# Developing IOS apps using objective-c


Presented By
Abdelrhman Sayed

Java™ Education
and Technology Services

Invest In Yourself ,
Develop Your Career

Java™ Education and Technology Services

# Lecture three

Objective-C

# Agenda

- Toolbar.
- UIAlertView.
- Working With images.
- Tab Bar Controller.
- User Defaults.
- pList Files.
- Swip.

# Toolbar

# Toolbar

- It is a visual element added to the views to increment the functionality of view.
- It works as button holder.
- Always appears at the bottom edge of a screen or view.

# Toolbar buttons

- It is a single event buttons.
- They are placed to the toolbar to make functions on this view.

# Toolbar Demo

# UIAlertController

# UIAlertController

**Usages:**

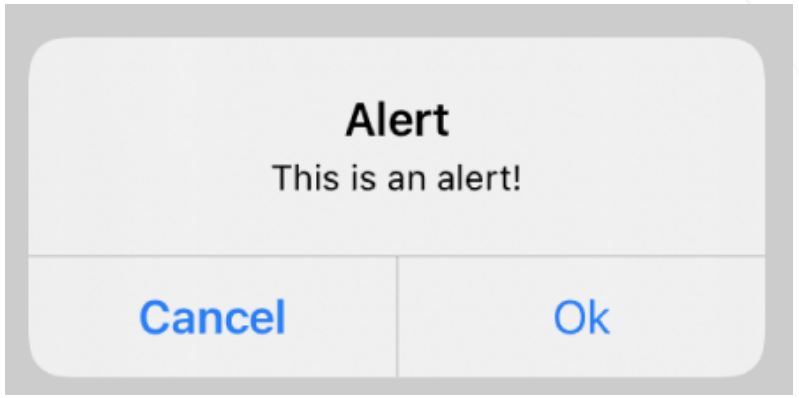
- It is used to notify an error, or message.
- It is also used to confirm an action, and providing cancelation action.

# UIAlertController Cont.

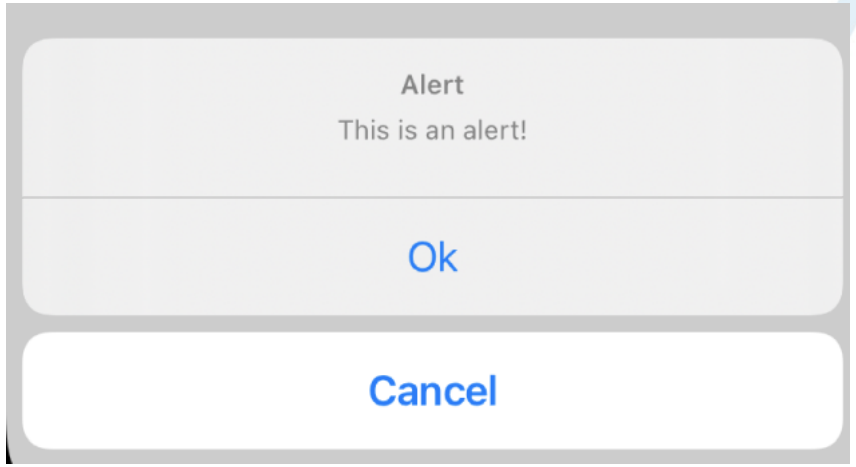
```
UIAlertController* alert = [UIAlertController alertControllerWithTitle:@"Alert"
                            message:@"This is an alert!"
                            preferredStyle:UIAlertControllerStyleAlert];

UIAlertAction* action = [UIAlertAction actionWithTitle:@"Ok"
                         style:UIAlertActionStyleDefault
                         handler:^(UIAlertAction * _Nonnull action) {

printf("Ok");
}];


UIAlertAction* cancel = [UIAlertAction actionWithTitle:@"Cancel"
                         style:UIAlertActionStyleCancel
                         handler:NULL];


[alert addAction:action];
[alert addAction:cancel];

[self presentViewController:alert animated:YES completion:NULL];
```

