DrawingApp

Swift application

Line

```
import UIKit

class Line{
    var start: CGPoint
    var end: CGPoint

    init(start _start: CGPoint, end _end: CGPoint)
   {
        start=_start
        end=_end

    }

}
```

UIKit

The UIKit framework (`UIKit.framework`) provides the crucial infrastructure needed to construct and manage iOS apps. This framework provides the window and view architecture needed to manage an app's user interface, the event handling infrastructure needed to respond to user input, and the app model needed to drive the main run loop and interact with the system.

In addition to the core app behaviors, UIKit provides support for the following features:

- A view controller model to encapsulate the contents of your user interface
- Support for handling touch- and motion-based events
- Support for a document model that includes iCloud integration; see *Document-Based App Programming Guide for iOS*
- Graphics and windowing support, including support for external displays; see *View Programming Guide for iOS*
- Support for managing the app's foreground and background execution
- Printing support; see *Drawing and Printing Guide for iOS*
- Support for customizing the appearance of standard UIKit controls
- Support for text and web content
- Cut, copy, and paste support
- Support for animating user-interface content
- Integration with other apps on the system through URL schemes and framework interfaces
- Accessibility support for disabled users
- Support for the Apple Push Notification service; see *Local and Remote Notification Programming Guide*
- Local notification scheduling and delivery; see *Local and Remote Notification Programming Guide*
- PDF creation
- Support for using custom input views that behave like the system keyboard
- Support for creating custom text views that interact with the system keyboard
- Support for sharing content through email, Twitter, Facebook, and other services

CGPoint

A structure that contains a point in a two-dimensional coordinate system.

Declaration

SWIFT

```
struct CGPoint { var x: a href="" CGFloat /a var y: a
href="" CGFloat /a }
```

OBJECTIVE-C

```
struct CGPoint { CGFloat x; CGFloat y; }; typedef struct
CGPoint CGPoint;
```

Fields

```
 x The x–coordinate of the point.
 y The y–coordinate of the point.
```

DrawView

```
import UIKit

class DrawView: UIView {

    var lines: [Line]=[]
    var lastPoint: CGPoint!


    override func touchesBegan(touches: NSSet, withEvent event: UIEvent) {
        lastPoint = touches.anyObject()?.locationInView(self)
    }
    override func touchesMoved(touches: NSSet, withEvent event: UIEvent) {
        var newPoint = touches.anyObject()?.locationInView(self)
        lines.append(Line(start: lastPoint, end: newPoint!))
        lastPoint=newPoint
        self.setNeedsDisplay()
    }

    override func drawRect(rect: CGRect) {
        var context = UIGraphicsGetCurrentContext()
        CGContextBeginPath(context)
        for line in lines{
            CGContextMoveToPoint(context, line.start.x, line.start.y)
            CGContextAddLineToPoint(context, line.end.x, line.end.y)
        }

        CGContextSetLineCap(context, kCGLineCapRound)
        CGContextSetRGBStrokeColor(context, 0, 0, 0, 1)

        CGContextSetLineWidth(context, 8)
        CGContextStrokePath(context)

    }
}
```

touchesBegan

Responding to Touch Events
Tells the receiver when one or more fingers touch down in a view or window.

Parameters

- *touches*
  A set of `UITouch` instances that represent the touches for the starting phase of the event represented by *event*.

- *event*
  An object representing the event to which the touches belong.

SWIFT
```
func touchesBegan(_ touches: NSSet,
       withEvent event: UIEvent)
```

OBJECTIVE-C
```
- (void)touchesBegan:(NSSet *)touches
         withEvent:(UIEvent *)event
```

Discussion

The default implementation of this method does nothing. However immediate UIKit subclasses of `UIResponder`, particularly `UIView`, forward the message up the responder chain. To forward the message to the next responder, send the message to `super` (the superclass implementation); do not send the message directly to the next responder. For example,

SWIFT
```
    super touchesBegan touches  withEvent  event
```

•

OBJECTIVE-C
```
     super touchesBegan:touches withEvent:event
```

OBJECTIVE-C
```
    @import UIKit;
```

SWIFT
```
    import UIKit
```

If you override this method without calling `super` (a common use pattern), you must also override the other methods for handling touch events, if only as stub (empty) implementations.
Multiple touches are disabled by default. In order to receive multiple touch events you must set the a `multipleTouchEnabled` property of the corresponding view instance to YES.

touchesMoved

Tells the receiver when one or more fingers associated with an event move within a view or window.

Parameters

`touches`

A set of `UITouch` instances that represent the touches that are moving during the event represented by *`event`*.

