16. MVVM

- In MVC – view send events to the controller , the controller then send request to model for data and perform business logic if required, model notifies controller in case of any data change and controller send data to view to present
- MVC – resulted into massive view controller which were difficult to manage
- In MVVM – view controller and view is combined to become "view"
- And one more layer is added called "viewmodel", this new layer will communicate with model and process the data and perform some business logic if required and sends final data to viewcontroller that is only required to update the UI.
- Example –
  - For a student entry form -
    - StudentViewController – will act as view to display final result on screen
    - StudentViewModel class – will act as viewmodel to perform heavy tasks and business logic
    - StudentModel – will be model which will represent the database model for student class

17. DYNAMIC DISPATCH

- *Dynamic dispatch is the process of selecting which implementation of a polymorphic operation (method or function) to call at run time.*
- For example, if a subclass overrides a method of its superclass, dynamic dispatch figures out which implementation of the method needs to be invoked, that of the subclass or that of the parent class.
- By applying the "dynamic" declaration modifier to a member of a class, you tell the compiler that dynamic dispatch should be used to access that member.
- "dynamic" declaration modifier can only be used for members of a class. Structures and enumerations don't support inheritance, which means the runtime doesn't have to figure out which implementation it needs to use.
- Piror to Swift 4, a function with "dynamic" modifier is implicitly visible to Objective-C. Meanwhile Swift 4 requires you to explicitly declare it with "@objc" attribute.
- Swift provides 2 ways to achieve dynamism: **table dispatch** and **message dispatch**.
- **table dispatch :** With this method, a class is associated with a so-called **virtual table** which comprises an array of function pointers to the real implementation corresponding to that class. Note that the vtable is constructed at compile time. Thus, there are only two additional instructions (read and jump) as compared to static dispatch. So the dispatch should be theoretically pretty fast.
- **message dispatch :** it is Objective-C that provides this mechanism. Every time an Objective-C method is called, the invocation is passed to "objc_msgSend" which handles the look ups. Unlike table dispatch, the *message passing dictionary* could be modified at runtime, enabling us to adjust the program behaviours while running.
-

19. CLOSURE

- Closures are self-contained blocks of functionality that can be passed around and used in your code.
- Closures are headless functions. Closures are functions without the func keyword and the function name. They are also known as anonymous functions.
- Syntax -

```
{ (parameter) -> returntype  in
        statement
}
Ex -
Var sayHello = {(name:String) -> String in
  return "Hello \(name)"
}
sayHello("Richa")  // Hello Richa
```

- Passing inside a function Syntax -

```
Func sayHello(name:String,  closure(String) -> void){
        closure("Hello \(name)")
}
sayHello( name: "Richa"){ name in
        print(name)      // Hello Richa
}
```

22. [unowned self] & [week self]

- The only time where you really want to use [unowned self] or [weak self] is when you would create a strong reference cycle. A strong reference cycle is when there is a loop of ownership where objects end up owning each other (maybe through a third party) and therefore they will never be deallocated because they are both ensuring that each other stick around.
- In case of a closure, you just need to realize that any variable that is referenced inside of it, gets "owned" by the closure. As long as the closure is around, those objects are guaranteed to be around. The only way to stop that ownership, is to use the [unowned self] or [weak self]. So if a class owns a closure, and that closure captures a strong reference to that class, then you have a strong reference cycle between the closure and the class. This also includes if the class owns something that owns the closure.
- If **self** could be nil in the closure use **[weak self]**.
- If **self** will never be nil in the closure use **[unowned self]**.

24. Operation & GCD

- Both are used to do any type of multithreading operation in iOS
- **GCD** is a lightweight way to represent units of work that are going to be executed concurrently. You don't schedule these units of work; the system takes care of scheduling for you. Adding dependency among blocks can be a headache. Canceling or suspending a block creates extra work for you as a developer!
- **Operation** adds a little extra overhead compared to GCD, but you can add dependency among various operations and re-use, cancel or suspend them.
- Operation and OperationQueue are built on top of GCD. As a very general rule, Apple recommends using the highest-level abstraction, then dropping down to lower levels when measurements show this is necessary.

26. OperationQueue

- A queue that regulates the execution of operations
- An operation queue executes its queued Operation objects based on their priority and readiness. After being added to an operation queue, an operation remains in its queue until it reports that it is finished with its task.
- OperationQueue is particularly powerful because it lets you control precisely how many simultaneous operations can run and what quality of service you need, while also letting you schedule work using closures. You can even ask the operation queue to wait until all its operations are finished, which makes scheduling easier.

```
let queue = OperationQueue()
        for image in images {
                queue.addOperation {
                        self.process(image)
                }
        }
queue.waitUntilAllOperationsAreFinished()
```

- You can add as many operations as you want, but they don't all get executed at the same time. Instead, OperationQueue limits the number of operations based on system conditions – if it's a more powerful device that isn't doing much right now, you'll get more operations than a less powerful device or a device that's busy with other work.
- You can override this behavior if you need something specific:
    - queue.maxConcurrentOperationCount = 4

29. Codable & Decodable in Swift 4

- *Make your data types encodable and decodable for compatibility with external representations such as JSON.*
- Codable protocol is new **protocol** introduced by **Apple** in **Swift 4** can provide Encodable and Decodable built-in feature. It will make **JSON** parsing easier.
- For below JSON –

```
{
"userId": 1,
"id": 1,
"title": "delectus aut autem",
"completed": false
}
```

- Codable Model will look like this –

```
struct User: Codable{
  var userId: Int
  var id: Int
  var title: String
  var completed: Bool
}
```

- And we can parse JSON like this –

```
do {
  //here dataResponse received from a network request
  let decoder = JSONDecoder()
  let model = try decoder.decode([User].self, from:
  dataResponse) //Decode JSON Response Data
  print(model)
 } catch let parsingError {
  print("Error", parsingError)
}
```