J2ME-Tutorial

(Java 2 Micro Edition)

Datum: **10.03.2004**
Verfasser: **M. Frei, R. Wittwer, B. Studer**

Inhaltsverzeichnis

Einleitung

In diesem Tutorial lernen Sie die Java 2 Micro Edition (J2ME) kennen, welches eine Programmiersprache für mobile Geräte darstellt. In einem ersten Teil werden die theoretischen Grundkenntnisse über J2ME erarbeitet, darin enthalten ist der Aufbau, die Konfigurationen und die virtuelle Maschine. Im zweiten Kapitel wird der Wireless Toolkit näher vorgestellt, welches die Entwicklungsumgebung und ein –tool für J2ME zur Verfügung stellt. Anhand dieses Leitfadens kann das Toolkit auf dem eigenen Rechner installiert werden, um dann später eine erste Applikation programmieren zu können. In einem nächsten Schritt wird die Struktur eines MIDlets betrachtet, damit der Weg zu einem MIDlet sowie dessen Rumpf erlernt wird.

Danach werden Sie mit einem geführten Tutorial die erste J2ME-Applikation programmieren. Dabei handelt sich um eine kleine Telefonbuch-Applikation, wo Einträge betrachtet oder hinzugefügt werden können. Das gesamte MIDlet wird schrittweise mit Übungen aufgebaut bis am Schluss eine lauffähiges Telefonbuch erstellt ist.

Viel Spass bei der Durcharbeitung dieses Tutorials!

Voraussetzung

Für die Bearbeitung dieses Tutorial sind die Java-Kenntnisse aus dem Informatikunterricht (1. bis 3.Semester) des Studiengangs Telekommunikation und Informatik erforderlich. Mit diesen Voraussetzungen und dem erarbeiteten Wissen aus dem Theorieteil werden alle Übungen gut lösbar sein.

Lernziele

Nach der Durcharbeitung dieses Tutorials können Sie:

- erklären, was J2ME bedeutet und wie es aufgebaut ist
- sagen, wo J2ME eingesetzt wird
- das Wireless Toolkit bedienen, um MIDlets zu erstellen
- den Weg zu einem MIDlet erläutern
- die Zustände eines MIDlets nennen
- einfache J2ME-Applikationen implementieren
- die wichtigsten Methoden anwenden
- ein MIDlet auf einem Emulator austesten

Zeitmanagement

Für die vollständige Durcharbeitung (Theorie und Übungen) ohne den Miteinbezug der Musterlösungen müssen die folgenden Zeiten eingerechnet werden:

Bereich	Titel / Übung	Kapitel	Geschätzte Zeit
Theorie	Aufbau J2ME	1	45 min
	J2ME Wireless Toolkit	2	30 min
	Installation Wireless Toolkit	2	20 min
	MIDlet-Struktur	3	15 min
	Die erste J2ME-Applikation	4	30 min
	Total		**140 min**
Übungen	Kontrollfragen	1-4	45 min
	Übung 2	4	30 min
	Übung 3	4	15 min
	Übung 4	4	15 min
	Übung 5	4	30 min
	Übung 6	4	30 min
	Übung 7	4	10 min
	Übung 8	4	10 min
	Übung 9	4	20 min
	Übung 10	4	20 min
	Übung 11	4	45 min
	Total		**270 min**
TOTAL			**410 min** **˜ 9 Lektionen**

1. Aufbau J2ME

1.1 Einführung

Sun Microsystems hat in den letzten Jahren mit ihren Anwendungsgebieten *Java 2 Enterprise Edition* (J2EE) und *Java 2 Standard Edition* (J2SE) immer mehr Zuspruch erhalten. Die J2EE ist die Plattform für verteilte Client-Server Anwendungen, während J2SE das klassische Java für den Einsatz auf Desktop-Computern darstellt. Es hat sich jedoch in den letzten Jahren gezeigt, dass ein Bedürfnis nach einer Software-Entwicklung für Embedded Devices bzw. mobile Endgeräte besteht. Um dieses Gebiet optimal unterstützen und abdecken zu können, wurde *Java 2 Micro Edition* (J2ME) entwickelt. Ein Übersicht über die Anwendungsbereiche der verschiedenen Editionen sowie der grundlegenden Aufbau sind in den folgenden Abbildungen ersichtlich. Die J2SE-Plattform dient dabei als Basis jeder Java-Plattform. Das Tutorial basiert auf den Quellen [Esch_2003] und [schr_2002].

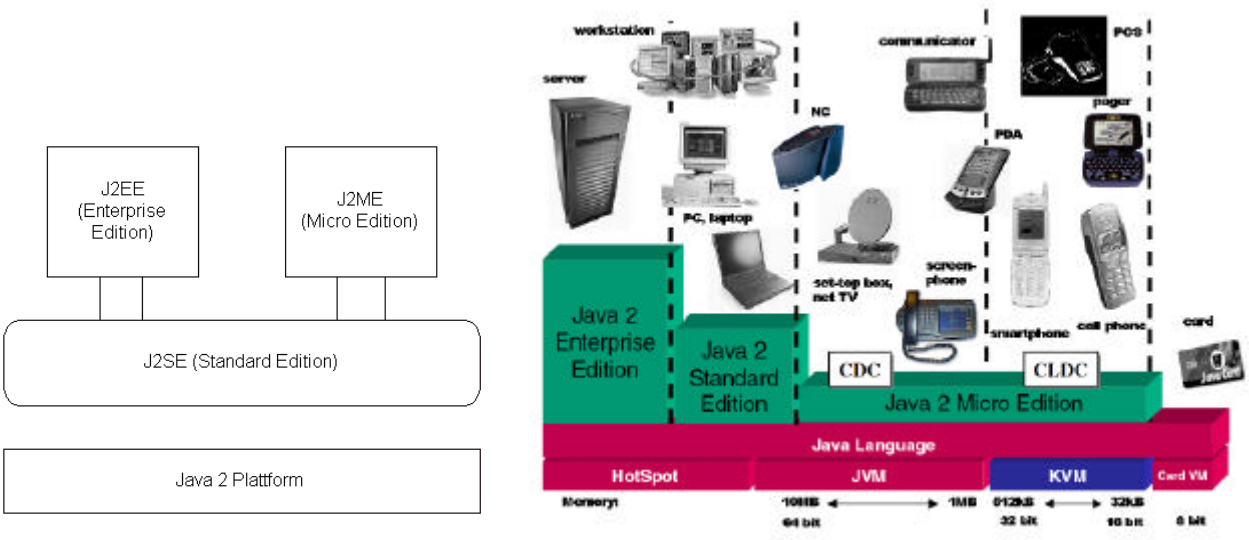

Abb. 1.1-1: a) Aufbau Java -Plattform b) Anwendungsbereiche der Editionen [schr_2002]

1.2 Aufbau der Java 2 Micro Edition

Die J2ME ist speziell für Geräte mit limitierten Ressourcen ausgerichtet. Normalerweise sind dies Einschränkungen, welche die Displaygrösse, Speicherausstattung und Prozessorleistung betreffen. Oft verfügen solche Geräte auch nicht über die Kommunikationsmöglichkeiten und Übertragungsbandbreite von grösseren Systemen. Um diesen Anforderungen gerecht zu werden, musste man auf bestimmte Java-Funktionalitäten verzichten. Andererseits verfügen mobile Geräte über andere Hardware-Komponenten, wie z.B. ein Kartenlesegerät, die ebenfalls in den Anwendungen integriert werden können.
Für die unterschiedlichen Einsatzgebiete muss es also eine identische *Java Virtual Machine* geben, welche das Ausführen solcher Anwendungen erlaubt. Für die Konzeption hat man daher eine spezielle Struktur verwendet. Im Unterschied zu den herkömmlichen Java-Umgebungen gibt es eine *Configurations*- und eine *Profile*-Ebene, welche von zentraler Bedeutung sind. Die Unterteilung dient den unterschiedlichen Konfigurationen der verschiedenen Geräteklassen (bzgl. Grösse und Leistungsfähigkeit). Oft redet man dabei auch von einer horizontalen Unterteilung. Die Struktur ist der folgenden Abbildung ersichtlich.

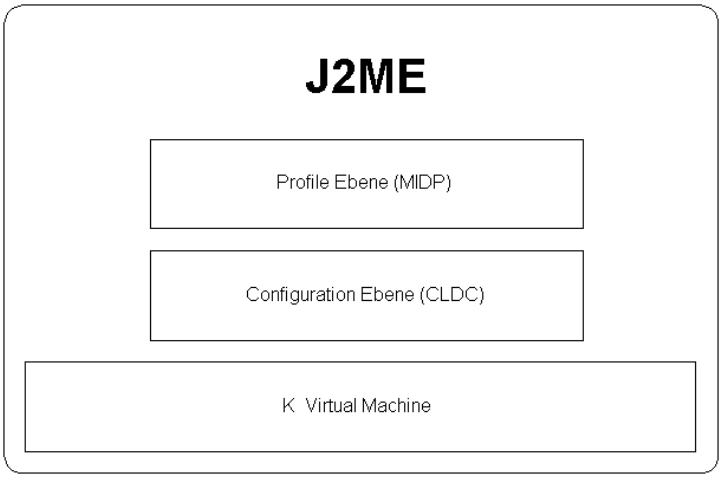

Abb. 1.2-1: Struktur von J2ME (für CLDC-Configuration)

1.3 Configurations

Mit der *Configurations*-Ebene wird man den Anwendungs-Ansprüchen an die verschiedenen Endgeräte gerecht. Eine Configuration stellt basierend auf Hardwaremerkmalen eine Java-Plattform für eine bestimmte Gruppe von Endgeräten dar, welche die jeweils festgelegten Merkmale unterstützen. Mit der Wahl der Configuration werden die folgende Punkte festgelegt:

- Merkmale und Features der Java-Programmiersprache
- Merkmale und Features der Java Virtual Machine
- Java-APIs

Im Moment werden von Sun Microsystems zwei verschiedene Configurations für die Unterstützung bestimmter Geräteklassen angeboten. Dies sind *Connected Device Configuration* (CDC) und *Connected Limited Device Configuration* (CLDC).

