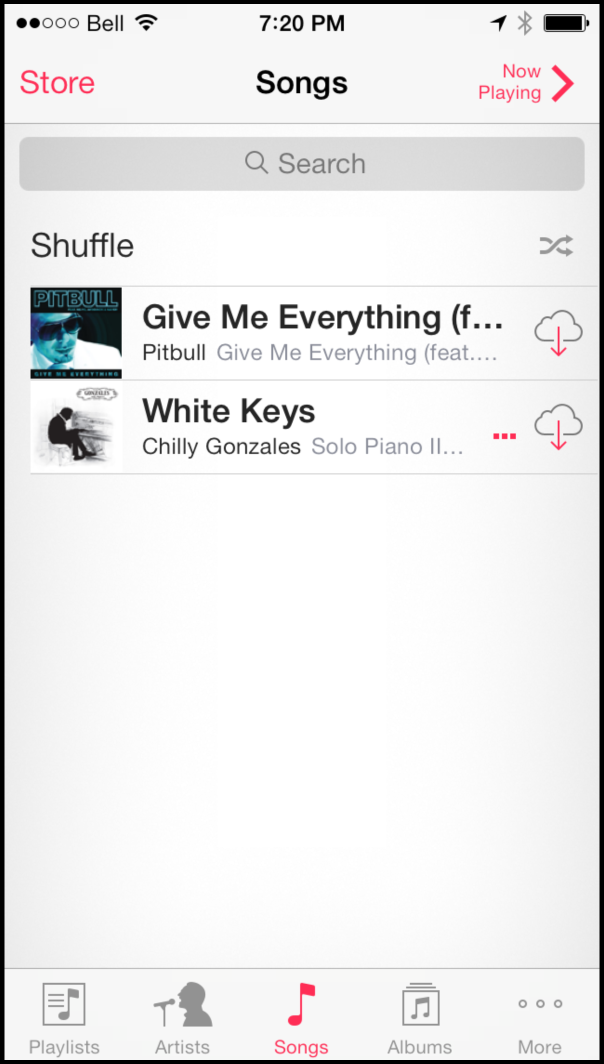
Mobile Application Development

Aileen Pierce

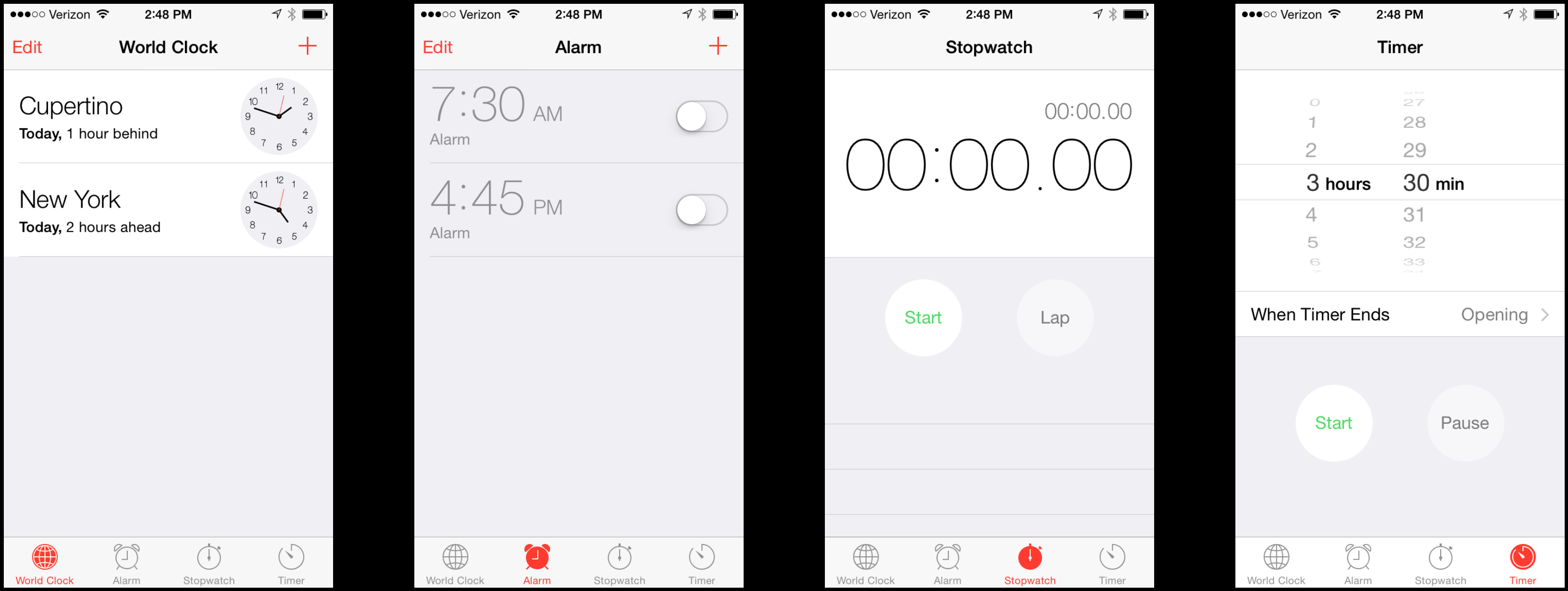
TAB BAR CONTROLLER

Tab Bar Controller

- The tab bar controller organizes multiple views in a list of tabs
- Each tab points to a view controller
- **`UITabBarController`** class

