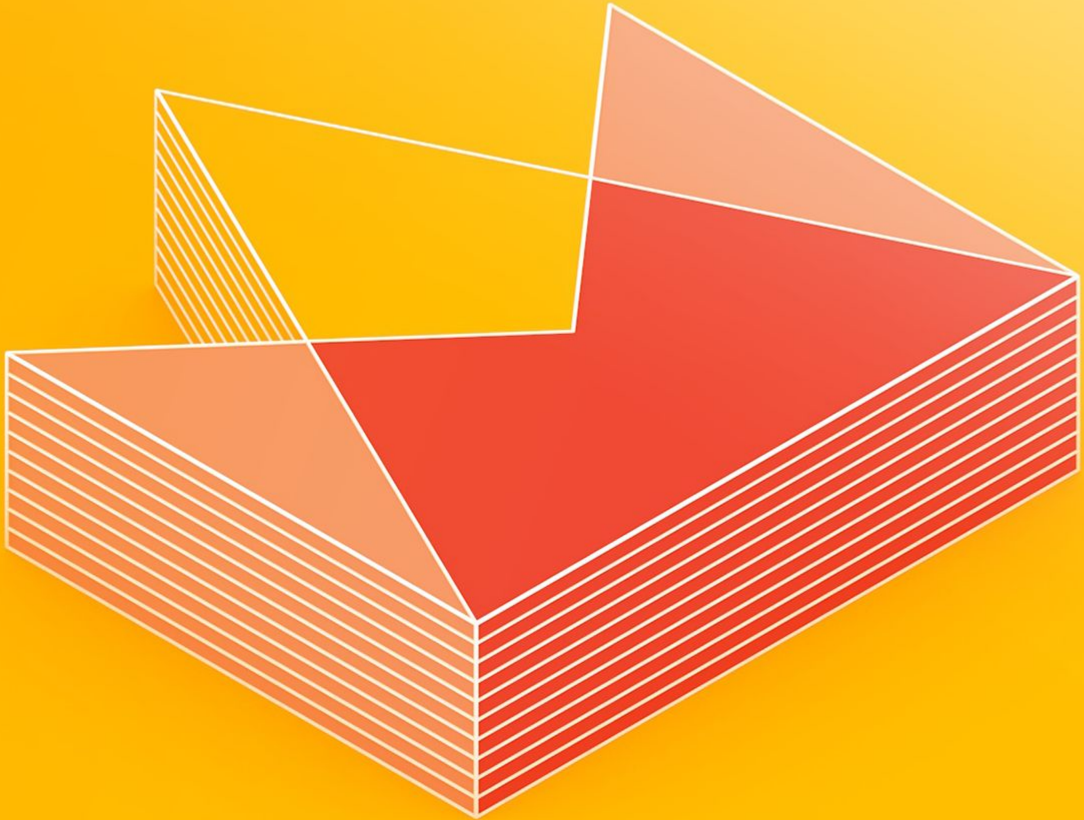
СЕЗОН КУРСІВ

2019
2020



supported by
MOC Global

Dependency Injection by Controlling the World

(inspired by pointfree.co and copied from talk by @stephencelis)

MASTERS ACADEMY

“Dependency injection is a great technique for **decoupling code** and making it easier to **test**.”

John Sundell

MASTERS ACADEMY

WHY ?

WHY NOT ?

MASTERS ACADEMY

HOW TO CONTROL THE WORLD

MASTERS ACADEMY

STEP ONE:

DESCRIBE THE WORLD

```
struct World {


}
```

```swift
struct World {
    // ???
}
```

START SMALL: **CONTROL TIME**

MASTERS ACADEMY

START SMALL: **CONTROL TIME**

```
Date() // "Nov 28, 2019 at 11:02 PM"
```

START SMALL: **CONTROL TIME**

```
Date() // "Nov 28, 2019 at 11:02 PM"
Date() // "Nov 28, 2019 at 11:03 PM"
```

START SMALL: **CONTROL TIME**

```
Date() // "Nov 28, 2019 at 11:02 PM"
Date() // "Nov 28, 2019 at 11:03 PM"
Date() // "Nov 28, 2019 at 11:04 PM"
```

```
struct World {

}
```

```swift
struct World {
  var date: () -> Date
}
```

```swift
struct World {
  var date: () -> Date = { Date() }
}
```

```swift
struct World {
  var date = { Date() }
}
```

☑️ 1. DESCRIBE THE WORLD

```swift
struct World {
  var date = { Date() }
}
```

STEP TWO:

CREATE THE WORLD

```swift
struct World {
  var date = { Date() }
}
```

```
struct World {
  var date = { Date() }
}
var Current = World()
```

