CS371m - Mobile Computing

2D Graphics
A Crash Course in Using
(Android) 2D Graphics Libraries
2D Graphics Crash Course

• Render
  – Heating animal remains to extract fat
  – OR
  – generating an image from a (mathematical) model
Using Graphics

• Not an end in itself
• Create a richer, easier to use UI
• Display information in a easier to understand way
Richer UI

• The (OLD) Apple Effect?
UI Design Changes

• The (New) Apple Effect
iOS 7 and iOS 6
SIDETRACK ON THE VISUAL DISPLAY OF INFORMATION
Visual Display of Information

• Edward Tufte
  – Yale professor (poly sci, stats, and cs)
  – spark lines
  – small multiple
  – critical of PowerPoint
Visual Display

• Spark Lines

- S&P 500
  - 2,059.28
  - -8.61 (0.42%)

- Dow
  - 17,701.82
  - -74.30 (0.42%)

- Nasdaq
  - 4,870.93
  - -29.95 (0.61%)

• Small Multiple

  - Engineering
  - Operations
  - Accounting

  - HR
  - IT
  - Manufacturing

  Departmental Salary Expenses
Joseph Minard
Napoleon's Invasion of Russia

• Location, size of army, data, temperature
John Snow - Deaths from Cholera, 1854
Broad Street Outbreak (The Ghost Map)
Visualizing Data

• Ben Fry, one of the creators of Processing
All Roads
Files on Disk - WinDirStats
S&P 500 Heat Map
## CS324E Heat Map

<table>
<thead>
<tr>
<th>Stock</th>
<th>Percent Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>BA</td>
<td>1.29</td>
</tr>
<tr>
<td>BAC</td>
<td>1.01</td>
</tr>
<tr>
<td>JPM</td>
<td>0.94</td>
</tr>
<tr>
<td>VZ</td>
<td>0.7</td>
</tr>
<tr>
<td>AA</td>
<td>0.68</td>
</tr>
<tr>
<td>MMM</td>
<td>0.61</td>
</tr>
<tr>
<td>GE</td>
<td>0.61</td>
</tr>
<tr>
<td>AXP</td>
<td>0.47</td>
</tr>
<tr>
<td>IBM</td>
<td>0.44</td>
</tr>
<tr>
<td>PFE</td>
<td>0.4</td>
</tr>
<tr>
<td>CVX</td>
<td>0.38</td>
</tr>
<tr>
<td>KO</td>
<td>0.38</td>
</tr>
<tr>
<td>INTC</td>
<td>0.37</td>
</tr>
<tr>
<td>DIS</td>
<td>0.35</td>
</tr>
<tr>
<td>HPQ</td>
<td>0.35</td>
</tr>
<tr>
<td>XOM</td>
<td>0.33</td>
</tr>
<tr>
<td>PG</td>
<td>0.33</td>
</tr>
<tr>
<td>UNH</td>
<td>0.28</td>
</tr>
<tr>
<td>CAT</td>
<td>0.25</td>
</tr>
<tr>
<td>UTX</td>
<td>0.19</td>
</tr>
<tr>
<td>CSCO</td>
<td>0.11</td>
</tr>
<tr>
<td>MCD</td>
<td>0.06</td>
</tr>
<tr>
<td>HD</td>
<td>0.02</td>
</tr>
<tr>
<td>MRK</td>
<td>0.02</td>
</tr>
<tr>
<td>JNJ</td>
<td>-0.01</td>
</tr>
<tr>
<td>MSFT</td>
<td>-0.06</td>
</tr>
<tr>
<td>WMT</td>
<td>-0.17</td>
</tr>
<tr>
<td>TRV</td>
<td>-0.46</td>
</tr>
<tr>
<td>T</td>
<td>-0.67</td>
</tr>
<tr>
<td>DD</td>
<td>-0.86</td>
</tr>
</tbody>
</table>

**Update Values**
Image Processing
Color Histogram
Histogram Equalization
ANDROID 2D GRAPhICS
Android Graphics

• NOT the Java awt or swing packages
• custom set of classes
• Canvas: class that holds code for various "draw" methods
  – Similar to Java Graphics, Graphics2D objects
• Paint: Controls the drawing. A whole host of properties.
• Bitmap: the things drawn on
• Drawable: the thing to draw. (rectangles, images, lines, etc.)
• Typeface: for fonts
Two Methods for Displaying 2D Graphics

• Two approaches

• draw graphics or animations into a View object that is part of layout
  – define graphics that go into View
  – simple approach for non dynamic graphics or simple "tweened" animations

