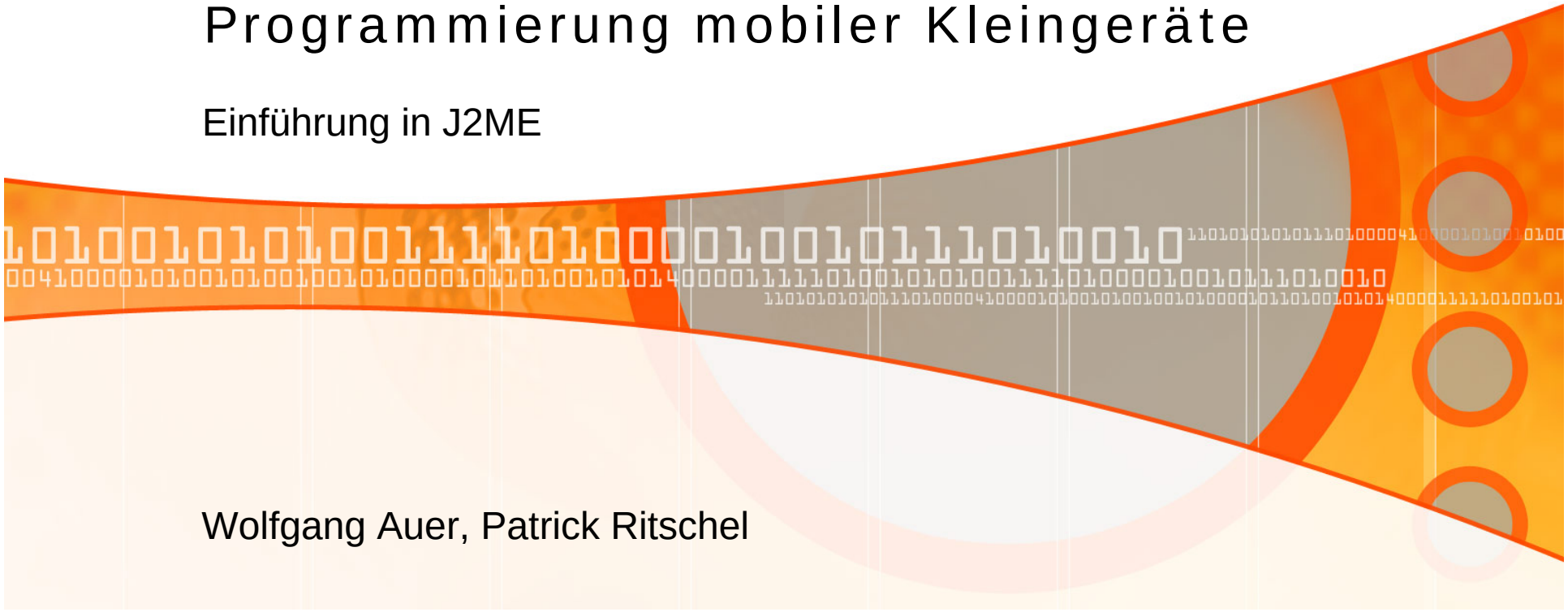
