Simple Views

Outline

1. UIView
2. UILabel
3. UIImageView
4. UIProgressView

1. UIView

- A view object renders content within its bounds rectangle and handles any interactions with that content.

- The `UIView` class is a concrete class that you can instantiate and use to display a fixed background color.

- To display labels, images, buttons, and other interface elements commonly found in apps, use the view subclasses provided by the `UIKit` framework rather than trying to define your own.

1. UIView (cont)

- Because view objects are the main way your application interacts with the user, they have a number of responsibilities. Here are just a few:
  - Drawing and animation:
    - Views draw content in their rectangular area using UIKit or Core Graphics.
    - Some view properties can be animated to new values.

1. UIView (cont)

- Because view objects are the main way your application interacts with the user, they have a number of responsibilities. Here are just a few:

  - Layout and subview management:

    - Views may contain zero or more subviews.
    - Views can adjust the size and position of their subviews.
    - Use Auto Layout to define the rules for resizing and repositioning your views in response to changes in the view hierarchy.

1. UIView (cont)

- Because view objects are the main way your application interacts with the user, they have a number of responsibilities. Here are just a few:

  - Event handling

    - A view is a subclass of UIResponder and can respond to touches and other types of events.
    -
    - Views can install gesture recognizers to handle common gestures.

1. UIView (cont)

- Creating a view from code:

```
class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        let frame = CGRect(x: 50, y: 50, width: 50, height: 50)
        let blueSquare = UIView(frame: frame)
        blueSquare.backgroundColor = .blue
        view.addSubview(blueSquare)
    }
}
```

2. UILabel

- The appearance of labels is configurable, and they can display attributed strings, allowing you to customize the appearance of substrings within a label.

- You can add labels to your interface programmatically or by using Interface Builder.

2. UILabel (cont)

- The following steps are required to add a label to your interface:
  - ➔ Supply either a string or an attributed string that represents the content.
  - ➔ If using a non-attributed string, configure the appearance of the label.
  - ➔ Set up Auto Layout rules to govern the size and position of the label in your interface.
  - ➔ Provide accessibility information and localized strings.

2. UILabel (cont)

- Creating a label from code:

```
import UIKit

class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()

        let label = UILabel()
        label.text = "Hello World!"
        label.font = UIFont.preferredFont(forTextStyle: .body)
        label.adjustsFontForContentSizeCategory = true
        label.textColor = .red
        label.backgroundColor = .yellow
        label.numberOfLines = 0
        view.addSubview(label)
    }
}
```

3. UIImageView

- Image views let you efficiently draw any image that can be specified using a `UIImage` object. `UIImageView` class can be used to display the contents of many standard image files, such as JPEG and PNG files.

- You can configure image views programmatically or in your storyboard file and change the images they display at runtime.

- For animated images, you can also use the methods of this class to start and stop the animation and specify other animation parameters.

3. UIImageView (cont)

- Creating an `UIImageView` with code:

```
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        let image = UIImage(named: "afternoon")
        let backgroundImage = UIImageView(image: image)
        backgroundImage.frame = CGRect(x: 0, y: 0, width: 100, height: 200)
        view.addSubview(backgroundImage)
    }
}
```

4. UIProgressView

- The `UIProgressView` class provides properties for managing the style of the progress bar and for getting and setting values that are pinned to the progress of a task.

- For an indeterminate progress indicator—or, informally, a “spinner”—use an instance of the `UIActivityIndicatorView` class.

4. ProgressView (cont)

- Creating an `UIProgressView` with code:

```
class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        let progressView = UIProgressView(progressViewStyle: .bar)
        progressView.center = view.center
        progressView.setProgress(0.5, animated: true)
        progressView.trackTintColor = .lightGray
        progressView.tintColor = .blue
        view.addSubview(progressView)
    }
}
```

Question & Answer?