Neuerungen im JDK 1.4 :
Die Java Logging API und Java Web Start

Projektarbeit SommerSemester 2002 von Derk Wesemann

Diese Projektarbeit dient dazu, einen kurzen Überblick über neue Bestandteile im aktuellen Java Development Kit 1.4 zu geben. Im ersten Teil werden die Möglichkeiten der neuen Java Logging API besprochen und deren Umsetzung anhand eines einfachen Beispiels erläutert. Der zweite Abschnitt behandelt die Funktionsweise des Produktes Java Web Start und beschreibt mögliche Anwendungsfälle.

Inhalt :

Die Java Logging API

Das Logging an sich dient der Ablaufdokumentation einer Anwendung. Diese Dokumentation bietet bei eventuell auftretenden Problemen oder Fehlern dann eine einfache Möglichkeit zur Problemfindung und Problembehebung, speziell in Situationen wo kein Debugger verwendet werden kann. Bisher mussten diese Mechanismen jedoch aufwendig in den jeweiligen Java-Anwendungen implementiert werden, als einzige einfache Möglichkeit bot sich das häufig verwendete „System.out.println“ an.

An dieser Stelle setzt nun die neue Logging API an und ermöglicht es Entwicklern, mit verhältnismäßig geringem Aufwand Log-Nachrichten in der jeweilig gewünschten Form zu erzeugen.Grundsätzlich besteht die Logging API (java.util.logging) aus den folgenden Klassen:

Allgemein:
- ErrorManager
- Level
- Logger
- LoggingPermission
- LogManager
- LogRecord

Handler:
- ConsoleHandler
- FileHandler
- Handler
- MemoryHandler
- SocketHandler
- StreamHandler

Formatter:
- Formatter
- SimpleFormatter
- XMLFormatter

Interfaces:
- Filter

Der Rahmen dieser Projektarbeit lässt es leider nicht zu, sich mit allen Klassen auseinander zu setzen. Im Folgenden werden daher nur diejenigen behandelt, die zum Erstellen von Log-Nachrichten unabdingbar sind und am häufigsten verwendet werden.

Die Java Logging API

Die am meisten verwendete Komponente der Logging API ist das sogenannte Logger-Objekt. Es lässt sich am schnellsten für die meisten Projekte adaptieren und erzeugt aus der ihm übergebenen Textzeile ein LogRecord-Objekt. Dieses Objekt kann dann einem der oben aufgeführten Handler übergeben werden, welche dann das LogRecord-Objekt entsprechend ihrer Funktion darstellen, im Falle des FileHandlers z.B. in Form einer Datei. Das Aussehen der Datei, bzw. ihre Formatierung ist vom gewählten Handler und Formatter abhängig. Der Standard-Formatter für den FileHandler ist beispielsweise der XML-Formatter, das bedeutet die LogRecords werden in XML-Tags ausgegeben. Eine andere Variante wäre der SimpleFormatter, der eine zusammenhängende Textausgabe bewirkt. Das folgende Diagramm veranschaulicht den Prozess des so beschriebenen Loggings:

Jedes LogRecord-Objekt hat einen dazugehörigen Level, welcher die Wichtigkeit des Objektes beschreibt und darüber entscheidet, ob das Objekt ausgegeben wird oder nicht. Die folgenden Level sind durch die Logging API vorgegeben:

severe (als höchste Priorität)
warning
info
config
fine
finer
finest (als niedrigste Priorität)

Zusätzlich gibt es einen definierten Level „all“, welcher das Loggen aller Objekte zulässt. Sollte dem Logger-Objekt kein Level zugewiesen werden, gilt als Grundeinstellung der Level „info“, das heißt, alle LogRecord-Objekte welche einen geringeren Level aufweisen, werden nicht an den Handler weitergegeben. Der Sinn dieser Level besteht darin, den Verarbeitungsaufwand für das Loggen einer Anwendung den jeweiligen Gegebenheiten anzupassen. So kann z.B. ein Logger-Objekt während der Entwicklung eines Programms hochauflösende Log-Einträge zur Überprüfung des Programms erstellen, erzeugt jedoch während der späteren Anwendung nur noch einen Bruchteil davon. Dies kommt natürlich der Geschwindigkeit des Programms zugute und lässt gleichzeitig die Möglichkeit zur späteren ausführlichen Diagnose, durch einfaches Setzen einer Variablen.

Zur besserem Veranschaulichung des eben Erklärten bietet sich das folgende Beispielprogramm an. Es beinhaltet eine einfache Client/Server-Anwendung. Über den Client wird eine Nachricht in Form eines Strings erstellt und per TCP/IP-Protokoll an den Server versendet. Dieser liest die Nachricht, stellt sie auf dem Bildschirm dar und schreibt Inhalt und Absender der Nachricht in eine Log-Datei zur späteren Ansicht.

