CS193P - Lecture 5

iPhone Application Development

Views
Drawing
Animation

Announcements お知らせ（省略）

- Assignment 1 grades are out. Contact Paul or Dave if you didn't get yours 成績

- Contact Paul or Dave if you need a loaner iPod Touch 貸出用

- Assignments 2A and 2B due Wednesday, 1/20

Questions from Monday?

- Model, View, Controller
- Interface Builder & Nibs
- Delegate　デリゲート：別オブジェクトの代理として振る舞わせること
  - Allows one object to act on behalf of another object
- Target-Action

Today's Topics

- Views — View（UIView）
- Drawing — 絵を描く
- Text & Images — テキストと画像
- Animation — アニメーション

Views （UIView のインスタンスを意味する）

View Fundamentals UIView の基本

- Rectangular area on screen スクリーン上の矩形（長方形）領域
- Draws content コンテンツを描く
- Handles events イベントを処理する
- Subclass of UIResponder (event handling class)
  UIView は UIResponder のサブクラス
- Views arranged hierarchically
  - every view has one superview
  - every view has zero or more subviews

階層的にアレンジされる
全ての View は
親 View を１つだけもつ
0個以上の子 View をもつ

（最上位の View は UIWindow であり，これだけは親をもたない）

View の階層　　UIWindow

View Hierarchy - UIWindow

- Views live inside of a window　View は windows の中に住んでいる
- UIWindow is actually just a view　UIWindow は View の一種
  - adds some additional functionality specific to top level view
    最上位 View に特化した追加機能をもつ
- One UIWindow for an iPhone app　１つのアプリに１つの UIWindow
  - Contains the entire view hierarchy　UIWindow の中に View 階層の全てが入る
  - Set up by default in Xcode template project　Xcode のテンプレートで このように設定される

（View の）操作

View Hierarchy - Manipulation

IB ならば
ドラッグ&ドロップ

Subview を親に入れる/親から外す

- Add/remove views in IB or using UIView methods

```
- (void)addSubview:(UIView *)view;
- (void)removeFromSuperview;
```

View の階層をマニュアル操作

- Manipulate the view hierarchy manually:

```
- (void)insertSubview:(UIView *)view atIndex:(int)index;
- (void)insertSubview:(UIView *)view belowSubview:(UIView *)view;
- (void)insertSubview:(UIView *)view aboveSubview:(UIView *)view;
- (void)exchangeSubviewAtIndex:(int)index
        withSubviewAtIndex:(int)otherIndex;
```

ある View の直下にある子 View は複数あってよく，
奥から前への順序で「配列」に入れられている．

View Hierarchy - Ownership

所有権

- Superviews retain their subviews　親 View は子 View を retain

子 View が 親 View のみから retain されることも多い

- Not uncommon for views to only be retained by superview
  - Be careful when removing!
  - Retain subview before removing if you want to reuse it
    子 View を再利用したかったら自分で retain すること

- Views can be temporarily hidden

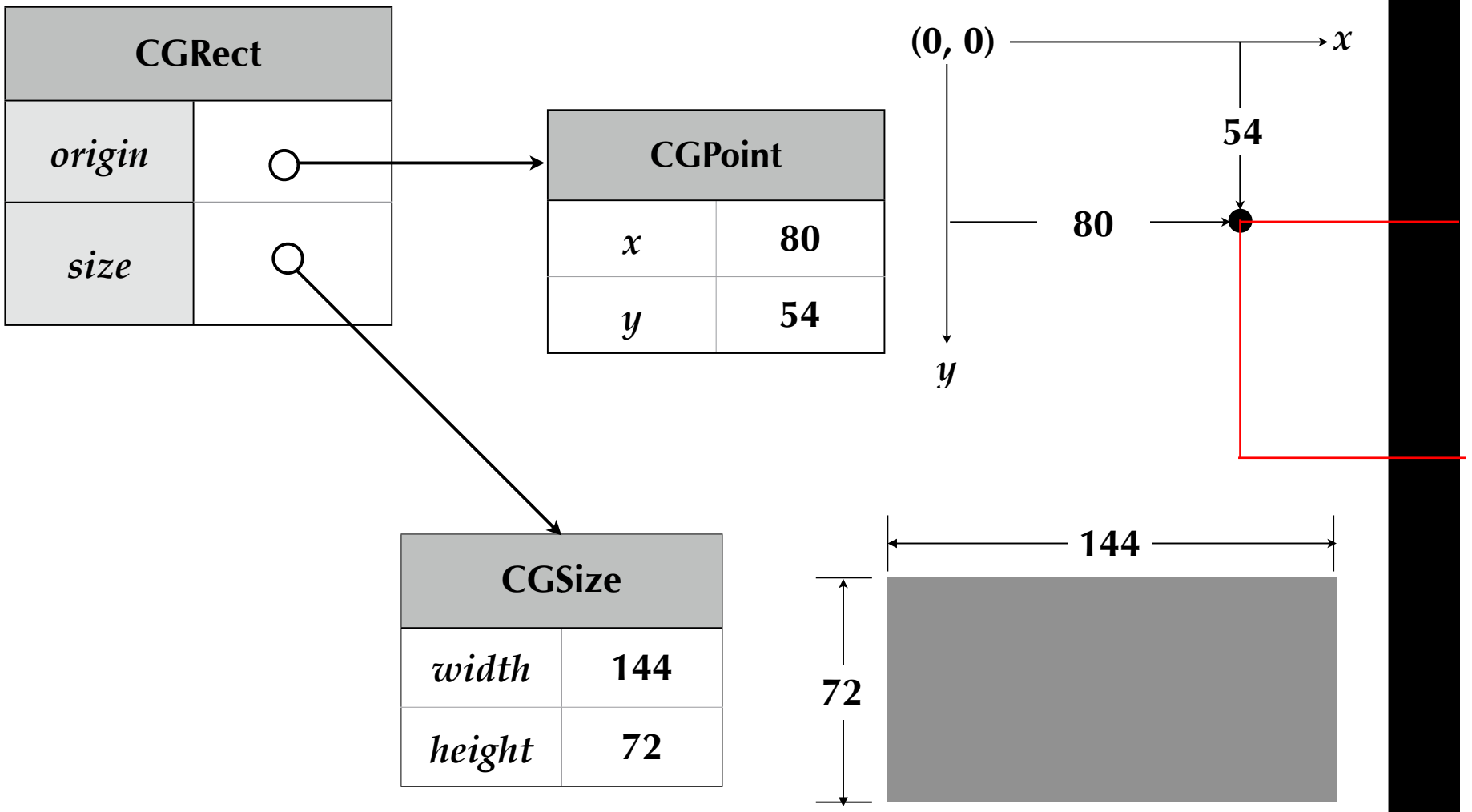
```
theView.hidden = YES;
```

一時的に View を隠すには
こうすればよい

View-related Structures

- CGPoint
  - location in space: { x , y }　位置（点）
- CGSize
  - dimensions: { width , height }　寸法（サイズ：幅と高さ）
- CGRect
  - location and dimension: { origin , size }　長方形（位置と寸法）

Rects, Points and Sizes



View-related Structure

Creation Function 生成関数	Example いずれも構造体データを返す（たとえば {100.0, 200.0}）

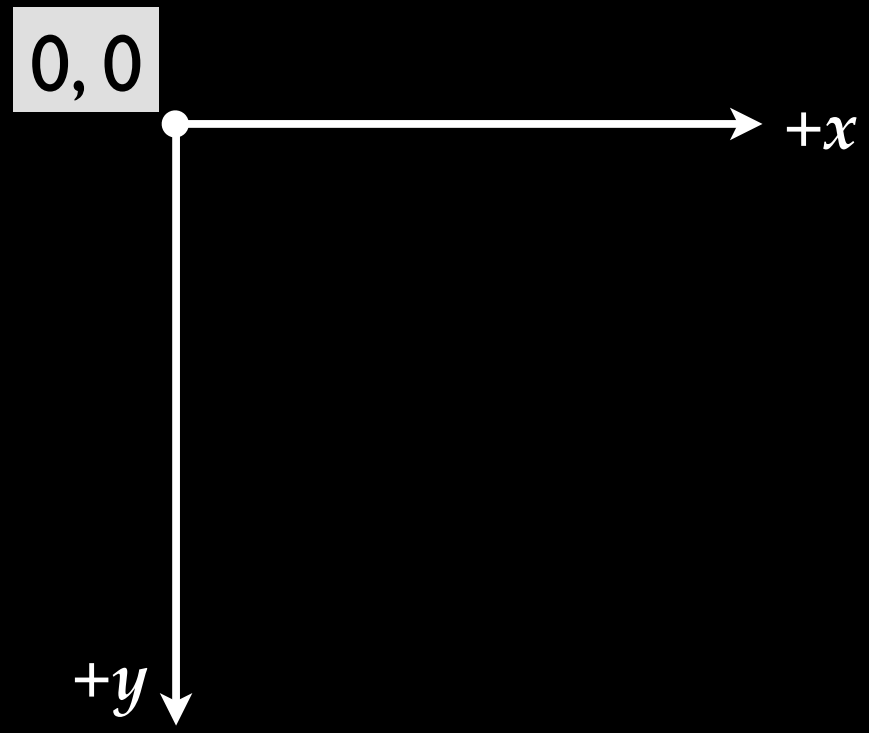
`CGPointMake (x, y)`	`CGPoint point = CGPointMake (100.0, 200.0);`
	`point.x = 300.0;`
	`point.y = 30.0;`

`CGSizeMake (width, height)`	`CGSize size = CGSizeMake (42.0, 11.0);`
	`size.width = 100.0;`
	`size.height = 72.0;`

`CGRectMake (x, y,`
`width, height)`
	`CGRect rect = CGRectMake (100.0, 200.0,`
	`42.0, 11.0);`
	`rect.origin.x = 0.0;`
	`rect.size.width = 50.0;`

