



**JavaOne**<sup>SM</sup>  
Sun's 2001 Worldwide Java Developer Conference™

# The Java™ 2 Platform, Micro Edition: Game Development for MID Profile

**Roman Verhovsek**  
CEO  
Cocoasoft d.o.o.

**Albin Lozar**  
CTO  
Cocoasoft d.o.o.

# Overall Presentation Goal

To educate you how to develop critical parts of arcade games

To show a set of tips and trick of game development

To show you possible solutions for different kinds of problems regarding device limitations



# Learning Objectives

- At the end of this presentation, you will be able to:
  - Develop an arcade game
  - Exhaustively use low-level GUI API
  - Use HTTP connections
  - Create persistence objects
  - Reduce code size and memory usage



# Speaker's Qualifications

- Roman Verhovsek is co-founder and Chief Executive Officer at Cocoasoft
- Albin Lozar is co-founder and Chief Technology Officer at Cocoasoft



# Presentation Agenda

- Sprites
- Networking
- Object persistence
- Code optimization



# Sprites

- Image manipulation
- Animation
- Movement
- Collision detection



# Loading Images

- JAR size limitations (Motorola 50K, NTT DoCoMo 10K, J-Phone 30K, KDDI 50K)
- Using HTTP connection for retrieving images over the network ([javax.microedition.io](http://javax.microedition.io), [javax.io](http://javax.io))
- Images are retrieved as stream of bytes coded in PNG format



# Loading Images—Source

```
public byte[] loadImage(String url) throws ...
{
    HttpURLConnection conn = ...
    int length = (int)conn.getLength();
    DataInputStream in = ...
    byte value[] = new byte[length];
    in.read(value);
    ...
    return b;
}
```



# Storing Images

- Try to avoid downloading the same image multiple times
- Using internal database to store images (javax.microedition.rms)
- Preparing database
  - Fill with default values at the beginning in progressive order (for ID tracking)
  - `RecordStore.addRecord(...)` is replaced with `RecordStore.setRecord(...)`



# Storing Images—Source

```
RecordStore db = ...  
  
...  
byte value[] = new byte[0];  
for (int i = 1; i <= MAX_ID; i++)  
{  
    db.addRecord(value, 0, 0);  
}  
  
...  
db.setRecord(IMG_ID, newValue, 0,  
            newValue.length);
```



# Creating Images

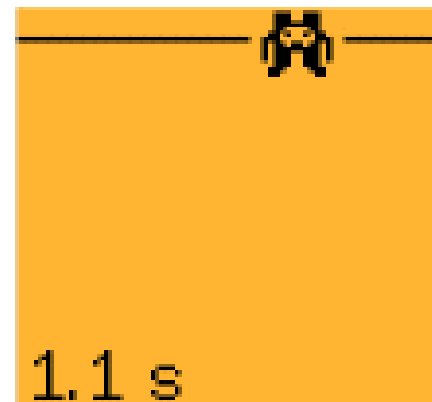
```
public Image getImage(int pos) throws ...
{
    RecordStore db = ...
    byte value[] = db.getRecord(pos);
    return Image.createImage(value, 0,
        value.length);
}
```

- How fast is the database?



# Transparency

- Sprites should not look as they do below
- Transparency is not part of MIDP specification
- Solution: Creating your own transparency



