

# CS193P - Lecture 11

## iPhone Application Development

Text Input  
Presenting Content Modally

# Announcements

# Announcements

- Paparazzi 3 assignment is due Wednesday 2/17

# Announcements

- Paparazzi 3 assignment is due Wednesday 2/17
- This Friday's extra session will feature Evan Doll

# Today's Topics

- Threading Wrap-Up
- iPhone Keyboards
- Customizing Text Input
- Presenting Content Modally

# Alternatives to Threading

# Alternatives to Threading

- Asynchronous (nonblocking) functions
  - Specify target/action or delegate for callback
  - **NSURLConnection** has synchronous and asynchronous variants

# Alternatives to Threading

- Asynchronous (nonblocking) functions
  - Specify target/action or delegate for callback
  - **NSURLConnection** has synchronous and asynchronous variants
- Timers
  - One-shot or recurring
  - Specify a callback method
  - Managed by the run loop



# Alternatives to Threading

- Asynchronous (nonblocking) functions
  - Specify target/action or delegate for callback
  - **NSURLConnection** has synchronous and asynchronous variants
- Timers
  - One-shot or recurring
  - Specify a callback method
  - Managed by the run loop
- Higher level constructs like **operations**

# NSOperation

- Abstract superclass
- Manages thread creation and lifecycle
- Encapsulate a **unit of work** in an object
- Specify priorities and dependencies

# Creating an NSOperation Subclass

# Creating an NSOperation Subclass

- Define a **custom init method**

```
- (id)initWithSomeObject:(id)someObject
{
    self = [super init];
    if (self) {
        self.someObject = someObject;
    }
    return self;
}
```

# Creating an NSOperation Subclass

- Define a **custom init method**

```
- (id)initWithSomeObject:(id)someObject
{
    self = [super init];
    if (self) {
        self.someObject = someObject;
    }
    return self;
}
```

- **Override -main method** to do work

```
- (void)main
{
    [someObject doLotsOfTimeConsumingWork];
}
```

# NSOperationQueue

- Operations are typically scheduled by **adding to a queue**
- Choose a maximum number of concurrent operations
- Queue runs operations based on priority and dependencies

# Using an NSInvocationOperation

- Concrete subclass of NSOperation
- For lightweight tasks where creating a subclass is overkill

# Using an NSInvocationOperation

- Concrete subclass of NSOperation
- For lightweight tasks where creating a subclass is overkill

```
- (void)someAction:(id)sender
{
    NSInvocationOperation *operation =
        [[NSInvocationOperation alloc] initWithTarget:self
                                                selector:@selector(doWork:)
                                                object:someObject];

    [queue addObject:operation];

    [operation release];
}
```