Quelltext für Klasse SimpleServer

```
// SimpleServer - einfaches Server-Testprogramm für die Java Logging API

// Importieren der notwendigen Bibliotheken
import java.net.*;
import java.io.*;
import java.util.logging.*;

public class SimpleServer
{ // Variablen

// Für den Client:
      public static Socket mySocket; // Der Client-Sockel
      public static InetAddress ClientIpNumber; // Die Client-IP-Adresse
      public static String ClientAddress; // Der Name des Clients
      public static int ClientPort; // Die Port-Nummer des Clients

// Für den Server
      public static ServerSocket myServerSocket; // Der Server-Sockel
      public static InetAddress ServerIpNumber; // Die server-IP-Adresse
      public static String ServerAddress; // Der Name des Servers
      public static int ServerPort; // Die Port-Nummer des servers

// Der Datenstrom vom Client
      private DataInputStream streamIn;


// Methoden

public SimpleServer(int port)
{ try
      { // Bestimmung der eigenen Host-Adresse
      ServerIpNumber = ServerIpNumber.getLocalHost();
      ServerAddress = ServerIpNumber.getHostName();

      System.out.println(„Binding to port „ + port + „, please wait ...“);

// Speichern der Server-Port-Nummer
      ServerPort = port;

// Ein Server-Sockel-Objekt auf dem oben bestimmten Host aufsetzen
      myServerSocket = new ServerSocket(port, 20, ServerIpNumber);

      System.out.println(„Server started: „ +ServerAddress+ „ , port : „
                        +ServerPort);
      System.out.println(„Waiting for a client ...“);

// Anbindung an Client ermöglichen
      mySocket = myServerSocket.accept();
```

Quelltext für Klasse SimpleServer
[Forts.]

```
// Bestimmung des Namens, des Ports und der Adresse des Clients
      ClientIpNumber = mySocket.getInetAddress();
      ClientAddress = ClientIpNumber.getHostName();
      ClientPort = mySocket.getLocalPort();

      System.out.println(„Client accepted: „ +ClientAddress+ „ , port : „
                        +ClientPort);

      open(); // Aufruf der Methode „open“, siehe unten

// Erzeugung eines Logger-Objektes aus der Klasse logHandler
      LogHandler.newLogger();

// Erstellung von Log-Nachrichten auf dem Level „INFO“ :
      LogHandler.myLogger.info(„ Logrecord generated by class
                               SimpleServer.java“);
      LogHandler.myLogger.info(„ origin = „ +SimpleServer.ClientAddress+ „,
                               port = „ +SimpleServer.ClientPort);

// Einlesen des Datenstroms vom Client bis die Textzeile das Wort „exit“ ent-
hält
      boolean done = false;
      while (!done)
      { try
            { // Einlesen des Datenstroms im unicode-Format
            String line = streamIn.readUTF();

            // Ausgabe der Textzeile auf die Konsole
            System.out.println(line);

            // Weitergabe der Textzeile an die Klasse LogHandler als
               LogRecord-Objekt
            LogHandler.myLogger.finer(„message from client „ +ClientAddress+ „
                                      : „ +line);

            // Abbruch, wenn die Textzeile „exit“ enthält
            done = line.equals(„exit“);
            }
            // Falls Fehler beim Einlesen aufgetreten ist
            catch(IOException ioe)
            { done = true;
            }

      }
```

Quelltext für Klasse SimpleServer
[Forts.]

```
// Ausgabe weiterer Log-Nachrichten auf Level „FINE“ :
      LogHandler.myLogger.fine(„ This log message was sponsored by Sun
                                Microsystems Inc.“);
// Aufruf der Methode writeLogFile, siehe Klasse LogHandler
      LogHandler.writeLogFile();
      close(); // Aufruf der Methode „close“, siehe unten
      }
      catch(IOException ioe)
      { System.out.println(ioe);
      }
}

public void open() throws IOException
{
// Erzeugung des Datenstrom-Objektes, welches die Textzeile vom Client ein
   liest
      streamIn = new
      DataInputStream(newBufferedInputStream(mySocket.getInputStream()));
}

public void close() throws IOException
{
      // Client-Sockel schliessen
      mySocket.close();

      // Eingehenden Datenstrom schliessen
      streamIn.close();
}

public static void main(String args[])
{
      // Initialisierung der ServerSocket-Variablen
      SimpleServer myServerSocket = null;

      // Abbruch, falls unpassende Anzahl an Argumenten eingegeben wird
      if (args.length != 1)
            System.out.println(„Usage: java SimpleServer port“);
      else
      // Erzeugung des ServerSocket-Objektes mit dem zu verwendenden Port als
         Argument
      myServerSocket = new SimpleServer(Integer.parseInt(args[0]));
      }
}
```