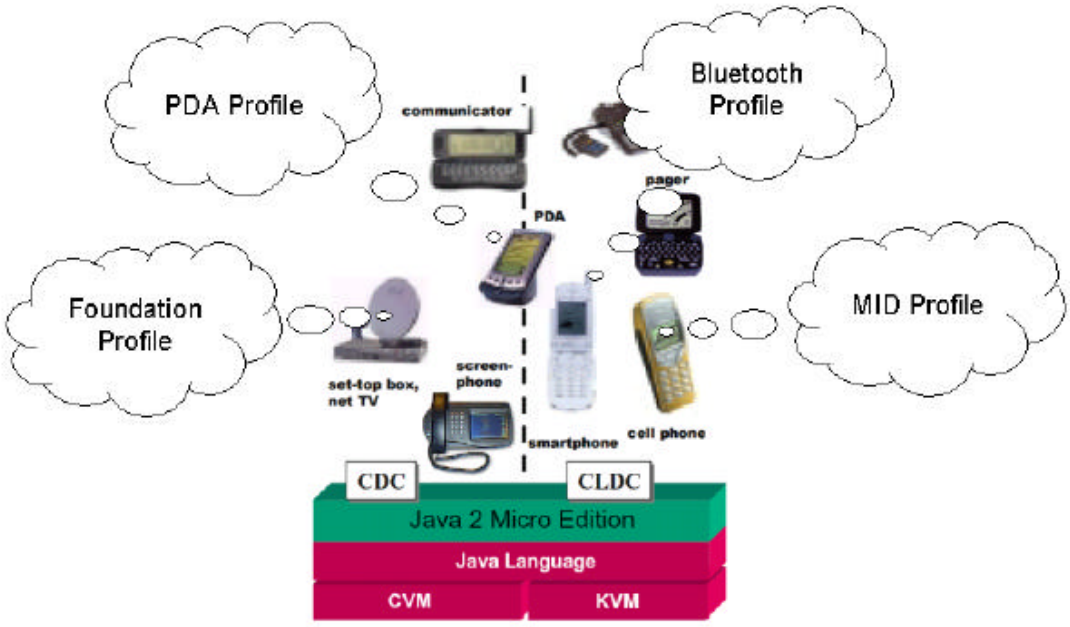

Abb. 1.3-1: Einsatzbereich der Configurations [schr_2002]

Teilweise ist es schwierig, zwischen diesen beiden Konfigurationen eine Abgrenzung zu setzen. Die Praxis hat jedoch gezeigt, dass CLDC viel häufiger zum Einsatz gelangt. Gerade im Schwerpunktbereich der Handys und der mobilen Organizer findet der Connected Limited Device Configuration ein ideales Anwendungsgebiet. Eine dritte Configuration, die für den speziellen Einsatz für PDA's konzipiert wird, steckt derzeit noch in der Entwicklung. Damit sollen die Abgrenzungen zwischen den Hardwaregruppen noch genauer vorgenommen werden können.

1.3.1 Connected Device Configuration

Die Connected Device Configuration deckt den Bereich grösserer und leistungsfähiger Geräte ab. Dazu gehören vor allem grössere PDAs, Set-Top-Boxen oder Smart-Phones. Die CDC schränkt die Leistungsfähigkeit im Vergleich zur Standard JVM nicht ein, verfügt aber über eine auf den reduzierten Speicherbedarf optimierte Virtuelle Maschine, die sogenannte *CVM*. Als Minimum werden Geräte verlangt, welche über ein vollständiges Betriebssystem und mindestens zwei Megabyte Speicher verfügen. Es müssen mindestens 512 Kilobyte für die Java-Umgebung und 256 Kilobyte für die Laufzeit-Umgebung von Java vorhanden sein.
Ein wichtiges Kompatibilitätsmerkmal der CDC ist, dass sie die Möglichkeit besitzt, auch Profile der CLDC zu verwenden. Das heisst CLDC-Profile und Applikationen sind aufwärtskompatibel zu CDC, welche das Foundation-Profile, RMI-Profile und das Personal-Profile unterstützt.

1.3.2 Connected Limited Device Configuration

Die *Connected Limited Device Configuration* bezieht sich auf Geräte, die kleiner und leistungsschwächer sind als Geräte, die unter CDC fallen. Typischerweise sind dies Mobiltelefone, Pager oder kleinere PDA's.
Die CLDC gibt mit der Kilobyte Virtual Machine (KVM) eine VM vor, welche einen geringeren Speicherbedarf als die CVM der CDC benötigt. Im Gegensatz zur CDC (32-Bit-Prozessor) wird nur ein 16-Bit-Microprozessor und ein 160 kB Arbeitsspeicher verlangt. Dieser setzt sich aus den 128 Kilobyte für die Java-Umgebung und den 32 Kilobyte für die Laufzeitumgebung von Java zusammen. Die Geräte können auch über eine mobile Stromversorgung verfügen. CLDC unterstützt das *Mobile Information Device Profil* (MIDP), *Java API for Bluetooth* und das *PDA*-Profile, welches jedoch noch nicht endgültig spezifiziert ist.

1.4 Profiles

Die *Profiles* setzen den Configurations auf und erweitern somit die zur Verfügung stehenden APIs. Die Configurations implementieren die Unterstützung für eine grosse Anzahl an Endgeräten, welche zu einer speziellen Gerätegruppe gehören. Diese Einteilung genügt jedoch nicht, da sich auch Geräte innerhalb einer solchen Gruppe in vielen Punkten unterscheiden können. Zum Beispiel gibt es bei den Mobiltelefonen die verschiedensten Displayvarianten. Um den Herstellern eine Möglichkeit zu geben, Java 2 Micro Edition an ihre speziellen Eigenschaften anzupassen, existiert diese zusätzliche Profil-Ebene.
Am meisten wird das *Mobile Information Device Profile* (MIDP) verwendet, welches speziell auf die Anforderungen von mobilen Geräten wie beispielsweise Handys ausgerichtet ist

1.4.1 Mobile Information Device Profile

Das *Mobile Information Device Profile* ist für CLDC-Anwendungen gedacht und wird für kleine, mobile Geräte mit Netzanbindung eingesetzt. Neben Methoden zur Ansteuerung von kleinen Displays stellt das MIDP ein Framework zur Ausführung sogenannter MIDlet's (vergleichbar mit den Java-Applets) zur Verfügung. Dies ermöglicht, dass ein *MIDlet* von einem Webserver heruntergeladen und auf einem Endgerät ausgeführt wird. Sie sind in Form von *MIDlet-Suiten* in JAR-Archiven abgelegt. Es ist durchaus denkbar mehrere MIDlet's gleichzeitig auszuführen. Die *JAR-Datei* dient zur einfachen Weitergabe von Anwendungen, indem die Programmdateien in einem komprimierten Archiv zusammengefasst werden. Solche Archive werden durch die JAR-Manifeste innerhalb des JAR-Archives beschrieben. Diese geben unter anderem Auskunft über den MIDlet-Namen, die Versionsnummer, Herstellerangaben und die Profilzugehörigkeit. Um jedoch alle wichtigen Informationen bereits vor dem Download bzw. der Installation abrufbereit zu haben, gibt es eine weitere Beschreibungsdatei. Diese wird *Java-Application Descriptor* genannt, wovon die Dateiendung *JAD* stammt. Für die Ausführung in einem Emulator ist die JAD-Datei ebenfalls zwingend notwendig. Weil der Descriptor im Prinzip die gleichen Informationen wie die Manifest-Datei innerhalb des Archivs enthält, unterscheidet sich auch deren Aufbau nur geringfügig. In der folgenden Auflistung ist erkennbar, wie eine solche JAD-Datei prinzipiell aussieht:

```
MIDlet-1:                  test, test.png, test
MIDlet-Description:        Erster Test
MIDlet-Jar-Size:           913
MIDlet-Jar-URL:            test.jar
MIDlet-Name:               Test-MIDlet
MIDlet-Vendor:             Markus Frei, Roger Wittwer
MIDlet-Version:            1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile:      MIDP-2.0
```

In der Tabelle sind die Attribute der Manifest und die JAD-Datei aufgelistet:

Attribut	**Bedeutung**
MIDlet-Name	Name der MIDlet-Suite
MIDlet-Version	Versionsnummer der MIDlet-Suite
MIDlet-Vendor	Hersteller
MIDlet-Description	Allgemeine Beschreibung
MIDlet-Info-URL	URL für weitere Informationen
MIDlet-Data-Size	Mindestspeicher für die Speicherung des MIDlets
MIDlet-1	Name, Icon und Klasse des MIDlets
MicroEdition-Profile	Name und Version des Profile
MicroEdition-Configuration	Name und Version der Configuration
MIDlet-JAR-URL	URL zum Download
MIDlet-Icon	Pfadangabe für Icon
MIDlet-JAR-Size	Grösse der MIDlet-Suite JAR-Date

MIDlet-Suite steht für den Gesamtbegriff für alle MIDlets und Ressourcen, die innerhalb eines JAR-Archivs zusammengefasst werden. Das Prinzip der JAD-Datei und des Manifestes ist in der folgenden Grafik ersichtlich.

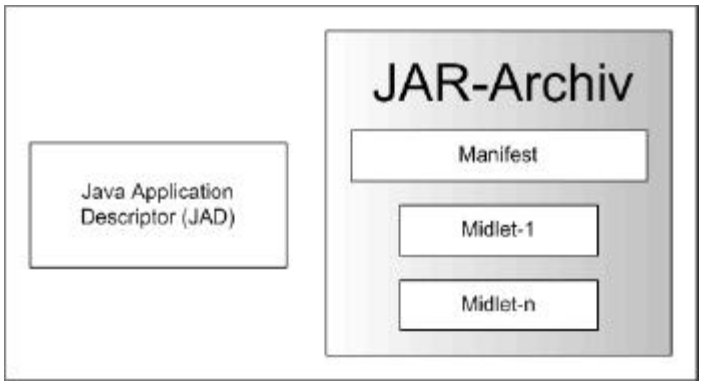

Abb. 1.4-1: Prinzip der JAD-Datei

1.4.2 Java API for Bluetooth (Profile)

Das *Java API for Bluetooth* baut auf CLDC auf und ist bereits vollständig spezifiziert.

1.4.3 PDA Profile

Das *PDA Profile* stellt eine Erweiterung der CLDC dar und ist für spezielle Funktionen von PDAs gedacht. Es ist jedoch noch nicht endgültig spezifiziert.

1.4.4 Foundation Profile

Das *Foundation Profile* wird von der CDC unterstützt. Es ist nur eine Basis für weiterer Profile, welche ebenfalls auf CDC aufbauen.

1.4.5 RMI Profile

Das *RMI Profile* baut ebenfalls auf CDC auf, ist jedoch noch nicht abschliessend spezifiziert. Damit werden die RMI-Funktionalitäten (Remote Method Invocation) zur CDC hinzufügen.

