ES3 Lab 2

InterfaceBuilder and user interfaces

This lab

- InterfaceBuilder
  - Creating components
  - Linking them to your code
  - Adding buttons, labels, sliders
- UITableView
  - Creating a tableview
  - Customizing cells
- UINavigationController
  - Hierarchical table access

ASSESSED!

- THIS IS THE ASSESSED EXERCISE FOR THE iPhone SECTION OF THE COURSE!

Hand in two weeks from today

email me your source code

Individual work -- don't copy someone elses!

Feel free to use any online tutorials or references to learn more BUT don't just copy and paste code in

In the later parts of the exercise you will need to refer to the API docs and/or other resources extensively

Exercise

- Create the Vehicle Recorder and Observer app
  - Imagine a scenario where a police officer is recording suspicious vehicles...

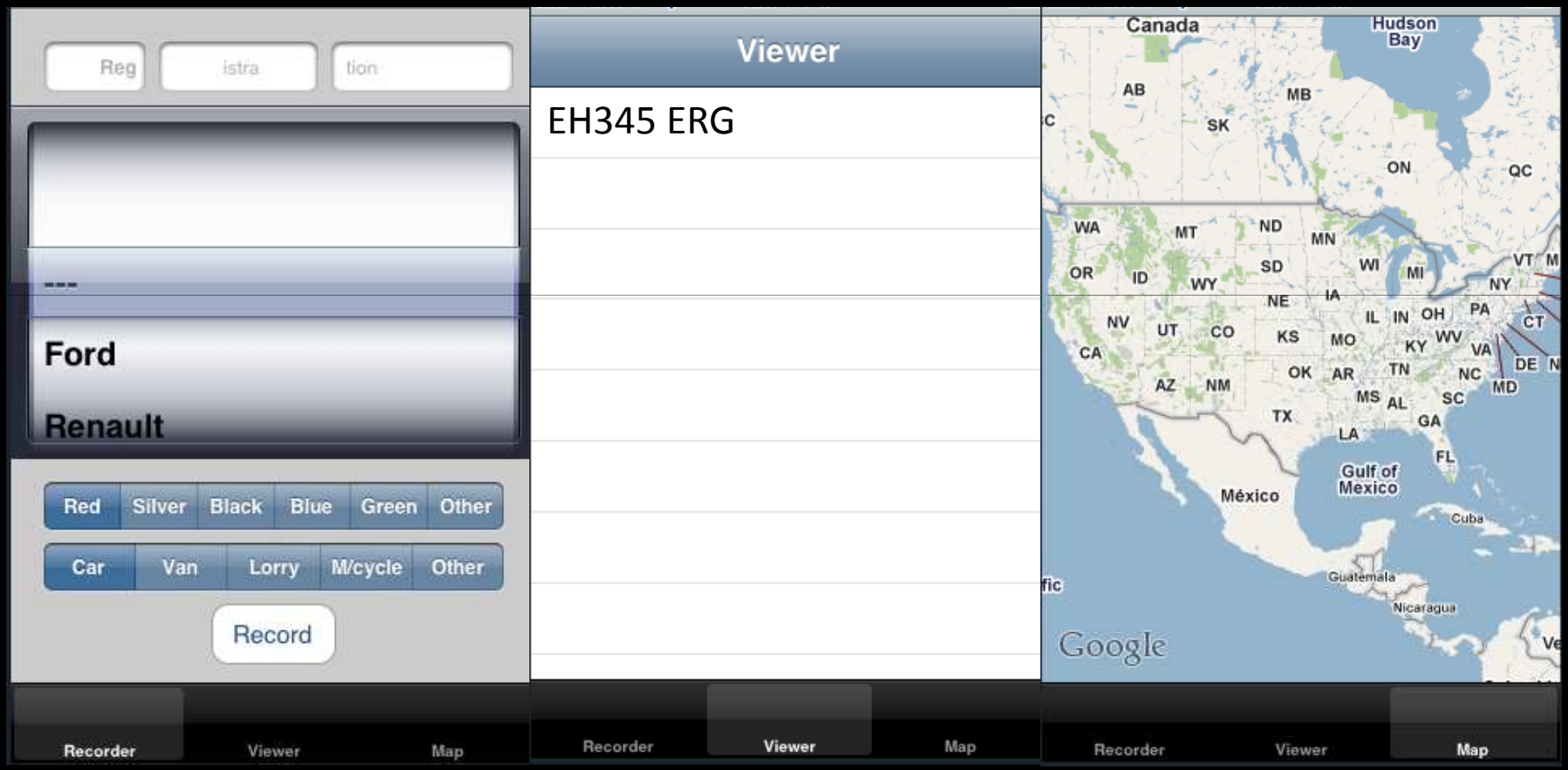
- Enter basic data about vehicles you see (color, make, registration, location, time, photo)
- Use location and timestamp, photo from device
- Store it persistently
- Show data in a hierarchical menu view and a linked map view

