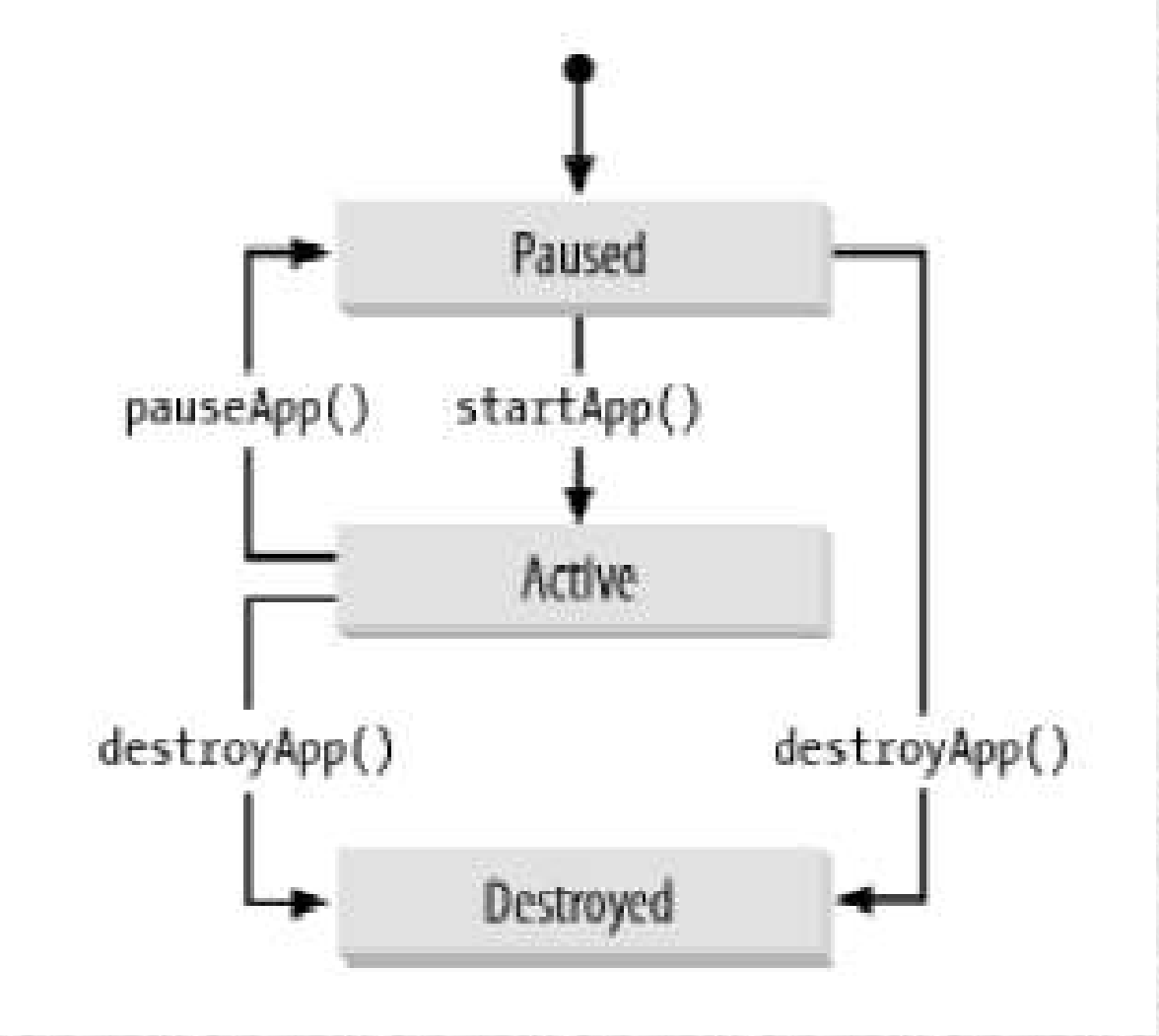
J2ME
Część II

P.J.Podsiadły

Stany aplikacji



Szkielet aplikacji

```
import javax.microedition.midlet.*;
public class MyMIDlet extends MIDlet {

public MyMIDlet() {
// konstruktor
}
public void startApp() {
// przejście w stan aktywny
}
public void pauseApp() {
// stan wstrzymania
}
public void destroyApp( boolean unconditional) {
// koniec aplikacji
}
}
```

Szkielet aplikacji cd.

