Mobile Systeme
Grundlagen und Anwendungen standortbezogener Dienste

Location Based Services in the Context of Web 2.0

Department of Informatics - MIN Faculty - University of Hamburg
Lecture Summer Term 2007

Dr. Thilo Horstmann

CLDC

NMEA

MIDP

Google Earth

OpenGIS

SQL

KML

Bluetooth

Mash-Ups

Web 2.0

J2ME

Loxodrome

Euler Spaces

RDMS

GPS

PostGIS

GPX

Maps

JSR 179

Polar Coordinates

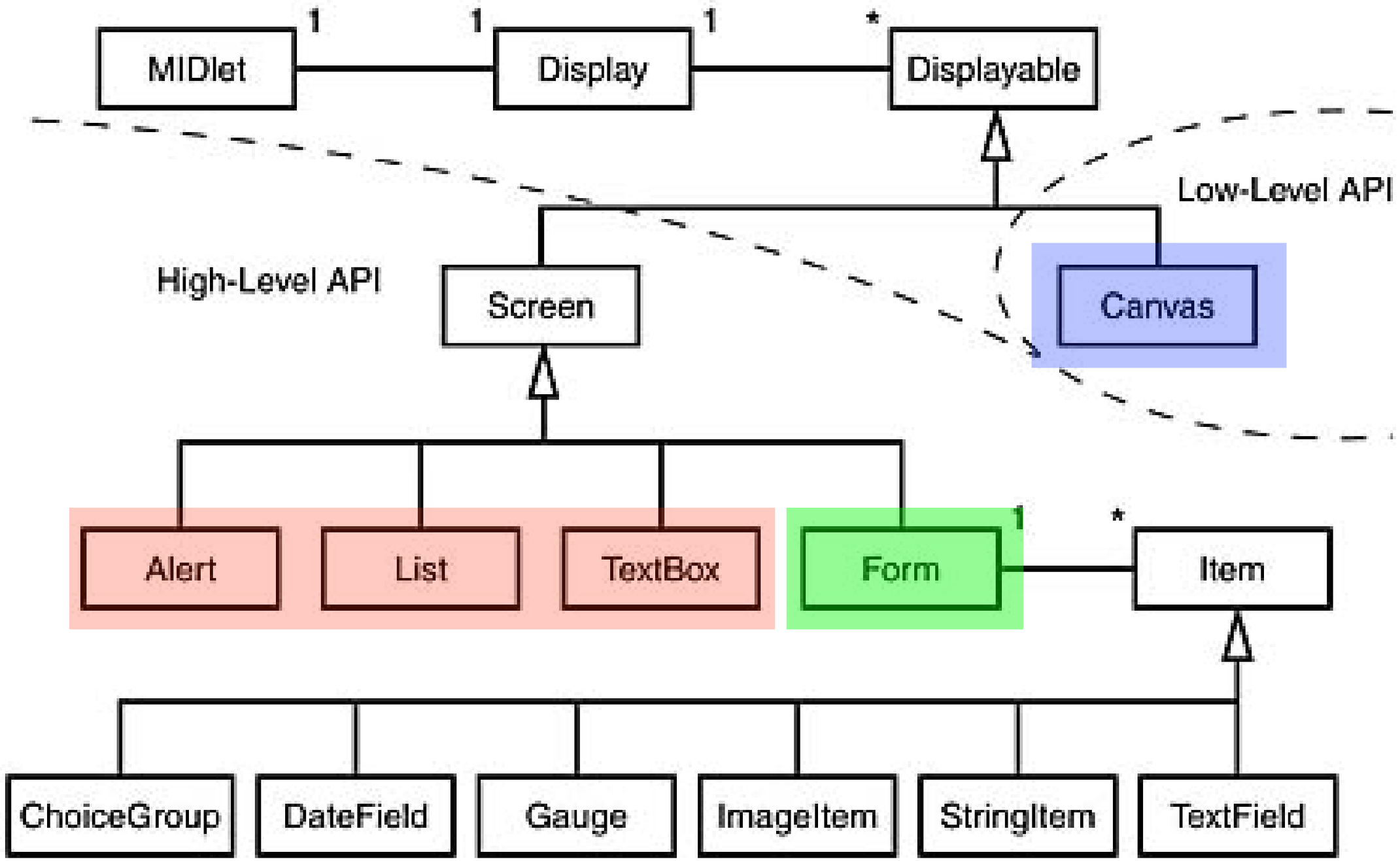
API

Threads

Today: J2ME (IV)

- The Low Level GUI API (MIDP 2.0)
- Canvas, CustomItem, GameCanvas
- Demonstration of low level GUI objects

J2ME GUI components



Canvas, paint(), and Graphics

J2ME Low level API

- The low-level API provides very little abstraction.
    - This API is designed for applications that need precise placement and control of graphic elements, as well as access to low-level input events. Some applications also need to access special, device-specific features. A typical example of such an application would be a game.

- Using the low-level API, an application can:
    - Have full control of what is drawn on the display.
    - Listen for primitive events like key presses and releases.
    - Access concrete keys and other input devices.
- The classes that provide the low-level API are Canvas and Graphics.

J2ME Low level API

- Applications that program to the low-level API
  - are not guaranteed to be portable, since the low-level API provides the means to access details that are specific to a particular device.
  - If the application does not use these features, it will be portable.
  - It is recommended that applications use only the platform-independent part of the low-level API whenever possible. This means that the applications should not directly assume the existence of any keys other than those defined in the Canvas class, and they should not depend on a specific screen size. Rather, the application game-key event mapping mechanism should be used instead of concrete keys, and the application should inquire about the size of the display and adjust itself accordingly.

Getting the Device Properties

- Since portability cannot be guaranteed we must retrieve the capabilities and properties at run time.
- the classes Display and Canvas provide methods for the retrieval of screen dimensions, colors, pointer events etc.