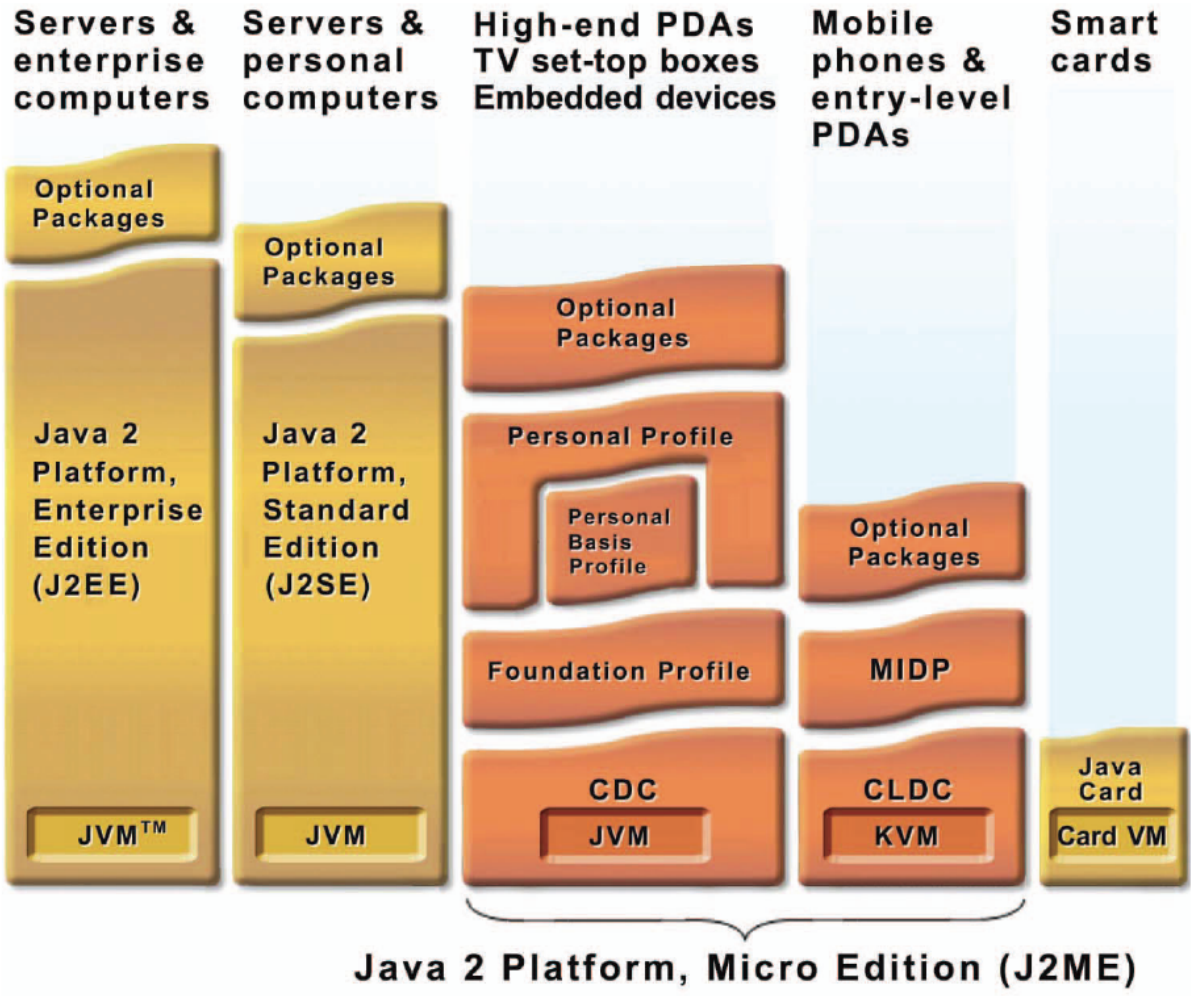
Programmierung mobiler Kleingeräte