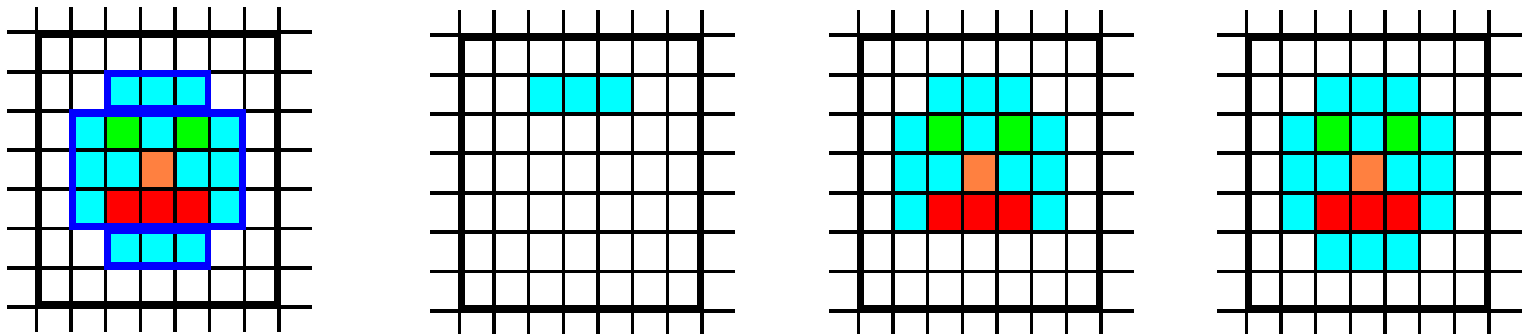
# Developer Managed Transparency

- Solution 1: Draw image in primitive way with `Graphics.drawLine(...)`, etc.—Too slow!
- Solution 2: Put the compound image together with other smaller rectangular parts—If image is round, too many parts are involved!
- Solution 3: Implement own transparency by calling `Graphics.setClip(...)`—Is drawing the image N-times on the screen N-times slower even if the area is clipped?



# Dev. Man. Transparency—Detailed

- Repeating action: Drawing area is clipped for each part and then image is drawn
- NOTE: Don't forget to reset the clipped area to full screen at the end!



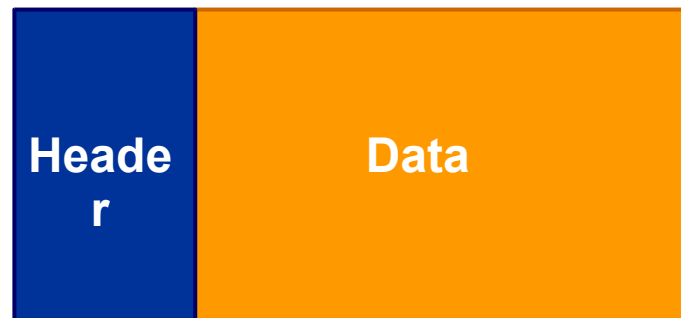
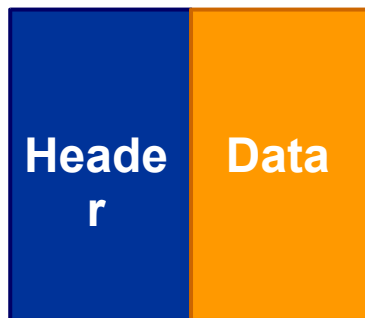
# Dev. Man. Transparency—Source

```
private int clipArea[][] = {{2, 1, 3, 1},...
public void drawSprite(Graphics g, int x, int y)
{
    for (int i = 0; i < clipArea.length; i++)
    {
        g.setClip(clipArea[i][0] + x,
                 clipArea[i][1] + y, clipArea[i][2],
                 clipArea[i][3]);
        g.drawImage(img, x, y, ...);
    }
    g.setClip(0, 0, WIDTH, HEIGHT); // RESET!!!
}
```



# Image Size Reduction

- PNG image contains different kind of data
  - Header (also includes color palette with alpha channel)
  - Image data
- Smaller image, worse size ratio between header and image data



# Image Size Reduction—Detailed

- Suggestion: Put more images in one PNG file
- Need: Ability to extract specific image from the file
- Solution: Using `Graphics.setClip(...)`
- Drawing the complete image which is visually limited by the clipped area
- QUESTION: How fast is the drawing if the image is large?



# Double-buffering

- Flickering is always annoying
- MIDP: double-buffering is not a must
- Check with `Canvas.isDoubleBuffered()`
- Great approach on i-mode: `Graphics.lock()` and `Graphics.unlock(boolean force)`
- Developer managed double-buffering is memory hungry (check heap memory)



# Double-buffering—Source

```
Image image = Image.createImage(WIDTH, HEIGHT);
Graphics bufG = image.getGraphics();
public void paint(Graphics g)
{
    if (isDoubleBuffered())
        drawScene(g);
    else
    {
        drawScene(bufG);
        g.drawImage(image, 0, 0, ...);
    }
}
```



