




# **Tehnologii avansate de programare**

## ***Curs 12***

Cristian Frăsinaru

`acf@infoiasi.ro`

Facultatea de Informatică  
**Universitatea "Al. I. Cuza" Iași**





# Platforma Java 2 Micro Edition





# Cuprins

---

- Introducere
- Arhitectura generală J2ME
- Crearea unui MIDlet simplu
- Dezvoltarea suitelor de midlet-uri
- Interfața grafică cu utilizatorul
- Tratarea evenimentelor
- Crearea jocurilor (Game API)
- Asigurarea persistenței datelor (RMS)
- Distribuția unei suite de midlet-uri

# Introducere

# Comunicații *Wireless*

- Domeniu extrem de vast
- Comunicațiile pot fi de naturi diferite:
  - Radio-Televiziune
  - Telefonie mobile
  - ...
- Comunicațiile pot fi:
  - *locale*
  - *la distanță*
- Există o serie protocoale standard
- Complexitate, fragmentare

# Ce este J2ME ?

J2ME reprezintă un set de tehnologii și specificații concepute pentru dispozitive mobile cu capacități limitate, cum ar fi pagere, telefoane mobile, sisteme de navigare, PDA-uri, etc.

J2ME aduce conceptul de *portabilitate* la nivelul dispozitivelor mobile, oferind un mediu optimizat, bazat pe Java, de dezvoltare a aplicațiilor pentru acest segment software.

# Tehnologii J2ME

- **CLDC - Connected Limited Device Configuration**

CLDC, Mobile Information Device Profile (MIDP), Java Technology for the Wireless Industry (JTWI), Wireless Messaging API (WMA), Mobile Media API (MMAPI), Mobile 3D Graphics, J2ME Web Services APIs (WSA), Bluetooth API (JSR-82, Motorola, Java Partner Site), ...

- **CDC - Connected Device Configuration**

CDC, J2ME RMI, JDBC, ...

- **Java Card**

- **Altele:** Java Telephony API, Java Embedded Server Technology, PersonalJava Technology, Java TV API, JavaPhone API, ...



# JSR + JCP

Specificațiile pentru J2ME (dar și pentru J2EE, J2SE) sunt create sub egida **Java Community Process (JCP)**, primul pas în dezvoltarea acestora fiind o cerere de tip **Java Specification Request (JSR)**, venită din partea comunității Java. Ulterior, acestea sunt analizate de un grup de experți și rafinate până la forma lor finală.

Specificațiile J2ME sunt referite adeseori sub forma numărului corespunzător JSR, experții care contribuie la standardizarea lor fiind firme ca Sun, Nokia, Motorola, Ericsson, Fujitsu, Siemens, Samsung, Vodafone, France Telecom, Orange, etc.

# J2ME Wireless Toolkit

- Utilitar pentru dezvoltarea de aplicații wireless bazate pe CLDC și MIDP (Mobile Information Device Profile)
- Include o serie de emulatoare pentru diverse medii de execuție
- Include API-ul necesar dezvoltării de aplicații, documentație, exemple, etc.
- Cel mai bun produs în 2003 conform Developer.com ("Wireless Development Product of the Year")
- Este gratuit

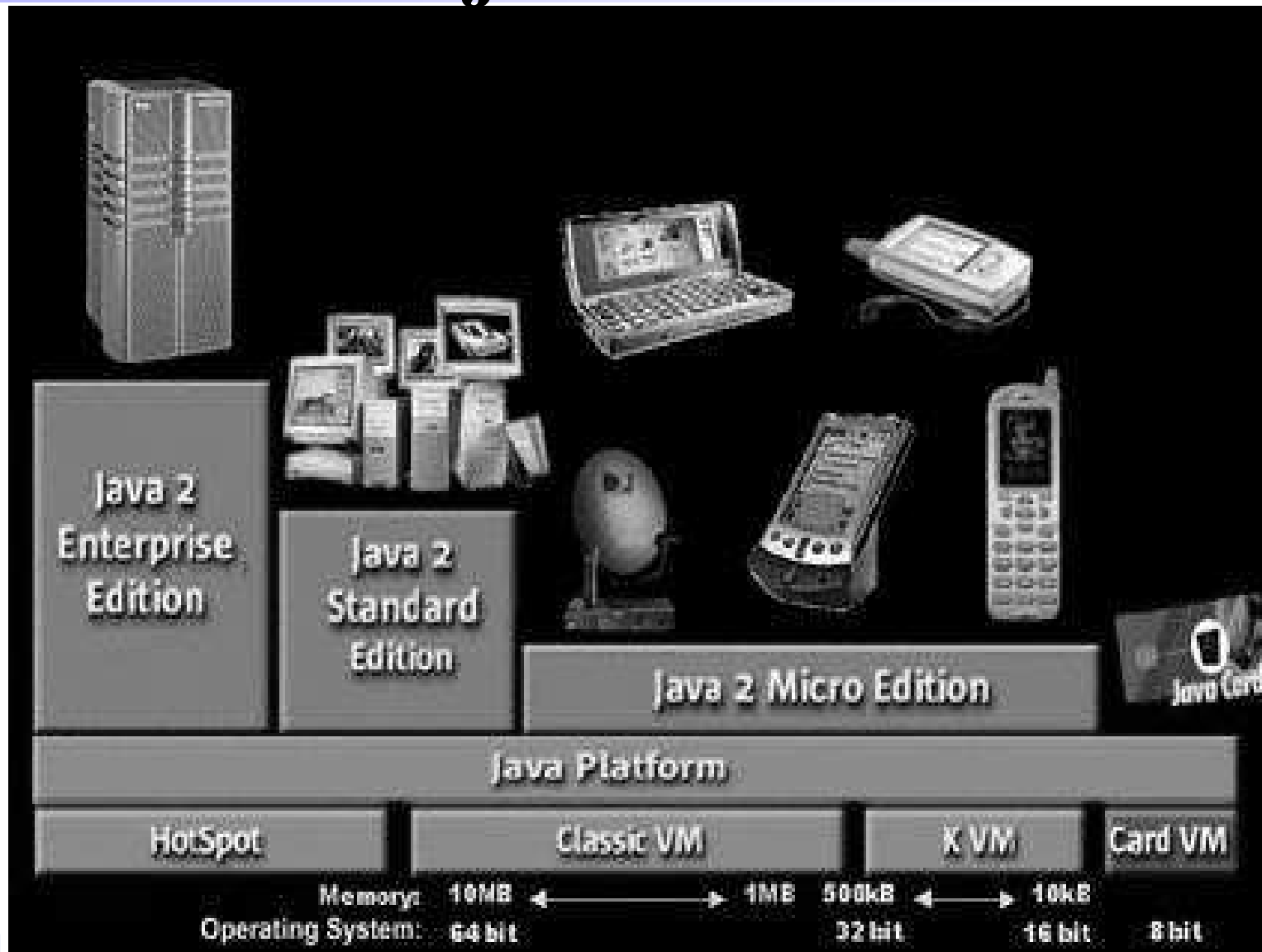
[http://java.sun.com/products/j2mewtoolkit/download-2\\_2.html](http://java.sun.com/products/j2mewtoolkit/download-2_2.html)