- Demo: Retrieval of device properties (cf. [1], p. 100)

Basic Drawing of Geometric Object

- Subclass the abstract class canvas and overwrite the paint method:

```
public class MyCanvas extends Canvas {
    public void paint(Graphics g) {
        g.setColor(255, 0, 0);
        g.fillRect(0, 0, getWidth(), getHeight(  ));
        g.setColor(255, 255, 255);
        g.drawString("Hello World!", 0, 0, g.TOP | g.LEFT);
    }}

//in your Midlet instantiate the canvas screen as usual
public void startApp(  ) {
        Canvas canvas = new MyCanvas(  );
        Display display = Display.getDisplay(this);
        display.setCurrent(canvas);
    }
```

Canvas painting

- paint() will be called from outside your application, i.e. the application management software. Never call paint() directly.
- paint() needs to draw the complete screen. There is no guarantee that portions of the screen will stay the same.
- the graphics object g is only valid within the paint method. Do not create references as they will be invalid after paint() has completed.
- Colors are defined in the RGB model:
  - 0x<red><green><blue>, e.g. 0x0000FF for blue, 0x000000 for black
  - MIDP implementations must support the rendering of fully opaque pixels and fully transparent pixels in immutable images.
    - alpha blending is optional

The Graphics Object

- Provides simple 2D geometric rendering capability.
- Drawing primitives are provided for text, images, lines, rectangles, and arcs. Rectangles and arcs may also be filled with a solid color. Rectangles may also be specified with rounded corners.
- A 24-bit color model is provided, with 8 bits for each of red, green, and blue components of a color. Not all devices support a full 24 bits' worth of color and thus they will map colors requested by the application into colors available on the device. Facilities are provided in the Display class for obtaining device characteristics, such as whether color is available and how many distinct gray levels are available. Applications may also use getDisplayColor() to obtain the actual color that would be displayed for a requested color. This enables applications to adapt their behavior to a device without compromising device independence.

