MOBILE GAMES

Developing sprite based 2D games

NUS National University of Singapore

Objectives

- Sprite based 2D games and its types
- Game framework, Sprites, Event handling mechanisms
- Sprite animation
- collision detection
- Layers: Tiled layers, Layer Management
- 2D SVG (new)

Source: Gamesoft.com

NUS National University of Singapore

J2ME Game API (JSR 118) www.jcp.org

- Available from MIDP 2.0 (JSR 118, Lead by Motorola)
- Package
  - javax.microedition.lcdui.game.*
- Contains five classes
  - GameCanvas
  - Layer
  - Sprite
  - TiledLayer
  - Layer Manager

NUS National University of Singapore

GameCanvas & Event Handling

- The *GameCanvas* class provides the basis for a game user interface.
- It is a Displayable object.
- Provides **graphics object** for drawing and supports double buffering.
- Provides facility to **repaint selected region** in the entire Canvas.
- Provides methods for **key polling**.
- Provides methods to flush graphics buffer to screen.

NUS National University of Singapore

Game Canvas

Constructor:

protected ***GameCanvas****(boolean suppressKeyEvents)*

- Creates a new instance of a *GameCanvas*. A new buffer is also created for the *GameCanvas* and is initially filled with white pixels.
- *suppressKeyEvents* - *true* to suppress the regular key event mechanism (key event callback method) for game keys, otherwise *false.*

Q: What is key-event call back methods?

NUS National University of Singapore

Key Polling

- For example, the following code grabs the key states and checks whether the right game key is pressed.

```
int keyStates = getKeyStates();
if ((keyStates & RIGHT_PRESSED) != 0)
    // do something
```

- This is attractive for game because it gives your application more control. Instead of waiting for the system to invoke the key callback methods in the Canvas, we can immediately find out the state of the device keys.
- Multiple-key events can be easily handled

Q: Compare key interrupt and key polling.

NUS National University of Singapore

Structure of a typical Game-Loop

» The following code snippet illustrates the structure of a typical game loop:

```
// Get the Graphics object for the off-screen
bufferGraphics g = getGraphics();

while (true) {     //GAME-LOOP
      // Check user input and update positions if necessary
      int keyState = getKeyStates();
      if ((keyState & LEFT_PRESSED) != 0) {
            sprite.move(-1, 0);       }
      else if ((keyState & RIGHT_PRESSED) != 0) {
           sprite.move(1, 0);       }

      g.setColor(0xFFFFFF);  // Clear the background to white
      g.fillRect(0,0,getWidth(), getHeight());
      sprite.paint(g);     // Draw the Sprite
      flushGraphics();  // Flush the off-screen buffer

      try { Thread.sleep(1); }     //Sleep Thread for atleast 1 ms
      catch (InterruptedException ie) {}   //to give room for other events
}

//Created by BHOJAN ANAND
```

NUS National University of Singapore

GameCanvas Methods

Method	Purpose
flushGraphics()	Flushes the off-screen buffer to the display.
flushGraphics(int x, int y, int width, int height)	Flushes the specified region of the off-screen buffer to the display.
getGraphics()	Returns the Graphics object for rendering a *GameCanvas*.
getKeyStates()	Gets the states of the physical game keys.
paint(Graphics g)	Paints this *GameCanvas*.

Also supports the inherited methods from Canvas class.
Eg. **setFullScreenMode(boolean true/false),**
repaint(), repaint(x,y,w,h)…

NUS National University of Singapore

Handling Images

- PNG (Portable Network Graphics) is the only image type supported by MIDP 1.0 and MIDP 2.0 specifications. Implementations can support other image types (.gif, .jpg..) -- http://www.w3.org/TR/PNG/
- Images can be Mutable or Immutable
- Creating a mutable image
  - *Image img = Image.createImage(int width(), int height());*
    - Creates an empty mutable image with white pixels for off-screen drawing.
    - Use draw methods of the ***Graphics*** class to draw the image.
- Getting graphics object for off-screen drawing
  - *Graphics g = img.getGraphics();*

NUS National University of Singapore

Handling Images

- Creating a immutable image from a image file (PNG). IMPORTING IMAGE
  - *Image img = Image.createImage(image source);*
    - Source : relative path to the source file.
- Converting "immutable image >> mutable image"
  - *g.drawImage(img);*
    - Where g is a Graphics object.
    - The *drawImage()* method of the Graphics class allows you to display interactive, editable images called *mutable* images on a the low-level Canvas screen. Mutable images can be modified by all the methods provided by the Graphics class.
- MIDP 2.0 supports alpha processing (transparency) for immutable images.
- A fully transparent pixel in the source data will result in a fully transparent pixel in the new image.

Further reading: MIDP 2.0 API reference

NUS National University of Singapore

Sprites

- A *Sprite* is a basic visual element that can be rendered with one of several frames stored in an Image.
- Different frames can be shown to animate the Sprite.
- Several transforms such as flipping and rotation can also be applied to a *Sprite* to further vary its appearance.
- A Sprite's location can be changed and it can also be made visible or invisible.

Creating a Sprite (without animation]

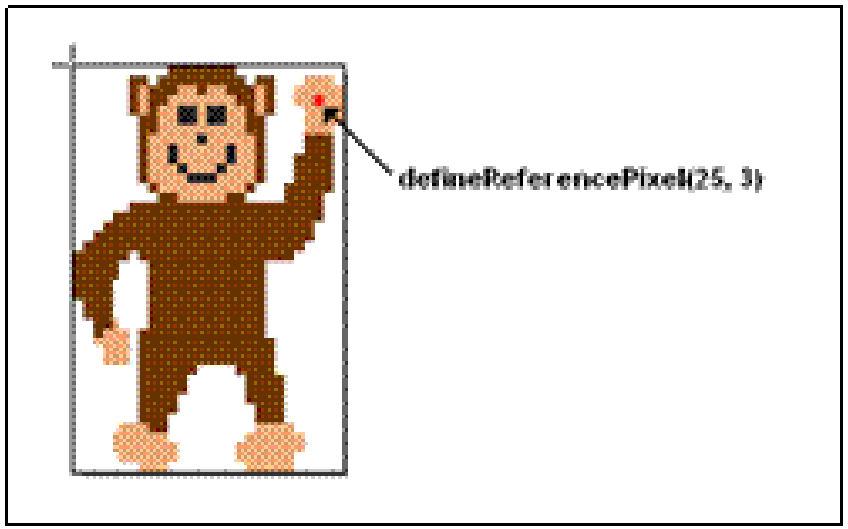
```
Image flightImage = null;
try{
    flightImage = Image.createImage("/mustang.png");
}catch(IOException e) {}
    Sprite mFlight = new Sprite(flightImage, 36, 18);
```