Guide

- You will get at least a passing mark if you manage to implement just the basic recorder, which stores data persistently and shows the registrations in the list view

- You will get full credit if:
  - the recorder works fully, including taking pictures
  - data is stored peristently
  - table view with navigation controller to select details
  - details show picture when car is selected
  - map view with annotations
  - annotations can link to the table view and vice versa
    - this last bit is tricky!

Result


Reg
istra
tion

Ford
Renault
Red
Silver
Black
Blue
Green
Other
Car
Van
Lorry
M/cycle
Other
Record
Recorder
Viewer
Map
Viewer
EH345 ERG
Recorder
Viewer
Map
Canada
Hudson Bay
Gulf of Mexico
México
Cuba
Guatemala
Nicaragua
Google
Recorder
Viewer
Map

XCode + Interface Builder

- XCode and InterfaceBuilder are seperate applications
- **Remember to save before switching between!**
- In InterfaceBuilder you can preview the interface with **File/Simulate Interfac**e, or actually build and test the app with **File/Build and Go in XCode**
- Double-click XIB files to open InterfaceBuilder

Procedure

- Construct the form for recording the data (in **InterfaceBuilder**)
- Link it to your code
- Make a button which takes a picture, reads the form and stores the data persistently
- Put the form in a tab bar

- Create a table view which lists all the vehicles recorded
- Put this table view in the tab bar
- Make it a hierarchical view with UINavigationController
- Add a table view to display the details
- Add an action when a row is selected to display the details

---

- Add a map view and put it in the tab bar
- Add annotations to the map view
- Link annotations to the detail view and vice versa

VehicleView

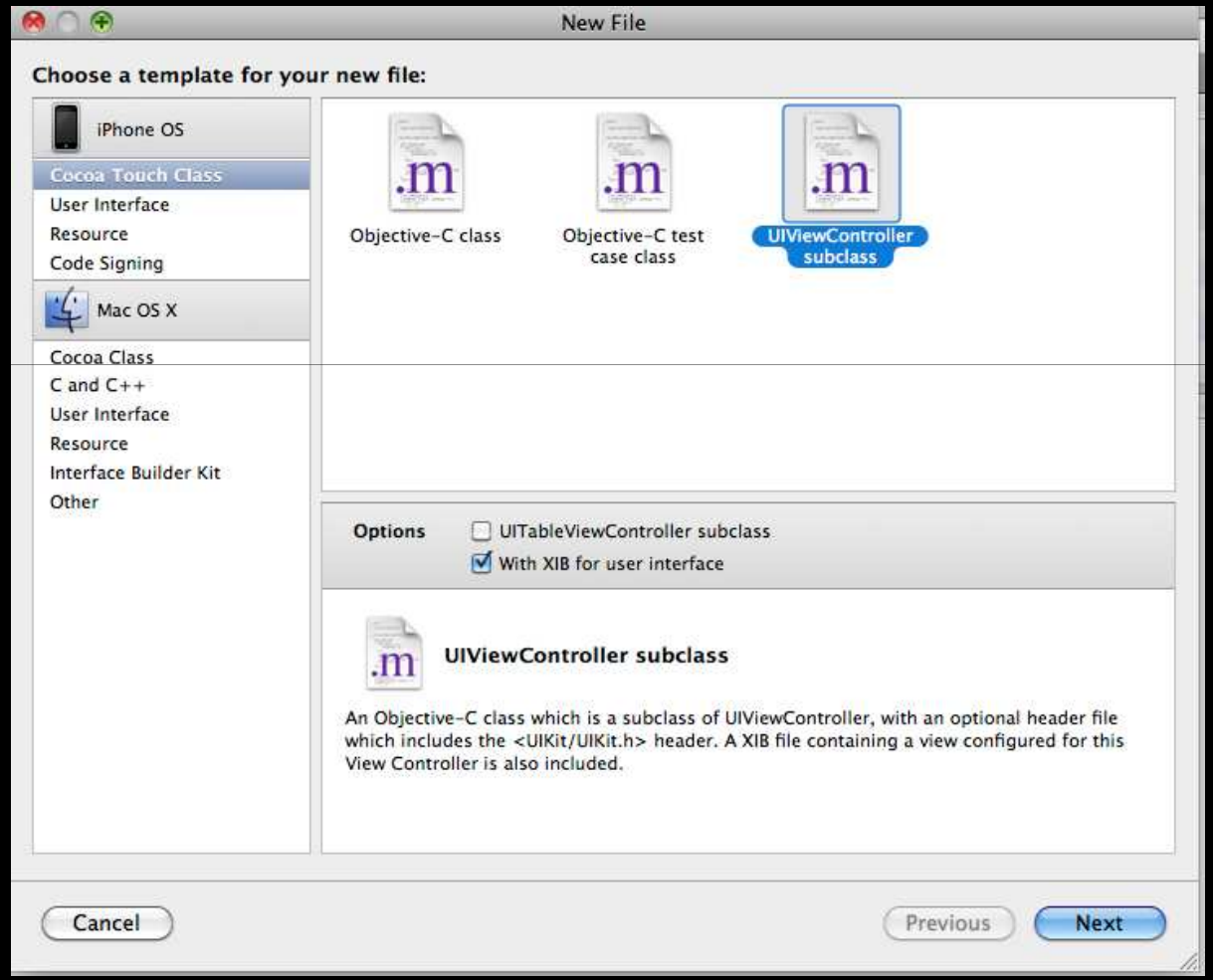
- We need a new **UIViewController** to represent our form where we will enter data about vehicles