Einführung in J2ME



Wolfgang Auer, Patrick Ritschel

Java 2 Plattformen



Quelle: Sun, Datasheet Java 2 Platform, Micro Edition

Konfigurationen
horizontale Strukturierung

- *Konfigurationen* gruppieren Geräte **horizontal** anhand Minimalanforderungen hinsichtlich Speicher, Prozessor und Konnektivität

- Eine Konfiguration definiert:
  - Spezifikation der virtuellen Maschine
  - Obligatorische Pakete

CDC und CLDC

- *CDC 1.0 (JSR 36)*: Connected Device Configuration
  - Spezielle Embedded Geräte
  - High-end PDAs
- *CLDC 1.1 (JSR 139)*: Conneced, Limited Device Configuration
  - Mobiltelefon
  - Low-end PDAs
  - SmartPhones

Abgrenzung CDC und CLDC (1)

Abgrenzung zwischen CDC und CLDC ist fließend $\Rightarrow$ Kategorisierung anhand der vorhandenen Ressourcen

- Speicher für Java + Applikationen
  - CLDC: ROM 160 KB, RAM 32 KB
  - CDC: > 2MB
- Prozessor
  - CDC: 32-bit
  - CLDC: 16-bit oder 32-bit

Abgrenzung CDC und CLDC (2)

- Stromversorgung und - verbrauch
- Größe des Bildschirms
- Kommunikation
  - CDC:
    - ständig vorhanden
    - hohe Bandbreite
  - CLDC:
    - nicht ständig vorhanden
    - geringe Bandbreite

Profile (1)
vertikale Strukturierung

- Problemstellungen innerhalb einer Konfiguration
  - Große Vielfalt an Gerätetypen und Hardwarekonfigurationen
  - Usablity-Unterschiede
    - Tastatur-, Stylus-, Sprachsteuerung
    - Größe und Art des Bildschirms
- Einteilung innerhalb einer Konfiguration nach **vertikalen** Marktsegmenten $\Rightarrow$ *Profile*

Profile (2)
CDC

- Foundation Profile 1.0 (JSR 46)
  - Zusätzliche J2SE-Klassen, allerdings keine GUI-Klassen
  - Stellt Basis für erweitere Profile dar
- Personal Basis Profile 1.0 (JSR 129)
  - Erweiterung des Foundation Profile mit entsprechenden GUI-Klassen
- Personal Profile 1.0 (JSR62)
  - Ersatz von PersonalJava

Profile (3)
CLDC

- MIDP 2.0 (JSR 118) (Mobile Information Device Profile)
- Information Module Profile 1.0 (JSR 195)

CLDC - High-level Architektur

*CLDC design goals: Highly portable, minimum footprint Java **application development platform** for resource-constrained, connected devices*

- Aufwärtskompatibilität zu J2SE
  - Bibliothek setzt sich aus einer Untermenge von J2SE-Klassen und CLDC- spezifischen Klassen (z.B. Generic Connection Framework) zusammen
- Fähigkeit zur parallelen Ausführung von mehreren Java-Applikationen ist nicht vorgeschrieben
- Spezielle Anforderungen hinsichtlich Sicherheit (Pre-verifikation, Class-loading, ..)
- CLDC setzt kein Dateisystem voraus

Java - Sprachspezifikation

- CLDC 1.1 unterstützt Java Sprachspezifikation mit folgenden Einschränkungen:
  - Objekt-Finalization wurde weggelassen ⇒ Vereinfachung der Aufgaben des *Garbage Collectors*
  - Laufzeitfehler: CLDC definiert nur für `Error`, `OutofMemoryError` `VirtualMachineError`, `ClassDefinitionNotFound` das Verhalten. Die Behandlung aller anderen Fehler ist VM-Implementierungsabhängig:
    - Wiederaufsatz ist gerätespezifisch
    - Overhead für die VM-Implementierung
  - Keine asynchronen *Exceptions*