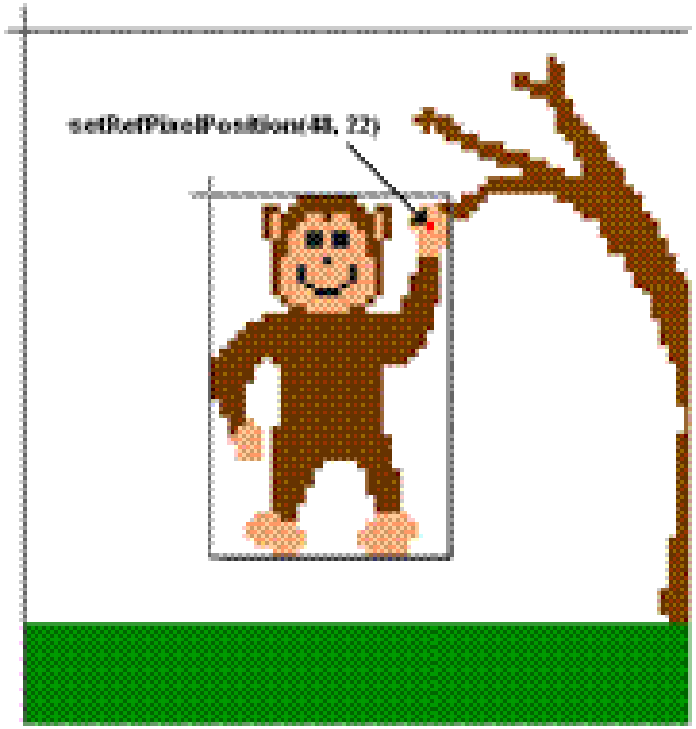
mustang.png

A sprite can be created from another sprite : ***Sprite****(Sprite s)*

NUS National University of Singapore

Reference Pixel of a Sprite

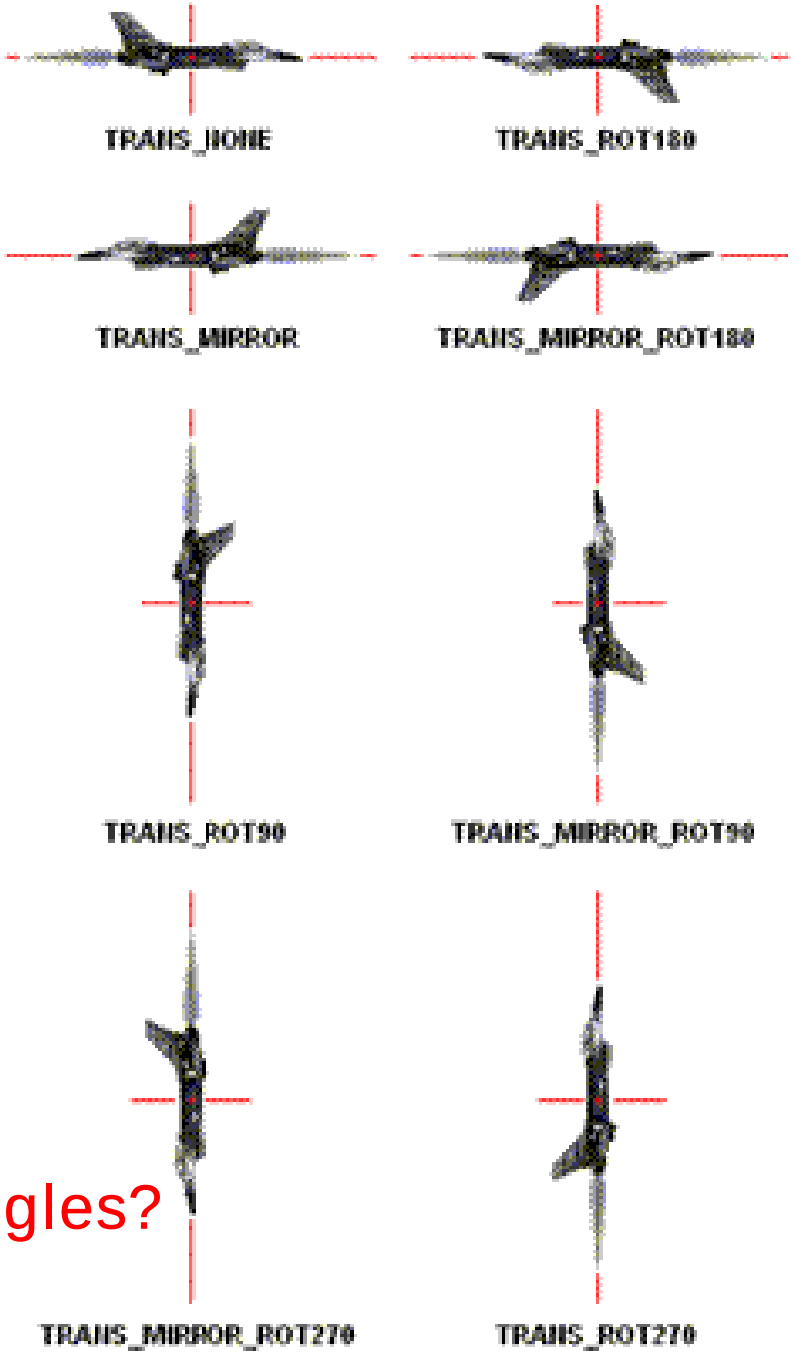




defineReferencePixel and setReferencePixel

Sprite Transformations

- Various transforms can be applied to a *Sprite*.
- The available transforms include rotations in multiples of 90 degrees, and mirrored (about the vertical axis) versions of each of the rotations.
- A Sprite's transform is set by calling *setTransform(transform)*.



Q: How about rotating in other angles?

NUS National University of Singapore

Collision Handling

- Game API supports two techniques
  - Collision of Bounding Rectangles
  - Pixel level collision detection
- Bounding rectangle / Collision rectangle
  - Sprite has a *collision rectangle*. It is defined by the coordinate system of the *Sprite* itself, like the reference pixel.
  - *By default the collision rectangle is located at (0,0) with the same height and width as the Sprite.*
  - We can change the collision rectangle using the following method.
    - *Public void defineCollisionRectangle(int x, int y, int width, int height);*

NUS National University of Singapore

Collision Handling

– Sprite's collision with other Sprites, TiledLayers, and Images

Method	**pixelLevel = *false***	**pixelLevel = *ture***
public final boolean ***collidesWith(Sprite*** *s, boolean pixelLevel)*	Compares collision rectangles	Compares pixels inside the collision rectangles
public final boolean ***collidesWith(TiledLayer*** *t, boolean pixelLevel)*	Compares the Sprite's collision rectangle and tiles in the TiledLayer	Compares pixels inside the Sprite's collision rectngle with pixels in the TiledLayer
public final boolean ***collidesWith(Image*** *image, int x, int y, boolean pixelLevel)*	Compares the Sprite's collision rectangle and the Image's bounds	Compares pixels inside the Sprite's collision rectangle with the pixels in the image.