1.4.6 Personal Profile

Das Personal Profile integriert das Personal Java (früherer Ansatz von J2ME) in die CDC. Es besteht im Wesentlichen aus Paketen welche GUI-Funktionalitäten und Java-Applets bereitstellen, da im Prinzip ja schon alle Pakete im Foundation Profile enthalten sind.

1.5 Virtual Machine

Wie bereits die Java Virtual Machine (JVM) bei der Java Standard Edition (J2SE) bildet auch bei der J2ME die *Virtual Machine* die zentrale Komponente für die Ausführung einer Java-Anwendung. Die VM stellt dabei sicher, dass alle Java-Funktionen korrekt auf dem jeweiligen System zur Ausführung gelangen.

Bei J2ME unterscheidet man zwischen der *Kilobyte Virtual Machine* (KVM) und der *C Virtual Machine* (CVM). Die KVM kommt bei CLDC und die CVM bei CDC zum Einsatz. Aus den nun bereits erklärten Abgrenzungen in der J2ME-Welt ergeben sich folgende mögliche Strukturen:

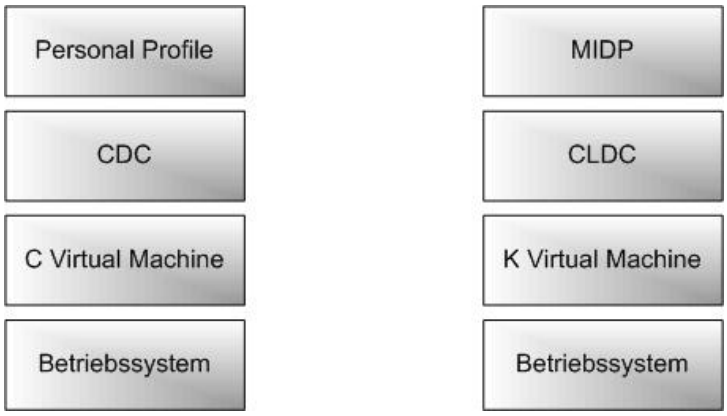

Abb. 1.5-1: Zwei Strukturen einer J2ME-Anwendung

Zur Zeit benützt man für die Entwicklung von Anwendungen mobiler Geräte vor allem die rechte Struktur mit dem MIDP Profil, dem CLDC und der K Virtual Machine.

1.5.1 Kilobyte Virtual Machine

Wegen den limitierten Endgeräten im CLDC-Bereich wurde die KVM völlig neu entwickelt und gegenüber der JVM von J2SE stark abgespeckt. Trotz dieser Begrenzung zählt die KVM laut den Richtlinien der Java Virtual Machine Specification als vollwertige Virtual Machine.

1.5.2 C Virtual Machine

Weil die Geräte im CDC-Bereich grösser sind und praktisch keine Ressourcengrenzen haben, gibt es im Gegensatz zur KVM keine Limitierungen. Somit stehen fast alle Features und Möglichkeiten der herkömmlichen J2SE Virtual Machine zur Verfügung.

1.6 Kontrollfragen zu Kap. 1

1. Für welche Geräte wurde J2ME entwickelt?

.....

.....

.....

.....

2. Aus welchen Ebenen besteht die J2ME-Umgebung?

- .....
- .....

3. Wie sieht der Stack der bedeutendsten Anwendung aus?

.....
.....
.....
Betriebssystem

4. Welches Archiv sorgt für die einfache Weitergabe der MIDlet-Suite, z.B. um die Daten von einem Webserver herunterzuladen und auf einem Client anzusehen?

- ☐ JAR
- ☐ JAD
- ☐ MIDP

2. J2ME Wireless Toolkit

2.1 Einleitung

Um die Vorgänge der Kompilierung und Pre-verifizierung der J2ME-Dateien einfacher zu gestalten, stellt Sun ein spezielles Tool zur Verfügung. Es heisst J2ME Wireless Toolkit und ist zur Zeit in der Version 2.0 Beta 2 vorhanden. Mit Hilfe dieses Programmes wird die Entwicklung eines MIDlets stark vereinfacht und die erstellten Programme können auf einem Emulator getestet werden.
Wie in einem späteren Kapitel noch genauer erklärt wird, bedeutet das Erstellen von J2ME-Anwendungen, eine Reihe von Schritten in einer bestimmten Reihenfolge auszuführen, bis das Programm lauffähig ist. Dieser gesamte Ablauf ist jedoch während einer Programmentwicklung sehr mühsam. Hierfür bietet das Toolkit mit der KToolbar eine einfach zu bedienende Applikation an, womit man während der Entwicklung viel Zeit sparen kann. Für den Einsatz dieses Tools sprechen im Grunde zwei entscheidende Punkte:

- Der gesamte Entwicklungsprozess eines MIDlets wird stark vereinfacht
- Mit den verschiedenen Emulatoren können die Applikationen vor der Portierung auf das Endgerät ausgiebig getestet werden.

Aus diesen Gründen wird für den weiteren Verlauf dieses Tutorials jeweils mit dem Wireless Toolkit gearbeitet. [Esch_2003], [I_sun]

2.2 Voraussetzung

Damit das Wireless Toolkit überhaupt funktioniert, muss die Java Standard Edition auf dem Rechner installiert sein. Dies wird in der weiteren Bearbeitung in diesem Tutorial vorausgesetzt.
Zu diesem Zweck muss nur das Java Development Kit (JDK) von der Internet-Seite von Sun heruntergeladen werden. Der Download kann unter der Adresse http://java.sun.com/j2se/1.4/ vorgenommen werden. Nach der Installation müssen auf einem Windows-Rechner zusätzlich zwei Systemvariabeln gesetzt werden:

- path = C:\j2sdk1.4.1_02\bin;C:\j2sdk1.4.1_02\lib;
- classpath = C:\j2sdk1.4.1_02\lib;

2.3 Download

Das aktuelle Toolkit findet man auf der folgenden Adresse der Sun-Internetseite: http://java.sun.com/products/j2mewtoolkit/

Unter dem Titel Downloads sind die neusten Versionen des J2ME Wireless Toolkits aufgelistet. In dieser Arbeit wurde mit der Version 2.0 Beta 2 gearbeitet. Damit der Download-Vorgang fortgesetzt werden kann, ist jedoch eine Registrierung bei der Sun notwendig. Danach kann das Produkt kostenlos heruntergeladen und gebraucht werden.
Damit das richtige Produkt bezogen wird, muss die richtige Plattform ausgewählt werden. Dabei sind die Optionen *Solaris SPARC*, *Linux* und *Windows* auswählbar.

Nach der Auswahl der betreffenden Plattform wird der Download mit dem continue-Button fortgesetzt. Das folgende „Binary Software Evaluation Agreement“ kann akzeptiert werden, so dass man danach auf den Link der Installationsdatei trifft. Diese exe-Datei wird auf der Festplatte abgespeichert.

Abb. 2.3-1: Downloadportal von Sun

2.4 Installation

Die Ausführung der Installation des Wireless Toolkits ist sehr einfach zu bewerkstelligen, das Installationsprogramm führt den Benutzer schrittweise durch das Menu. Zu diesem Zweck muss zuerst das vorher abgespeicherte File (Z.B. j2me_wireless_toolkit-2_0-beta2-bin-windows.exe) gestartet und danach den Installationsanweisungen gefolgt werden. Dabei müssen verschiedene Pfade ausgewählt beziehungsweise gesetzt werden. Standardmässig werden vom Installationsprogramm die Einstellungen gemäss der linken Grafik erstellt. Während der Installation werden nun sämtliche Daten automatisch konfiguriert, ohne dass der Benutzer weitere Einstellungen vornehmen muss.

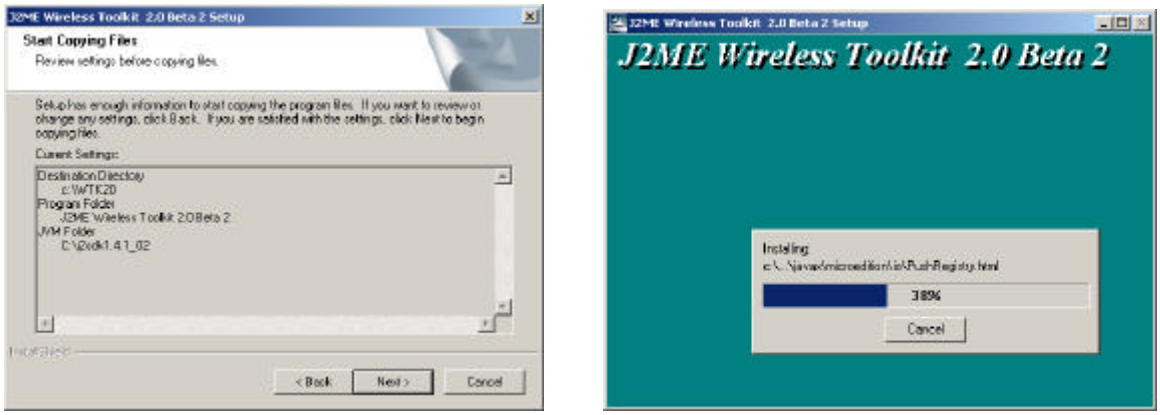

Abb. 2.4-1: Installationsvorgänge des Wireless Toolkits

2.5 Konfiguration

Unter Windows findet man im Menu Start/Programme einen entsprechenden Eintrag für das J2ME Wireless Toolkit. Dabei stehen die folgenden Auswahlpunkte zur Verfügung:

- Default Device Selection
- Documentation
- KToolbar
- Preferences
- Run MIDP Application
- Utilities

Über all diese Punkte kann das gesamte Toolkit konfiguriert und betrieben werden. Das wichtigste Element ist jedoch eindeutig die KToolbar, mit der die MIDlets schlussendlich erzeugt werden.

2.5.1 Default Device Selection

Über diesen Menupunkt kann der Standardemulator ausgewählt werden. Mit einem Emulator können die J2ME-Programme auf einem Desktop-PC simuliert werden. Dazu gibt es Ansichten von verschiedenen Engeräten. Nach der Standardinstallation sind im Toolkit fünf Emulatoren auswählbar, die unterschiedliche Geräte darstellen.

Abb. 2.5-1: Auswahlmenu

Name	Beschreibung	Ansicht
DefaultColorPhone	Dieser Emulator bietet eine Plattform, bei welchem ein Handy mit einem Farbdisplay emuliert wird.	
DefaultGrayPhone	Hier wird ein Handy mit einem Zweifarbdisplay dargestellt. Damit können die Applikationen ohne Farben betrachtet werden, was bei vielen Handys vorkommen kann.	
MMEmulator	Der MMEmulator ist vorwiegend für Multimedia-Anwendungen gedacht. Allerdings stürzt dieser Emulator sehr oft ab und ist deshalb nicht brauchbar.	