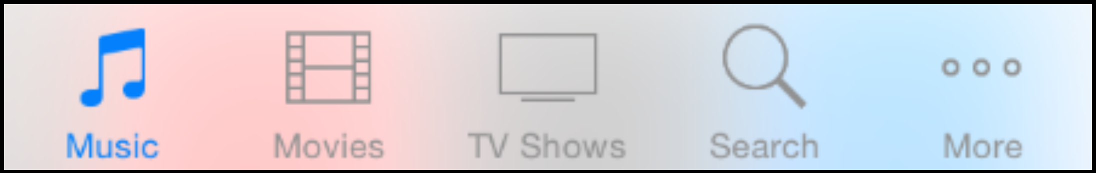


Tab Bar Controller



- Present different perspectives for data
- Access different subtasks related to the app's overall function

Tab Bar Controller

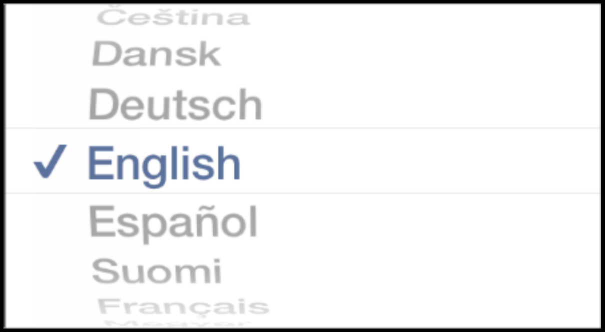


- The tab bar class is **`UITabBarItem`**
- 5 tabs are shown in horizontal compact
- When there are more tabs, the tab bar controller will automatically display a “More” tab where the rest of the items are listed

Tab Bar Controller Design

- You can customize the text for each tab
- You can use a standard image or add a custom one
  - About 25x25 pixels for 1x (max 48x32)
  - png format
  - Colors are ignored, alpha values from 0 (completely invisible) to 1 (completely visible) are used.
  - Different versions for unselected and selected(filled in)

Pickers



- A picker is a slot-machine looking UI element that is used when you have a list of values.
  - Date picker also available for date and/or time
- A picker can have multiple components (columns) that are independent or dependent
- UIPickerView class

Delegate

- The UIPickerViewDelegate protocol must be adopted

  `pickerView(_, titleForRow, forComponent)`
  - called when the picker view needs the title for a row in a component

  `pickerView(_, didSelectRow, inComponent)`
  - called by the picker view when the user selects a row in a component

Data Source

- The UIPickerViewDataSource protocol must be adopted for the picker view to display data

  `numberOfComponentsInPickerView(_)`

  - called by the picker view when it needs the number of components

  `pickerView(_, numberOfRowsInComponent)`

  - called by the picker view when it needs the number of rows for a component

Arrays

- An array stores multiple values of the same type in an ordered list.
  - Must specify the type
  - All values in the array must be of the specified type

```
var animals = [“dog”, “cat”]
```

  - Type inference creates an array of String

```
var animals = [String]()
```

  - Creates and initializes an empty array of type String

```
var animals = [String]!
```

  - Creates an empty array of type optional String

Arrays

animals[0] is “**dog**”

animals.count is 2

animals[1] = “bunny”

animals[1] is now **bunny**

animals.append(“fish”)

animals[2] is now “**fish**”

- You can easily iterate through an array

```
for pet in animals{
    print(pet)
}
```

Dictionaries

- A dictionary stores a collection of key-value pairs
  - The key is the identifier to look up the value
  - Items are not in a set order
  - Must specify the type for the keys and values

```
var teams = [“baseball” : “Rockies”,
“football” : “Broncos”, “basketball” :
“Nuggets”]
```

- Type inference creates a dictionary of [String, String]

```
var teams = [String, String]()
```

- Creates and initializes an empty dictionary of type [String, String]

```
var teams = [String, String]!
```

- Creates an empty dictionary of type optional [String, String]

Dictionaries

`teams[“baseball”]` has the **`value “Rockies”`**

`teams[“hockey”] = “Avs”` adds a key-value pair

`teams[“hockey”] = “Avalanche”` changes the value for the key “hockey”

`teams[“basketball”] = nil` removes a key-value pair

`let removedTeam = teams.removeValueForKey (“baseball”)` removes a key-value pair

– returns the removed value or nil if no value existed

Dictionaries

- You can easily iterate through a dictionary

```
for (sport, team) in teams{
    print(“The Denver \(sport) team
is \(team)”)
}
```

- Remember dictionaries are unordered so iterating over a dictionary can produce a different order each time

Arrays and Dictionaries

- .count returns the number of items
- .isEmpty is a Boolean property to check whether count is 0

`teams.count` is 4 (before removing items)

`teams.isEmpty` is false

Property Lists

- A property list is a simple data file in XML
- Use the NSBundle class to access the plist
  - An NSBundle object represents a location in the file system that groups code and resources that can be used in a program
  - One primary reason to use a bundle is to get access to the files we added to Resources.
  - **`mainBundle()`** returns a NSBundle object of our application.
  - **`pathForResource(_, ofType:)`** returns the full path name

Property Lists

- You can initialize arrays or dictionaries in your code or using a property list.
- Swift Array and Dictionary types have no way to load data from external sources.
- NSArray and NSDictionary (and the mutable versions) have initialization methods that allows you to initialize instances from the path to a property list file.

```
NSDictionary(contentsOfFile:)
```