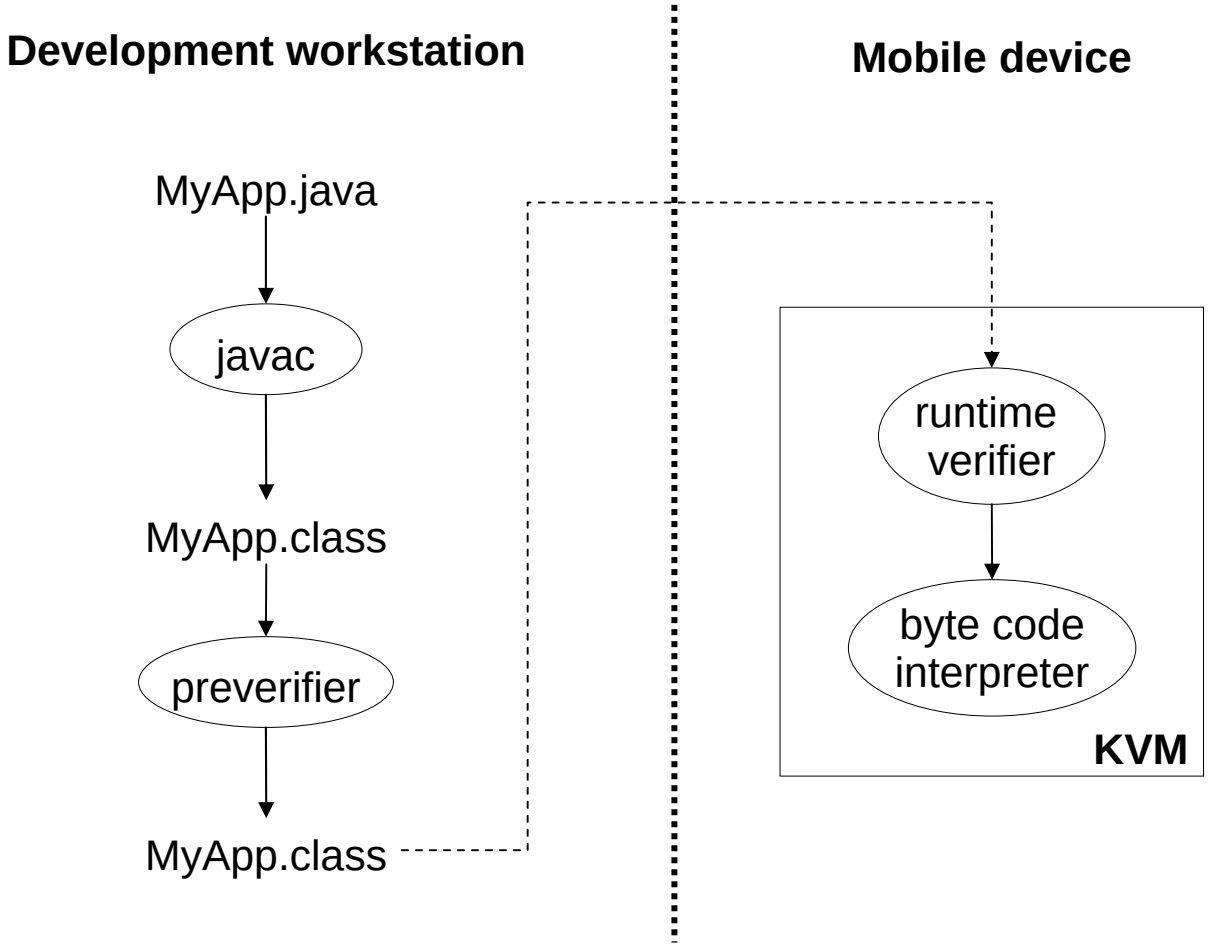
JVM - Spezifikation

- CLDC 1.1 unterstützt JVM Sprachspezifikation mit folgenden Einschränkungen
  - Keine benutzerdefinierten *class loaders*
  - Keine *thread groups* und *daemon threads*
  - Keine Objekt-Finalisierung
  - Eingeschränkte Fehlerbehandlung und keine *asynchronen Exceptions*
  - class file verification
    - Aufteilung in einen Pre-verifier und einen Runtime-Verifier
  - Kein *JNI* und *Reflection*
    - Keine Serialization
    - Kein Standard Debug-Support, profiling interface

Sicherheit
Überblick

- Verifikation des Bytecodes
- Spezielle Anforderungen hinsichtlich des Ladens von Klassen
- Kein JNI bzw. nur eine vordefinierte Menge an Operationen

Sicherheit
Class file verification (1)



Sicherheit
Class file verification (2)

- Aufgaben des *Pre-verifiers*
  - Erstellen von *StackMap*-Attributen, die vom *Runtime-Verifier* verwendet werden (z.B. Typ der lokalen Variablen)
  - Inlining von subroutines
    - Ersetzen aller `jsr`, `jsr_w`, `ret` und `wide ret` bytecodes
- Aufgaben des *Runtime-Verifiers*
  - Prüfen auf Typsicherheit
  - Prüfen auf Speicherkorruption

Sicherheit
Class File Format und Class-loading

- CLDC liest alle *class-file*-Formate ab 1.1; Nicht verwendete Attribute (deprecated, ...) werden ignoriert
- Weder die Reihenfolge des Ladens von Klassen noch die Handhabung des Klassenpfades sind in CLDC festgelegt
  - Nur *system class loader*
  - *class-file lookup* kann nicht manipuliert werden
  - Eine Applikation kann in Form einer JAR-Datei *deployed* werden
  - Dynamisches Nachladen von Klassen ist nur aus der eigenen JAR-Datei erlaubt
  - Systemklassen und Bibliotheken dürfen nicht manipuliert werden

