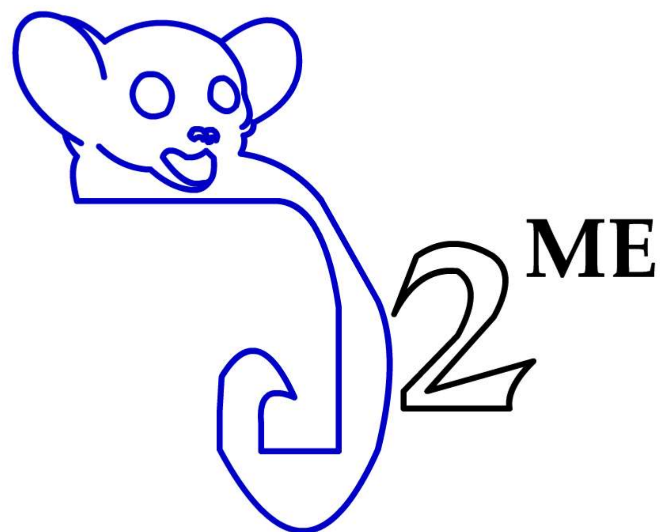
Entwicklungstools für



Ausarbeitung innerhalb des Proseminars Mobile Java (J2ME)

Entwicklungstools für J2ME

Einführung

Bei der Java 2 Platform, Micro Edition, abgekürzt J2ME, handelt es sich um eine Umsetzung der Programmiersprache Java für so genannte »embedded consumer products« wie etwa Mobiltelefone oder PDAs.
Da mit ihr Programme aber noch immer auf dem PC geschrieben und compiliert werden, danach jedoch auf anderen Systemen laufen sollen, werden an die Entwicklungstools bzw. -umgebungen für J2ME erhöhte Ansprüche gestellt.
Im Folgenden möchte ich einen kleinen Einblick in das Thema Entwicklungsumgebungen geben. Anschließend werde ich an einem Beispiel eine Entwicklungsumgebung vorstellen, und darüber hinaus einige Besonderheiten ansprechen, welche bei der Erstellung von Applikationen mit der Micro Edition speziell zu beachten sind.

Allgemeines zu Entwicklungsumgebungen

Unter einer integrierten Entwicklungsumgebung, abgekürzt IDE (engl.: Integrated Development Environment), versteht man eine Applikation, in welcher eine Programmierumgebung, z.B. in Form eines GUI (graphical user interface) builders, eines Text- bzw. Code-Editors, mit einem zugehörigen Compiler, Interpreter und Debugger zusammengefasst sind. Durch das Zusammenführen der verschiedenen Funktionen in einem Programm und zusätzlich eingebundenen Hilfen soll so dem Softwareentwickler die Arbeit erleichtert werden.

Während heutzutage eine Reihe von Entwicklungsumgebungen für die Programmiersprache Java zu Verfügung stehen, ließen die ersten wirklich gut funktionierenden und umfassenderen IDEs (Integrated Development Environment) in den Anfangszeiten von Java (JDK (Java Development Kit) 1.0 bzw 1.1) lange auf sich warten.
Damals konnte man nicht je nach Geschmack zwischen verschiedenen Umgebungen wählen, bzw. beschränkte sich diese Wahl lediglich auf die Auswahl des bevorzugten Texteditors, in welchem man den Javacode schrieb. Anschließend kompilierte man sein als Quellcode vorliegenes Programm mittels der dem JDK mitgelieferten command-line tools.

(Auch uns wurde in den ersten Wochen der Praktischen Informatik I Vorlesung empfohlen, uns auf diesem Wege zuerst mit dem Grundgedanken der Programmiersprachen und ihrer Compiler auseinanderzusetzen, um so besser die Funktionsweise und das Zusammenspiel der einzelnen Programme zu verstehen.)

Gerade das Debugging, also die Fehlersuche und -behebung, ist aber ohne den Einsatz von Entwicklungsumgebungen teils sehr zeit- und folglich letzendlich auch kostenintensiv.
In der heutigen Zeit, und besonders seit dem Erscheinen der Java 2 Platform, haben die IDEs der J2SE (Java 2 Standard Edition) einen Standard erreicht, in dem sie zuverlässig ihre Aufgaben erfüllen und einem somit viel Arbeit ersparen können. Somit kann durch eine IDE nicht nur die Komfortabilität sondern auch die Geschwindigkeit und Übersichtlichkeit des Programmierens stark erhöht werden.