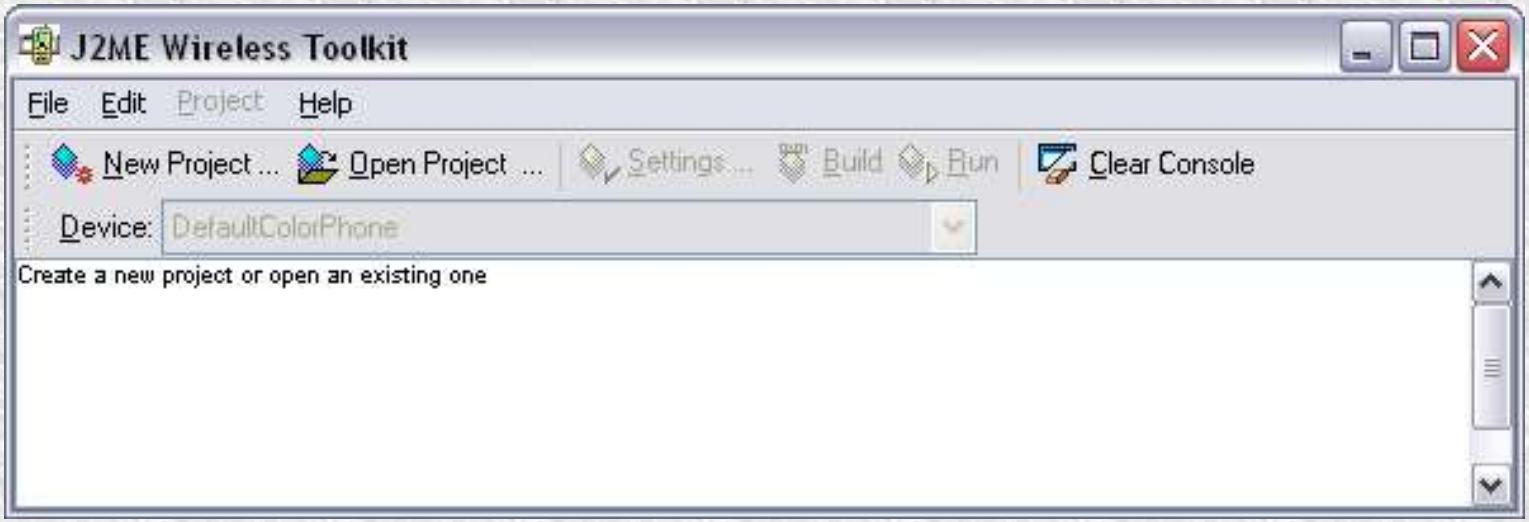
Aplikacja może zmieniać swój stan jeśli jest to potrzebne. Są zdefiniowane 3 metody (w javax.microedition.midlet.MIDlet) do przechodzenia aplikacji między stanami:

`public void notifyPause()` - informuje Java Application Manager-a że aplikacja przechodzi w stan wstrzymania

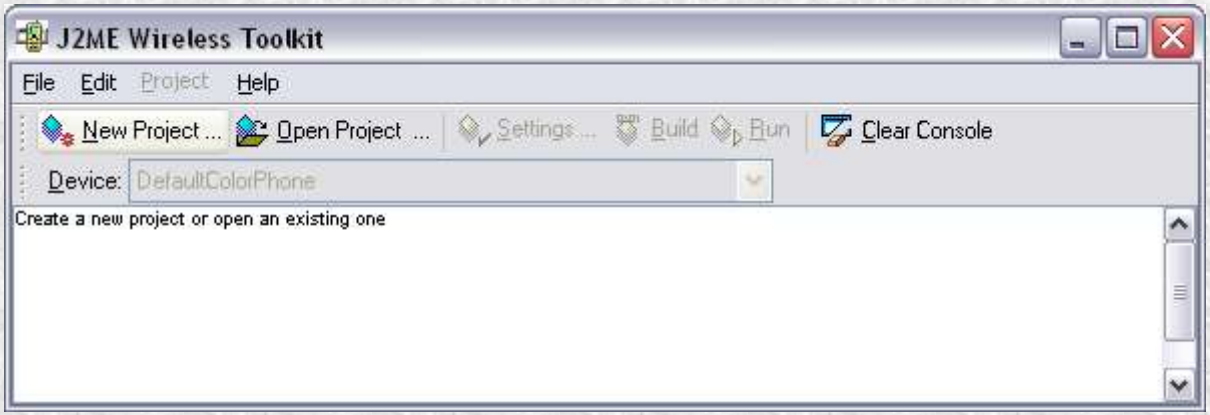
`public void resumeRequest()` - wywołujemy, jeśli jesteśmy zainteresowani przejściem w stan aktywny