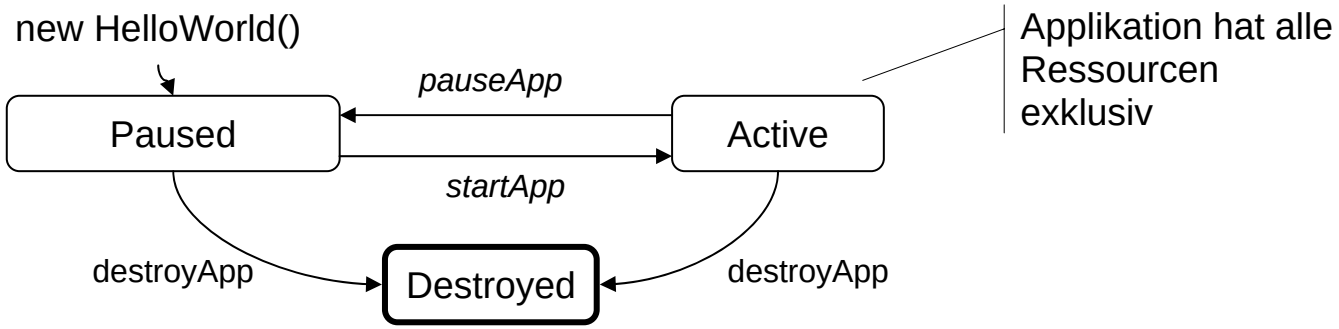
Mobile Infomation Device
Profile *MIDP 2.0*

- Hardwareanforderungen
  - 256 KB ROM, 128 KB RAM, 8 KB persistent
  - Auflösung 96 x 54, 1bit Farbtiefe
- Wichtigste Funktionalität
  - GUI (Event-Handling + Timer) und Zeichnen
  - Persistenter Speicher
  - Konnektivität: HTTP, HTTPS, UDP, TCP und Unterstützung für die serielle Schnittstelle
  - Push-Architektur
  - Sicherheit: Trusted MIDlet-Suites, SSL, WTLS
  - Over-the-air Provisioning