# Animation

- Sprite animation must be equally time-sliced
- 10 frames per second is currently the upper limit
- TimerTask implements task functionality (checking keyboard, move sprites,...)
- Timer manipulates with tasks
- For games: Schedule tasks at fixed rate
- NOTE: Work time + Repaint time < Task time!!! (Solution: Your own threading)



# Animation Thread—Source

```
while (running)
{
    long time = System.currentTimeMillis();
    ... // Task functionality + serviceRepaints()
    time = System.currentTimeMillis() - time;
    if (time >= delay) continue;
    try
    {
        Thread.sleep(delay - time);
    }
    catch (Exception e) {}
}
```

# Keyboard 1/2

- Key handling with `keyPressed(int code)`, `keyReleased(int code)` and `keyRepeated(int code)`
- ITU-T codes for 0..9, \* and #
- LEFT, RIGHT, UP, DOWN and FIRE are game key codes—use `getGameAction(int code)`
- Key pressed duration is not supported (TIP: create global variable that receives code value when key is pressed and -1 when key is released)



# Keyboard 2/2

- Movement should be done in a thread
- Experience from J2ME™ MIDP for Palm: Don't implement touchable key-emulation because it is already implemented. Holding the 'key' doesn't have the same impact on a palm devices as in mobile phones
- Sprite directions: Using MathFP library (Go to: [rr.com/ohommes/MathFP/index.html](http://rr.com/ohommes/MathFP/index.html))



# Sprite Collision

- Solution 1: Simple rectangle collision—  
FAST AND INACCURATE
- Solution 2: Image data collision—  
TOO SLOW ON SMALL DEVICES
- Solution 3: Shrunk rectangle collision—  
GOOD COMPROMISE
- Solution 3 is also good enough for  
combination of rectangles to determine  
the level of collision



# Networking With HTTP

- MIDP supports HTTP
- Other protocols are optional
- Connection is created with `Connector.open(String name)`
- Type-cast to `HttpConnection`
- Operate with URL parameters for sending data
- Operate with `DataInputStream` for receiving data



# HTTP Client—Source

```
HttpConnection conn = (HttpConnection)Connector.  
    open("http://server/servlet?name=Roman");  
conn.setRequestMethod(HttpConnection.POST);  
DataInputStream in = ...  
...  
StringBuffer buf = ...  
int ch;  
while ((ch = in.read()) != -1)  
{  
    buf.append((char)ch);  
}
```



# Server Side: Servlet

- Server side must support HTTP
- Solution: Web server with Servlet engine
- Using doGet() or doPost()
- Using HttpServletRequest for receiving data from client
- Using HttpServletResponse for sending data to client



# Server—Source

```
public void doPost(HttpServletRequest request
    HttpServletResponse response)
{
    // Input
    String value = request.getParameter("name");
    ... // Functionality
    // Output
    PrintStream out = new PrintStream(response.
        getOutputStream());
    out.write(returnValue, 0, size);
    out.close();
}
```



# Object Persistence

- Objects held in heap memory are alive for the time of MIDlet life (or less)
- The need to make data persistent (e.g., Top scoring)
- MIDP supports its own internal database (javax.microedition.rms)
- Create database with `RecordStore.openRecordStore(dbName, true)`



# Storing Data

- Be aware of database space (check with `RecordStore.getSizeAvailable()`)
- Storing into database by using `addRecord(...)` and `setRecord(...)`
- ID is unique, progressive and non-repeatable
- Use `ByteArrayInputStream` and `ByteArrayOutputStream` for storing data as array of bytes



# Storing Data—Source

```
ByteArrayOutputStream out = new
    ByteArrayOutputStream();
DataOutputStream dos = new
    DataOutputStream(out);
dos.writeInt(reportId);
dos.writeLong(reportDate.getTime());
dos.writeUTF(reportTitle);
byte data[] = out.toByteArray();
dos.close();
out.close();
db.setRecord(id, data, 0, data.length);
```



# Retrieving Data

- Retrieve data by calling `getRecord(int id)`
- ID should be known in advance
- For searching: call `enumerateRecords(...)`
  - `RecordFilter` (filtering)
  - `RecordComparator` (sorting)
- Be careful with enumeration: `nextRecordId()` and `nextRecord()` both moves current position in queue