• Draw graphics directly to a Canvas
  – the complex way, but with more control
Simple Graphics

• Use Drawables in Views
• Create a folder res/drawable
• Add images
  – png (preferred)
  – jpg (acceptable)
  – gif (discouraged)

• Images can be added as background for Views
ADDING DRAWABLES TO VIEWS
Adding Drawables to Views

- Change background to an image
  - previously used background colors

```xml
<LinearLayout xmlns:android="http://schemas.android.com/
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical"
android:background="@drawable/home_office"/>
```
Top Ten With Image Background

Pick The Date for Top 10 List

Feb 21 2012

Find Top 10

From the Home Office in St. Charles, Missouri
Add ImageView to Layout

• In the main.xml for top ten

```xml
<ImageView
    android:id="@+id/imageView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/home_office" />
```
ImageView Attributes

• scaleType: how image should be moved or resized in the ImageView
• tint: affects color of image
• more to position image in ImageView
Changing ImageView Programmatically

• Randomly set the alpha (transparency of the image)
• Or pick an image randomly
• Or set image based on month (Season)

```java
private void changeImage() {
    ImageView iv = (ImageView) findViewById(R.id.imageView1);
    Drawable image = getResources()
        .getDrawable(R.drawable.home_office2);
    image.setAlpha((int) (Math.random() * 100) + 50);
    iv.setImageDrawable(image);
}
```
DRAWING VIA A CANVAS
Using a Canvas

• Simple way -> Create a custom View and override the `onDraw` method
• The Canvas is a parameter to `onDraw`
• Create a class that extends View
  – override the 2 parameter constructor
  – override the `onDraw` method
  – perform custom drawing in the `onDraw` method
  – add the View to the proper layout
Requesting Redraws of a View

• call invalidate() on a View to redraw it
  – invalidate redraws the whole View
  – possible to redraw only a portion of the View, the part that changed
  – several overloaded versions of invalidate that accept a region of the View
  – only that portion redrawn

• Override the onDraw method for the View to redraw

• for really complex drawing and graphics move drawing off of the UI thread to a SurfaceView (more complex)
Simple Graphics Example

Add CustomView to XML

• in main.xml
• add custom View as element in LinearLayout

<scottm.examples.GraphicsView
  android:id="@+id/graphicsView"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:padding="5dp" />
public class GraphicsView extends View {

    private static final String TAG = "GraphicView";

    public GraphicsView(Context context, AttributeSet attrs) {
        super(context, attrs);
        Log.d(TAG, "in 2 param constructor");
    }

    public GraphicsView(Context context) {
        super(context);
        Log.d(TAG, "in 1 param constructor");
    }
}
@Override
protected void onDraw(Canvas canvas) {

    Paint p = new Paint();
    p.setColor(Color.WHITE);
    int w = getWidth();
    int h = getHeight();
    int offset = 10;
    canvas.drawRect(offset, offset,
            w - offset * 2, h - offset * 2, p);

    int circleRadius = Math.min(w, h) / 3;
    p.setColor(getResources().getColor(R.color.BurntOrange));
    canvas.drawCircle(w/2, h/2, circleRadius, p);
Simple Graphics Result
GuessFour

- Board drawn in `onDraw` method of a View
- Board will resize for different devices
- Lines, ovals, rectangles, and texts
Paint

• typically create Paint with anti aliasing enable

• `Paint p =
  new Paint(Paint.ANTI_ALIAS_FLAG);`
Anti Aliasing - The Jaggies
Using the Canvas Class

• methods to draw:
  • rectangles
  • lines
  • arcs
  • paths
  • images
  • circles
  • ovals
  • points
  • text
  • and many more

• Ability to set the "clip", portion of canvas where drawing takes place
  – commands to draw something that is outside clip are ignored

• Ability to translate, rotate, and scale the canvas
Paint Object

• many, many attributes and properties including:
  – current color to draw with
  – whether to fill or outline shapes
  – size of stroke when drawing
  – text attributes including size, style (e.g. underline, bold), alignment,
  – gradients
Gradients