The Graphics Object (cont.)

- For all rendering operations, source pixels are always combined with destination pixels using the Source Over Destination rule. Other schemes for combining source pixels with destination pixels are not provided.

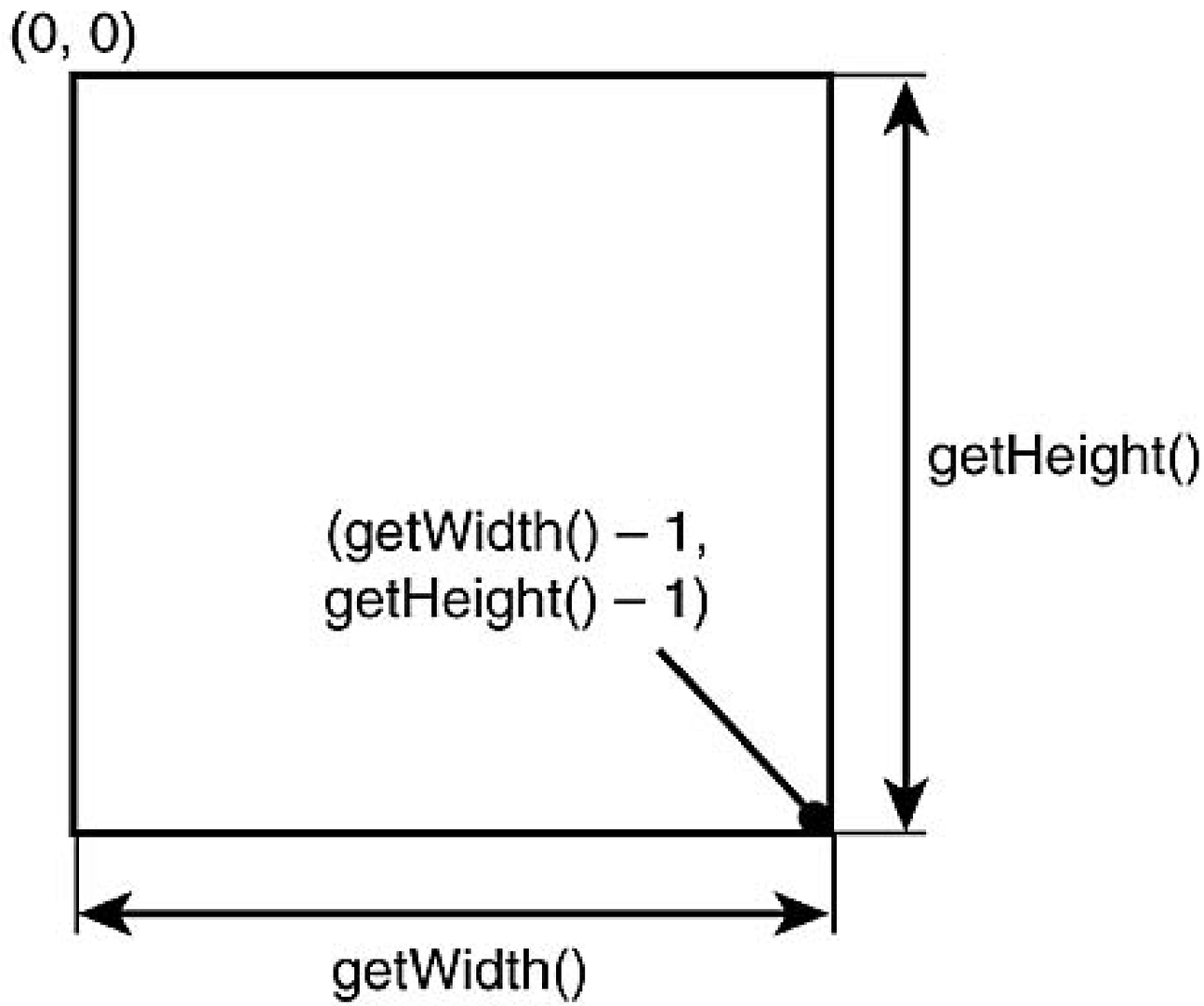
- For the text, line, rectangle, and arc drawing and filling primitives, the source pixel is a pixel representing the current color of the graphics object being used for rendering. This pixel is always considered to be fully opaque. With source pixel that is always fully opaque, the Source Over Destination rule has the effect of pixel replacement, where destination pixels are simply replaced with the source pixel from the graphics object.