NUS National University of Singapore

FYORP

Basic Sprite Methods

- **collidesWith**(Image image, int x, int y, boolean pixelLevel)
  - Checks for a collision between this Sprite and the specified Image with its upper left corner at the specified location.
- **collidesWith**(Sprite s, boolean pixelLevel)
  - Checks for a collision between this Sprite and the specified Sprite.
- **collidesWith**(TiledLayer t, boolean pixelLevel)
  - Checks for a collision between this Sprite and the specified TiledLayer.
- **defineReferencePixel**(int x, int y)
  - Defines the reference pixel for this Sprite.
- **setRefPixelPosition**(int x, int y)
  - Sets this Sprite's position such that its reference pixel is located at (x,y) in the painter's coordinate system.

NUS National University of Singapore

FYORP

Basic Sprite Methods (...)

- **getRefPixelX**()
  - Gets the horizontal position of this Sprite's reference pixel in the painter's coordinate system.
- **getRefPixelY**()
  - Gets the vertical position of this Sprite's reference pixel in the painter's coordinate system.
- **paint**(Graphics g)
  - Draws the *Sprite*.
- **setImage**(Image img, int frameWidth, int frameHeight)
  - Changes the Image containing the Sprite's frames.
- **setTransform**(int transform)
  - Sets the transform for this *Sprite*.

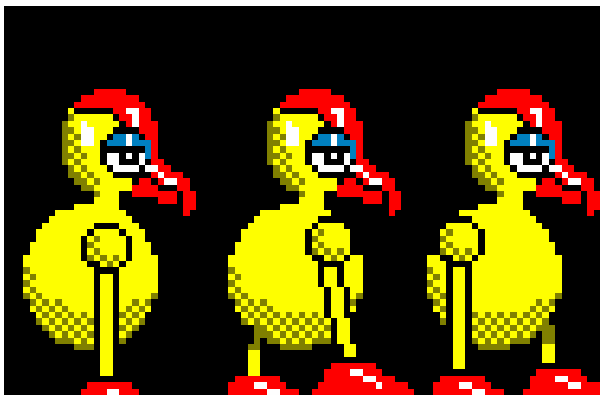
Refer to MIDP 2.0 API doc for full list

NUS National University of Singapore

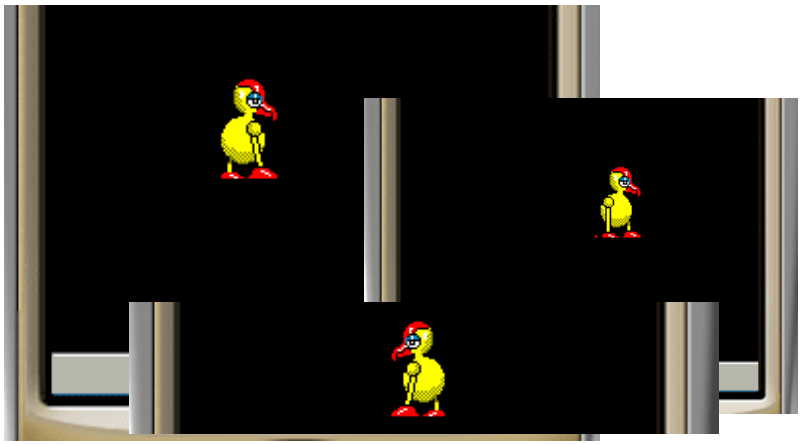
Animated Sprite

```
try{
  spriteImage =
    Image.createImage("/midp2proj/bird.png");
}catch(IOException e) {}
mSprite = new Sprite(spriteImage, 31, 61);
mSprite.setPosition(getWidth()/2,getHeight()/2);
mSprite.defineReferencePixel(15, 30);
```

- parameters 31,61 represents the frame width and height



bird.png with 3 frames



NUS National University of Singapore

Frame Control

- The developer must manually switch the current frame in the frame sequence. This may be accomplished by calling ***setFrame(int)***, ***prevFrame()***, or ***nextFrame()***.

- Other Methods
  - **getFrame**()
    - Gets the current index in the frame sequence.
  - **getFrameSequenceLength**()
    - Gets the number of elements in the frame sequence.
  - **getRawFrameCount**()
    - Gets the number of raw frames for this *Sprite*.
  - **setFrameSequence**(int[] sequence)
    - Set the frame sequence for this Sprite. Each integer represents the frame number, which starts from 0.

NUS National University of Singapore

Layer

- Using layers, one can logically separate the distinct components in the game and the order in which they are drawn. A game can have as many layers as required.
- GAME API defines an abstract Layer class, which serves as a base for two types of Layers (Sprites and TiledLayers)
- Each Layer has position (in terms of the upper-left corner of its visual bounds), width, height, and can be made visible or invisible.
- Layer's (x,y) position is always interpreted as ***painter's coordinate system***.

NUS National University of Singapore

FYORP

Essential Layer Methods

Method	Purpose
getHeight()	Gets the current height of this layer, in pixels.
getWidth()	Gets the current width of this layer, in pixels.
getX()	Gets the horizontal position of this Layer's upper-left corner in the painter's coordinate system.
getY()	Gets the vertical position of this Layer's upper-left corner in the painter's coordinate system.
isVisible()	Gets the visibility of this *Layer*.
move(int dx, int dy)	Moves this Layer by the specified horizontal and vertical distances.
paint(Graphics g)	Paints this *Layer* if it is visible.
setPosition(int x, int y)	Sets this Layer's position such that its upper-left corner is located at (x,y) in the painter's coordinate system.
setVisible(boolean visible)	Sets the visibility of this *Layer*.