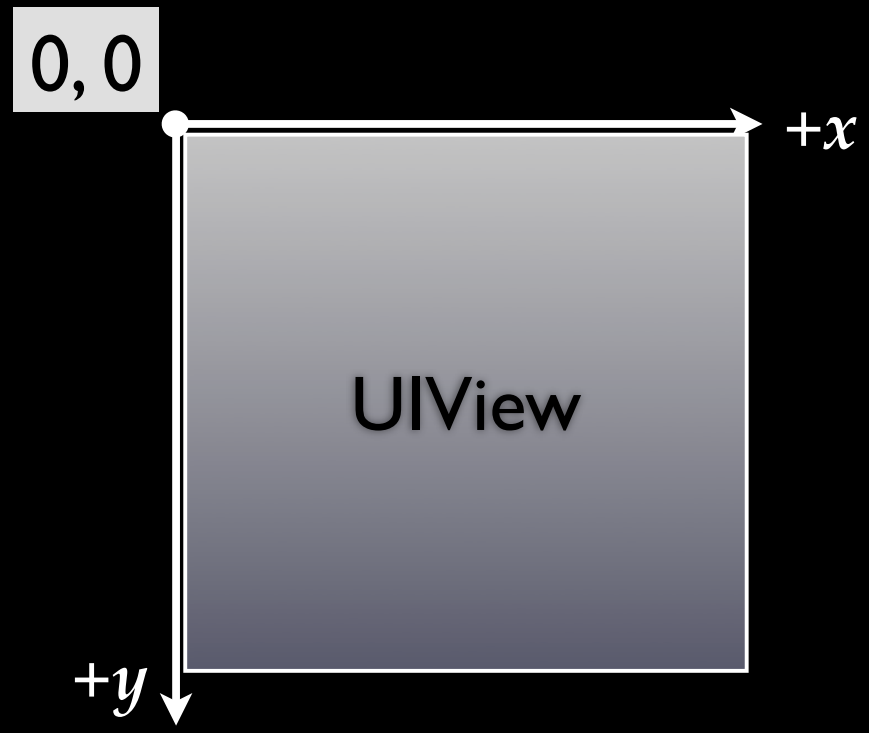
UIView Coordinate System UIView の座標系

- Origin in upper left corner 原点は「左上」
- y axis grows downwards y軸は下にのびる



UIView Coordinate System UIView の座標系

- Origin in upper left corner 原点は「左上」
- y axis grows downwards y軸は下にのびる



Location and Size 位置とサイズ

- View's location and size expressed in two ways
  - Frame is in superview's coordinate system
  - Bounds is in local coordinate system

Frame：親 View の座標系

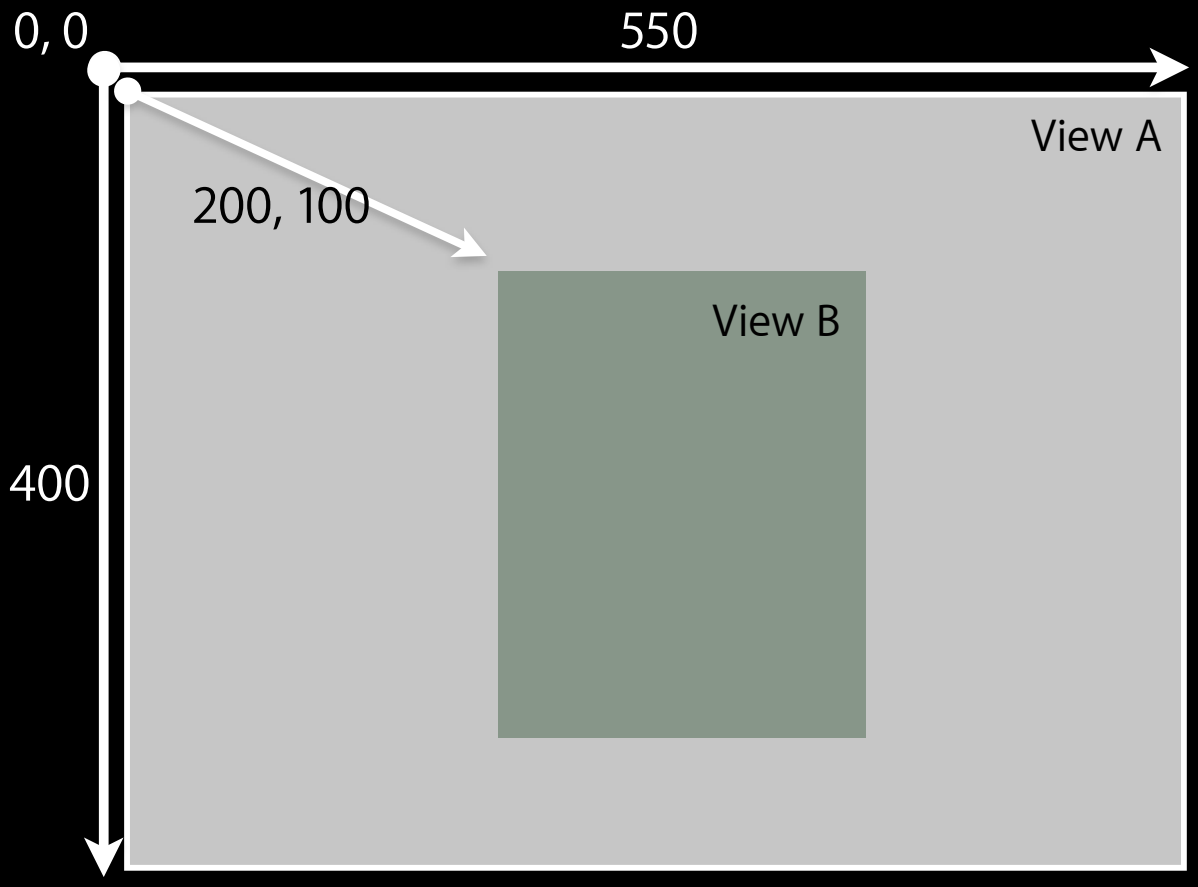
Bounds：自分の座標系

View A frame:
origin: 0, 0
size: 550 x 400

View A bounds:
origin: 0, 0
size: 550 x 400

Location and Size 位置とサイズ

- View's location and size expressed in two ways
  - Frame is in superview's coordinate system **Frame：親 View の座標系**
  - Bounds is in local coordinate system **Bounds：自分の座標系**

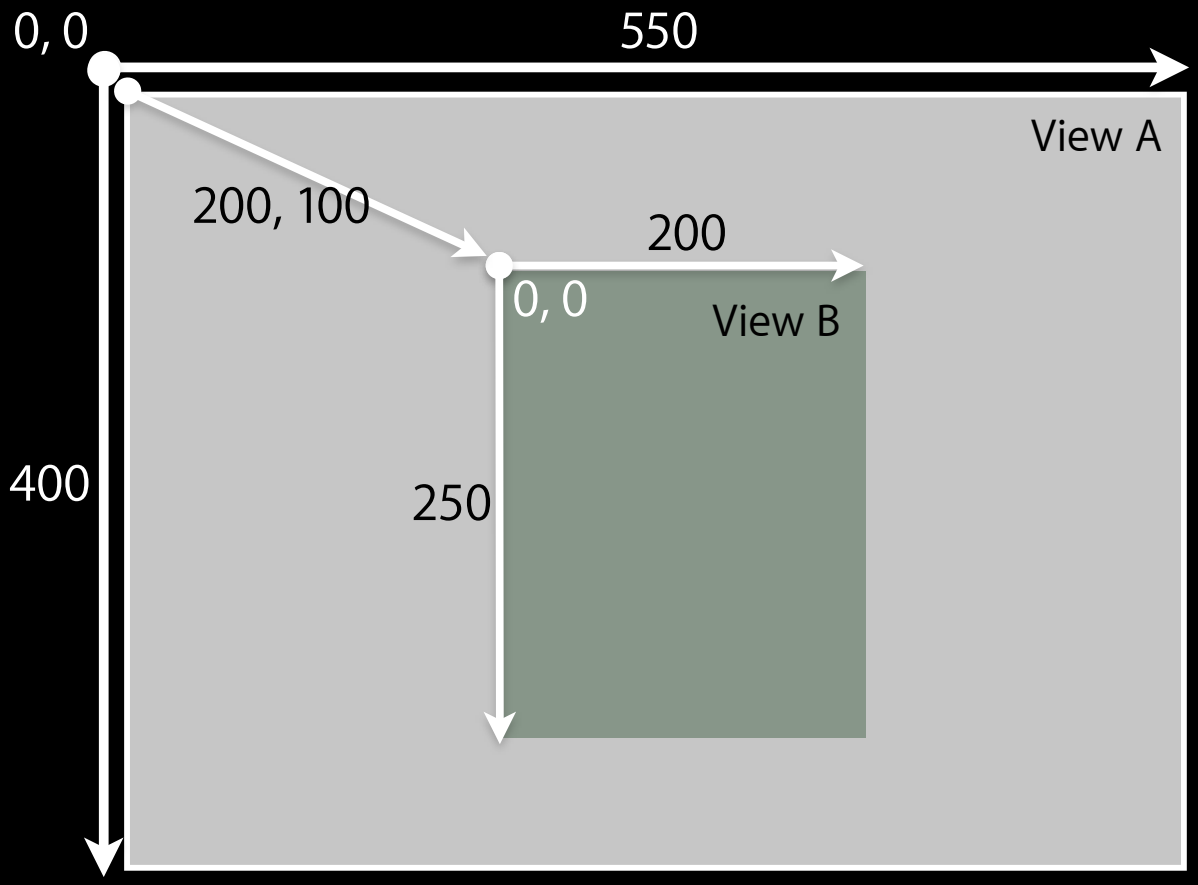


View A frame:
origin: 0, 0
size: 550 x 400

View A bounds:
origin: 0, 0
size: 550 x 400

Location and Size 位置とサイズ

- View's location and size expressed in two ways
  - Frame is in superview's coordinate system **Frame：親 View の座標系**
  - Bounds is in local coordinate system **Bounds：自分の座標系**