- The drawImage() and drawRegion() methods use an image as the source for rendering operations instead of the current color of the graphics object. In this context, the Source Over Destination rule has the following properties: a fully opaque pixel in the source must replace the destination pixel, a fully transparent pixel in the source must leave the destination pixel unchanged, and a semitransparent pixel in the source must be alpha blended with the destination pixel. Alpha blending of semitransparent pixels is required. If an implementation does not support alpha blending, it must remove all semitransparency from image source data at the time the image is created.

Graphics object: coordinate system

- The default coordinate system's origin is at the upper left-hand corner of the destination. The X-axis direction is positive towards the right, and the Y-axis direction is positive downwards.

- The coordinate system represents locations between pixels, not the pixels themselves. The first pixel in the upper left corner of the display lies in the square bounded by coordinates (0,0) , (1,0) , (0,1) , (1,1)

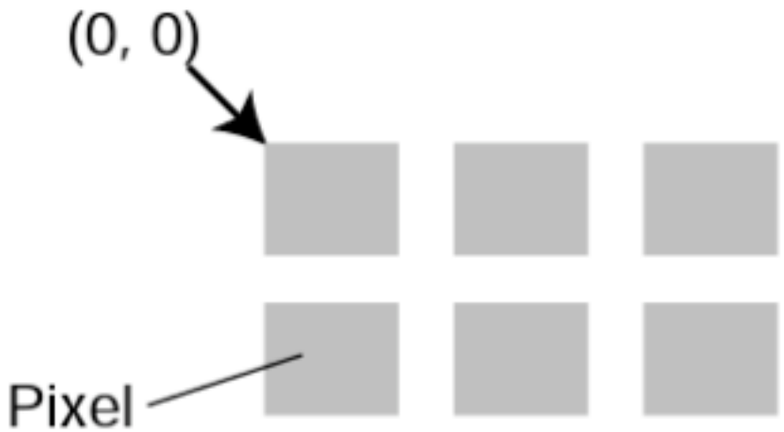


Important Graphics methods for painting

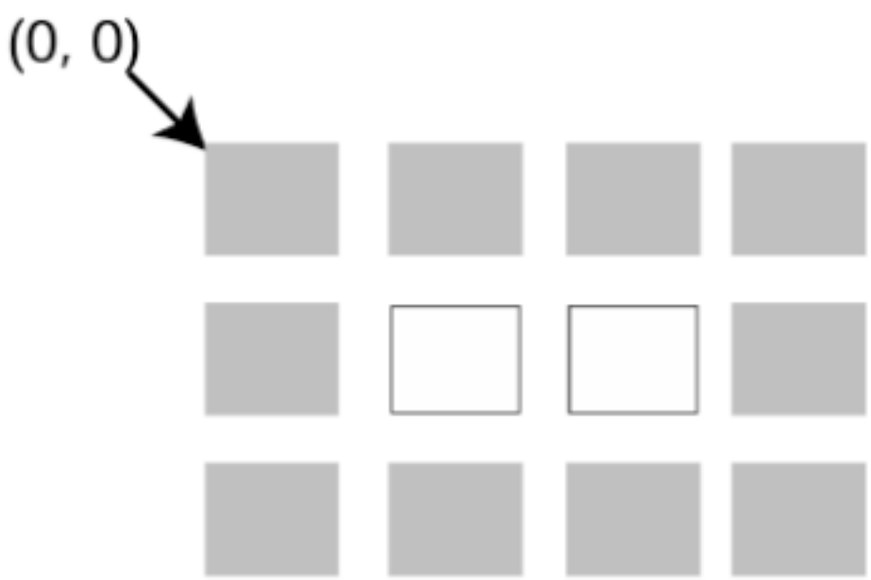
- setColor(int red, int green, int blue), setColor(int red, int green, int blue)
- fillRect(int x, int y, int width, int height)
- drawRect(int x, int y, int width, int height)
- drawLine(int x1, int y1, int x2, int y2)
- drawString(String str, int x, int y, int anchor)
- drawImage(Image img, int x, int y, int anchor)
- fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)
- drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)
- fillTriangle(int x1, int y1, int x2, int y2, int x3, int y3)