NUS National University of Singapore

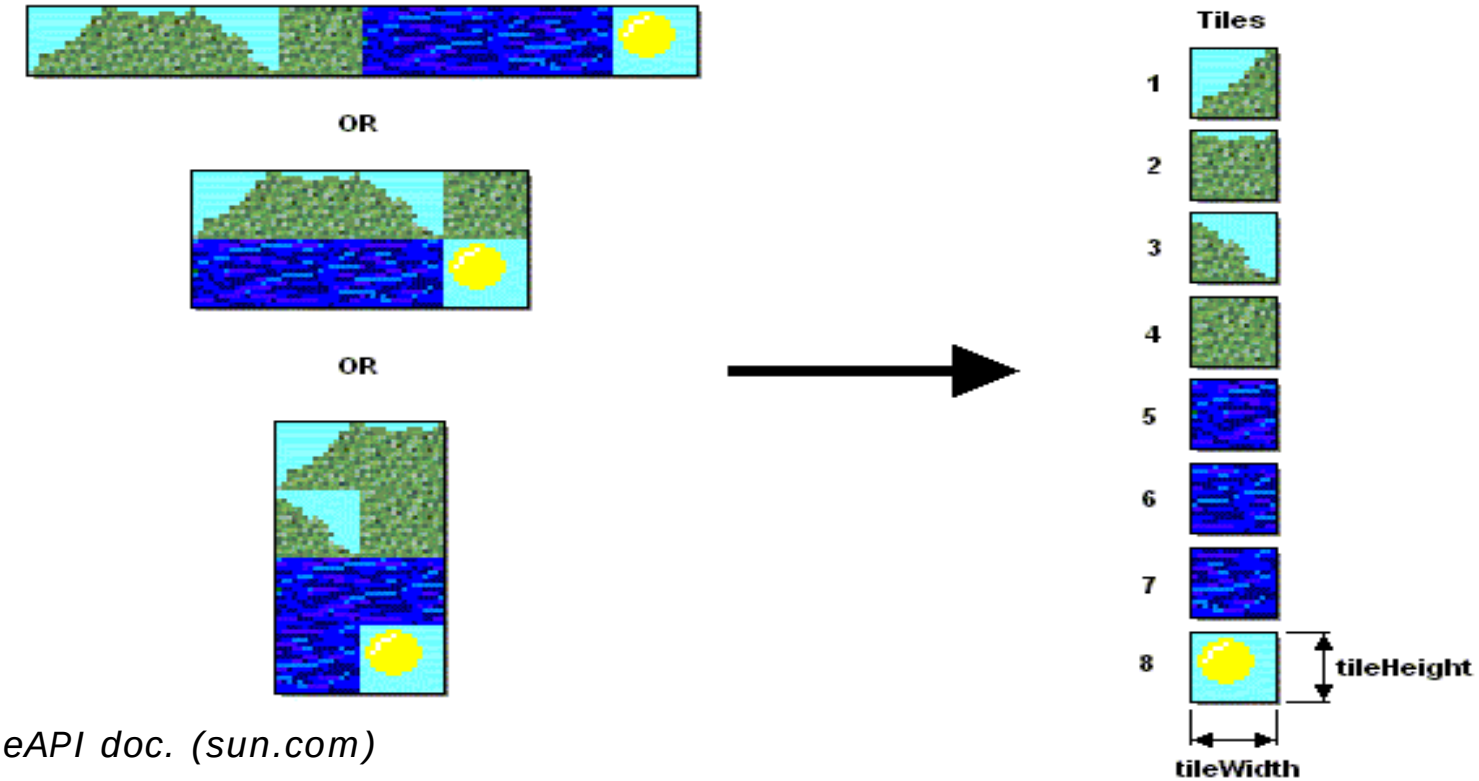
Tiled Layer

- A tiled layer is made from sets of equally sized graphics (tiles), just as set of decorative tiles to create a pretty design next to the bathtub.
- The ***tiles*** come from a single image (mutable or immutable) that is divided into equal-sized pieces to fill the TiledLayer's ***cells*** in desired order.
- By arranging (and repeating) these tiles, you can present proportionally large areas small source images.
- This technique is commonly used in 2D gaming platforms to create very large scrolling backgrounds.

NUS National University of Singapore

Tiles and Tiled Layer (Different arrangement of a tile set in an image)

- Each tile is assigned a unique index number. The tile located in the upper-left corner of the Image is assigned an index of 1. (Follows ROW major order)



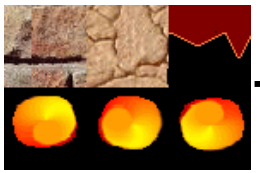
Source: GameAPI doc. (sun.com)

Question to ponder: Can you observe any difference in indexing when compared to Java indexing schemes?

NUS National University of Singapore

Creating Tiled Layer

```
try{
    bkImage = Image.createImage("/midp2proj/background.png");
}catch(IOException e) {}
int[][] map = {    //Two-D Array showing TiledLayer Map/cells
    {3,3,3,3,3,3,3,3},
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0},
    {1,2,2,1,2,1,1,2}
};
mBackground = new TiledLayer(8,4,bkImage,48,48);
for (int i=0; i<4; i++)   //ROWS
for (int j=0; j<8; j++)   //COLUMNS
  mBackground.setCell(j, i, map[i][j]);
```



NUS National University of Singapore

Background for Multiple Levels of Game

- Create a new tiled layer each time.

- If the size of the tiled layer is same, GAME API provides ,
  - **setStaticTileSet**(Image image, int tileWidth, int tileHeight)

  Method to change the TileSet.

- The number of rows and columns can not be changed.

Filling single tile for entire Tiled Layer

- **fillCells**(0,0,T1.getRows(), T.getColumns())
  - Where T1 is the *TiledLayer*

Question: Pros and Cons of Using Tiled layer instead of one single image as background.

Animated Tiles in Tiled Layer

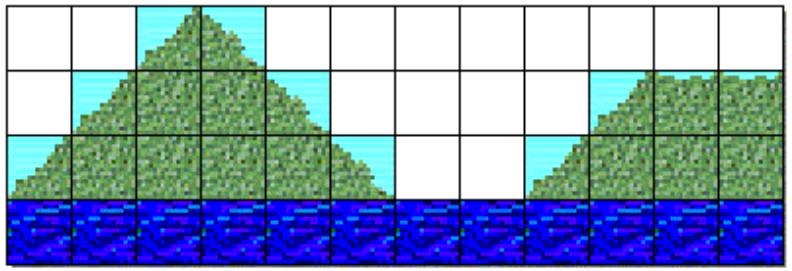
- GAME API provides facilities to define several *animated tiles*.
- An animated tile is a virtual tile that is dynamically associated with a static tile; the appearance of an animated tile will be that of the static tile that it is currently associated with.