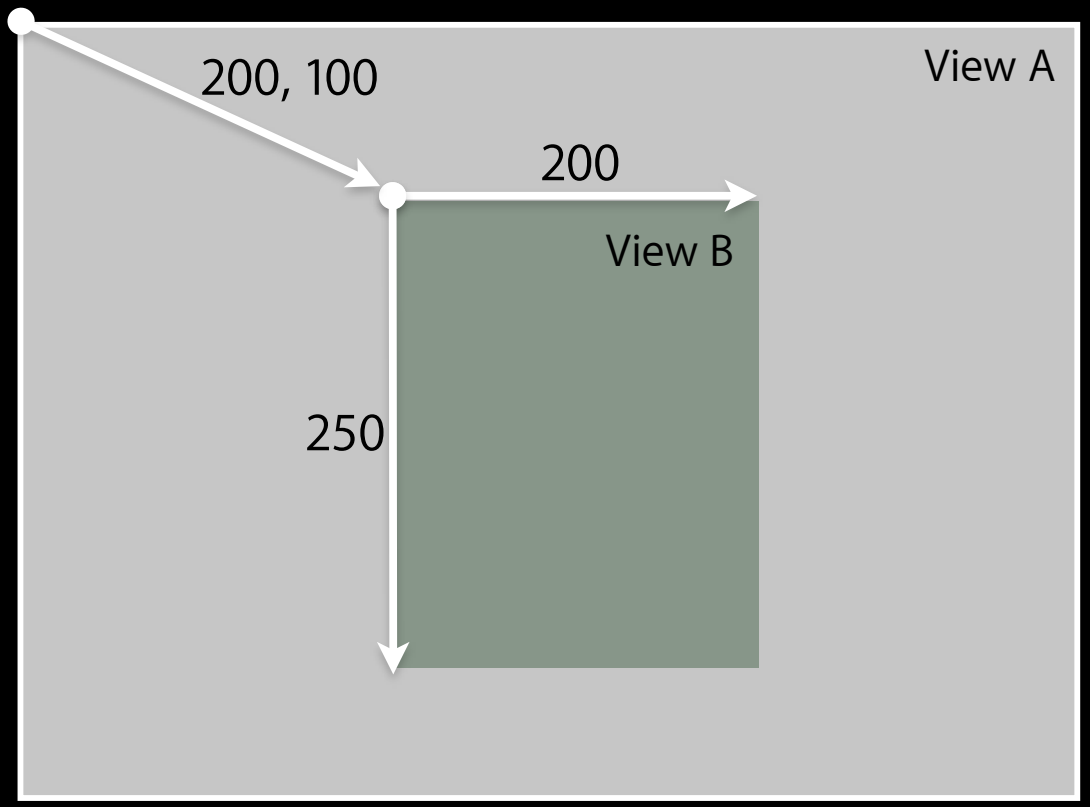
View A frame:
origin: 0, 0
size: 550 x 400

View A bounds:
origin: 0, 0
size: 550 x 400

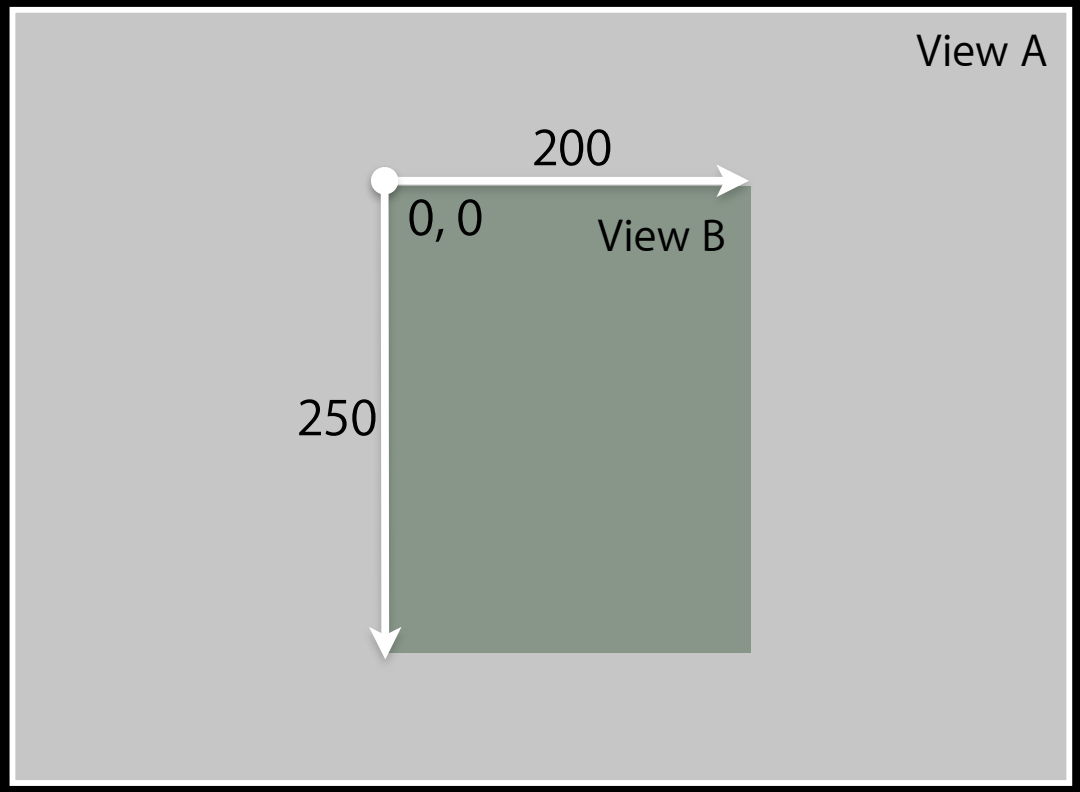
View B frame:
origin: 200, 100
size: 200 x 250