An artifact of the coordinate system

- fillRect(0, 0, 3, 2)



- drawRect(0, 0, 3, 2)



Canvas Events

- In addition to paint() the abstract class Canvas provides empty event methods which *can* be implemented in a (concrete) subclass:
- showNotify(), hideNotify()
- keyPressed(), keyRepeated(), keyReleased()
- pointerPressed(), pointerDragged(), pointerReleased()
- These methods are all called serially
  - the implementation will never call an event delivery method before a prior call to any of the event delivery methods has returned
  - serviceRepaints() method is an exception to this rule
    - Forces any pending repaint requests to be serviced immediately.

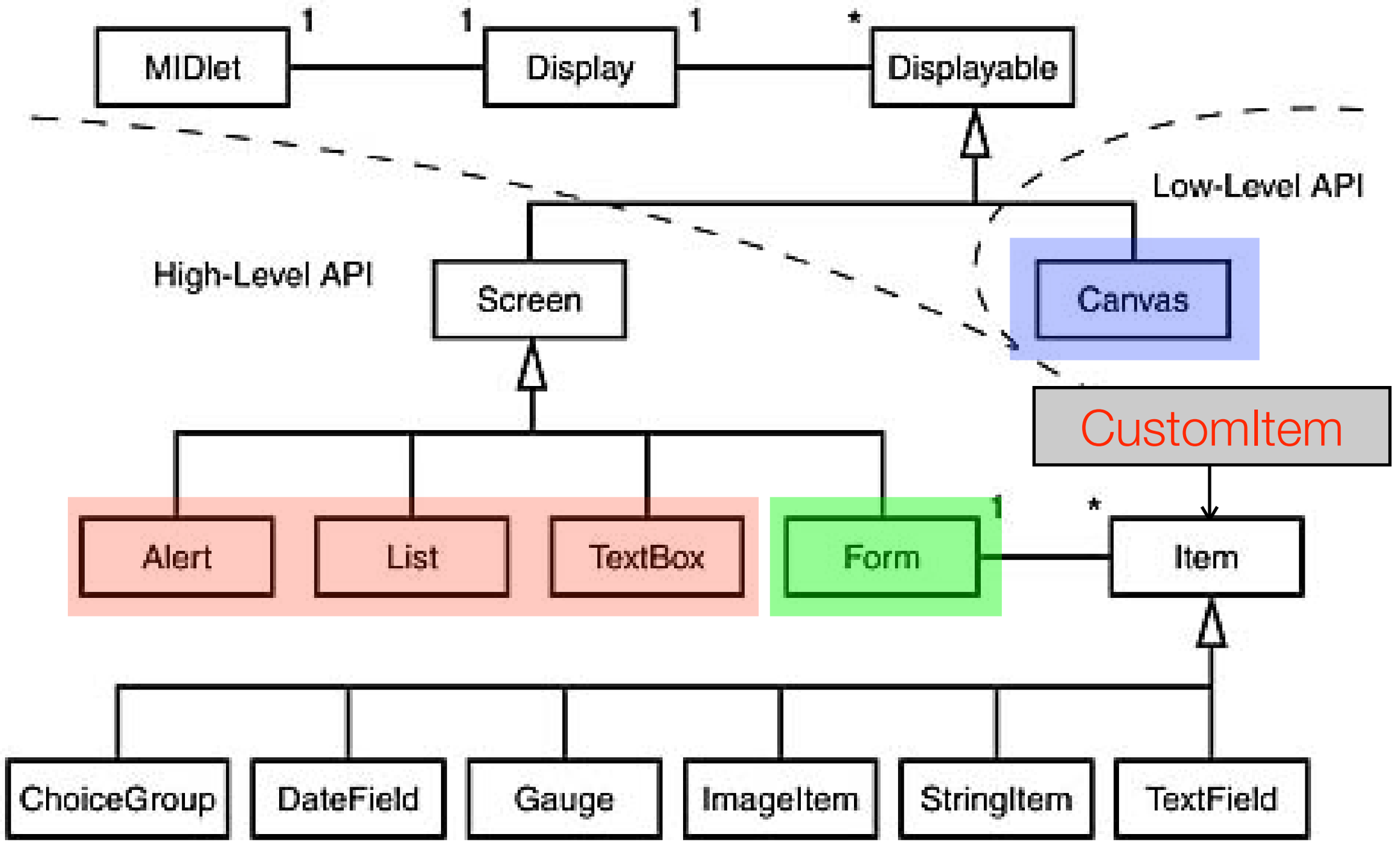
Key events

- MIDP defines the following key codes:
  - KEY_NUM0, KEY_NUM1...KEY_NUM9, KEY_STAR, and KEY_POUND
  - ITU-T standard telephone keypad
- Other keys are assigned a corresponding unicode key, if possible
  - if not, negative codes must be used
- if (keyCode > 0) { char ch = (char)keyCode; // ... }
  - not sufficient for text input! Always use text boxes or text fields
- Portable applications that need arrow key events and gaming-related events should use game actions in preference to key codes and key names: UP, DOWN, LEFT, RIGHT, FIRE, GAME_A, GAME_B, GAME_C, and GAME_D.

Demo: Canvas Example

The best of two worlds: CustomItem

J2ME GUI components



CustomItem: Customizable Item for Form

- A CustomItem is customizable by subclassing to introduce new visual and interactive elements into Forms.
- Subclasses are responsible for their visual appearance including sizing and rendering and choice of colors, fonts and graphics.
    - i.e., the CustomItem is responsible for painting its contents whenever the paint method is called.
- Subclasses are responsible for the user interaction mode by responding to events generated by keys, pointer actions, and traversal actions.
- Subclasses are responsible for calling Item.notifyStateChanged() to trigger notification of listeners that the CustomItem's value has changed.