# UIAlertController Style

```
preferredStyle:UIAlertControllerStyleAlert
```

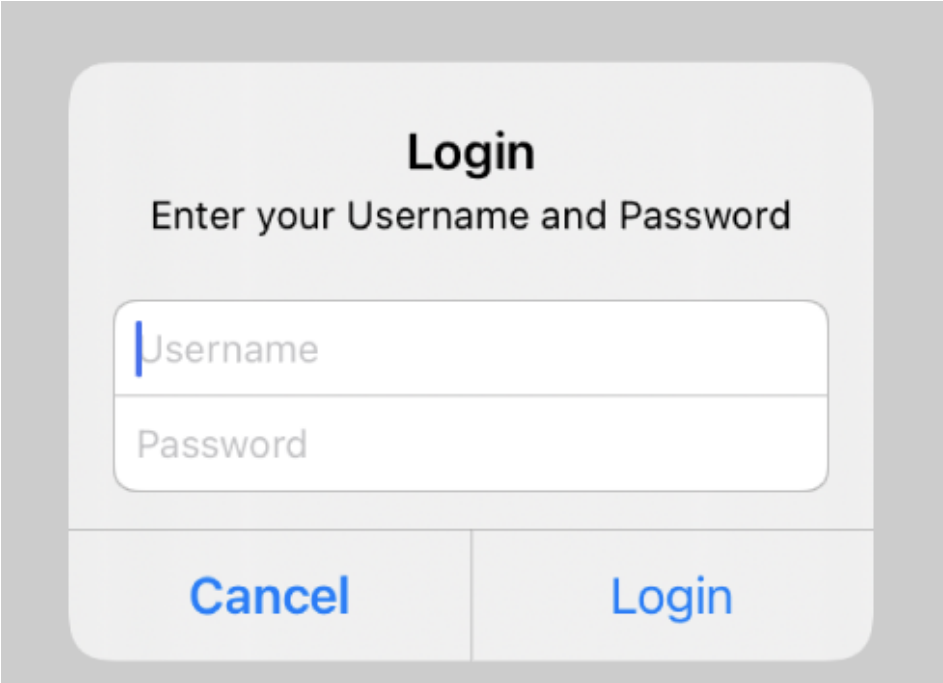



```
preferredStyle:UIAlertControllerStyleActionSheet
```




Objective-C

# UIAlertController With TextField



# UIAlertController With TextField Cont

```
UIAlertController* alert = [UIAlertController alertControllerWithTitle:@"Login"
                            message:@"Enter your Username and Password"
                            preferredStyle:UIAlertControllerStyleAlert];


[alert addTextFieldWithConfigurationHandler:^(UITextField * _Nonnull textField) {
    textField.placeholder = @"Username";
}];

[alert addTextFieldWithConfigurationHandler:^(UITextField * _Nonnull textField) {
    textField.placeholder = @"Password";
}];



UIAlertAction* action = [UIAlertAction actionWithTitle:@"Login"
                         style:UIAlertActionStyleDefault
                         handler:^(UIAlertAction * _Nonnull action) {

    NSString* userName = [[alert textFields][0] text];
    printf("%s",[userName UTF8String]);
}];


UIAlertAction* cancel = [UIAlertAction actionWithTitle:@"Cancel"
                         style:UIAlertActionStyleCancel
                         handler:NULL];
```

Objective-C

# UIAlertController Demo

# Working With Images

# Main Components

- The image it self should be stored under resources, you can add it by drag and drop from any folder.
- The object representing the image is UIImageView object.
- You place the image on UI design by adding image view and setting its image value to the image file name.

# Setting Image

- After creating an UIImageView object “img”, simply call

```
UIImage * image = [UIImage imageNamed:@"JETS.png"];
[img setImage:image];
```

- As “JETS.png” is an image file dragged and dropped under Resources tab.