`public void notifyDestroyed()` - jeśli chcemy zakończyć aplikację

KToolbar


J2ME Wireless Toolkit
File
Edit
Project
Help
New Project ...
Open Project ...
Settings ...
Build
Run
Clear Console
Device:
DefaultColorPhone
Create a new project or open an existing one

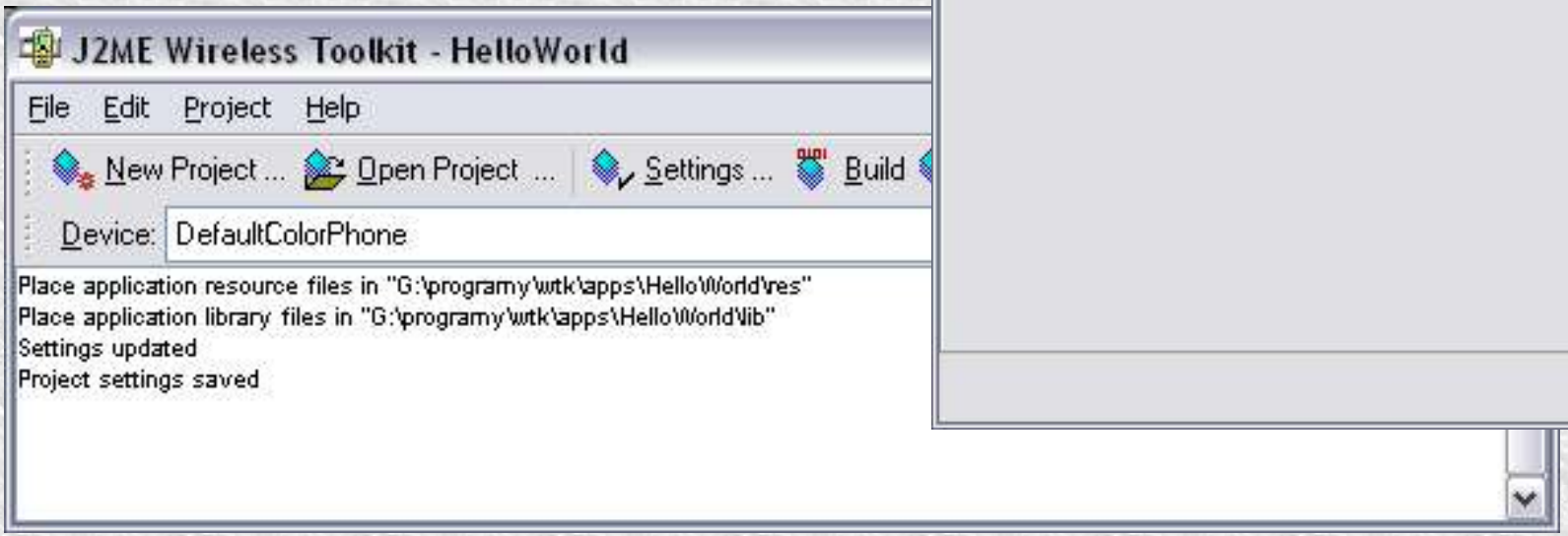
1. Klikamy New Project

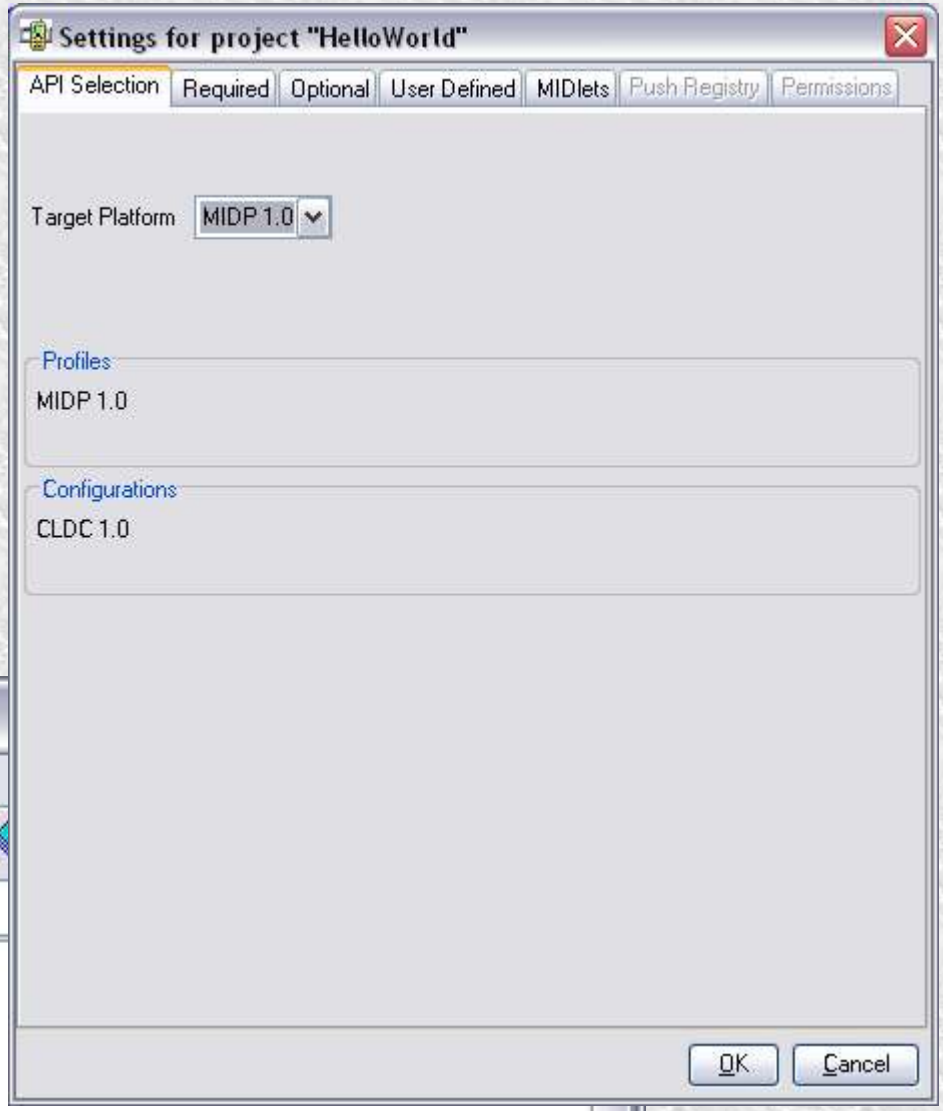


2. Wpisujemy nazwę projektu i klasy a następnie klikamy Create Project

3. Pojawia się okienko z opcjami projektu. Chwilowo ustawiamy Target platform na MIDP 1.0, a resztę pozostawiamy bez zmian.