# Arhitectura generală J2ME



# Platforme java



# Configurații și profile

J2ME are o arhitectură **modulară** și **ierarhică**:

- **Configurații**

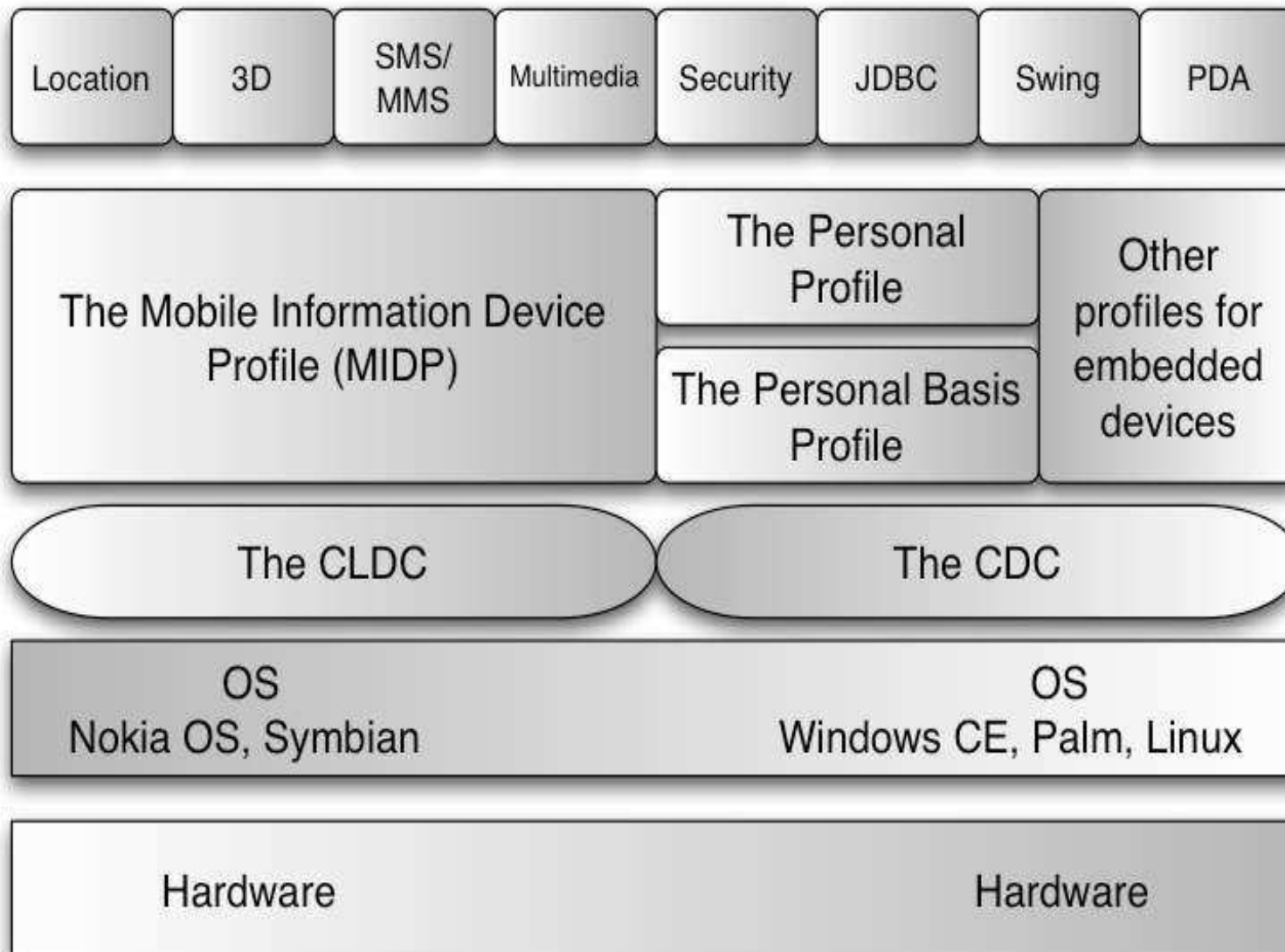
Specificații ce detalizează o mașină virtuală Java și un API restrâns ce stă la baza dezvoltării de aplicații pentru o anumită categorie de dispozitive.