• 3 kinds of gradients
• LinearGradient
• RadialGradient
• SweepGradient
• at least 2 color, but possibly more
• flows from one color to another
Linear Gradient

```java
public LinearGradient(float x0, float y0, float x1, float y1, int color0, int color1, Shader.TileMode tile)

Create a shader that draws a linear gradient along a line.

Parameters
- x0: The x-coordinate for the start of the gradient line
- y0: The y-coordinate for the start of the gradient line
- x1: The x-coordinate for the end of the gradient line
- y1: The y-coordinate for the end of the gradient line
- color0: The color at the start of the gradient line.
- color1: The color at the end of the gradient line.
- tile: The Shader tiling mode

// Simple gradient
Paint p = new Paint(Paint.ANTI_ALIAS_FLAG);
LinearGradient lg = new LinearGradient(0, 0, 25, 50,
    Color.RED, Color.BLUE, Shader.TileMode.MIRROR);
p.setShader(lg);
canvas.drawOval(new RectF(0, 0, 300, 200), p);
```
LinearGradient

GraphicsExamples
public RadialGradient (float x, float y, float radius, int color0, int color1, Shader.TileMode tile)

Create a shader that draws a radial gradient given the center and radius.

Parameters

x The x-coordinate of the center of the radius
y The y-coordinate of the center of the radius
radius Must be positive. The radius of the circle for this gradient
color0 The color at the center of the circle.
color1 The color at the edge of the circle.
tile The Shader tiling mode

// radial gradient
RadialGradient rg = new RadialGradient(200, 400, 125,
    Color.BLUE, Color.GREEN, Shader.TileMode.MIRROR);
p.setShader(rg);
canvas.drawCircle(200, 325, 125, p);
RadialGradeint

- add depth to pegs and open spots in Guess Four game
public **SweepGradient** (float cx, float cy, int[] colors, float[] positions)

A subclass of Shader that draws a sweep gradient around a center point.

**Parameters**

- **cx**
  - The x-coordinate of the center
- **cy**
  - The y-coordinate of the center
- **colors**
  - The colors to be distributed between around the center. There must be at least 2 colors in the array.
- **positions**
  - May be NULL. The relative position of each corresponding color in the colors array, beginning with 0 and ending with 1.0. If the values are not monotonic, the drawing may produce unexpected results. If positions is NULL, then the colors are automatically spaced evenly.
// sweep gradient
int numColors = 4;
int angleIncrement = 360 / numColors;
int[] rainbow = new int[numColors * 2];
float[] hsv = {0, 1, 1};
for(int i = 0; i < rainbow.length / 2; i++) {
    rainbow[i] = Color.HSVToColor(hsv);
    hsv[0] += angleIncrement;
}
for(int i = rainbow.length / 2; i < rainbow.length; i++) {
    rainbow[i] = rainbow[rainbow.length - i];
}
SweepGradient sg = new SweepGradient(300, 600, rainbow, null);
p.setShader(sg);
canvas.drawCircle(300, 600, 125, p);
SweepGradient
SweepGradient

```java
SweepGradient sg = new SweepGradient(300, 600,
    new int[] {Color.RED, Color.YELLOW, Color.RED},
    null);
p.setShader(sg);
canvas.drawCircle(300, 600, 125, p);
```
SweepGradient

SweepGradient sg = new SweepGradient(300, 600,
   new int[] {Color.RED, Color.YELLOW, Color.RED},
   new float[] {0, 0.3f, 1});
p.setShader(sg);
canvas.drawCircle(300, 600, 125, p);
ANIMATION
Simple Animation

• Animation altering some property of a 2D primitive
  – position
  – size
  – color
  – alpha

• Simplest animation loop, after onDraw, call invalidate
  – at the mercy of the UI frame rate
Simple (And Poor) Animation Approach

```java
@Override
protected void onDraw(Canvas canvas) {
    handleFrameRateChecks();

    int x = getWidth() / 2;

    canvas.drawCircle(x, y, CIRCLE_RADIUS, p);
    y++;
    if(y < getHeight())
        invalidate();
}
```

- draw as fast as possible
  - emulator frame rate (on my machine) 12 fps
  - dev device frame rate, high 50s fps
Better Animation Loop

• Create a Handler to delay / sleep
• update method will call sleep method on the Handler

```java
private RefreshHandler mRedrawHandler = new RefreshHandler();

class RefreshHandler extends Handler {

    @Override
    public void handleMessage(Message msg) {
        GraphicsView.this.update();
        GraphicsView.this.invalidate();
    }

    public void sleep(long delayMillis) {
        this.removeMessages(0);
        sendMessageDelayed(obtainMessage(0), delayMillis);
    }
}
```
• **animation loop**
  – update tells handler to sleep
  – handler calls update when it wakes up ...

```java
public void update() {
    if (mode == RUNNING) {
        handleFrameRateChecks();
        long now = System.currentTimeMillis();
        if (now - prevTime > moveDelay) {
            prevTime = now;
            x = getWidth() / 2;
            y += SPEED;
            if (y > getHeight())
                mode = STOPPED;
        }
        mRedrawHandler.sleep(moveDelay);
    }
}
```
TWEENED ANIMATIONS
Simple Animations