Quelltext für Klasse LogHandler

```
// LogHandler - Aufruf von Logger und Handler für die Klasse SimpleServer

// Importieren der erforderlichen Bibliotheken
import java.util.logging.*;
import java.io.*;
import java.lang.*;

public class LogHandler
{
// Variablen

      // Das Logger-Objekt
      public static Logger myLogger;

      // Datei- und Speicher-Handler für den Logger
      public static FileHandler myFileHandler;
      public static MemoryHandler myMemoryHandler;


// Methoden

// Zur Initialisierung der logger und Handler:
      public static void newLogger() throws IOException
      {
      // Erzeugung eines Logger-Objekts mit einem bestimmten Namen
      myLogger = Logger.getLogger(„SimpleServer.LogHandler“);

      // Erzeugung eines FileHandler-Objekts, welches die beschriebene Datei
         anlegt
      myFileHandler = new FileHandler(„c:/logrecords/log.htm“);

      // Erzeugung eines MemoryHandler-Objekts.
      // Bis zu 50 LogRecord-Objekte werden zwischengespeichert,
      // alle LogRecords, deren Level grösser als „finer“ ist
      // werden dem FileHandler übergeben
      myMemoryHandler = new MemoryHandler(myFileHandler, 50, Level.FINER);

      // MemoryHandler mit dem Logger verknüpfen
      myLogger.addHandler(myMemoryHandler);

}
```

Quelltext für Klasse LogHandler
[Forts.]

```
// Zur Initialisierung der Level und zum Schreiben der Datei:
public static void writeLogFile() throws IOException
{
      // Logging-Level für das Logger-Objekt setzen
      myLogger.setLevel(Level.FINER);

      // Schreiben des Speicherinhaltes und Schliessen der Handler
      try
      { myMemoryHandler.flush();
      myMemoryHandler.close();
      myFileHandler.close();
      }
// Falls keine Schreibrechte vorhanden sind um die Datei zu schliessen
catch (SecurityException se)
{ System.out.println(se);
```

Quelltext für Klasse SimpleClient

```
// SimpleClient - einfache Client-Anwendung zum Versenden von Textzeilen an
einen Server

// Importieren der erforderlichen Bibliotheken
import java.net.*;
import java.io.*;

public class SimpleClient
{ // Variablen

      // Sockel des Servers, der erreicht werden soll (IP-Nummer und Port)
      private Socket mySocket;

      // Datenstrom von der Tastatur
      private DataInputStream streamIn;

      // Datenstrom zum Server
      private DataOutputStream streamOut;

      // Zu übertragende Textzeile
      public String line = „“;
```

Quelltext für Klasse SimpleClient
[Forts.]

```
// Methoden

public SimpleClient(String serverName, int serverPort)
{ System.out.println(„Establishing connection. Please wait ...“);
      try
      { // Verbindungsaufbau zum Server „serverName“, port „serverPort“
      mySocket = new Socket(serverName, serverPort);
      System.out.println(„Connected: „ + mySocket);
      start(); // Aufruf der Methode „start“, siehe unten
      }
      //Falls Name oder IP-Nummer nicht erreichbar sind
      catch(UnknownHostException uhe)
      { System.out.println(„Host unknown: „ + uhe.getMessage());
      }
      catch(IOException ioe)
      { System.out.println(„Unexpected exception: „ + ioe.getMessage());
      }

      // „exit“ in der Textzeile bewirkt Abbruch
      while (!line.equals(„exit“))
      { try
      { // Einlesen der Tastaturzeile
      line = streamIn.readLine();

      // Schreiben der Tastaturzeile in den DataOutputStream (unicode-Format)
      streamOut.writeUTF(line);

      // Auslesen des DataOutputStreams an den Server
      streamOut.flush();

      }
      // Falls Fehler bei Lese-/Schreibvorgang
      catch(IOException ioe)
      { System.out.println(„Sending error: „ + ioe.getMessage());
      }
      }
stop(); //Aufruf der Methode „stop“, siehe unten
}

public void start() throws IOException
{ // Initialisierung der Variablen für Input- und OutputStreams

      // Zuordnung des InputStreams zur Konsole (System.in)
      streamIn = new DataInputStream(System.in);

      // Zuordnung des OutputStreams zum Server (Name und Port)
      streamOut = new DataOutputStream(mySocket.getOutputStream());
}
```