MediaControlSkin	Der MediaControlSkin ist gleich, wie das DefaultColorPhone aufgebaut. Lediglich die Tasten haben Zusatzinformationen für Media-Anwendungen (Play, Stop, etc.)	
QwertyDevice	Dieser Emulator stellt einen Standard-PDA mit einer Tastatur dar. Dabei ist das Display um einiges Grösser als bei den Handy-Modellen.	

2.5.2 Documentation

Die *Documentation* liefert den Zugriff auf einige Hilfedokumente im PDF-Format. Die Zusammenstellung ist in einer HTML-Seite dargestellt und mit den betreffenden Links gelangt man in die gewünschte Hilfe. Es werden die folgenden Themen in Englisch angeboten:

- Release notes
- Binary Release License
- User Guide
- Basic Customization Guide
- MIDP 2.0 APIs
- Wireless Messaging APIs
- Mobile Media APIs
- Demo Applications' Documentation

2.5.3 KToolbar

2.5.3.1 Bedienoberfäche

Die *KToolbar* ist das eigentliche Herzstück des Toolkits, welche die Entwicklung eines MIDlets stark vereinfacht. Es handelt sich dabei um keine komplette Entwicklungsumgebung für J2ME, da beispielsweise ein Code-Editor und Syntaxhervorhebungen nicht dazugehören. Die KToolbar stellt als grafische Bedienoberfläche eine optimale Ergänzung zu den anderen Komponenten des Wireless Toolkits dar. Durch diese Anwendung behält man den Überblick über die erstellten J2ME-Projekte und beschleunigt den gesamten Übersetzungsvorgang (Kompilieren, Pre-verifizieren, etc.) massiv.
Die KToolbar ist komplett in Java geschrieben und steht für die verschiedensten Plattformen zur Verfügung (Windows, Linux, Solaris). Allerdings ist die Applikation nicht sehr schnell und benötigt vor allem bei grösseren Projekten eine gewisse Zeit, bis Manipulationen (Aufstarten, Emulator-Ansicht, usw.) durchgeführt werden.

Die Bedienoberfläche der KToolbar ist einfach aufgebaut, der Benutzer findet sich damit schnell zu Recht. Die meisten Buttons sind für den Programmierer selbsterklärend. Nach dem Aufstarten präsentiert sich auf dem Bildschirm diese Darstellung.

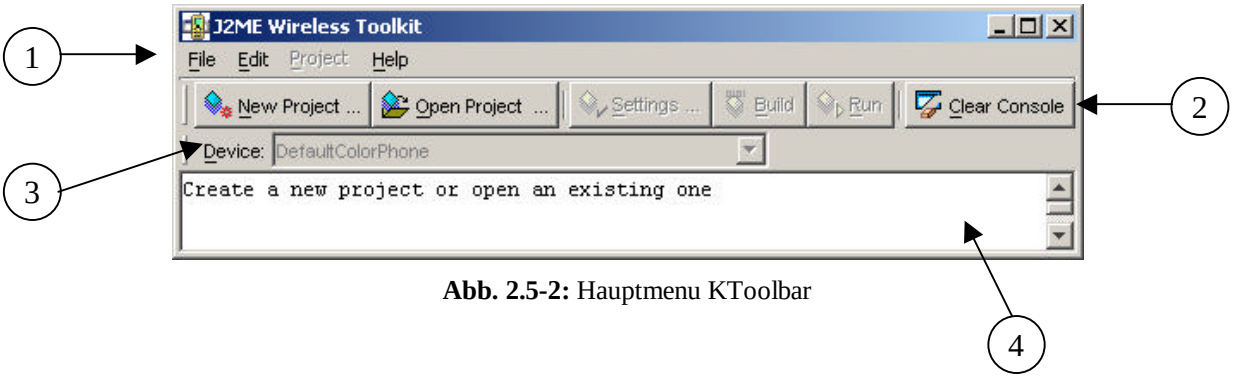

Abb. 2.5-2: Hauptmenu KToolbar

1. **Menupunkte**
   Für sämtliche Befehle und Einstellungen in diesem Tool
2. **Buttonleiste**
   Die Buttons stellen die wichtigsten Befehle dar. Sie sind nur dann aktiviert, wenn sie auch ausgeführt werden können.
3. **Informationsfeld**
   Zeigt an, welcher Emulator ausgewählt ist. Zu Beginn wird immer der Standardemulator angezeigt (siehe Default Device Selection)
4. **Hauptbereich**
   In diesem Feld werden Informationen und Warnungen für den Entwickler angegeben. Ebenfalls werden Ausgaben vom Befehl System.out.println auf diese Konsole geschrieben.

2.5.3.2 Bearbeitung

Ein wichtiger Bestandteilt der KToolbar ist das Projektmanagement, dabei ist vor allem das Anlegen von neuen Projekten entscheidend. Ein Projekt ist in der MIDlet-Sprache gleichbedeutend mit dem Ausdruck MIDlet-Suite. In einer MIDlet-Suite können mehrere MIDlets verwaltet und erstellt werden, gleichzeitig existiert aber nur eine JAR-Datei sowie ein Java Application Descriptor (JAD).
Mit der Schaltfläche *New Project* kann ein neues Projekt erstellt werden. Im darauffolgenden Dialog müssen der Projektname und der MIDlet Class Name eingegeben werden.

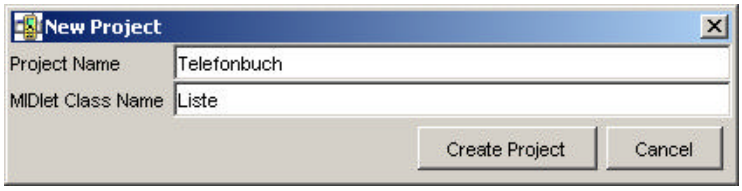

Abb. 2.5-3: Erstellung eines neuen Projektes

Nach dem Betätigung des *Create Project* - Buttons werden die Ordner und die nötigen Dateien automatisch in einem neuen Verzeichnis erstellt. Die aufgeführten Verzeichnisse werden relativ zum Installationspfad des J2ME Wireless Toolkits erzeugt. Gemäss Standardinformation würde die Installation im *Pfad C:\WTK20* stattfinden, andernfalls muss der Pfad angepasst werden. In den weiteren Beschreibungen wird dieser Pfad jeweils als {WTK} angegeben. Die KToolbar generiert bei jedem neu erstellten Projekt ein bestimmtes Verzeichnisschema gemäss der Struktur, die in der untenstehenden Tabelle aufgezeigt wird.

Verzeichnis	Inhalt allgemein
`{WTK}\apps\[Projektname]\bin`	Dieser Ordner enthält die Quellcodedateien des Projektes. Damit sind das kompilierte MIDlet in Form einer jar-Datei sowie die MIDlet-beschreibende jar-Datei gemeint. Bereits nach dem Anlegen eines neuen Projektes werden hier zwei Files erstellt: eine jad-Datei (Telefonbuch.jad) und ein MF-File (Manifest.mf).
`{WTK}\apps\[Projektname]\res`	Im res-Verzeichnis werden sämtliche Ressourcedaten wie beispielsweise Bilder oder Text des Projektes abgespeichert.

Während ein neues Projekt erstellt wird, erscheint ein Dialog, der zahlreiche Einstellungsmöglichkeiten für das betreffende Projekt enthält. Die Daten werden zwar standardmässig eingefügt, können aber vom Programmieren abgeändert werden. Das Fenster kann später auch über den Button *Settings* erreicht werden. Der Dialog stellt die Parameter der jad-Dateien zur Verfügung, welche in verschiedene Registerkarten unterteilt sind:

- Required: Attribute, die zwingend gesetzt werden müssen
- Optional: Diese Attribute können bei Bedarf gesetzt werden
- User Defines: Hier können benutzerdefinierte Attribute gesetzt werden
- MIDlets: Liste der im Projekt enthaltenen MIDlets
- Push Registry: Netzwerkverbindungen für MIDlet einstellen (optional)
- Permissions: Zuständigkeit für die Sicherheit (optional)

Die wichtigsten Einstellungen für ein einfaches MIDlet sind die Daten in der ersten Registerkarte *Required*. Hier werden alle relevanten Angaben für die jad-Dialoge konfiguriert. Beim Erstellen eines neuen Projektes werden diese Daten wie folgt eingetragen.

MIDlet-Jar-Size	100
MIDlet-Jar-URL	Telefonbuch.jar
MIDlet-Name	Telefonbuch
MIDlet Vendor	Sun Microsystems
MIDlet-Version	1.0
MicroEdition-Configruation	CLDC-1.0
MicroEdition-Profile	MIDP-2.0

Das Kompilieren der Java-Sourcen kann per Knopfdruck (*Build*) vorgenommen werden. Voraussetzung ist, wie bei der Java Standard Edition, dass die Syntax des Codes korrekt ist. Die java-Datei muss vorgängig ins Verzeichnis *src* des betreffenden Projektes gelegt werden.

Die Schaltfläche *Build* aktiviert den gesamten Erstellungsprozess für die gesamte MIDlet-Suite: Kompilieren, Pre-verifizieren sowie das Erstellen des kompletten jar-Archivs für die Anzeige auf dem Emulator. Während dem Build-Vorgang werden auf der Konsole Meldungen ausgegeben, mit denen man erkennen kann, ob ein Fehler aufgetreten oder die Applikation nun ausführbar ist. Wurde der Build-Vorgang erfolgreich beendet, kann das MIDlet über den *Run*-Button gestartet werden, dabei wird der eingestellte Emulator für die Simulierung aufgerufen.

2.5.4 Preferences

Der Auswahlpunkt Preferences beinhaltet verschiedene Einstellungen für die DefaultEmulatoren und den MMEmulator. Dies sind vorwiegend Konfigurationen für die Netzwerkverbindungen, die Sicherheit sowie Features für die Tonausgabe. In diesem Bericht werden jedoch auf die Möglichkeiten dieses Menus vorläufig nicht näher eingegangen.

2.5.5 Run MIDP Application

Der Name dieses Menupunktes gibt dessen Funktionalität bereits wieder: hier können die MIDlets direkt auf dem Standardemulator laufen gelassen werden. Dafür muss jedoch zuerst die zum MIDlet zugehörige jad-Datei im Dateiverzeichnis angewählt werden. Dieses File ist im jeweiligen Projekt im Ordner bin zu finden. Mit dem *Run*-Button kann danach das MIDlet manuell gestartet werden, sofern dieses MIDlet bereits Kompiliert und Pre-verifiziert worden ist.