Demo: CustomItem Example

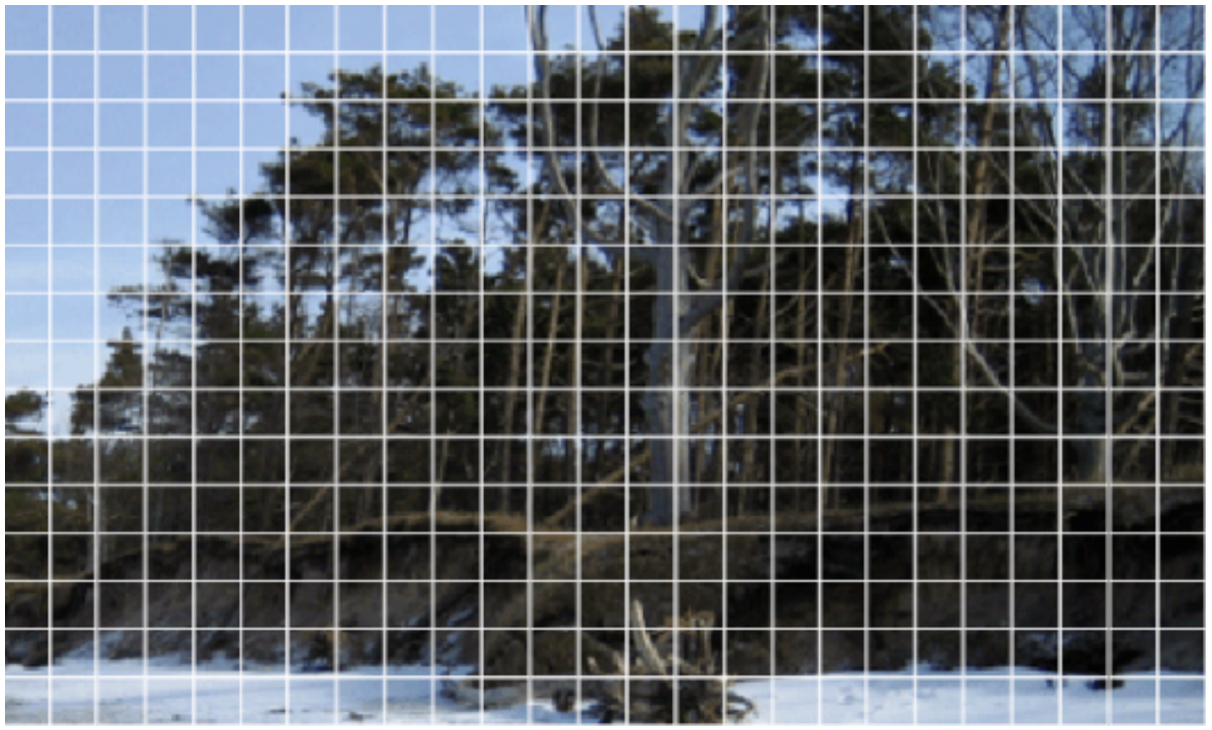
Game API (javax.microedition.lcdui.game)

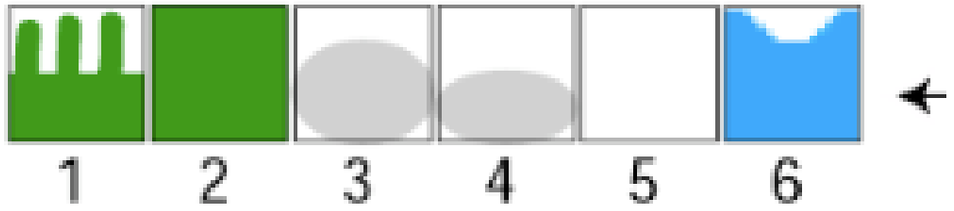
- API is intended to improve performance by minimizing the amount of work done in Java; this approach also has the added benefit of reducing application size. The API's are structured to provide considerable freedom when implementing them, thereby permitting the extensive use of native code, hardware acceleration and device-specific image data formats as needed.