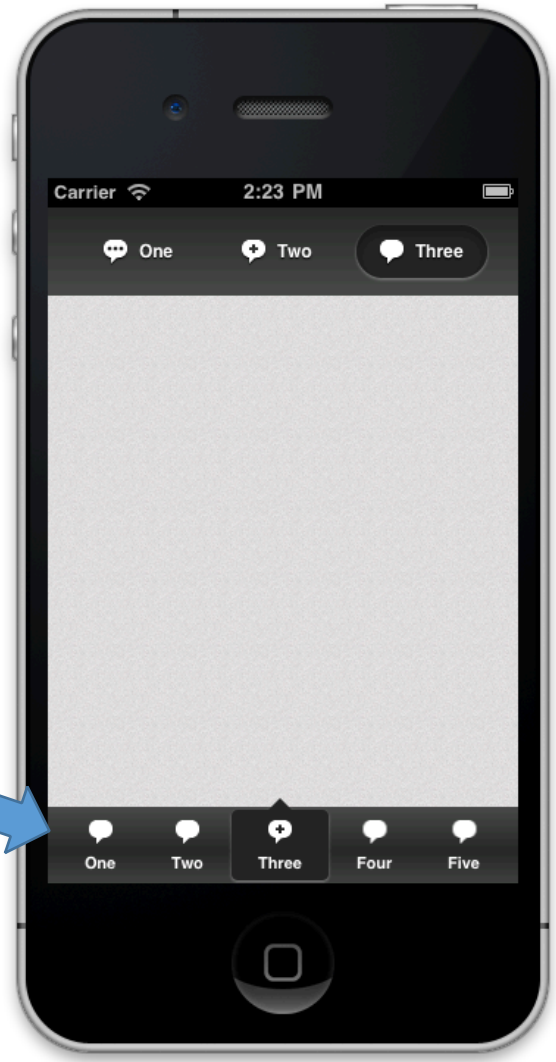
# Tab Bar Controller

# Tab View

- Tab View is used in general for displaying view in the same level.
- Creating tab view is similar to creating navigation controller view.
- The tab bar interface displays tabs at the bottom of the window for selecting between different modes, and for displaying the views for that mode.
- Each tab of a tab bar controller interface is associated with a specific view controller.

# Tab View

Tab bar



# Tab Bar Controller Demo

# User Defaults

# User Defaults

- NSUserDefaults allows you to easily read and save data from anywhere in your application.
- The class provides methods for accessing common types such as floats, doubles, integers, and Booleans.
- You can also store objects of type NSData, NSString, NSNumber, NSDate, NSArray, or NSDictionary.
- NSUserDefaults is appropriate for single data values, such as user preferences.

# User Defaults Cont.

Java™ Education and Technology Services

Definition :

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

Methods:

```
-(void)setBool:(BOOL)value forKey:(NSString *)defaultName

-(BOOL)boolForKey:(NSString*)defaultName
```

- The same setters and getters are found for string, arrays, integers and floats.

# User Defaults Cont.

- The best location for initializing the user defaults is in the +initialize method.
- It is a static method called once at run time before any other method called.

# Save Custom Objects in User Defaults

Create the Custom class :

```
@interface Meal : NSObject<NSCoding ,NSSecureCoding>

    @property NSString* name;
    @property int price;

- (void) encodeWithCoder:(NSCoder *)encoder;

@end
```

Objective-C

# Save Custom Objects in User Defaults Cont

```
10 @implementation Meal
11
12 - (void) encodeWithCoder:(NSCoder *)encoder {
13     [encoder encodeObject:_name forKey:@"name"];
14     [encoder encodeInt:_price forKey:@"price"];
15 }
16
17 - (id)initWithCoder:(NSCoder *)decoder {
18     if ((self = [super init])) {
19     _name = [decoder decodeObjectOfClass:[NSString class] forKey:@"name"];
20     _price = [decoder decodeIntForKey:@"price"];
21     }
22     return self;
23 }
24
25
26 + (BOOL)supportsSecureCoding{
27
28     return YES;
29 }
30
31
32 @end
```

Objective-C

# Save Custom Objects in User Defaults Cont



## In the main class :

```
Meal* meal_1 = [Meal new];
meal_1.name = @"Pizza";
meal_1.price = 20;

Meal* meal_2 = [Meal new];
meal_2.name = @"Salad";
meal_2.price = 9;


NSMutableArray* foodArr = [NSMutableArray new];
[foodArr addObject:meal_1];
[foodArr addObject:meal_2];



// Save Meals Array In User Defaults :-


NSError *error;
NSData* archiveData =  [NSKeyedArchiver archivedDataWithRootObject:foodArr requiringSecureCoding:YES error:&error];

[defaults setObject:archiveData forKey:@"mealsArray"];
```