• Tweened Animations
  – also know as View Animations
• provide a way to perform simple animations on Views, Bitmaps, TextViews, Drawables
• provide start point, end point, size, rotation, transparency, other properties
• Can set up tweened animation in XML or programmatically
Tweened Animations

• Used to alter one of four properties of a single Drawable resource
  – Bitmap graphic (image), shape, TextView, any View (background)

• Tweened animations can affect
  – Alpha (transparency / opaqueness)
  – Rotation
  – Scale (size)
  – Location (movement, Translate)
Tweened Animations

• define interpolator in XML (optional)
  – allow animation to be repeated, accelerate, decelerate, "bounce", and more
  – can use built in interpolators or create your own

• define animation in XML
  – alpha, rotate, scale, translate
  – define from value, to value, and duration

• In program load and start animation in response to some event
  – Guess Four, invalid choice, solved puzzle
Guess Four res/anim

• shake up down

```xml
<translate
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:fromYDelta="0"
    android:toYDelta="25"
    android:duration="2000"
    android:interpolator="@anim/cycle_7" />
```

• shake left right

```xml
<translate
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:fromXDelta="0"
    android:toXDelta="25"
    android:duration="1000"
    android:interpolator="@anim/cycle_7" />
```
GuessFour res/anim

- cycle_7.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<cycleInterpolator
 xmlns:android="http://schemas.android.com/apk/res/android"
 android:cycles="7" />
```

- spin.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<rotate xmlns:android="http://schemas.android.com/apk/res/android"
 android:duration="3000"
 android:interpolator="@android:anim/linear_interpolator"
 android:pivotX="50%"
 android:pivotY="15%"
 android:toDegrees="1080" />
```
GuessFour Example

• On error board shakes back and forth
• On win board spins
• From BoardView in GuessFour

```java
public void shakeLeftRight() {
    Log.d(TAG, "in shake! Trying to start animation!");
    startAnimation(AnimationUtils.loadAnimation(game, R.anim.shake));
}

public void shakeUpDown() {
    Log.d(TAG, "in shake! Trying to start animation!");
    startAnimation(AnimationUtils.loadAnimation(game, R.anim.shake_up_down));
}

public void spin() {
    Log.d(TAG, "in shake! Trying to start animation!");
    startAnimation(AnimationUtils.loadAnimation(game, R.anim.spin));
}
```
More Tweened Examples

• hyperspace example from android dev site
• rotate and change alpha
• animation types:
  – alpha
  – scale
  – translate
  – rotate

More Tweened Examples

• Moving Button

• **Note**, tweened animations draw the button in a different spot

• But, the button's location is not really changed
Up and Down Animation

```xml
<?xml version="1.0" encoding="utf-8"?>
<translate
    xmlns:android="http://schemas.android.com
    android:fromYDelta="0"
    android:toYDelta="700"
    android:duration="2000"
    android:repeatCount="infinite"
    android:repeatMode="reverse"/>
```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.button_move);
    randNumGen = new Random();
    Intent intent = getIntent();
    int animationType = TWEEN;
    if(intent.hasExtra(ANIMATION_TAG)) {
        animationType = intent.getExtras().getInt(ANIMATION_TAG);
    }
    if(animationType == TWEEN)
        tweenedAnimation();
    else
        propertyAnimation();
}

Use extra from Intent to determine what type of animation to perform.
Tweened Animation

```java
public void tweenedAnimation() {
    Button movingButton
        = (Button) findViewById(R.id.change_background);

    movingButton.startAnimation(
        AnimationUtils.loadAnimation(this,
            R.anim.up_and_down));
}
```
Change Background Color

• Called when button clicked
  – onClick attribute

```java
public void changeBackgroundColor(View v) {
    View target = (View) findViewById(R.id.linear_layout_button);
    int red = randNumGen.nextInt(NUM_SHADES);
    int green = randNumGen.nextInt(NUM_SHADES);
    int blue = randNumGen.nextInt(NUM_SHADES);
    target.setBackgroundColor(Color.argb(255, red, green, blue));
}
```