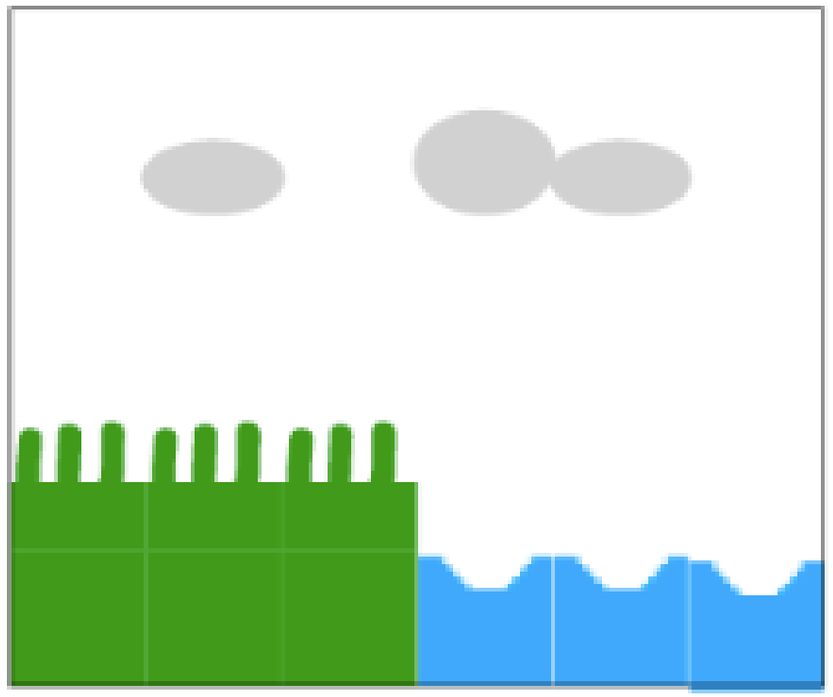
- The API uses the standard low-level graphics classes from MIDP (Graphics, Image, etc.) so that the Game API classes can be used in conjunction with graphics primitives. For example, it would be possible to render a complex background using the Game API and then render something on top of it using graphics primitives such as drawLine, etc.

- Methods that modify the state of Layer, LayerManager, Sprite, and TiledLayer objects generally do not have any immediately visible side effects. Instead, this state is merely stored within the object and is used during subsequent calls to the paint() method. This approach is suitable for gaming applications where there is a game cycle within which objects' states are updated, and where the entire screen is redrawn at the end of every game cycle.

Game API: TiledLayer







5x6 TiledLayer

Game API: GameCanvas, Layer, LayerManager

- GameCanvas: This class is a subclass of LCDUI's Canvas and provides the basic 'screen' functionality for a game. In addition to the methods inherited from Canvas, this class also provides game-centric features such the ability to query the current state of the game keys and synchronous graphics flushing; these features simplify game development and improve performance.

- Layer: The Layer class represents a visual element in a game such as a Sprite or a TiledLayer. This abstract class forms the basis for the Layer framework and provides basic attributes such as location, size, and visibility.

- LayerManager: For games that employ several Layers, the LayerManager simplifies game development by automating the rendering process. It allows the developer set a view window that represents the user's view of the game. The LayerManager automatically renders the game's Layers to implement the desired view.

Game API: Sprite, TiledLayer

- Sprite: A Sprite is basic animated Layer that can display one of several graphical frames. The frames are all of equal size and are provided by a single Image object. In addition to animating the frames sequentially, a custom sequence can also be set to animation the frames in an arbitrary manner. The Sprite class also provides various transformations (flip and rotation) and collision detection methods that simplify the implementation of a game's logic.

- TiledLayer: This class enables a developer to create large areas of graphical content without the resource usage that a large Image object would require. It is a comprised of a grid of cells, and each cell can display one of several tiles that are provided by a single Image object. Cells can also be filled with animated tiles whose corresponding pixel data can be changed very rapidly; this feature is very useful for animating large groups of cells such as areas of water.

Demo: Game API Example

Homework :-)

- Design and implement a CustomItem „CompassRoseItem“ which can be used as an item on a Form. It should provide a method setDirection(float direction). When called the rose turns to the given direction. 0 respectively 360 is the top direction.

- To demonstrate the item create an event handle for the Joystick buttons: The rose turns right as long as the middle Joystick button is pressed. If another Joystick button is pressed the rose should turn in that direction (top=North, right=East, etc.

This Lecture

- Breymann, U., Mosemann, H.: Java ME – Anwendungsentwicklung für Handys, PDA und Co., Hanser, 2006, www.java-me.de
  - Chapter 5.4, 5.5, 5.6
- MIDP Programming with J2ME:
  - http://www.developer.com/java/j2me/article.php/10934_1561591

Thank you!

Dr. Thilo Horstmann

e-mail: thilo.horstmann@gmail.com
blog: http://www.das-zentralorgan.de