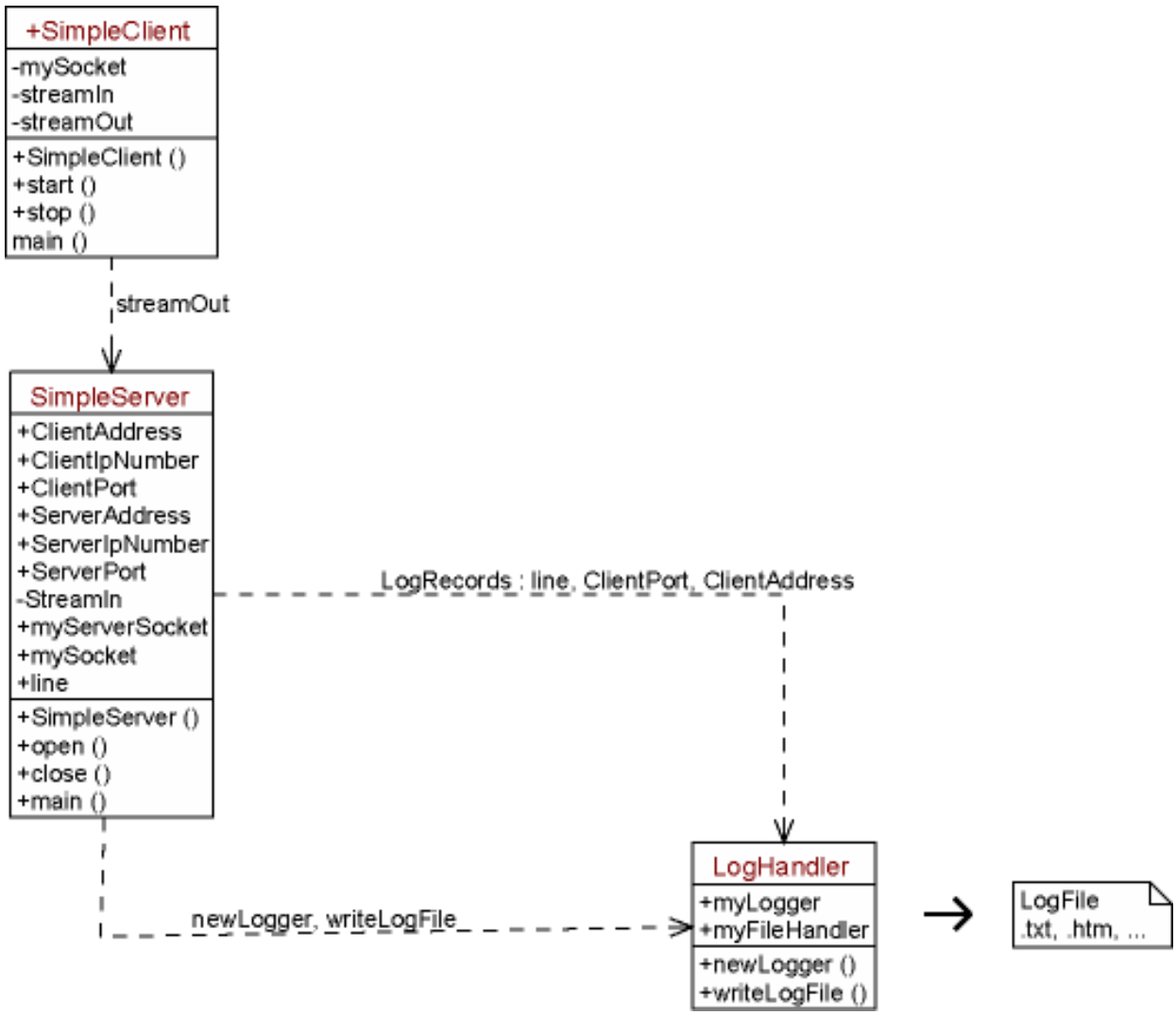
Quelltext für Klasse SimpleClient
[Forts.]

```
public void stop()
      { try
      { // Schliessen der Streams und des Sockets
      streamIn.close();
      streamOut.close();
      mySocket.close();
      }
      // Falls ein Stream oder der Socket nicht geschlossen wurde
      catch(IOException ioe)
      { System.out.println(„Error closing ...“);
      }
}

public static void main(String args[])
{ // Erzeugung des SimpleClient-Variablen, Initialwert null
      SimpleClient client = null;

      // Falls die erforderlichen Variablen nicht eingegeben wurden
      if (args.length != 2)
            System.out.println(„Usage: java SimpleClient host port“);
      else
      // Initialisierung des SimpleClient-Objektes mit den Parametern
         serverName, serverPort
      client = new SimpleClient(args[0], Integer.parseInt(args[1]));
}
}
```

