Dependency Injection

•••

Adithya Reddy
iOS engineer at UrbanClap

What. Why. How.

What is dependency Injection**?**

It’s a fancy name for a fairly simple concept.

Dependency injection

From Wikipedia, the free encyclopedia

In software engineering, **dependency injection** is a technique whereby one object supplies the dependencies of another object. A dependency is an object that can be used (a service). An injection is the passing of a dependency to a dependent object (a client) that would use it. The service is made part of the client's state.[1] Passing the service to the client, rather than allowing a client to build or find the service, is the fundamental requirement of the pattern.

Is DI used in this code snippet below?

```
APICommunicator.default.request("GET", url, parameters: nil, headers: nil, completion: { _ in })
```

NO

What about this?

```
func set(_ event: Event) {
    _nameLabel.text = event.name
    _descriptionLabel.text = event.description

    _dueLabel.isHidden = event.date > Date()
}
```

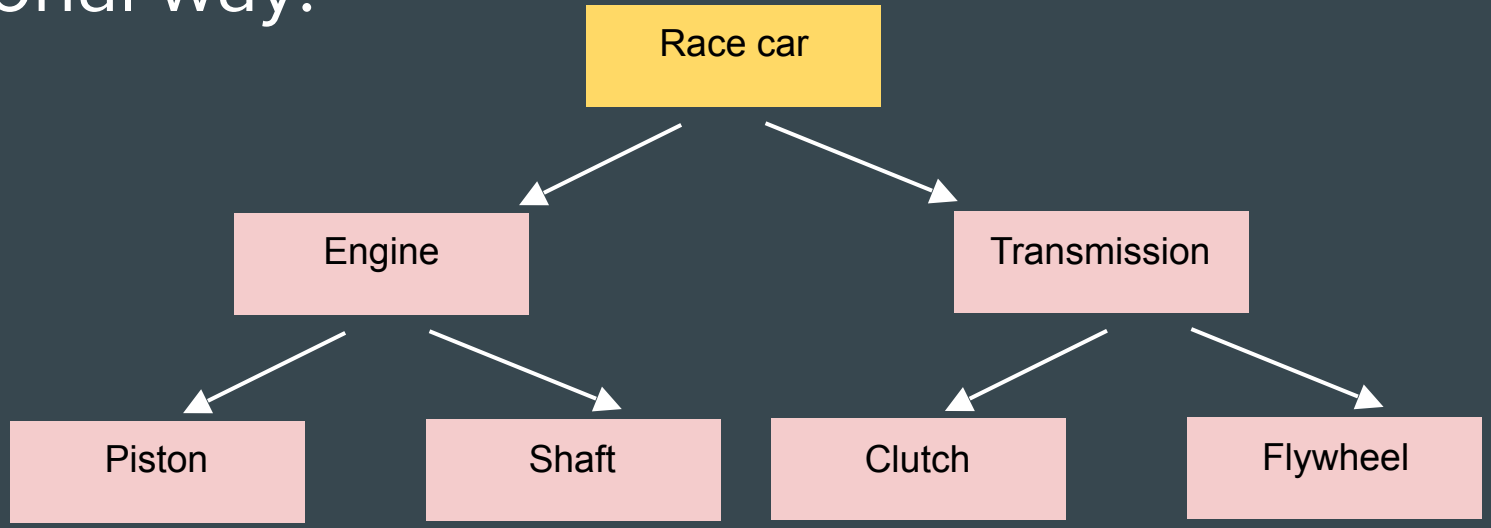
YES

And this?

```
class ViewController: UIViewController {

    // MARK: - Private outlets
    fileprivate lazy var _tableView: UITableView = self._makeTableView()

    // MARK: - Public properties
    let viewModel = ViewModel()

    // MARK: - View lifecycle
    override func viewDidLoad() {
        super.viewDidLoad()

        title = "Events"
        view.backgroundColor = .white

        viewModel.getEvents { [weak self] _ in
            self?._tableView.reloadData()
        }
    }
}
```