View B bounds:
origin: 0, 0
size: 200 x 250

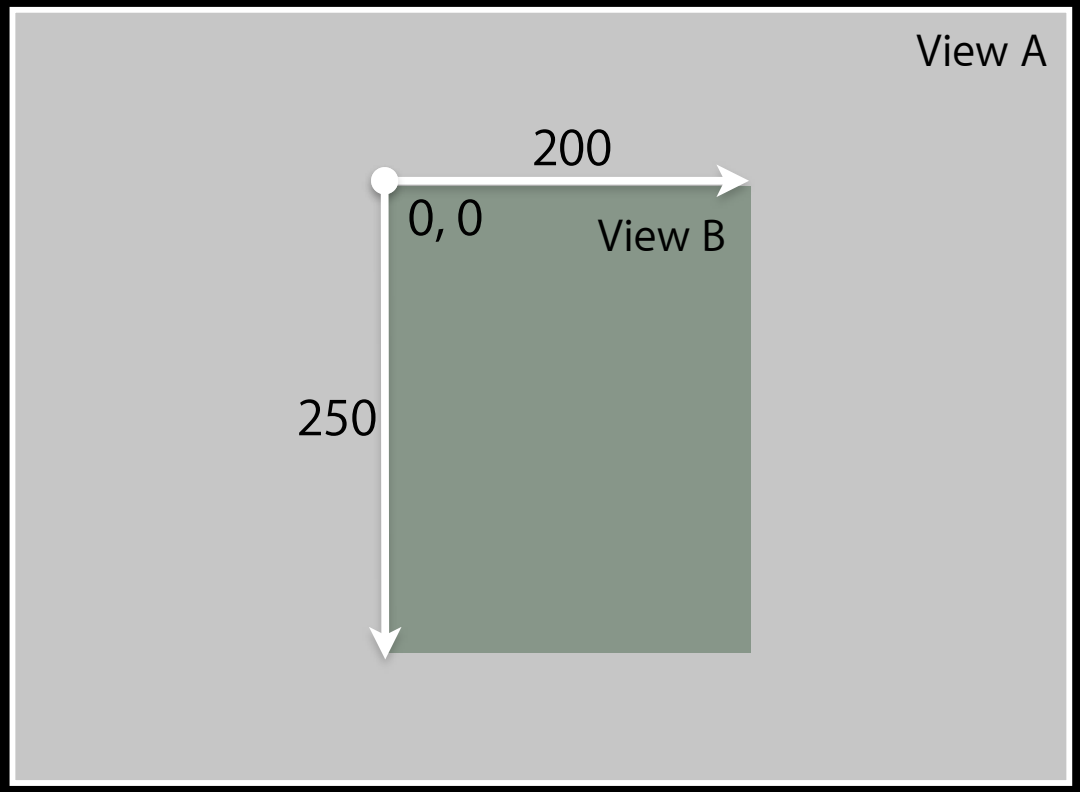
Frame is Computed Frame は（その場で）計算される



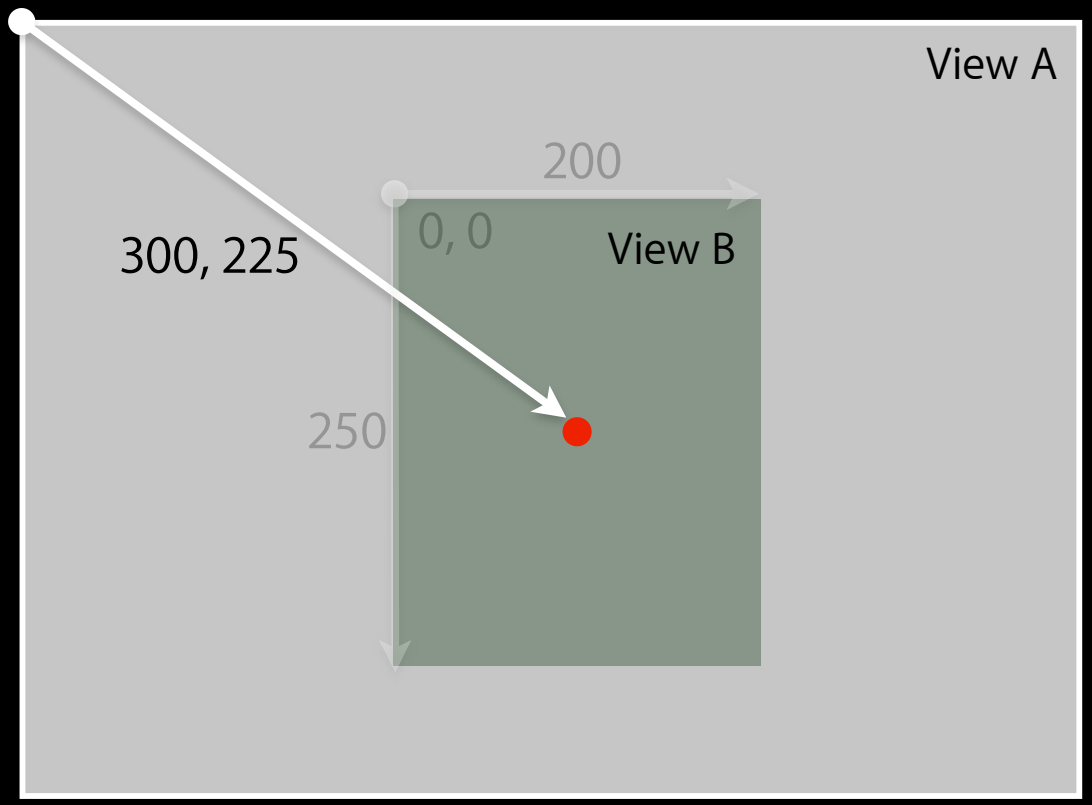
Bounds View B が保持する内部変数のひとつ



Center View B が保持する内部変数のひとつ

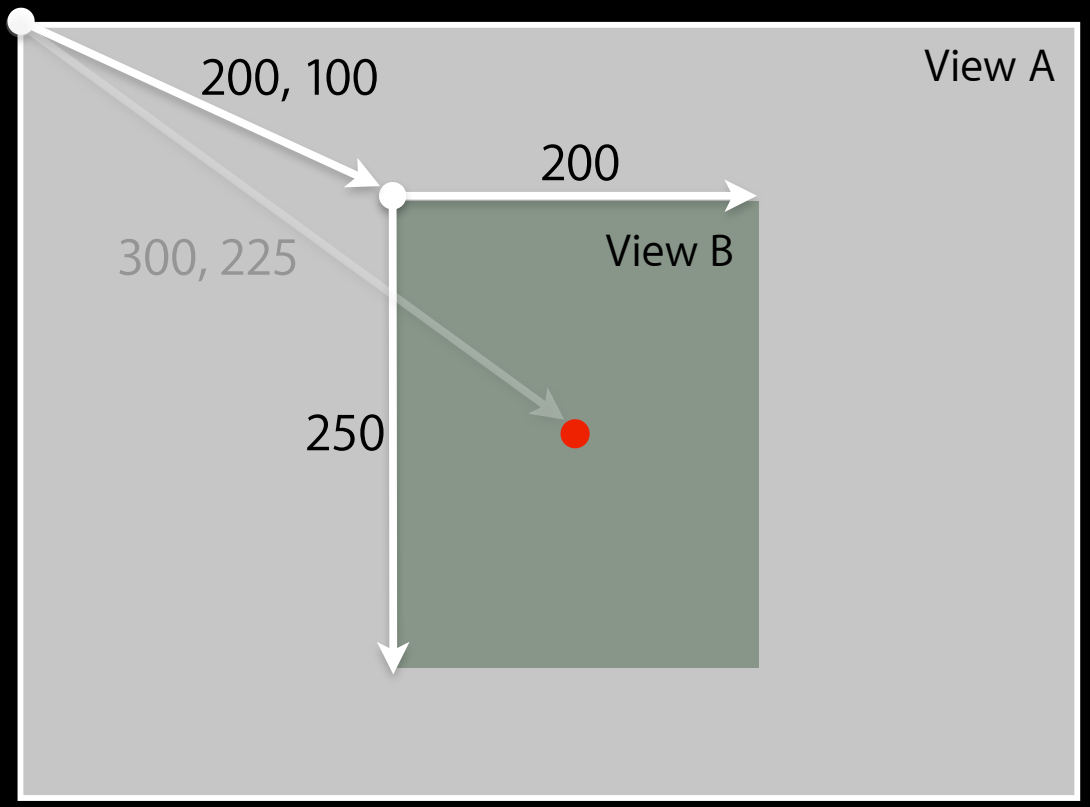


Center View B が保持する内部変数のひとつ



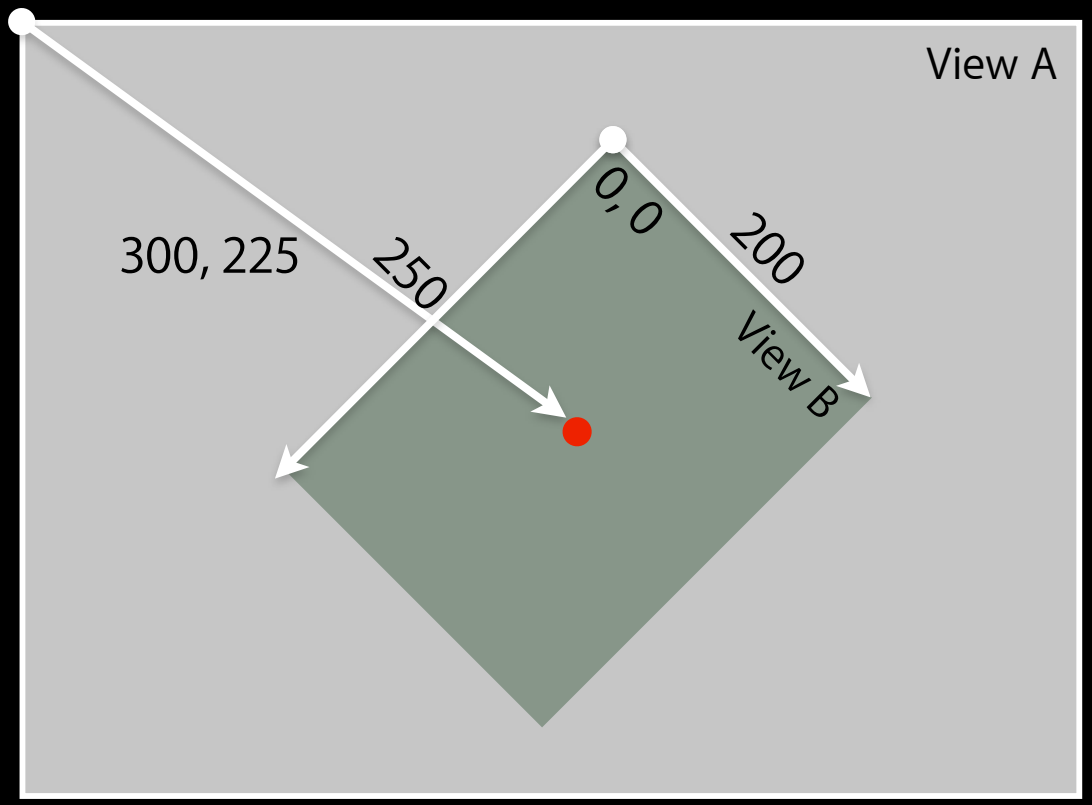
Frame

Frame は計算される



Transform アフィン変換（回転，平行移動，拡大縮小）

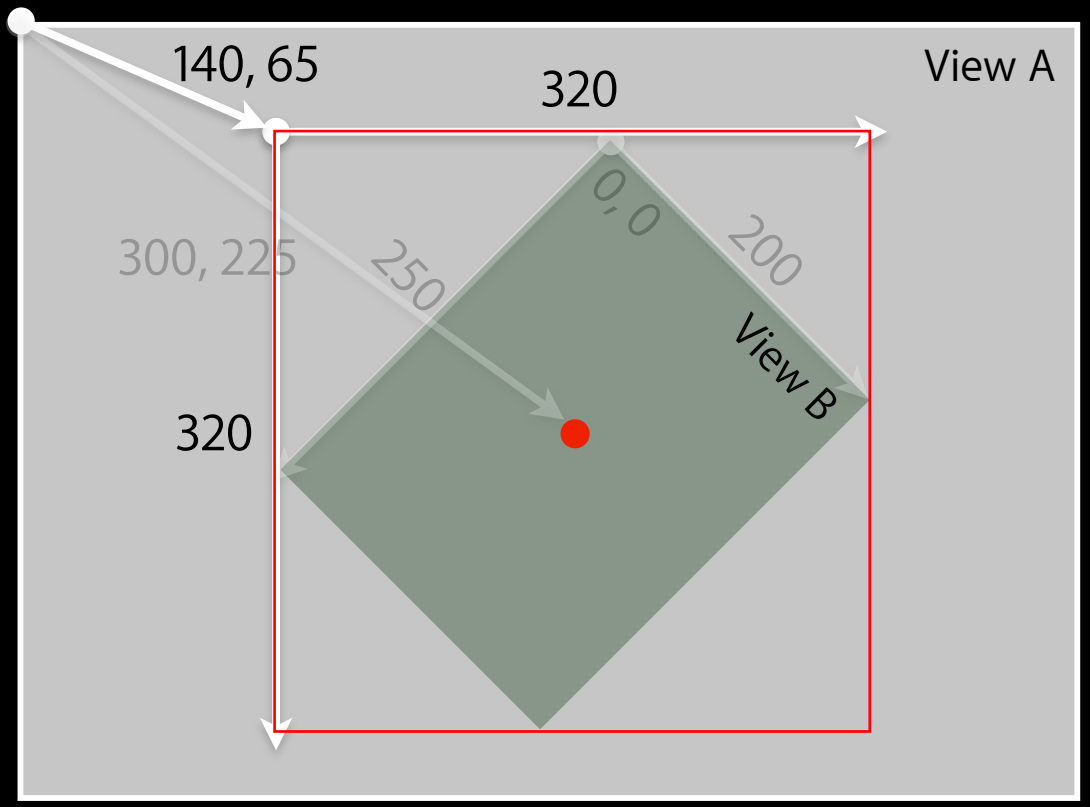
- 45° Rotation **45度 回転**



Frame

回転した View B の Frame はこのように計算される

- The smallest rectangle in the superview's coordinate system that fully encompasses the view itself
  View A 座標系の長方形で，View B を覆う最小のもの