# Read Custom Objects  From User Defaults Cont

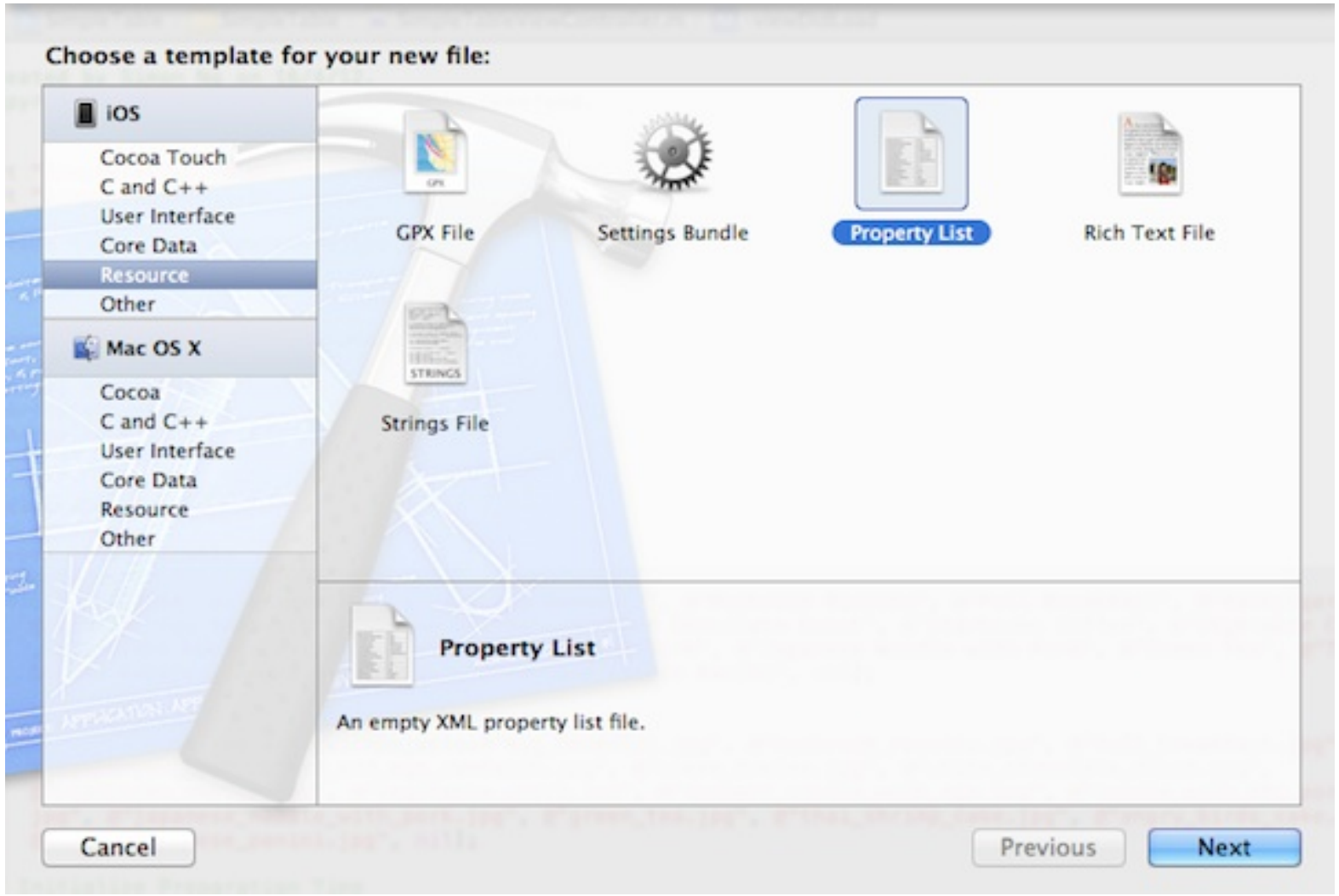
<u>In the main class :</u>

```
NSError *error;
NSData *savedData = [defaults objectForKey:@"mealsArray"];

NSSet *set = [NSSet setWithArray:@[
                      [NSArray class],
                      [Meal class],
                      ]];



NSArray< Meal*> *mealArray = (NSArray*)[NSKeyedUnarchiver unarchivedObjectOfClasses:set fromData:savedData error:&error];

for (int i = 0;i<mealArray.count;i++){


    printf("meal: %s , price = %d \n",[mealArray[i].name UTF8String],mealArray[i].price);
}


}
```