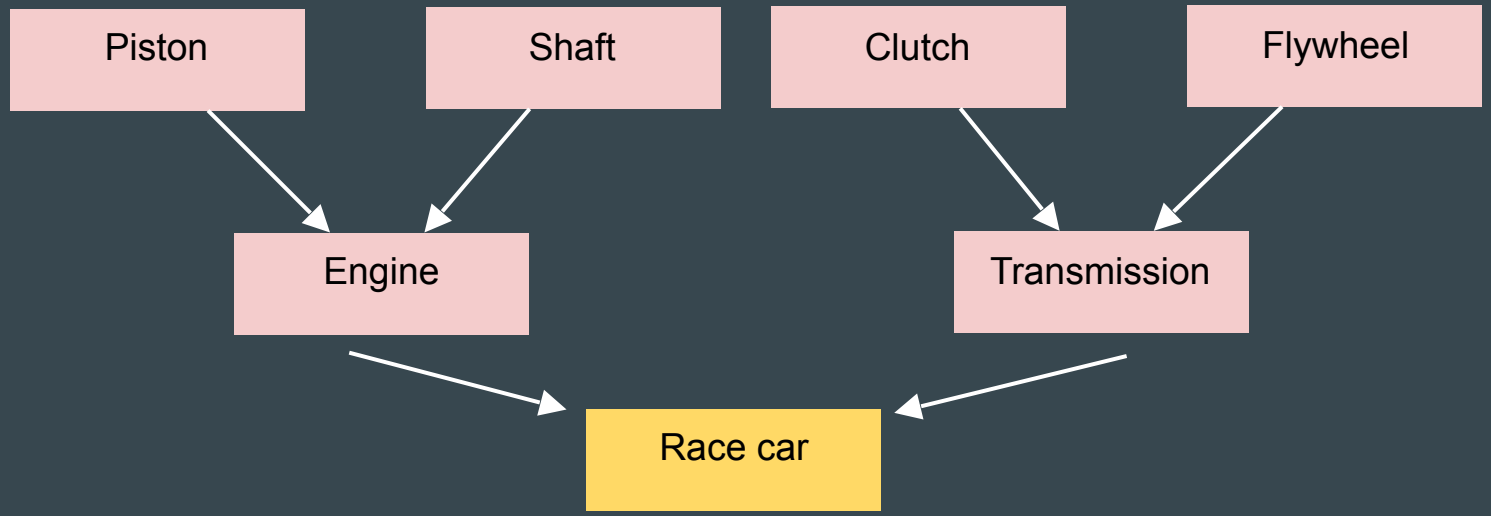
NO

Traditional way:




```
init() {
    super.init()
    ...
    ...
    ...

    self.engine = Engine()
    self.transmission = Transmission()


}
```

Dependency injection way:




```
init(engine: Engine, transmission: Transmission) {
    super.init()
        ...
        ...
        ...

    self.engine = engine
    self.transmission = Transmission()

}
```

Why use dependency Injection**?**

3 C’s

- Clarity
- Customizability
- Concerns

Clarity

Callers need to know only about one thing, i.e. initializer

init() is clean and not dumped with constructors

No unexpected behavior

Example: if a new person joins the team, he need not go throught the viewcontroller to find the dependent objects in it. He just needs to look at init and inject.

Customizability

Have control over which implementation to use. For example: I can plug in Engine with FastEngine without worrying much

Also, highly useful in testing. X1 in debug mode and X2 in release mode.

Concerns

Race car is just a race car. Need not worry about the configuration of other parts.

Specific purposes lying in each component

Allows composing instead of subclassing

Constructor injection vs Setter injection

Constructor injection : use when values dont change

Setter injection : use when required. When you want to set the dependencies via properties

How to setup and use dependency injection?

Now, I will demonstrate how to integrate the **Typhoon framework** to achieve DI.

Concept of Assemblies

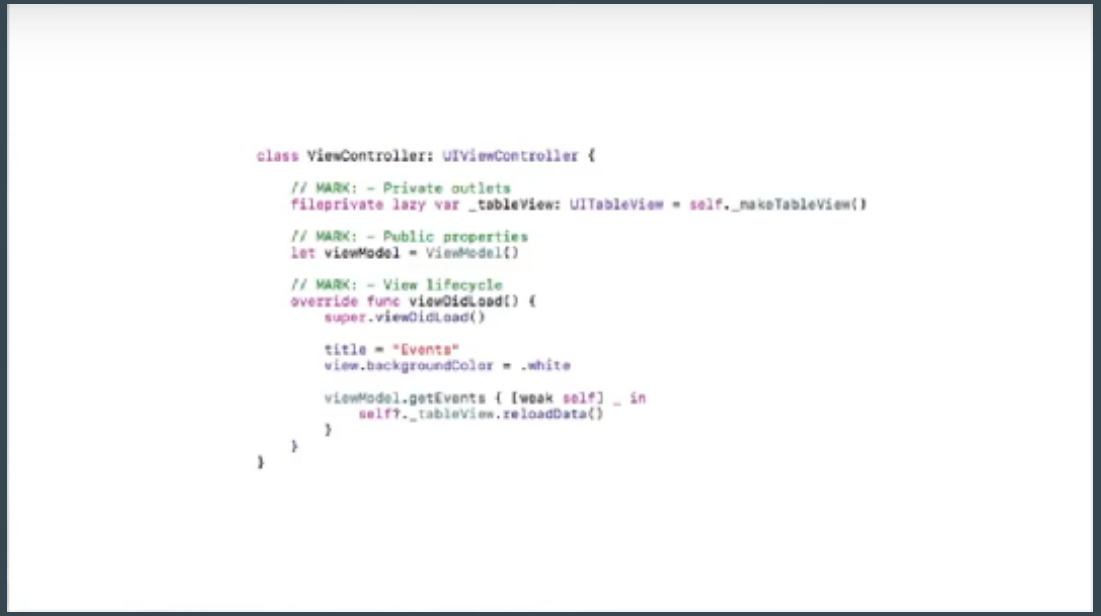
Coming back... Now... you must be able to answer why the code snippet below doesnt confirm to DI...

```
APICommunicator.default.request("GET", url, parameters: nil, headers: nil, completion: { _ in })
```

Why does this confirm to DI?

```
func set(_ event: Event) {
    _nameLabel.text = event.name
    _descriptionLabel.text = event.description

    _dueLabel.isHidden = event.date > Date()
}
```

And finally, why doesnt this confirm to DI?



```
class ViewController: UIViewController {

    // MARK: - Private outlets
    fileprivate lazy var _tableView: UITableView = self._makeTableView()

    // MARK: - Public properties
    let viewModel = ViewModel()

    // MARK: - View lifecycle
    override func viewDidLoad() {
        super.viewDidLoad()

        title = "Events"
        view.backgroundColor = .white

        viewModel.getEvents { [weak self] _ in
            self?._tableView.reloadData()
        }
    }
}
```