- **Profile**

Profilele sunt atașate configurațiilor, adăugând API-ul specific necesar creării efective a unei aplicații.

- **Pachete opționale**

# Privire generală





# Dezvoltarea de aplicații wireless



# Categorii de aplicații

- **Aplicații locale (independente)** - nu necesită surse externe de date sau comunicații prin rețea (jocuri, programe utilitare: agendă, calculator, etc)
- **Aplicații de rețea** - conțin atât componente locale cât și la distanță, aflate pe servere Web (client de email).



# Conșiderații generale

Crearea unei aplicații *performante* J2ME este o "artă" ce trebuie să țină cont de:

- Memoria (volatilă sau persistentă) este limitată drastic
- Procesoare lente
- Lipsa unui sistem de fișiere clasic
- Limitări, eventual severe, ale conectivității
- Modalitatea de instalare a aplicației
- ...

# Diferențe față de J2SE

- Nu sunt permise operații în virgulă mobilă
- Nu mai există finalizarea obiectelor
- Diferențe în modul de tratare a excepțiilor
- Nu există suport pentru JNI, introspecție, serializare
- Nu există grupuri de fire de execuție, demoni.
- Nu pot fi definite class loader-e proprii
- Verificarea claselor este realizată diferit (după compilare are loc etapa de *preverificare*)

# MIDlet-uri



Aplicații create pentru:

- Configurația CLDC (Connected Limited Device Configuration)
- Profilul MIDP (Mobile Information Device Profile)

## Software necesar

- JDK 1.3+
- Wireless Toolkit



# Suite de midlet-uri

Midlet-urile sint distribuite sub forma unor **suite**. Acestea sunt formate din unul sau mai multe midlet-uri, arhivate împreună care:

- fie oferă o serie de funcții similare (cum ar fi o colecție de jocuri), sau
- fie lucrează împreună, fiind module ale unui sistem mai complex
- au acces la aceleași resurse (imagini, date, etc.)
- pot comunica prin setarea/obținerea unor informații dintr-o zonă comună de memorie

# MIDlet API

## ● Pachete CLDC

- `java.lang`  $\subset J2SE$
- `java.io`  $\subset J2SE$
- `java.util`  $\subset J2SE$
- `javax.microedition.io`

## ● Pachete MIDP

- `javax.microedition.midlet`
- `javax.microedition.lcdui`
- `javax.microedition.rms`
- ...

# Crearea unui midlet simplu (1)

1. Crearea unui proiect nou: → New Project :  
Hello

```
Wtk22/apps/Hello
[bin]
[lib]
[res]
[src]
project.properties
```

