CS5014 Mobile Applications

Paul Ahern

May 31, 2007

Abstract

Notes from lectures.

Contents

1 Key Facts to Know

Exam: Two questions, answer both.

Questions will have been covered in the assignments.

Know the rough order of arguments of standard methods.

Know the structure of applications for particular problem (e.g. the sort of items that can be put on a form - StringItem, Spacer, etc.).

Not coming up: Turtle Graphics, 3D and PDA.

2 Lecture 12 January 2007

Lecturers Sabin Tabirca & Daniel Doolan

Subject Java for Mobile Devices

Website www.cs.ucc.ie/~stabirca/cs5014/

Book Wireless Java by Jonathan Knudsen

2.1 Acronyms

J2ME = Java Micro Edition

CLDC = Connected Limited Device Configuration

CDC = Connected Device Configuration

KVM = Kilobyte virtual machine.

CVM = Consumer and embedded virtual machine.

MIDP = Mobile information device profile.

PDA = Personal digital assistant device.

MIDlet = Java program for mobile devices.

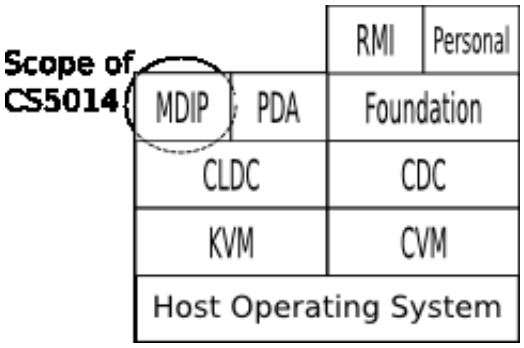

Figure 1: J2ME General Architecture

2.2 MIDlet Life Cycle

javax.microedition.midlet.MIDlet is an abstract class defines three life cycle methods:
pauseApp(), startApp(), and destroyApp().
MIDlet cycles: Paused, Active, Destroyed.

- public void startApp() This method indicates that the MIDlet is moving from a paused state to an active state. Here, the MIDlet will typically initialize any objects that are required while the MIDlet is active, and set the current display.
- public void pauseApp() This method is called when the MIDlet is moving from an active state to a paused state. This means that it will pause any threads that are currently active, as well as optionally setting the next display to be shown when the MIDlet is re-activated. Data can be persisted, if necessary, and retrieved later when the MIDlet is activated again.
- public void destroyApp(boolean unconditional) This method indicates that the MIDlet is moving to the destroyed state. It should free or close all resources that have been acquired during the life of the MIDlet. In addition, the method should persist any data that it wishes to save for future use.

2.3 Run a MIDlet

Hard but Safer Way:

1. Write the MIDlet (Java program)
2. Compile the MIDlet's source code (javac)
   javac -bootclasspath c:\j2me\midp2.0fcs\classes HelloWorld.java
3. Preverify the MIDlet's class file
   preverify -classpath c:\j2me\midp2.0fcs\classes -d HelloWorld
4. Package the application in a JAR file (optional)
   jar -cvf HelloWorld.jar HelloWorld.class
5. Run the MIDlet

Simple Way by Using a J2ME Toolkit (several emulators):

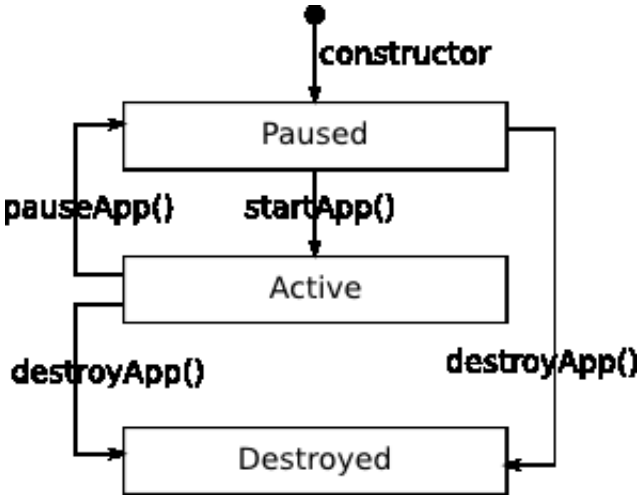

Figure 2: MIDlet Life Cycle

1. Write the MIDlet (Java program).
2. Create a new J2ME project.
3. Add the java file in the src folder of the project.
4. Build the project.
5. Run the project using the emulator you want.

3 Lecture 19 January 2007

Course is very practical 50% or marks for final exam, 50% for practicals. There will be a practical each week. The APIs will be taught: J2ME, M3G (probably) & Java Messaging (possibly, if there is time). Business Intelligence students to deliver the game with some elements of intelligence.

Java development rules:

1. Create an Object (ClassName obj = new ClassName(...))
2. Obj.methodName(...) (non-Static methods)
3. ClassName.methodName(...) (Static methods)

Java naming conventions:

- ClassNames begin with uppercase letter and each word in the name starts with an uppercase letter.
- packagename all lowercase, usually one word.
- CONSTANTS all uppercase.
- methodName, variableName etc begin with lowercase letter and each word in the name starts with an uppercase letter.

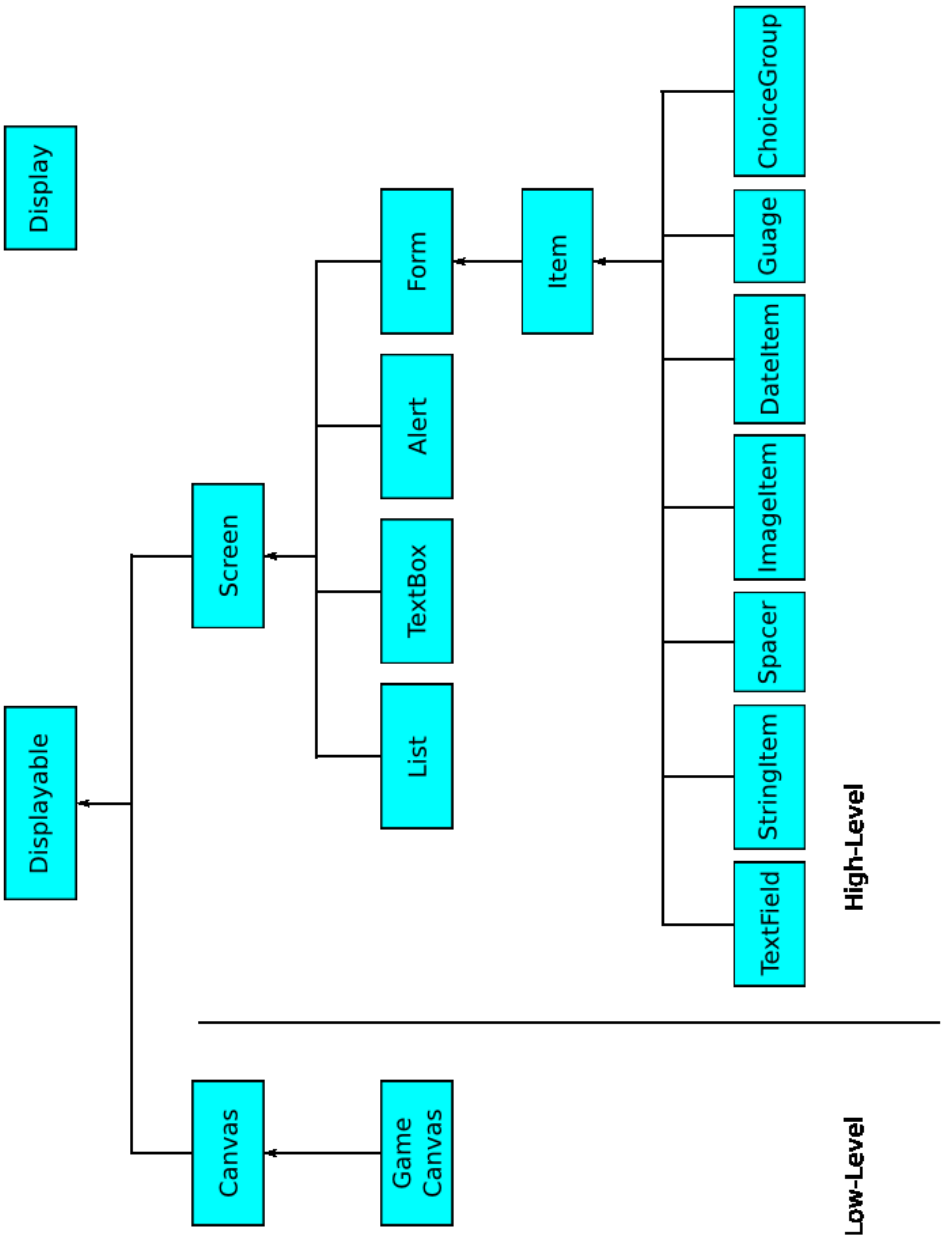

Figure 3: Class Structure

3.1 High Level User Interfaces

Displayable Objects to display on the screen.

List A sequence of choices.

TextBox Space to work with a small amount of text.

Alert Screen to display an alert message.

Form Screen that contains Items.

- **TextField** One row box to manipulate text.
- **StringItem** Label.
- **Spacer** Empty area.
- **ImageItem** Image.
- **DateItem** Date and Time.
- **Guage** To get a number or provide an animation.
- **ChoiceGroup** Item which contains selections.

Display is the screen manager.

Displayable generates objects that can be displayed on the screen. Displayable objects have a title and a ticker, some commands associated and a Command Listener.

Displayable methods: getTitle(), setTitle(str), getTicker(), setTicker(str) getHeight(), setHeight(h), getWidth(), setWidth(w), sizeChanged(w,h) addCommand(cmd), removeCommand(cmd), setCommandListener(l).

Display is the manager of device screen. It contains methods to get the device properties and to set the Displayable obj.

Display methods: getDisplay(Midlet m); returns the unique Display for this midlet. getCurrent(), setCurrent(disp), setCurrent(alert, nextDisp), setCurrentItem(it) getBestImageWidth(), getBestImageHeight(), getColor(), isColor(), getBorderStyle(), vibrate() etc.

One instance per midlet in startApp() or constructor.

Commands are objects associated with Displayable objs that fire actions.

Constructors:

Command (name, type, priority);

Command(shortName, longName, type, priority);

type is one of: BACK, OK, CANCEL, STOP, HELP, SCREEN

Methods only to get the name, type, priority.

getCommandType(), getPriority(), getLabel().

Steps to work with commands:

The Midlet must implement CommandListener.

The commands must be added to Displayable.

The Displayable object must be registered with CommandListener

The event must be dealt with in

commandAction(Command c, Displayable s).

In this function each Command must have a block like:

```
If (c == myCmd && d == myDisplay) {
  do something
}
```

3.2 Class Definitions

3.2.1 TextBox

TextBox-s are Screens to manipulate long strings (<256).

One constructor: TextBox(title, initText, maxSize, constraints);

ANY, NUMERIC, DECIMAL, PHONENUMBER, EMAILADDR, URL

Several method to process the text: getString(), setString(str), insert(str, pos), delete(offset, nr), size()

Several method for the TextBox properties: setTitle(str), setConstraints(conts), getMaxSize(), setMaxSize(), etc.

No append method

3.2.2 Alert

Alerts are Screens that contains an informative message. Alerts are displayed for a certain time or forever (modal).

Constructors: Alert(title); Alert(title, alertText, alertImage, alertType)

AlertType-s: ALARM, CONFIRMATION, ERROR, INFO

Alert methods to get its information and to work with commands: getString(), setString(str), getImage(), setImage(im), getType(), setType(alertType), getTimeout(), setTimeout(time), addCommand(cmd), removeCommand(cmd), setCommandListener()

3.2.3 List

Lists are Screen which contains some selectable items. The items have a name and optionally an icon.

Lists can be:

1. MULTIPLE multiple selections are allowed.
2. EXCLUSIVE one single selection.
3. IMPLICIT after selection a command is fired.

Constructors: List (String title, int listType);

List(String title, int listType, String[] stringElems, Image[] imageElems);

Methods to populate the list: set(itemPos, itemName, itemImage); set(itemPos, itemName, itemImage);

append(itemName, itemImage); delete(itemPos); deleteAll();

Methods to work with selections: isSelected(pos); getSelectedIndex(); setSelectedIndex(pos, bool); getSelectedFlags(boolean [] bools); setSelectedFlags(boolean [] bools); size() the size of the list

Handling Command Event from Implicit Lists

Implicit lists generate Command events that are identified by List. SELECT_COMMAND

3.2.4 Form

Forms are container screens which can hold any arbitrary collection of Items. Items can be TextField, StringItem, Spacer, ImageItem, DateItem, Gauge, ChoiceGroup and CustomItem.

Constructors: Form (String title); Form (String title, Item [] items);

Methods to manage Items: append(Item it); append(String str); append(Image im); set(int index, Item it); insert(int index, Item it); delete(int index); size();

Form Commands: Forms inherit from Displayable the Command methods.

addCommand(Command c); removeCommand(Command c);

setCommandListener(CommandListener cl);

Form Layout:

StringItems and Images are displayed on horizontal.

The other Items are usually displayed on vertical.

Control the Layout using methods:

getMinimumWidth(); getMinimumHeight();

getPreferredWidth(); getPreferredHeight(); setPreferredSize(); setLayout(int layout);getLayout();

Best to let the layout manager on the device use its default settings. any overrides which we do may not work on all devices.