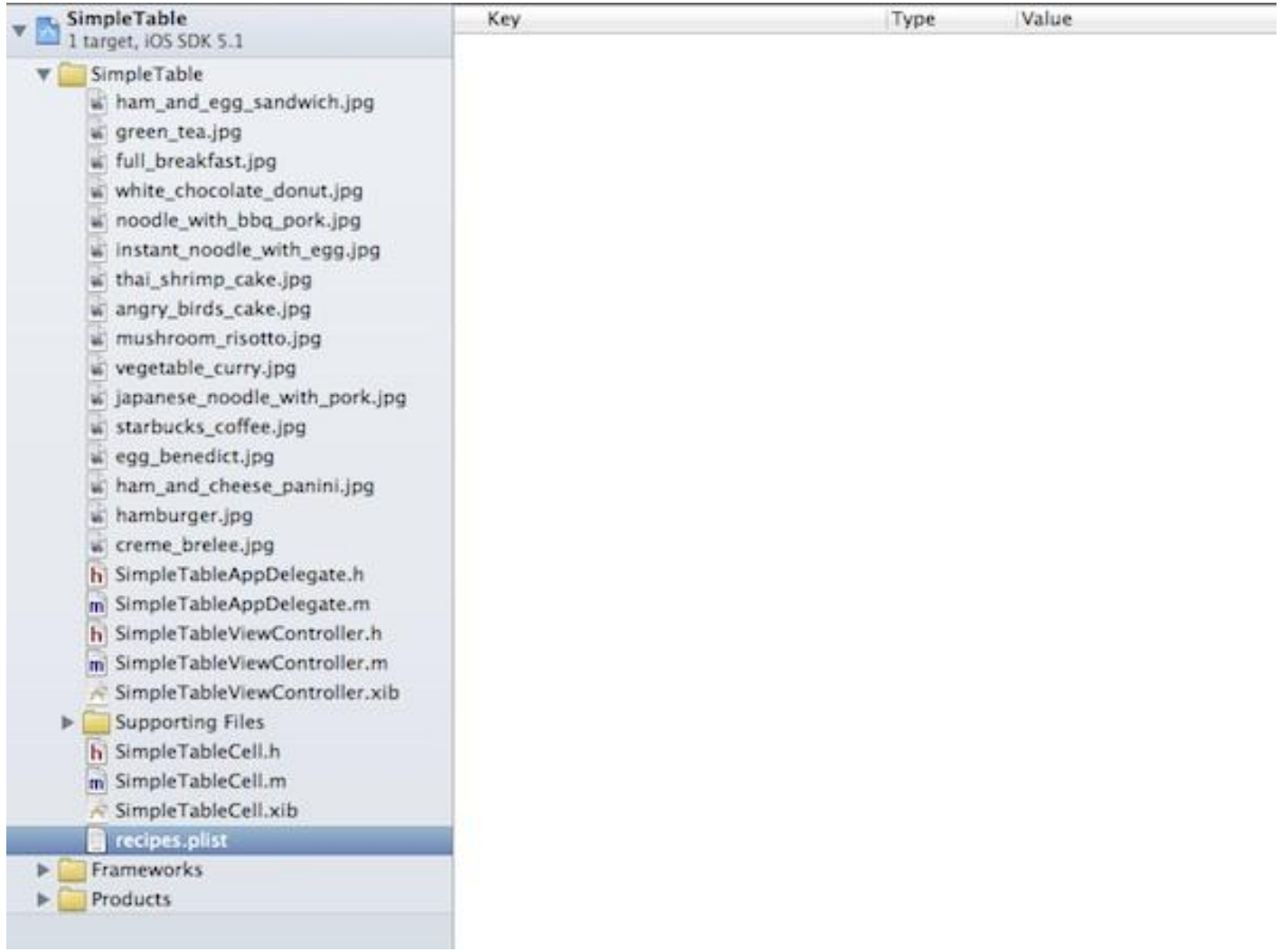
Objective-C

# PList Files

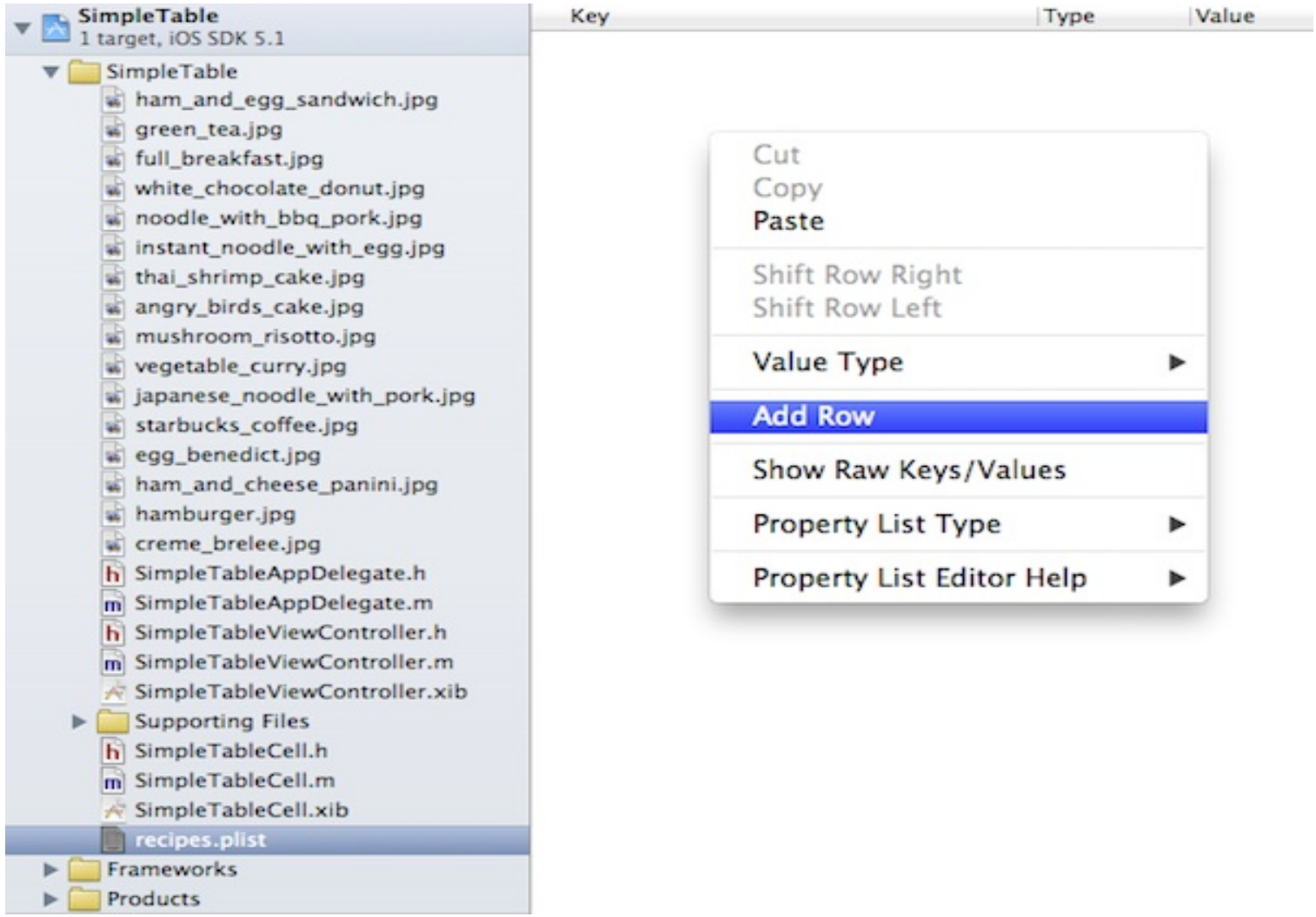
# Create Plist File



# Create Plist File Cont.



# Adding New Rows



# Adding New Rows Cont.

- We'll add three rows with "array" type. Name them with the keys: **RecipeName**, **Thumbnail** and **PrepTime**. The key serves as an identifier and later you'll use it in your code to pick the corresponding array.

