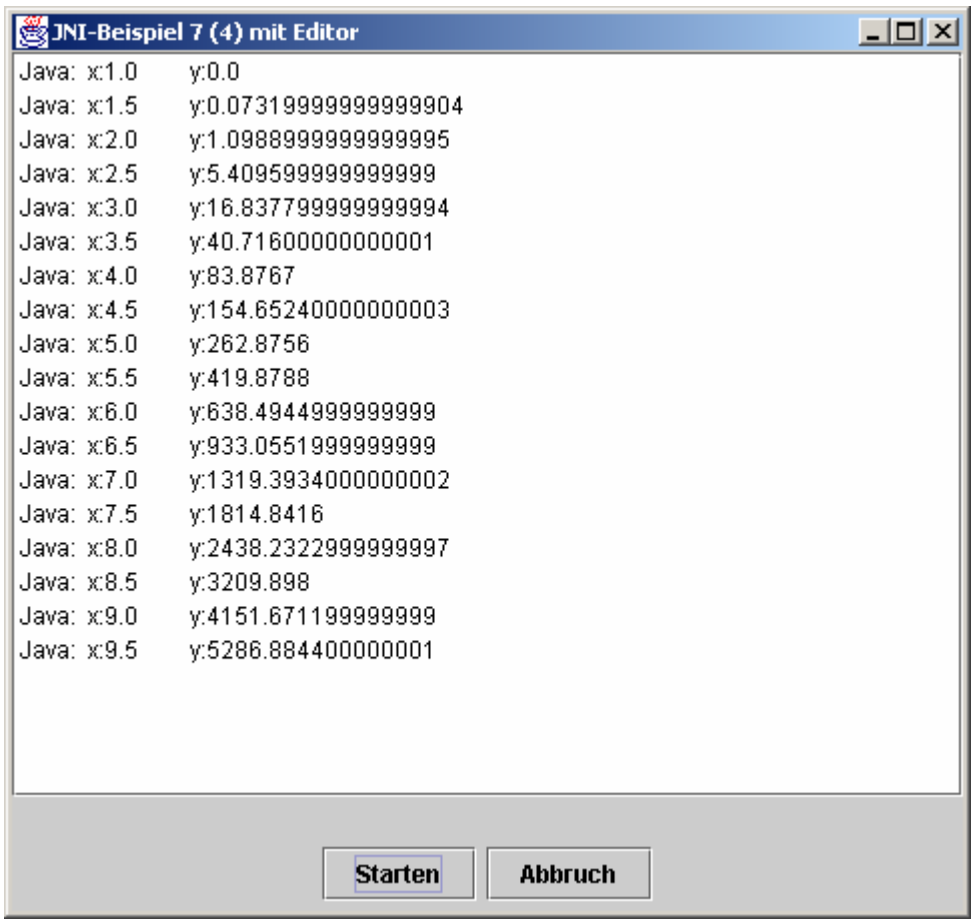


Abbildung 9 Aufruf des siebten Beispiels

Als Erweiteurng kann man natürlich die drei double Werte eingeben und eventuell die Funktion per Radiobutton auswählen.

6 Literatur

6.1 Allgemeine Literatur

[1] Sheng Liang
The Java Native Interface : Programmer's Guide and Specification
Java Series

[2] Rob Gordon, Robert Gordon, Alan McClellan
Java Native Interface (Essential Java)

[3] Tim Lindholm, Frank Yellin
The Java Virtual Machine Specification(2nd Ed)

[4] Doug Lea
Concurrent Programming in Java: Design Principles and Patterns

[5] Java Native Interface: Programmer's Guide and Specification
Addison-Wesley Java Series book

[6] Bruce Eckel
Thinking in Java

[7] The Java Tutorial
JavaSoft

[8] Java.www.jguru.com

7 Indexverzeichnis