Cells

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

Animated Tiles

-1	=	5



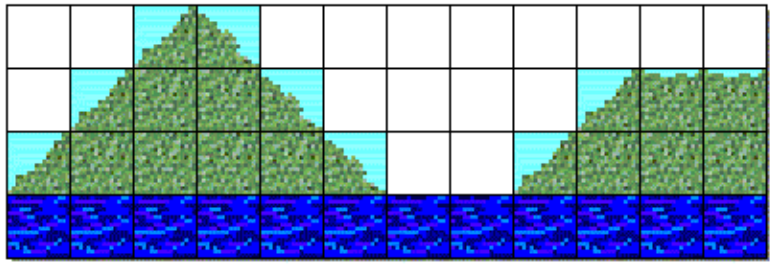
setAnimattedTile(-1,5)

Cells

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

Animated Tiles

-1	=	7

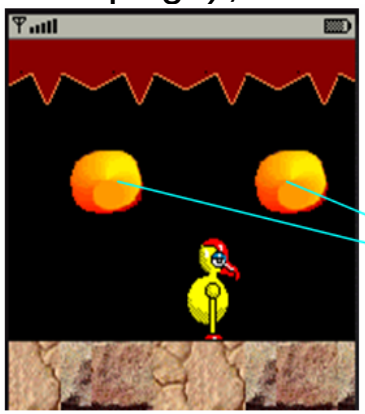


setAnimattedTile(-1,7)

Source: GameAPI doc. (sun.com)

Creating Animated Tiles

```
try{
  bkImage = Image.createImage("/midp2proj/background.png");
}catch(IOException e) {}
int[][] map = {
  {3,3,3,3,3,3,3,3},
  {0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0},
  {1,2,2,1,2,1,1,2}
};
mBackground = new TiledLayer(8,4,bkImage,48,48);
for (int i=0; i<4; i++)   //ROWS
for (int j=0; j<8; j++)   //COLUMNS
    mBackground.setCell(j, i, map[i][j]);
```




```
mAnimatedIndex = mBackground.createAnimatedTile(4);
mBackground.setCell(3,1,mAnimatedIndex);
mBackground.setCell(5,1,mAnimatedIndex);
```

Initial Tile

.....

Changing frames/tiles for animation (in GAME loop)

```
mBackground.setAnimatedTile(mAnimatedIndex,aniTiles++);
if (aniTiles > 6) aniTiles = 4;
```

NUS National University of Singapore

FYORP

Basic Methods of TiledLayer

- **createAnimatedTile**(int staticTileIndex)
  - Creates a new animated tile and returns the index that refers to the new animated tile.
- **fillCells**(int col, int row, int numCols, int numRows, int tileIndex)
  - Fills a region cells with the specific tile.
- **getAnimatedTile**(int animatedTileIndex)
  - Gets the tile referenced by an animated tile.
- **getCell**(int col, int row)
  - Gets the contents of a cell.
- **getCellHeight**()
  - Gets the height of a single cell, in pixels.
- **getCellWidth**()
  - Gets the width of a single cell, in pixels.
- **getColumns**()
  - Gets the number of columns in the TiledLayer grid.

Basic Methods of TiledLayer (...)

- **getRows**()
  - Gets the number of rows in the TiledLayer grid.
- **paint**(Graphics g)
  - Draws the TiledLayer.
- **setAnimatedTile**(int animatedTileIndex, int staticTileIndex)
  - Associates an animated tile with the specified static tile.
- **setCell**(int col, int row, int tileIndex)
  - Sets the contents of a cell.
- **setStaticTileSet**(Image image, int tileWidth, int tileHeight)
  - Change the static tile set.

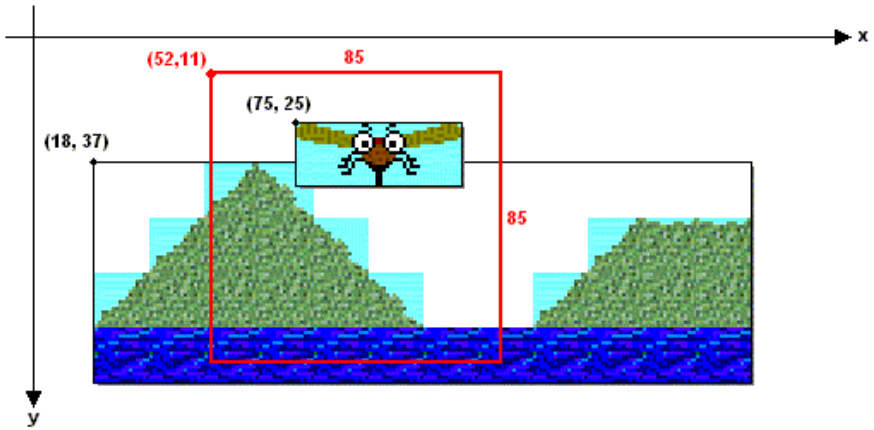
NUS National University of Singapore

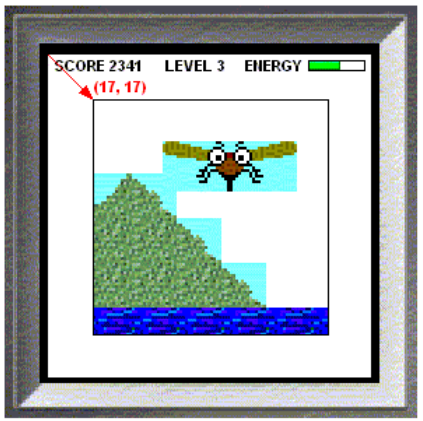
Layer Manager

- The *LayerManager* manages a series of Layers (z-order). The LayerManager simplifies the process of rendering appropriate region of Layers.
- Layers have an index, which indicates their position top to bottom. A position (index) 0 is on top.
- The indices are always contiguous; that is, if a Layer is removed, the indices of subsequent Layers will be adjusted to maintain continuity.
- Simple constructor.
  - **LayerManager**()
- Adding Layers
  - **append**(Layer l) [*at bottom, highest index*]
  - **insert**(Layer l, int index)
  - **paint**(Graphics g, int x, int y)
    - » Renders the LayerManager's current view window at the specified location.

Layer Manager – View window

- *View window* - rectangular portion of the scene that will be drawn / painted / rendered.
- By Default, the view window has its origin at 0,0 and is as large as it can be (Integer.MAX_VALUE for both width and height).
- ***setViewWindow(int x, int y, int width, int height)*** method is used to set the view window. Usually fixed at a size that is appropriate for the device's screen.
- Scrolling, panning, controls the size for users view.





Source: GameAPI doc. (sun.com)

NUS National University of Singapore

FYORP

Layer Manager

Method	Purpose
append(Layer l)	Appends a Layer to this LayerManager behind all other Layers. (gets highest index)
getLayerAt(int index)	Gets the Layer with the specified index.
getSize()	Gets the number of Layers in this LayerManager.
insert(Layer l, int index)	Inserts a new Layer in this LayerManager at the specified index.
paint(Graphics g, int x, int y)	Renders the LayerManager's ***current view window*** at the specified location.
remove(Layer l)	Removes the specified Layer from this LayerManager.
setViewWindow(int x, int y, int width, int height)	Sets the view window on the LayerManager.