# Retrieving Data—Source

```
byte data[] = db.getRecord(id);  
ByteArrayInputStream in = new  
    ByteArrayInputStream(data);  
DataInputStream dis = new DataInputStream(in);  
reportId = dis.readInt();  
reportDate.setTime(dis.readLong());  
reportTitle = dis.readUTF();  
dis.close();  
in.close();
```



# Optimization 1/2

- Naming: Class, public method and public field names influence on class size (1 character = 1 byte)
- Each class brings additional overhead—use limited O.O. approach
- At non-critical situations use `Runtime.getRuntime().gc()`
- Avoid frequent dynamic creation of objects (whenever possible: use arrays instead of vectors)



# Optimization 2/2

- Be aware of heap memory fragmentation
- Heap memory is not just data container
- Use StringBuffer instead of String
- Always limit number of sprites on the screen
- Don't use full screen on Palm-like devices



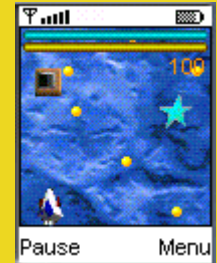
# Summary

- Problems and solutions
  - Sprites
    - Image manipulation
    - Transparency
    - Animation
    - Keyboard
    - Collision detection
  - Networking
  - Data storage
  - Optimization





**JavaOne**<sup>SM</sup>  
Sun's 2001 Worldwide Java Developer Conference™



roman@cocoasoft.com  
albin.lozar@cocoasoft.com



<http://www.cocoasoft.com>  
<http://www.javamobile.org>





**JavaOne**<sup>SM</sup>

Sun's 2001 Worldwide Java Developer Conference\*



**JavaOne**<sup>SM</sup>  
Sun's 2001 Worldwide Java Developer Conference™

# The Java™ 2 Platform, Micro Edition (J2ME™): Game Development for MID Profile

**Markus Kreitmair**

Produkt Manager Smart Devices  
Siemens Mobile Phones

# Overall Presentation Goal

1. Present the J2ME technology on Siemens' mobile phones with OEM extensions and GameAPI
2. Show the difference between programming with MIDP and OEM extensions
3. Give tips to be compatible
4. Explain the Siemens Wireless Emulator, for the J2ME platform



# Learning Objectives

- At the end of this presentation, you will be able to:
  - Know when you should use MIDP and OEM extensions (like the GameAPI)
  - Write MIDlets for mobile devices
  - Implement sprites
  - Create sound and effects on mobile devices
  - Know how the Siemens Wireless Emulator works within the J2ME platform



# Speaker's Qualifications

- **Markus Kreitmair**  
Product Manager Siemens Mobile Phones
- **Jan Eichholz**  
Siemens Mobile Phones Java Development
- **Michael Becker**  
Siemens Corporate Technology Java Task Force



# Key Topic Areas

- MIDP versus OEM extension and GameAPI
- Developing with GameAPI
  - Overview
  - Sprites
  - Sound and effects
- Compatibility between different devices
- Java™ technology on Siemens Mobile Phones