3.2.5 StringItem and Spacer

StringItems are simple text labels. They can also have functionality if they have ItemCommands associated with them.

Constructors: StringItem(String label, String text); StringItem(String label, String text, int appearance); appearance can be PLAIN, HYPERLINK, BUTTON.

Methods: getText(); setText(text); setLabel(label); setPreferredSize(w,h); setFont(font); getFont();

Spacer represents empty space that can be used in Form.

Spacer(minWidth, minHeight);

3.2.6 TextField

TextFields are Items to manipulate one line texts.

Constructors: TextField(label, text, maxSize, constraints) constraints are combination of: ANY, DECIMAL, EMAILADDR, NUMERICAL, URL, PHONENUMBER PASSWORD, SENSITIVE, UNEDITABLE, NON_PREDICTIVE etc

Methods for TextField info: getLabel(); setLabel(label); getMaxSize(); setMaxSize(size); getConstraints(); setConstraints(int const); size()

Methods to manipulate the text: getString(); setString(text); insert(text, offset);delete(offset, nr);

3.2.7 DateField

DataFields are Items to manipulate (get, set) dates and times.

Constructors: DataField(String label, int mode); DataField(String label, int mode, TimeZone timeZone); where mode is DATE, TIME OR DATE_TIME;

Methods of DateField: getLabel(); setDate(Date d); getDate();

These methods work with the Date class from java.util.*.

If Date d is a Date obj then d.getTime() represents the number of milliseconds since 01/01/1970 00:00.

3.2.8 Gauge

Gauges are Items to read an integer value (J2ME sliders).

Interactive Gauge - Users can modify the Gauge current position.

Non-Interactive Gauge - Graphics of something happening

Continuous - Gauge is updated continuously

Incremental - Gauge is updated only when something happens.

Constructors: Gauge(String label, boolean interactive, int maxValue, int initValue);

Non-Interactive Gauge: maxValue - INDEFINITE, initValue - INCREMENTAL_UPDATING, INCREMENTAL_IDLE, CONTINOUS_RUNNING, CONTINOUS_IDLE

Methods: getLabel(); setLabel(label); getMaxValue(); setMaxValue(max); getValue(); setValue(val);

3.2.9 ChoiceGroup

ChoiceGroup are list Items that contain selectable Items (Lists for Forms).

- MULTIPLE - multiple selections.
- EXCLUSIVE - single selection.
- POPUP - popup selection.

Constructors: ChoiceGroup(String label, int choiceType); ChoiceGroup(String label, int choiceType, String [] names, Image[]icons);

Methods - similar to List class

Manipulate the ChoiceGroup: set(itemPos, itemName, itemImage); set(itemPos, itemName, itemImage); append(itemName, itemImage); delete(itemPos); deleteAll();

Get Selections: getSelectedIndex();setSelectedIndex(pos, bool); getSelectedFlags(boolean [] bools); setSelectedFlags(boolean [] bools); size() - the size of the list.

3.3 Golden Rules To Work With Forms

For Portability Reasons:

1. Do not associate Commands to your Items.
2. Do not overload the layout - leave the implementation to decide.

For Technical Reasons.

1. Develop separate classes for your Forms (if they have many Items).
2. The midlet should then create the form and do navigation.

3.4 MIDlet Skeleton

1. Declarations for the MIDlet elements.
2. Constructor:
   (a) Get the display.
   (b) Construct the interface element.
   (c) Load the resources.
3. startApp - set the first Displayable.
4. pauseApp - empty for now.
5. destroyApp - empty for now.
6. commandAction - to deal with the Commands.

3.5 References

Problems to Solve:

Read about the UI Components

References:

1.Jonathan Knudsen, Wireless Java, APress, 2001
2. Chapter 8 in Core J2ME from UCC e-Library

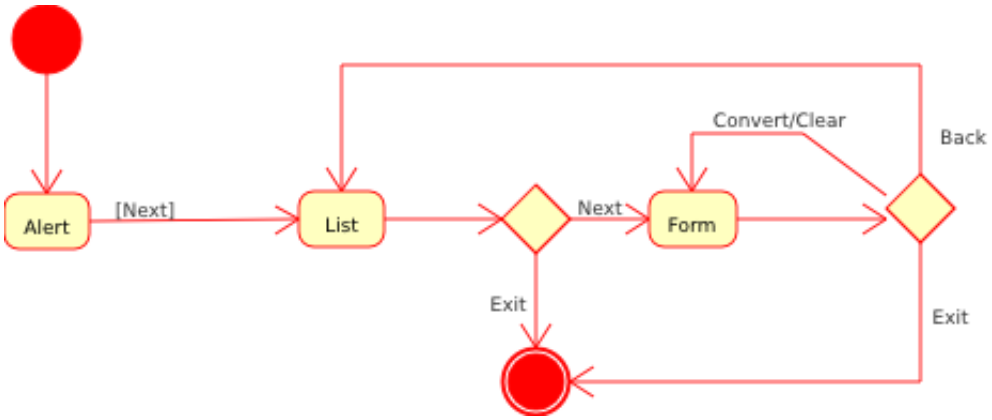

Figure 4: Process Flow

4 Lecture 26 January 2007

Low-Level User Interface

4.1 Canvas

Canvas is a Displayable capable of:

- drawing and filling some basic shapes, Images etc.
- working with Colors, Fonts, types of Lines etc.
- allow the user to discover the device features and to design UI according to them in paint().
- event handling methods for the key events, repaint etc.

Remark: Canvas fits nicely with the other Displayables.
Constructor: Canvas();
Methods to work with the screen sizes:
getWidth(); getHeight(); setFullScreenMode(boolean mode); sizeChanged(w,h)

4.1.1 Important methods to paint:

public void paint(Graphics g) → abstract method to draw the Canvas.
public void repaint() → method called in the Canvas to trigger paint.
public void repaint(int x, int y, int w, int h) → method called to paint only a part of the canvas.
public void serviceRepaints() → force the midp implementation to service immediately repaints
Drawing only parts of the screen is called "Clipping". This is an important feature when "rendering" is complicated especially in Games.
Canvas Space:
Coordinate system with integers representing pixel positions.
Origin is top-left corner; translation of it is possible using g.translate()

4.2 Graphics Class

4.3 Drawing and Filling Methods

Graphics has a few methods to draw and fill basic shapes.

drawLine(x1,y1,x2,y2); fillTriangle(x1,y1,x2,y2,x3,y3); drawRect(x, y, w, h); fillRect(x, y, w, h); drawRoundRect(x, y, w, h, arc_w, arc_h); fillRoundRect(x, y, w, h, arc_w, arc_h); drawArc(x, y, w, h, startAngle, arcAngle); fillArc(x, y, w, h, startAngle, arcAngle);

The shapes are drawn with the current line style.

Two styles: SOLID, DOTTED.

getStrokeStyle(); setStrokeStyle(int style);

4.3.1 Color Methods

Graphics has methods to manipulate the color that is used to draw.

Colors are integers (4 bytes) whose bytes represent:

Unused	Red	Green	Blue

Methods to work with colors:

setColor(); getRedComponent(); getGreenCompenent(); getBlueComponent(); setColor(int color); setColor(inr r, int g, int b)

Different color capabilities on mobile devices from 1 bit to 24 bit color.

The midp implementation decides on how the color is represented.

4.3.2 Drawing with Strings

Graphics can draw Strings of arrays of char with the current color and font.

The string is drawn at a location using a type of anchoring.

drawString(str,x,y,anchor); drawString(str, offset, len, x, y, anchor); drawChar(ch, x, y, anchor); drawChars(chars, offset, len, x, y, anchor);

The anchor is a combination of the fields

TOP, BOTTOM, BASELINE, VCENTER, LEFT, RIGHT, HCENTER. E.g Top-left is Graphics.TOP|Graphics.LEFT (| is bitwise OR).

The class Font offers few flavours to set the font face, style and size:

Face: FACE_SYSTEM, FACE_MONOSPACE, FACE_PROPORTIONAL

Style: STYLE_PLAIN, STYLE_BOLD, STYLE_ITALICS, STYLE_ UNDERLINE

Size: SIZE_SMALL, SIZE_MEDIUM, SIZE_LARGE.

Methods:

getFont(); setFont(); getFace(); getStyle(); getSize(); isPlain(); isBold(); etc.

4.4 Some Tips, Golden Rules

Golden Rules:

- Develop a separate class to extend Canvas and override paint().
- First thing to do in paint() is to set the background, drawing color, etc.
- Use the whole screen for drawing your graphics.
- Use a form before the canvas to get the elements of the graphics.

Tip:

- repaint() paints over the previous content of the Canvas so the elements might overlap. Drawing the background first removes the previous content in Canvas.

4.5 Graphics in Form (CustomItem)

CustomItem is a Form Item in which users can draw (Canvas for Forms).
Users can use CustomItem to create their own:

- customised items to display more Item components e.g. tables, trees etc.
- graphics to draw shapes, images, strings etc.

CustomItem is an abstract class with 5 methods to override:

1. int getPrefContentWidth(int w); → preferred sizes to display the comp
2. int getPrefContentHeight(int h);
3. int getMinContentWidth(int h); → minimum sizes to display the comp
4. int getMinContentHeight(int h);
5. void paint(Graphics g); → draw the CustomItem

CustomItem has the same methods as Canvas to work with repaint(), key events etc.

4.6 Recursive Methods

A method can call any (known) methods from the class or libraries.
A method is recursive when invokes itself.
Important Rule: A recursive method must have a termination step that solves directly the problem.

4.6.1 Principles of Turtle Geometry

1. Define Recursively the figure F_n.
   - Termination: give the shape of F_0 (point, line, etc)
   - Recursion: define F_n based on F_{n-1}.
2. Use a Turtle object to draw the figure.
   - the direction of the Turtle object must be the same.

Turtle Geometry represents the simplest way to construct geometrical fractals.
Important Fractals: Trees, Koch's curves, Sierpinski's curves etc.

4.6.2 Binary Tree

Consider $T(n, l)$ the binary tree.
n - the age or order; l - the length.
$T(0, l)$ = a line with the length l.
$T(n, l)$is defined as follows:

1. construct the trunk
2. left 45° ($\frac{\pi}{4}$)
3. construct $T(n-1, l/2)$
4. right 90° ($\frac{\pi}{2}$)
5. construct $T(n-1, l/2)$
6. left 45° ($\frac{\pi}{4}$)
7. go back to the root

4.6.3 Koch's Fractal

Consider $K(n, l)$ the Koch curve.
$K(0, l)$ = a line.
$K(n, l)$ is defined as follows:

1. construct $K(n-1, l/3)$;
2. left 60° ($\frac{\pi}{3}$); construct $K(n-1, l/3)$
3. right 120° ($\frac{2\pi}{3}$); construct $K(n-1, l/3)$
4. left 60° ($\frac{\pi}{3}$); construct $K(n-1, l/3)$

The snow flake $F(n, k)$

1. construct $K(n, l)$; left 120° ($\frac{2\pi}{3}$);
2. construct $K(n, l)$; left 120° ($\frac{2\pi}{3}$);
3. construct $K(n, l)$; left 120° ($\frac{2\pi}{3}$);

$F(n, k)$ is a fractal representing an infinite curve bounding a finite area.

4.6.4 Sierpinski's Fractal (curve)

Consider $S(n, l)$ the Sierpinski's curve.
Find $d = sqrt(l)$;
$S(0, l)$ = nothing.
$S(n, l)$is defined as follows:

1. construct $S(n-1, l)$; right 45° ($\frac{\pi}{4}$); forward d
2. right 45° ($\frac{\pi}{4}$); construct $S(n-1, l)$;
3. left 90° ($\frac{\pi}{2}$); forward l; left 90° ($\frac{\pi}{2}$);
4. construct $S(n-1, l)$;
5. right 45° ($\frac{\pi}{4}$);forward d; right 45° ($\frac{\pi}{4}$)
6. construct $S(n-1, l)$;

$S(n, k)$ is a fractal representing an infinite curve bounded by finite area.

4.7 Problems to Solve:

Read about the CustomItem, Graphics and Canvas.
References:
1.Chapters 7 and 10 in Wireless Java.
2. Chapter 9 in Core J2ME from UCC e-Library

5 Lecture 2 February 2007

5.1 Image

J2ME Image can be considered as a sequence of r,g,b byte values.

J2ME Images can be classified in

- Immutable Images: created from external sources.
- Mutable Images: created as blank and generated afterwards.

Immutable Images can be created using createImage() from:

- an existing Image.
- a part of an existing Image.
- an external png Image given as String.
- A sequence of RGB/Color values.

Empty Mutable Images are created by giving the width and height.

Drawing an image can be done using drawImage(im, x, y, anchor).

The external png images must be located in the res folder of your project.

J2SE image class works with gif, png & jpeg. Java Advanced Images (planar images) can work with almost all image formats.