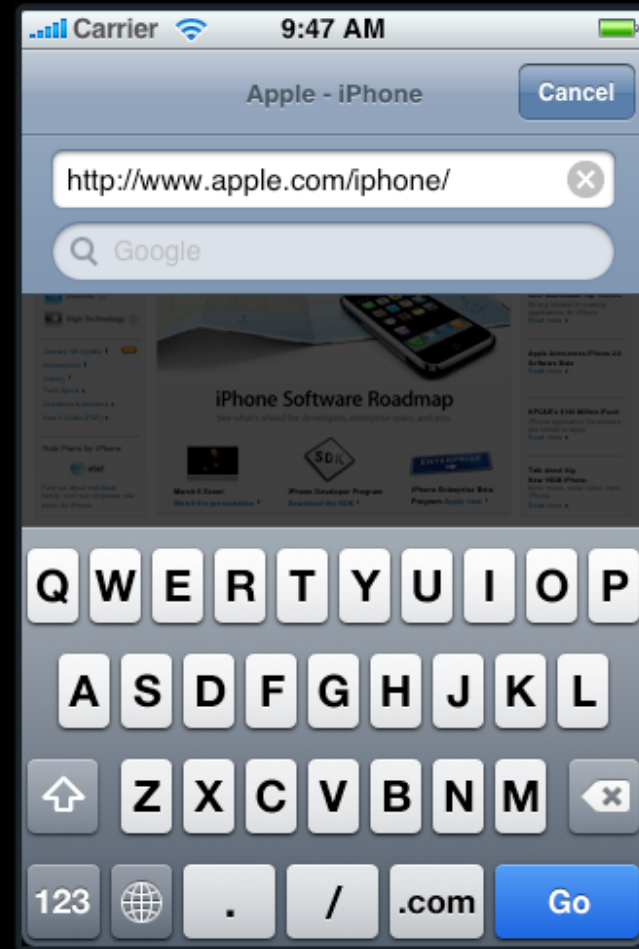


# iPhone Keyboards

# Virtual keyboard Appears when needed



# Virtual keyboard Appears when needed





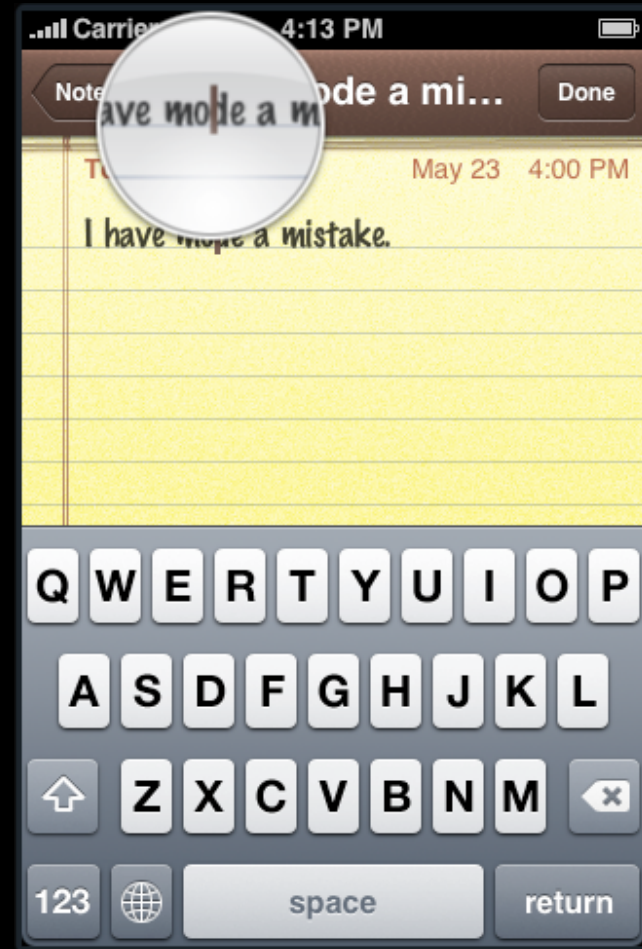




## Portrait and Landscape

# Simple selection model

## Text loupe/magnifier



# Many keyboard types Adapted to task

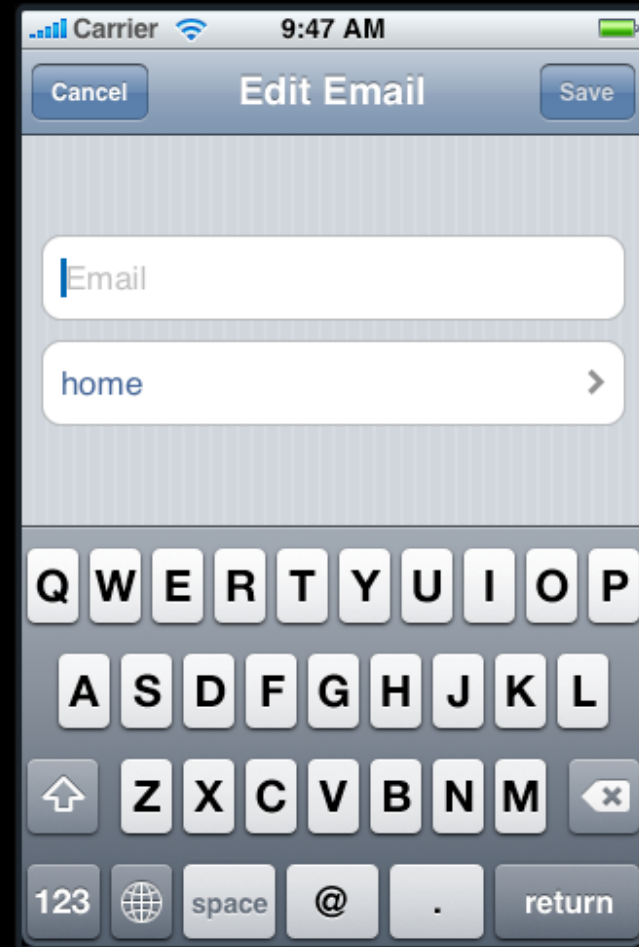




# Many keyboard types Adapted to task



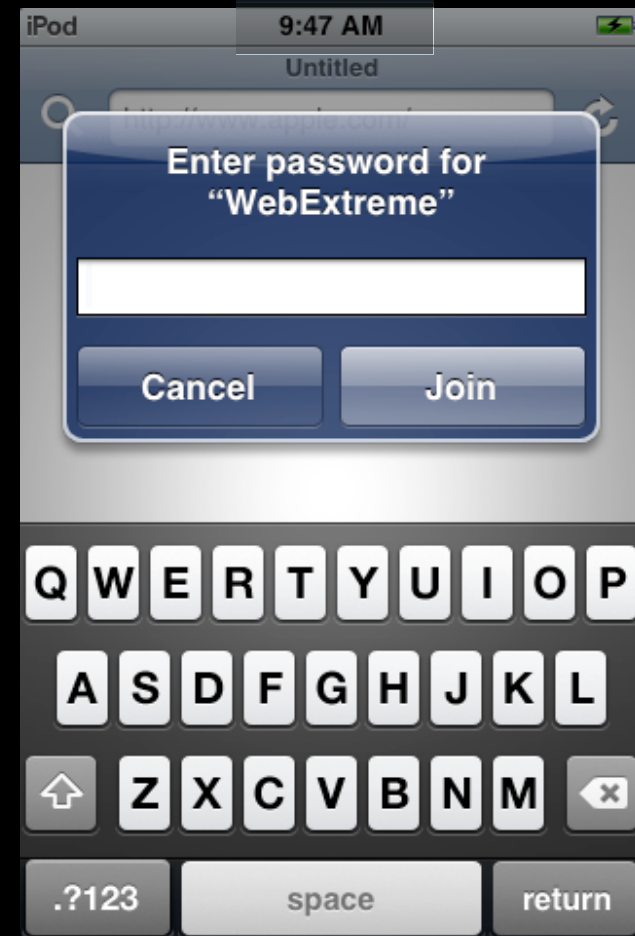
Many keyboard types  
Adapted to task



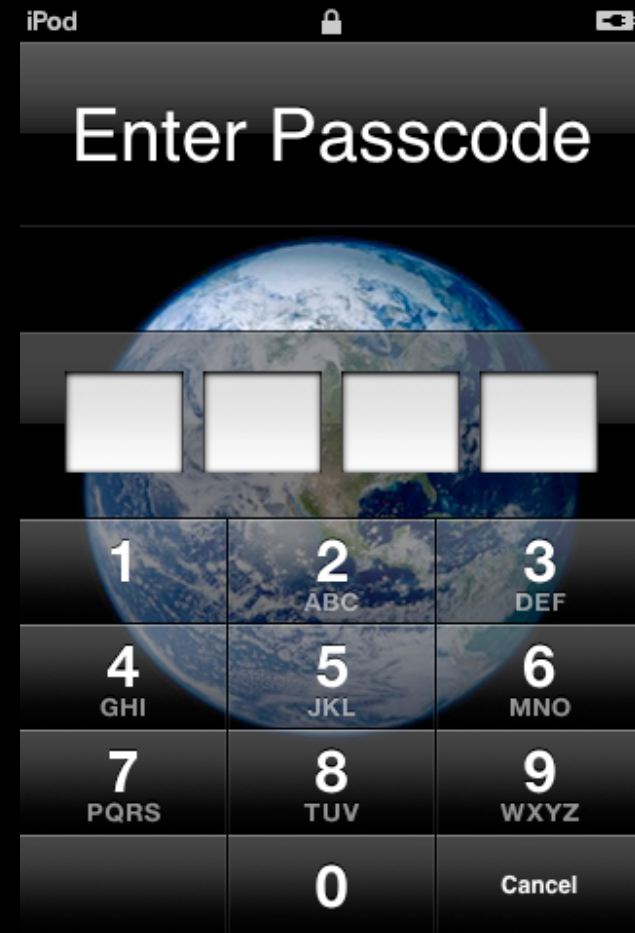
**Many keyboard types  
Adapted to task**