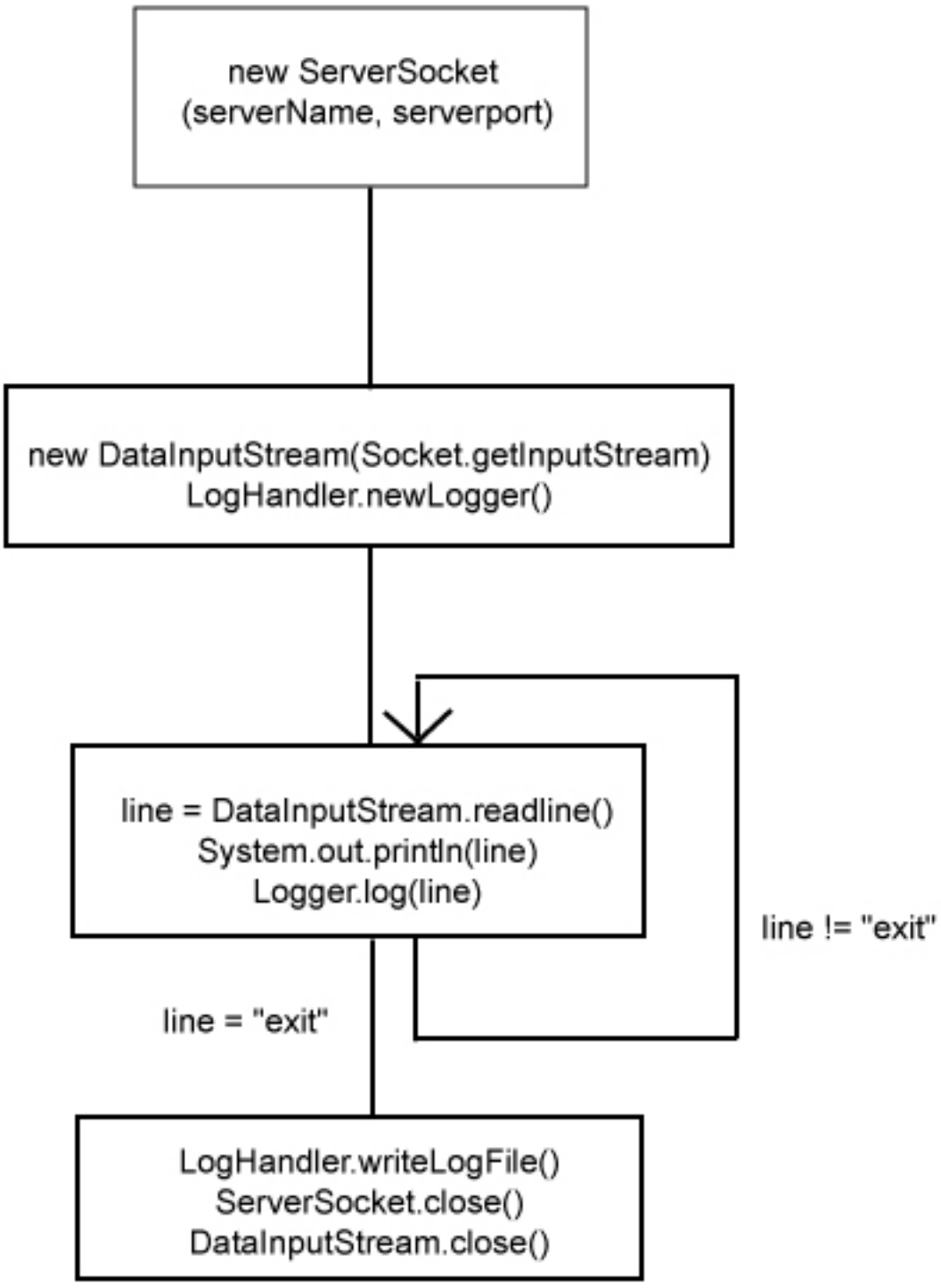
Aussehen der erstellten Log-Datei

```
<?xml version="1.0" encoding="windows-1252" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
<record>
      <date>2002-05-27T14:26:10</date>    <millis>1022502370440</millis>
      <sequence>0</sequence>
      <logger>SimpleServer.LogHandler</logger>  <level>INFO</level>
      <class>SimpleServer</class>   <method>&lt;init&gt;</method>
      <thread>10</thread>
      <message> Logrecord generated by class SimpleServer.java</message>
</record>
<record>
      <date>2002-05-27T14:26:10</date>    <millis>1022502370990</millis>“
      <sequence>1</sequence>
      <logger>SimpleServer.LogHandler</logger>  <level>INFO</level>
      <class>SimpleServer</class>   <method>&lt;init&gt;</method>
      <thread>10</thread>
      <message> origin  = MURPHY, port = 5100</message>
</record>
<record>
      <date>2002-05-27T14:26:14</date>    <millis>1022502374340</millis>
      <sequence>2</sequence>
      <logger>SimpleServer.LogHandler</logger>  <level>FINER</level>
      <class>SimpleServer</class>   <method>&lt;init&gt;</method>
      <thread>10</thread>
      <message>message from client MURPHY : Nachricht1</message>
</record>
<record>
      <date>2002-05-27T14:26:17</date>    <millis>1022502377690</millis>
      <sequence>3</sequence>
      <logger>SimpleServer.LogHandler</logger>  <level>FINER</level>
      <class>SimpleServer</class>   <method>&lt;init&gt;</method>
      <thread>10</thread>
      <message>message from client MURPHY : Nachricht2</message>
</record>
<record>
      <date>2002-05-27T14:26:19</date>    <millis>1022502379670</millis>
      <sequence>4</sequence>
      <logger>SimpleServer.LogHandler</logger>  <level>FINER</level>
      <class>SimpleServer</class>   <method>&lt;init&gt;</method>
      <thread>10</thread>
      <message>message from client MURPHY : exit</message>
</record>
<record>
      <date>2002-05-27T14:26:19</date>    <millis>1022502379720</millis>
      <sequence>5</sequence>
      <logger>SimpleServer.LogHandler</logger>  <level>FINE</level>
      <class>SimpleServer</class>   <method>&lt;init&gt;</method>
      <thread>10</thread>
      <message> This log message was sponsored by Sun Microsystems Inc.
      </message>
</record>
</log>
```

