Model-View-Controller

* The Model-View-Controller (MVC) is a design pattern or an Architecture used by many programming languages.
* The Model-View-Controller (MVC) design pattern it differentiates the objects in the following three ways to tell what kind of these objects.
  1. View
  2. Controller
  3. Model
* The Model-View-Controller (MVC) pattern not only defines the roles of objects in the application, but it also defines the way that objects communicate with each other.
* Each of the three types of objects is separated from the others by abstract boundaries and communicates with objects of the other types across those boundaries.
* In many applications the collection of objects of a certain MVC type is also referred to as a layer—for example, model layer, view layer, controller layer.
* In iOS MVC is central to a good design for a Cocoa type application and the benefits of adopting this pattern are numerous.
* By using MVC in our App it makes sure that many objects in the applications are more reusable, and their interfaces are going to be better defined.
* Those applications which are adopting the MVC design are more easily extensible than the other applications which are not adopting this design pattern.
* Moreover, many Cocoa technologies and architectures are based on MVC and require that the custom objects play one of the MVC roles as mentioned above.



Figure1: MVC Design Pattern

Courtesy: Apple Documentation

## Model Objects:

* The Model objects encapsulate the data specific to an application and define the logic and computation that manipulate and process that data. For example, a model object might represent a character in a game or a contact in an address book.
* The model object can have one to-one and one to-many relationship with other model objects, and so sometimes the model layer of an application effectively is one or more object graphs.
* In model much of the data that is part of the persistent state of the application (the persistent state can be stored in files or databases) should reside in the model objects after the data is loaded into the application.
* The model objects represent knowledge and expertise related to a specific problem domain, they can be reused in similar problem domains.
* Usually the model object should not have any explicit connection to the view objects to present its data and allow users to edit that data—it should not be concerned with user-interface and presentation issues.

**Communication**:

* Whatever user’s actions in the view layer that creates or modifies that data are communicated through a controller object and this results the creation or updating of a model object.
* When a model object changes (for example, new data is received over a network connection), it notifies to the controller object, which updates the appropriate view objects.

## View Objects

* In iOS the view object is an object in an application that users can see.
* The view object knows how to draw itself and how to respond to the users actions.
* The main use of these view objects are to display data from the application’s model objects and to enable the editing of that data.
* The view objects are typically decoupled from model objects in an MVC application.
* In any application we typically reuse and reconfigure files so view objects provides consistency between applications
* In iOS applications both the UIKit and AppKit frameworks provide collections of view classes, and Interface Builder offers dozens of view objects in its Library.

**Communication**:

* In View Objects there is direct connection between View and Model.
* Whatever data that is entered in View first it will go to Controller and then to Model.
* Whatever the data came from Model will not go directly to the View, first it will go to the controller and then to the View.
* Example: The text entered in text field of the View, first it will go to the Controller object and then to the View.

## Controller Objects

* The controller object acts as an intermediary between one or more of an application’s view objects and one or more of an models objects.
* The Controller objects are thus a conduit through which view objects learn about changes in model objects and model objects learn about changes in the view.
* The Controller objects can also perform setup and coordinating tasks for an application and manage the life cycles of other objects.

**Communication**:

* The controller object explains the users actions made in the view objects and communicates new or changed data to the model layer.
* Thus when model objects changes, then controller object communicates that new model data to the view objects so that the View can display it.

References:

* https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/TheAppLifeCycle/TheAppLifeCycle.html