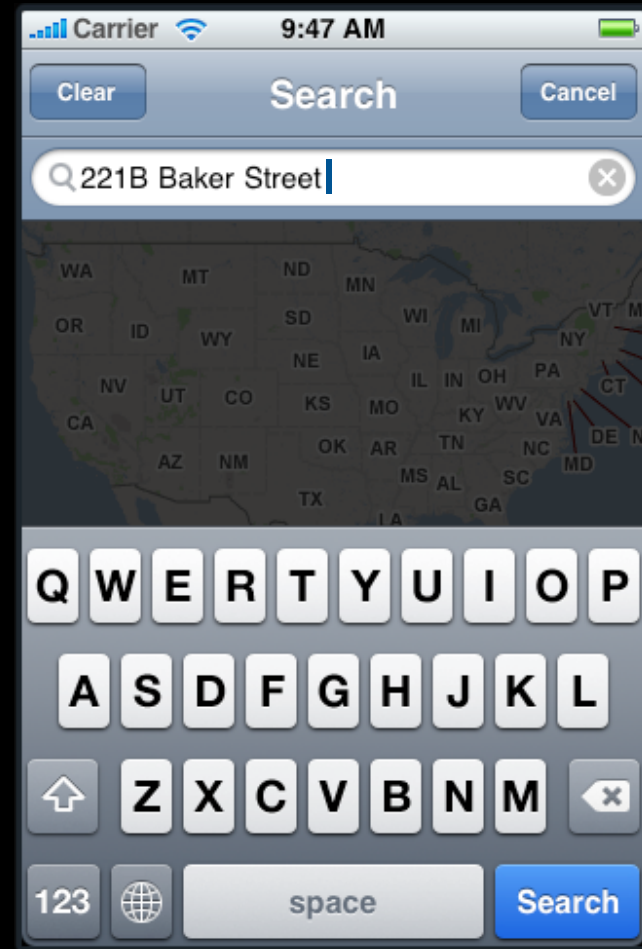
Many keyboard types  
Adapted to task



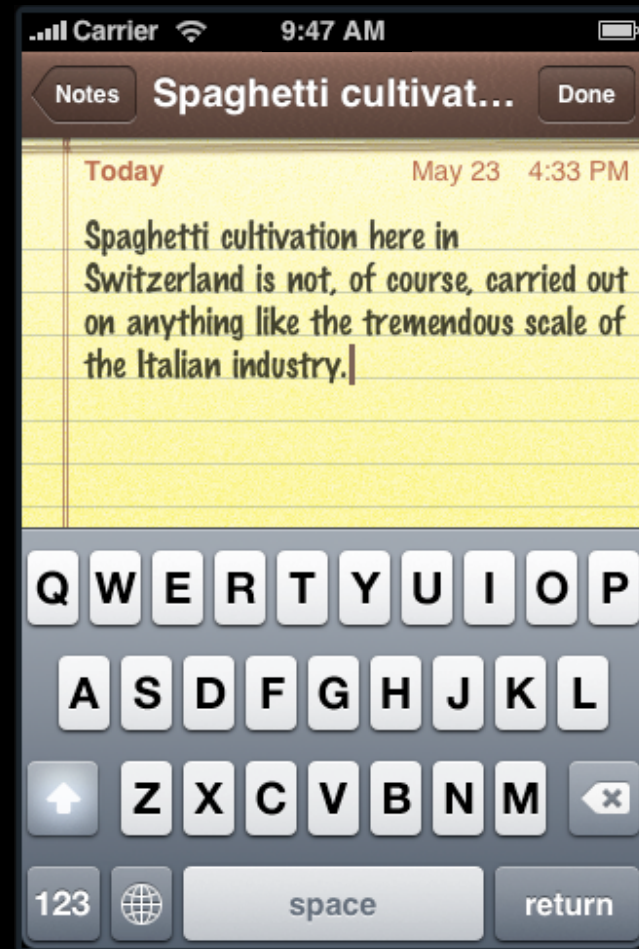
Many keyboard types  
Adapted to task



# Single line editing

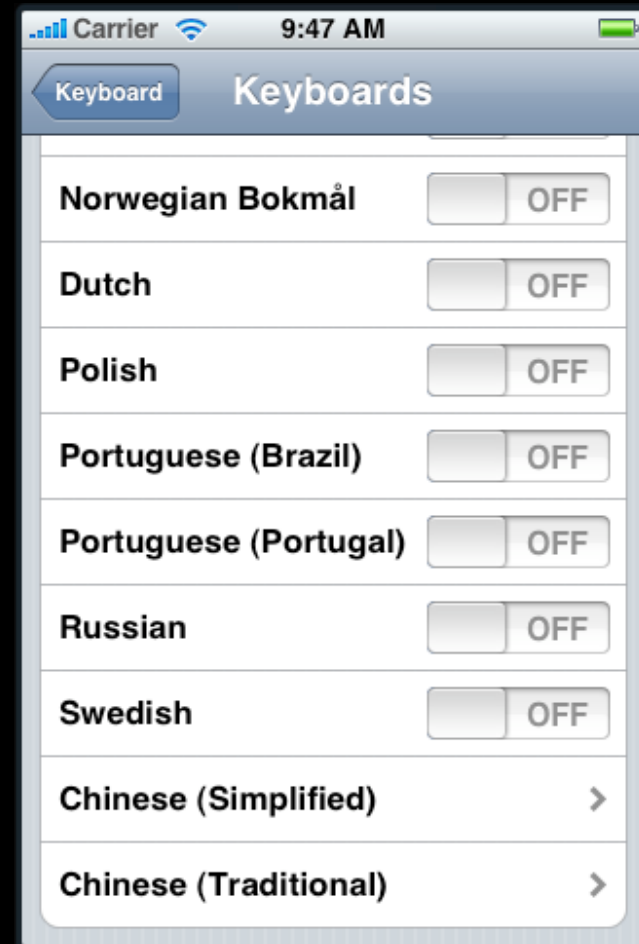


# Multi-line editing



40

Languages

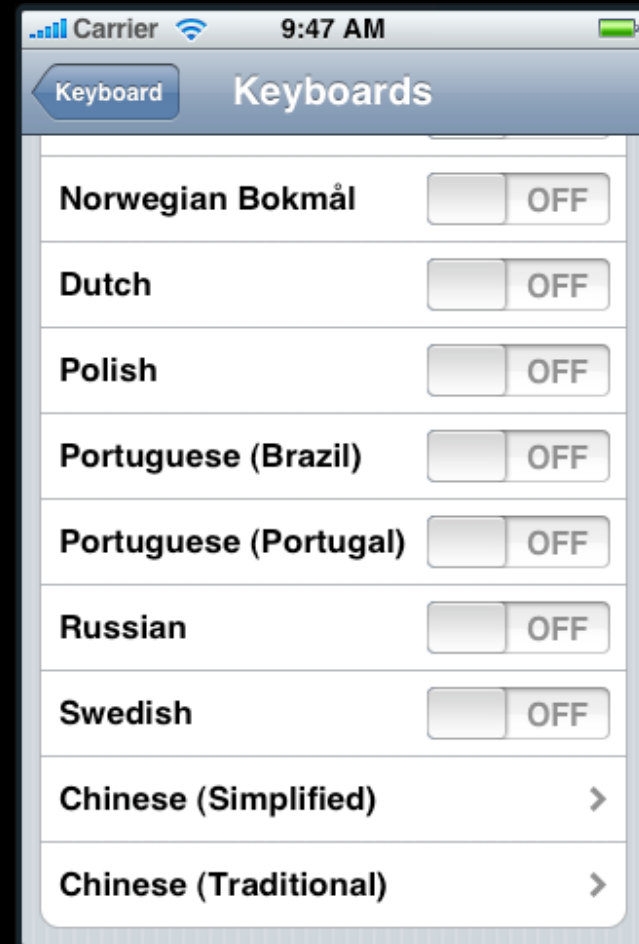




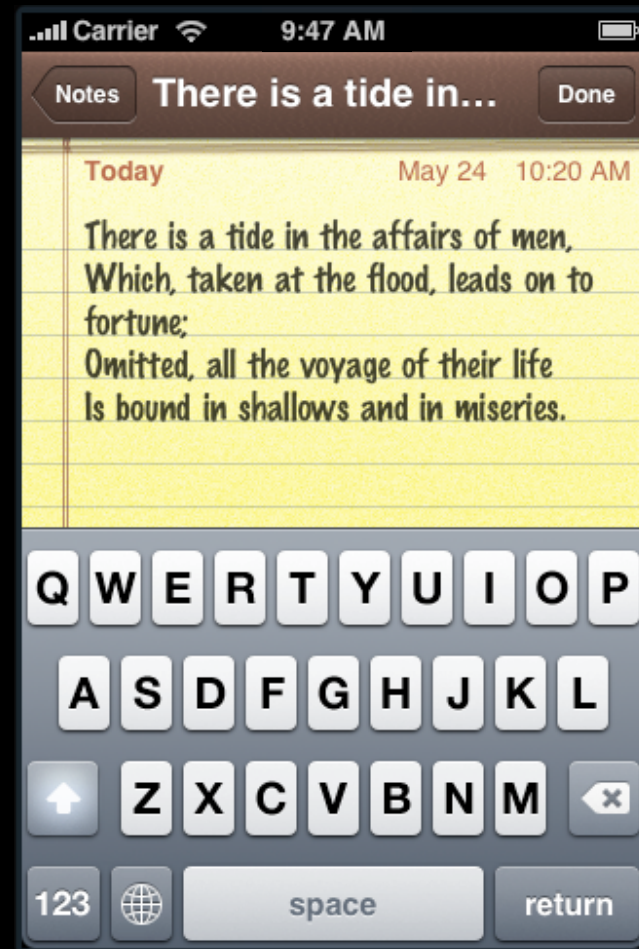
40

Languages

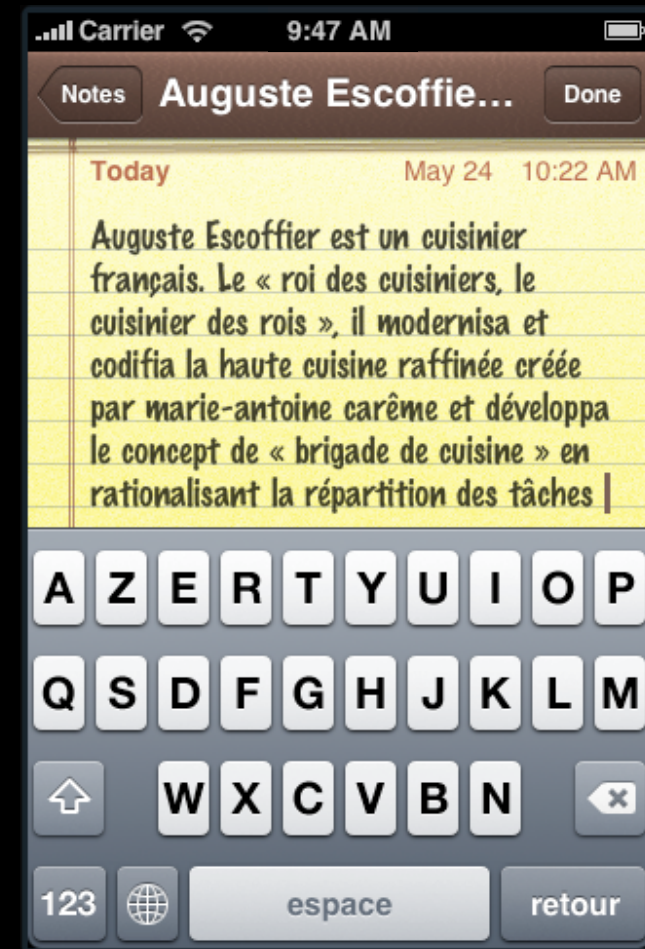
Full dictionary support



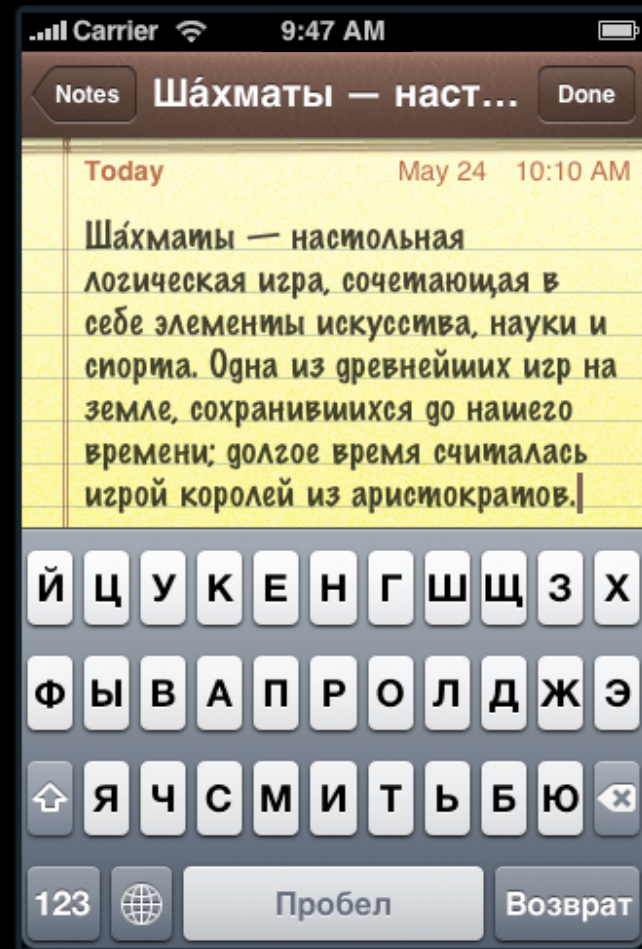
# English



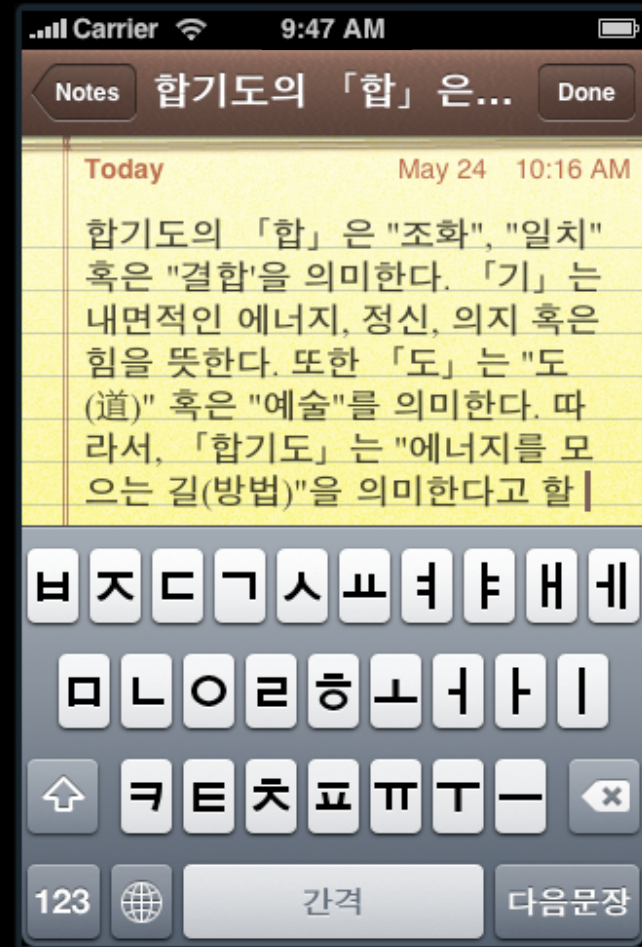