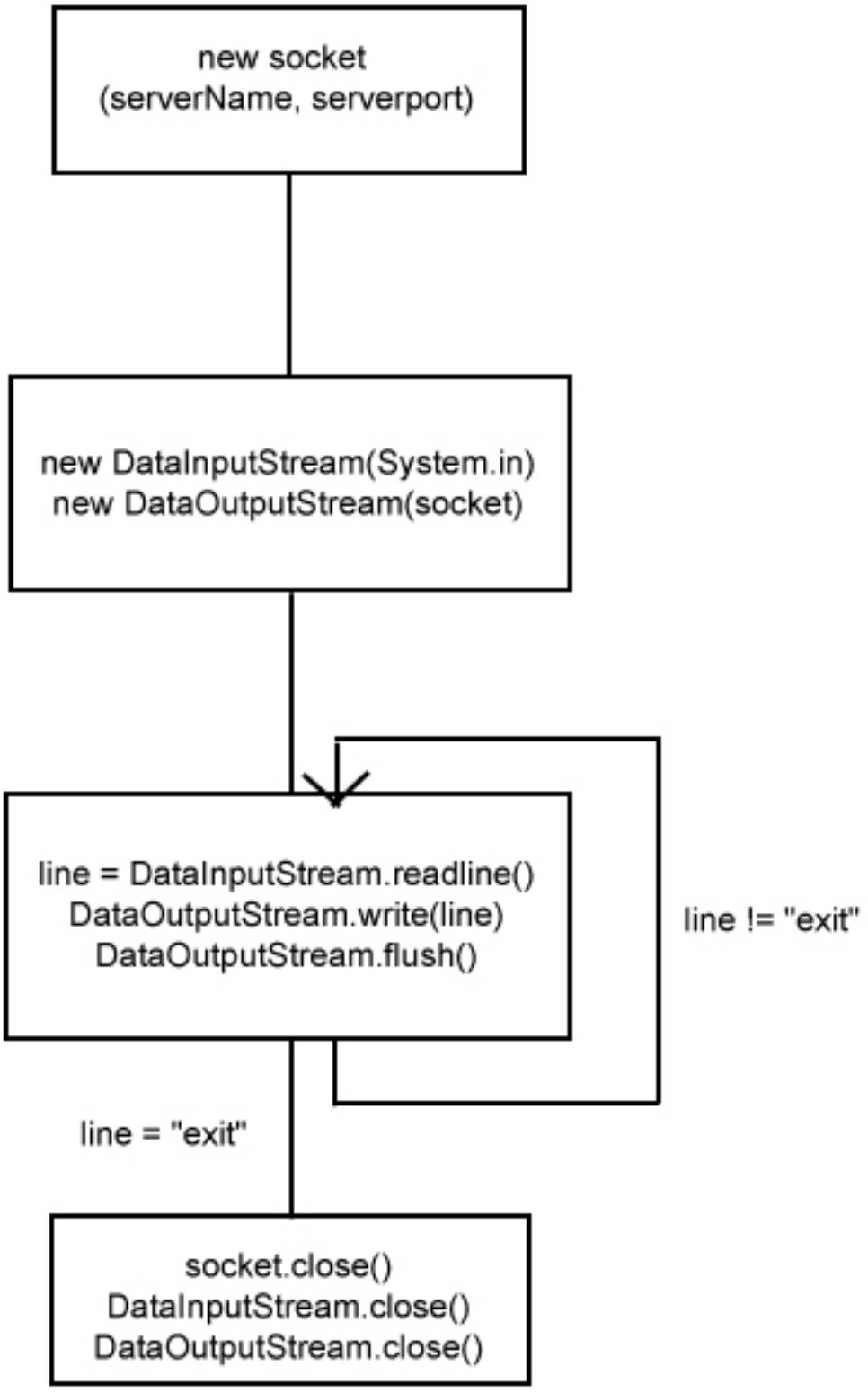
Klassendiagramm für SimpleServer und SimpleClient