4. Otrzymujemy komunikat: Project settings saved.

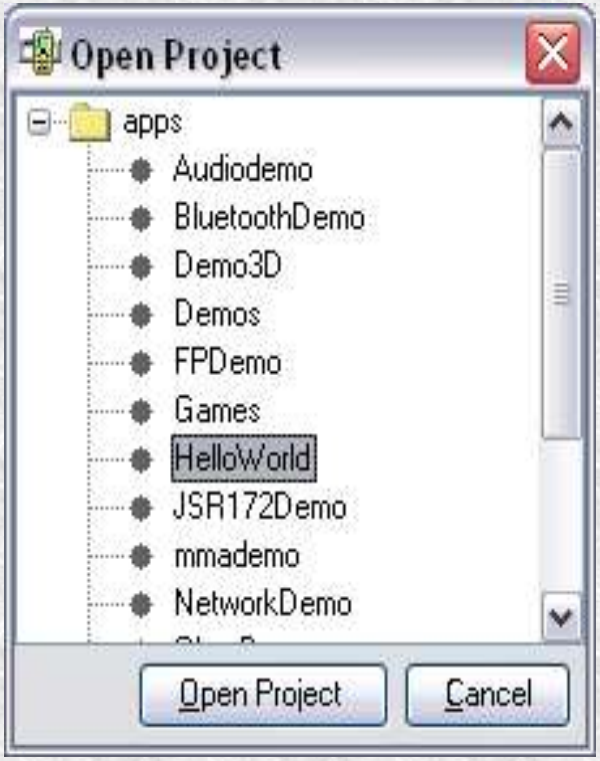




5. Szukamy katlogu w którym zainstalowany jest Wireless Toolkit, następnie wchodzimy do katalogu apps -> HelloWorld -> src
W tym miejscu umieszczamy pliki źródłowe naszego programu.

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

//przykładowe hello world
public class HelloWorldclass extends MIDlet
{
    //display
    private Display display;
    // i text box do wyswietlenia tekstu
    TextBox box = null;

    //konstruktor
    public HelloWorldclass()
    {
    }

    //tutaj rozpoczyna sie wykonywanie programu
    public void startApp()
    {
        display = Display.getDisplay(this);
        box = new TextBox("prosty przyklad", "Hello world", 20,0);
        display.setCurrent(box);
    }

    public void pauseApp()
    {
    }

    //tutaj czyscimy wszystko czego nie wyczysci garbage collector
    public void destroyApp (boolean unconitional)
    {
    }}
```

6. Klikamy Open project i wskazujemy projekt HelloWorld



7. Wybieramy urządzenie docelowe (np. Default gray phone) a nstępnie Run

8. Jeśli w kodzie źródłowym nie było błędów powinniśmy zobaczyć emulator telefonu z wyświetloną listą MIDletów do uruchomienia. Strzałkami wybieramy HelloWorld (jesli mamy więcej niż jeden) i wciśkamy Launch.

9. Aplikację kończymy przy pomocy klawisza z czerwoną słuchawką.

Instalowanie aplikacji

Aplikacje na urządzeniu użytkownika końcowego można zainstalować na 2 sposoby:

1. Korzystając z odpowiedniego kabla do transmisji danych lub innego połącznia (np. IRDa)
2. Korzystając z sieci

Pierwsza metoda jest dobra dla urządzeń PDA, które często są podłączne do komputera w celu synchronizacji danych.

Druga metoda jest lepsza dla telefonów komórkowych. Proces dostarczania MIDletów tą metodą nosi nazwę **OTA** (on-the-air) **provisioning**

Aby skorzystać z tej metody należy:
- dodać nowy typ MIME do konfiguracji serwera www

```
text/vnd.sun.j2me.app-descriptor jad
```

Np. dla serwera **Apache** należy dodać następujący wpis do pliku **web.xml**

```
<mime-mapping>
   <extension>jad</extension>
   <mime-type>text/vnd.sun.j2me.app-descriptor</mime-type>
</mime-mapping>
```

- na stronie www (lub WAP) umieścić odpowiedni link:

```
<a href="HelloWorld.jad">Kliknij aby zainstalować MIDlet</a>
```

Przykładowy plik JAD