2. Scrierea surselor
3. Compilarea: → Build
4. Execuția (testarea): → Run
5. Distribuția

# Scrierea surselor

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

public class Hello extends MIDlet implements CommandListener {
    public void startApp() {
        Display display = Display.getDisplay(this);
        Form mainForm = new Form("Hello");
        mainForm.append("Hello world!");
        Command exitCommand = new Command("Exit", Command.EXIT, 0);
        mainForm.addCommand(exitCommand);
        mainForm.setCommandListener(this);
        display.setCurrent(mainForm);
    }
    public void pauseApp () {}
    public void destroyApp(boolean unconditional) {}
    public void commandAction(Command c, Displayable s) {
        if (c.getCommandType() == Command.EXIT)
            notifyDestroyed();
    }
}
```

# Compilarea și execuția

- **Compilarea** presupune:

- Apelul compilatorului **javac** din kitul standard Java.
- Preverificarea claselor rezultate cu utilitarul **preverify** din `Wtk22/bin`.

În urma compilării va fi creat directorul `classes`.

- **Execuția**

Va fi realizată folosind un **emulator**. Acesta va afișa lista midleturilor disponibile în suita selectată.



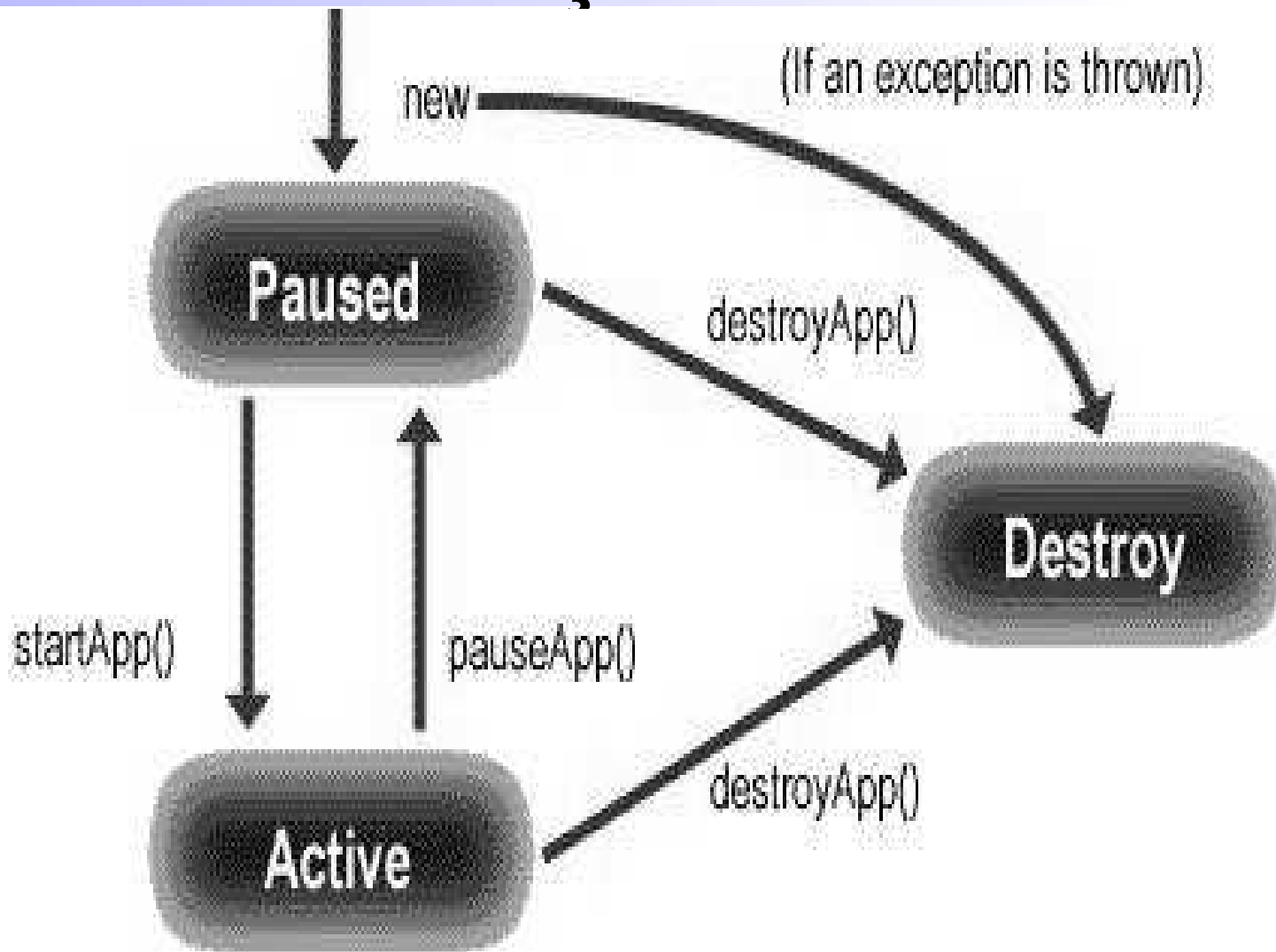


# Dezvoltarea suitelor de midlet-uri

# Modul de lucru

- Un midlet este format din unul sau mai multe **ecrane (screens)**. Acestea pot fi
  - **structurate** - au o anumită funcționalitate, sunt portabile, nu permit accesul direct la acran sau la mecanisme *low-level*.
  - **nestructurate (canvas-uri)** - permit controlul ecranului la nivel de pixel, tratarea evenimentelor generate de apăsarea tastelor, etc.
- Fiecare ecran are asociată o serie de **comenzi (abstract commands)**. Acestea permit utilizatorului navigarea între ecrane și execuția unor acțiuni specifice.

# Ciclul de viață



# Structura unui midlet

```
public class MyMIDlet extends MIDlet {
    public MyMIDlet() {
        // Crearea de ecrane
        // Crearea si asocierea de actiuni ecranelor
        // Stabilirea claselor de tip listener pentru actiuni
    }
    public void startApp() {
        // Setarea eranului initial/curent
        // Pornirea unor fire de executie
    }
    public void pauseApp () {
        // Eliberarea temporara a unor resurse
        // Oprirea unor fire de executie
    }
    public void destroyApp(boolean unconditional) {
        // Eliberarea tuturor resurselor
        // Salvarea starii persistente a aplicatiei
    }
}
```

# Interfața grafică

Interfața grafică cu utilizatorul a fost implementată ținând cont de particularitățile fizice ale dispozitivelor mobile. Specificațiile CLDC nu au nici o referire la GUI, ele fiind definite în MIDP.

Pachetele care oferă suport pentru crearea GUI:

- **javax.microedition.lcdui**
- **javax.microedition.lcdui.game**

Gestiunea ecranului propriu-zis și a celorlalte mecanisme fizice accesibile ale dispozitivului se face prin intermediul clasei **Display**, aceasta fiind responsabilă și de gestiunea screen-urilor aplicației.

# Clasa *Displayable*

Este supeclasa abstractă a ecranelor, definind funcționalitatea comună a acestora:

- pot avea un titlu
- pot avea un *ticker* (un text ce se deplasează în mod continuu pe ecran)
- pot avea zero sau mai multe comenzi
- pot avea un `CommandListener`

Implementările de bază ale acestei clase sunt:

- **Screen** - ecrane structurate
- **Canvas** - ecrane nestructurate

# Clasa *Screen*

Este superclasa abstractă a ecranelor structurate, implementările directe fiind:

- **Alert** - Mesaje de avertizare, erori, etc.
- **Form** - Permite crearea de ecrane compuse din alte componente, cum ar fi imagini (`Image`) sau articole derivate din `Item`:

`ChoiceGroup`, `CustomItem`, `TextField`, `Gauge`,  
`ImageItem`, `Spacer`, `StringItem`, `TextField`

- **List** - Ecran pentru selectarea unei opțiuni dintr-o mulțime prestabilită.
- **TextBox** - Ecran pentru introducerea unui text.

# Crearea form-urilor

Articolele unui ecran `Form` sunt referite prin **indexul** lor (un număr între 0 și `size() - 1`), gestiunea lor fiind realizată prin intermediul metodelor: **append**, **insert**, **set**, **delete**.

Poziționarea articolelor se face pe linii, după un algoritm prestabilit. În cazul în care articolele nu încap în ecran, va fi creată automat o bară verticală de derulare.