Frame and Bounds

- Which to use? Frame と Bounds，どちらを使う？
  - Usually depends on the context 文脈しだい…
- If you are *using* a view, typically you use frame
- If you are *implementing* a view, typically you use bounds
- Matter of perspective 見かた次第
  - From outside it's usually the frame 外から
  - From inside it's usually the bounds 内から

（外から）View を操作するなら Frame

View（の中身）を実装するなら Bounds（例. タッチ応答）

例

- Examples: View の生成・位置決め --- Frame を使う
  - Creating a view, positioning a view in superview - use frame
  - Handling events, drawing a view - use bounds

イベント処理，View（内部）の描画 --- Bounds を使う

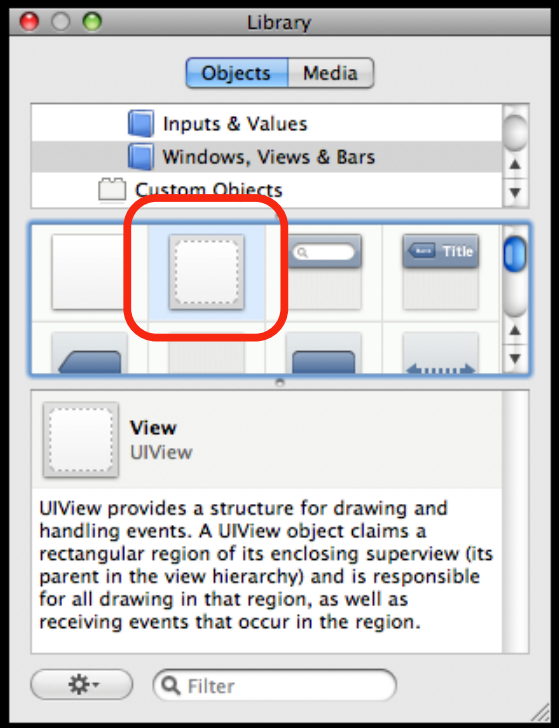
Creating Views View の生成

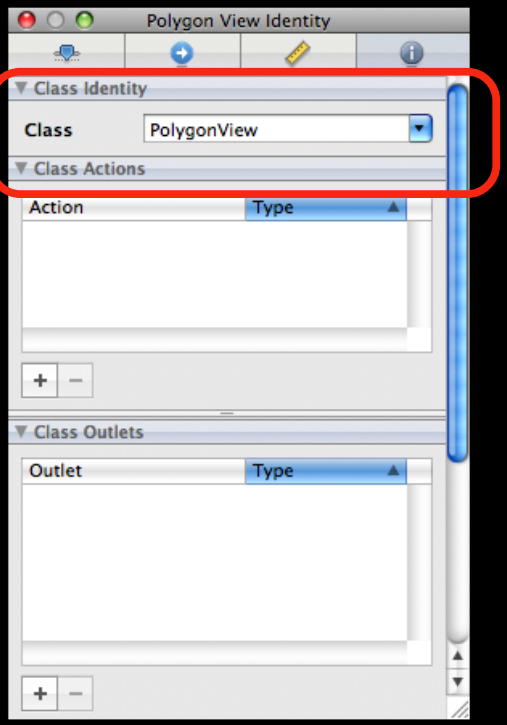
View はどこから来るのか？

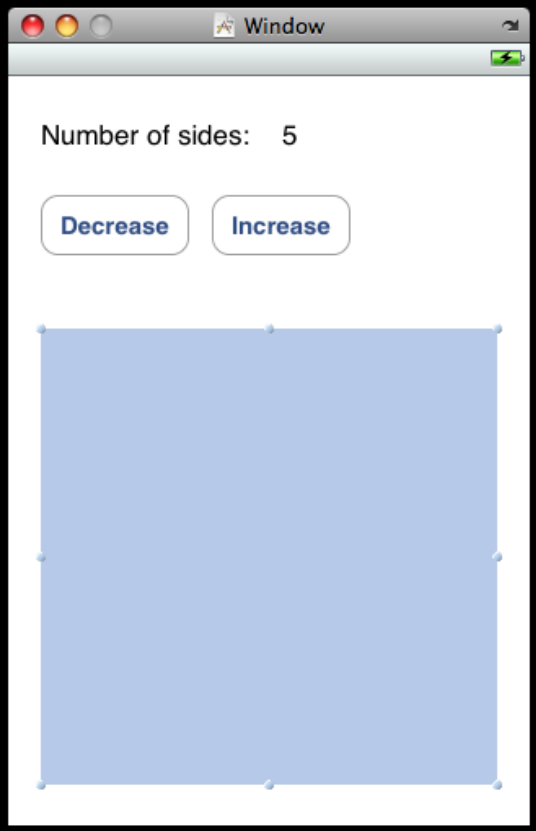
Where do views come from?

- Commonly Interface Builder 通常は IB から
- Drag out any of the existing view objects (buttons, labels, etc)
- Or drag generic UIView and set custom class

普通の UIView ＞ カスタムクラスを設定







Manual Creation （コードで）マニュアル生成

- Views are initialized using -initWithFrame:

```
CGRect frame = CGRectMake(0, 0, 200, 150);
UIView *myView = [[UIView alloc] initWithFrame:frame];
```

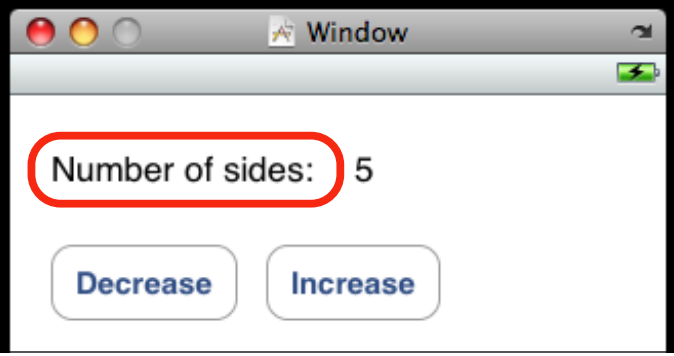
- Example:

```
CGRect frame = CGRectMake(20, 45, 140, 21);
UILabel *label = [[UILabel alloc] initWithFrame:frame];

[window addSubview:label];
[label setText:@"Number of sides:"];
[label release];  // label now retained by window
```

親 View（window）に登録

この release により，window だけが label を retain することになる



Defining Custom Views カスタム View をつくる

- Subclass UIView

- For custom drawing, you override: カスタム描画するには，drawRect を上書きする

```
- (void)drawRect:(CGRect)rect;
```

- For event handling, you override:

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event;
```

UIView が受け取るイベントは，これら４つ．
これらを上書きすることで， View にタッチ応答をさせる．

Drawing Views

View の描画メソッド drawRect:rect

- (void)drawRect:(CGRect)rect

- -[UIView drawRect:] does nothing by default
  デフォルトでは何も描画しない
  - If not overridden, then backgroundColor is used to fill
    上書きされないかぎり，「背景色」で（全面を）塗りつぶすだけ
- Override - drawRect: to draw a custom view 上書きして「描画」！
  - rect argument is area to draw
    引数 rect は描画領域を表わす
- When is it OK to call drawRect:?
  じゃあ，いつ drawRect を呼び出せばよいか？

Be Lazy だらだら行こう

- drawRect: is invoked automatically drawRect は自動的に実行される
  - Don't call it directly! 直接呼び出さない！
- Being lazy is good for performance 「だらだら」するのは パフォーマンスを上げるのによい
- When a view needs to be redrawn, use: 描画が必要になったら 「描画予約」を入れる (次のイベントループで描画)

```
- (void)setNeedsDisplay;
```

- For example, in your controller: たとえば controller の中で…

```
- (void)setNumberOfSides:(int)sides {
    numberOfSides = sides;
    [polygonView setNeedsDisplay];
}
```

「辺の数」を変更したら
「描画予約」する
カスタム UIView

CoreGraphics and Quartz 2D

= 描画のための フレームワーク

- UIKit offers very basic drawing functionality

```
UIRectFill(CGRect rect);
UIRectFrame(CGRect rect);
```

UIKit の描画メソッドは，四角枠描き・四角塗りのみ．

- CoreGraphics: Drawing APIs
- CG is a C-based API, not Objective-C

複雑な描画は CoreGraphics で（Obj-C ではなく普通の C）

- CG and Quartz 2D drawing engine define simple but powerful graphics primitives — シンプルかつ強力な描画要素を提供する
  - Graphics context — 色・線種などが設定された「描画の場」
  - Transformations — 変換（回転・平行移動・拡大縮小など）
  - Paths — パス（直線・折れ線・カーブなど）
  - Colors — 色情報（RGB/HSVなど）
  - Fonts — Helvetica Bold Italic 24pt など
  - Painting operations — アルファ（透明〜不透明）など

Graphics Contexts

コンテクスト＝文脈・状況
色や線種が設定された「描画の場」

- All drawing is done into an opaque graphics context

  すべての描画は「描画の場」に対して行われる．

- Draws to screen, bitmap buffer, printer, PDF, etc.

  画面だけでなく，メモリ領域・プリンタ・PDFファイルにも描画可．

- Graphics context setup automatically before invoking drawRect:
  - Defines current path, line width, transform, etc.
  - Access the graphics context within drawRect: by calling

    ```
    (CGContextRef)UIGraphicsGetCurrentContext(void);
    ```

  - Use CG calls to change settings

    あとは CG〜関数で設定変更！

drawRect の中で
Context を取り出すには…

- Context only valid for current call to drawRect:
  - Do not cache a CGContext!

Context は drawRect 呼び出しの
中だけで有効（保存しても再利用不可）

CG Wrappers

いくつかの CG〜関数は，
UIKit をとおして利用できる．

- Some CG functionality wrapped by UIKit
- UIColor
  - Convenience for common colors
  - Easily set the fill and/or stroke colors when drawing

一般的な色名による指定や
RGB/HSV による色指定が可能

```
UIColor *redColor = [UIColor redColor];
[redColor set];
// drawing will be done in red
```

- UIFont
  - Access system font　+systemFontOfSize:
  - Get font by name　+fontWithName:size:

```
UIFont *font = [UIFont systemFontOfSize:14.0];
[myLabel setFont:font];
```

UIColor/UIFont はオブジェクトを返す（よってポインタで受け取る）

Simple drawRect: example

- Draw a solid color and shape　色のついたベタな形状を描画

```
- (void)drawRect:(CGRect)rect {
    CGRect bounds = [self bounds];

    [[UIColor grayColor] set];
    UIRectFill (bounds);

    CGRect square = CGRectMake (10, 10, 50, 100);
    [[UIColor redColor] set];
    UIRectFill (square);

    [[UIColor blackColor] set];
    UIRectFrame (square);
}
```

grayColor を「描画の場」にセット
bounds 全体（全面）を塗りつぶし

redColor を「描画の場」にセット
(10,10) の位置に 50x100 の四角を
塗りつぶし

blackColor を「描画の場」にセット
同じ四角の輪郭を描く.

Simple drawRect: example

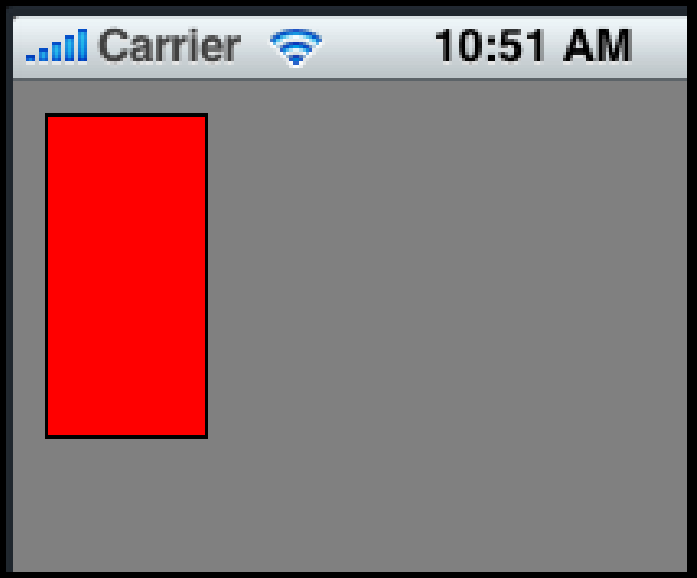
- Draw a solid color and shape

```
- (void)drawRect:(CGRect)rect {
    CGRect bounds = [self bounds];

    [[UIColor grayColor] set];
    UIRectFill (bounds);

    CGRect square = CGRectMake (10, 10, 50, 100);
    [[UIColor redColor] set];
    UIRectFill (square);

    [[UIColor blackColor] set];
    UIRectFrame (square);
}
```



より複雑な形状の描画

Drawing More Complex Shapes

- Common steps for drawRect: are
  - Get current graphics context
  - Define a path
  - Set a color
  - Stroke or fill path
  - Repeat, if necessary

一般的な手順はつぎのとおり

- 「描画の場」をゲット
- パスを決める
- 色を設定する
- そのパスをその色で
  輪郭描画 or 塗りつぶし
- 必要に応じて繰り返す

Paths パス（経路）

- CoreGraphics paths define shapes パスによって「形状」を決める
- Made up of lines, arcs, curves and rectangles 直線・円弧・曲線・四角
- Creation and drawing of paths are two distinct operations
  - Define path first, then draw it

描画するには
１．パスを定義
２．それを描画

CGPath パスをつくる２つの方法

- Two parallel sets of functions for using paths
  - CGContext “convenience” throwaway functions 使い捨てパス
  - CGPath functions for creating reusable paths 再利用可能パス

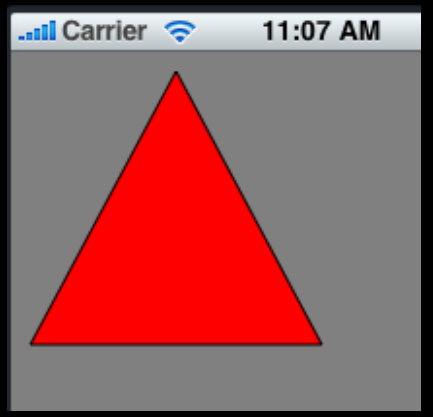
使い捨て CGContext	CGPath 再利用可	
CGContextMoveToPoint	CGPathMoveToPoint	ペンを置く
CGContextAddLineToPoint	CGPathAddLineToPoint	線を引く
CGContextAddArcToPoint	CGPathAddArcToPoint	円弧を引く
CGContextClosePath	CGPathCloseSubPath	パスを閉じる
And so on and so on . . .		

Simple Path Example

```
- (void)drawRect:(CGRect)rect {
    CGContextRef context = UIGraphicsGetCurrentContext();
                          「描画の場」をゲット
    [[UIColor grayColor] set];
    UIRectFill ([self bounds]);   背景を塗りつぶし

パス開始 CGContextBeginPath (context);
ペン置き CGContextMoveToPoint (context, 75, 10);
  線引き CGContextAddLineToPoint (context, 10, 150);
  線引き CGContextAddLineToPoint (context, 160, 150);
パス閉じ CGContextClosePath (context);

    [[UIColor redColor] setFill];      内部は赤
    [[UIColor blackColor] setStroke];  輪郭は黒
    CGContextDrawPath (context, kCGPathFillStroke);
}       パスを描画                  （内部と輪郭の描画）
```



内部のみ　kCGPathFill
輪郭のみ　kCGPathStroke

More Drawing Information より詳しくは…

- UIView Class Reference
- CGContext Reference
- “Quartz 2D Programming Guide”
- Lots of samples in the iPhone Dev Center

開いたパスについて
Stroke すれば開いた折れ線を描画
Fill すれば強制的にパスを閉じて塗りつぶし

Images & Text 画像とテキスト

UIImage ビットマップ画像

- UIKit class representing an image 画像表示のためのUI 要素

- Creating UIImages: 画像はどこから持って来るか
  - Fetching image in application bundle Resource に入れたファイルから
    - Use +[UIImage imageNamed:(NSString *)name]
    - Include file extension in file name, e.g. @"myImage.jpg"
  - Read from file on disk ディスクファイルから
    - Use -[UIImage initWithContentsOfFile:(NSString *)path]
  - From data in memory メモリから（NSData* で渡す）
    - Use -[UIImage initWithData:(NSData *)data]

「描画の場」から（自分で描くことで）画像をつくる

Creating Images from a Context

- Need to dynamically generate a bitmap image
  その場でビットマップ画像を「生成」するときに使う
- Same as drawing a view
  View に描画するのと全く同じやり方
- General steps その一般的な手順は…
  - Create a special CGGraphicsContext with a size
  - Draw
  - Capture the context as a bitmap
  - Clean up

サイズ付きの「描画の場」を生成

パス定義＋描画

「描画の場」をビットマップとして「キャプチャ」

あとかたづけ

Bitmap Image Example

```
- (UIImage *)polygonImageOfSize:(CGSize)size {
    UIImage *result = nil;
    指定サイズの新しい「描画の場」を生成
    UIGraphicsBeginImageContext (size);

    // call your drawing code...
    「描画の場」に描画する
    （詳細は省略）

    result = UIGraphicsGetImageFromCurrentContext();
    ビットマップとして画像をゲット
    UIGraphicsEndImageContext();
    「描画の場」を廃棄
    return result;
}
    UIImage を返す（autorelease オブジェクト）
```

Getting Image Data

UIImage（ビットマップ）を
JPG/PNG に変換

- Given UIImage, want PNG or JPG representation