5.1.1 Flavours of createImage

Image does not have a constructor but there are several flavours of createImage().

static Image createImage(Image im); clone an Image

static Image createImage(Image im, int x, int y, int w, int h, int trans); create a new Image from part of im using a transform

static Image createImage(String name); create the image from a png file

static Image createImage(InputStream is); create the image from an InputStream from a png file

static Image createImage(int w, int h); create an empty image with the sizes w and h.

static Image createImage(int [] rgb, int w, int h, boolean alpha); create an image from an array of w*h ints.

static Image createImage(byte [] data, int offset, int length); create an image from an array of bytes representing r,g,b

Example: Load an image from an external png file

```
private Image loadImage(String name){
  Image im = null;
  try {
    image = createImage(name)
  }
  catch(IOException ioe) {
    System.out.println(ioe);
  }
return image;
}
```

5.1.2 Some other methods

Image has methods to get sizes and Graphics, and to text if mutable.

int getWidth(); int getHeigth(); Graphics getGraphics(); boolean isMutable();

Example: Create a mutable Image from another Image

```
public Image createMutableImage(Image image) {
  Image newImage = null;
  try {
    newImage = createImage(image.getWidth(), image.getHeight());
  }
  catch(IllegalArgumentException iae) {
    System.out.printf(iae);
  }
  Graphics g = newImage.getGraphics();
  g.drawImage(image, 0,0, Graphics.TOP|Graphics.LEFT);
  return newImage;
}
```

5.1.3 Drawing Images

Graphics has a few methods that can draw Images.

void drawImage(Image im, int x, int y, int anchor)

anchor is TOP, BOTTOM, BASELINE, VCENTER, LEFT, RIGHT, HCENTER.

Advanced Drawing method

void drawRegion(Image im, int srcX, int srcY, int w, int h, int transf, int destX, int destY, int anchor);

transf is TRANSF_NONE, TRANSF_ROT90, TRANSF_ROT180, TRANSF_ROT270, TRANSF_MIRROR, TRANSF_MIRROR_ROT90, etc

Drawing Image as Array of Integers

void drawRGB(int [] rgbData, int offs, int len, int x, int y, int w, int h, boolean alpha)

Note These three methods come from the Graphics class.

5.1.4 Special Features of Drawing Images

Blitting - copying a region of the back screen to another location.

void copyArea(int srcX, int srcY, int w, int h, int destX, int destY, int anchor);

Clipping - drawing to a specified region; any drawing outside of clipping rectangle is not displayed.

setClip(int x, int y, int w, int h);

getClipX(); getClipY(); getClipWidth(); getclipHeigth();

Grabbing Image data

getRGB(int [] data, int offset, int length, int x, int y, int w, int h);

The method collects the data from the image area of (x,y,w,h) and stores it in the array data as sequence of colors. The methods throws several exceptions.

5.2 Mobile Animation

Animation = displaying a sequence of images with a delay time.

The canvas content can be images, shapes, etc.

Any Java animation requires:

- double buffering, to avoid flickering.

- threads, to speed up

MIDP 2.0 might provide these elements implemented as default.

We can test if a canvas is double buffered using isDoubleBuffered();

5.2.1 Double Buffering

Draw the background + Repaint = Flickering.

Double Buffering uses a off-screen buffer (image) to prepare the image.

Back buffer: draw the background + draw the new frame.

Front buffer (the screen): draw the back buffer image when ready.

Steps to work.

1. Create a virtual screen.
   Image backBuffer = createImage(size().width,size().height);
   Graphics backGC = backBuffer.getGraphics();

2. Do the graphics (clear + draw) in the virtual screen backGC.

3. Paste the image backBuffer in g when ready.

```
public void paint (Graphics g) {
  backGC.setColor(255,255,255);
  backGC.fillRect(0,0,getWidth(),getHeight());
  backGC.drawImage(im[imageIndex % 4], 0, 0, 0);
  g.drawImage(backBuffer,0,0,0);
}
```

5.2.2 Animation Threads

Midlets have a main thread that is responsible with handling the UI.

Golden Rule:

1. Any expensive computation must be executed in a separate thread.

2. For animation use an "Animation Thread" to achieve:

   - drawing the animation Frames using eventually double buffer.
   - delaying (synchronising) the Frames

Use threads in the animation canvas.

- Implement Runnable.
- Create a new Thread.
- Start this Thread in start() and stop it stop().
- Override the method run() to describe the repetition
  - Delay for a while
  - Go to the next image index.
  - Repaint()

5.3 Image Processing

Image Processing = Change the structure of an image (Color, Sizes).
Classification:

Point Transform Change pixel colors + Keep pixel positions.

Geometrical Transform Keep pixel colors + Change pixel positions.

Spatial Transform Neighbours colors change the pixel color.

Global Transform All the image pixels give the value of a pixel.

Colours are integers (4 bytes) whose bytes give r, g, b.
Between the colour col and the components r, g, b we have:
r = (col & 0xff0000) >> 16; g = (col & 0xff00) >> 8; b = col & 0xff; col = r << 16 + g << 8 + b;

5.3.1 Working with Positions

Geometrical Transformations change only positions and preserve colours.
The element index of the array pixels gives the pixel location
row = index / w and col = index % w.
If the pixel (row, col) has the following index in the array pixel1
index = row * w + col Geometrical Transformations also change the image sizes.

6 Lecture 9 February

Presentation by Sean O'Sullivan (sos@rococosoft.com) on Bluetooth.
This week's lecture will be completed at next week's lab.

6.1 Animated Splash Screen Assignment

Design a splash screen from:

1. A couple of Images representing screenshots from the game.
2. One or two image processing transformations.
3. Our own ideas on how to animate.

7 Lecture 16 February 2007

7.1 Games

Games = Enhanced Animation + User Interaction.
The principles of Animation still apply:

- Double buffer if not available
- Usage of the animation thread.

User Interaction

- Deal with it in keyPressed() or keyReleased()
- Map the key to a Game Action key.
- Compare with: UP, DOWN, LEFT, RIGHT, FIRE, GAME_A, GAME_B, GAME_C, GAME_D

7.1.1 Managing a game:

Game Front End must have

- Splash screen with some animation
- Menu and SubMenus: New Game, Resume, Settings, Change keys, Help
- Navigation through these screens.

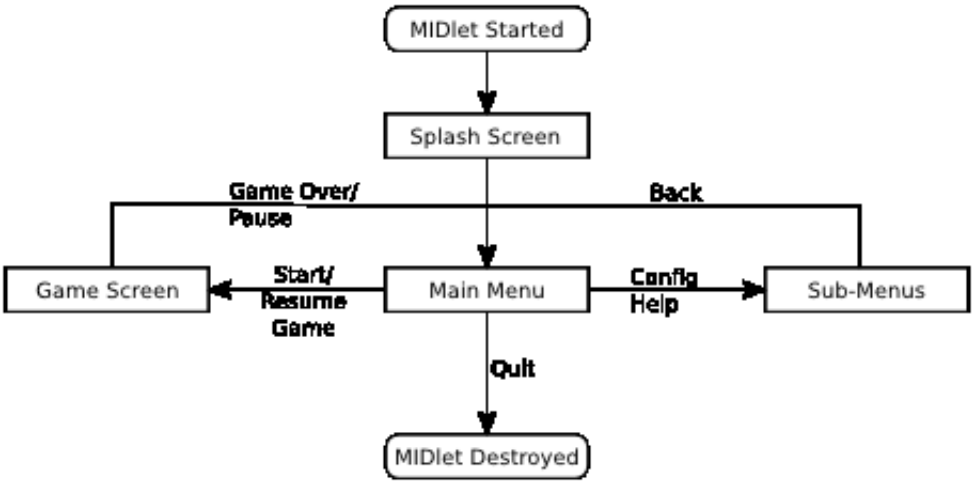

Figure 5: Game Structure

7.2 Working with Key Events.

Canvas handles events coming for keys by calling several callbacks.

Canvas has two methods to make an explicit answer to these events: void keyPressed(int keyCode); void keyReleased(int keyCode)

The key code that is passed is one of the constants: KEY_NUM0,..., KEY_NUM9, KEY_STAR, KEY_POUND.

keyCode = the value of the UNICODE char.

The keys can be mapped to game action keys like: UP, DOWN, LEFT, RIGHT, FIRE, GAME_A, GAME_B, GAME_C, GAME_D by using int getGameAction(int keyCode);

7.3 Structure of the GAME API

Specialised package to for J2ME gaming is javax.microedition.lcdui.game:

GameCanvas → Manage the animation and key polling.
Sprite → Create and manipulate animated sprites.
Layer, TiledLayer → Build background scenes
LayerManager → Manage the layers.
Huge advantages:

- Sprites are simple to create, interact each other, animate.
- Multiple layers gives nice, realistic scenes.
- Efficient rendering when, where we want it.

7.4 How GameCanvas works

Canvas rendering:

- Describe the rendering in paint().
- Repaint() starts the rendering process that is done when the midlet is ready.

GameCanvas rendering is totally different:

- Get the Graphics object from getGraphics().
- Do any type of rendering, wherever you want using the Graphics methods
- Update the GameCanvas using flushGraphics() which takes effect immediately.
  public void flushGraphics(int x, int y, int w, int h) updates only the target area.

```
Graphics g = getGraphics();
// draw what it is needed
flushGraphics();
```

7.5 GameCanvas

GameCanvas gives:

1. Simple methods to manage the canvas work:
   (a) flushGraphics(); flushGraphics(x,y,w,h);
   (b) getGraphics();
   (c) paint(Graphics g)
2. One method plus some fields to test key events:
   (a) getKeyStates()
   (b) UP_PRESSED, DOWN_PRESSED, GAME_D_PRESSED
3. One constructor GameCanvas(supressKeyEvents):
   (a) if false then the normal key event mechanism applies e.g. keyPressed(), keyReleased().
   (b) if true then the canvas polls the key states.

7.6 Polling Keys

GameCanvas can disable the key events callbacks by using a direct polling of the keys states using public int getKeyStates().

The method immediately returns the GameAction keys' status as bits "latched" in an integer values. Those bits can be tested afterward using the key fields and a game action taken.

How to test:

```
int ks = getKeyStates();
if(ks&GameCanvas.UP_PRESSED) moveUp();
else if(ks&GameCanvas.DOWN_PRESSED) moveDown();
...
flushGraphics();
```

7.7 Sprites

Sprites are the "GameCanvas" actors/protagonists/players:

- Sprites are built from a sequence of frames encapsulated in one image. Those frames can be animated to obtain animated sprites.
- Sprites can be transformed using some rotate and/or mirror transformations around a reference point.
- Sprite can handle collision detection with other sprites etc.

Constructors: Sprite(Image im); Sprite(Sprite s); Sprite(Image im, int frameW, int frameH);

Working with frames:

setFrameSequence(int [] seq); → the order of frames to create the animated sprite.

getFrameSequencelength(); → how many frames we have in the sprite.

getFrame(); setFrame(index); prevFrame(); nextFrame(); → to manipulate the frame

Collision:

collidesWith(Sprite s, boolean pixelLevel);
collidesWith(TiledLayer tl, boolean pixelLevel);
collidesWith(Image im, int x, int y, boolean pixelLevel);
defineCollisionRectangle(int x, int y, int w, int h);

Transformation and Ref Point:

getRefPixelX(); getRefPixelY(); setRefPixelPosition(x,y);
setTransform(transf);
The transformation is one of the Sprite's fields: TRANS_NONE, TRANS_ROT90

7.7.1 Constructing Sprites

The steps to construct a sprite are:

1. load the image containing the frames.
2. construct the sprite from that image using a w and h.
3. define one or even more frame seqs and set one to the sprite.

```
Image quatschImage = Image.createImage("/quatsch.png");
Sprite quatsch = new Sprite(quatschImage, 48, 48);
int [] runningSeq = {0,1,2}; int [] stillSeq = {3};
quatsch.setFrameSequence(runningSeq);
```

The animation does not take place automatically, it should be done through nextFrame():

```
quatsch.nextFrame(); // or quatsch.previousFrame();
```

7.7.2 Functionality of Sprites

Sprites can have a transformation associated with them by using setTransform(trasf);

This transformation changes the sprite's current frame around the ref point so it remains unchanged before and after the transformation.

quatsch.setTransform(Sprite.TRANS_MIRROR); → move the sprite on opposite direction.

A sprite can detect collision/proximity with another sprite/TiledLayer/Image.

Each sprite has a collision detection rectangle to detect when:

1. Two collision rectangles overlap.

2. Two opaque pixels overlap.

Example: quatsch.collidesWidth(door,false); → find if quatsch is in door's proximity.

Sprite are layers so Layer's methods can apply e.g. move();
quatsch.move(x,y);

7.8 Layers

Layers are GameCanvas objects that can be combined to create a scene.

Layer's methods deal with location, size and visibility.
Sizes: getWidth(); getHeight();
Location: getX(); getY(); setPosition(x,y); move(x,y); → the layers moves with (x,y) from the current position.
Visibility: getVisible(); setVisible(vis);

7.8.1 Managing Layers

GameCanvas scenes are built by overlapping/using several layers.