```
MIDlet-1: HelloWorld, HelloWorld.png, HelloWorldclass
MIDlet-Jar-Size: 100
MIDlet-Jar-URL: HelloWorld.jar
MIDlet-Name: HelloWorld
MIDlet-Vendor: Unknown
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
```

UWAGI: Pliki JAD są case-sensitive, nazwa pliku JAD i JAR moze mieć max 16 znaków z rozszerzeniem (.JA?), Długość pliku JAR umieszczona w pliku JAR musi być dokładna

Oprócz pliku JAR potrzeby jest także plik Manifest.mf (znajduje się w tym samym katalogu) oba pliki muszą mieć jednakowo ustawione atrybuty:

MIDlet-Name, MIDlet-Version, MIDlet-Vendor

jęsli będą one różne, telefon nie zainstaluje MIDletu

Budowanie GUI

MDIP GUI posiada zarówno wysoko jak i niskopozimowe API które można wykorzystać we własnych aplikacjach.

Wysokopozimowe API jest przeznczone dla plikacji dla których ważną cechą jest przenośność między rónymi urządzeniami. Ceną za przenośność jest niewielki wpływ na wygląd i zachowanie kontrolek.

Wszytkie klasy które implementują wysokopoziomowe API dziedziczą po klasie javax. microedition.lcdui.Screen

Niskopoziomowe API jest przeznaczone dla aplikacji, w których wymagana jest pełna kontrola nad tym co jest rysowane na ekranie, oraz kontrola nad zdarzeniami niskiego poziomu jak wciśnięcie poszczególnych klawiszy.

To API jest implementowane przez klasy:

`javax.microedition.lcdui.Canvas`
`javax.microedition.lcdui.Graphics`

Nie gwarantuje się, że applety wykorzystujące niskopoziomowe API będą w pełni przenośne, poniewarz mogą wykorzystywać specyficzne właściwości konkretnych urządzeń.

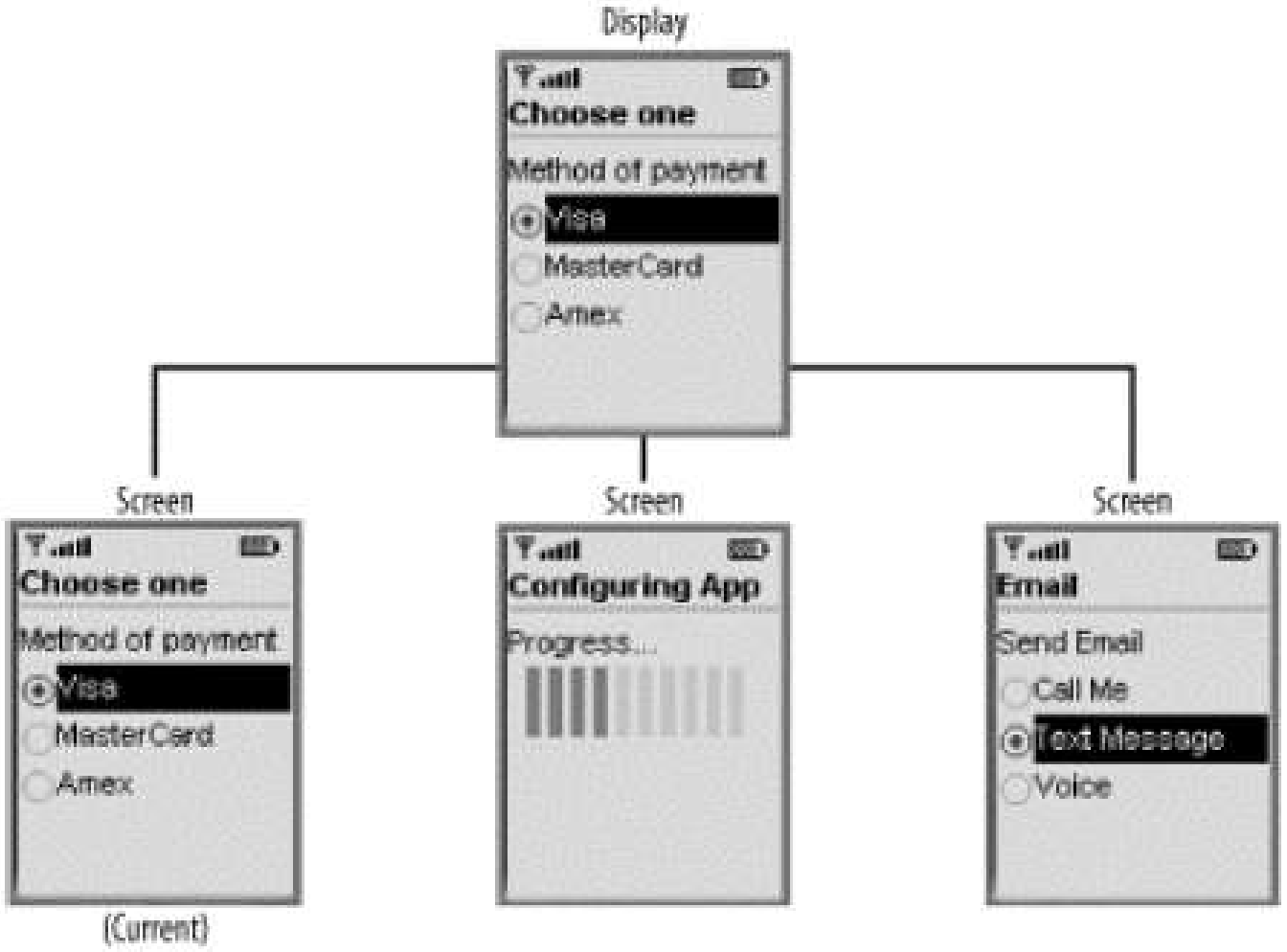
Display i Screen

Aby wyświetlić cokolwiek na ekranie, trzeba uzyskać do niego dostęp.
Ekran urządzenia reprezentowany jest przez klasę Display. Każdy aktywny MIDlet posiada tylko jedna instancję kalsy Display, która zapewnia dostęp do informacji o fizycznych możliwościach ekranu urządzenia.

Obiekty klasy Screen enkapsulują złożone componenty interface-u użytkownika oraz zapewniają kontrolę nad wejściem danych.
Obiekty klasy Screen są wyświtlane przez obiekt klasy Display przy użyciu metody setCurrent().

Każda aplikacja może posiadać kilka obiektów typu screen, ale tylko jednen z nich może być widoczny na ekranie, a użytkownik może się poruszać tylko w obrębie komponentów tego obiektu klasy Screen