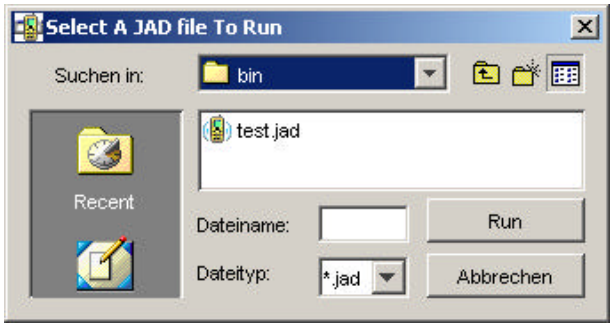

Abb. 2.5-4: Run MIDlet mit jad-Datei

2.5.6 Utilities

Mit diesem Tool können weitere Einstellungen für die MIDlet-Anwendungen vorgenommen werden. Für diese Arbeit werden diese Eigenschaften vorläufig noch nicht benutzt, weshalb auf eine genauere Erläuterung verzichtet wird.

2.6 Fazit

Das Wireless Toolkit ist ein wichtiges Tool im Zusammenhang mit J2ME. Es ist deutlich ersichtlich, dass die KToolbar den Entwicklungsprozess für J2ME-MIDlets durch die grafische Bedienoberfläche erheblich vereinfacht. Zudem sind auch die Vorgänge für die Kompilierung und Per-verifizierung problemlos zu machen. Über die Kommandozeile ist dies zeitintensiv, mit der KToolbar kann es mit einen einzigen Klick komplett ausgeführt werden. Der Programmierer muss sich aber bewusst sein, dass die Source fehlerfrei und sythaktisch korrekt sein muss. Mit den verschiedenen Emulatoren können die Applikationen bereits im Entwicklungsstudium simuliert werden, damit der Programmierer bereits einen Eindruck über die Funktionsweise uns das Aussehen des Programmes machen kann.

2.7 Kontrollfragen zu Kap. 2

1. Welche Vorteile bietet der Wireless Toolkit?

.....

.....

.....

.....

2. Wie nennt man ein Projekt in der MIDlet-Welt?

- ☐ MIDlet-Tool
- ☐ MIDlet-Klassen
- ☐ MIDlet-Suite

3. Was ist ein Emulator?

.....

.....

.....

.....

3. MIDlet-Struktur

Wenn das *J2ME Wireless Toolkit* erfolgreich installiert ist, ist man nun in der Lage erste J2ME-Applikationen ausführen zu können. Dieser Teil beschränkt sich an dieser Stelle auf MIDP-Applikationen (MIDlet), diese Begrenzung beruht auf der Neuigkeit von J2ME, denn es sind noch nicht alle Profile und Configurationen endgültig spezifiziert. Wirklich verbreitet ist erst das MIDP-Profil, welches vor allem für mobile Geräte mit beschränkten Ressourcen zum Einsatz gelangt.

3.1 Der Weg zum MIDlet

Der Weg bis eine Applikation lauffähig ist, besteht aus mehreren Schritten. Die Automatisierung über KToolbar ist ja bereits im Tutorial beschrieben worden. Prinzipiell sieht die Reihenfolge der Entwicklung bis zur fertigen Applikation folgendermassen aus:

1) Schreiben des Java-Quellcodes	Jedes Programm (Quelltext) muss natürlich zuerst geschrieben werden.	
2) Kompilierung	Nach dem Erstellen muss der Sourcecode compiliert werden ✍Erzeugt Klassendatei	javac –bootclasspath %MIDAPI% -classpath %J2MECLASSPATH% TestMidlet.java
3) Pre-verifizieren	Bei der Pre-Verifizierung werden alle Klassen nach Laufzeitfehler überprüft. Denn die KVM besitzen aus ressourcensparenden Gründen keine Pre-Verifizierer.	preverify -classpath %MIPAPI% TestMidlet
4) Erstellen einer JAR-Datei	Um ein MIDlet ausliefern zu können, werden Klassendateien und Ressourcen in einem JAR-Archiv zusammengefasst.	jar cvf TestMidlet.jar TestMidlet.class
5) Erstellen einer JAD-Datei	In der JAD-Datei (Java Application Descriptor) sind Informationen über ein MIDlet gespeichert.	Mit einem Tool oder im Editor erstellen
6) Testen des MIDlets	Das MIDlet kann jetzt in einem Emulator oder nach Übertragung auf das mobile Gerät getestet werden.	

Um den Sourcecode kompilieren und Pre-verifizieren zu können, muss natürlich der JDK (Java Development Kit von J2SE) und die Komponenten von der J2ME-Plattform installiert sein. Anstelle der J2ME-Installation von CLDC und MIDP kann auch zur Vereinfachung nur auf das J2ME Wireless Toolkit zugegriffen werden. Somit kann alles über die KToolbar ausgeführt werden und die DOS-Befehle werden nicht mehr benötigt. Ansonsten müssen zwei Variablen definiert werden:

```
MIDPAPI :=  C:\J2mewtk\lib\midapi.zip
J2MECLASSPATH := C:\J2mewtk\wtklib\kenv.zip;C:\J2mewtk\wtklib\kvem.jar;C:\J2mewtk\wtklib\lime.jar
```

In den weiteren Ausführungen in diesem Tutorial wird jedoch immer von der einfachen Handhabung mit dem J2ME Wireless Toolkit ausgegangen, Voraussetzung dafür ist dessen korrekte Installation.

3.2 Zustände eines MIDlets

Ein MIDlet ist vergleichbar mit einem Applet und durchläuft während der Ausführung drei unterschiedliche Phasen:

- **Active:**
  Während dieser Phase wird gerade eine Funktion ausgeübt und das MIDlet ist damit aktiv, deshalb auch der Name *Active*.

- **Paused:**
  Der Zustand *Paused* wird nach dem Ausführen des Konstruktors und vor der Aktivierung durch den Applikations-Manager eingeleitet. Der Zustand kann auch während dem Ausführen mehrfach eingeleitet und wieder verlassen werden. Diese Phase zeigt die Inaktivität des MIDlets ✍ zur Zeit nicht aktiv!

- **Destroyed:**
  Wenn die Ausführung beendet und die Ressource wieder freigegeben ist, gilt ein MIDlet als *Destroyed* (zerstört).

Um in den Zustand *Destroyed* zu gelangen muss jeweils zuvor in den Zustand *Paused* gewechselt werden. Das heisst, ein aktives MIDlet muss vor dem Beenden zuerst in den Paused-Zustand versetzt werden. Der Lebenszyklus eines MIDlet's ist aus der folgenden Abbildung ersichtlich.

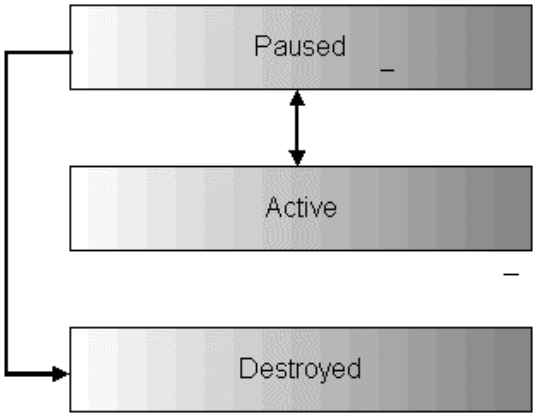

Abb. 3.2-1: Zustände während eines Programmablaufs

Aus diesen Zuständen ist ersichtlich, dass natürlich auch im MIDlet Methoden dazu vorhanden sein müssen. Um ein lauffähiges MIDlet zu erhalten, müssen daher folgende Methoden der Klasse MIDlet überschrieben werden:

- startApp()
- pauseApp()
- destroyApp()

Daraus ergibt sich folgender Grundrumpf für ein MIDlet:

```
Public class MIDletTest extends MIDlet
{
      public MIDletBespiel()    //Konstruktor
            {
             }

            public void startApp()
            {
             }

            public void pauseApp()
            {
             }

            public void destroyApp(boolean unconditional)
            {
             }
```

Die Zustandswechsel werden durch den *Application Manager* vorgenommen, können jedoch auch explizit durch die Methoden resumeRequest(), notifyPaused() und notifyDestroyed() gefordert werden.

3.3 Kontrollfragen Kap. 3

1. Wie lautet die richtige Reihenfolge zur Erzeugung eines MIDlet's!
(Nummerieren Sie die Kästchen mit der Reihenfolge)

- ☐ Erstellen einer JAR-Klasse
- ☐ Pre-Verifizierung
- ☐ Testen des MIDlets
- ☐ Kompilierung
- ☐ Schreiben des Java-Codes
- ☐ Erstellen der JAD-Datei

2. Welche Zustände kann ein MIDlet durchlaufen?

- .....
- .....
- .....

3. In welchen dieser Zustände wechselt man beim Beenden eines MIDlets?

.....

4. Die erste J2ME-Applikation

Mit Hilfe des Tutorials soll eine einfache Applikation programmiert werden. Das Ziel ist es, einen ersten Einblick in die Möglichkeiten von J2ME zu erhalten. So wird ein simples MIDlet zu einer kleinen Telefonbuch-Applikation erweitert.
Allfällige Hilfe bietet auch die J2ME-Sprachspezifikation auf der Webseite der Firma sun. Leider ist es zur Zeit nicht möglich, die Spezifikation online zu betrachten, sondern sie ist nur als zip-File abrufbar. Deshalb muss die zip-Datei heruntergeladen und auf dem Rechner entpackt werden. Zum Starten der Spezifikation die Datei index.html im Internet Explorer öffnen.

Infos unter http://java.sun.com/apis.html#j2me

Übung 1

Lesen Sie die Kapitel 1 und 3 dieses Tutorials sorgfältig durch, denn darin sind sämtliche J2ME-Grundlagen enthalten. Installieren Sie danach den Wireless Toolkit mit Hilfe des Kapitels 2 auf einem geeigneten Laufwerk des Rechners!