# French



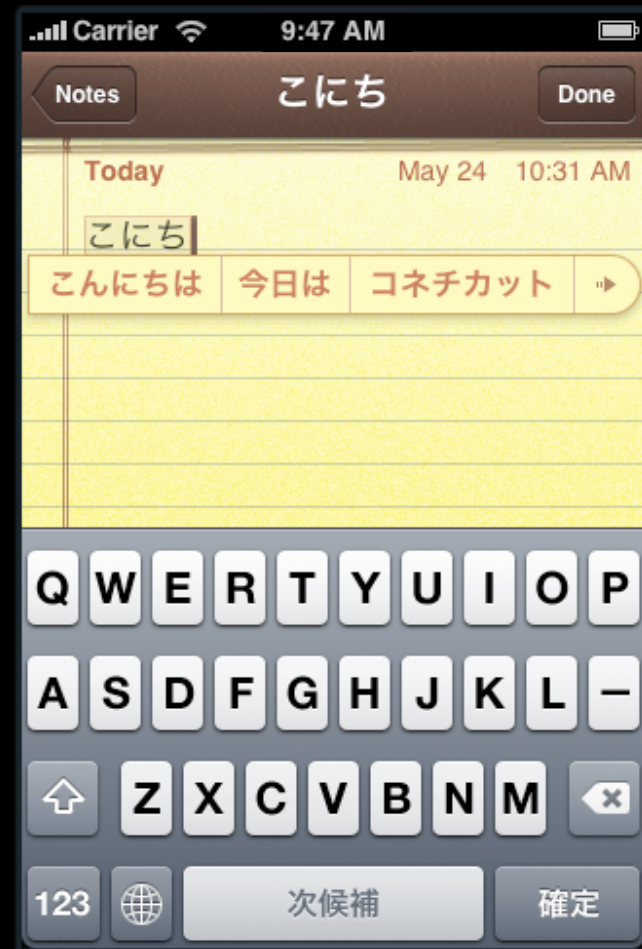
# Russian



# Korean



# Japanese Romaji

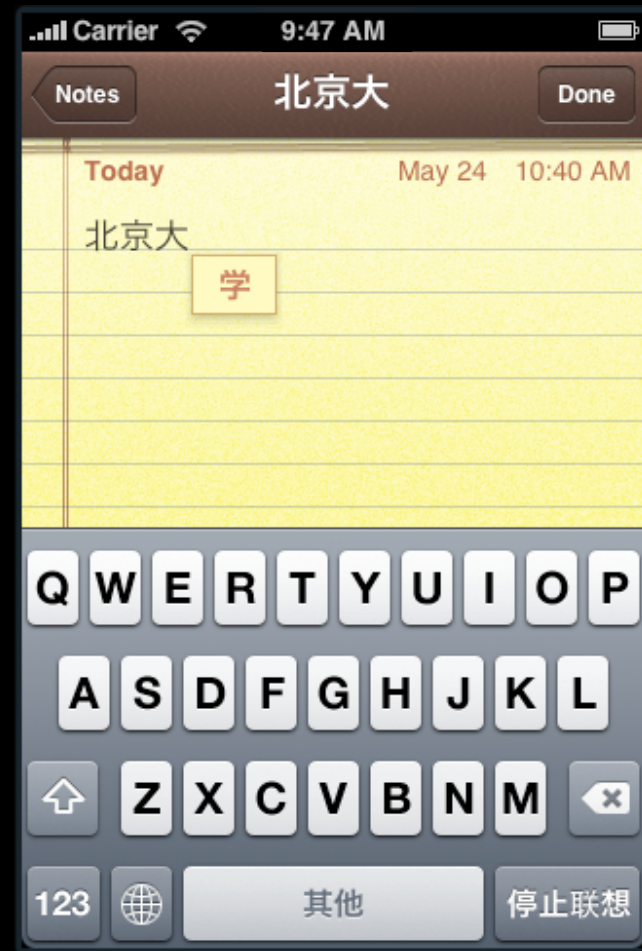


# Japanese Kana





# Chinese Pinyin





# Chinese Handwriting

Simplified  
Traditional



# Customizing Text Input

A blue puzzle piece is centered on a black background. The puzzle piece has a standard interlocking shape with a semi-circular protrusion on the left and a corresponding semi-circular indentation on the right. The text "Text Containers" is written in white, bold, sans-serif font across the middle of the puzzle piece.