Result?
PROPERTY ANIMATIONS

-AVAILABLE POST GINGERBREAD
Property Animations

• A more general animation framework
• Tweened Animations can only affect alpha, scale, rotation, and position
• Property Animations can affect any property of an object
  – typically a View or Drawable
• can be defined in sets, change multiple properties at a time
• animation created separate from target object, reuse with different objects

Property Animation - Classes

• ValueAnimator
  – base class for property animations

• Object Animator
  – convenience class for animating specific object and property

• ViewPropertyAnimator
  – optimized, simple animation of View objects

• evaluator classes such as ArgbEvaluator to animate property by defining how it changes over time
Some Animations Simple

• API levels 12+
• View objects have animate() method

```java
public ViewPropertyAnimator animate()
```

This method returns a ViewPropertyAnimator object, which can be used to animate specific properties on this View.

Returns
ViewPropertyAnimator The ViewPropertyAnimator associated with this View.

• ViewPropertyAnimator
• methods for alpha, rotation, scale (size), translation (position)
ViewPropertyAnimator Example

• onClick method for a button:

```java
public void changeBackground(View v) {
    if(v.getAlpha() == 0)
        v.animate().alpha(1);
    else
        v.animate().alpha(0);
}
```

• button will disappear and then reappear next time clicked
More Complex Property Animation

• Object animation
• from moving button example

```java
private void propertyAnimation() {
    Button movingButton = (Button) findViewById(R.id.change_background);
    ObjectAnimator anim = ObjectAnimator.ofFloat(movingButton, "y", 0, 700);
    anim.setRepeatCount(ObjectAnimator.INFINITE);
    anim.setRepeatMode(ObjectAnimator.REVERSE);
    anim.setDuration(2000);
    anim.start();
}
```

• animated class must have a "set<Property>" method
Button Class

```java
public void setY (float y)
```

Sets the visual y position of this view, in pixels. This is equivalent to setting the `translationY` property to be the difference between the y value passed in and the current `top` property.

**Parameters**

- `y` The visual y position of this view, in pixels.

```java
public float getY ()
```

The visual y position of this view, in pixels. This is equivalent to the `translationY` property plus the current `top` property.

**Returns**

The visual y position of this view, in pixels.

- How does the Object Animator change the y value?
ObjectAnimator

```java
public static ObjectAnimator ofFloat (Object target, String propertyName,
float... values)
```

Constructs and returns an ObjectAnimator that animates between float values. A single value implies that that value is the one being animated to. Two values imply starting and ending values. More than two values imply a starting value, values to animate through along the way, and an ending value (these values will be distributed evenly across the duration of the animation).

**Parameters**

- `target` The object whose property is to be animated. This object should have a public method on it called `setName()`, where `name` is the value of the `propertyName` parameter.
- `propertyName` The name of the property being animated.
- `values` A set of values that the animation will animate between over time.

**Returns**

An ObjectAnimator object that is set up to animate between the given values.
• How can the ObjectAnimator call the right methods on the object passed?
• Declared type is Object
• must be calling setY method, right?
• Discover reflection
A Sidetrack on Reflection
Reflection

• Advanced feature of Java.
• Commonly used by programs that "examine or modify the runtime behavior of applications running in the Java virtual machine"
• The Android Property Animation framework uses reflection
Why Reflection

• Extensible features
  – like Android Property Animator framework

• Class Browsers and Visual Development Environments

• Debugger and Testing Tools
  – am I testing all the public methods?
  – coverage
Recall: Property Animation

• ObjectAnimator class a subclass of ValueAnimator
• Convenience class for property animation
• When animator created set animation time, property to animate, and the starting and ending values
• "The constructors of this class take parameters to define the target object that will be animated as well as the name of the property that will be animated. Appropriate set/get functions are then determined internally and the animation will call these functions as necessary to animate the property."
ObjectAnimator Example

• Button class must have gety and sety methods that return and accept a float

```java
private void propertyAnimation() {
    Button movingButton
        = (Button) findViewById(R.id.change_background);
    ObjectAnimator anim
        = ObjectAnimator.ofFloat(movingButton, "y", 0, 700);
    anim.setRepeatCount(ObjectAnimator.INFINITE);
    anim.setRepeatMode(ObjectAnimator.REVERSE);
    anim.setDuration(2000);
    anim.start();
}
```

• ofFloat, ofInt, ofObject, ofMulti...
Object Animator

• How does the Object animator affect the y value of the Button?

```java
public static ObjectAnimator ofFloat (Object target, String propertyName, float... values)
```

Not a button

• Recall Java, declared type, actual type
• What methods does allow compiler allow?
Class objects