NUS National University of Singapore

Summary

- GameCanvas
- Layer – Sprite, TiledLayer
- LayerManager

Demo of 2D Games

NUS National University of Singapore

Low Level Graphics API support (Graphics object)

- Creating Graphics object for off-screen drawing (Full screen Graphics object)

  – *Image img = Image.createImage(getWidth(), getHeight());*
  Graphics g = img.getGraphics();

  g.<primitive drawing methods>

NUS National University of Singapore

Graphics object – drawing methods

Method	Purpose
drawString(String text, int x, int y, int anchor)	Draws the specified String at the given position using the current font and color.
drawImage(Image image, int x, int y, int anchor)	Draws the specified image at the given position.
drawLine(int x1, int y1, int x2, int y2)	Draws a line between the coordinates (x1,y1) and (x2,y2) using the current color and stroke style.
drawRect(int x, int y, int width, int height)	Draws the outline of the specified rectangle using the current color and stroke style
fillRect(int x, int y, int width, int height)	Draws a filled rectangle with the current color.

NUS National University of Singapore

Graphics object – drawing methods

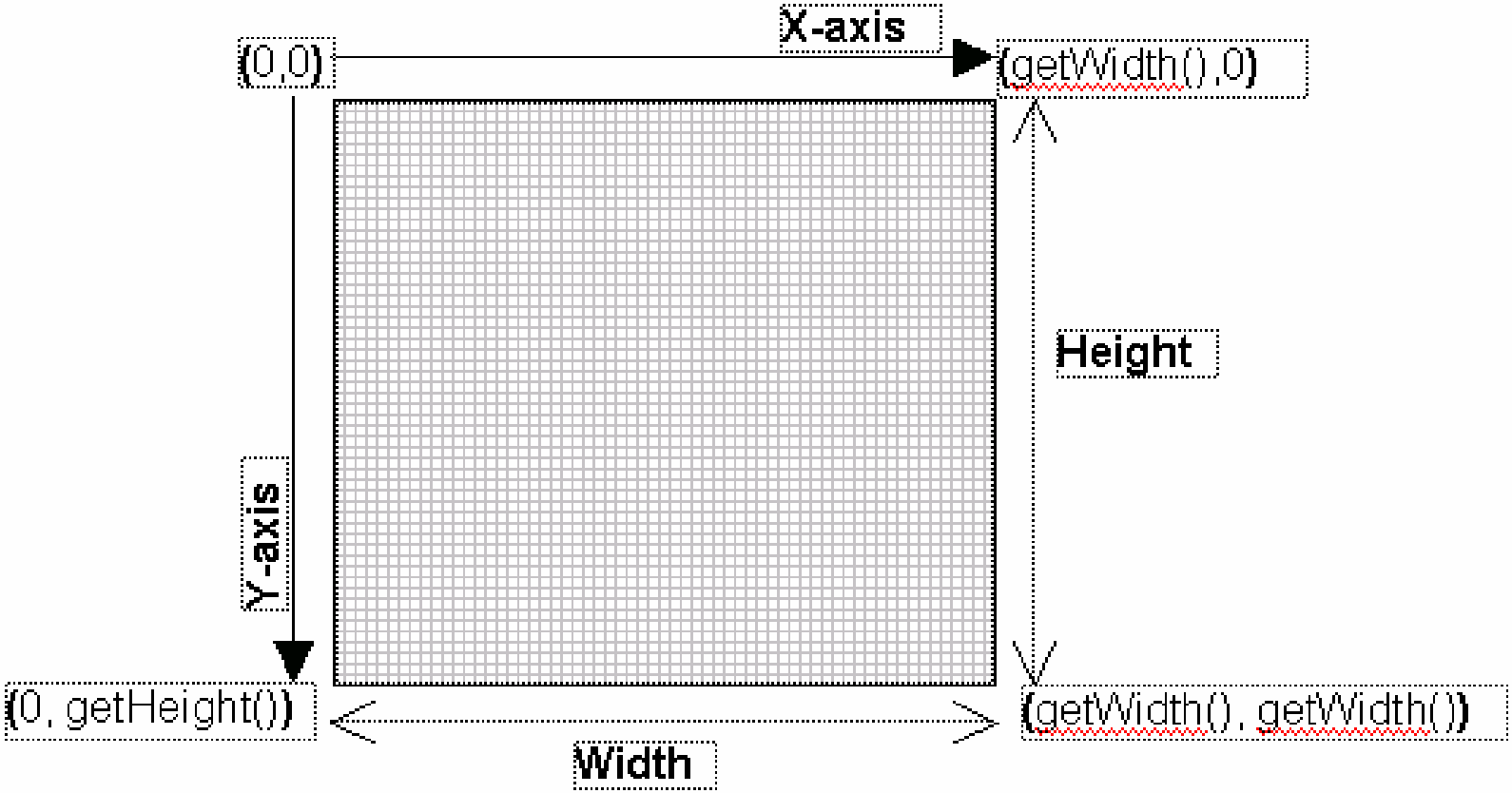
Method	Purpose
setColor(int red, int green, int blue)	Sets the current color to the specified RGB values
setFont(Font font)	Sets the font for all subsequent text rendering operations
setGrayScale(int value)	Sets the current grayscale to be used for all subsequent rendering operations
setStrokeStyle(int style)	Sets the stroke style used for drawing lines, arcs, rectangles, and rounded rectangles

For full-list: refer MIDP 2.0 API/Graphics object

NUS National University of Singapore

Graphics object: Coordinate System

- The coordinate system represents locations between pixels, not the pixels themselves.



NUS National University of Singapore

Graphics object: Coordinate System

- The origin of the coordinate system can be changed using the translate (int x, int y) method. It will add the coordinates (x,y) with all the subsequent drawing operations automatically. For example,

g.translate(getWidth()/2,getHeight/2)

»will cause the center point of the screen to be the origin for the subsequent drawings.

NUS National University of Singapore

Graphics object: Clipping

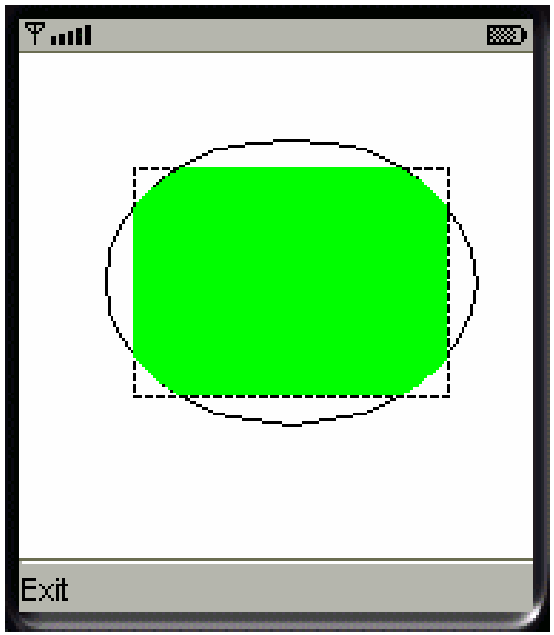
- A clip is a rectangle region in the destination of the *Graphics* object that responds to the subsequent drawing operations. There can be one clip per *Graphics* object.