Durch eine (integrierte) Entwicklungsumgebung soll der Anwender also bei der Konfiguration, der Programmierung, der Fehlersuche und der Überwachung seines Systems unterstützt werden.

Spezielle Erfordernisse der Micro Edition von Java 2

Für einen der neuesten Abkömmlinge der Java Familie, die Micro Edition, kurz J2ME, ist das Ganze ein wenig schwieriger. Anders als bei „normalen" Programmiersprachen sind die hier geschriebenen Programme schließlich dafür ausgelegt, auf mobilen Geräten wie Palmtops oder Handys zu laufen.
Somit wird zusätzlich zu Editor und Compiler ein Emulator nötig, der in der Lage ist, solche Mobilen Geräte auf dem PC zu simulieren. Dieser muss sowohl die dafür benötigte, teils abgespeckte, Java-Laufzeitumgebung (KVM – Kilobyte Virtual Maschine) mobiler Geräte, welche aus dem MIDP (Mobile Information Device Profile) und der CLDC (Connected Limited Device Configuration) besteht, berücksichtigen als auch in der Lage sein, die Hardwarebeschränkungen der einzelnen Geräte zu simulieren.
Darüber hinaus sind auch möglichst umfangreiche DEBUG-Möglichkeiten für effizientere Fehlersuche von Nöten.
Außerdem sollte es möglichst einfach sein, ein Bundle der benötigten Dateien zu erstellen, mit welchem die mobilen Geräte umgehen können.

Das J2ME Wireless Toolkit

Für verschiedene mobile Geräte bieten die Hersteller (z.B. Siemens) teils eigene Kits zur Software-Entwicklung an. Diese erhalten allerdings oft Erweiterungen, die speziell auf die eigenen Produkte ausgelegt sind und somit über den MIDP 1.0 bzw 2.0 Standard hinausgehen. Gerade die Verfügbarkeit der optionalen Pakete variiert teils nicht nur unter verschiedenen Geräteherstellern, sondern auch je nach Modell innerhalb einer Produktlinie.
Greifen Applikationen auf optionale Pakete zurück, ist die Portabilität daher in der Praxis eingeschränkt.
Um möglichst sicherzustellen, dass die geschriebenen Anwendungen mit den verschiedenen mobilen Geräten kompatibel sind, sollte man somit eher zum direkt von SUN angebotenen J2ME Wireless Toolkit greifen, welches entweder separat benutzt oder in einige der weitverbreiteten IDEs (z.B. das Sun One Studio 4) integriert werden kann.

Auf dieses J2ME Wireless Toolkit möchte ich im Folgenden näher eingehen.

Das J2ME Wireless Toolkit 1.0.4_01 basiert auf der Referenz Implementation der MIDP 1.0 und CLDC 1.0 Spezifikationen. Es beinhaltet eine graphische Benutzeroberfläche, die es ermöglicht MIDlet suites zu erstellen und laufen zu lassen. Die neuste fertig gestellte Version, das J2ME Wireless Toolkit 2.1 Production Release, bietet außerdem noch zusätzliche Features bezüglich der Fernverwaltung über das Mobilfunknetz (Over-the-Air-Provisioning, OTA) und bezüglich HTTPS (hypertext transfer protocol secure), welche beide zur offiziellen MIDP 2.0 Spezifikation gehören. (Zusätzlich unterstützt es inzwischen CLDC 1.1, WMA 1.1 (Wireless Messaging API), MMAPI 1.1. (Multi Media API), als auch die aktuelle J2ME Web Services API (JSR-172).)

Bei der Installation des J2ME Wireless Toolkit bekommt man eine Anzahl von Tools zur Verfügung, die ich jetzt kurz vorstellen möchte:

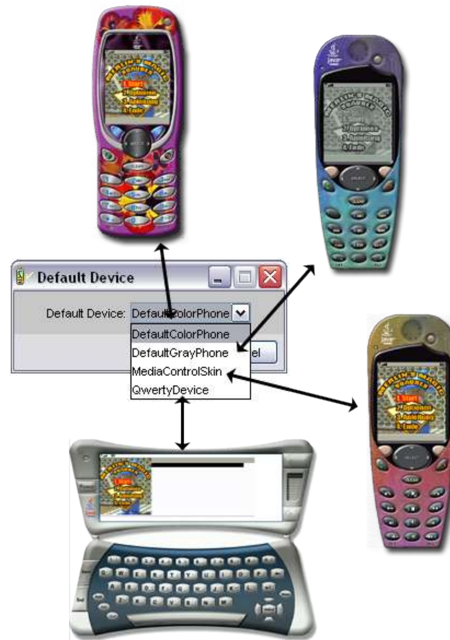
<u>Überblick über das J2ME Wireless Toolkit v 2.1:</u>

<u>1) KToolBar:</u>
Sie ist sozusagen das Hauptprogramm des Toolkits, welches die anderen Features unter einer Applikation vereint. Mit ihrer Hilfe kann man neue Projekte erstellen bzw. bestehende verwalten und kompilieren, MIDlets erzeugen, ausführen und debuggen, aus den Projekt Files Packages erstellen, und Attribute der MIDlet Suites verändern.