- Use Add/New file... to add a new sublass of **UIViewController**
  - Tick the "**with XIB**" option!
  - This is ESSENTIAL -- it creates the nib file to go with the UIViewController

- Add a **VehicleView** instance to the application delegate
  - add a property and synthesize it too
- Initialise it with **initWithNibName**
  - pass the name of the nib file that XCode created for you (@"VehicleView")

```
self.vehicleView = [[[VehicleView alloc] initWithNibName:@"VehicleView" bundle:[NSBundle
mainBundle]] autorelease];
[window addSubview:self.vehicleView.view];
```

VehicleView

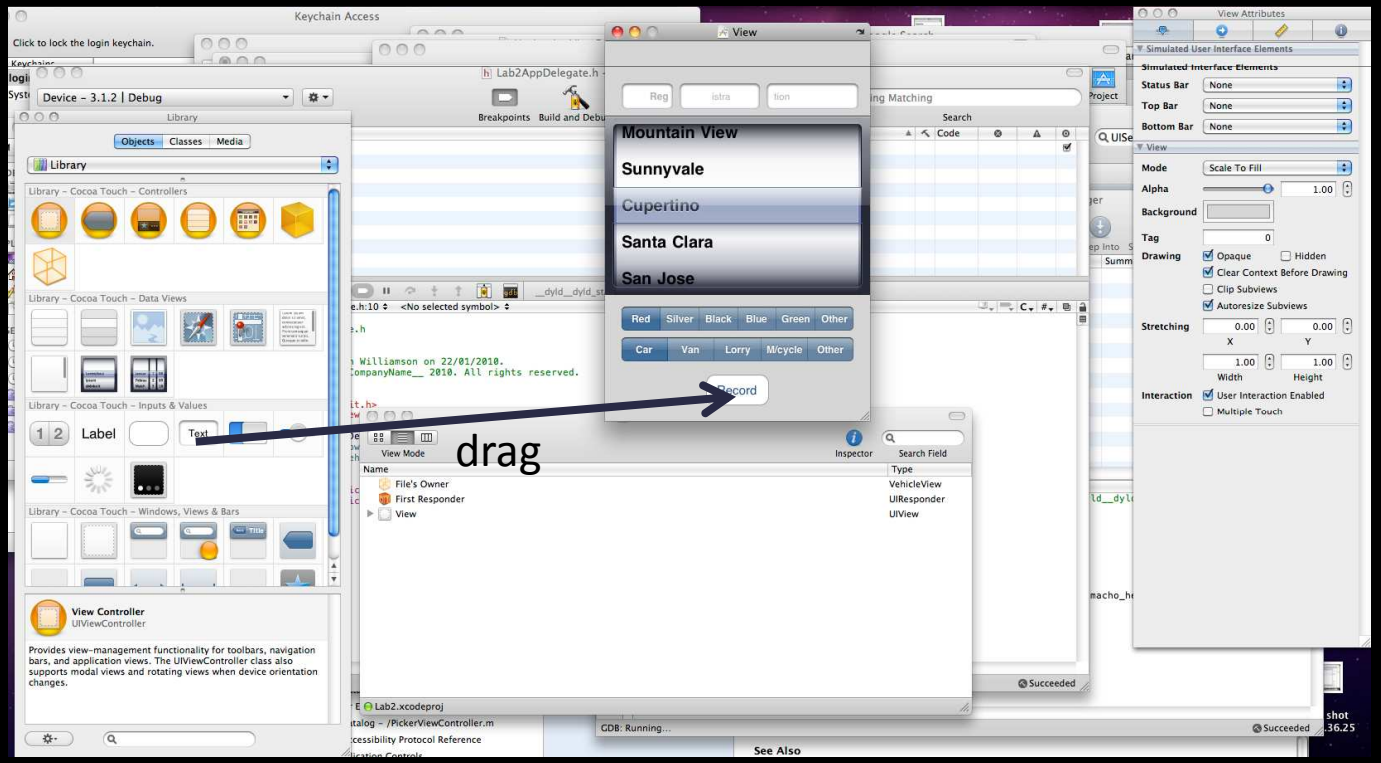


Creating a form in InterfaceBuilder

- Open VehicleView.xib
  - it will have a **File's Owner**, **First Responder** and a **View**

Adding controls

- Open the **View** by double-clicking it
- Drag and drop controls from the library window onto the view
  - if Library isn't visible, **Tools/Library** will bring it up



Creating the form

- Make the interface look like this:
  - remember, you can change visual properties using the Inspector window
  - click the tab with the "slider" icon at the top of the Inspector to set attributes

Textfields
UITextField

Picker view
UIPickerView

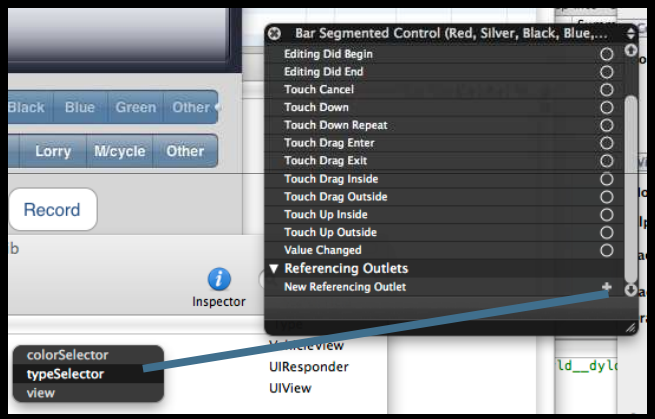
Segmented control
(bar style)
UISegmentedControl

Button
UIButton

Linking things

- Create outlets for the textfields, the picker and the segmented bar controls
- Just create variables in **VehicleView**, mark them **IBOutlet**
  - **e.g. of type UIButton * or UISegmentedControl ***
  - **IBOutlet goes before the type**

    ```
    IBOutlet UIButton *myButton;
    ```