Method	Purpose
setClip(int x, int y, int width, int height)	Sets a new rectangle clip region specified by the coordinates. Subsequent drawings will be effective only inside this region. Any drawing outside this region will be ignored.
getClipX(), **getClipY**(), **getClipHeight**(), **getClipWidth**()	Returns the X offset, Y offset, height and width of the current clipping area.

NUS National University of Singapore

Graphics object: Clipping

```
g.setColor(255,255,255);
g.fillRect(0,0,getWidth(), getHeight());
g.setColor(0,0,0);
g.drawArc(30,30,130,100,0,360);
g.setStrokeStyle(g.DOTTED);
g.drawRect(40,40,110,80);
g.setClip(40,40,110,80);
g.setColor(0,255,0);
g.fillArc(30,30,130,100,0,360);
```



NUS National University of Singapore

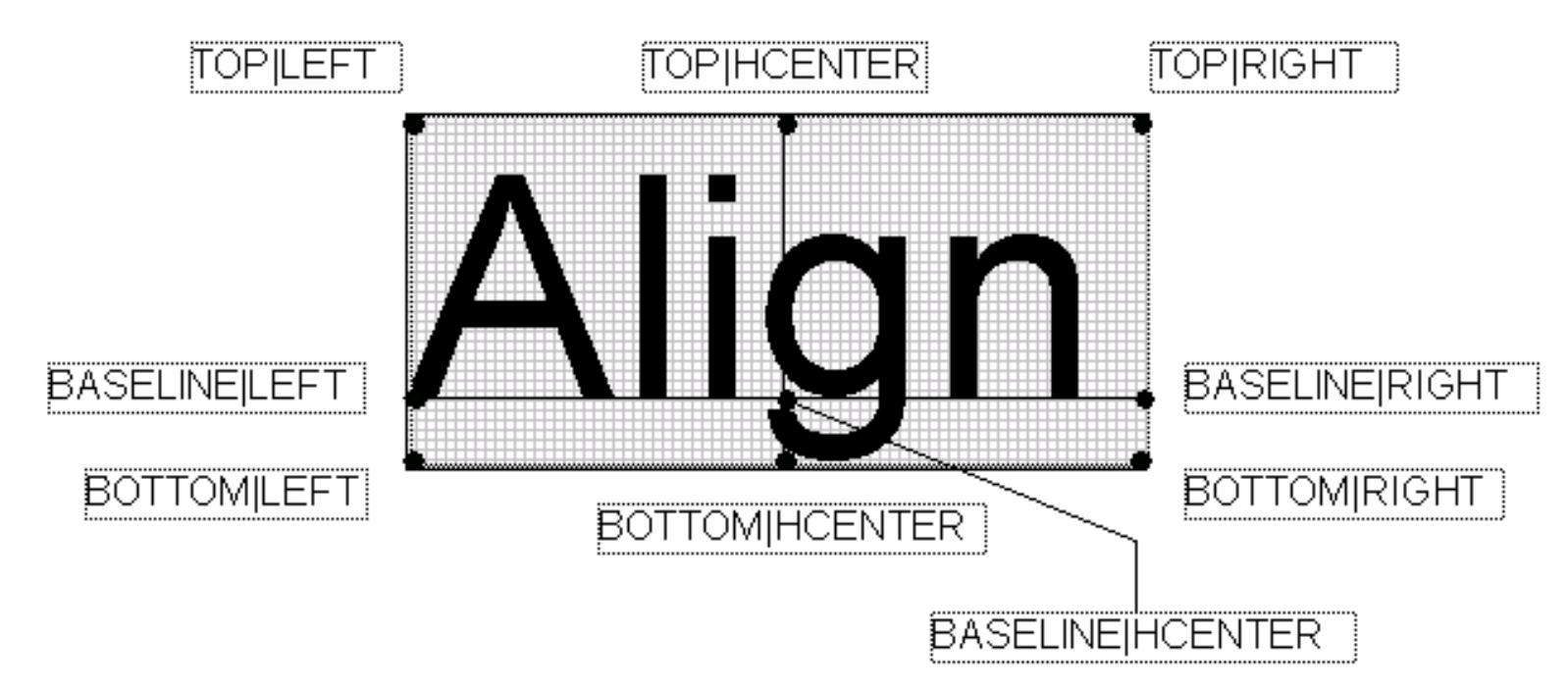
Graphics object: Drawing Texts (Fonts)

- The method *g.drawString()* is used to draw text on the screen.
- *g.setFont()* sets the font for subsequent text rendering to the Font passed as parameter to the *g.setFont()* method.
- Font class
  - To create a Font object, the *lcdui* defines a Font class with the *getFont()* method which takes three parameters: Size, Style and Face.
  - ***getFont****(int face, int style, int size)*

Parameter	**Constants**
Face	FACE_SYSTEM, FACE_MONOSPACE, FACE_PROPORTONAL
Style	STYLE_PLAIN, STYLE_ITALIC, STYLE_BOLD, STYLE_UNDERLINED
Size	SIZE_SMALL, SIZE_MEDIUM,SIZE_LARGE