<u>2) Run MIDP Application:</u>
Will man nicht das ganze Toolkit starten, sondern lediglich eine Applikation laufen lassen, so verwendet man am besten diese Option. Ein Dialog ermöglicht es einem, auf dem Computer nach MIDlet Suites zu suchen. Diese bestehen aus einem gepackten JAR file und einem damit verknüpften JAD file (MIDP Java Application Descritor). Die Run MIDP Application lässt die ausgewählte Suite mit dem aktuell als Standard eingestellten emulierten Gerät laufen. Äquivalent zum Auswählen per Dialogbox ist (zumindest standardmäßig) der Doppelklick auf ein solches verknüpftes JAD file. Auf den Inhalt dieser Files komme ich später noch zu sprechen.

3) Default device selection

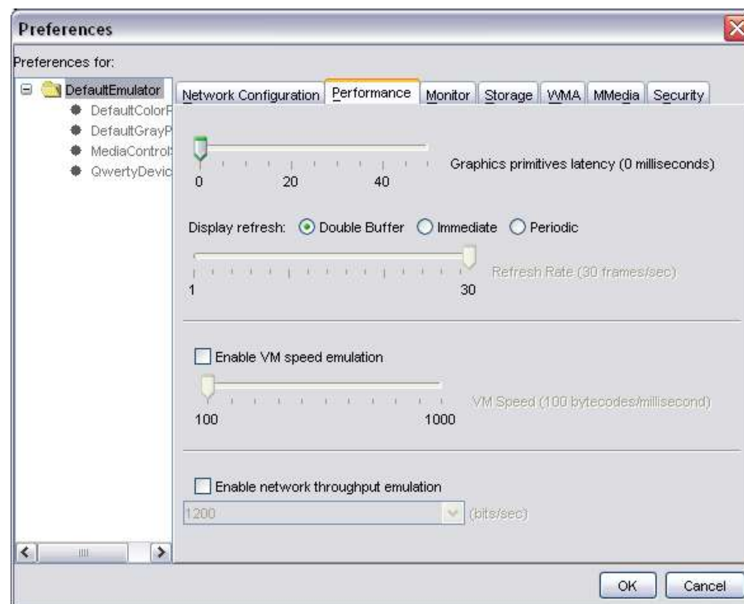
Hier lässt sich das Gerät wählen, welches bei Ausführung einer MIDlet Suite per Run MIDP Application verwendet werden soll.



[Abb. 1: Auswahlmöglichkeit verschiedener Geräte für die Darstellung]

4) Preferences

Hier können verschiedene Einstellungsmöglichkeiten gewählt werden.
So z.B. die Netzwerkadresse, die Http Version, die Leistungsmerkmale des zu simulierenden Geräts, Überwachungsfunktionen bezüglich des Speichers und des Netzwerks, unterstützte Media Formate usw.
Interessant ist auch die Möglichkeit, packet loss zu simulieren, und somit realere Testbedingungen zu schaffen.



[Abb. 2: Einstellmöglichkeiten des Toolkits]

5) Utilities

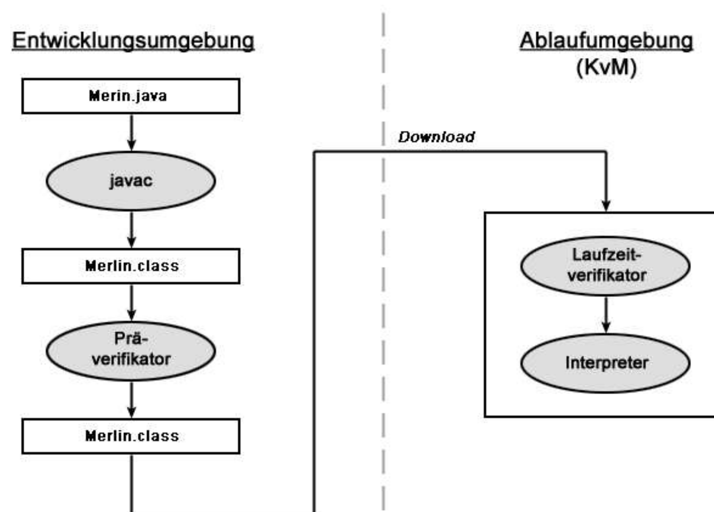
Hier findet man einige verschiedene Optionen, wie das Löschen der Datenbank, das Überwachen von Speicher und Netzwerk, Profilersessions, die WMA Console, die Möglichkeit MIDlets mit Zertifikaten zu signieren und einen Stub Generator, der eine Stub Klasse basierend auf einem Web Services Description Language (WSDL) file erstellt.