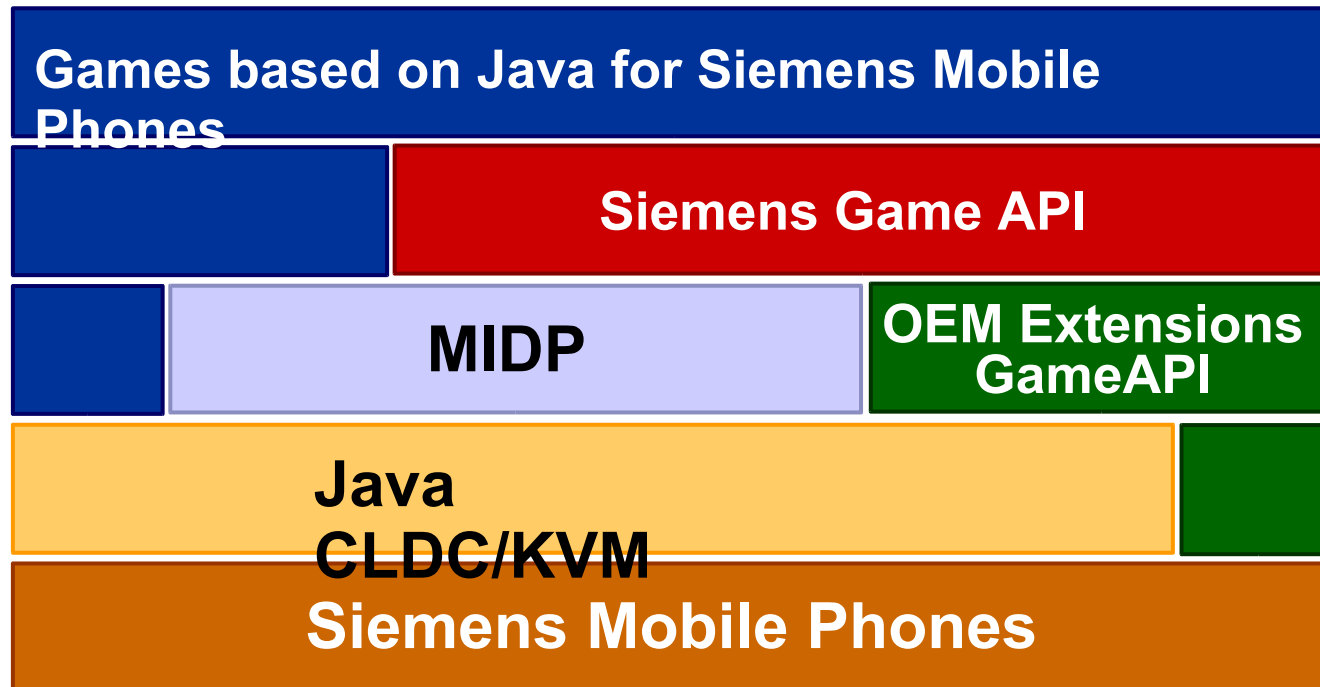
# MIDP versus OEM Extensions

- The Siemens Game API extends the Java™ MIDP API to provide functionality for game development on Siemens Mobile Phones
- The GameAPI was written to provide additional gaming functions with higher performance
- All MIDP/CLDC functions access the native functions from the mobile phone through bridge-classes
- MIDP/CLDC Parameters are quickly accessed via pop and push-functions through the stack



# MIDP versus OEM Extensions

## OEM Extensions and Game API



# Sprites—Overview

- The sprite class provides functionality to draw a bitmap-sprite to the display
  - Sprites are figures or elements on the screen that have the capability of moving independently of one another
  - With a mask overlapping regions can handled (same as transparent color)
- The properties of a sprite are (Sprite States)
  - Internal representation of screen appearance
  - Screen location
  - Visibility
  - Priority
  - Updateability
  - Painting, the sprite paints itself to the screen



# Sprites—Java Source Code Example (1/2)

```
// Create empty image for background (double buffer)
Image gameScreenImage =Image.createImage (100,100);

// Create predefined bitmap for sprite
Image ufo_image=Image.createImage ("ufo_image.png");

// Create predefined bitmap for mask
Image ufo_mask=Image.createImage ("ufo_mask.png");

// Create sprite object
Sprite ufo_sprite = new Sprite(ufo_image, ufo_mask, 1);
```



# Sprites—Java Source Code Example (2/2)

```
// Create sprite manager object
GraphicObjectManager spriteManager =
new GraphicObjectManager();

// Add sprite to object manager
spriteManager.addObject(ufo_sprite);

// Set sprite parameter e.g. position
ufo_sprite.setPosition(10,10);

// Paint sprite to double buffer
spriteManager.paint(gameScreenImage, 0, 0);

// Copy double buffer to display (game screen)
Graphics.drawImage(doublebuffer);
```



# Game API—Sound and Effects (1/2)

- **Vibra**
  - This class provides basic access to the phone's vibrator. You can activate the vibrator either for a given period of time and duration
- **Light**
  - This class provides basic access to the phone's LCD back light. You can activate or deactivate the back light
- **Sound**
  - This class provides basic access to the phone's sound capability. You can play different static tones



# Game API—Sound and Effects (2/2)

- Melody and MelodyComposer
  - This class provides basic access to the phone's melody composer. You can compose a melody using predefined tones and macros
- ExtendedImage
  - The ExtendedImage is a standard Java Image with additional methods to manipulate the image data (setPixel, getPixels, ...)