4.1 Einfache Anzeige eines Textes

In diesem Kapitel soll lediglich eine Textausgabe auf dem Display erzeugt werden. Als Titel wird „Mein Telefonbuch“ und als Text „Willkommen“ ausgegeben. Die Ausgabe sollte schlussendlich folgendermassen aussehen:

Abb. 4.1-1: Erste MIDlet-Ausgabe

Zuerst muss mit der Methode *getDisplay()* eine Referenz auf das jeweilige Display resp. den Emulator hergestellt werden. Diese Referenz ist auch vom Typ Display. Die eigentliche Textausgabe geschieht in einer *TextBox*. Der Konstruktor dieser TextBox sieht wie folgt aus:

```
TextBox(String title, String text, int maxSize, int constraints)
```

Dieses Objekt erwartet also vier Parameter, dies sind vorwiegend der Titel und der gewünschte Text. Der Parameter *maxSize* definiert die maximale Grösse der Buchstaben. Mit der Abfrage-Methode *getMaxSize()* (von der Klasse Textbox) kann diese Grösse nachträglich kontrolliert werden. *Constraints* dient zur speziellen Formatierung der Textbox, es legt die erlaubten Zeichen fest, die in dieser Textbox stehen dürfen. So kann man definieren, dass nur Zahlen, URLs, Passwörter (Anzeige mit ***) usw. möglich sind. Jeder dieser Form ist eine Zahl zugeordnet, so bedeutet eine 0 alles möglich und eine 1 legt fest, dass nur Zahlen darin enthalten sein können.

Um jetzt die Anzeige auf dem Bildschirm darzustellen, muss die TextBox dem Display hinzugefügt werden. Dies geschieht mit der Methode *display.setCurrent(TextBox)*. Damit die nötigen Methoden vorhanden sind, müssen noch die Packages *javax.microedition.midlet.** für das MIDlet und *javax.microedition.lcdui.** für das Display importiert werden.

Übung 2

Implementieren Sie anhand der obigen Beschreibung ein MIDlet mit dem Titel „Mein Telefonbuch“ und der Textausgabe „Willkommen“! Erstellen Sie hierzu mit dem Wireless Toolkit ein neues Projekt namens Telefonbuch, die zugehörende MIDlet Class Name soll TelefonbuchMIDlet benannt werden. Bearbeiten Sie diese Klasse mit einem Editor!

4.2 Erweiterung des einfachen MIDlets

Dieses Kapitel hat das Ziel, ein einfaches MIDlet in ein etwas komplexeres MIDlet zu überführen. Es wird keine feste Struktur in Form einer TextBox mehr verwendet. Zudem soll es auch möglich sein mehrere Seiten miteinander zu verknüpfen (Link).
Um die ganze Darstellung zu verstehen ist jedoch zuerst etwas Theoriewissen nötig. Denn das MIDP-GUI-Modell unterscheidet sich wesentlich vom bekannten GUI-Design der Java Standard Edition, da es wesentlich besser auf die Anforderungen mobiler Endgeräte eingehen muss.

4.2.1 Aufbau MIDP-GUI-API

Das MIDP-API gliedert sich in das *High-Level-API* und das *Low-Level-API*. Das High-Level-API ist für die Darstellung ohne direkten Zugriff auf die Hardware zuständig. Damit sind die Möglichkeiten eingeschränkt, es existiert jedoch eine möglichst grosse Kompatibilität zu anderen Plattformen. Das Low-Level-API hingegen unterstützt weitere Möglichkeiten zur Manipulation und Darstellung. Es stellt jedoch eine Applikation dar, welche speziell für ein bestimmtes Endgerät entwickelt worden ist. Das Low-Level-API besteht aus den Klassen *Canvas* und *Graphics* vom lcdui-Package.

4.2.2 Display und Screen

Wie bereits im ersten Beispiel erwähnt, braucht es für die Darstellung immer eine Referenz auf das Display. Dies wird durch die Methode *getDisplay()* der Klasse Display im lcdui-Package ermöglicht. Vorzugsweise wird dies immer in der Methode *startApp()* realisiert. Pro MIDlet kann nur ein einziges Display-Objekt bestehen.
Die Darstellung, welche gerade auf dem Display aktiv ist, wird als *Screen* bezeichnet. Es ist jeweils nur möglich ein Screen darzustellen, zudem muss jeder Screen einem Display zugeordnet werden. Es besteht die Möglichkeit zwischen den Screens beliebig zu wechseln. Um einen Screen darzustellen, muss vom Display eine Referenz erzeugt werden. Dies geschieht mit der Methode *setCurrent()* von der Display-Klasse. Die möglichen Konstruktoren sehen folgendermassen aus:

```
public void setCurrent(Displayable d);
public void setCurrent(Alert alert, Displayable d);
```

Der Typ *Displayable* stellt einen Screen dar. Mit dieser Methode kann somit schnell einen Screenwechsel vorgenommen werden, was in der zu programmierenden Applikation auch noch

angewendet wird. Die zweite Form erlaubt zudem noch eine Ausgabe vom Typ *Alert*, welcher später noch beschrieben und verwendet wird. Mit der Methode *getCurrent()* kann der aktuellen Screen in Erfahren gebracht werden.
Somit ist es möglich, ein MIDlet mit mehreren Anzeigen zu haben. Im ersten Beispiel stellte das Objekt vom Typ Textbox bereits einen Screen dar. Es gibt prinzipiell jedoch drei verschiedene Arten von Screen-Objekten:

- Low-Level-API
  Objekt der Unterklasse von Canvas oder Graphics bildet das Screenobjekt
- Eine Formkomponente, die aus mehreren Formkomponenten zusammengesetzt ist (z.B. Textfeld, Eingabefelder, Buttons usw.)
- Die Typen List und TextBox repräsentieren schon selber ein Screen-Objekt.

4.2.3 Alert

Die Klasse Alert existiert zur Ausgabe von Meldungen auf dem Display. Neben Textformen können auch Bilder dargestellt werden. Der aktuelle Programmablauf wird durch ein Alert immer unterbrochen. Die Fortsetzung des Programms erfolgt automatisch nach einer gewissen Zeitspanne oder durch manuelles Betätigen. Es sind die beiden folgende Konstruktoren vorhanden:

```
public Alert(String titel);
public Alert(String titel, String alertText, Image image, AlertType
alertType);
```

Der erste Typ verlangt nur einen Parameter und stellt wegen den fehlenden Typangabe nur einen temporären Dialog dar. Die Darstellungszeit hängt vom Standardwert der Plattform ab, sie kann jedoch durch die Methode *getDefaultTimeout()* abgefragt werden. Dieser Wert kann durch die Methode *setTimeout(int i)* beeinflusst und mittels der Methode *getMethode()* abgefragt werden, wobei der int-Wert für die Anzahl Millisekunden steht.
Weitere Einstellungen sind durch den zweiten Konstruktor möglich, welcher ähnlich wie die bereits verwendete TextBox aufgebaut ist. Neben dem Titel kann noch eine eigentlich Meldung ausgegeben werden. Mit dem dritten Parameter ist es möglich, ein Image-Objekt zu übergeben. Der Alert-Typ definiert die Art der Ausgabe, zum Beispiel ob es sich um eine unkritische Information (INFO) oder eine Warnung (WARNING) handelt. Die genauen Möglichkeiten sind in den Spezifikationen nachzulesen. Die verschiedenen Typen unterscheiden sich auch durch die Klänge, welche bei der Darstellung erfolgen.

4.2.4 Erweiterung mit Alert

Unser Willkommen-Screen soll nur für eine gewisse Zeitspanne dargestellt werden, bevor die Anzeige automatisch auf ein Auswahlmenu wechselt. Dazu eignet sich ein einfacher Alert, der nur temporär angezeigt wird. Nach dem Timeout wechselt der Bildschirm automatisch vom alert- auf den Text-Screen. Dies kann mit der Erweiterung der bereits bekannten Methode *setCurrent()* erreicht werden:

```
display.setCurrent(alert, text);
```

Übung 3

Die Ausgabe auf dem Display soll nun mit einem Alert dargestellt werden, ersetzen Sie hierzu die Textbox durch einen Alert! Der Text-Screen muss dabei noch nicht implementiert werden.

Mit diesem Alert wird wegen der Parameterangabe *null* noch kein Bild angezeigt. Zudem verschwindet die temporäre Alert-Anzeige nach einer bestimmten Zeit vom Display (normalerweise standardmässig nach 2000ms). Weil jedoch die Folgeseite in diesem Kapitel noch nicht implementiert werden soll, muss die Dauer der Anzeige angepasst werden. Dies geschieht mit folgender Methode:

```
alert.setTimeout(-2);
```

Das Timeout kann auf FOREVER (für immer) gestellt. Weil für diesen Parameter jedoch ein Integer-Wert verlangt ist, wird anstelle von FOREVER der Integer-Wert **-2** eingesetzt.

Nun soll versucht werden, eine Grafik mit Hilfe des *Alert* darzustellen. Dabei muss das anzuzeigende Bild (picture.png) im png-Format vorliegen und in den Ordner *apps\[projektname]\res* des Wirless Toolkits kopiert werden. Die Klasse Alert verlangt den Typ Image, weshalb zuerst mit *createImages()* ein solcher Typ erzeugt werden muss. Für das Laden des Bildes wird ein ExceptionHandling (import der io-Klasse) angewendet. Dies hat die gleiche Syntax und Funktion wie bei J2SE.

Übung 4

Fügen Sie dem Alert irgendeine Grafik mit dem Dateinamen „picture.png“ hinzu. Das Exception Handling nicht vergessen!

Wenn diese Übung erfolgreich absolviert ist, sollte auf dem Emulator die folgende Anzeige vorhanden sein.

Abb. 4.2-1: Applikations-Darstellung

4.3 Auswahlmöglichkeit

In der nächsten Erweiterung geht es darum ein Auswahlmenu zu kreieren, wo man Auswählen kann, ob im Telefonbuch ein neuer Eintrag hinzufügt wird oder bestehende Einträge angesehen werden. Dieser Auswahldialog soll automatisch nach dem Willkommen-Screen erscheinen. Für die Auswahl wird die Klasse List benützt.

4.3.1 List

Listen werden dann verwendet, wenn der Benutzer eine Auswahl treffen soll. Es gibt je nach Situation verschieden Listentypen, welche unterscheiden, ob es sich z.B. um eine Mehrfach-

oder Exklusivauswahl handelt. Mit der Typangabe kann auch festgelegt werden, dass ein Ereignis ausgelöst wird.

Choice.Exclusive (value 1)	Damit ist nur eine Auswahl möglich ✍Exklusivauswahl
Choice.Multiple (value 2)	Mit diesem Typ können zur gleichen Zeit mehrere Einträge der Liste ausgewählt werden. ✍Mehrfachauswahl
Choice.Implicit (value 3)	Implicit ist eine Exklusivauswahl, bei der jedoch bei der Auswahl eines Elementes ein Ereignis ausgelöst wird.