- **Save the header file (IMPORTANT!)**



- Go back to InterfaceBuilder
  - Right-click the segmented bar control
  - Choose "New referencing outlet"
  - Drag a connection to **File's Owner**
    - remember, the owner is the **VehicleView** instance!
  - The name of the instance variable should pop up
  - Click on the connection and it should appear
- Repeat for the textfields, the picker and the other segmented bar control
- If this doesn't work, either the instance variables are of the wrong type, you didn't mark them IBOutlet, or you didn't save before switching to InterfaceBuilder

Setting up the textfields

- There are three textfields
  - One for each part of a registration number
  - This is really just to show the different possibilities
- Using the Inspector, set
  - first text field : capitalized, default keyboard, no correction
  - second text field: phone keypad
  - third text field: capitalized, default keyboard, no correction
- Set the placeholder text in the Inspector

UITextFieldDelegate

- Textfields must have delegates
- Make VehicleView conform to **UITextFieldDelegate** by changing the interface in the .h file

```
@interface VehicleView:UIViewController <UITextFieldDelegate> {
```

- Implement the **textFieldShouldReturn** as follows

```
- (BOOL) textFieldShouldReturn:(UITextField *)textField {
 [textField resignFirstResponder];
 return YES;
}
```

- In IntefaceBuilder, link all three of the textfields' **delegate** properties to **File's Owner**

- Save, build it, and check that the UI appears
  - Try entering text, check that the fields work
  - The picker will not appear!
  - It has no data source...