LayerManager keeps the layers ordered and sets the Viewing window.
Viewing window is the rectangle that can be seen on the screen.
Methods:
append(Layer l); insert(Layer l, int index); remove(Layer l);
getLayerAt(int index); setViewWindow(int x, int y, int w, int h);

7.8.2 TiledLayers

TiledLayers represents a grid of tiles etc that can be constructed from an image.

```
public TiledLayer(int col, int row, Image im, int tileW, int tileH)
```

The elements from the constructor can be find using:
getRows(); getColumns(); → layer sizes
getCellWidth(); getCellHeight(); → tile/cell sizes
The image im is split into small tiles that are added to the layer in some order.
setCell(col, row, tileIndex); getCell(col, row);
fillCells(col, row, numCols, numRows, tileIndex); → a tile fills a region of cells

7.8.3 Construct TiledLayers

The steps to construct a TiledLayers are:

1. Load the image with the tiles

2. Construct the TiledLayer

3. Define the map of tiles

4. Set the tiles to the cells

```
Image backgroundImage = Image.createImage("/background_tiles.png");
TiledLayer background = new Tiledlayer(8,4, backgroundImage,48,48);
int [] map = {1,2,0,0,0,0,0,0,
  3,3,2,0,0,0,5,0
  3,3,3,2,4,1,3,2
  6,6,6,6,6,6,6,6};
for(int i=0;i<map.length;i++){

    int col = i%background.getColumns();
    int row = i/background.getColumns();
    background.setCell(col,row,map[i]);

}
```

7.8.4 Animated Tiles

TiledLayers can have tiles that are animated.

The methods to work with are:

public int createAnimatedTile(int tileIndex); → return the index of a new animated.

public int getAnimatedTile(int animatedTileIndex) → return the tile index associated.

public void setAnimatedTile(int animatedTileIndex, int tileIndex); → change a tile

Example: one anim tile in the previous background

```
int firstAnimTile = background.createAnimatedTile(8); // sun
int secondAnimTile = background.createAnimatedTile(5); // gate
background.setCell(2,2, firstAnimTile);
background.setCell(10,2, secondAnimTile);
setAnimatedTile(firstAnimTile,9);
// change the tile associated to the first animated tile.
```

7.9 Putting the bits together

GameCanvas is double buffered.

The anim thread must be created.

To create the game elements:

1. construct a LayoutManager
2. define the background as Tiledlayer and add it to layManager
3. define each sprite of the game and add them to layManager

Loop the game computation

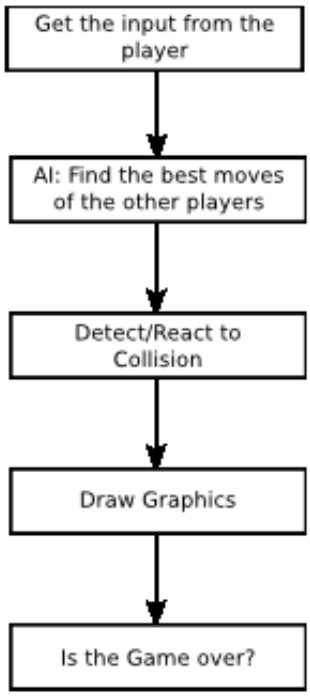

Figure 6: Game life-cycle

7.10 References

1. Chapters 11 from Wireless Java.

2. Chapter 10, 11,12,13 from J2me Game Programming (Martin Wells) UCC e-Library.

8 Lecture 23 February 2007

8.1 Persistent Data

MIDP Applications don't care where the persistent data is stored.

All they know about are small databases called Record Stores.

It is up to the device's MIDP implementation to map record stores to whatever persistent storage is available.

Record Stores contain small amounts of data.

The MIDP specification requires a minimum of 8KB for persistent storage.

Persistent storage in MIDP is centered around record stores.

A record store is a small database that contains pieces of data called records.

Represented by instances of javax.microedition.rms.RecordStore.

The scope of a record store is limited to a single MIDlet suite in 1.0 but open to more in 2.0.

A MIDlet can only access record stores that were created by a MIDlet in the same suite.

Record Stores are identified by a name, the names must be unique.

8.2 Managing Record Stores

The Record Store class serves two purposes.

- First, it defines an API for managing record stores.
- Second, it defines an API (mainly static methods) for manipulating individual records.

Opening / Creating Record Stores

To open a record store, all that is necessary is to give it a name.

```
public static RecordStore openRecordStore(String recordStoreName,
   boolean createIfNecessary) throws
   RecordStoreException, RecordStoreFullException,
     RecordStoreNotFoundException
```

If the record store does not exist, the createIfNecessary parameter determines whether a new record store will be created or not.

If the record store does not exist, and the createIfNecessary parameter is false, then a RecordStoreNotFoundException will be thrown.

The following line opens a record store named "Address".

```
RecordStore rs =  RecordStore.openRecordStore("Address", true);
```

Note: the record store will be created if it does not exist.

An open record store can be closed by calling

```
closeRecordStore() method, for example rs.closeRecordStore();
```

Note: When finished with a record store you should close it to save system resources.

Finding and Deleting Record Stores

To find out all the record stores available to a particular MIDlet suite, call the listRecordStores() method.

```
Public static String[] listRecordStores();
```

To remove a record store call the deleteRecordStore() method.

Record Store Size

Record Stores consists of Records; each record is simply an array of bytes. Public int getSize() Will return the size of the RecordStore in bytes.

```
int recSize = rs.getSize();
```

You can find out how much free space is available by calling the following method.

```
public int getSizeAvailable();
```

Note: this method returns the total available space in the record store. There is some overhead associated with each record. The method returns the amount of space available for both record data and overhead.

Version and Time Stamp

Record Stores maintain both a version number and a timestamp. The version number is updated every time the record store is modified. It can be retrieved by calling getVersion().

It also remembers the last time it was modified. The moment in time is represented as a long, which can be retrieved by calling getLastModified().

The long represents the number of seconds since midnight on January 1st 1970.

8.2.1 Working with Records

A record is simply an array of bytes. Each record in a RecordStore has an integer identification number.

The lengths of records in a store are not usually similar

Adding Records

To add a record, supply a byte array to the addRecord() method.

```
public int addRecord(byte[] data, int offset,
int numBytes) throws RecordStoreNotOpenException,
RecordStoreException, RecordStoreFullException
```

The record added will be numBytes long, taken from the data array starting at offset. The new records ID will be returned.

Most other record operations need this ID to identify a particular record.

Adding a Record Example

The following code fragment illustrates the process of adding a new record to a record store named rs.

A byte array is created from a String, and then the entire byte array is written into a new record.

```
String record = "THIS IS A NEW RECORD"
byte[] data = record.getBytes();
int id = rs.addRecord(data,0,data.length);
```

We can store only byte[] so that different formats must be converted to byte[]:

- String str.toByteArray(); or str.getBytes();
- For other formats, just convert them to a String (e.g. "" + object)

Retrieving Records

A record can be retrieved by supplying the record ID to the following method.

```
public byte[] getRecord(int RecordId) throws
RecordStoreNotOpenException, InvalidRecordIDException,
RecordStoreException
```

An alternative method places the record data into an array that you supply. It also returns (int) the number of bytes copied into the buffer.

```
public int getRecord(int RecordId, byte[] buffer, int offset)
  throws RecordStoreNotOpenException, InvalidRecordIDException,
    RecordStoreException
```

Retrieving Records Example

Given an RecordStore rs and a record ID id, a record may be retrieved in the following way.

```
byte[] retrieved = new byte[rs.getRecordSize(id)];
rs.getRecord(id,retrieved,0);
String retrievedString = new String(retrieved);
```

Note: if you are pulling a large number of records from the RecordStore, then creating a byte[] array each time is inefficient.

One solution to this is to create a buffer as large as the largest record in the RecordStore.

Deleting and Replacing Records

To delete a record from a RecordStore call the method deleteRecord(int id).

To replace the data of a existing record a call to the following method is necessary.

```
public void setRecord(int recordId, byte[] newData, int offset,
  int numBytes) throws RecordStoreNotOpenException,
InvalidRecordException,RecordStoreException,
RecordStoreFullException
```

RecordStore Information

The RecordStore keeps an internal counter that it uses to assign record IDs.

The ID of the next record can be found by calling getNextRecordID().

```
int recordID = rs.getNextRecordID();
```

The number of records existing in the RecordStore can be determined by calling getNumRecords().

```
int numRecords = rs.getNumRecords();
```

Some Simple Methods

```
private RecordStore rs = null;
static final String REC_STORE = "db_1";
// Method to Open a Recod Store
public void openRecStore() {
  try {rs = RecordStore.openRecordStore(REC_STORE, true);}
  catch (Exception e){System.out.println(e.toString()); }
}
// Method to Close a Record Store
public void closeRecStore()
{
try{rs.closeRecordStore();}
catch (Exception e){System.out.println(e.toString());}
}
// Method to delete a Record Store
public void deleteRecStore() {
  if (RecordStore.listRecordStores() != null) {
    try { RecordStore.deleteRecordStore(REC_STORE);}
    catch (Exception e){System.out.println(e.toString());}
    }
}
public void writeRecord(String str) {
  byte[] rec = str.getBytes();
  try{rs.addRecord(rec, 0, rec.length);}
  catch (Exception e){System.out.println(e.toString());}
}
public void readRecords() {
  try {
```

```
    byte[] recData = new byte[10]; int len;
    for (int i = 1; i <= rs.getNumRecords(); i++) {
      if (rs.getRecordSize(i) > recData.length)
        recData = new byte[rs.getRecordSize(i)];
        len = rs.getRecord(i, recData, 0);
        System.out.println("Record #" + i + ": " +
           new String(recData, 0, len));
      }
    }
    catch (Exception e){System.out.println(e.toString());}
  }
```

RecordStore Example

```
import javax.microedition.rms.*;
import javax.microedition.midlet.*;
public class ReadWrite extends MIDlet {
  private RecordStore rs = null;
  static final String REC_STORE = "db_1";
  public ReadWrite() {
  openRecStore(); // Create the record store

  // Write a few records and read them back
  writeRecord("J2ME and MIDP");
  writeRecord("Wireless Technology");
  readRecords();
  closeRecStore(); // Close record store
  deleteRecStore(); // Remove the record store
}
public void destroyApp(boolean unconditional) { }
public void startApp() {
  destroyApp(false);
  notifyDestroyed();
}
public void pauseApp() { }
```

Listening for Record Changes

Objects can listen for changes to a record store by registering themselves as listeners.

The listener interface is javax.microedition.rms.RecordListener. The following two methods allow for the adding and removing of a listener.

```
public void addRecordListener(RecordListener listener)
public void removeRecordListener(RecordListener listener)
```

The RecordListener interface has three methods: recordAdded(), recordChanged(). and recordDeleted().

The above methods are called whenever a record is added, changed, or deleted. Each method is passed the RecordStore involved and the ID of the record in question.

8.2.2 Performing RecordStore Queries

Just like in a database it is possible to carryout a query. To achieve this method enumerateRecords() is required.

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

The enumerateRecords() method returns a sorted subset of the records in a RecordStore. The RecordFilter determines which records will be included in the subset, while the RecordComparator is used to sort them.

The returned RecordEnumeration allows you to navigate through the returned records.

RecordFilter

When the enumerateRecords() method is called on a RecordStore, each of the record's data is retrieved. RecordFilter has a single method, matches(), which is called for each record.

The record filter examines the record data and should return true, if the record should be included in the results returned by from the enumerateRecords() method.

```
public class SevenFilter implements
   javax.microedition.rms.RecordFilter {
  public boolean matches(byte[] candidate) {
    if(candidate.length == 0) return false;
    if(candidate[0] == 7) return true;
    else return false;
  }
}
```

The above example will only select records whose first byte of data is 7.

RecordComparator

The RecordComparator is used to determine the order of two sets of record data. To implement the interface required the definition of just a single method.

```
public int compare(byte[] rec1, byte[] rec2)
```

The method examines the data contained in rec1 and rec2 and determines which of them should come first in a sorted list. One of the following constants should be returned.

PRECEDES indicates that rec1 should come before rec2.

FOLLOWS indicates that rec1 should come after rec2.

EQUIVALENT signals that rec1 and rec2 are the same (in terms of sorting).

RecordComparator Example

```
public class SimpleComparator implements
   javax.microedition.rms.RecordComparator{
  public int compare(byte[] rec1, byte[] rec2) {
  int limit = Math.min(rec1.length, rec2.length);
  for(int i = 0; i < limit; i++) {
    if(rec1[i] < rec2[i])
      return PRECEDES;
    else if(rec1[i] > rec2[i])
      return FOLLOWS;
  }
  return EQUIVALENT;
  }
}
```

The example above compares each byte of the given records and sorts them numerically. This is carried out up to the length of the shortest record.

If both records have the same data up to the length of the shorter record then they are deemed EQUIVALENT.

Working with RecordEnumeration

The main function of RecordEnumeration is to allow you to iterate through the records retrieved from the RecordStore.

RecordEnumeration allows you to scroll through its contents in both forward and backward directions. You can also examine the next or previous record ID.

RecordEnumeration also allows you to keep its data synchronized with the actual RecordStore. Behind the scenes this is accomplished by registering the RecordEnumeration as a listener for RecordStore changes.

To find out if there is another record by calling the hasNextElement() method.

If the record exists, its data can be retrieved by calling the following method.

```
public byte[] nextRecord() throws
InvalidRecordIDException,
RecordStoreNotOpenException,
RecordStoreException
```

The ID of the next record may be obtained by calling the following method.

```
public int nextRecordId() throws
InvalidRecordIDException
```

One problem with the nextRecord() and nextRecordId() methods is that they both advance the RecordEnumeration on to the next record.

If you want to retrieve both the ID and the data for the next record, you need to call nextRecordId() and then retrieve the record data directly from the RecordStore.