# Text Containers



## **Text Containers**

**Delegates  
Notifications  
Methods**



**Text Containers**

**Text Input Traits**



Text Input Traits

**Protocol**

UITextField

UITextView



## Text Input Traits

Autocapitalization

Autocorrection

Keyboard Type

Keyboard Appearance

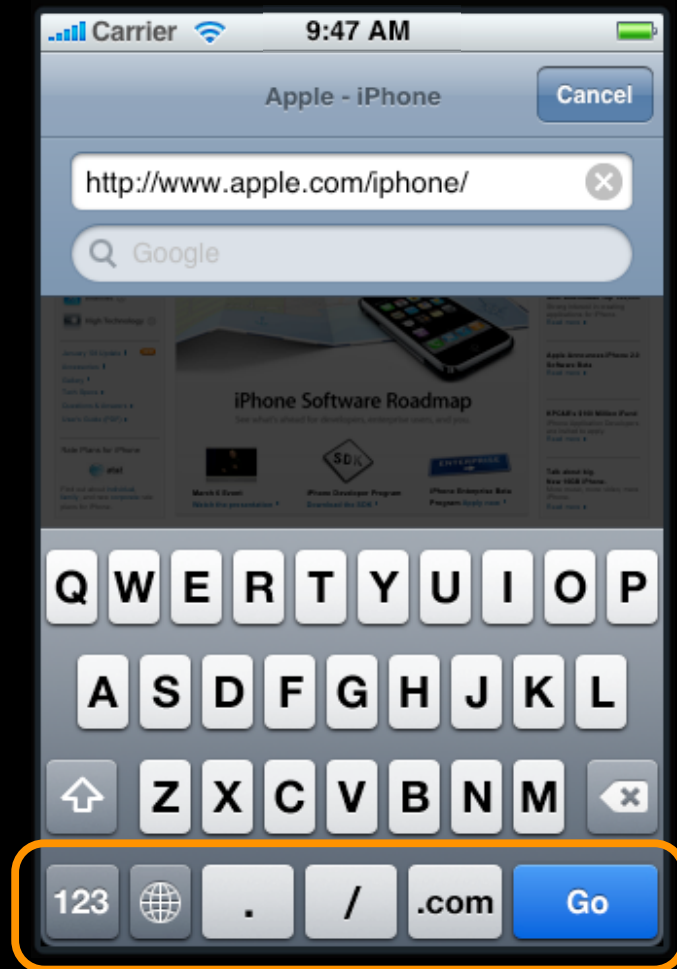
Return Key Type

Return Key Autoenabling

Secure Text Entry

## Text Input Traits

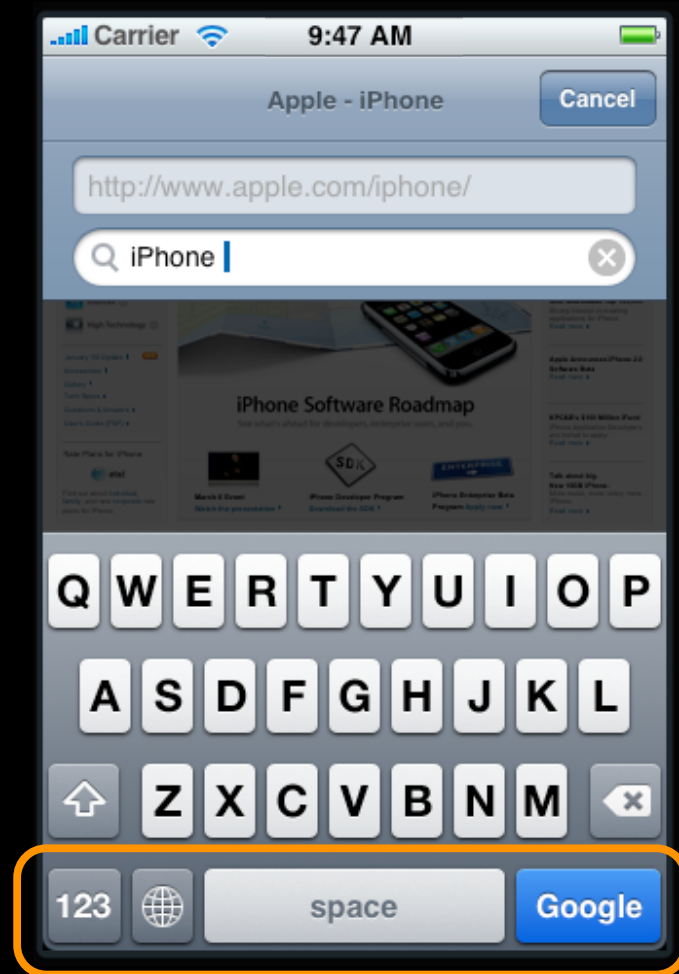
URL Keyboard  
Go button





## Text Input Traits

Default Keyboard  
*Google* button





# Text Containers

Text Input Traits

Delegates

Notifications

Methods

A blue puzzle piece is centered on a black background. The puzzle piece has a standard interlocking shape with a semi-circular protrusion on the left and a corresponding semi-circular indentation on the right. The text "Text Containers" is written in white, bold, sans-serif font across the middle of the puzzle piece.