UIPickerDataSource

- Make VehicleView conform to **UIPickerViewDataSource** and **UIPickerViewDelegate**, as well as **UITextFieldDelegate**
- putting everything in one class like this is not recommended for larger interfaces, but it simplifies things here
- Now we need a list of cars
  - Use an **NSMutableArray**
- Create an **NSMutableArray** instance variable in **VehicleView**, make it a property and create an instance

The Data Source

- In **viewDidLoad**, something like this:

```
//capacity is just initial capacity; it will expand automatically
self.vehicleModels = [NSMutableArray arrayWithCapacity:32];
[self.vehicleModels addObject:@"---"]; // no model
[self.vehicleModels addObject:@"Ford"];
[self.vehicleModels addObject:@"Renault"];
//... etc ...
```

- Now in **VehicleView** implement
  - **numberOfComponentsInPickerView** (just return 1)
    - only one section
  - **numberOfRowsInComponent** (return [self.vehicleModels count])
    - i.e. number of vehicle models
  - **titleForRow (**return [self.vehicleModels objectAtIndex:row])
    - the title for each row is the name at that index in the array
    - note: full signature is

```
- (UIView *) pickerView:(UIPickerView *)picker titleForRow:(NSInteger)row
forComponent:(NSInteger)component;
```

Check it!

- Save, go to InterfaceBuilder and connect the dataSource and delegate properties of the picker to **File's Owner**
- Save the interface
- Build it, and check that the picker now appears correctly!

Getting a value from the controls

- To read the values from the controls in your code, there are a few useful methods and properties
  - Segmented controls store the currently selected index in the property **selectedSegmentIndex**
    - You can look up the label for a segment using **titleForSegmentAtIndex**
  - Text fields have a simple **text** property
  - Pickers have a a **selectedRowInComponent** method
    - you look up the row in the array which provides the data for the picker

Linking class data to UI data

- Create a **Vehicle** class to represent a single vehicle
  - make sure it is a subclass of **NSObject**
  - check carefully when you add the new class!

- For each entry, create a class variable for it
  - time, location, registration, color, make, type
  - make the variables of appropriate type

- When the user presses "**Record**", create a new instance, write the values from the UI to the instance variables

The record button

- Record should:
  - take a picture
  - record the time
  - record the location
  - construct a new Vehicle object with these values and the values from the form
  - add it to the array of vehicles
  - write it to disk

- Add a method to the view controller
  - mark it as **IBAction**
  - link it to the Record button's touchUpInside in **InterfaceBuilder**

Taking a photo

- Use **UIImagePickerController** to take a picture

```
//create a picker
UIImagePickerController *imagePicker = [[UIImagePickerController alloc] init];

// allow user to pan and crop image
imagePicker.allowsImageEditing = YES;

//send messages to this view controller
imagePicker.delegate = self;

// select the camera as the input source
// on the emulator use UIPickerControllerSourceTypePhotoLibrary
imagePicker.sourceType = UIPickerControllerSourceTypeCamera;

// present the picker
[self presentModalViewController:imagePicker animated:YES];
```

Photo taking

- When the user takes the photo, the message **imagePickerController:didFinishPickingMediaWithInfo** is sent to the delegate
  - Make sure you implement this method
- You **MUST** dismiss the image picker in this method

```
[picker dismissModalViewControllerAnimated:YES];
```

- The image is a UIImage instance in the info dictionary
  - in key **UIImagePickerControllerEditedImage**
- **Write it to disk**
  - find the home directory (see following)
  - construct a random filename (see following)
  - Use **UIImagePNGRepresentation** to get a **NSData** block representing the file
  - Write the **NSData** to the file

Random filename

- You can construct a random filename as follows:

```
NSString *randomName = [NSString stringWithFormat:@"image-
%X%X.png",arc4random(),arc4random()];
```

- This uses the ARC4 random number generator to create a 64 bit hex code (%X formats a number as hex)

- Remember to pass the filename to the method that records the data
  - Otherwise you will have no reference to the file!