✅ 1. DESCRIBE THE WORLD

```
struct World {
  var date = { Date() }
}
```

✅ 2. CREATE THE WORLD

```
var Current = World()
```

MASTERS ACADEMY

HOW TO CONTROL THE WORLD

MASTERS ACADEMY

```
Current.date() // "Nov 28, 2019 at 11:02 PM"
Current.date() // "Nov 28, 2019 at 11:02 PM"

// Send the world back in time!
Current.date = { .distantPast }
Current.date() // "Nov 28, 2019 at 11:02 PM"

// Send the world back in time!
Current.date = { .distantPast }
Current.date() // "Jan 1, 1 at 2:02 AM"
```

MASTERS ACADEMY

```
Current.date() // "Nov 28, 2019 at 11:02 PM"

// Send the world back in time!
Current.date = { .distantPast }
Current.date() // "Jan 1, 1 at 2:02 AM"
 // Or into the future!
Current.date = { .distantFuture }
```

MASTERS ACADEMY

```
Current.date() // "Nov 28, 2019 at 11:02 PM"

// Send the world back in time!
Current.date = { .distantPast }
Current.date() // "Jan 1, 1 at 2:02 AM"
 // Or into the future!
Current.date = { .distantFuture }
Current.date() // Jan 1, 4001 at 2:00 AM"
```

MASTERS ACADEMY

```
Current.date() // "Nov 28, 2019 at 11:02 PM"

// Send the world back in time!
Current.date = { .distantPast }
Current.date() // "Jan 1, 1 at 2:02 AM"
 // Or into the future!
Current.date = { .distantFuture }
Current.date() // Jan 1, 4001 at 2:00 AM"
Current.date() // Jan 1, 4001 at 2:00 AM"
```

MASTERS ACADEMY

```
Current.date() // "Nov 28, 2019 at 11:02 PM"

// Send the world back in time!
Current.date = { .distantPast }
Current.date() // "Jan 1, 1 at 2:02 AM"
 // Or into the future!
Current.date = { .distantFuture }
Current.date() // Jan 1, 4001 at 2:00 AM"
Current.date() // Jan 1, 4001 at 2:00 AM"
Current.date() // Jan 1, 4001 at 2:00 AM"
```

MASTERS ACADEMY

```
Current.date() // "Nov 28, 2019 at 11:02 PM"

// Send the world back in time!
Current.date = { .distantPast }
Current.date() // "Jan 1, 1 at 2:02 AM"
 // Or into the future!
Current.date = { .distantFuture }
Current.date() // Jan 1, 4001 at 2:00 AM"
Current.date() // Jan 1, 4001 at 2:00 AM"
Current.date() // Jan 1, 4001 at 2:00 AM"

// Restore the balance
Current.date = Date.init
Current.date() // "Nov 30, 2019 at 11:59 PM"
```

MASTERS ACADEMY

HOW TO CONTROL THE WORLD

MASTERS ACADEMY

FIND-AND-REPLACE

Wherever we see:

```
Date()
```

Replace with:

```
Current.date()
```

```swift
func application(_ application: UIApplication,
              didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?)
           -> Bool {

  Current.date = { Date.init(timeIntervalSinceReferenceDate: 0) }

  return true
}
```

```swift
struct World {
  var date = { Date() }
}
```

```
struct World {
  var date = { Date() }
}
```

```
let formatter = DateFormatter()
formatter.string(from: Current.date())
```

```swift
let formatter = DateFormatter()
formatter.calendar // Calendar
formatter.locale // Locale
formatter.timeZone // TimeZone
formatter.string(from: Current.date())
```

LET’S TAKE **CONTROL!**

```
struct World {
  var date = { Date() }
}
```

LET'S TAKE **CONTROL!**

```
struct World {
  var calendar = Calendar.autoupdatingCurrent
  var date = { Date() }
  var locale = Locale.autoupdatingCurrent
  var timeZone = TimeZone.autoupdatingCurrent
}
```

MASTERS ACADEMY

FIND-AND-REPLACE

Wherever we see:

```
Calendar.autoupdatingCurrent
Locale.autoupdatingCurrent
TimeZone.autoupdatingCurrent
```

Replace with:

```
Current.calendar
Current.locale
Current.timeZone
```

LET'S TAKE **CONTROL!**

```
let formatter = DateFormatter()

formatter.calendar = Current.calendar
formatter.locale = Current.locale
formatter.timeZone = Current.timeZone

formatter.string(from: Current.date())
```