# Text Containers



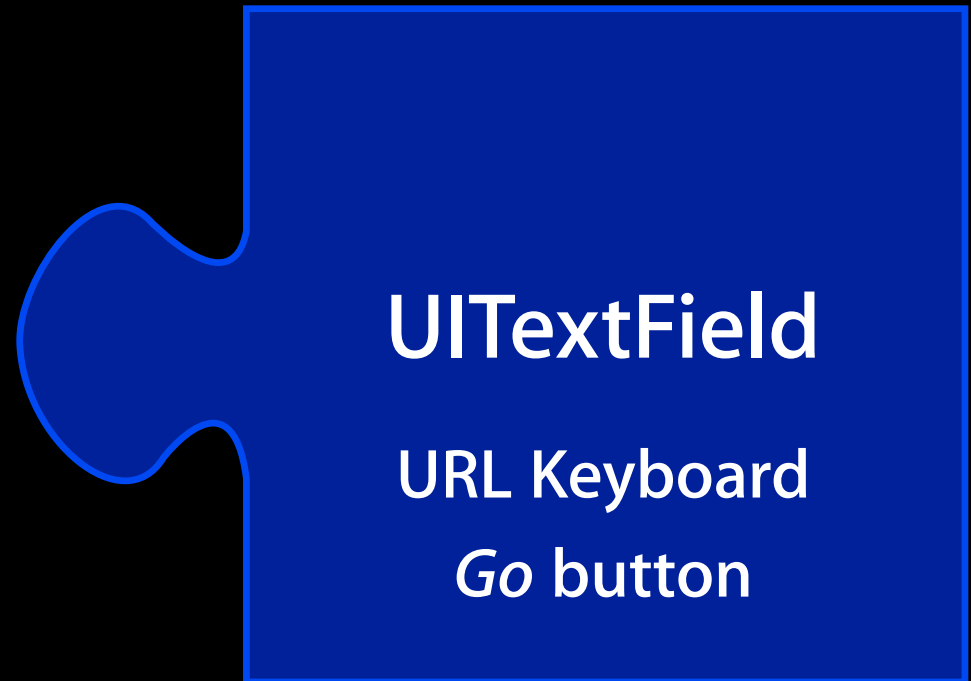
UITextField

Design time

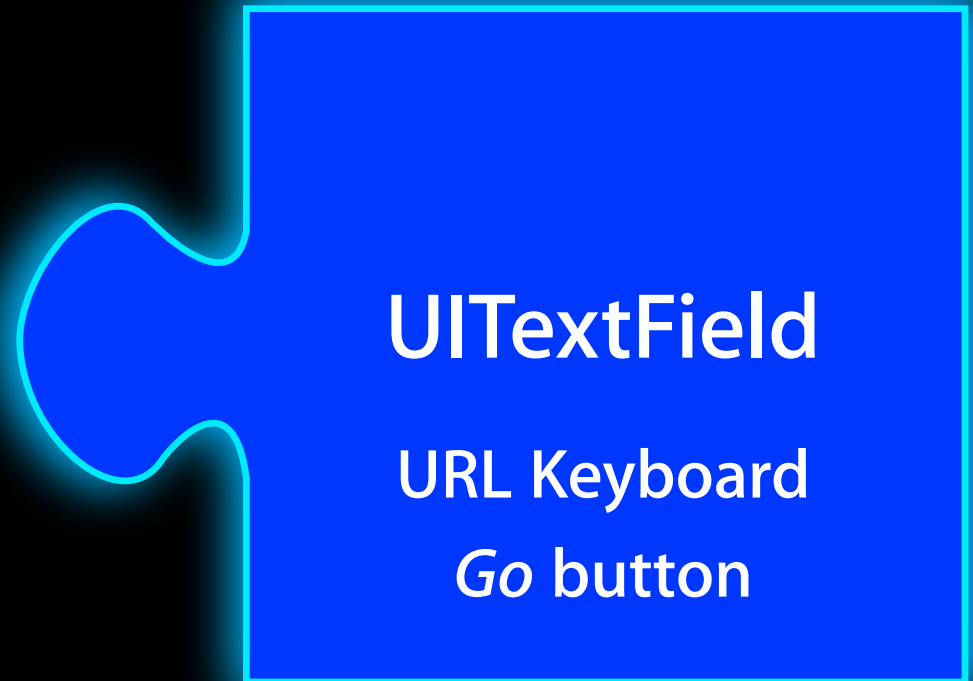


UITextField  
URL Keyboard  
Go button

Design time



Run time



Become first responder



Keyboard

UITextField

URL Keyboard

Go button

Become first responder





Keyboard

UITextField

URL Keyboard

Go button

Become first responder



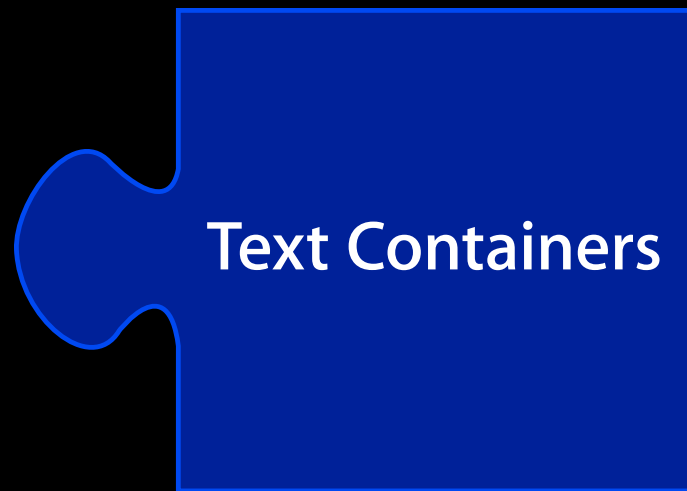
Become first responder



**Keyboard adopts traits**



**Keyboard adopts traits**



**UITextField**

**UITextView**

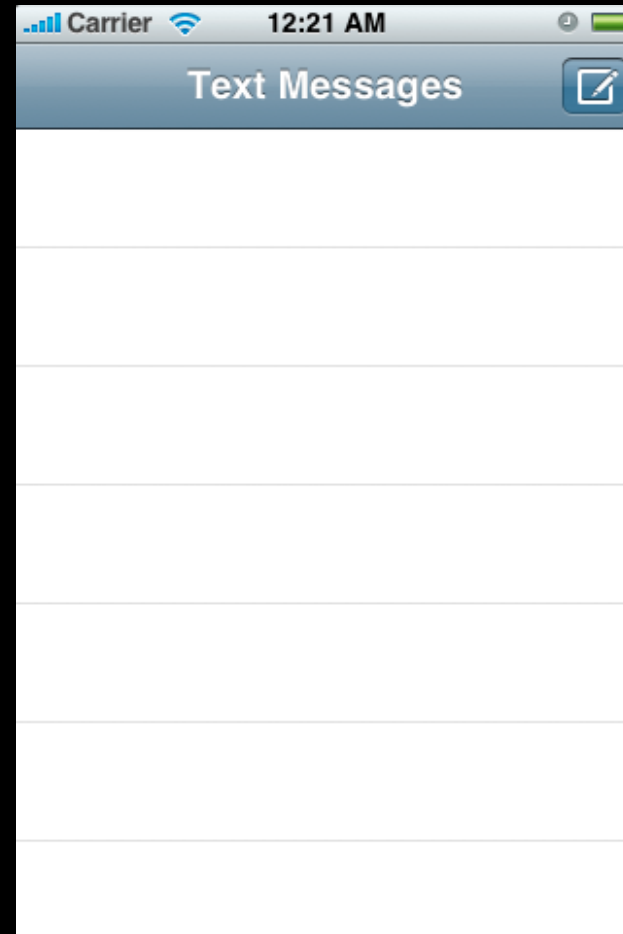
**Web Forms**

# Demo: Text Input

# Presenting Content Modally

# Presenting Content Modally

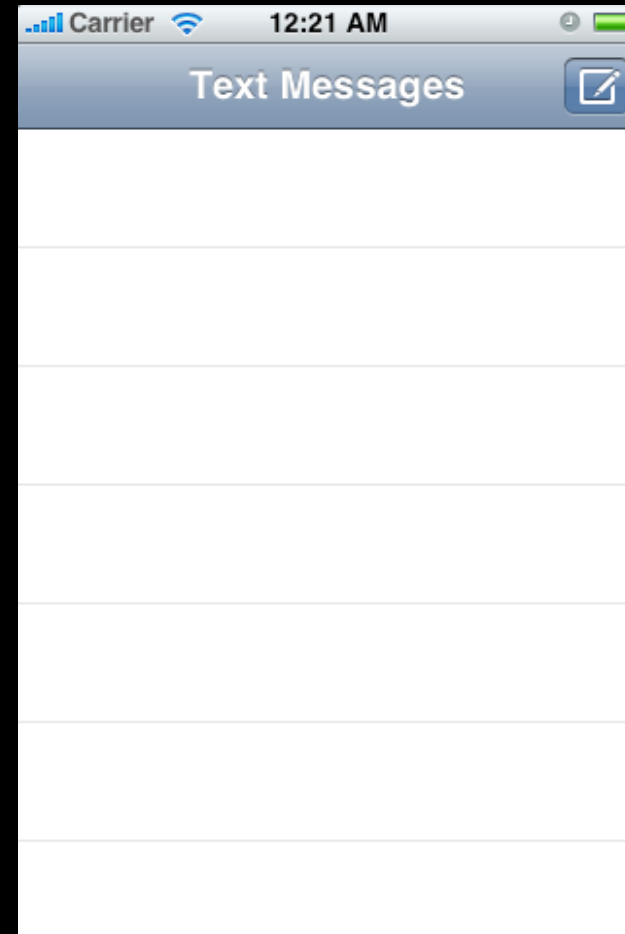
- For **adding** or **picking** data





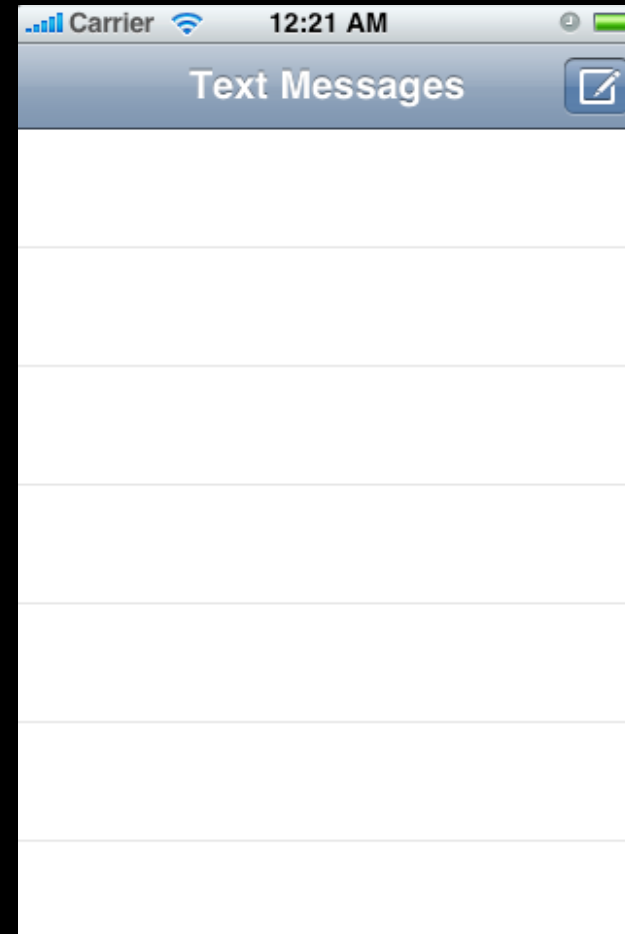
# Presenting Content Modally

- For **adding** or **picking** data