- Note: you could just store the UIImage directly in Vehicle, but that means that all the images must be loaded when the vehicle record is created
  - It's much more efficient just to load images when they are viewed

- Use the following to get the home directory

```
NSArray *documents = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomanMask, YES);
NSString *homeDirectory = [documents objectAtIndex:0];
```

Time and location

- Use **NSDate** to get the current date and time
  - The **date** method returns the current time
- Use **CoreLocation** to get positional updates (see following instructions)

Getting location data

- Make sure you import **<CoreLocation/CoreLocation.h>**
  - Also add the CoreLocation framework to the frameworks

- Add a instance variable of **CLLocationManager** to the view controller
  - make property and synthesize
  - instantiate in **viewDidLoad** with alloc/init (remember to release!)
  - call **startUpdatingLocation**

- Now location information is available in **locationManager.location**
  - check that it's not nil before using it (it will be until the first update)

- In the simulator, this location will always return the co-ordinates of Apple HQ, so don't worry if the values seem strange

Making Vehicle serializable

- **Vehicle** must conform to the **NSCoding** protocol so it can be read and written
  - add **initWithCoder** and **encodeWithCoder** methods
  - encode and decode each member variable using **encodeObject/decodeObject** method of the coder you get passed
  - You will only be encoding or decoding objects that already conform to **NSCoding**
  - Encode each object in order, and in initWithCoder, decode each object in *exactly the same order*
  - *All you have to do is call **encodeObject/decodeObject** on each variable*

Vehicle List

- Make an **NSMutableArray** variable in the VehicleView class
  - remember to make it a property
- This will store the list of vehicle records
  - It can be serialized and deserialized from disk
- To create it, first test if there is an existing archive, and if not, create a new instance -- otherwise use the one unarchived from disk
  - **NSKeyedUnarchiver** returns nil if the archive doesn't exist
  - so try unarchiving, if it returns nil, just create a new empty array
  - If you don't do this, the archive will be reset each time you run the recorder!
- Note: the archive should be written to a file in the app's home directory

Saving the data

- After the image picker sends a message saying it has finished
  - Record the time, the location and construct the new **Vehicle** object
  - Add it to the array of **Vehicle**s
  - Write it to disk with **NSKeyedArchiver**
    - NOTE: write the *entire array* as the root object -- don't try and make a key indexed archive

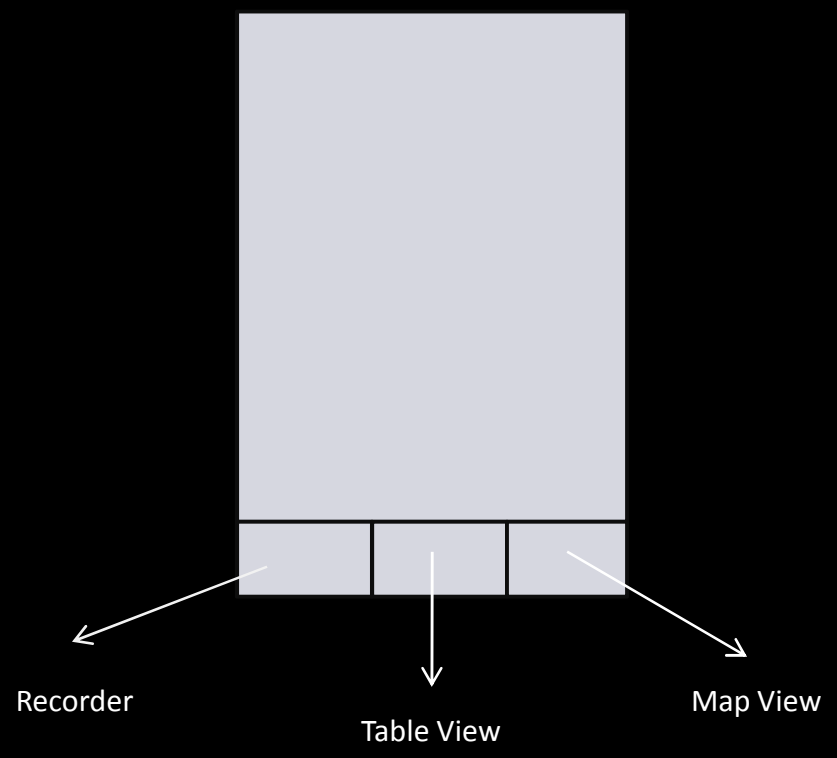
An extension...

- Try making the picker view for the vehicle manufacturer depend on the type of vehicle
  - Kawasaki don't make many cars, and Ferrari don't make many vans...