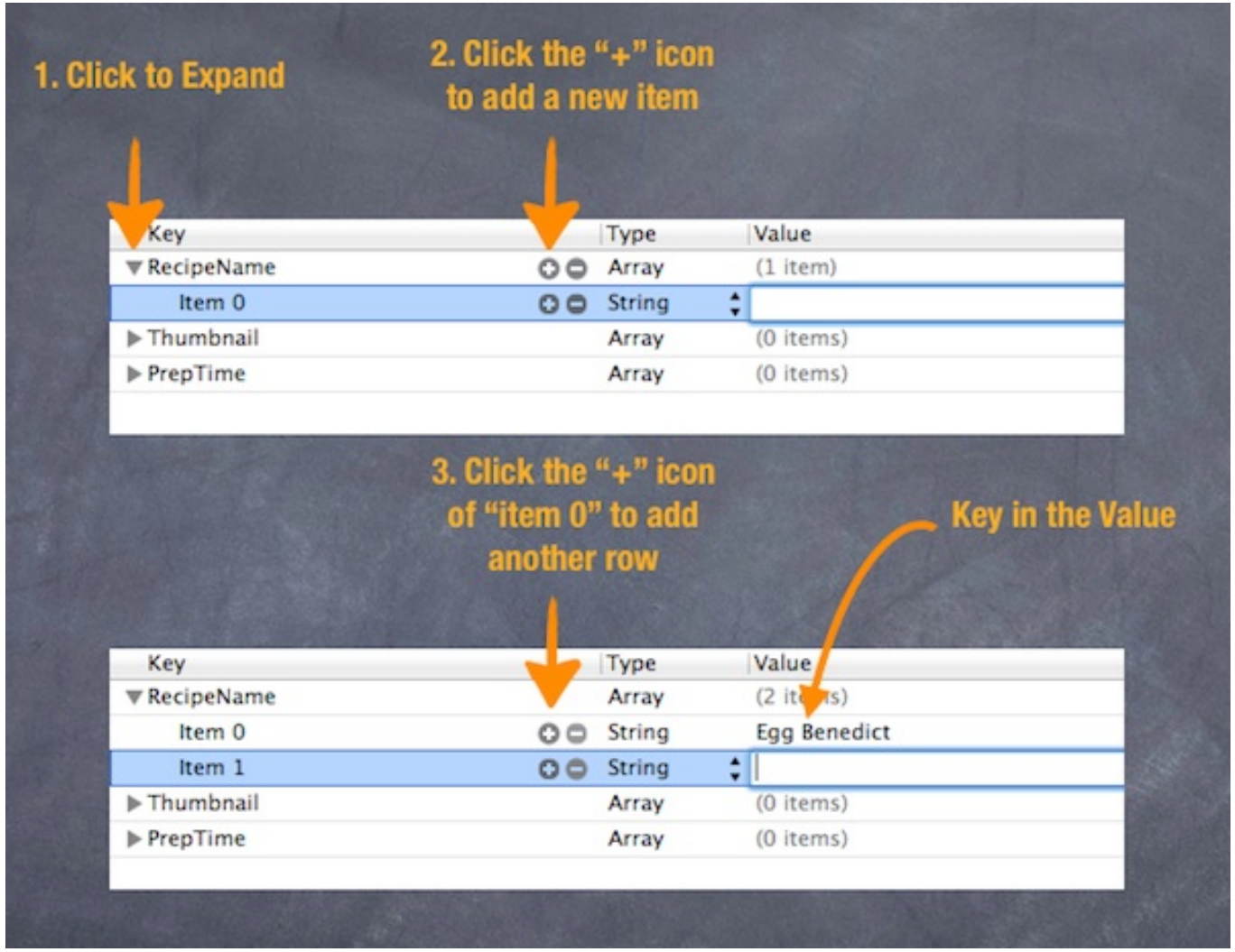
| Key | Type | Value |
|---|---|---|
| ▶ RecipeName | Array | (0 items) |
| ▶ Thumbnail | Array | (0 items) |
| ▶ PrepTime | Array | (0 items) |

# Adding New Rows Cont.

- To add data in the array, just expand it and click the “+” icon to add a new item.

  Follow the steps in the below illustration if you don’t know how to do it.

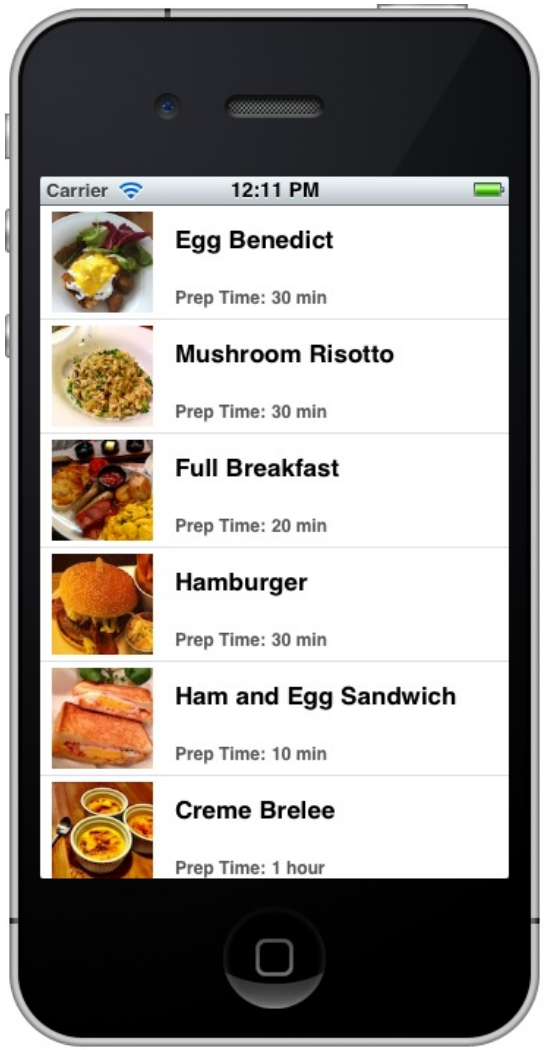
# Adding New Rows Cont.