```
mainForm.append("Introduceți datele:");  
mainForm.append(new TextField("Nume", "", 50, TextField.ANY));  
mainForm.append(new DateField("Data", DateField.DATE));
```

Tratarea evenimentelor generate de modificare conținutului unui articol se face prin implementarea interfeței **ItemStateListener**.



# Interfața *ItemStateListener*

Metoda interfeței este: **itemStateChanged(Item item)**.

```
Form mainForm = new Form("Hello");
StringItem s = new StringItem(null, "Hello world");
TextField tf = new TextField("Nume", "", 50, TextField.ANY);

...

mainForm.append(s);
mainForm.append(tf);

mainForm.setItemStateListener(new ItemStateListener() {
    public void itemStateChanged(Item item) {
        s.setText("Hello " + tf.getString());
    }
});
```

# Clasa *Canvas*

Este clasa de bază pentru aplicații care necesită desenare la nivel de pixel și tratarea unor evenimente *low-level* (→ jocuri !!)

- **Desenarea** se face în metoda **paint**
- **Tratarea evenimentelor** se face în metode specifice cum ar fi: `keyPressed`, `keyRepeated`, `keyReleased`, etc.

Derivată din `Canvas`, este **GameCanvas** care oferă facilități suplimentare pentru dezvoltarea de jocuri.

# Definirea comenzilor

Comenzi abstracte - *high-level*: **Command**

- **Etichetă**: nume scurt / lung
- **Tip**: BACK, CANCEL, EXIT, HELP, ITEM, OK, SCREEN, STOP
- **Prioritate**: număr întreg; cel mai mic număr - cea mai importantă comandă.

Dispunerea comenzilor pe ecran va ține cont de tipurile și prioritățile acestora.

Adăugare de comenzi pentru un ecran: **addCommand**.

Tratarea evenimentelor generate de acționarea comenzilor: **CommandListener**.

# Interfața *CommandListener*

Metoda interfeței este:  
**commandAction(Command c, Displayable s).**

```
public class MyListener implements CommandListener {  
  
    public void commandAction(Command c, Displayable s) {  
        if (c.getCommandType() == Command.EXIT)  
            notifyDestroyed();  
  
        if (c.getLabel().equals("Play"))  
            letsPlayTheGame();  
        ...  
    }  
}
```

# Imagini

Imaginile sunt utilizate prin intermediul clasei **Image**.  
Formatul admis este **PNG (Portable Network Graphics)**

Crearea se realizează prin metoda: **Image.createImage**,  
conținutul imaginii putând fi:

- preluat din resursele midletului (immutable)
- creat explicit din program (mutable)

# Folosirea imaginilor

## • *componente în cadrul form-urilor*

```
Form mainForm = new Form("Hello");  
Image img = Image.createImage("/Taz.png");  
mainForm.append(img) ; // sau  
ImageItem item = new ImageItem("Poza", img, Item.LAYOUT_CENTER, "  
mainForm.append(item) ;
```

## • *desenare în canvas-uri*

```
public void paint(Graphics g) {  
    Image img = Image.createImage("/Taz.png");  
    g.drawImage(0, 0, HCENTER | VCENTER); }  
}
```

## • *double-buffering*

```
Image img = Image.createImage(width, height);  
Graphics gmem = img.getGraphics();  
gmem.drawLine(0, 0, 10, 10);
```



# Crearea jocurilor de "acțiune"



# Varianta veche

```
public void MyGameCanvas extends Canvas
    implements Runnable {

    public void run() {
        while (true) {
            // Actualizeaza starea jocului
            repaint();
            // Pauza
        }
    }

    public void paint(Graphics g) {
        // Desenare
    }

    protected void keyPressed(int keyCode) {
        // Tratare apasare taste
    }
}
```



# Varianta nouă



```
public void MyGameCanvas extends GameCanvas
    implements Runnable {

    public void run() {
        Graphics g = getGraphics();
        while (true) {
            // Actualizeaza starea jocului

            int keyState = getKeyStates();
            // Tratare apasare taste

            // Desenare

            flushGraphics();