- Hints:
  - Use a **NSMutableDictionary** to link vehicle types to arrays of manufacturer names
  - Send the **UIPickerView** instance a **reloadAllComponents** message to refresh it
  - Listen to **UIControlEventValueChanged** in the segmented control

Putting it in a tab bar

- We're going to have three tabs: recorder (which we just created), viewer (a hierarchical table view) and a map view



Adding the tab bar

- In the app delegate, add an **UITabBarController** instance (property and synthesize)

- Now, originally you added **VehicleView's** view to the window in **appDidFinishLaunching**
- **Instead**
  - Create a **NSMutableArray** (not an instance variable, just locally)
  - Put **VehicleView** in it
  - Instantiate the **UITabBarController**
  - Set it's **viewControllers** property to the array containing VehicleView
  - Set **VehicleView's title** propety to @"Recorder"

- Build it, and test that the tab bar appears!
  - You may have to rearrange your form to fit

Adding a table view controller

- The second tab will have a table listing all of the registration numbers
- Create a subclass of **UITableViewController**
  - In XCode, click subclass of **UIViewController**, then tick the **UITableViewController** box at the bottom
- You need to implement
  - **numberOfSectionInTable** (return 1)
  - **cellForRowAtIndexPath** (return a cell with the text set to the registration number at that index)
  - **numberOfRowsInSection** (return the size of the Vehicle array)
- NOTE: you need to find a way of making sure the Recorder and the Table view share the same array of Vehicle objects
  - think about a clean way of doing this

Adding the controller to the tab bar

- In the app delegate
  - Create an instance variable for the new table viewer
  - in **appDidFinishLaunching**, instantiate it and add it to the tab bar array
  - Set its **title** property

- Check that the table view appears!

Making it a navigation controller

- In the app delegate, instead of directly adding the controller into the tab bar:
  - create a new instance of **UINavigationController** with the root controller being the table view
  - add this to the tab bar
- When you run it, there should now be a title

The detail table view

- Create a new subclass of **UITableViewController** to represent the details of the car
  - so when the user taps on the registration number, the detail view pops up

- Create a constructor that takes a **Vehicle** object
  - it will use this to populate the table

Pushing a new view

- Modify the table view's **didSelectRowAtIndex** so that it
  - creates a new instance of **the detail view controller** with the details of the selected vehicle
  - then pushes the new viewcontroller

```
Vehicle *vehicle = [self.vehicleRecords objectAtIndex:indexPath.row];
DetailViewController *detailViewController = [[DetailViewController alloc]
initWithVehicle:vehicle];
[self.navigationController pushViewController:detailViewController];
[detailViewController release];
```

Table layout

- Make it have two sections
- Set the title for section 0 to be Information, section 1 to Image
- In **cellForRowAtIndexPath**, use the cells **textLabel** property to set the text for the first section (color, registration, etc)
- For the second section, use the **imageView** property
  - set the **image** property to a newly constructed **UIImage** loaded from the filename
- Use the **heightForRowAtIndexPath** to make the row for section 1 (the photo) to be reasonably large (e.g. 320 pixels tall - this will make portrait photos fullscreen)
  - use **self.tableView.rowHeight** for the other rows to get the default height

Adding a map view

- Create a new view controller, without a nib file
- Add a **MKMapView** component to the controller
  - make sure you include the MapKit framework and import!
- Make sure the users position is visible
  - hint: see the lecture notes!
- Add it to the tab bar
- Check that the map appears
  - On the simulator, the map will always be centered on Cupertino...

Annotations

- Create a new class
- Make it conform to MKMapViewAnnotation
  - It needs a title, subtitle and coordinate property
- Add instances of it to the map
- You need to implement the **MKMapViewDelegate** protocol in order to return views for each pin
- You should also add a callout
  - set the **canShowCallout** property of the **MKPinAnnotationView** to YES when the delegate returns the new **MKAnnotationView**
  - set the **rightCalloutAccessoryView** to have a UIButton instance
    - **make the button select the detail page in the table view**
    - **This is tricky!**

Polish

- There's an icon in the Lab.2zip file on the web (**Icon.png**)
- Create your own **Default.png**
  - Capture the emulator window using Apple-Shift-4
  - Select just the region of the interface, excluding the top bar
    - It should be 320x460 in size
  - This will put a PNG file on your desktop
    - Rename it to default.png and put it in the project