MASTERS ACADEMY

LET'S TAKE **CONTROL!**

```
extension World {
  func dateFormatter() -> DateFormatter {
    let formatter = DateFormatter()
      formatter.calendar = self.calendar
      formatter.locale = self.locale
      formatter.timeZone = self.timeZone
      return formatter
    }
}

Current.dateFormatter()
```

MASTERS ACADEMY

LET'S TAKE **CONTROL!**

```
Current.dateFormatter().string(from: Current.date())
// "September 13, 2018 at 5:00 PM"
Current.calendar = Calendar(identifier: .buddhist)
Current.locale = Locale(identifier: "es_ES")
Current.timeZone = TimeZone(identifier: "Pacific/Honolulu")!
Current.dateFormatter().string(from: Current.date())
// "13 de septiembre de 2561 BE, 17:00"
```

```
func application(_ application: UIApplication,
                 didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?)
            -> Bool {

   Current.calendar = Calendar(identifier: .buddhist)
   Current.locale = Locale(identifier: "es_ES")
   Current.timeZone = TimeZone(identifier: "Pacific/Honolulu")!

  return true
}
```

```swift
struct World {
  var calendar = Calendar.autoupdatingCurrent
  var date = { Date() }
  var locale = Locale.autoupdatingCurrent
  var timeZone = TimeZone.autoupdatingCurrent
}
```

```
APIClient.shared.token = token
APIClient.shared.fetchCurrentUser { result in
  // ...
}
```

```swift
APIClient.shared.token = token
APIClient.shared.fetchCurrentUser { result in
  // ...
}

struct API {
  var setToken = { APIClient.shared.token = $0 }
  var fetchCurrentUser = APIClient.shared.fetchCurrentUser
}
```

MASTERS ACADEMY

```swift
APIClient.shared.token = token
APIClient.shared.fetchCurrentUser { result in
  // ...
}

struct API {
  var setToken = { APIClient.shared.token = $0 }
  var fetchCurrentUser = APIClient.shared.fetchCurrentUser
}

struct World {
  var api = API()
  // ...
}
```

FIND-AND-REPLACE

Wherever we see:

```
APIClient.shared.token = token
APIClient.shared.fetchCurrentUser { result in
```

Replace with:

```
Current.api.setToken(token)
Current.api.fetchCurrentUser { result in
```

```swift
// Simulate being logged-in as a specific user
Current.api.fetchCurrentUser = {
  callback in callback(.success(User(name: "Blob")))
}
```

MASTERS ACADEMY

```
// Simulate being logged-in as a specific user
Current.api.fetchCurrentUser = {
  callback in callback(.success(User(name: "Blob")))
}

// Simulate specific errors
Current.api.fetchCurrentUser = { callback in
  callback(.failure(APIError.userSuspended))
}
```

MASTERS ACADEMY

THIS **IS NOT** HOW WE DO THINGS

AREN'T SINGLETONS EVIL?

AREN'T SINGLETONS EVIL?

— singletons are only a problem when they're out of our control

WHAT ABOUT GLOBAL `MUTATION`?

WHAT ABOUT GLOBAL `MUTATION`?

— the **option** to mutate, not the requirement (avoid mutation in release mode)

WHAT ABOUT GLOBAL `MUTATION`?

— the **option** to mutate, not the requirement (avoid mutation in release mode)
— exercise restraint (with code review and lint checks)

WHAT ABOUT GLOBAL `MUTATION?`

— the **option** to mutate, not the requirement (avoid mutation in release mode)
— exercise restraint (with code review and lint checks)

```
# .swiftlint.yml
custom_rules:
  no_current_mutation:
    included: ".*\\.swift"
    excluded: ".*Tests\\.swift"
    name: "Current Mutation"
    regex: '(Current\.\S+\s+=)'
    message: "Don't mutate the current environment"
```

WHY STRUCTS?