            // Pauza
        }
    }
}
```



# "Planurile" unui joc

Reprezentarea unui joc de "acțiune" la un moment dat constă în suprapunerea unor desene, fiecare aflat într-un anumit plan (layer), unele dintre ele fiind statice iar altele mobile. Stratificarea aplicației în acest fel permite manevrarea mai ușoară și independentă a planurilor. Suportul pentru crearea planurilor este oferit de clasele:

- **Layer**
  - **TiledLayer**
  - **Sprite**
- **LayerManager**

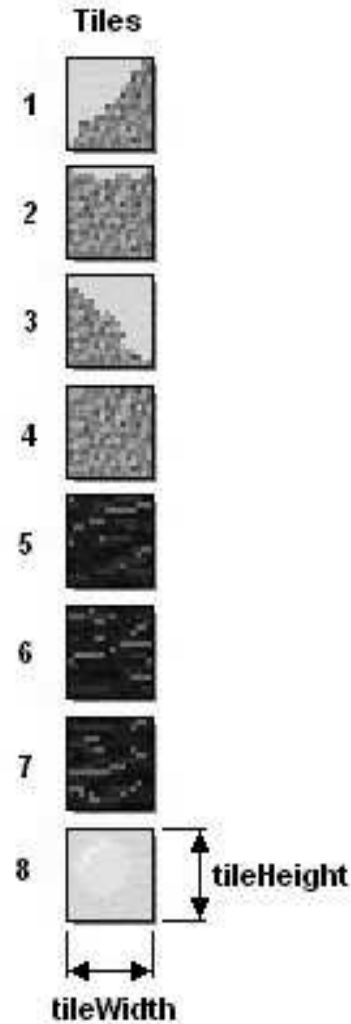
# Definirea componentelor *tile*



OR



OR



# Crearea fundalului

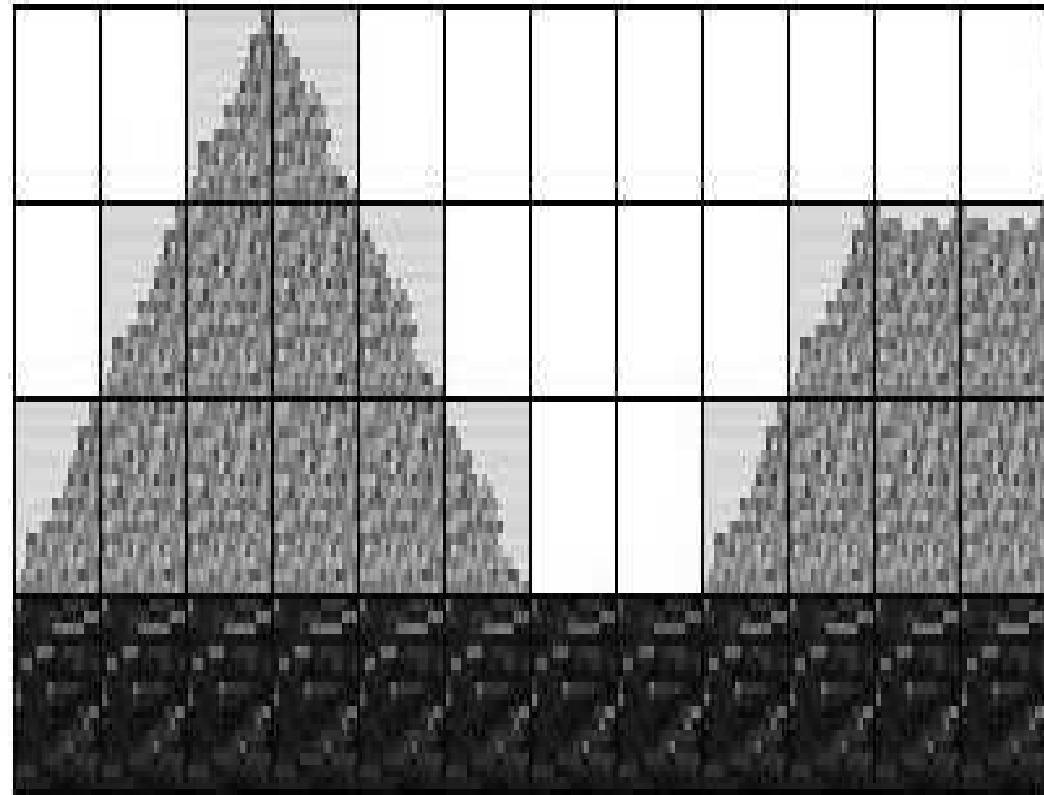


Cells

0	0	1	3	0	0	0	0	0	0	0	0
0	1	4	4	3	0	0	0	0	1	2	2
1	4	4	4	4	3	0	0	1	4	4	4
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Animated Tiles

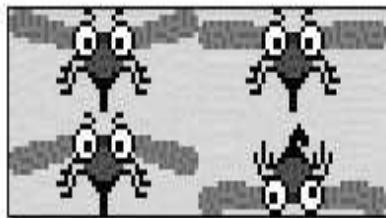
$$\boxed{-1} = \boxed{5}$$



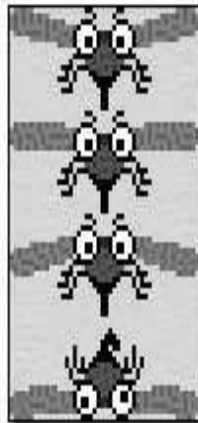
# Definirea 'spiridușilor'



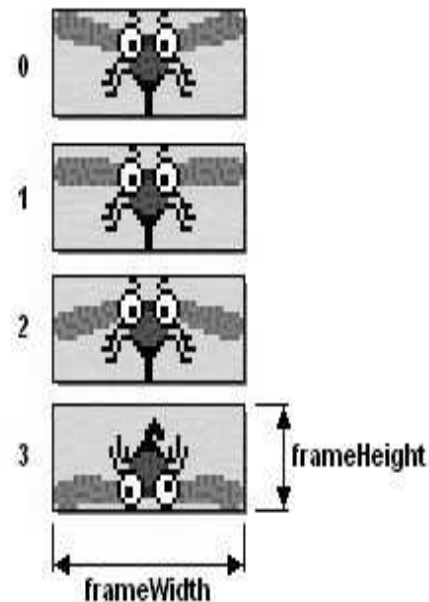
OR



OR



Frames

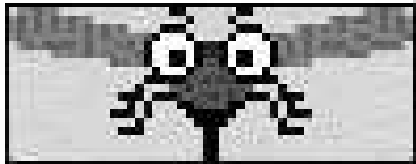


# Definirea animației

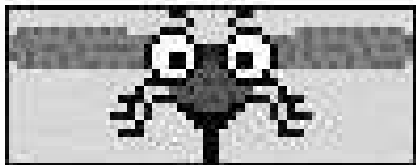
## Default Frame Sequence

0	1	2	3
---	---	---	---

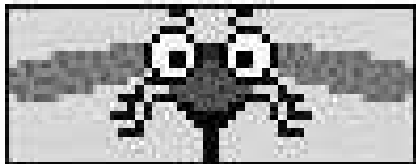
0



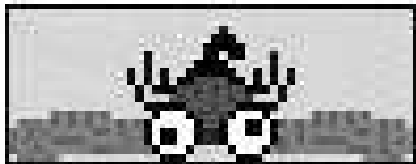
1