# Presenting Content Modally

- For **adding** or **picking** data



# Presenting a View Controller



# Presenting a View Controller

```
// Recipe list view controller
- (void)showAddRecipe {
    RecipeAddViewController *viewController = ...;
    [self presentViewController:viewController animated:YES];
}
```



# Presenting a View Controller

```
// Recipe list view controller
- (void)showAddRecipe {
    RecipeAddViewController *viewController = ...;
    [self presentViewController:viewController animated:YES];
}
```



# Dismissing a View Controller



# Dismissing a View Controller

```
// Recipe list view controller  
- (void)didAddRecipe {  
    [self dismissModalViewControllerAnimated:YES];  
}
```



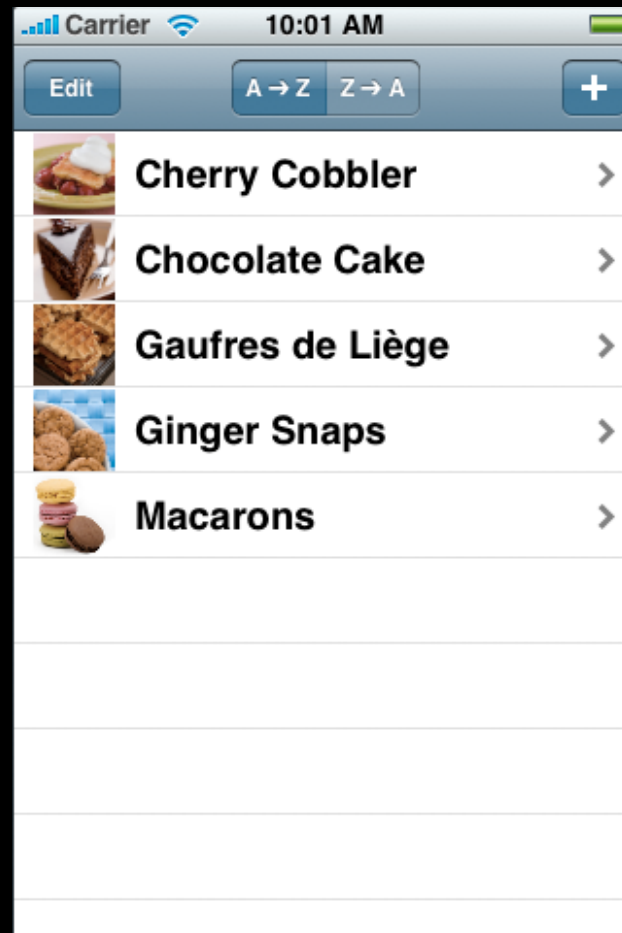
# Dismissing a View Controller

```
// Recipe list view controller
- (void)didAddRecipe {
    [self dismissModalViewControllerAnimated:YES];
}
```