Typical Example of RecordEnumeration.

```
RecordStore rs, RecordFilter rf, RecordComparator rc
RecordEnumeration re = rs.enumerateRecords(rf,rc,false);
while(re.hasNextElement()) {
  Byte[] recordData = re.nextRecord();
  . . . .
}
```

RecordEnumeration previousRecord

Not only can you move forward with nextRecord() you can also move backwards using hasPreviousRecord(), previousRecord() and previousRecordId().

Summary: there are five ways to move the current position in the RecordEnumeration.

nextRecord(), nextRecordId(),

previousRecord(), previousRecordId().

The final method resets the record pointer back to the beginning of the selected records reset().

Once finished using a RecordEnumeration, you should release its resources. This can be done by calling destroy(), after the call the RecordEnumeration is no longer usable.

RecordEnumeration Updating

In a multithreaded environment, it is possible that a RecordStore will change at the same time you are iterating through a RecordEnumeration for the same RecordStore.

Calling the rebuild() method, will explicitly rebuild the RecordEnumeration based on the RecordFilter and RecordComparator originally specified.

An alternative is to request the RecordEnumeration to automatically update with any changes to the underlying RecordStore. This can be achieved by passing true to the keepUpdated parameter of the RecordStore's enumerateRecords() method.

To find out if a RecordEnumeration is automatically updated you can call the isKeptUpdated() method.

You can also change its state by calling keptUpdated()

Automatically updating RecordEnumerations is an expensive operation. Each time the RecordStore is changed, the RecordEnumeration is rebuilt.

If there are a lot of changes to the RecordStore, the rebuilding of the RecordEnumeration could take up valuable computation cycles.

Simple Sort of Strings

```
public void readRecords() {
  try {
    if (rs.getNumRecords() > 0) {
      Comparator comp = new Comparator();
      RecordEnumeration re = rs.enumerateRecords(null, comp, false);
      while (re.hasNextElement()) {
        String str = new String(re.nextRecord());
        System.out.println(str);
      }
    }
  }
  catch (Exception e){System.out.println(e.toString()); }
}
class Comparator implements RecordComparator {
  public int compare(byte[] rec1, byte[] rec2) {
    String str1 = new String(rec1), str2 = new String(rec2);
    int result = str1.compareTo(str2);
    if (result == 0) return     RecordComparator.EQUIVALENT;
    else if (result < 0) return RecordComparator.PRECEDES;
    else return RecordComparator.FOLLOWS;
  }
}
```

References

Jonathan Knudsen, Wireless Java: Developing with Java 2, Micro Edition, Apress, 2001.

9 Lecture 2 March 2007

9.1 Mobile Media API

The Mobile Media API (MMAPI) is an optional package that supports multimedia applications on J2ME-enabled devices.

This standard Java specification, defined by the Java Community Process (JCP) in JSR 135, is highly flexible.

It has been designed to run with any protocol and format.

It doesn't specify that the implementation must support particular transport protocols such as HTTP or Real-Time Transport Protocol (RTP), or media formats such as MP3, MIDI, or MPEG-4.

MMAPI has been designed to run on any J2ME-based virtual machine, including the CDC and CLDC VMs.

The J2ME Wireless Toolkit comes with the MMAPI.

Some of the MMAPI features are now in MIDP 2.0 in javax.microedition.media.*.

Manager - class to create a player

Player - class for the player functions

ToneControl - class to build a Tone sequence.

Volume Control - class to control the Player volume

9.1.1 MMAPI - Features

Support for Tone Generation, Playback, and Recording of Time-Based Media: The package supports any time-based audio or video content.

Small Footprint: MMAPI works within the strict memory limits of CLDC devices.

Protocol- and Content-Independent: The API is not biased towards any specific content type or protocol.

Subsettable: Developers can limit support to particular types of content, basic audio for example.

Extensible: New features can be added easily without breaking older functionality. More importantly, additional formats can be easily supported, and the framework is in place for additional controls.

9.1.2 Multimedia Processing

There are two parts to multimedia processing:

Protocol Handling: reading data from a source such as a file or a streaming server into a media-processing system.

Content Handling: parsing or decoding the media data and rendering it to an output device such as an audio speaker or video display.

To facilitate these operations, the API provides two high-level object types:

DataSource encapsulates protocol handling by hiding the details of how the data is read from its source. This object's utility methods enable the Player object to handle the content.

Player reads the data from DataSource, processes it, and renders it to an output device. This object provides methods to control media playback, including methods for type-specific controls to access features for specific media types.

MMAPI specifies a third object, a factory mechanism known as the Manager, to enable your application to create Players from DataSources, and also from InputStreams.

9.1.3 Player

The Manager object provides the method createPlayer(), which is the top-level entry point into the API. Here's an example:

```
Player player = Manager.createPlayer(String url);
```

The url specifies the protocol and the content, using the format <protocol>:<content location>.

The application uses the methods of the returned Player to control the retrieval and playback of time-based media.

The player's life-cycle includes five states: UNREALIZED, REALIZED, PREFETCHED, STARTED, and CLOSED. Six of its methods result in state transitions:

realize(), prefetch(), start(), stop(), deallocate(), close()

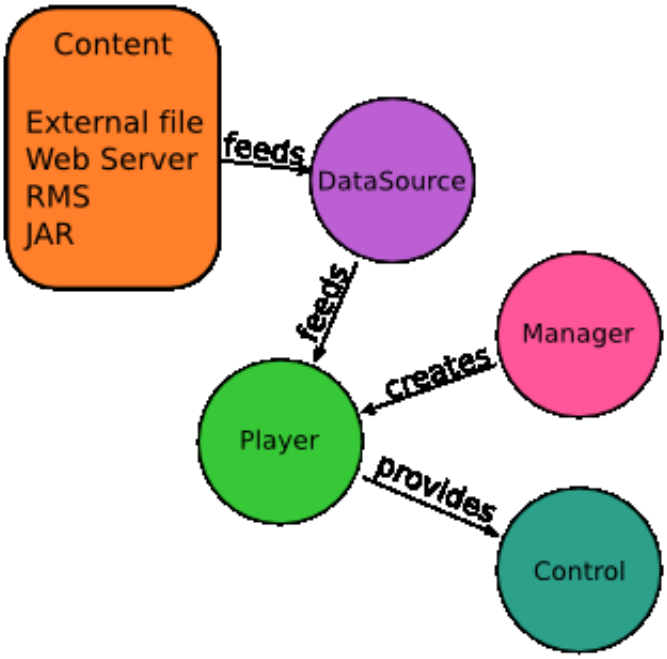

Figure 7: Data Flow

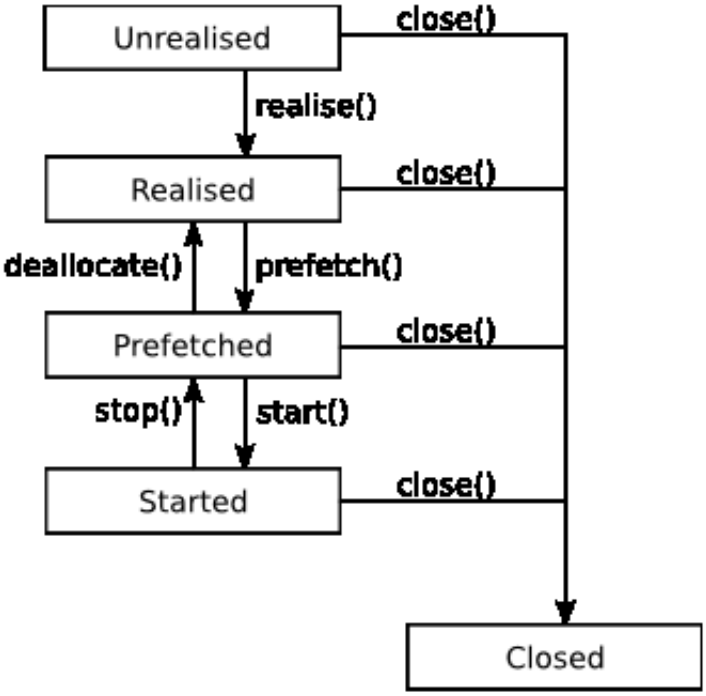

Figure 8: Player States

When a player is created, it is in the UNREALIZED state.

Calling realize() moves it to the REALIZED state and initializes the information the player needs to acquire media resources.

Calling prefetch() moves it to PREFETCHED, establishes network connections for streaming data, and performs other initialization tasks.

Calling start() causes a transition to the STARTED state, where the player can process data.

When it finishes processing (reaches the end of a media stream), it returns to the PREFETCHED state.

Calling close() moves the player to the CLOSED state.

Player Controls

A Player provides controls specific to the particular types of media it processes.

The application uses getControl() to obtain a single control, or getControls() to get an array of them.

As an example, if a player for MIDI media invokes getControl() it gets back a MIDIControl.

9.1.4 Tones

Tone generation is a characterized by frequency and duration.

This type of media is important for games and other audio applications, especially on small devices, where it might be the only form of multimedia capability available.

The Manager.playTone() method generates tones. Its implementation can be mapped to the hardware's tone generator. You specify the note, duration, and volume:

```
try {
// play a tone for 4000 milliseconds at volume 100
   Manager.playTone(ToneControl.C4, 4000, 100);
Manager.playTone(60, 4000, 100);
} catch(MediaException me) { }
```

A Player may also be created for creating a sequence of tones.

```
player = Manager.createPlayer(Manager.TONE_DEVICE_LOCATOR);
```

This type of player provides a ToneControl that can be used to program a tone sequence. This feature is available only on the more powerful devices.

The fields of ToneControl are:

- VERSION, BLOCK_START, BLOCK_END, PLAY_BLOCK, REPEAT, SET_VOLUME, SILENCE, TEMPO

The method of ToneControl is void setSequence(byte [] seq)

A tone is formed by:

Note Number: 60 is middle C and 69 is A.

Duration as multiple of "resolution" - 1/64 of 4/4 times.

```
byte [] seq ={ToneControl.VERSION,1
67, 16,
69, 16,
67, 8,
65,8
};
```

Block might be created in order to repeat them:

```
sequence = new byte[] {
ToneControl.VERSION, 1,
ToneControl.TEMPO, 22,
ToneControl.BLOCK_START, 0,
60, 8, 62, 4, 64, 4, 65, 4, 67, 4, 69, 4,
71, 4, 72, 4, 74, 4, 76, 4, 77, 4, 79, 4,
81, 4, 83, 4, 84, 4,
83, 4, 81, 4, 80, 4, 81, 4, 86, 4, 84, 4,
83, 4, 81, 4, 81, 4, 79, 4, 78, 4, 79, 4,
60, 4, 79, 4, 88, 4, 79, 4,
57, 4, 77, 4, 88, 4, 77, 4, 59, 4, 77, 4,
86, 4, 77, 4, 56, 4, 76, 4, 86, 4, 76, 4,
57, 4, 76, 4, 84, 4, 76, 4,
53, 4, 74, 4, 84, 4, 74, 4, 55, 4, 74, 4,
83, 4, 74, 4, 84, 16, ToneControl.SILENCE,
16,
ToneControl.BLOCK_END, 0,
ToneControl.BLOCK_START, 1,
79, 4, 84, 4, 88, 4, 86, 4, 84, 4, 83, 4,
81, 4, 79, 4, 77, 4, 76, 4, 74, 4, 72, 4,
71, 4, 69, 4, 67, 4, 65, 4,
64, 8, 76, 8, 77, 8, 78, 8, 79, 12, 76, 4,
74, 8, ToneControl.SILENCE, 8,
ToneControl.BLOCK_END, 1,
ToneControl.SET_VOLUME, 100, ToneControl.PLAY_BLOCK, 0,
ToneControl.SET_VOLUME, 50, ToneControl.PLAY_BLOCK, 0,
ToneControl.SET_VOLUME, 100, ToneControl.PLAY_BLOCK, 1,
ToneControl.SET_VOLUME, 50, ToneControl.PLAY_BLOCK, 1,
ToneControl.SET_VOLUME, 100, ToneControl.PLAY_BLOCK, 0,
};
```

How to Play Tone Sequences:

```
try{
  Player p = Manager.createPlayer(
    Manager.TONE_DEVICE_LOCATOR);
  p.realize();
  ToneControl c = (ToneControl)p.getControl("ToneControl");
  c.setSequence(mySequence);
  p.start();
}
catch (IOException ioe) { }
catch (MediaException me) { }
// Simple Media Playback
try {
  Player p = Manager.createPlayer(
    "http://webserver/music.mp3");
  p.setLoopCount(5);
  p.start();
}
catch (IOException ioe) { }
catch (MediaException me) { }
```

Note: The above example will play the mp3 file 5 times, to set it in a infinite loop supply -1 to the setLoopCount() method.

9.1.5 Simple Media Playback

Accessing an Audio file in the Resources Directory (included within the JAR file).

```
InputStream is = getClass().getResourceAsStream("mymusic.mp3");
Player p = Manager.createPlayer(is, "audio/mpeg");
p.start();
```

Media can also be stored in a RecordStore (typically has only 8Kb of storage!) and accessed by doing something like this:

```
InputStream is = new ByteArrayInputStream(
myRecordStore.getRecord(id));
Player player = Manager.createPlayer( is, "audio/mpeg");
player.start();
```

9.2 References

Jonathan Knudsen, Wireless Java: Developing with Java 2, Micro Edition, Apress, 2001.

http://developers.sun.com/techtopics/mobility/midp/articles/mmapioverview/

http://www.ociweb.com/jnb/jnbSep2004.html

http://www.java201.com/resources/browse/121-all.html

10 Lecture 9 March 2007

10.1 Generic Connection Framework (GCF)

There is only one class within the javax.microedition.io package called Connector.