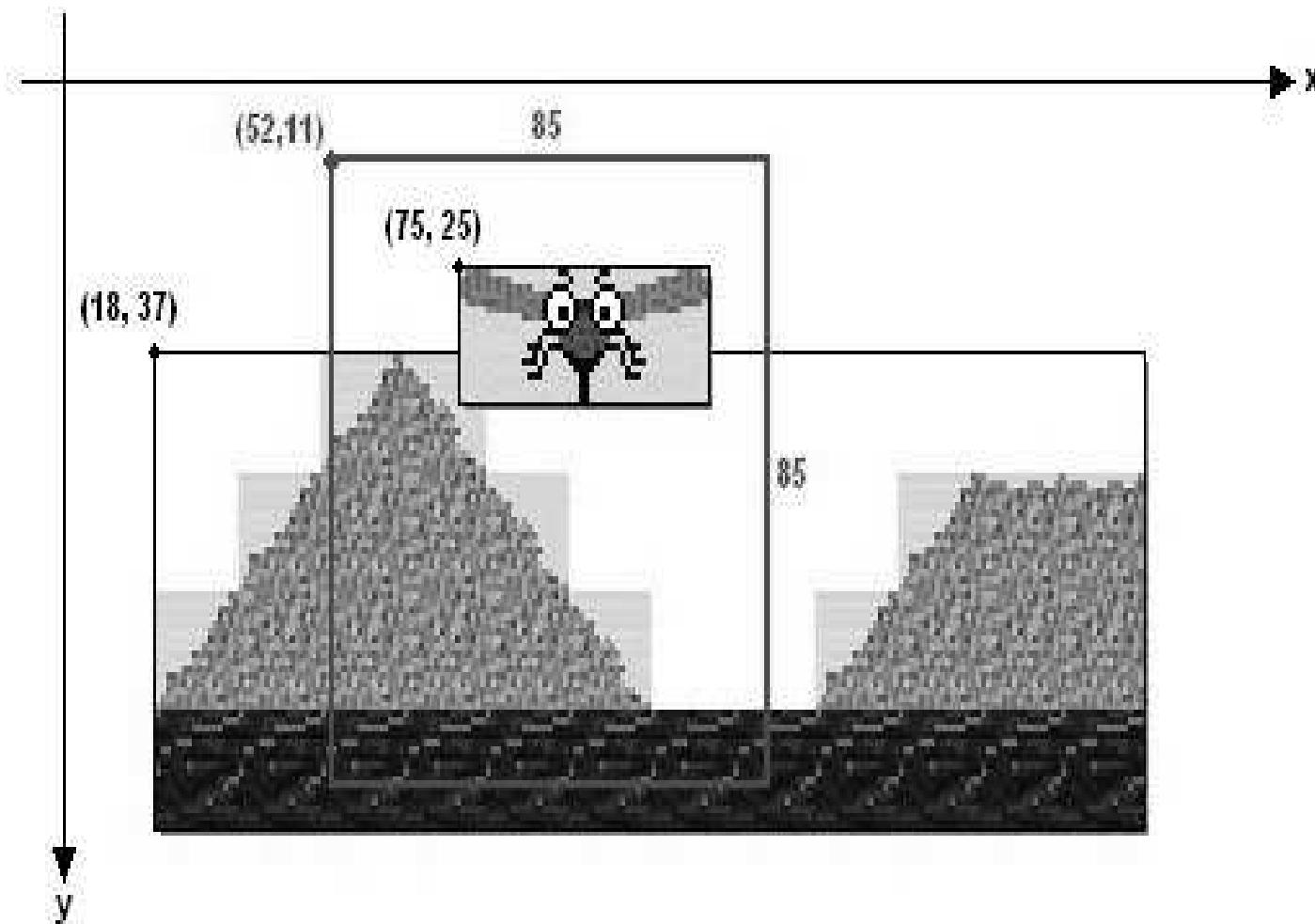
2



3



# Crearea jocului propriu-zis





# Asigurarea persistenței datelor





# Record Management System

Prin clasele sale, RMS oferă suportul pentru asigurarea persistenței unor informații între două execuții succesive ale unui midlet.

Memorarea datelor este realizată într-o **tabelă de persistență**, formată din **articole**. Fiecare articol este compus din:

- **ID** - valoare asignată automat ce identifică în mod unic articolele din tabela de persistență.
- **conținut** - un tablou de octeți, lungimea acestuia putând varia de la un articol la altul.

# Tabele de persistență

## **javax.microedition.rms.RecordStore**

Un midlet poate crea oricâte tabele de persistență, fiecare fiind identificată prin numele său care trebuie să fie unic în cadrul suitei.

Fiecare midlet dintr-o suită poate avea acces la tabelele celorlalte. Aceeași tabelă poate fi utilizată concurent de mai multe midleturi, implementarea clasei `RecordSet` asigurând sincronizarea accesului.

Detectarea modificărilor:

- Obținerea numărului de *versiune*
- Setarea unui **RecordListener**.

# Gestiunea tabelelor

Operațiile ce pot fi efectuate: *creare / ștergere, deschidere / închidere, etc.*

