Dependency Injection (DI) is a design pattern used to manage dependencies between components in a way that promotes testability, modularity, and maintainability. In Swift, DI can be implemented using various techniques, including **initializer injection, property injection, method injection, and using dependency injection frameworks**.

**1. Types of Dependency Injection in Swift**

**1.1. Constructor Injection (Initializer Injection)**

This is the most common form of DI, where dependencies are passed via the initializer.

swift

CopyEdit

protocol NetworkService {

func fetchData() -> String

}

class APIService: NetworkService {

func fetchData() -> String {

return "Data from API"

}

}

class ViewModel {

private let networkService: NetworkService

init(networkService: NetworkService) {

self.networkService = networkService //constructor based injection

}

func loadData() -> String {

return networkService.fetchData()

}

}

// Usage

let apiService = APIService()

let viewModel = ViewModel(networkService: apiService)

print(viewModel.loadData()) // "Data from API"

✅ **Pros:**

* Strong dependency control
* Easy to test with mock objects
* Encourages immutability

❌ **Cons:**

* Requires explicit initialization

**1.2. Property Injection**

Dependencies are assigned via properties instead of the initializer.

swift

CopyEdit

class ViewModel {

var networkService: NetworkService?

func loadData() -> String {

return networkService?.fetchData() ?? "No data"

}

}

// Usage

let viewModel = ViewModel()

iewModel.networkService = APIService() // Inject dependency

print(viewModel.loadData()) // "Data from API"

✅ **Pros:**

* More flexible than initializer injection
* Works well for optional dependencies

❌ **Cons:**

* Dependency might be missing if not set
* Not as safe as constructor injection

**1.3. Method Injection**

Dependencies are passed directly into methods instead of being stored in properties.

swift

CopyEdit

class ViewModel {

func loadData(using networkService: NetworkService) -> String {

return networkService.fetchData()

}

}

// Usage

let viewModel = ViewModel()

let apiService = APIService()

print(viewModel.loadData(using: apiService)) // "Data from API"

✅ **Pros:**

* More explicit and flexible
* Useful when dependencies are used infrequently

❌ **Cons:**

* Can lead to method signature clutter