Applikationsmodell
MIDlet

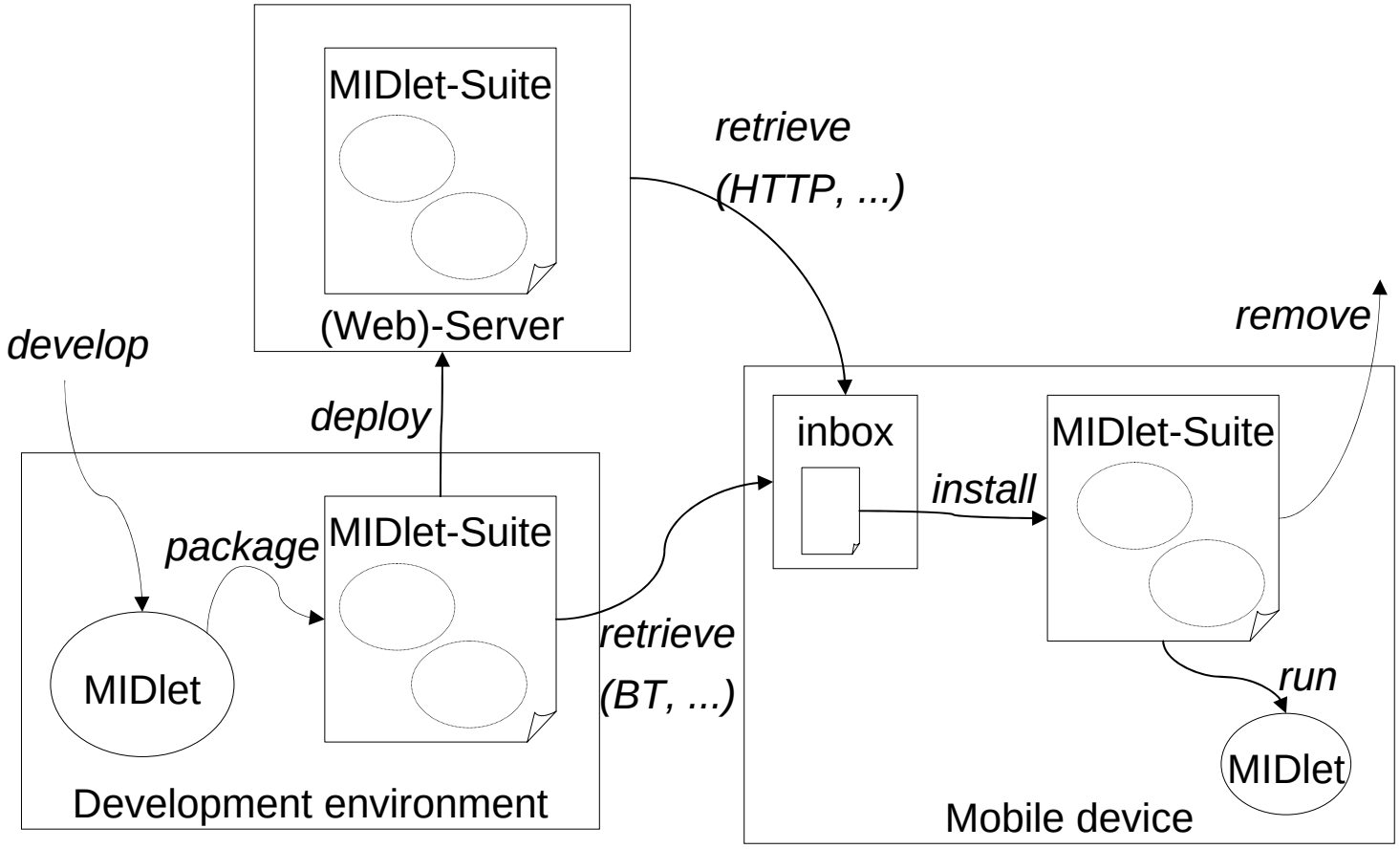

```
public class HelloWorld extends MIDlet {
    private TextBox m_TextBox;

    public HelloWorld() {
        m_TextBox = new TextBox("Hello MIDlet", "Hello World!", 15, 0);
        m_TextBox.addCommand(new Command("Exit", Command.EXIT, 1));
        m_TextBox.setCommandListener(new CommandListener() {
            public void commandAction(Command cmd, Displayable arg1) {
                try {
                    destroyApp(false);
                } catch (MIDletStateChangeException e) {
                    e.printStackTrace();
                }
                notifyDestroyed();
            }
        });
    }

    protected void startApp() throws MIDletStateChangeException {
        Display.getDisplay(this).setCurrent(m_TextBox);
    }

    protected void pauseApp() {}

    protected void destroyApp(boolean arg0) throws
         MIDletStateChangeException {}
}
```

Applikationsmodell
MIDlet - Zustandsdiagramm



- *Paused*:
  - Das MIDlet sollte in diesem Zustand möglichst wenig Ressourcen halten
  - Asynchrone Nachrichten können empfangen werden
  - Suspendierung der JVM bedeutet nicht zwangsläufig, dass die laufende Java-Applikation in den Zustand *paused* kommt!

Applikationsmodell
MIDP Applikation Lebenszyklus



Applikationsmodell
MIDlet Suites

- MIDlet Suite: Sammlung von MIDlets
  - JAR-Datei: class-Dateien, Ressourcen und Manifest mit MIDlet-Beschreibungen (MIDlet Provider, Version, zusätzliche Properties)
  - JAD-Datei: (Application descriptor) Beschreibung der MIDlets
    - Erlaubt das Prüfen der Kompatibilität der MIDlet Suite hinsichtlich des CLDC-Devices
    - Zusätzliche *Properties*, die angepasst werden können
- Sicherheit
  - Klassen können nur aus der Suite geladen werden
  - Austauschen von Daten zwischen MIDlets ist nur innerhalb einer Suite möglich
    - Teilen von statischen Feldern
    - Objekt-Heap
    - Namesraum für persistenen Speicher