All diese graphischen Utilities findet man auch als command-line tools im *bin* directory der Toolkit Installation. Zusätzlich findet man hier auch das MEKeyTool, mit welchem man Zertifikate für HTTPS verwalten kann, sowie das emulator command, welches den Emulator aufruft.

Die Bytecode-Verifikation

Was ist nun zu tun, um von meinem im Editor verfassten Code zu einem auf mobilen Geräten lauffähigen Programm zu kommen?

Grundsätzlich muss der Code natürlich kompiliert werden. Hier kommt es aber zu einem ersten Problem. Wie bereits angesprochen besitzt die Implementierung der KVM einige Einschränkungen gegenüber einer vollwertigen Java Virtual Machine. Einen speziellen Compiler gibt es für sie jedoch nicht, sodass man auf den Standard J2SE-Compiler javac zurückgreift. Eine von diesem erzeugte .class – Datei enthält daher unter Umständen Bytecode, den die KVM später nicht verarbeiten kann.
Um jedoch Bytecodes, die über den Funktionumfang der KVM hinausgehen, in einer .class-Datei herauszufiltern wird der Bytecode-Verifikator verwendet.
Der konventionelle Bytecode-Verifikator der J2SE arbeitet zur Laufzeit direkt auf dem System, auf welchem auch die VM läuft. Für ressourcenbeschränkte mobile Endgeräte ist er aufgrund relativ hoher Anforderung an Speicherplatz (50kb für Binärcode; 30-100kb Arbeitsspeicher) und Rechenleistung ungeeignet.
Daher wurde in der CLDC eine Alternative verwendet, die das mobile Endgerät entlastet.



[Abb. 3; In Anlehnung an Abb. 2-3 aus Quelle 6]

Die Bytecode-Verifikation ist hier in zwei Phasen unterteilt:
Der erste Schritt wird vom sogenannten Präverifikator bereits in der Entwicklungsumgebung getan. Ziel ist es, die zeit- und ressourcenintensiven Überprüfungen bereits hier zu erledigen. Zusätzlich zur Prüfung des Bytecodes nach Aufrufen von VM Features, welche von CLDC nicht unterstützt werden, fügt der Präverifikator auch spezielle Attribute in die .class-Datei ein, die die anschließende Laufzeitverifikation erleichtern. Die so reorganisierten, semantisch jedoch äquivalenten .class-Dateien haben daher einen Größenzuwachs von etwa 5 bis 15%.
Erst jetzt gelangt die Datei in die Ablaufumgebung. Mit nur etwa 10 kB Binärcode und im Durchschnitt benötigten 100 Byte Arbeitsspeicher ist der Bytecode-Verifikator der KVM um einiges schlanker. Er akzeptiert jedoch nur präverifizierte .class-Dateien und stellt sicher, dass vom Code weder das Java-Typsystem verletzt noch auf ungültige Speicherbereiche zugegriffen werden kann.