Zależność między Display i Screen



lcdui package

Interface	Description
`Choice`	Defines an API for a user interface component that implements a selection from a predefined number of choices
`CommandListener`	Used by applications that need to receive high-level events from implementations
`ItemStateListener`	Used by applications that need to receive events that indicate changes in the internal state of the interactive items

`Alert`	A screen that shows data to the user and waits for a certain period of time before proceeding to the next screen.
`AlertType`	A utility class that indicates the nature of the alert.
`Canvas`	The base class for writing applications that need to handle low-level events and to issue graphics calls for drawing to the display.
`ChoiceGroup`	A group of selectable elements intended to be placed within a `Form`.
`Command`	A construct that encapsulates the semantic information of an action.
`DateField`	An editable component for presenting calendar data and time information that may be placed into a `Form`.
`Display`	A utility that represents the manager of the display and input devices of the system.
`Displayable`	An object that has the capability of being placed on the display.
`Font`	A utility that represents font and font metrics.
`Form`	A screen that contains an arbitrary mixture of items (images, text, text fields, or choice groups, for instance).

`Gauge`	A utility that implements a bar graph display of a value intended for use in a form.
`Graphics`	A utility that provides a simple two-dimensional geometric rendering capability.
`Image`	A utility that holds graphical image data.
`ImageItem`	A utility that provides layout control when `Image` objects are added to a form or alert.
`Item`	A superclass for all components that can be added to a `Form` or `Alert`.
`List`	A screen containing a list of choices.
`Screen`	The superclass of all high-level user interface classes.
`StringItem`	An item that can contain a `String`.
`TextBox`	A screen that allows the user to enter and edit text.
`TextField`	An editable text component that can be placed into a `Form`.
`Ticker`	A ticker-type piece of text that runs continuously across the display. It can be attached to all screen types except `Canvas`.

Display