The idea is that you pass a connection string to one of the Connector's static methods and you get back some Connection implementation.

A connection string looks like a URL, but there are various other possibilities.

The connection string socket://xyz.com:79 might open a TCP/IP connection to xyz.com on port 79, then it returns a *StreamConnection* implementation.

MIDP simplifies the generic framework by requiring only one type of connection. Hypertext Transfer Protocol (HTTP).

You pass a HTTP URL to Connector and get back an implementation of HttpConnection.

Although MIDP compliant devices may support additional types of connection, HTTP is the only one you should depend on.

MIDP uses a subset of the full HTTP 1.1; only the GET, POST and HEAD commands are required.

Establishing a connection is a very time/resource consuming process that must be dealt with carefully - e.g. use a thread and watch for exceptions.

10.1.1 HTTP

For full details on HTTP the you can read Request for Comments (RFC) 2616, http://www.faqs.org/rfcs/rfc2616.html.

Request and Response

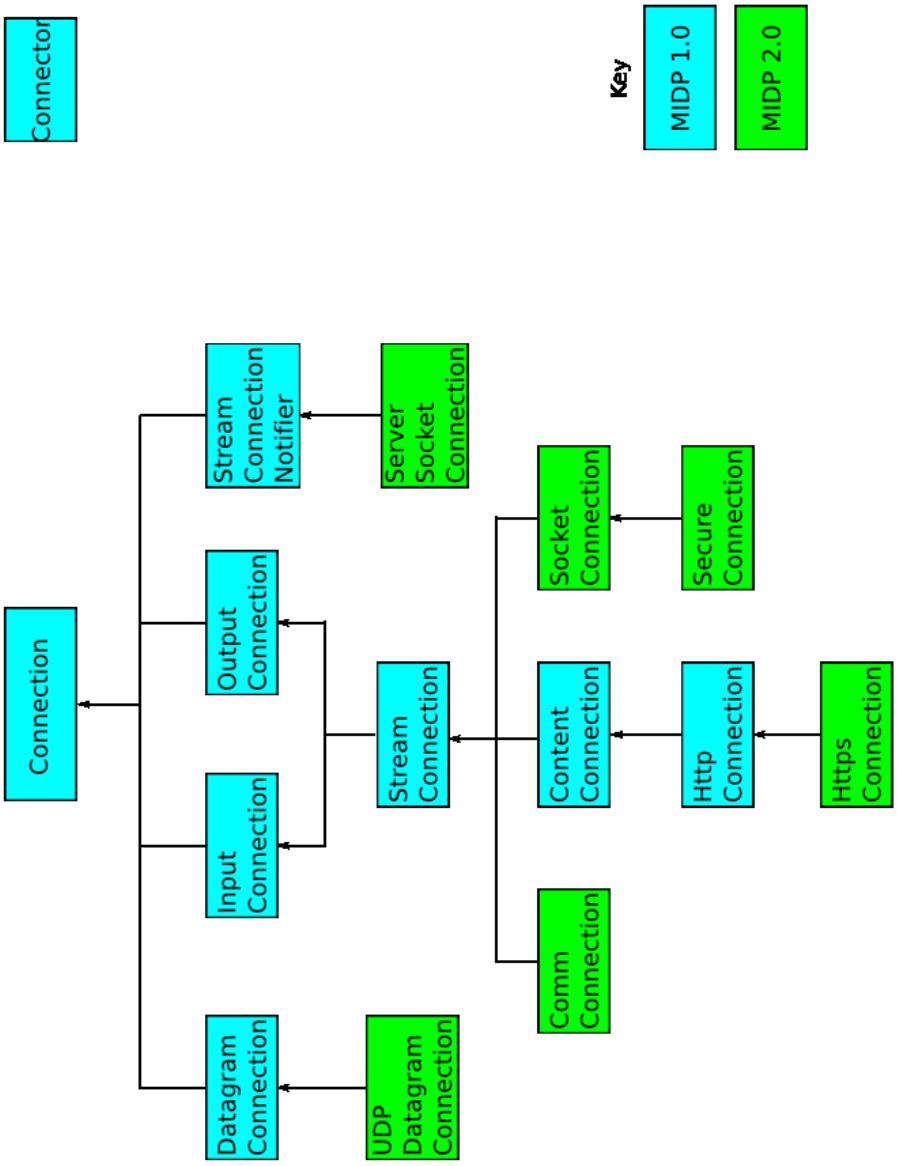

Figure 9: Generic Connection Framework

HTTP is built around Requests and Reponses. A client sends a request to a server: for example "give me index.html". The server sends back a response like "file not found".

Request and Response's have two parts: headers and content. If you type a URL into a browser, the browser creates an HTTP request (mainly headers) and sends it to a server. The response content is the data file itself.

Clients can pass parameters to the server. Parameters are simple name value pairs.

For example a client might send a "userid" parameter with the value "cheese" to a server.

Encoding parameters is quite simple as illustrated below.

- The Space character is converted to a plus (+) sign.
- All alphanumeric characters remain unchanged, as well as the period (.), the hyphen(-), the asterisk (*) and the underscore (_).
- All other characters are converted into "%xy" where "xy" is a hexadecimal number that represents the lower eight bits of the character.

GET Operation

The simplest HTTP operation is GET. This is what happens when you type a URL into your browser.

With a GET request the parameters are added to the end of the URL in encoded form. The following URL may map to a servlet or some other server-side component, http://xyz.com/server.

Adding parameters is very straight forward. To pass a parameter with the name of "user" and the value "john" the following URL would be used.

http://xyz.com/server?user=john.

Additional name and value pairs can be added, by separating each pair with an ampersand (&). http://xyz.com/server?user=john&password=pass.

HEAD & POST Operation

The HEAD operation is identical to GET, except the server only sends back the headers of the response.

POST is mainly the same as GET, but parameters are handled differently.

Instead of being tagged on to the end of the URL, the parameters are passed within the body of the request. They are however encoded using the same format as GET.

10.1.2 Making a Connection

Loading data from a server is very simple, especially if you use HTTP GET. Simply pass a URL to the Connector's static open() method.

The returned Connection will probably be an implementation of HttpConnection, but you can treat it as an InputConnection.

```
String urlString = "http://multimedia.ucc.ie";
InputConnection ic = (InputConnection) Connector.open(urlString);
InputStream in = ic.openInputStream();
//Read data from the input Stream ic.close();
```

Most of the methods can throw java.io.IOException, and so should be surrounded with try and catch blocks.

With HTTP GET, all parameters are passed to the server in the body of the URL. This makes it easy to send parameters to the server. The following example demonstrates the passing of two parameters.

```
String urlString = "http://xyz.com?pOne=one&pTwo=two";
InputConnection ic = (InputConnection) Connector.open(urlString);
InputStream in = ic.openInputStream();
```

The first parameter is named "pOne" and has "one", as a value. The second parameter is named "pTwo" and has "two" as a value.

10.2 Posting a Form with HTTP Post

Posting a form is a little more complicated on the MIDlet side. In particular there are request headers that need to be set in HttpConnection before the server is Contacted.

Connection Process.

- Obtain an HttpConnection from Connector's open() method.
- Modify the header fields of the request. In particular you need to change the request method by calling setRequestMethod(), and you should set the "Content-Length" and "Content-Type" headers by calling setRequestProperty(). This is the length of the parameters you will be sending.
- Obtain the output stream for the HttpConnection by calling openOutputStream(). This sends the request headers to the server.
- Send the request parameters on the output stream returned from the HttpConnection.
- Read the response from the server for the input stream retrieved from HttpConnection's openInputStream() method.

Posting a Form with HTTP Post Steps:

1. Get the URL and create the http connection String url = getAppProperty("PostMIDlet-URL"); hc = (HttpConnection) Connector.open(url);
2. Change the request method hc.setRequestMethod(HttpConnection.POST);
3. Set the headers. hc.setRequestProperty("Content-Type", "application/x-www-form-urlencoded"); hc.setRequestProperty("Content-Length", Integer.toString(message.length()));

10.2.1 The server side computation

The servlet that responds to PostMIDlet does:

1. Receive the value of the parameter.
2. Add up some other information on it.
3. Send it back.

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class PostServlet extends HttpServlet{
  public void doPost(HttpServletRequest request,
    HttpServletResponse response) {
    String name = request.getParamenter("name");
```