Die Klasse List verfügt über zwei verschiedene Konstruktoren.

```
public List(String title, int listTyp);
public List(String title, int listTyp, String[] stringElements,
Image[] imageElements);
```

Der erste Konstruktor kreiert eine neue, leere Liste, welche mit einem String als Titel und dem jeweiligen Listentyp spezifiziert ist. Beim zweiten Konstruktor werden zusätzlich zwei Arrays übergeben, womit die Elemente übergeben werden. Dies kann ein Text (String) oder ein Bild (Image) sein. Falls der erste Konstruktor verwendet wird, können mit der Methode *append(String stringPart, Image imagePart)* die Elemente hinzugefügt werden. Falls kein Bild übergeben werden soll, ist der Wert auf null zu setzen.

4.3.2 Erweiterung mit der Auswahl-Liste

Damit der Auswahldialog automatisch nach dem Willkommen-Screen erscheint, muss das Timeout nicht mehr auf unendlich eingestellt sein. Zur Vereinfachung und zur Codetrennug werden ab dieser Aufgabe alle Methoden und Operationen welche nichts mit dem Display zu tun haben im Konstruktor ausgeführt. Zur Darstellung der Auswahl muss einfach die Methode setCurrent(alert) ergänzt werden auf:

```
display.setCurrent(alert, listname);
```

Übung 5

Verschieben Sie die Methode alert.setTimeout und das Alert-Object in den Konstruktor. Danach soll eine List erstellt werden, welcher die beiden Auswahlkriterien „Einträge ansehen“ und „Einträge hinzufügen“ angehängt werden (mittels append!).

Das Resultat dieser Übungen ist es nun, dass ein Auswahlmenu mit den zwei erwähnten Möglichkeiten angezeigt wird.

Abb. 4.3-1: Auswahlmöglichkeit

Es ist sicherlich aufgefallen, dass bei einer Aktivierung einer dieser beiden Auswahlpunkt nichts geschieht. Das Ziel wäre es jedoch, dass ein entsprechendes Programmereignis ausgelöst wird. Dazu muss aber die MIDlet-Ereignisbehandlung etwas genauer erläutert werden.

4.3.3 MIDlet-Ereignisbehandlung

Die *Ereignisbehandlung* wird unterteilt in Ereignis und Behandlung. Eine Ereignis wird normalerweise durch eine Interaktion des Benutzers ausgelöst. Dies kann das Betätigen einer Taste oder die Aktivierung einer Schaltfläche sein. Mit der Ereignisbehandlung kann man beim Auftreten eines bestimmten Ereignisses die gewünschte Funktion ausführen lassen. Beim MIDlet kommt diese Behandlung vor allem bei einem Screenwechsel zum Tragen. In diesem Kapitel geht es nicht darum, die ganze Ereignisbehandlung und dessen Funktionen zu beschreiben. Es sollen nur kurz die Grundeigenschaften erläutert werden, damit einfache Beispiele realisiert werden können.

Für das Abfangen eines Ereignisses wird die Klasse *Command* benutzt, sie steht stellvertretend für ein Ereignis.

```
public Command(String label, int commandType, int priority)
```

Als *Label* kann ein beliebiger String angegeben werden, der lediglich als Bezeichner dient. Der int-Wert spezifiziert einen bestimmten *commandType*. Im Moment ist nur der Type SCREEN (1) von Interesse. Bei der *Priorität* kann ein beliebiger int-Wert eingesetzt werden, wobei eine 1 für die höchste Prioritätsstufe steht. Das Command-Objekt kann mit der folgenden Methode dem gewünschten Objekt angehängt werden.

```
public void addCommand(Command cmd)
```

Für unsere Liste würde es dann etwa folgendermassen aussehen:

```
menu = new List("Auswahl",3);
menu.append("Einträge ansehen", null);
menu.append("Einträge hinzufügen", null);
Command listCommand = new Command(„List“, 1, 1)
menu.addCommand(listCommand);
```

Um auf ein Ereignis speziell reagieren zu können, muss das Interface *CommandListener* implementiert werden. Dieses Interface zieht auch die Implementation der Methode

```
public void commandAction(Command c, Displayable d)
```

nach sich. Darin wird geregelt, wie auf ein bestimmtes Ereignis reagiert werden soll. Der entsprechende *CommandListener* für die Liste muss sich beim betreffenden Objekt mit der untenstehenden Methode registrieren.

```
public void setCommandListener(CommandListener cl)
```

4.3.4 Erweiterung mit Ereignisbehandlung

Nun soll auf die verschiedenen Auswahl-Ereignisse in der Liste reagiert werden. Man kann mit *getselected()* überprüfen, welches Listenelement aktiviert ist. Als Rückgabe erhält man dann den Index des entsprechenden Elementes. In diesem Beispiel soll nur mit *System.out.println()* eine einfache Ausgabe auf der Konsole erzeugt werden.

Übung 6

Nun soll eine Ereignisbehandlung realisiert werden, die auf die Auswahl aus der List reagiert. Erstellen Sie einen Command „listCommand“ und implementieren Sie die folgenden Objecte:

- Command der List anhängen
- CommandListener an List binden
- CommandAction mit System.out.println(...) erstellen

4.4 Werte speichern und ausgeben

4.4.1 Objekt-Klasse Person erzeugen

Natürlich müssen die Einträge, welche man später darstellen will, auch noch irgendwie abgelegt werden. Dazu wird eine neue Klasse *Person* erzeugt, in welcher alle Daten für einen Telefonbucheintrag (Name, Telefonnummer) abgelegt werden können. Diese Objekt-Klasse ist natürlich mit beliebigen Parametern erweiterbar. Die Methode *getName()* und *getNummer()* werden zur Rückgabe der Werte genutzt. Auf dessen Einsatz wird später noch eingegangen.
Diese Klasse muss normal mit dem Java-Compiler (javac) kompiliert und das class-File in das Verzeichnis *[projektname]/classes* eingefügt werden.

```
class Person
{
   private String name;
   private String nummer;

   public Person (String name, String nummer)
   {
      this.name = name;
      this.nummer = nummer;
   }

   public String getName()
   { return name; }

   public String getNummer()
   { return nummer; }
}
```

Übung 7

Erstellen Sie die obenstehende Klasse, danach kann die Klasse ins Verzeichnis *F:\WTK20\apps\Telefonbuch\src* kopiert werden. Beim Ausführen vom WirelessTooKit wird diese Klasse automatisch kompiliert.

In der Klasse *TelefonbuchMIDlet* sind dann einige Einträge standardmässig generiert. Der *Array pa[]* dient als Referenz auf die erstellten Objekte vom Typ Person. Die Arraygrösse wurde als default 20 festgelegt.

```
pa = new Person[20];
pa[0] = new Person ("Frei","0791234567”);
pa[1] = new Person ("Wittwer","0797654321");
```

Übung 8

Im Konstruktor (TelefonbuchMIDlet) sollen zwei Personen (=Telefonbucheinträge) nach obigem Beispiel erzeugt werden. Deklarieren Sie zu Beginn des MIDlets die Variable pa (Person [] pa).

4.4.2 Anzeige der Einträge

Nun soll über die Auswahl „Einträge ansehen“ auch eine Anzeige erstellt werden, mit welcher der Benutzer seine Einträge überprüfen kann. Wenn er den Menupunkt *Einträge ansehen* aktiviert, soll eine Auflistung sämtlicher Einträge im Telefonbuch mit Name und Rufnummer erfolgen.

Abb. 4.4-1: Auswahldialog

Dazu muss die Methode *commandAction()* geändert und die Methode *ausgabe()* neu geschrieben werden. In der Event-Handling-Methode *commandAction()* wird nur beim Ausführen der Auswahlmöglichkeit *Einträge ansehen* (Index 0) die Methode *ausgabe()* gestartet. In dieser Methode wird zuerst mit einer for-Schlaufe überprüft, wie viele Einträge es im Array pa hat. Danach wird ein Object *Form* erzeugt, welchem entsprechende Einträge zugewiesen werden. Als erster wird ein Index erzeugt, welcher zur Nummerierung dient. Als Textübergabe wird ein *StringItem* verwendet.

```
StringItem(String label, String text)
```

Dem *StringItem* können zwei Parameter übergeben werden: der Label und der eigentliche Text! Die Einträge werden über die Methoden *getName()* bzw. *getNummer()* vom Objekt pa gelesen und den StringItems als Parameter übergeben. Mit der Methode *append()* können nun diese StringItems der Form hinzugefügt werden. Am Schluss muss diese neu erstellte Form noch mit der Methode *setCurrent(Form viewForm)* auf dem Display dargestellt werden.

Übung 9

Ändern Sie zuerst den commandAction beim Index 0, indem Sie das *System.out.println("Anzeigen")* durch *ausgabe()* ersetzen.
Studieren Sie die untenstehende Methode *ausgabe()*, damit die einzelnen Schritte nachvollzogen werden können. Diese Methode wird dem MIDlet hinzugefügt. Zu diesem Zweck müssen in der MIDlet-Klasse am Anfang folgende Variabeln deklariert werden:

- int anzahl;
- Form viewForm;
- String index;
- StringItem str1, str2;

```
public void ausgabe()
 {
  //Anzahl einträge evaluieren
  for(int i = 0; i<20 ; i++)
    {
     if(pa[i]==null)
      {
       anzahl=i;
       break;       //springt aus for-Schlaufe
      }
    }

  viewForm = new Form("Einträge");  //Darin werden die Einträge angezeigt
  for(int i=0;i<anzahl;i++)
    {
     index = String.valueOf(i+1);   //Nummeriert die einzelnen Einträge
     str1 = new StringItem(index,pa[i].getName());    //Namen in StringItem
     str2 = new StringItem(null,pa[i].getNummer());  //Nummer in StringItem
     viewForm.append(str1);   //Die beiden StringItem's der Form anhängen
     viewForm.append(str2);
    }
  display.setCurrent(viewForm);     //Form auf Display anzeigen
 }
```

Nach der Implementierung dieser Übung 9 können bei Betätigung der Auswahlmenus „Einträge ansehen" sollte auf dem Display die folgende Ansicht zu sehen sein.

Abb. 4.4-2: Anzeige der Einträge

Nun ist beim Ausführen dieser Applikation sicherlich aufgefallen, dass es von dieser Anzeige-Seite kein zurück mehr gibt. Um auch dies zu implementieren, sind einige Implementierungen im nächsten Kapitel dokumentiert.