• Everything in Java a primitive (byte, short, int, long, float, double, char, boolean) or an Object
  – arrays and Enums are Objects
• The Java virtual machine instantiates an immutable instance of java.lang.Class for every type of Object necessary in a running program
• Entry point for reflection
Getting the Class object

// via getClass
String str = "Olivia";
Class<? extends String> c1 = str.getClass();

// via class literal
Class<? extends String> c2 = String.class;

// via for name
Class<?> c3;
try {
    c3 = Class.forName("java.lang.String");
} catch(ClassNotFoundException exception) {
    c3 = null;
}

if(c1 == c2 && c2 == c3) {
    System.out.println("Same class");
} else {
    System.out.println("NOT same class");
}
Accessing Internals

• Class object may be used to access fields, methods, and constructors
  —... including private members
• methods that enumerate members and methods that search for a member given a name
• Like a spy
Security

• This appears to be dangerous stuff
• The ability to find out about private methods and fields
  ... and even change them
• Thus many of the methods in the Class class and Reflection API throw SecurityExceptions
• If a SecurityManager is present and permission for reflection is not granted, exceptions occur
• "If you remove this sticker, the warranty is void"
public static void listFields(Object obj) {
    Class<?> c = obj.getClass();
    for (Field f : c.getDeclaredFields()) {
        System.out.println(f);
    }
}

private final char[] java.lang.String.value
private int java.lang.String.hash
private static final long java.lang.String.serialVersionUID
private static final java.io.ObjectStreamField[] java.lang.String.serialPersistentFields
public static final java.util.Comparator java.lang.String.CASE_INSENSITIVE_ORDER
private static final long java.util.ArrayList.serialVersionUID
private static final int java.util.ArrayList.DEFAULT_CAPACITY
private static final java.lang.Object[] java.util.ArrayList.EMPTY_ELEMENTDATA
private static final java.lang.Object[] java.util.ArrayList.DEFAULTCAPACITY_EMPTY_ELEMENTDATA
private int java.util.ArrayList.size
private static final int java.util.ArrayList.MAX_ARRAY_SIZE
Enumerating Methods

```java
public static void listMethods(Object obj) {
    Class<?> c = obj.getClass();
    for (Method m : c.getDeclaredMethods()) {
        System.out.println(m);
    }
}
```

```java
public boolean java.lang.String.equals(java.lang.Object)
public java.lang.String java.lang.String.toString()
public int java.lang.String.hashCode()
public int java.lang.String.compareTo(java.lang.String)
public int java.lang.String.compareTo(java.lang.Object)
public int java.lang.String.indexOf(java.lang.String, int)
public int java.lang.String.indexOf(java.lang.String)
public int java.lang.String.indexOf(int, int)
public int java.lang.String.indexOf(int)
static int java.lang.String.indexOf(char[], int, int, char[], int, int)
static int java.lang.String.indexOf(char[], int, int, java.lang.String)
```
public class Button {
    private int x;
    private int y;

    private String text;

    public Button(int x, int y, String text) {
        this.x = x;
        this.y = y;
        this.text = text;
    }

    public int gety() {
        return y;
    }

    public void sety(int y) {
        this.y = y;
    }

    public String toString() {
        return "Button @ " + x + ", " + y + " with " + text;
    }
}
System.out.println("Object at start: " + obj);
Class<?> c = obj.getClass();
Method getMethod = c.getMethod("get" + property);
int current = (Integer) getMethod.invoke(obj);

Method setMethod = c.getMethod("set" + property, int.class);
System.out.println();
System.out.println("Object after set start: " + obj);
System.out.println();
int step = (stop - current) / steps;
for(int i = 0; i < steps; i++) {
    current += step;
    setMethod.invoke(obj, current);
    System.out.println("At step " + (i + 1) + ": " + obj);
}
Results of Method Calls - Animate!

Object at start: Button @ 50, 100 with Go!

Object after set start: Button @ 50, 100 with Go!