Die MIDlet-Suite

MIDlets sind die Anwendungen für das MIDP. Abgeleitet von der abstrakten Klasse MIDlet aus der MIDP-Bibliothek handelt es sich ähnlich wie bei Applets um eine gewöhnliche Java-Klasse, in der sogenannte Lebenszyklusmethoden zum Starten, Pausieren und Beenden implementiert sind. Natürlich umfasst eine Anwendung zusätzlich meist weitere Klassen bzw Interfaces.
Ein bzw. mehrere MIDlets bilden eine MIDlet-Suite, welche die kleinste installierbare Einheit ist; Nicht einer Suite zugehörige MIDlets sind für den Zweck der Installation ungeeignet.
Das Bilden von MIDlet-Suites hat folgende Konsequenzen:
Die Rechte die einer MIDlet-Suite eingeräumt werden, z.B. die Verwendung sensibler Stellen der Programmierschnittstellen, werden an die einzelnen MIDlets weitergegeben. Eine genauere Differenzierung ist darüber hinaus nicht möglich.
Auch was die Lese- und Schreibrechte bezüglich der Datenbestände von anderen MIDlets betrifft, kann man innerhalb einer MIDlet Suite Speicher als öffentlich definieren, während MIDlets aus anderen Suites außen vor bleiben, und nicht darauf zugreifen können.
Auch als static vereinbarte Variablen (somit Klassenvariablen) werden von allen MIDlets einer MIDlet-Suite geteilt.
Über letztere zwei Punkte wird den MIDlets einer Suite die Möglichkeit gegeben, sich während der Laufzeit Ressourcen zu teilen und somit untereinander etwa Daten auszutauschen und zu kommunizieren.
Eine MIDlet-Suite besteht aus folgenden zwei Dateien:

- Dem Java Applikations Deskriptor (JAD) File, welches einen festgelegten Satz an Attributen enthält, die es der Application Management Software ermöglichen, die MIDlets zu identifizieren, abzurufen und zu installieren.
  Hier ein Beispiel-Quelltext eines solchen JAD Files:

```
MIDlet-Version: 1.0
MIDlet-Vendor: Alex
MIDlet-Jar-URL: Merlin.jar
MicroEdition-Configuration: CLDC-1.0
MIDlet-1: Merlin'sMagicSquares, , merlin.merlin
MicroEdition-Profile: MIDP-1.0
MIDlet-Jar-Size: 1240843
MIDlet-Name: Merlin
```

- Dem Java Archiv (JAR) File, welches die getrennten und gemeinsamen Java Klassen (in präverifizierter Form) der MIDlets, welche zur Suite gehören enthält. Zusätzlich befinden sich hier auch die Ressourcen z.B. in Form von Bildern oder Audiofiles. Als letztes besitzt das JAR auch eine Textdatei, welche an und für sich (bis auf die fehlende MIDlet-Jar-URL und -Size) dem JAD File entspricht, und ebenfalls Metainformationen in Form von Attributvereinbarungen enthält.
  Hier ein Beispiel-Quelltext eines solchen MANIFEST.MF:

```
Manifest-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MIDlet-Name: Merlin
MIDlet-Vendor: Alex
MIDlet-1:  Merlin'sMagicSquares, , merlin.merlin
MIDlet-Version: 1.0
MicroEdition-Profile: MIDP-1.0
```

Der Vorteil des außerhalb des JAR Files befindlichen JAD-Files ist es, schon vor dem kompletten download einer MIDlet-Suite bestimmte Informationen über deren Anbieter, Filegröße wie die benötigte Systemkonfiguration erhalten zu können.
Der Java Applikations Deskriptor ist nicht verpflichtend zu implementieren. Allerdings sollte man darauf achten, dass es nicht zu widersprüchlichen Angaben innerhalb der beiden Attributvereinbarungen kommt. Während gemäß dem MIDP 1.0 in so einem Fall Angaben des Applikationsdeskriptors Priorität haben, führen Diskrepanzen bei Attributwerten für Trusted MIDlet-Suites mit dem MIDP 2.0 zum Installationsabbruch.

Entwicklungsumgebungen wie die KToolBar oder das Sun ONE Studio 4, Mobile Edition automatisieren das Packen der MIDlet Suites. Um es selbst zu erledigen benötigt man zum Erstellen der JAR Files das J2SE SDK JAR tool sowie einen Text Editor zum Verfassen des JAD-Files.