— protocols can be a premature abstraction
— protocols require a **ton** of boilerplate

```swift
protocol APIClientProtocol {
  var token: String? { get set }
  func fetchCurrentUser(_ @escaping completionHandler: (Result<User, Error>) -> Void)
}
```

```
protocol APIClientProtocol {
  var token: String? { get set }
  func fetchCurrentUser(_ @escaping completionHandler: (Result<User, Error>) -> Void)
}

extension APIClient: APIClientProtocol {}
```

```
protocol APIClientProtocol {
  var token: String? { get set }
  func fetchCurrentUser(_ @escaping completionHandler: (Result<User, Error>) -> Void)
}

extension APIClient: APIClientProtocol {}

class MockAPIClient: APIClientProtocol {
  var token: String?

  var currentUserResult: Result<User, Error>?
  func fetchCurrentUser(_ completionHandler: (Result<User, Error>) -> Void) {
    completionHandler(self.fetchCurrentUserResult!)
  }
}
```

```swift
protocol APIClientProtocol {
  var token: String? { get set }
  func fetchCurrentUser(_ @escaping completionHandler: (Result<User, Error>) -> Void)
}

extension APIClient: APIClientProtocol {}

class MockAPIClient: APIClientProtocol {
  var token: String?

  var currentUserResult: Result<User, Error>?
  func fetchCurrentUser(_ completionHandler: (Result<User, Error>) -> Void) {
    completionHandler(self.fetchCurrentUserResult!)
  }
}

struct World {
  var api: APIClientProtocol = APIClient.shared
}
```

```swift
struct API {
  var setToken = { APIClient.shared.token = $0 }
  var fetchCurrentUser = APIClient.shared.fetchCurrentUser
}

struct World {
  var api = API()
}
```

```
protocol APIClientProtocol {
  var token: String? { get set }
  func fetchCurrentUser(_ @escaping completionHandler: (Result<User, Error>) -> Void)
}

extension APIClient: APIClientProtocol {}

class MockAPIClient: APIClientProtocol {
  var token: String?

  var currentUserResult: Result<User, Error>?
  func fetchCurrentUser(_ completionHandler: (Result<User, Error>) -> Void) {
    completionHandler(self.fetchCurrentUserResult!)
  }
}

struct World {
  var api: APIClientProtocol = APIClient.shared
}
```

WHY STRUCTS?

— protocols can be a premature abstraction
— protocols require a **ton** of boilerplate

ISN'T DEPENDENCY INJECTION BETTER?

ISN’T DEPENDENCY INJECTION BETTER?

— passing dependencies requires a **lot** more boilerplate

```swift
class MyViewController: UIViewController {
  let api: APIClientProtocol
  let date: () -> Date
  let label = UILabel()

  init(_ api: APIClientProtocol, _ date: () -> Date) {
    self.api = api
    self.date = date
  }

  func greet() {
    self.api.fetchCurrentUser { result in
      if let user = result.success {
        self.label.text = "Hi, \(user.name)! It’s \(self.date())."
      }
    }
  }
}
```

```swift
class MyViewController: UIViewController {
  let api: APIClientProtocol
  let date: () -> Date

  init(_ api: APIClientProtocol, _ date: () -> Date) {
    self.api = api
    self.date = date
  }

  func presentChild() {
    let childViewController = ChildViewController(
      api: self.api, date: self.date
    )
  }
}

class ChildViewController: UIViewController {
  let api: APIClientProtocol
  let date: () -> Date
  let label = UILabel()

  init(_ api: APIClientProtocol, _ date: () -> Date) {
    self.api = api
    self.date = date
  }

  func greet() {
    self.api.fetchCurrentUser { result in
      if let user = result.success {
        self.label.text = "Hi, \(user.name)! It's \(self.date())."
      }
    }
  }
}
```

MASTERS ACADEMY

```swift
protocol APIClientProvider {
  var api: APIClientProtocol { get }
}
protocol DateProvider {
  func date() -> Date
}

extension World: APIClientProvider, DateProvider {}

class MyViewController: UIViewController {
  typealias Dependencies = APIClientProvider & DateProvider

  let label = UILabel()
  let dependencies: Dependencies

  init(dependencies: Dependencies) {
    self.dependencies = dependencies
  }

  func greet() {
    self.dependencies.api.fetchCurrentUser { result in
      if let user = result.success {
        self.label.text = "Hi, \(user.name)! It's \(self.dependencies.date())."
      }
    }
  }
}
```

MASTERS ACADEMY

```
// UPD: Xcode 11
class MyViewController: UIViewController {
  typealias Dependencies = APIClientProvider & DateProvider

  var dependencies: Dependencies!

  override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "child" {
      let childViewController = segue.destinationViewController as! ChildViewController
      childViewController.dependencies = self.dependencies }
  }
}

class ChildViewController: UIViewController {
  typealias Dependencies = APIClientProvider & DateProvider
  var dependencies: Dependencies!
  @IBOutlet var label: UILabel!
  func greet() {
    self.dependencies.api.fetchCurrentUser { result in
      if let user = result.success {
        self.label.text = "Hi, \(user.name)! It’s \(self.dependencies.date())."
      }
    }
  }
}
```

WITH Current:

```
class MyViewController: UIViewController {}

class ChildViewController: UIViewController {
  @IBOutlet var label: UILabel!
  func greet() {
    Current.api.fetchCurrentUser { result in
      if let user = result.success {
        self.label.text = "Hi, \(user.name)! It’s \(Current.date())."
      }
    }
  }
}
```