`event`

An object representing the event to which the touches belong.

OBJECTIVE-C

```
@import UIKit;
```

SWIFT

```
import UIKit
```

The default implementation of this method does nothing. However immediate UIKit subclasses of `UIResponder`, particularly `UIView`, forward the message up the responder chain. To forward the message to the next responder, send the message to `super` (the superclass implementation); do not send the message directly to the next responder. For example,

SWIFT

```
func touchesMoved(_ touches: NSSet,
        withEvent event: UIEvent)
```

OBJECTIVE-C

```
- (void)touchesMoved:(NSSet *)touches
           withEvent:(UIEvent *)event
```

Multiple touches are disabled by default. In order to receive multiple touch events you must set the a `multipleTouchEnabled` property of the corresponding view instance to `YES`.
If you override this method without calling `super` (a common use pattern), you must also override the other methods for handling touch events, if only as stub (empty) implementations.

setNeedDisplay

`- setNeedsDisplay`
Marks the receiver's entire bounds rectangle as needing to be redrawn.

Declaration
SWIFT
```
func setNeedsDisplay()
```
OBJECTIVE-C

OBJECTIVE-C
```
@import UIKit;
```
SWIFT
```
import UIKit
```

Discussion
You can use this method or the `setNeedsDisplayInRect:` to notify the system that your view's contents need to be redrawn. This method makes a note of the request and returns immediately. The view is not actually redrawn until the next drawing cycle, at which point all invalidated views are updated.

NOTE
If your view is backed by a `CAEAGLLayer` object, this method has no effect. It is intended for use only with views that use native drawing technologies (such as UIKit and Core Graphics) to render their content.
You should use this method to request that a view be redrawn only when the content or appearance of the view change. If you simply change the geometry of the view, the view is typically not redrawn. Instead, its existing content is adjusted based on the value in the view's `contentMode` property. Redisplaying the existing content improves performance by avoiding the need to redraw content that has not changed.

drawRect

OBJECTIVE-C
`@import UIKit;`
SWIFT
`import UIKit`

`- drawRect:`
Draws the receiver’s image within the passed-in rectangle.

SWIFT
```
func drawRect(_ rect: CGRect)
```
OBJECTIVE-C
```
- (void)drawRect:(CGRect)rect
```

Parameters
`rect`
The portion of the view’s bounds that needs to be updated. The first time your view is drawn, this rectangle is typically the entire visible bounds of your view. However, during subsequent drawing operations, the rectangle may specify only part of your view.

The default implementation of this method does nothing. Subclasses that use technologies such as Core Graphics and UIKit to draw their view’s content should override this method and implement their drawing code there. You do not need to override this method if your view sets its content in other ways. For example, you do not need to override this method if your view just displays a background color or if your view sets its content directly using the underlying layer object.
By the time this method is called, UIKit has configured the drawing environment appropriately for your view and you can simply call whatever drawing methods and functions you need to render your content.

You can get a reference to the graphics context using the `UIGraphicsGetCurrentContext` function, but do not establish a strong reference to the graphics context because it can change between calls to the `drawRect:` method.

UIGraphicsGetCurrentContext()

Returns the current graphics context.

Declaration
SWIFT
```
func UIGraphicsGetCurrentContext() -> a href="" CGContext /a !
```
OBJECTIVE-C
```
CGContextRef UIGraphicsGetCurrentContext ( void );
```

Return Value
The current graphics context.
Discussion
The current graphics context is `nil` by default. Prior to calling its `drawRect:` method, view objects push a valid context onto the stack, making it current. If you are not using a `UIView` object to do your drawing, however, you must push a valid context onto the stack manually using the `UIGraphicsPushContext` function.
This function may be called from any thread of your app.

Import Statement
OBJECTIVE-C
```
@import UIKit;
```
SWIFT
```
import UIKit
```

CGContextMoveToPoint

Begins a new subpath at the point you specify.

Declaration

SWIFT

```
func CGContextMoveToPoint(_ c: a href="" CGContext /a !,
                        _ x: a href="" CGFloat /a ,
                        _ y: a href="" CGFloat /a )
```

OBJECTIVE-C

```
void CGContextMoveToPoint ( CGContextRef c, CGFloat x, CGFloat y );
```

Parameters

c	*A graphics context.*
x	*The x-value, in user space coordinates, for the point.*
y	*The y-value, in user space coordinates, for the point.*

Discussion

This point you specify becomes the start point of a new subpath. The current point is set to this start point.

Import Statement

OBJECTIVE-C

```
@import CoreGraphics;
```

SWIFT

```
import CoreGraphics
```