```
import javax.microedition.rms.*;

RecordStore rs = null;
try {
    boolean createIfNecessary = true;
    rs = RecordStore.openRecordStore("mydata", createIfNecessary);
    ...
    rs.closeRecordStore();
}
catch( RecordStoreNotFoundException e ){
    // tabela nu exista
catch( RecordStoreNotOpenException e ){
    // tabela nu este deschisa
} catch( RecordStoreException e ){
    // alta eroare (spatiu insuficient, etc.)
}
```

# Gestiunea articolelor



ID (autoincrement)	Conținut (buffer de octeți)
--------------------	-----------------------------

```
byte[] data = {0, 1};
try {
    int id = rs.addRecord( data, 0, data.length );
    ...
    int numBytes = rs.getRecord(id, data, 0 );
    rs.deleteRecord(id);
}
catch( RecordStoreFullException e ){
    // spatiu insuficient
} catch( InvalidRecordIDException e ){
    // id-ul specificat nu exista
} catch( RecordStoreException e ){
    // general error
}
```



# Scrierea/Citirea tipurilor primitive

## Scrierea

```
ByteArrayOutputStream baos = new ByteArrayOutputStream();
DataOutputStream out = new DataOutputStream(baos);
out.writeUTF("Hello");
out.writeInt(123);
byte data[] = baos.toByteArray();
rs.addRecord( data, 0, data.length );
```

## Citirea

```
byte data[];
rs.getRecord(id, data, 0);
ByteArrayInputStream bais = new ByteArrayInputStream();
DataInputStream in = new DataInputStream(bais);
String s = in.readUTF();
int n = in.readInt(123);
```

# Enumerarea articolelor

Enumerarea articolelor unei tabele de persistență:

```
public RecordEnumeration enumerateRecords(  
    RecordFilter filter,  
    RecordComparator comparator,  
    boolean keepUpdated)
```

- **RecordEnumeration** - interfață ce descrie un enumerator bidirecțional pentru parcurgerea enumerării.
- **RecordFilter** - interfață pentru filtrarea enumerării conform unor criterii proprii.
- **RecordComparator** - interfață pentru compararea / sortarea articolelor din enumerare.



# Distribuția unei suite de midlet-uri



# Crearea unei suite de midleturi (2)

1. Crearea unui proiect nou
2. Scrierea surselor
3. Arhivarea → `Project/Package/Create package`
  - Compilarea
  - Preverificarea
  - Arhivarea (.jar)
  - Crearea unui fișier descriptor (.jad)
4. Instalarea



# Instalarea pe un server Web

1. Copierea fișierelor aplicației (`games.jar` și `games.jad`) pe server
2. Configurarea tipurilor MIME recunoscute de server:

```
jar - application/java-archive  
jad - text/vnd.sun.j2me.app-descriptor
```

3. În `games.jad`:

```
MIDlet-Jar-URL: games.jar  
...devine...  
MIDlet-Jar-URL: http://YourWebServerAddress:port/pathTo/games.jar
```

4. Creare unei pagini Web:

```
<html> <body>  
<a href="http://YourWebServerAddress:port/pathTo/games.jad">  
  Suita de jocuri minunate! </a>  
</body> </html>
```

# Instalarea pe un dispozitiv mobil

Testarea aplicației poate fi făcută cu WTK, pur și simplu accesând din browser legătura de pe pagina Web către midlet.

Dispozitivele ce oferă suport pentru J2ME conțin un **manager de aplicații Java (JAM)**, responsabil cu *descărcarea, instalarea și configurarea* programelor J2ME.

Pentru a simula comportamentul unui JAM, poate fi folosit utilitarul **emulator** din distribuția WTK:  
`emulator -Xjam.`

# OTA: Over The Air