MASTERS ACADEMY

```swift
// UPD: Xcode 11
class MyViewController: UIViewController {
  typealias Dependencies = APIClientProvider & DateProvider

  var dependencies: Dependencies!

  override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "child" {
      let childViewController = segue.destinationViewController as! ChildViewController
      childViewController.dependencies = self.dependencies }
  }
}

class ChildViewController: UIViewController {
  typealias Dependencies = APIClientProvider & DateProvider
  var dependencies: Dependencies!
  @IBOutlet var label: UILabel!
  func greet() {
    self.dependencies.api.fetchCurrentUser { result in
      if let user = result.success {
        self.label.text = "Hi, \(user.name)! It's \(self.dependencies.date())."
      }
    }
  }
}
```

MASTERS ACADEMY

WITH Current:

```swift
class MyViewController: UIViewController {}

class ChildViewController: UIViewController {
  @IBOutlet var label: UILabel!
  func greet() {
    Current.api.fetchCurrentUser { result in
      if let user = result.success {
        self.label.text = "Hi, \(user.name)! It’s \(Current.date())."
      }
    }
  }
}
```

MASTERS ACADEMY

GUIDELINES FOR KEEPING IT SIMPLE

1. singletons can be good (when there’s only one and you can control it)
2. global mutation can be good (when you’re not using it in production)
3. sometimes, you don’t need a protocol, and a struct can save you a ton of boilerplate
4. dependency injection is maybe more complicated of a solution than what we need

NEXT STEPS?

NEXT STEPS?

```swift
class TestCase: XCTestCase {
  override func setUp() {
    super.setUp()
    Current = World(
      api: Api(
        setToken: { _ in },
        fetchCurrentUser: { callback in
          callback(.success(User(name: "Blob"))
         }
       ,
      calendar: Calendar(identifier: .gregorian),
      date: { Date(timeIntervalSinceReferenceDate: 0) }
      locale: Locale(identifier: "en_US"),
      timeZone: TimeZone(identifier: "UTC")!
    )
  }
}
```

MASTERS ACADEMY

```swift
extension API {
  static let mock = API(
    setToken: { _ in },
    fetchCurrentUser: { callback in
      callback(.success(User(name: "Blob")))
     }
  )
}
extension World {
  static let mock = World(
    api: .mock,
    calendar: Calendar(identifier: .gregorian),
    date: { Date(timeIntervalSinceReferenceDate: 0) }
    locale: Locale(identifier: "en_US"),
    timeZone: TimeZone(identifier: "UTC")!
  )
}
```

```swift
class TestCase: XCTestCase {
  override func setUp() {
    super.setUp()
    Current = .mock
  }
}
```

NEXT STEPS?

MASTERS ACADEMY

TESTING ANALYTICS

```
struct World {
  var track = Analytics.shared.track
}
class TestCase: XCTestCase {
  var events: [Analytics.Event] = []
  override func setUp() {
    super.setUp()
    Current = .mock
    Current.track = events.append
}
  func testLoggingIn() {
    // ...
    XCTAssertEqual([.loginStart, .loginSuccess], self.events)
  }
}
```

MASTERS ACADEMY

TESTING LOCALIZATION

```
struct World {
  var preferredLanguages = Locale.preferredLanguages
}
func localizedString(key: String, value: String) -> String {
  // ...
}
```

IT CAN'T ALL BE THAT SIMPLE!

IT CAN’T ALL BE THAT SIMPLE!

— more complicated dependencies, like those following the delegate pattern, may require adopting simpler wrappers

IT CAN’T ALL BE THAT SIMPLE!

— more complicated dependencies, like those following the delegate pattern, may require adopting simpler wrappers

— ephemeral/local dependencies (like view controls and view delegates) shouldn’t be controlled on the world

IN CONCLUSION…

CONTROLLING THE WORLD IS GOOD

— unlock the ability to simulate external state

CONTROLLING THE WORLD IS SIMPLE

— no need for the excessive boilerplate of protocols and dependency injection: store the minimal details of the world in a struct

USEFUL LINKS

How to Control the World (Stephen Celis)

presentation

video

Dependency Injection Made Easy (Point-Free)

video + transcript

Dependency Injection Made Comfortable (Point-Free)

video + transcript

The Two Sides of Writing Testable Code (Brandon Williams)

video + presentation + transcript

MASTERS ACADEMY

Follow Us

#WeAreMA

masters_academy__

cherkasy.masters

Got Questions