# How to Be Compatible?

- Standard versus OEM ?
  - If you are writing a game like car racing:  
Use the vibra function from the Siemens OEM extensions
    1. *Write a interface like vibra\_on*
    2. *Implement two classes, one with the vibra support and the other one with MIDP compliant alternative or dummy*
    3. *Find out on which device your MIDLet is running, by using the getProperty function and load the corresponding class*



# How to Be Compatible ?

- Standard versus OEM?
  - This game will run on a Siemens phone with vibra support and on other phone not supporting this extension without vibrator
- Conclusion:
  - It is possible to use extensions without getting proprietary



# Java™ Technology on Siemens Mobile Phones

## ➤ 3Q/2001: Global product launch

- ◆ Develop joint market roll-outs for Java applications

**“U35k”**

prototype of a  
voice centric  
mobile phone



**SX45**

PocketPC  
data centric  
mobile phone



## ➤ The future of Java technology and Siemens

- ◆ Step by step implementation in nearly all Siemens Mobile Phones



# Java Technology and Siemens Mobile Phones



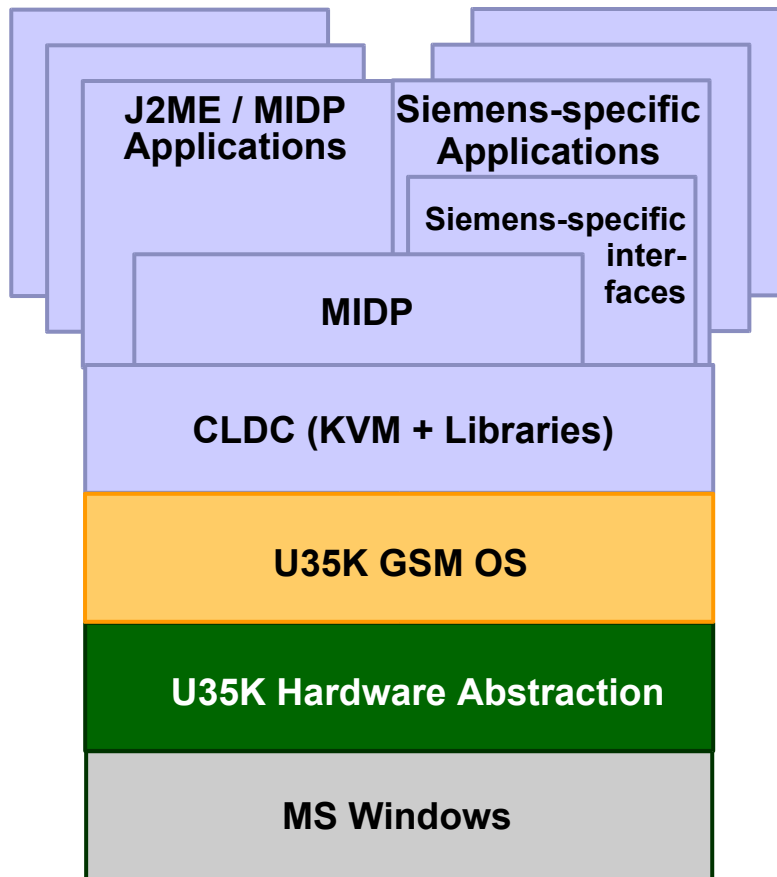
## Siemens Wireless Emulator for the J2ME™ platform

- MMI emulation window
- Soft keypad control
- event window



# Java Technology on Siemens Mobile Phones

## Emulator Structure



1. Emulates complete phone behaviour like handling calls and SMS
2. U35k look and feel
3. Persistent storage (MMC) mapped to PC file system

 A *real* U35K emulator!





**JavaOne**<sup>SM</sup>  
Sun's 2001 Worldwide Java Developer Conference™

# Q&A

**markus.kreitmair@mch.siemens.de**  
**michael.becker@mchp.siemens.de**

**Partnering Program:**  
**bernhard.geisberger@mch.siemens.de**

**<http://www.siemens-mobile.de>**



**JavaOne**<sup>SM</sup>

Sun's 2001 Worldwide Java Developer Conference<sup>®</sup>