NUS National University of Singapore

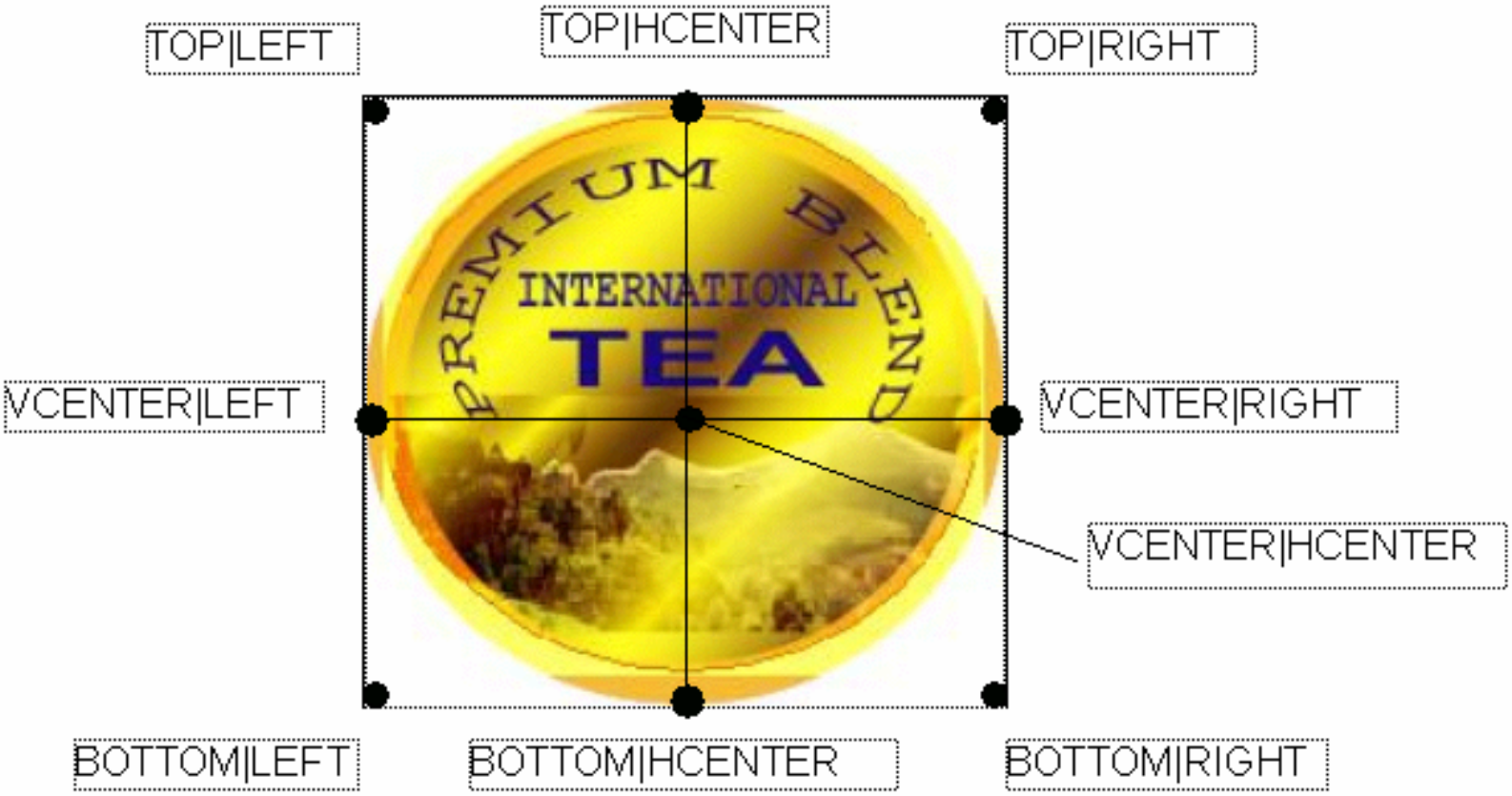
Graphics object: Drawing Texts (Anchor points)



NUS National University of Singapore

Drawing Images (Anchor points)

- *g.drawImage()* – displays mutable images.



NUS National University of Singapore

The Canvas

- Super-class of GameCanvas
- Requires implementation of **paint** method. It will be called when the Canvas is shown on the screen. (When it becomes current object of the Display)

```
protected void paint(Graphics g) {
  g.<primitive drawing methods>

   .......

}
```

NUS National University of Singapore

The Canvas… (Key interrupt)

- Key interrupt handling
  - Key events return a *Key Codes*, which are directly bounded to the physical keys. The mapping from key to key code is device dependent.
  - MIDP defines the following key codes, which represents the keys on a ITU-T standard keypad: KEY_NUM0, KEY_NUM1, KEY_NUM2, KEY_NUM3, KEY_NUM4, KEY_NUM5, KEY_NUM6, KEY_NUM7, KEY_NUM8, KEY_NUM9, KEY_STAR, KEY_POUND.
  - *Canvas defines another method* ***getKeyName()*** *which returns the name of the key for the given key code.*
  - Canvas provides the following key event callback methods: ***keyPressed(), keyReleased(),*** **and** ***keyRepeated().***

- Compared the key polling, it is easier to deal with **key-released** and **key-repeated** events through key interrupt methods

NUS National University of Singapore

The Canvas... (Game Action)

- Applications, which need only game related events and arrow key events, can use the game actions rather than key codes to maximize portability.
- MIDP has defined the following game actions: UP, DOWN, LEFT, RIGHT, FIRE, GAME_A, GAME_B, GAME_C, and GAME_D. The game actions are mapped to one or more keys.
- Portable applications can call the ***getGameAction()*** method to get the game action represented by the given key code.

- *Listing 4.4 events and game actions*

NUS National University of Singapore

The Canvas… (Pointer Interrupt)

- Pointing devices – mouse, touch screen, stylus and trackball, etc.
- Canvas class provides three methods to handle pointer events: ***pointerPressed()***, ***pointerDragged()***, and ***pointerReleased()***.
- Example : *Listing 4.5 Pointer events.*

NUS National University of Singapore

Vector Graphics in Mobile Devices

- Vector graphics Vs Raster graphics
  - Technology of choice for scalable image and animation.
  - Scalability
  - Animation
  - Interactivity
  - Search-ability

Images source: svg.org

NUS National University of Singapore

Scalable Vector Graphics

- SVG is a XML based language for describing two-dimensional vector graphics and graphical applications.
- Specification: http://www.w3.org/Graphics/SVG/
  - SVG 1.1, 1.2 for web
  - SVG Basic and SVG Tiny for Mobile Devices (Phones and PDAs resp.), Released Jan 2003 [Mobile SVG profiles]
  - SVG Print for printing and achieving SVG
  - sXBL is a binding language for SVG content
- Filename extension .svg
- SVG spec has three types of graphical objects:
  - vector graphics shapes
  - images
  - Text
- SVG specification includes a DOM (Document Object Model) API to allow high-level manipulation of content.
- Different from Vector Graphics used in **Flash**. (SWF format, is not XML based)
- Various Tools for Authoring:
  - eg. “Ikivo Animator SVG-T tool (SVG Tiny)” special free offer for Sony Ericsson Developers - http://developer.sonyericsson.com

NUS National University of Singapore

Phones with Built-in SVG

- **Nokia**: 3250, 5500 Sport, 6265, 6233, 6234, 6280, 6282, 7370, 7710, E50, E60, E61, E70, N70, N71, N72, N73, N80, N90, N91, N92, N93
- **Sony Ericsson**: D750, F500, K300, K310, K500, K508, K510, K600, K608, K610, K700, K750, K790, K800, M600, P990, S600, S700, S710, V600, V630, V800, W300, W550, W600, W700, W710, W800, W810, W850, W900, W950, Z500, Z520, Z530, Z550, Z710, Z800
- **Motorola**: C975, C980, E770V, E1000, i870, V3X, V975, V980, V1050
- http://svg.org/special/svg_phones - full list of SVG phones

NUS National University of Singapore

JSR 226: Scalable 2D Vector Graphics API for J2ME

- JSR 226 headed by Nokia in the Java Community Process (JCP) www.jcp.org
- Based on Tiny SVG specifications. New, phones with SVG API are yet to be released.
- The API supports,
  - loading and rendering of scalable vector images, stored in SVG/Tiny SVG graphics format.
  - low-level graphics primitives
  - supports Graphics Overlay and Layering

NUS National University of Singapore

SVG Demo

- More on SVG in future sessions …..