```
NSData *UIImagePNGRepresentation (UIImage * image);
NSData *UIImageJPGRepresentation (UIImage * image);
```

これでディスクに保存などができる.

- UIImage also has a CGImage property which will give you a CGImageRef to use with CG calls

Drawing Text & Images

- You can draw UIImages in -drawRect:

  UIImage（ビットマップ）を表示するには…

  - `[UIImage drawAtPoint:(CGPoint)point]` — 指定位置に元サイズで
  - `[UIImage drawInRect:(CGRect)rect]` — 指定位置・大きさに（伸び縮みする）
  - `[UIImage drawAsPatternInRect:(CGRect)rect]` — タイル状に敷きつめ（元サイズで）

- You can draw NSString in -drawRect:

  文字列を -drawRect の中で描画するには…

  - `[NSString drawAtPoint:(CGPoint)point withFont:(UIFont *)font]`

    位置とフォントを指定して描画

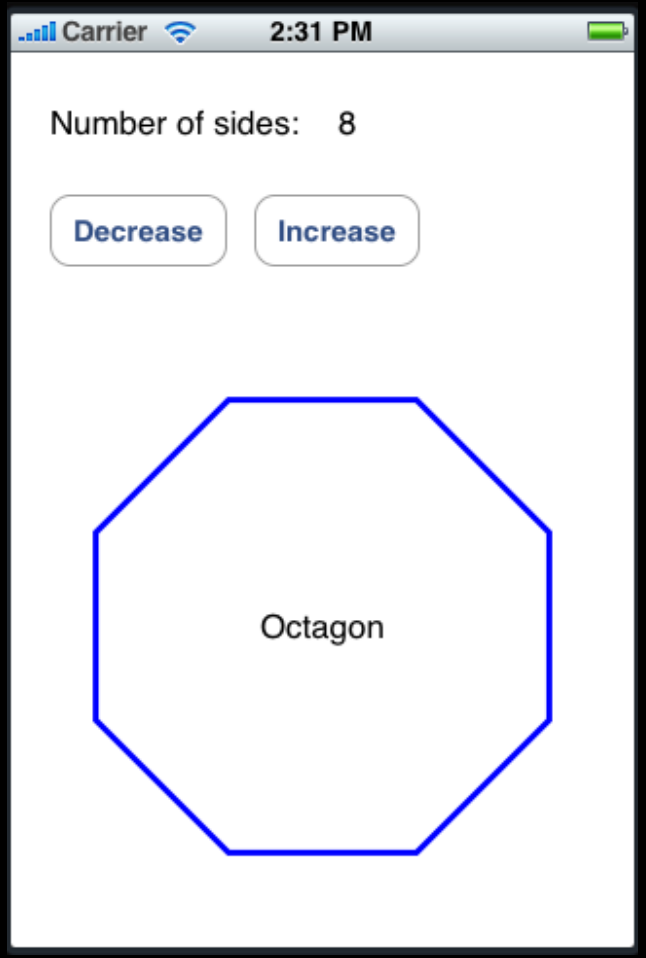
But there is a better way!

でも，もっとよい方法があるよ！

Text, Images, and UIKit views

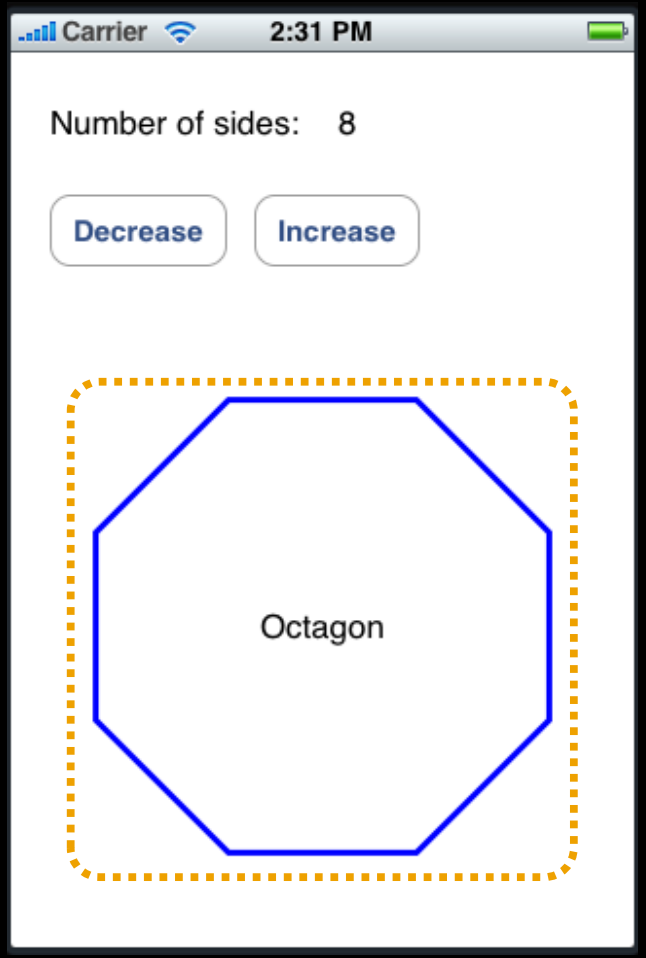
Constructing Views

- How do I implement this?
  これをどうやって実現するか？



Constructing Views

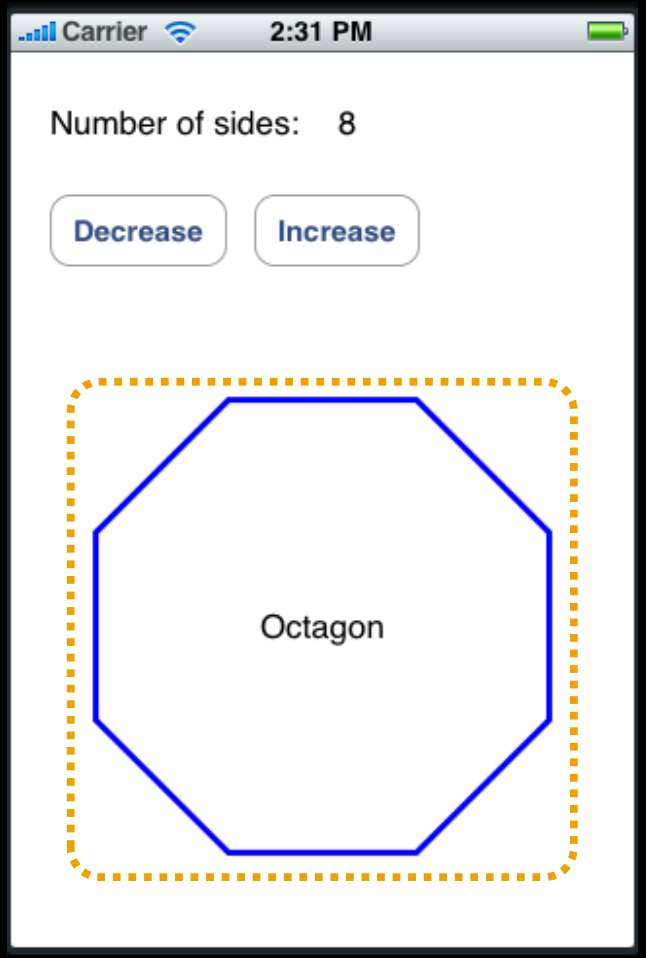
- How do I implement this?
  これをどうやって実現するか？
- Goal
  - PolygonView that displays shape as well as name
    多角形と名前を表示する PolygonView



Constructing Views

- How do I implement this?
  これをどうやって実現するか？
- Goal
  - PolygonView that displays shape as well as name
    多角形と名前を表示する PolygonView
- Initial thought　まず思いつく方法
  - Have PolygonView draw the text
  - Inefficient when animating

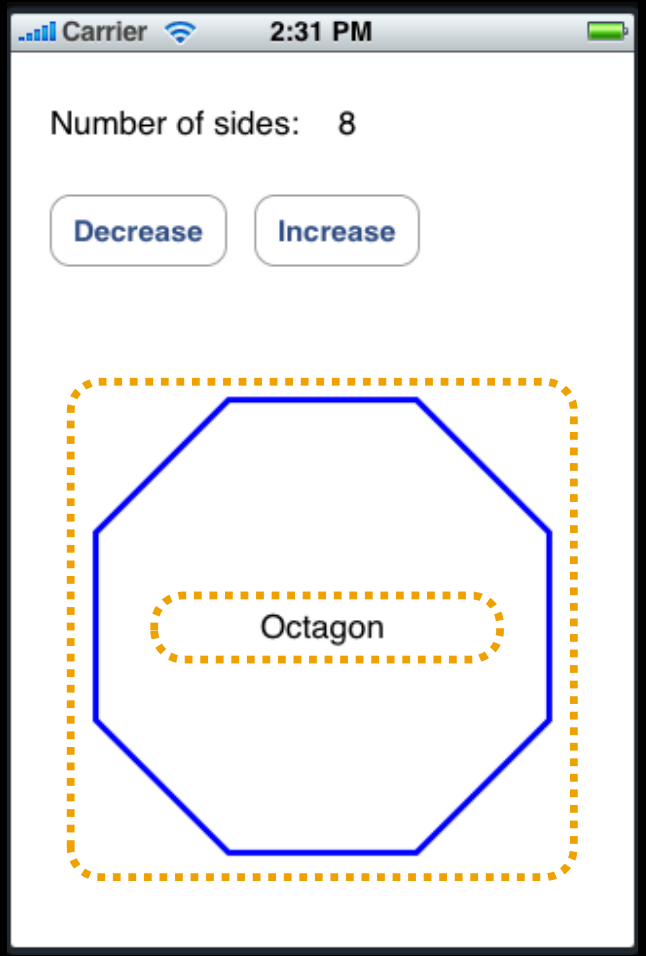
  PolygonView の中でのテキスト描画は
  アニメーションなどを考えると非効率



Constructing Views

- How do I implement this?
  これをどうやって実現するか？
- Goal
  - PolygonView that displays shape as well as name
    多角形と名前を表示する PolygonView
- Initial thought　まず思いつく方法
  - Have PolygonView draw the text
  - Inefficient when animating

  PolygonView の中でのテキスト描画は
  アニメーションなどを考えると非効率
- Instead use UILabel!
  - Tastes great　代わりに UILabel を使う
  - Less filling　軽い（楽に使える）



UILabel

- UIView subclass that knows how to draw text
  テキスト描画を専門とする UIView のサブクラス
- Properties include: UILabel の属性
  - font
  - textColor
  - shadow (offset & color) 影
  - textAlignment 割り付け（右詰めなど）

UIImageView

- UIView that draws UIImages

UIImage（ビットマップ）の描画を
専門とする UIView のサブクラス

- Properties include:
  - image
  - animatedImages
  - animatedDuration
  - animatedRepeatCount

アニメGIFと類似

画像の配列を与える（パラパラ漫画）

パラパラの時間間隔

パラパラ漫画の繰り返し回数

- contentMode property to align and scale image wrt bounds

wrt = with regard to
「〜に関して」

contentMode 属性によって，
画像をどのようにスケールするかを指定できる．
（例. Center, Aspect Fill, Scale To Fill, …）

UIControl

「コントロール」
ユーザからのアクションを受け付ける UI 要素

- UIView with Target-Action event handling

  イベント（タッチなど）を受け付ける UIView のサブクラス

  UIButton, UISlider, UITextField, ...

- Properties include:
  - enabled　受け付け可能か
  - selected　選択されているか
  - highlighted　ハイライトされているか

共通属性

- UIButton: font, title, titleColor, image, backgroundImage
- UITextField: font, text, placeholder, textColor

  ↑うっすらと「お名前をどうぞ」みたいに表示

- See UIKit headers for plenty more

  詳しくはヘッダファイルを見てね

View Properties & Animation

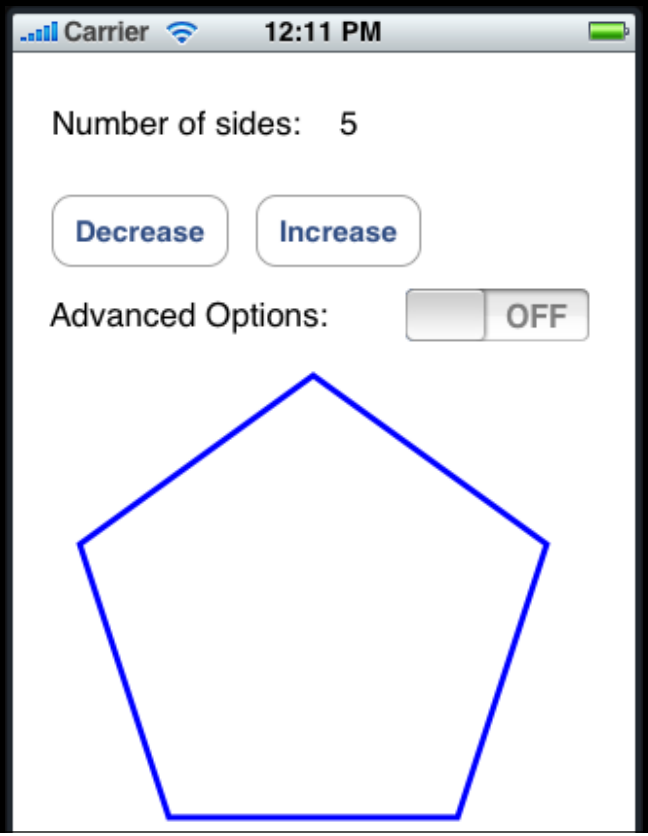
View の属性と「アニメーション」

Animating Views

UI 要素の配置をダイナミックに変えるには？

- What if you want to change layout dynamically?
- For example, a switch to disclose additional views...

隠し View を表に出すスイッチ…

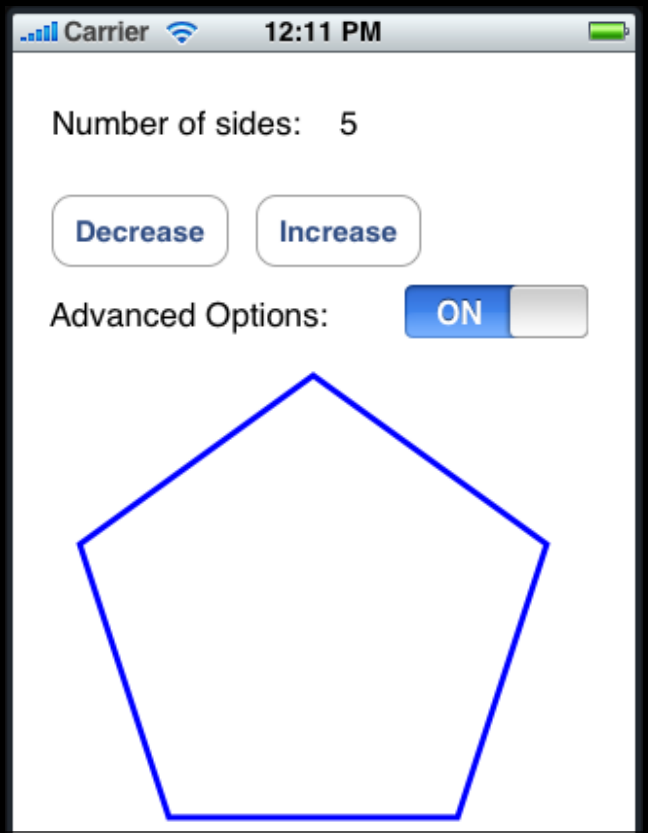


Animating Views

UI 要素の配置をダイナミックに変えるには？

- What if you want to change layout dynamically?
- For example, a switch to disclose additional views...

隠し View を表に出すスイッチ…



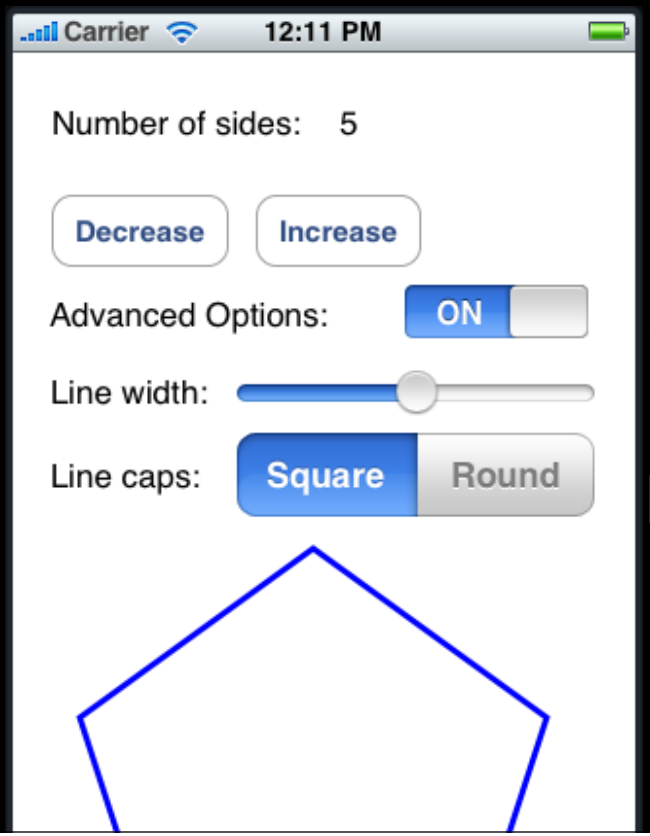
スイッチを入れると…

Animating Views

UI 要素の配置をダイナミックに変えるには？

- What if you want to change layout dynamically?
- For example, a switch to disclose additional views...

隠し View を表に出すスイッチ…



スイッチを入れると…

UIImage が下に下がって
隠し View がフェードイン

UIView Animations

UIView の「アニメーション」とは
属性の値を連続的に変化させること

- UIView supports a number of animatable properties
  - frame, bounds, center, alpha, transform

透明度　幾何変換

- Create “blocks” around changes to animatable properties

ブロック（開始値・終了値など）を指定して属性値を変化させる

- Animations run asynchronously and automatically

アプリ本体とは非同期的（独立して）自動的に
アニメーションは実行される．

ブロックを指定して「スタート」させれば
あとは勝手に動く．

Other Animation Options

- Additional animation options　ブロックに指定できること
  - delay before starting
  - start at specific time
  - curve (ease in/out, ease in, ease out, linear)　速度の「カーブ」
  - repeat count
  - autoreverses (e.g. ping pong back and forth)

ease＝楽・緩　　linear=線形

例. ピンポンのように行き来させる

View Animation Example

「隠しスイッチ」で呼び出されるメソッド

polygonView を下に移動し
optionView をフェードイン