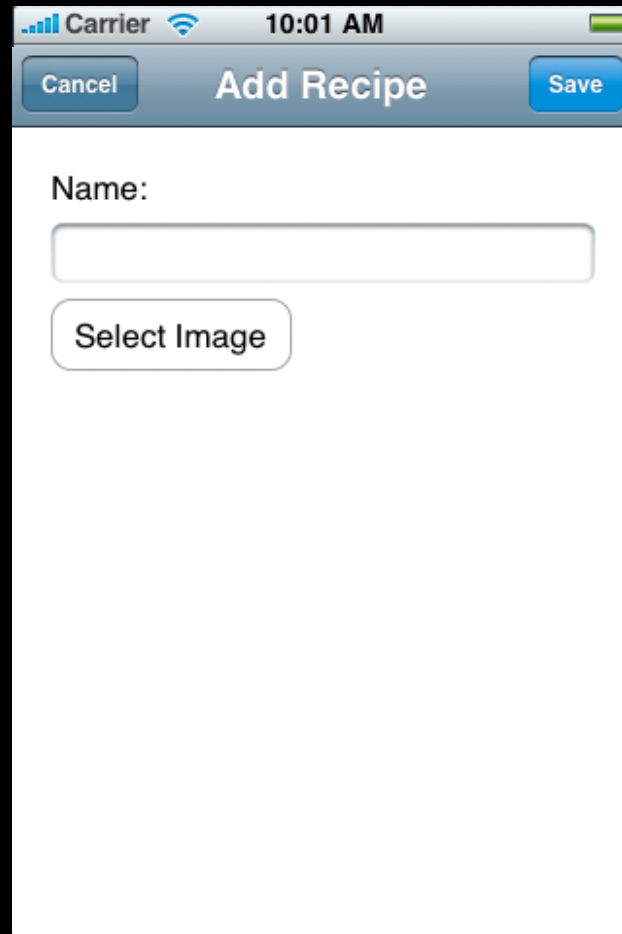




# Separate Navigation Stacks

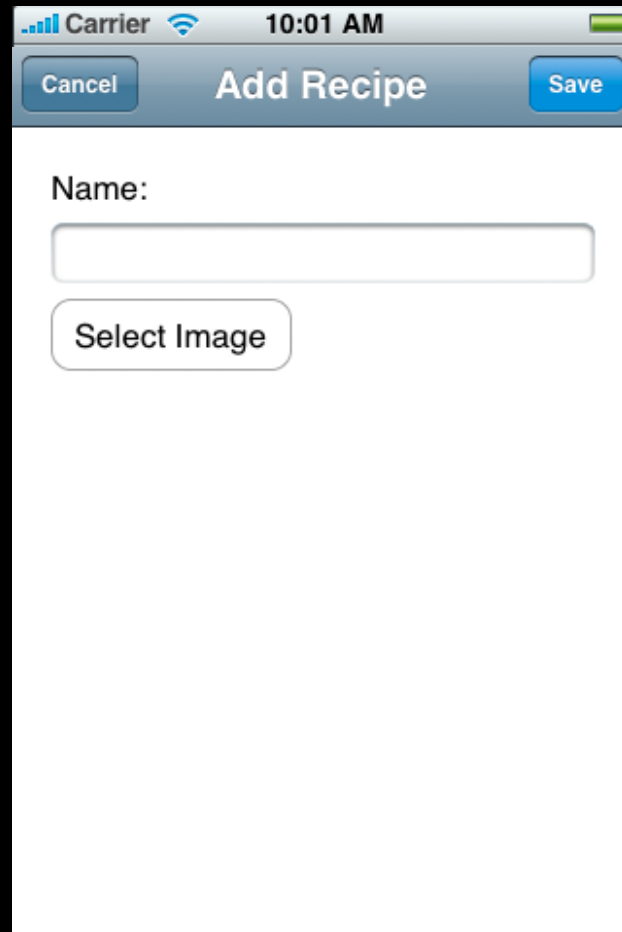


# Separate Navigation Stacks



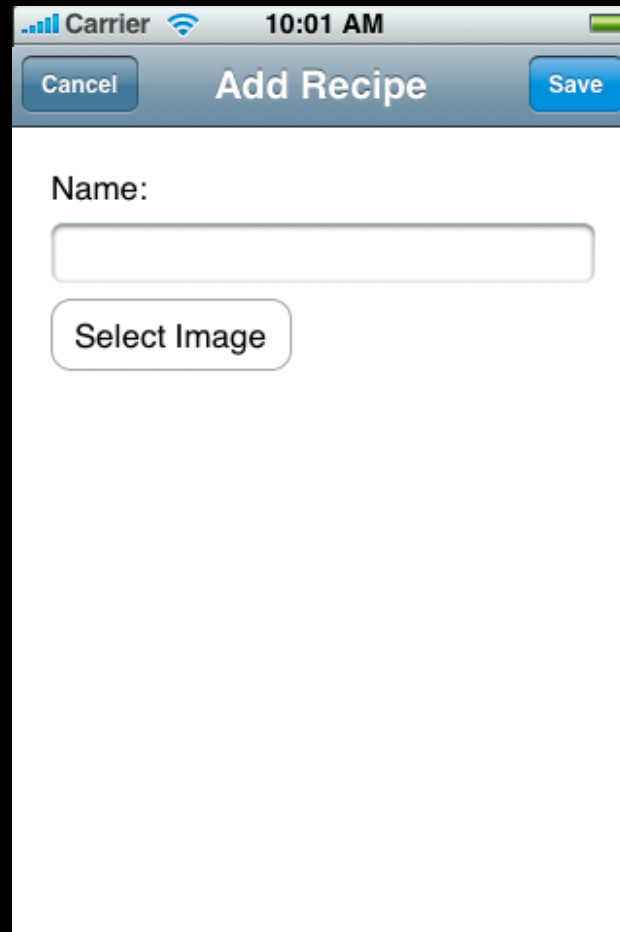
The screenshot shows an iPhone interface with a status bar at the top displaying 'Carrier', signal strength, Wi-Fi, and the time '10:01 AM'. Below the status bar is a modal form titled 'Add Recipe'. The form has a header bar with three buttons: 'Cancel' on the left, 'Add Recipe' in the center, and 'Save' on the right. The main content area of the form contains a label 'Name:' followed by a text input field. Below the input field is a button labeled 'Select Image'.

# Separate Navigation Stacks



The image shows a screenshot of an iOS application interface. At the top, the status bar displays 'Carrier', signal strength bars, a Wi-Fi icon, the time '10:01 AM', and a battery level icon. Below the status bar is a modal form titled 'Add Recipe'. The form has a header bar with three buttons: 'Cancel' on the left, 'Add Recipe' in the center, and 'Save' on the right. The main content area of the form contains a label 'Name:' followed by a text input field. Below the input field is a button labeled 'Select Image'.

# Separate Navigation Stacks



The image shows a screenshot of an iOS application interface. At the top, the status bar displays 'Carrier', signal strength bars, a Wi-Fi icon, the time '10:01 AM', and a battery level icon. Below the status bar is a modal form titled 'Add Recipe'. The form has a header bar with three buttons: 'Cancel' on the left, 'Add Recipe' in the center, and 'Save' on the right. The main content area of the form contains a label 'Name:' followed by a text input field. Below the input field is a button labeled 'Select Image'.

# Separate Navigation Stacks



# Dismissing a Modal View Controller

# Dismissing a Modal View Controller

- Who should do it?

# Dismissing a Modal View Controller

- Who should do it?
- Best practice is for the **same object** to call present and dismiss



# Dismissing a Modal View Controller

- Who should do it?
- Best practice is for the **same object** to call present and dismiss
- **Define delegate methods** for the presented controller

# Dismissing a Modal View Controller

- Who should do it?
- Best practice is for the **same object** to call present and dismiss
- **Define delegate methods** for the presented controller
  - Tell the delegate when the presented controller is done

# Dismissing a Modal View Controller

- Who should do it?
- Best practice is for the **same object** to call present and dismiss
- **Define delegate methods** for the presented controller
  - Tell the delegate when the presented controller is done
  - The delegate makes the call to dismiss

# Dismissing a Modal View Controller

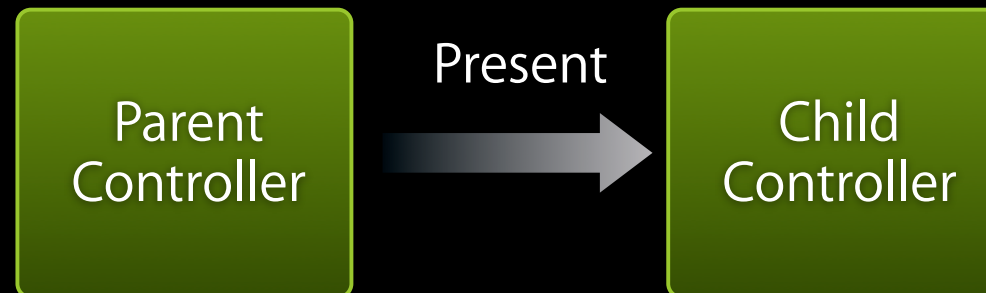
- Who should do it?
- Best practice is for the **same object** to call present and dismiss
- **Define delegate methods** for the presented controller
  - Tell the delegate when the presented controller is done
  - The delegate makes the call to dismiss



Parent  
Controller

# Dismissing a Modal View Controller

- Who should do it?
- Best practice is for the **same object** to call present and dismiss
- **Define delegate methods** for the presented controller
  - Tell the delegate when the presented controller is done
  - The delegate makes the call to dismiss



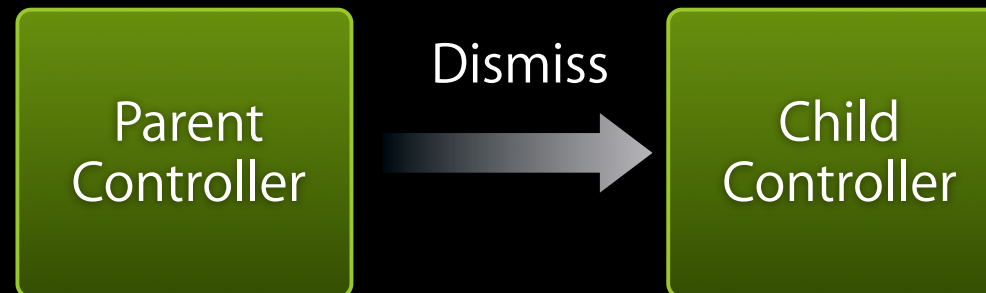
# Dismissing a Modal View Controller

- Who should do it?
- Best practice is for the **same object** to call present and dismiss
- **Define delegate methods** for the presented controller
  - Tell the delegate when the presented controller is done
  - The delegate makes the call to dismiss



# Dismissing a Modal View Controller

- Who should do it?
- Best practice is for the **same object** to call present and dismiss
- **Define delegate methods** for the presented controller
  - Tell the delegate when the presented controller is done
  - The delegate makes the call to dismiss



# Dismissing a Modal View Controller

- Who should do it?
- Best practice is for the **same object** to call present and dismiss
- **Define delegate methods** for the presented controller
  - Tell the delegate when the presented controller is done
  - The delegate makes the call to dismiss



Parent  
Controller



# Demo: Presenting Content Modally

# Questions?