```
        String processedName = "Processed Name is "+name;
        response.setContentType("text/plain");
        response.setContentlength( processedName.length());
        PrintWriter out = response.getWriter();
        out.print(processedName);
    }
}
```

10.2.2 Some Golden Rules

1. Work with Get instead of Post to avoid setting up the headers.
2. Keep the URL-s in the app descriptor.
3. Network work should be done in a Thread as it is very computationally expensive.
4. Handle exceptions carefully.
5. Use finally to clean up.

10.3 Assignment

Build an application to get weather information over the internet.

Last year www.bbc.co.uk/weather was used. Ideally find an XML page which will be easier to parse.

Application should have a couple of choice groups: Countries and Towns within countries. Will need to find out which code is used by the site to represent which town and then use that in the URL.

Can do any application you like so long as networking is used.

11 Lecture 16 March 2007

11.1 Mobile 3D Graphics

Overview Mobile 3D API's: There are several API's for Mobile Development:

- OpenGL ES
  - JSR 239: Java Binding for OpenGL ES API
  - Final Specification Release 12 Sept 2006
  - Maintenance Draft Review 17 Nov 2006
- M3G Mobile 3D Graphics (Java) JSR - 184 Mascot Capsule V3 (Proprietary)
  - Popular in Japan
  - Mobile Direct3D and Direct X from Microsoft

JSR 184: Mobile 3D Graphics API for J2ME

Objectives: To provide a scalable, small-footprint, interactive 3D API for use on mobile devices

Specification Lead: Tomi Aarnio, Nokia Corporation

Expert Group: 3d4W Corporation Aplix Corporation ARM Limited Bandai Networks Co., Ltd. BenQ Corporation Cellon France SAS Cingular Wireless France Telecom FueTrek Co., Ltd HI Corporation Hybrid Graphics Ltd. In-Fusio SA Insignia Solutions Intel Corp. Kekalainen, Fredrik MathEngine PLC Motorola Nokia

Corporation Research In Motion, LTD (RIM) Sony Ericsson Mobile Communications AB Sun Microsystems, Inc. Superscape Ltd Symbian Ltd Tammenkoski, Mika Texas Instruments Inc. Weng, Ray

Com2us - Leading Mobile Games Developer

Performance of your mobile:

- JBenchmark3D for (M3G) JBenchmarkHD for HW-accelerated Mobile 3D http://www.jbenchmark.com
- SPMark06 by Futuremark SPMark06 for JAVA 3DMarkMobile06 for OpenGL ES http://www.futuremark.com
- www.unteh.com http://www.unteh.com/products/test184/index.html (URL has a link to a 3D Application which you can download and run to see the 3D capabilities of your system.)

11.2 Asset Creation

Creating the Content for your World

Textures and Backgrounds

Develop Images using PNG File Format using Adobe Photoshop, Paint Shop Pro, Macromedia Fireworks or GIMP

Audio Resources

Develop Audio Resources for the commonly used formats Wave, AU, MP3, MIDI using Sony Sound Forge, ProTools, Garage Band, Sound Studio or Steinberg Cubase (MIDI Sequencer)

3D Studio Max

For a short tutorial on the creation of a Teapot model see:

http://www.mobilefish.com/tutorials/3dsmax/3dsmax_quickguide_teapot.html

For a tutorial on how to load the resulting m3g model into a midlet see:

http://www.mobilefish.com/emulators/j2me_wt/j2me_wt_quickguide_22_ build.html

M3GToolkit - m3g Converter

This application is capable of converting several differing 3d file formats into m3g files suitable for mobile development

http://www.java4ever.com/index.php?section=j2me&project=m3gtoolkit& menu=main&lang=_en (requires Java 1.4 or higher)

11.2.1 3D Modelling

Art of Illusion

A tool is available from http://www.artofillusion.org/

This is a free 3D Modelling package (created in Java).

Imports: .obj Exports: .obj, .pov, VRML

A large number of tutorials and a detailed manual are available on the website.

http://www.artofillusion.org/downloads

Runs on Mac OS/X, Windows, Unix/Linux

The application resides in a zip file (aoi20.zip) once extracted place the application launcher application in the same directory as ArtOfIllusion.jar.

Requires: Java 1.4 or higher.

Java Media Framework is an optional download, which will allow the exporting of animations as a QuickTime movie.

To run the application from the command line use the following command: java -Xmx512m -jar ArtOfIllusion.jar (this will allocate $\frac{1}{2}$ Gig of memory to the application).

Other Software

- http://www.3dcgi.com/learn/free/free-3d.htm Provides links to several other 3D Modelling packages such as Wings 3D and Blender.
- http://www.blender3d.org/cms/Home.2.0.html
- http://www.wings3d.com/

How to Create M3G Objects

M3G Exporter from http://www.m3gexporter.com/

This can export 3D Studio Max (3DS) files into M3G format. Only necessary for versions before 3DS 7.0 when an exporter was integrated into the application. How to Create M3G Objects 3DS Ver 7.

3D Studio Max Version 7 has an inbuilt M3G Exporter. Note: a 30 day trial is available from: http://www4.discreet.com/3dsmax/index.php

Some Limitations Integer precision: This limits the accuracy of small meshes of only a few units in size. For best results, use relatively large meshes. Scaled objects also might exhibit prevision problems. Linear interpolation: Smoothly curved paths defined by only a few keys are not supported.

Texture sizes are limited to "power of two" sizes, to a maximum of 256*256 pixels. No support for shadows.

Another tool: M3G Export http://www.m3gexport.com/index.jsp This tool has been developed to export 3D scenes from Maya.

Two editions are available, Limited and Standard

How to Create M3G Objects

M3G Export - Maya - Overview Platform Compatibility

MS Windows 2000/XP: Maya 5.0, Maya 6.0, Maya 6.5

Linux: Maya 5.0, Maya 6.0, Maya 6.5

Mac OS X: Maya 6.0, Maya 6.5

M3GExport for Maya is available in two editions, Limited and Standard.

The Standard version contains sophisticated optimisation options used mostly by commercial game developers.

The non-commercial Limited edition contains almost every feature found in the full commercial Standard version and can be upgraded later. The main difference is that file sizes of objects will be larger.

Another route.

The H3T file is created by using an export plugin that is available for various 3D Modelling packages (3DS, Maya, Softimage, Lightwave).

Once the 3D Model is exported to the H3T format you can run the M3GConverter program on it to create the M3G File.

Your can then use the M3G Viewer Application to check the m3g file. How to Create M3G Objects.

Alternatively use .obj files

http://fivedots.coe.psu.ac.th/~ad/jg/objm3g/index.html

An alternative solution to displaying 3d objects within your JSR 184 MIDlet is to use a set of applications available from the URL above.

This is a set of tools developed by Dr. Andrew Davis. The first Application is a simple java based viewer for examining .obj files.

Note: To be able to execute this program Java 3D must be installed:

http://java.sun.com/products/java-media/3D/download.html.

The OpenGL version is probably best, which can be downloaded as java3d-1_3_1-windows-i586-opengl-sdk.exe.

The first application to use is called ObjView. It consists of two java files ObjView.java and WrapObjView.java

Compile: javac *.java.

Run: java ObjView hand.obj

Running the Application allow you to view the .obj model, the application also outputs a text file called examobj.txt.

This text file is generated by reading the contents of the .obj file (a simple text file) and converts this into a series of java methods which are outputted to the file examobj.txt.

These methods can then be used to view the object.

11.3 Coordinate Systems

Right Hand Rule

Coordinate systems are used to describe the locations of points in space. In m3g the orthographic coordinate system is used.

In an orthographic coordinate system, the X, Y and Z axis are perpendicular to each other. The right-hand rule is used, with +Y (index finger) being up, +X (thumb) horizontal to the right, and +Z (middle) directed toward the viewer. http://www.mobilefish.com/developer/m3g/m3g_quickguide_matrix.html

Rotations

All angles or rotational representations are in degrees. A positive rotation is counter clockwise.

Three Coordinate Systems

In m3g, three coordinate systems plays an important role:

- World coordinate system. Is the base on which all other coordinates are being defined.
- Camera coordinate system. This coordinate system is aligned with the eye or camera. The +Y axis is up, and the -Z axis points out off the camera. In order to render the world, everything in the scene must be transformed to camera coordinates.
- Local (or object) coordinate system. Each object in the scene is defined in its own local coordinate system. By using a local coordinate system you can create many copies of the object, in different positions and orientations.

11.4 Mobile 3D Graphics

Mobile 3D Graphics API (M3G) Also known as JSR-184

Available with Version 2.2 of the J2ME Wireless Toolkit

Designed for mobile devices Primarily CLDC/MIDP And also CDC

Immediate & Retained Mode

M3G provides two ways for developers to draw 3D graphics: immediate mode and retained mode.

Immediate mode: Graphics commands are issued directly into the graphics pipeline and the rendering engine executes them immediately. When using this method, the developer must write code that specifically tells the rendering engine what to draw for each animation frame.

Retained mode: Uses a scene graph that links all geometric objects in the 3D world in a tree structure. Higher-level information about each object such as its geometric structure, position, and appearance is retained from frame to frame.

11.4.1 Overview of the M3G Classes

AnimationController Controls the animation sequence.

AnimationTrack Associates a KeyframeSequence with an AnimationController.

Appearance A set of objects that defines the rendering attributes of a Mesh or a Spring3D.

Background Defines how the viewport is cleared.

Camera A scene graph node that defines the position of the viewer in the scene and the projection from 3D to 2D.

CompositingMode An Appearance class that encapsulates per-pixel compositing attributes.

Fog An Appearance class that contains attributes for fogging.

Graphics3D A singleton 3D graphics context. All rendering is done through the render() methods in this class.

Group A scene graph node that stores an unordered set of nodes as its children.

Image2D A two-dimensional image that can be used as a texture, background, or sprite image.

IndexBuffer The class defines how to connect vertices to form a geometric object.

KeyframeSequence Encapsulates animation data as a sequence of time- stamped, vector-valued keyframes.

Light Represents different kinds of light sources. Loader Downloads and deserializes graph nodes and node components, as well as entire scene graphs.

Material Encapsulates material attributes for lighting computations.

Mesh Represents a 3D object defined as a polygonal surface.

MorphingMesh Represents a vertex-morphing polygon mesh.

Node An abstract class for all scene graph nodes. The five kinds are Camera, Mesh, Sprite3D, Light, and Group.

Object3D An abstract base class for all objects that can be part of a 3D world.

PolygonMode Encapsulating polygon-level attributes.

RayIntersection Stores a reference to an intersected Mesh or Sprite3D and information about the intersection point.

SkinnedMesh Represents a skeletally animated polygon mesh.

Sprite3D Represents a 2D image with a 3D position.

Texture2D Encapsulates a 2D texture image and a set of attributes specifying how the image is to be applied on submeshes.

Transform A generic 4x4 floating-point matrix representing a transformation.

Transformable An abstract base class for Node and Texture2D.

TriangleStripArray Defines an array of triangle strips.

VertexArray An array of integer vectors representing vertex positions, normals, colors, or texture coordinates.

VertexBuffer Holds references to VertexArrays that contain the positions, colors, normals, and texture coordinates for a set of vertices.

World A special group node that is a top-level container for scene graphs.

11.4.2 M3G Immediate Mode

The camera's position and properties define what will appear on the users screen.

The Viewing Angle, this is comparable to that of a camera lens Telephoto lens has narrower view than a wide-angle lens The Viewing Angle determines what you see to the sides

There are also two further Boundaries: The Near and Far Clipping Planes The View Frustum Viewing Angle + Near Clipping Plane + Far Clipping Plane = View Frustum

Creating A 3D Cube Cube

Vertex Positions:

The Cube will have its origin at 0,0,0 The eight Vertex positions ([ ] VERTEX_POSITIONS) vary from 1,1,1 to -1,-1,-1 The vertex positions are insufficient on their own to create a geometry To build geometry with M3G one must use Triangles To define the Cube with Triangles one would need 6 sides x 2 triangles x 3 vertices = 36 vertices This would be a waste of memory, due to duplication of vertex locations,

The VertexBuffer object allows for the assignment of several types of information to a vertex. Colour is one example. If you assign colours to the vertices, what happens to the pixels inside the triangle? In flat-shading only one vertex colour is used for the whole triangle The colour of the triangle's third vertex is used for this shading method In smooth-shading mode, each pixel inside the triangle gets its own colour via interpolation. Therefore the result is a gradient

The Winding of a Polygon: The order in which indices are defined is crucial when working with colour Each triangle has a front and a back face By default, counter-clockwise ordering indicates the front; a strips first triangle defines its winding.

You should if possible disable faces that cannot be seen, this results in the speed up of the rendering process. The process of excluding triangles from rendering is called culling.

Transformations

M3G provides three interfaces to achieve Transformations

- Transform.postScale(float sx, float sy, float sz) Scales the 3D object in the x, y, and z direction. Values greater than 1 enlarge the object by the given factor; values between 0 and 1 reduce it. Negative values will scale and mirror at the same time.

- Transform.postTranslate(float tx, float ty, float tz) Moves the 3D object by adding the given values to the x, y, and z coordinates. Negative values move the object in the direction of the negative axis.

- Transform.postRotate(float angle, float ax, float ay, float az) Rotates the object with the given angle around the axis that goes through (0, 0, 0) and (ax, ay, az). Positive values for the angle rotate the object clockwise if you look along the positive rotation axis. For example, postRotate(30, 1, 0, 0) rotates an object by 30 degrees around the x axis.

All operation names start with "post," which means that the current Transform object is multiplied from the right with the given transformation matrix. The order of matrix operations matters. Turn 90 degrees to the right and walk two steps differs from walk two steps then turn postRotate() and postTranslate() achieves the latter transformation. Due to post-multiplication, the transformation you use last is applied first.

M3G has a Transform class and a Transformable interface. All immediate mode APIs take a Transform object as a parameter that modifies its associated 3D object.

The Transformable interface is used to transform nodes that are part of a 3D world in that uses the retained mode.

Depth Buffer and Projection

Projection, defines how 3D objects map to a 2D screen.

Depth Buffer, is a method to render objects correctly according to their distance from the camera

To see the rendered image from the camera's point of view, you have to convert the 3D world to *camera space* by taking the camera position and orientation into account.

Camera.setPerspective() told M3G to do a perspective projection when converting from a 3D to 2D space.

Perspective projection works like the real world. When you look down a long, straight road, it seems that the road's boundaries meet at a point at the horizon. Objects along the road become smaller the further the distance to the camera. You can also ignore perspective and draw all objects the same size, no matter how far away they are. This can make sense for applications such as CAD programs, where it is easier to work on drawings without perspective. To disable perspective projection, call Camera.setParallel() rather than Camera.setPerspective().

Depth Buffer:

In camera space, an object's z coordinate specifies its distance to the camera. If you render several 3D objects with different z coordinates, you would expect objects closer to the camera to obscure objects further away. The depth buffer has the same width and height as the screen but holds z coordinates instead of color values. It stores the distances to the camera of all the pixels drawn on the screen. However, M3G draws a pixel only if the pixel is closer to the camera than the existing one at the same position.

Lighting

One can see nothing without Light However Vertex Colours & Textures do not need light, they will always display in their default colorations. Light however, can modify the appearance of such objects, and add depth to a scene. The direction of light reflected from an object depends on its alignment with the object. The main principle is that a direction vector that is orthogonal to the lit surface is required. This vector is called the Normal Vector or Normal Lighting A Normal is a per-vertex attribute. The shading of pixels between vertices is either interpolated (PolygonMode.SHADE_SMOOTH); Or taken from a triangles third vertex (PolygonMode.SHADE_FLAT).

Omnidirectional light originates in one point and shines equally in all directions. A light bulb without a lampshade produces such a light.

Directional light emits parallel rays in one direction. The sun is so far away that you can consider its rays parallel. A directional light doesn't have a position, only a direction.

Spotlight is comparable to a flashlight or a spot used in theatre. Its light casts a cone shape and illuminates objects where the cone meets with the surface.

In the real world, light also bounces off objects and illuminates others nearby. Raytracing is one mechanism to achieve realistic images by following the path from the camera to the light source. The drawback however is very long computation times.