At step 1: Button @ 50, 150 with Go!
At step 2: Button @ 50, 200 with Go!
At step 3: Button @ 50, 250 with Go!
At step 4: Button @ 50, 300 with Go!
At step 5: Button @ 50, 350 with Go!
At step 6: Button @ 50, 400 with Go!
At step 7: Button @ 50, 450 with Go!
At step 8: Button @ 50, 500 with Go!
At step 9: Button @ 50, 550 with Go!
At step 10: Button @ 50, 600 with Go!
And Then the Unthinkable Happened

```java
public static void alterField(Object obj, String property) {
    try {
        System.out.println();
        System.out.println("object at start: " + obj);
        Class<?> c = obj.getClass();
        Field f = c.getDeclaredField(property);
        f.setAccessible(true);
        f.setInt(obj, 10000);
        System.out.println("f.setInt(obj, 10000);");
        System.out.println("object after setting field: " + obj);
    }
    catch(Exception e) {
        System.out.println("Sample code failed");
        e.printStackTrace();
    }
}
```

object at start: Button @ 50, 600 with Go!
f.setInt(obj, 100000);
object after setting field: Button @ 10000, 600 with Go!
Recall Button Class

• x is private
• no methods to access or alter x
• YIKES

```java
public class Button {
    private int y;
    private int x;

    private String text;

    public Button(int x, int y, String text) {
        this.x = x;
        this.y = y;
        this.text = text;
    }

    public int gety() {
        return y;
    }

    public void sety(int y) {
        this.y = y;
    }

    public String toString() {
        return "Button @ " + x + ", " + y + " with ";
    }
}
```
But final is safe, right????

• String fields in Java 8

```java
private final char[] java.lang.String.value
private int java.lang.String.hash
private static final long java.lang.String.serialVersionUID
private static final java.io.ObjectStreamField[] java.lang.String.COMPARATOR
```

```java
public static void changeString(String str) {
    try {
        Class<? extends String> c = str.getClass();
        Field f = c.getDeclaredField("value");
        System.out.println(f);
        f.setAccessible(true);
        f.set(str, "Olivia");
    } catch(Exception e) {
        System.out.println("Sample code failed");
        e.printStackTrace();
    }
}
```
Result of changeString

private final char[] java.lang.String.value

Sample code failed
java.lang.IllegalArgumentException: Can not set final [C field java.lang.String.value
    at sun.reflect.UnsafeFieldAccessorImpl.throwSetillegalargumentexception
    at sun.reflect.UnsafeFieldAccessorImpl.throwSetillegalargumentexception
    at sun.reflect.UnsafeQualifiedObjectFieldAccessorImpl.set(Unknown Source)
    at java.lang.reflect.Field.set(Unknown Source)
    at ReflectionExamples.changeString(ReflectionExamples.java:139)
    at ReflectionExamples.main(ReflectionExamples.java:54)

• We are good right?
public static void changeString2(String str) {
    try {
        Class<? extends String> c = str.getClass();
        Field f = c.getDeclaredField("value");
        System.out.println(f);
        f.setAccessible(true);

        // final no more
        Field modifiersField = Field.class.getDeclaredField("modifiers");
        modifiersField.setAccessible(true);
        modifiersField.setInt(f, f.getModifiers() & ~Modifier.FINAL);

        f.set(str, "Olivia".toCharArray());
    }
    catch(Exception e) {
        System.out.println("Sample code failed");
        e.printStackTrace();
    }
}
String name = "Isabelle";
System.out.println("name String: " + name);
System.out.println("name length() " + name.length());
System.out.println("name hashCode() " + name.hashCode());
oneInts(name);
changeString2(name);
System.out.println();
System.out.println("name String: " + name);
System.out.println("name length() " + name.length());
System.out.println("name hashCode() " + name.hashCode());
Result

name String: Isabelle
name length() 8
name hashCode() 230492619

private final char[] java.lang.String.value

name String: Olivia
name length() 6
name hashCode() 1
RENDERING WITH COMPLEX CALCULATIONS
More Complex Graphics

• Don't want apps to become unresponsive
• If complex graphics or animation use SurfaceView class
• Main view not waiting on onDraw to finish
• secondary thread with reference to SurfaceView
• SurfaceView draws and when done display result
Using a SurfaceView

- extend SurfaceView
- implement SurfaceHolder.Callback
  - methods to notify main View when SurfaceView is created, changed or destroyed
Simple Example