Sequenzdiagramm für Klasse SimpleServer



Sequenzdiagramm für Klasse SimpleClient



Das Produkt Java Web Start

Java Web Start ist ein fester Bestandteil des neuen Java Development Kits 1.4. Der Sinn und Zweck von Java Web Start besteht darin, Anwendungen über ein Netzwerk zur Verfügung zu stellen, ohne daß diese auf dem jeweiligen Client-Rechner installiert werden müssen. In dieser Hinsicht kann das Produkt als eine Art Erweiterung zu den bisher verwendeten Java-Applets verstanden werden. Java Web Start ist eine Applikation und als solche auf jedem System lauffähig, welches die Java2-Plattform unterstützt. Dadurch bietet es gegenüber den Applets entscheidende Vorteile, da es nicht auf einen Browser und das entsprechende Java Plug-In angewiesen ist. Für Entwickler bedeutet dies, daß sie bei der Erstellung ihrer Anwendung nicht auf unterschiedliche Versionen des Java Plug-Ins Rücksicht nehmen müssen. Ein weitere Vorteil ist, daß bei jedem Start einer Anwendung eventuelle Updates automatisch ausgeführt werden, der Anwender arbeitet also immer mit der aktuellen Version.

Vorbereitungen auf der Serverseite

Als Voraussetzung für ein lauffähiges Programm werden hier zwei grundsätzliche Dinge benötigt: Zum Einen müssen alle Klassen, Bilddateien und sonstigen erforderlichen Dateien in ein oder mehrere JAR-Archive gepackt werden, die dann via HTTP zur Verfügung stehen. Die zweite erforderliche Komponente ist eine spezielle Datei mit der Endung „.jnlp“. „jnlp“ steht für „Java Network Launch Protocol“ und enthält die Informationen, die Java Web Start benötigt um die JAR-Archive zu finden und auszuführen. Als Format für diesen Dateityp wird XML verwendet. Das folgende Beispiel zeigt, wie eine solche Datei aussehen könnte und erläutert die jeweiligen Tags:

```
<?xml version=“1.0" encoding=“UTF-8"?>
```

Das „codebase“-Attribut definiert die Verzeichnisstruktur, in der die Archive gefunden werden können. Dies kann auf einem Server sein (HTTP://) oder auch auf einem lokalen Laufwerk (FILE:///).

```
<jnlp codebase=http://www.companySite.com/javaApp>
```

„href“ bezeichnet den eigenen Dateinamen. Dieser wird vom Web Start Application Manager weiterverwendet.

```
<href=“clientApp.jnlp“>
```

Der „information“-Tag bezeichnet den Titel der Anwendung, den Hersteller und ermöglicht Zusätze wie die Angabe der Homepage des Herstellers oder die Verwendung eines best. Logos für eventuell erstellte Shortcuts.

```
<information>
<title>This is my company’s Java client application</title>
<vendor>Company name</vendor>
<icon href=“companyLogo.gif“/>
<homepage ref=“reference/tips.html“>
<offline-allowed/>
</information>
```

Mehrere „resources“-Elemente sind erlaubt. Diese definieren unter Anderem das Betriebssystem, auf dem die Anwendung laufen soll, die für die verwendeten Klassen erforderliche Java-Version (mindestens 1.3) und natürlich die Namen der JAR-Archive.

```
<resources os=“Windows“/>
<resources>
<j2se version=1.3/>
<jar href=“companySong.jar“>
</resources>
```

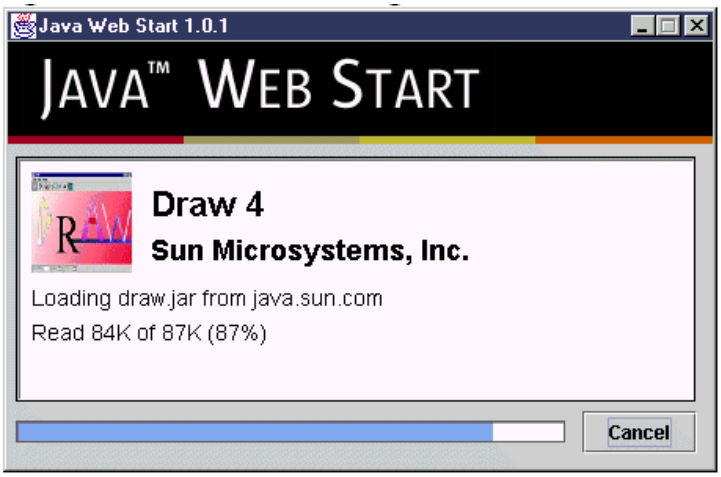
Mit dem Tag „application-desc main-class“ wird der Name der Klasse übergeben, die die main-Methode enthält.

```
<application-desc main-class=“com.company.ui.Client“/>
```

Über den „security“-Tag kann der Zugriff der Anwendung auf den Client geregelt werden. Dieses ist jedoch zur Zeit noch in der Entwicklung, die einzige Option bis dahin ist der Vollzugriff.