An alternative is Ambient Light, this illuminates objects from every direction at a constant rate.

The omnidirectional light is strongest at the vertex facing the light and slowly fades out. The spotlight, on the other hand, produces a sharp change between light

and dark, where the cone of the spot ends. If the spot had been defined with a larger cone, the result would have been similar to the omnidirectional light. The ambient light illuminates the cube from every direction. The cube looks flat because the image lacks shade. Finally, the directional light gives each side a different color. Within a side, the colour stays the same because of the light's parallel rays.

Materials

A light can cause different effects. A shiny silver ball reflects the light in a different way than a sheet of paper does. M3G models these material characteristics with the following attributes: Ambient reflection: The light that is reflected by an ambient light source.

Diffuse reflection: The reflected light is scattered equally in all directions.

Emissive light: An object can send out light to imitate glowing objects.

Specular reflection: The light that reflects off objects with a shiny surface. You can set a color for each material attribute. The shiny silver ball's diffused color would be silver and its specular component white. To get the final object color, the material's color mixes with the light's color. If you point a blue light toward the silver ball, it would turn bluish.

setMaterial() creates a new Material object and sets the colors with setColor() using the respective color component identifier.

The Material object is then assigned to an Appearance object, which is used in the call to Graphics3D.render().

One can use Material.setVertexColorTrackingEnable() in order to use the vertex colors for ambient and diffuse reflection instead of using Material.setColor().

Textures

Applying vertex colours and materials can help the appearance on an object.

Textures are images that are wrapped around 3D objects.

This requires another per-vertex attribute.

For each vertex, a texture coordinate defines which texture position will be used M3G will them map the texture to fit the object The texture's pixels are now referred to as texels.

Texture coordinates are named (s, t) in order to distinguish them from the (x, y) used for vertex positions (in literature, (u, v) is commonly used instead). Coordinates (s, t) are defined so (0, 0) is the texture's upper left corner and (1, 1) is its lower right corner. Accordingly, if you want to map the lower left corner (-1, -1) of the cube's front face to the texture's lower left corner, you must assign the texture coordinates (0, 1) to vertex 0.

A texture's width and height can be different, but must be powers of 2, such as 128 x 128.

Implementations must support texture sizes at least up to 256, which is one of the optional properties in M3G.

Graphics3D.getProperties() returns a Hashtable filled with implementation-specific properties, such as the maximum texture dimension or the maximum supported number of lights.

The documentation to getProperties() contains a list of properties and their minimum requirements. Before using a feature that exceeds that value, you should first check whether your device's implementation supports it.

When using texturing, you might still want to use colors obtained by lighting or directly assigned to vertices. For this purpose, M3G supports different blending functions that are set with a call to _cubeTexture.setBlending(). In init(), Texture2D.FUNC_DECAL is used, which blends the texture with the underlying vertex color depending on an alpha value. The gray bits in the texture image (seen previously) are 60 percent transparent. Instead of setting the vertex colors or using lighting, a default color for the cube with _cubeVertexData.setDefaultColor(),

which means that all triangles of the cube will have the same color. With blending, you can also use multiple textures on top of each other to achieve additional effects.

As seen in the lighting section, the rendering quality depends on the number of triangles you use The smaller the distance between vertices, the better the interpolation. The same is true for textures. High quality in context to textures means that the texture maps without distortion. Textures like the one seen earlier are vulnerable because they contain straight lines, which make a distortion obvious. M3G provides a cheap way, in terms of processing power, to solve the problem. An optional perspective correction flag can be set with PolygonMode.setPerspectiveCorrectionEnable()

11.4.3 Retained Mode

The Mobile 3D Graphics API retained mode lets you work with a scene graph representation of your 3D world. In retained mode, you define and display an entire world of 3D objects, including information on their appearance. Imagine retained mode as a more abstract, but also more comfortable, way of displaying 3D graphics. Retained mode is especially handy when you create your 3D scene in a modeling tool and import the data into your application.

Method Call Differences

Camera

- Immediate Set the current camera in Graphics3D before render call. Graphics3D.setCamera(Camera camera, Transform transform)
- Retained Add one or more cameras to the world as children and activate one of them. World.addChild(Node child) World.setActiveCamera(Camera camera)

Transform

- Immediate Pass Transform object in API call, for example: Graphics3D.render(Node node, Transform transform)
- Retained Use methods inherited from Transformable class, for example: Node. setTransform(Transform transform)

Light

- Immediate Set lights in Graphics3D object before render call. Graphics3D. addLight(Light light, Transform transform) Graphics3D.setLight(int index, Light light, Transform transform) Graphics3D.resetLights()
- Retained Add lights to the world as children. World.addChild(Node child)

Background

- Immediate Set background with Graphics3D.clear(). Always call clear() before rendering. Graphics3D.clear(Background background)
- Retained Set background in World object. If not set, the background is cleared to black automatically. World.setBackground(Background background)

Rendering

- Immediate Render scene graph nodes, (Sprite3D, Mesh, Group, and their subclasses), and submeshes (VertexBuffer). Graphics3D.render(Node node, Transform transform) Graphics3D.render(VertexBuffer vertices, IndexBuffer triangles, Appearance appearance, Transform transform, int scope) Graphics3D.render(VertexBuffer vertices, IndexBuffer triangles, Appearance appearance, Transform transform)

- Retained Render complete world including its children. Graphics.render(World world)

11.4.4 Combining Both Modes

You can combine retained mode and immediate mode by mixing the respective render commands.

After using Graphics3D.render(World), the active camera and lights in the rendered world automatically replace Graphics3D's current camera and lights.

This way, subsequent immediate mode rendering can easily use the same environment.

You can also render a world in immediate mode with Graphics3D.render(Node, Transform) because World inherits from Node.

In this case, the world's lights, cameras, and the background are ignored, but all other child nodes are rendered.

M3G's specification defines retained mode based on the type of render() call that you use for drawing.

The API operates in retained mode when you use render(World).

Because you can render a World object and its children as Node in immediate mode as well, there isn't much difference between either mode.

However, what is commonly referred to as retained mode is the ability to handle a group of 3D objects in a data structure called a scene graph, independent from the render code.

Scene Graph

The advantages of a scene graph become apparent when you define a more complex world.

In M3G, the class World organizes 3D objects in a tree structure whose nodes are classes derived from Node.

You've already seen two subclasses of Node in the previous example: Camera and Mesh.

Other commonly used objects include Light, which illuminates a 3D scene, and Group, which acts as a container for other nodes.

Grouped nodes can be treated as if they were a single object.

Scene Graph Transformations

Transformations work slightly differently in retained mode than in immediate mode.

In immediate mode, Transform represents a single matrix that manipulates a 3D object.

In retained mode, methods of the Node object inherited from Transformable are used instead.

Transformable has four transformation components: translation (T), orientation (R), scale (S), and a generic 4x4 matrix (M).

All four components can be set at the same time and together build the final transformation.

A vector p, such as a vertex coordinate, is transformed into p' by using the equation p'=TRSMp.

The easiest way to translate a transformation from immediate mode to retained mode is to use Transformable.setTransform() and thus change the generic matrix.

Otherwise, you can use the remaining three components, but note that the order of the matrix multiplication is fixed and can't be changed. Transformations also apply to Group nodes.

M3G's Binary Format

Creating 3D objects manually is tedious.

To create any substantial scene a 3D modeling tool is necessary to create the 3D world.

This can then be imported it into an M3G application.

An M3G application can load a m3g file by using the Loader.load() method.

Several tools are available for model creation, and the export to the m3g format.

11.5 Blender

Creating 3D Text Start with a new file (Ctrl-X) and remove the cube that Blender creates by default.

Position the cursor at the origin and snap the cursor to the grid in front view and side view (Shift-S-3). The cursor is now exactly at the origin.

In front view, add text (Add > Text), which creates an editable font object. Type in any text you want; I used "Hello, world!".

Switch to object mode (Tab) and display the Editing context (F9). Click the Center New button in the Curve and Surface panel. The text is now centered horizontally around the origin.

In the same panel, enter 0.100 in the Ext1 field, which extrudes the depth of the text and gives it a 3D look.

In top view, add a Bezier circle (Add > Curve > Bezier Circle) that's orthogonal to the text. I'll use the circle later to bend the text so it follows a circular path.

The circle is selected by default; scale it to make it bigger (type S for scaling).

Because the exporter script can only handle meshes, it is necessary to convert the font object. Afterwards, the resulting mesh will be bent around the circle.

In object mode (switch with Tab), select the text with a right click and Alt-C converts the object type. First from font to curve, then a second conversion from curve to mesh.

The text still selected, go to the Editing context (F9) and create a new material with the default values in the Links and Materials panel with the New button. A mesh must have a material; otherwise, the exporter script will report an error.

Select the text, shift select the circle, and create a parent-child relationship with Ctrl-P. A list of options opens, choose curve deform.

With only the text selected, jump to the Object context (F7). Select Track X in the Anim settings panel. This selects the x axis for alignment between the circle and the text. The text bends nicely in a circular shape now.

The text will now stay deformed by the circle. Transform the text to your liking (type G for translation and R for rotation).

Use Ctrl-Shift-A on the text to apply the deformation on the mesh and delete the circle because it's not needed any more. If the text looks strangely deformed after the circle is deleted, just press Tab twice to switch between object and edit mode.

The background color can be changed in the World panel (switch to Shading context with F5 and select World buttons).

In the Links and Materials panel of the Editing context (F9), rename the text's object name to Font#01. The number after the hash sign will be the user ID of the mesh.

The final step is to export the 3D model into an M3G file.

Install the exporter script by copying it into the .blender\scripts folder of your Blender installation.

The script uses some imports that Blender's default Python installation doesn't have. You can solve this by installing a separate Python interpreter. Instead, I just commented out the offending import statements and the script worked fine.

After restarting Blender, load your 3D model and run the script by selecting File > Export > M3G in J2ME.

Select Export into *.m3g binary file. The script can also produce the Java language source equivalent of your 3D model. This might be handy if you want to make manual changes. For my purpose, the binary file is a good choice.

Select the name and directory of the M3G file that you want to store and click the button Save M3G J2ME.

The big advantage of files exported from 3D tools is that you can split the work between developers and artists.

When you create a more complex model in Blender, you'll soon realize that this needs a complementary skill set to software development.

In commercial games for consoles and PC, the majority of manpower and money is spent on the artwork.

For mobile phones, the model does not only have to look good, but also has to be small.

Alignment & Picking

When you tee off in a golf game, you'd want the camera to always show the ball in the middle of the screen on its way to the ground. With Node.setAlignment(), a node, such as the camera, can be automatically oriented toward a reference, such as the golf ball. In the next example, we this to transform the text so it's directly facing the camera. Another common problem in games is selecting an object, which is called picking. When you design a first person shooter and fire a bullet, you have to check whether it hits its target. Group.pick() casts a ray that you supply with an origin and a direction. If the ray intersects with a mesh, the method returns true and reports the struck mesh object.

After loading the M3G file, the call to setAlignment() in init() sets the alignment information for the text mesh The first pair of parameters defines the z axis alignment The second pair defines that of the y axis Orient the text's z axis to the camera's y axis and leave the text's y axis unchanged The choice of axes takes the difference in Blender's and M3G's coordinate system into account The actual transformation on the text mesh is performed with a call to align() If you wanted to continually track an object, you'd call align() each time the reference moves init() also creates two images, which I use to draw a crosshair in paint().

Depending on the boolean value _crossHairOn, the crosshair is black (_crossHairImageOff) or white (_crossHairImageOn). You can easily mix calls to Graphics and Graphics3D. Just make sure that calls to Graphics3D are done between bindTarget() and releaseTarget() and calls to Graphics outside. The state of the crosshair is determined in keyPressed().

keyPressed() rotates the text either to the left or to the right when the corresponding key is pressed At the same time, isHit() checks whether the crosshair is aiming at the text. The method uses Group.pick(), which comes in two variants – one that takes an arbitrary ray origin and destination and one that defines the ray in viewport coordinates

The viewport is the plane onto which M3G projects the graphics Its origin is in the upper-left corner and the coordinates range from 0 to 1 As we are interested in picking a mesh based on the current camera, the latter variant is most convenient The coordinates, (0.5, 0.5), denote the middle of the screen because the viewport is the same size as the screen by default If an intersection is reported, the value of _crossHairOn is changed and the respective crosshair image is shown

Picking & Alignment a) Crosshair doesn't intersect with text b) Crosshair intersects

11.6 References

Wikipedia M3G, http://en.wikipedia.org/wiki/M3G

Wikipedia Scene Graph, http://en.wikipedia.org/wiki/Scene_graph

JSR 184, http://www.jcp.org/en/jsr/detail?id=184
Sony Ericsson Mobile 3D Wiki,
http://developer.sonyericsson.com/wiki/display/leftnav/Mobile+3D
Com2us - Leading Mobile Game Developer, http://www.com2us.com/ENGGlobal/
EDA075, Mobile Computer Graphics, Lund University, Sweden,
http://www.cs.lth.se/EDA075/
Khronos Group, http://www.khronos.org/opengles/
Developer case study: Com2uS tee-off with 10 facts for Mobile Java 3D game developers, http://developer.sonyericsson.com/site/global/newsandevents/
casestudies/a99de91f-e4b9-4c72-b5b0-f423c379cedd.jsp
Futuremark, press release http://www.futuremark.com/companyinfo/pressroom/pressreleases/50446/
JBenchmark - Java 3D Benchmark, http://www.jbenchmark.com Futuremark - Mobile Benchmarking, http://futuremark.com
JSR 239 - JavaTM Binding for the OpenGL® ES API,
http://jcp.org/en/jsr/detail?id=239
3D Graphics for Java Mobile Devices, Part 1 M3G's immediate mode, Claus Hofele, Oct 2005, http://www-128.ibm.com/developerworks/java/library/wi-mobile1/
3D Graphics for Java Mobile Devices, Part 2 M3G's retained mode, Claus Hofele, Dec 2005, http://www-128.ibm.com/developerworks/java/library/wi-mobile2/
Chapter 11 - 3D Graphics Fundamentals,
http://www.courseptr.com/downloads/chapterpreview/005853_11.pdf
M3GToolkit, http://www.java4ever.com/index.php?section=j2me&
project=m3gtoolkit&menu=main&lang=_en
Blender, http://blender.org
M3G Export, http://www.m3gexport.com/index.jsp
Art of Illusion, http://www.artofillusion.org/downloads

12 Lecture 23 March 2007

PDAs

13 Lecture 30 March 2007

13.1 Exam

Structure: Two questions, answer both.

Know the rough order of arguments of standard methods.

Know the structure of applications for particular problem (e.g. the sort of items that can be put on a form - StringItem, Spacer, etc.).

Not coming up: Turtle Graphics, 3D and PDA.

Tips:

- Focus on points that carry lots of marks first.
- When asked to "briefly" describe something, one sentence is sufficient.
- When "briefly" is not mentioned then a bigger answer is needed.
- Note what is being asked about: When a Method then give method only; When a MIDlet then give the elements; If asked to include Commands then do so, if not then include an exit command anyway.