Debugging

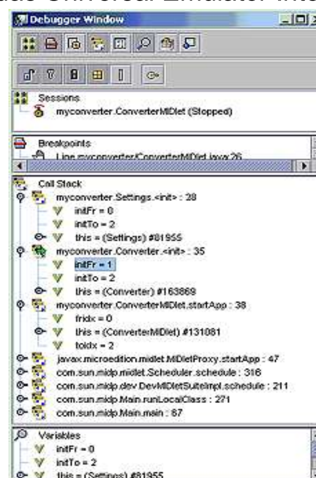
Es kommt leider nur selten vor, dass ein Programm auf Anhieb problemlos läuft. Um die jeweiligen Fehler zu finden und zu beseitigen ist, je größer das Projekt, das Debuggen der Anwendung von großem Nutzen.
Darunter versteht man Folgendes:
Man startet das Programm, behält sich aber die Möglichkeit vor, den Programmablauf zu beobachten, ihn an bestimmten Stellen bzw Zeiten zu unterbrechen und die darauf folgenden Befehle Schritt für Schritt ausführen zu lassen. Zusätzlich will man in der Lage sein, einzelne Variablen bzw Objekte zu beobachten und gegebenenfalls auch die Werte zu modifizieren.
Bevor eine Applikation debugged werden kann, muss sie erst einmal erfolgreich compiliert und ausgeführt worden sein.
Das J2ME Wireless Toolkit besitzt keinen eigenen Debugger, allerdings stellt es einen Proxy bereit, über den die KVM mit einem externen Debugger wie den einer IDE kommunizieren kann. Hierbei kann es teils zu Problemen kommen, falls auf dem System eine Firewall installiert ist, welche den gewählten Port sperrt. Generell ist es hierbei jedem freigestellt, für welche Portnummer er sich entscheidet, solange es sich dabei um einen freien Port handelt.
Um mittels der KToolbar eine Anwendung zu debuggen geht man folgendermaßen vor:
Über den Menüpunkt Project -> Debug gelangt man zu einem Dialog, der die Eingabe der TCP/IP Port Nummer fordert, auf welche der externe Debugger die Verbindung zum Emulator aufzubauen versucht.

[Abb. 4: Wireless Toolkit Portauswahl bzgl. Debug]

Nach Klick auf Debug startet der Emulator im Debug Mode und wartet auf eine eingehende Verbindung von einem Debugger. Das generierte .class File besitzt zusätzlich Informationen fürs Debugging, diese werden allerdings später bei Generierung des JAR Files wieder entfernt um die Größe des Files zu reduzieren.
Jetzt ist es an der Zeit einen Remote Debugger, sprich einen Debugger der im Gegensatz zum lokalen Debug in der Lage ist über TCP/IP eine Verbindung herzustellen, im Remote Modus unter Verwendung des gleichen Ports zu starten.

Sun ONE Studio Mobile Edition

Einfacher ist es allerdings direkt eine IDE wie zum Beispiel das Sun ONE Studio Mobile Edition zu verwenden. Diese frei verfügbare Version des Sun ONE Studios ist eine Entwicklungsumgebung in welches bereits eine Version des J2ME Wireless Toolkits implementiert ist. Sie beinhaltet die volle Java Platform Debugger Architecture (JPDA), die es Entwicklern ermöglichst, Anwendungen mit Emulatoren von mobilen Endgeräten zu debuggen, solange diese das Universal Emulator Interface (UEI) implementiert haben.



[Abb. 5: Darstellung der Variablenbelegungen im Debugger Window (Quelle 3; Figure 18)]

Das Sun ONE Studio - frühere Versionen wurden unter den Namen Forte und NetBeans angeboten - bietet reichhaltige Möglichkeiten bezüglich Breakpoints bzw Breakpointbedingungen, Beobachtern von Variablen bzw Ausdrücken und dem Manipulieren von deren Werten.

Quellenangaben

Gedruckte Form:

1) J2ME in a Nutshell: A Desktop Quick Reference ;O'REILLY; Kim Topley

Internetquellen:

2) Technical Articles & Tips
Debugging Wireless Applications with Mobile Edition: How to code a MIDP Application Using Sun ONE Studio Mobile Edition (Part 1 of 2); Beth Stearns
http://developers.sun.com/prodtech/javatools/jsstandard/reference/techart/WirelessApps.html

3) Technical Articles & Tips
Debugging Wireless Applications with Mobile Edition: How to Debug a MIDlet Application Using Sun ONE Studio Mobile Edition(Part 2 of 2); Beth Stearns
http://developers.sun.com/prodtech/javatools/jsstandard/reference/techart/WirelessApps2.html

4) Technical Articles and Tips
Introduction to Sun ONE Studio Mobile Edition: Installation, Features, and MIDlet Example; Vaughn Spurlin
http://developers.sun.com/prodtech/javatools/jsstandard/reference/techart/introductiontoME.html

5) Technical Articles and Tips
J2ME Wireless Toolkit: Using Wireless Toolkit Version 1.0.4_01 with Sun ONE Studio 4 Update 1 Mobile Edition; Anatole Wilson
http://developers.sun.com/prodtech/javatools/jsstandard/reference/techart/WirelessToolkit.html

6) Java 2 Micro Edition: Entwicklung mobiler Anwendungen mit CLDC und MIDP; Klaus-Dieter Schmatz
Kapitel 2: http://www.dpunkt.de/leseproben/3-89864-271-2/Inhalt.pdf