• Static Screen
• continuously draw several hundred small rectangles (points, with stroke = 10)
  — slowly fill screen and then keep changing

```java
public class StaticView extends SurfaceView
    implements SurfaceHolder.Callback {

    private static final String TAG = "Static";

    private StaticThread thread;

    public StaticView(Context context, AttributeSet attrs) {
        super(context, attrs);

        // register our interest in hearing about changes to
        SurfaceHolder holder = getHolder();
        holder.addCallback(this);
    }
```
Implement SurfaceHolder.Callback methods

```java
// called when surface changes size
@Override
public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {
}

// called when surface is first created
@Override
public void surfaceCreated(SurfaceHolder holder) {
    thread = new StaticThread(holder);
    thread.setRunning(true);
    thread.start();  // start the animation
}
```
// called when the surface is destroyed
@Override
public void surfaceDestroyed(SurfaceHolder holder) {
    // ensure that thread terminates properly
    boolean retry = true;
    thread.setRunning(false);

    while (retry) {
        try {
            try {
                thread.join();
                Log.d(TAG, "Thread stopped! " + thread);
                retry = false;
            }
            catch (InterruptedException e) {} 
        }
    }
}
Inner Class for Thread

```java
private class StaticThread extends Thread {

    private boolean running;
    private SurfaceHolder surfaceHolder;
    private Bitmap image;
    private Random random;
    private Paint paint;

    public StaticThread(SurfaceHolder sh) {
        surfaceHolder = sh;
        random = new Random();
        paint = new Paint();
        paint.setStyle(Paint.Style.STROKE);
        paint.setStrokeWidth(10);
        image = Bitmap.createBitmap(getWidth(), getHeight(),
                                   Bitmap.Config.ARGB_8888);
        Log.d(TAG, "width: " + image.getWidth());
        Log.d(TAG, "height: " + image.getHeight());
        Log.d(TAG, "image: " + image);
    }

    public void setRunning(boolean status) {
        running = status;
    }
}
```
Run Method in StaticThread

```java
@Override
public void run() {
    while (running) {
        if (surfaceHolder.getSurface().isValid()) {
            handleFrameRateChecks();
            Canvas canvas = surfaceHolder.lockCanvas();
            doDraw(canvas);
            surfaceHolder.unlockCanvasAndPost(canvas);
        }
    }
}
```

Standard Approach for Drawing on SurfaceView
Demo run()

- Pronounced flicker and jitter
- Double buffer under the hood
- We are drawing on two different Bitmaps
- Canvas does drawing onto Bitmap
Remove Flicker / Jitter

• If we draw background each "frame" then we don't redraw previous rectangles

• How about "saving" all the data?
  – points, colors
Alternative

• Recall two approaches:
  – draw on UI thread by overriding `onDraw`
    • create custom View (tutorial 4)
    • okay if not a lot of drawing
  – must keep UI thread responsive
    • complex drawing or animations using `SurfaceView`

• Third approach, possible variation on the above two approaches
  – maintain a separate Bitmap
Separate Bitmap

- StaticThread has a Bitmap instance var
- Initialize in constructor

```java
public StaticThread(SurfaceHolder sh) {
    surfaceHolder = sh;
    random = new Random();
    paint = new Paint();
    paint.setStyle(Paint.Style.STROKE);
    paint.setStrokeWidth(10);
    image = Bitmap.createBitmap(getWidth(), getHeight(), Bitmap.Config.ARGB_8888);
}"
```
Updates to Bitmap

```java
@Override
public void run() {
    while (running) {
        if (surfaceHolder.getSurface().isValid()) {
            handleFrameRateChecks();

            Canvas c = new Canvas(image);
            doDraw(c);

            Canvas canvas = surfaceHolder.lockCanvas();
            canvas.drawBitmap(image, new Matrix(), pt);
            surfaceHolder.unlockCanvasAndPost(canvas);
        }
    }
}
```
Demo Alt Version of run()

- Flicker and jitter?
- Also possible to save Bitmap to file for later use
Animations

• Frame based vs. Time based
• Frame based:
  – update every frame
  – simple, but difference in frame rates
• Time based
  – update every frame but based on time elapsed since last frame
  – more work, more accurate
  – sdk example lunar lander
Checking Frame Rate

- From StaticView
- Emulator 6-7 fps, dev phone 40 -45 fps

```java
private void handleFrameRateChecks() {
    long currTime = System.currentTimeMillis();
    long diff = currTime - prevTime;
    prevTime = currTime;
    // Log.d("Static", "time diff: " + diff);

    if(frameCount < 30) {
        frameCount++;
    } else {
        frameCount = 0;
        long timeDiff = currTime - startTime;
        startTime = currTime;
        double frameRate = 1000.0 / (timeDiff / 30.0);
        Log.d("Static", "frame rate: " + (int)frameRate);
        Log.d("Static", "timediff: " + timeDiff);
    }
}
```
Controlling Frame Rate

• Sleep after completing work in loop of run

• More complex than shown, use previous time and current time

```java
try{
    Thread.sleep(1000);
}
catch(Exception e) {}"