# Loading Property List

```
// Find out the path of recipes.plist
NSString *path = [[NSBundle mainBundle] pathForResource:@"recipes"
    ofType:@"plist"];

// Load the file content and read the data into arrays
NSDictionary *dict = [[NSDictionary alloc] initWithContentsOfFile:path];
tableData = [dict objectForKey:@"RecipeName"];
thumbnails = [dict objectForKey:@"Thumbnail"];
prepTime = [dict objectForKey:@"PrepTime"];
```

# Final Result



# Create And Save Data in Property List Programmatically

```
// Create and Save Data In plist

NSError *error;
NSData* archiveData =  [NSKeyedArchiver archivedDataWithRootObject:foodArr requiringSecureCoding:YES error:&error];


NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);
NSString *documentsDirectory = [paths objectAtIndex:0];
NSString *path = [documentsDirectory stringByAppendingPathComponent:@"Menu.plist"];

if ([archiveData writeToFile:path atomically:YES]){
NSLog (@"Written");
}else{
    NSLog (@"Not Written");
    }
```

Objective-C

# Loading Data From Property List

```
//Read Data From plist

NSError *error;
NSSet *set = [NSSet setWithArray:@[
    [NSArray class],
    [Meal class],
    ]];


NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);
NSString *documentsDirectory = [paths objectAtIndex:0];
NSString *sourcePath = [documentsDirectory stringByAppendingPathComponent:@"Menu.plist"];
NSData* dataFromPlist = [NSData dataWithContentsOfFile:sourcePath];


NSArray< Meal*> *mealsArray = (NSArray*)[NSKeyedUnarchiver unarchivedObjectOfClasses:set fromData:dataFromPlist error:&error];
```

# Swipe

# Swipe

- Why swipe ??
  - Using swipes is much easier in dealing with applications

Note: Swipe should be clear and used with care.

# Configuring Swiping

- **Steps for Adding swipe gesture recognizer to the specified view:**
  - Specifying the target.
  - Setting the direction of swiping (Left , Right , Up , Down).
  - Specifying the action that is done upon swiping.

# Swipe Methods

```
UISwipeGestureRecognizer *rec = [[UISwipeGestureRecognizer alloc]
    initWithTarget:self action:@selector(next)];
// Specifing target and action


rec.direction = UISwipeGestureRecognizerDirectionLeft;
// Setting direction of swipe

[self.view addGestureRecognizer:rec];
// Adding gesture recognizer to the view
```

# Present ViewController

# Present And Dismiss

- To present a new view controller use:

```
[self presentViewController:nextController animated:YES completion:nil]
```

- To return back to the old view controller use:

```
[self dismissViewControllerAnimated:YES completion:nil];
```

# Animation

- If we want to set a certain animation for presenting or dismissing a view controller we use:
  - setModalTransitionStyle method
- Some of the values for this method are:
  - UIModalTransitionStyleFlipHorizontal
  - UIModalTransitionStyleCrossDissolve

```
[nextController setModalTransitionStyle:UIModalTransitionStyleFlipHorizontal]
```

# Completion

- If certain functionality is required directly after presenting the controller

```
[self presentViewController:nextController animated:YES completion:^(void){}]
```

- If certain functionality is required directly after dismissing the controller

```
[self dismissViewControllerAnimated:YES completion:^(void){}]
```

# Swipe Demo

# Lab Exercise

# 1.Combined Tabbed and Navigation Controller

- Create application with two tabs one of its tab has navigation controller
- Try to navigate to another view controller using navigation controller.

# 2.Login using NSUserDefaults

- Create application which enable you to register with your phone and password
- Save your data using NSUserDefaults.
- Try to sign in if your data is correct you can login otherwise it will show alert tells you that your data is incorrect

# 3.Swipe

- Create an application with multiple views and navigate through them using swiping
  - Swipe left and right through viewcontrollers