```
- (void)showAdvancedOptions {
    // assume polygonView and optionsView

    [UIView beginAnimations:@"advancedAnimations" context:nil];
    [UIView setAnimationDuration:0.3];

    // make optionsView visible (alpha is currently 0.0)
    optionsView.alpha = 1.0;

    // move the polygonView down
    CGRect polygonFrame = polygonView.frame;
    polygonFrame.origin.y += 200;
    polygonView.frame = polygonFrame;

    [UIView commitAnimations];
}
```

設定開始

↑適当な ID をつける

0.3秒かけてアニメート

初期状態は透明

不透明への変化（を仕込む）

y位置を 200 増加（下へ）

アニメ開始

あとは View が勝手にアニメを実行

アニメーション終了を知るには

Knowing When Animations Finish

- UIView animations allow for a delegate

デリゲートによって
自分の controller につなぐ

```
[UIView setAnimationDelegate:myController];
```

自分の controller に以下のメソッドを用意しておく

- myController will have callbacks invoked before and after

```
- (void)animationWillStart:(NSString *)animationID
        context:(void *)context;
```

開始時に呼ばれる

```
- (void)animationDidStop:(NSString *)animationID
        finished:(NSNumber *)finished
        context:(void *)context;
```

終了時に呼ばれる

- Can provide custom selectors if desired, for example

```
[UIView setAnimationWillStartSelector:
                @selector(animationWillStart)];
[UIView setAnimationDidStopSelector:
                @selector(animationDidStop)];
```

別の名前のメソッドを使うこともできる.

How Does It Work? （アニメーション）の仕組み

- Is drawRect: invoked repeatedly? drawRect が繰り返し呼び出される？
- Do I have to run some kind of timer in order to drive the animation?
- Is it magic?
  それとも「魔法」なのか？

アニメーションを駆動するための
タイマを仕込む必要あるか？

Core Animation 裏方は「コア アニメーション」

- Hardware accelerated rendering engine
  ハードウェア（GPU）直結の OpenGL ベースの描画エンジン
- UIViews are backed by "layers" 裏方に CA Layer
  個々の UI 要素が「backing store (画面表示ビットマップ）」をもつ
- -drawRect: results are cached
  - Cached results used to render view
  - -drawRect: called only when contents change
  - Layers drawn from a separate render tree managed by separate process アニメーションは「別プロセス」で制御される

-drawRect は
要素の中身が変更された
ときのみ呼び出される

- Property animations done automatically by manipulating layers

View Transforms 幾何変換

- Every view has a transform property　拡大・縮小，回転，平行移動
  - used to apply scaling, rotation and translation to a view
- Default "Identity transform" デフォルトは恒等変換（無変換）
- CGAffineTransform structure used to represent transform
- Use CG functions to create, modify transforms

CGAffineTransform Functions (just a small example set)	
CGAffineTransformScale (transform, xScale, yScale)	拡大・縮小
CGAffineTransformRotate (transform, angle)	回転
CGAffineTransformTranslate (transform, xDelta, yDelta)	平行移動

などなど

More Animation Information

アニメーションについて
もっと詳しく…

- *iPhone OS Programming Guide*
  - “Modifying Views at Runtime” section
- *Core Animation Programming Guide*

Assignment 3 Hints

Hello Poly のヒント

アプリ再起動時に「状態」を受け継ぐには…

Saving State Across App Launches

preferences

- NSUserDefaults to read and write prefs & state

ユーザ設定 & アプリ状態

- Singleton object:

```
+ (NSUserDefaults *)standardUserDefaults;
```

```
NSUserDefaults *myDefs = [NSUserDefaults standardUserDefaults];
```

- Methods for storing & fetching common types:

```
- (int)integerForKey:(NSString *)key;
- (void)setInteger:(int)value forKey:(NSString *)key;

- (id)objectForKey:(NSString *)key;
- (void)setObject:(id)value forKey:(NSString *)key;
```

```
[myDefs setInteger:1230 forKey:@"money");
```

- Find an appropriate time to store and restore your state

```
int myMoney = [myDefs integerForKey:@"money];
```

Questions?