```
Public class MYMIDlet extends MIDlet
{
   Display display = null;
   List lista = null;
.
.
.
   public void startApp()
   {
      display = Display.getDisplay(this);
      //pozostałe metody
```

Uwaga: getDisplay() powinno być wywołane na początku startApp() a nie w konstruktorze. Wywołanie tej metody w kostruktorze nie gwarantuje poprawnego zaincjalizowania obiektu klasy Display – szczegóły w specyfikacji MIDP

```
        lista = new List (“Przyklad”, Choice.EXLUSIVE);
        lista.append(“pierwszy”, null);
        lista.append(“drugi”, null);
        display.setCurrent(lista);
        //wyświetlamy utworzony element GUI
        }
}
```

Inne metody Display:

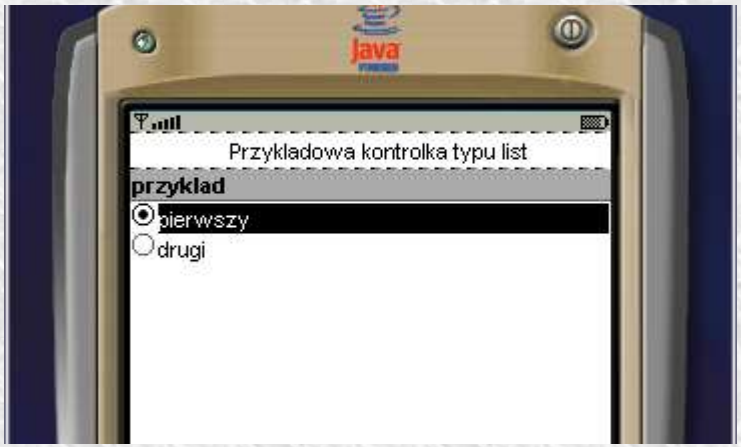
```
public void setCurrent(Alert alert,
                       Displayable d);

public Displayable  getCurrent();
public boolean isColor();
public int numColors();
```

Screen

Obiekt klasy screen może być jedną z następujących kontrolek:
TextBox, Alert, Form, List

Dodatkowo Screen może posiadać tytuł (Title) i Ticker do obsługi których służą następujące metody:

public void setTitle (String title);
public String getTitle();
public void setTicker (Ticker t);
public Ticker getTicker();



TextBox

public TextBox(String title, String text, int maxsize, int constraints);

Constraints:
TextField.ANY,
TextField.EMAILADDR
TextField.NUMBER
TextField.PASSWORD
TextField.PHONENUMBER
TextField.URL

Ograniczenie PASSWD może być użyte z dowolnym innym przy pomocy operatora “|” np.

```
TextBox t = new TextBox (“tel”, “”, 12, TextField.PHONENUMBER |
                                        TextField.PASSWORD);
```


+5550000 - DefaultColorPhone
MIDlet
Help
123
tel
****

Alert

Jest ekranem zawierającym tekst i grafike, zwykle służącym do przekazywania użytkownikowi informacji o błędach itp.

public Alert (String title);
public Alert (String title, String alertText, Image alertImage, AlertType typ);

inne metody:
public int getDefaultTimeout();
public int getTimeout();
public void setTimeout(int i);

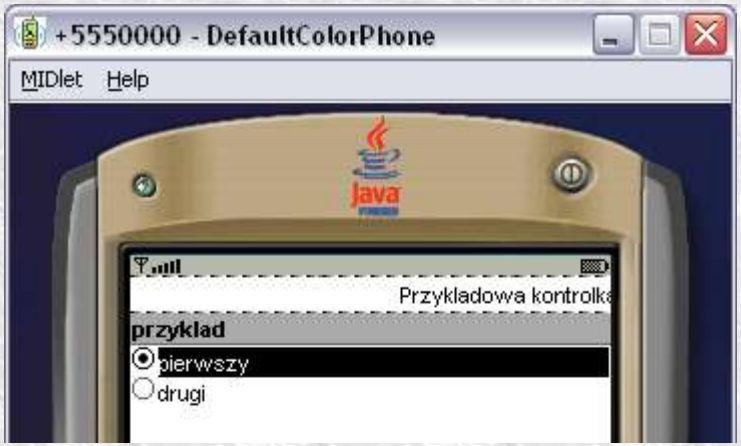
Alert alert = new Alert(“Przyklad 1”);
alert.setTimeout(4000);

Typy Alertów:
AlertType.CONFIRMATION,
AlertType.ALARM,
AlertType.ERROR,
AlertType.INFO
AlertType.WARNING

List

public List (String title, int listtype);
public List (String title, int listtype, String[] elements, Image[] imelements);

typy List
Choice.EXCLUSIVE,
Choice.IMPLICIT,
Choice.MULTIPLE.



Inne metody:
public int append(String element, Image imelemnt);
public void insert (int index, reszta j.w.);
public void set(int index, reszta j.w.);

public void delete(int index);
public int getSelectedIndex();
public boolean isSelected(int index);
public void setSelectedIndex(int index, boolean selected);

public int getSelectedFlags(boolean[] selected);
public void setSelectedFlags(boolean[] selected);

Form


Przykladowy form
przykld
przyklad
pierwszy
drugi
trzeci
czwarty
Glosnosc