4.4.3 Exit- und Back-Schaltfläche

Mit Hilfe von weiteren Schalflächen soll es nun möglich sein, das Programm zu beenden resp. zur letzten Anzeige zurückzukehren. Dafür müssen weitere *Command*'s hinzugefügt werden. Sobald man jetzt zwei Command's hat, erscheinen auf dem Display am unteren Rand die zwei möglichen Schalflächen, welche man über die entsprechenden Buttons aktivieren kann. Im Auswahlmenu steht nun *Exit* um die Applikation zu beenden und *Ausführen* um den entsprechenden Punkt auszuführen. Bei der Anzeige der Einträge wären die entsprechenden Menupunkte *Exit* und *Back*, um auf das Auswahlmenu zurückzukehren. Die Display-Darstellung sollte dann wie folgt aussehen:

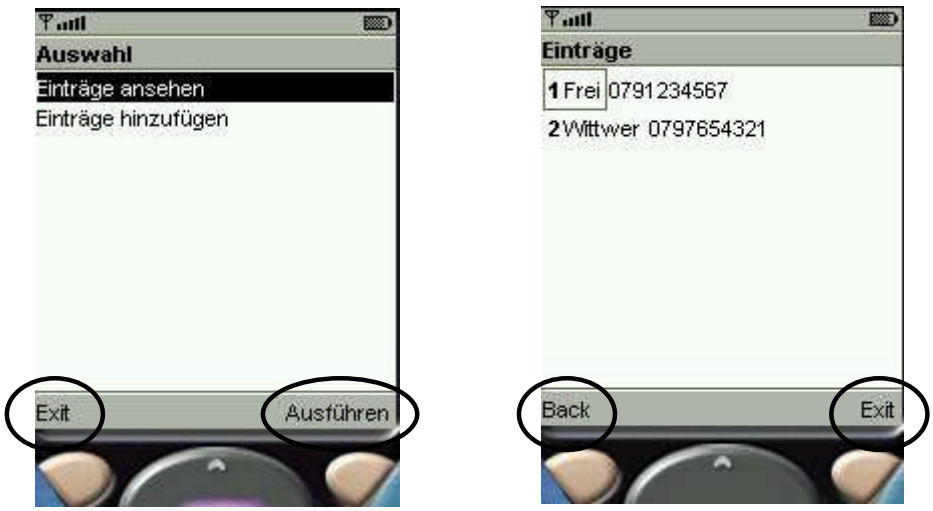

Abb. 4.4-3: Display-Darstellung a) im Auswahlmenu b) in der Eintrags-Anzeige

Der Konstruktor für ein Command sieht folgendermassen aus:

```
Command(String label, int commandType, int priority)
```

Der *commandType* muss dabei beim *exitCommand* auf 7 (für Exit) und beim *backCommand* auf 2 (für Back) gesetzt werden. Die beiden neuen Command's müssen natürlich auch mit der Methode *addCommand(Command)* dem jeweiligen Screen-Objekt hinzugefügt werden. Der viewForm für die Darstellung der Einträge muss danach auch der CommandListener mit der Methode *setCommandListener(this)* angehängt werden, damit die Ereignisse der Schaltflächen Exit und Back auch behandelt werden. Dazu muss jetzt in der Methode zur Ereignisbehandlung *commandAction()* mit einer if-Abfrage überprüft werden, um welchen Command es sich handelt. Falls es sich um ein exitCommand handelt, wird die Applikation einfach „zerstört". Wenn nur ein Back gewünscht wird muss einfach über *setCurrent()* das Auswahlmenu wieder dargestellt werden.

Übung 10

Ziel dieser Aufgabe ist es, die Exit- bzw. Back-Schaltfläche zu implementieren. Gehen Sie dazu schrittweise folgendermassen vor.

- Im Konstruktor ein exitCommand an die List anhängen
- In der Methode ausgabe() ein exitCommand und backCommand an die Form anhängen
- Studieren Sie die untenstehende commandAction-Methode ersetzen die bestehende Methode im MIDlet.

```
public void commandAction(Command c,Displayable d)
      {
        if(c == listCommand)
        {
         if(menu.getSelectedIndex()==0)      //Anzeigen
         { ausgabe(); }

         else if(menu.getSelectedIndex()==1) //Hinzufügen
         { System.out.println("Hinzufügen"); }
        }
        else if(c == exitCommand)
        {
          destroyApp(false);
          notifyDestroyed();
        }

        else if(c == backCommand)
        {
          display.setCurrent(menu);
        }
     }
```

4.5 Hinzufügen von Datensätzen (Optional)

Um nun auch Datensätze hinzufügen zu können müssen zwei neue Methoden geschrieben werden. Die erste Methode heisst *hinzufügen*(), welche beim Ausführen des Listenelementes „*Einträge hinzufügen*“ gestartet wird. Sie ist zuständig für die Eingabe des neuen Datensatzes. In dieser Methode wird eine neue Form erstellt, welcher jeweils ein Textfeld für die Nummer sowie für den Namen hinzugefügt werden (mit append). Die Eingabemaske ist im folgenden Bild ersichtlich.

Abb. 4.5-1: Maske für das Hinzufügen

Der Konstruktor für ein Textfeld sieht folgendermassen aus:

```
TextField(String label, String text, int maxSize, int constraints)
```

Die Bedeutung der Parameter sollten bereits bekannt sein. Als *constraints* für das Nummern-Textfeld empfiehlt sich hier, das Attribut *TelephonNumber (int 3)* zu setzen. Somit können in diesem Feld ausschliesslich Zeichen einer Telefonnummer eingesetzt werden, bei der Verwendung eines Handys bietet sich zudem der Vorteil, dass nicht jede Taste viermal gedrückt werden muss bis die Zahl erscheint. Der Parameter *text* kann null gesetzt werden, denn dieser Wert wird später vom Benutzer durch seine Eingabe gesetzt. Es empfiehlt sich in diesem Fall, zwei *Commands* zu implementieren. Zum einen kann der bereits bestehende *backCommand* verwendet werden, zum anderen muss ein Command erzeugt werden, welcher für das Hinzufügen zuständig ist. In der *commandAction* muss dieses Command natürlich auch abgefangen werden. Bei der Ereignis-Behandlung wird dann einfach die Methode *write()* gestartet, welche ebenfalls neu implementiert werden muss. Sie ist zuständig für das Abspeichern des Datensatzes.

In der Methode *write()* werden zuerst der Name und die Nummer vom Textfeld eingelesen. Dazu eigent sich die Methode *getString()* des Textfeldes. Einfachheitshalber soll bereits hier geprüft werden, ob eines dieser Felder leer (null) ist. In diesem Fall kann ein Fehler-Alert herausgegeben werden.
Falls die Eingabe in Ordnung ist, muss zuerst abgeklärt werden, an welcher Stelle des Arrays der Datensatz gespeichert werden soll. Die Berechnung der Grösse des Arrays ist ja bereits bei der Methode *ausgabe()* (erste for-Schleife) implementiert worden. Um Doppelspurigkeiten zu vermeiden empfiehlt es sich hier eine Methode *getsize()* zu schreiben, welche die bereits bestehenden Source übernehmen und von beiden Methoden verwendet werden kann. Nun kann ein neues Objekt Person erstellt werden und an der richtigen Stelle des Arrays gespeichert werden. Es empfiehlt sich hier, einen kurzen *Alert* herauszugeben, um dem Benutzer das

korrekte Speichern mitzuteilen. Nachher müsste auf die Menuauswahl gewechselt werden. Dazu wird folgende Methode verwendet:

```
Display.setCurrent(Save-Alert, menu)
```

Damit wäre die komplette Applikation realisiert, es können nun die Datensätze angesehen und neu hinzugefügt werden. Es gilt jedoch zu beachten, dass neu hinzugefügte Datensätze nicht persistent gespeichert werden, d.h. beim Programmende auch zerstört werden. Für eine dauerhaft Speicherung müssten *Record Stores* verwendet werden, auf diese hier nicht eingegangen wird.

Übung 11

Versuchen Sie das „Hinzufügen von Datensätzen“ zu implementieren und den bisherigen Source-Code zu erweitern. Gehen sie zu diesem Zweck nach folgendem Muster vor.

- Im Konstruktor zwei neue *Alert* (✍ fehler, save) und ein neuer *Command* (addaComand) deklarieren.
- Methode *getsize()* mit der ersten for-Schleife von *ausgabe()* schreiben.
- Methode *hinzufügen()* implementieren. Darin *Textfelder* erstellen und den Inhalt in *Form* schreiben.
- Methode write() implementieren, worin das Objekt Person erzeugt werden.
- Im *CommandAction* das System.out.println(..) beim Index 1 durch *hinzufügen()* ersetzen. Zudem eine neue Option (c == addaCommand) mit der Methode *write()* als Inhalt.

4.6 Kontrollfragen zu Kap. 4

1. Welches Object wird für ein Auswahlmenu verwendet?

- ☐ List
- ☐ Alert
- ☐ Textbox

2. Auf welches Object muss für die Darstellung auf dem Gerät immer referenziert werden?

- ☐ Exit
- ☐ Display
- ☐ Alert

3. Was wird im CommandAction jeweils vorgenommen?

.....

.....

4. Was kann bei einer Textbox in J2ME speziell formatiert werden?

.....

.....

5. Methodenübersicht

In dieser Tabelle sind die meisten verwendeten Methoden mit einer kurzen Beschreibung aufgelistet.

Methode mit Beschreibung
public **Alert**(String title, String alertText, Image alertImage, AlertType alertType) Timer-abhängige Ausgabe von Text oder Bilder auf dem Bildschirm
public **Command**(String label, int commandType, int priority) Definiert ein bestimmtes Ereignis, welches Auftreten kann
protected abstract void **destroyApp**(boolean unconditional)throws MIDletStateChangeException Wird ausgeführt, wenn MIDlet beendet wird und Ressourecen freigegeben werden
public boolean **equals**(Object anObject) Vergleichen von zwei Strings [string.equals("...")
public int **getSelectedIndex**() Gibt den ausgewählten Punkt einer List wieder [list.getSelectedIndex()==0]
protected abstract void **pauseApp**() Deren Funktionen werden ausgeführt, wenn MIDlet inaktiv ist
public void **setCurrent**(Alert alert,Displayable nextDisplayable) Ein Object dem Display übergeben [display.setCurrent(text)]

protected abstract void **startApp**() throws MIDletStateChangeException Deren Funktionen werden ausgeführt, wenn MIDlet aktiv ist
public **StringItem**(String label, String text) Textformat (wie ein String) mit zwei Parameter (label und text)
public **TextField**(String label, String text, int maxSize, int constraints) In diesem Feld können auf dem Bildschirm Texte eingegeben werden

6. Abkürzungsverzeichnis

CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
CVM	C Virtual Machine
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JDK	Java Development Kit
JVM	Java Virtual Machine
KVM	Kilobyte Virtual Machine
MIDP	Mobile Information Device Profile
RMI	Remote Method Invocation

7. Abbildungsverzeichnis