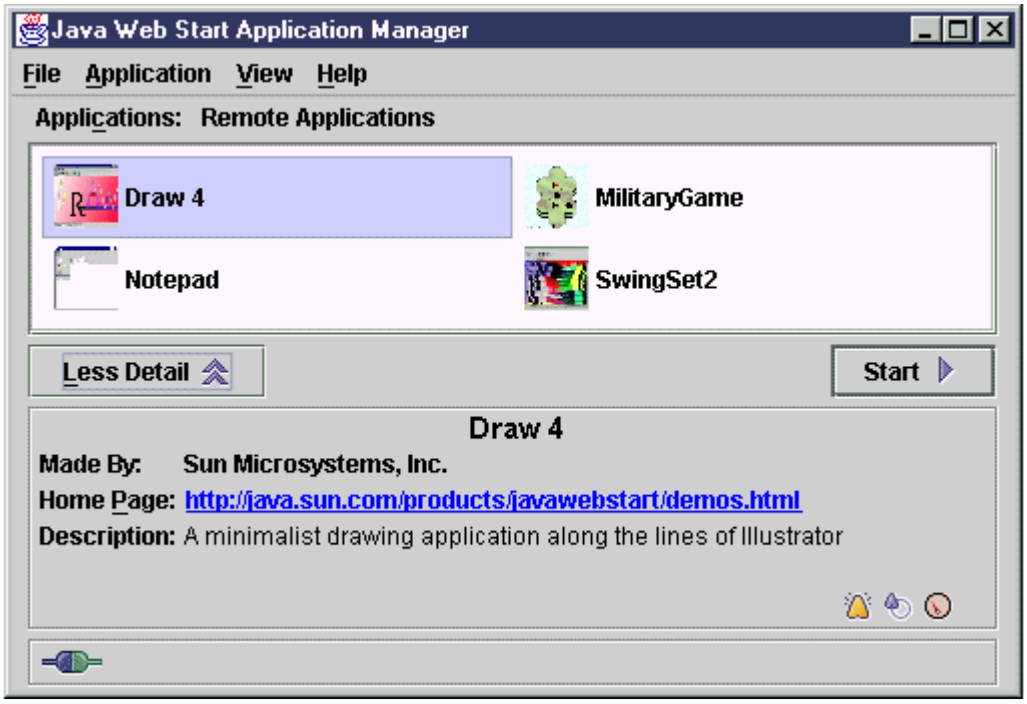
```
<security>
<all-permissions/>
</security>
</jnlp>
```

Diese Applikation ließe sich beispielsweise über einen Browser starten, in dem im HTML-Text eine Hypertext-Referenz (href) eingebaut wird, die auf die entsprechende .jnlp-Datei verweist. Durch Aufrufen dieser Referenz startet der Browser dann Java Web Start. Dieses sucht dann zunächst im lokalen Cache des Clients, ob die erforderlichen Archive dort vorliegen. Ist dies nicht der Fall oder sind diese unvollständig (oder veraltet), werden die fehlenden Archive vom jeweiligen Server angefordert. Sollte die Anwendung zum wiederholten Male gestartet werden erfolgt eine Abfrage, ob Shortcuts für diese Anwendung erzeugt werden sollen. Für den Fall, daß Anwendung Zugriffe auf lokale Dateien ermöglicht erfolgt nochmals eine Sicherheitsabfrage, ob sie wirklich gestartet werden soll. Sind dann alle Archive vollständig geladen wird die Anwendung gestartet. Sie läuft somit vollständig auf dem Client-Rechner. Das folgende Bild ist ein Beispiel für das Aussehen des Ladebildschirmes:



Das Produkt Java Web Start

Eine weitere interessante Komponente von Java Web Start ist der sogenannte Application-Manager. Dieser merkt sich die Anwendungen, die bisher ausgeführt wurden und stellt sie über ein einfaches Auswahlmenü beim Aufruf von Java Web Start zur Verfügung, ähnlich dem von Windows bekannten Startmenü. das folgende Bild zeigt ein Beispiel, wie das Erscheinungsbild des Application-Managers aussehen könnte:



Anhang

Literatur:

- http://java.sun.com
  JDK 1.4 API-Dokumentation
  Java support readiness documents
  Java specification documents

- http://www.ociweb.com
  Object computing inc. - java news brief

- http://www.javamagazin.de
  Tutorial - Java Web Start Workshop

- http://www.javaworld.com
  Artikel - Java web Start to the rescue
  Artikel - Logging API’s for Java

- http://www.ibm.com
  Artikel - IBM developerWorks, java technology

- http://udk.openoffice.org
  Artikel - the logging service

Entwicklungsumgebungen:

- BlueJ
  http://www.bluej.org

- Symantec Visual Café
  http://www.symantec.com

- Java2 standard edition (J2SE)
  http://java.sun.com