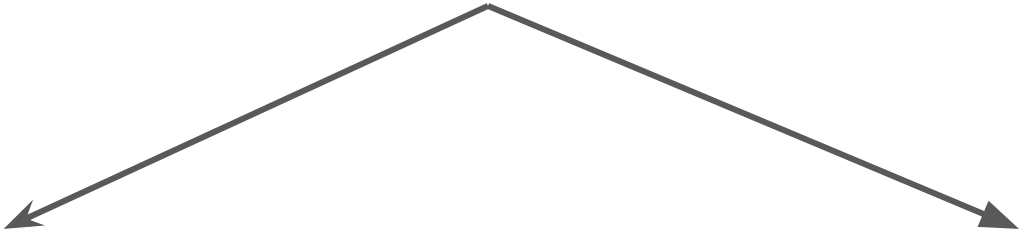
Dependency Injection (DI)

with Dry::Container

Oleksandr Polieno

Ruby Tuesday #39

Build software that is easy to change!



Reduce mental load

Reduce fear of change

SOLID DI TDD DDD XP

Single Responsibility (**S**OLID)

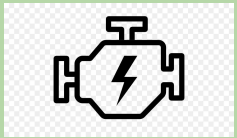
Create pressure in the fuel system

Store fuel

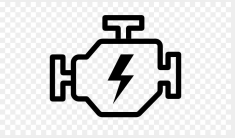
Interface Segregation (SOLID)

So you are an engine? There is a fuel pump for you!

Dependency Inversion (SOLI**D**)

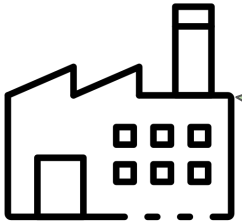
I need an 

I need a Toyota 2JZGTE!

I am an 

Inversion of Control (IoC)

I'll take Toyota 2JZGTE engine



Here is Toyota 2JZGTE engine

Inheritance vs Composition

class BaseCar

🚫

class BaseCarWithEngine < BaseCar

class Engine

✅

class Car

Coupling

```ruby
class Engine
  def accelerate
  end

  def start
  end

  def stop
  end

  def rpm
  end
end

class FuelPump
end
```

Single Responsibility

```ruby
class Engine2JZGTE < Engine
  def initialize
    @fuel_pump = FuelPump.new
  end

  def start
    super
  end
end

class Car
  def initialize
    @engine = Engine2JZGTE.new
  end
end

car = Car.new
```

Interface Segregation

Decouple

```ruby
class Engine2JZGTE < Engine
  def initialize(fuel_pump: FuelPump)
    @fuel_pump = fuel_pump
  end

  def start
    super
  end
end

class Car
  def initialize(engine: Engine)
    @engine = engine
  end
end

fuel_pump = FuelPump.new
engine = Engine2JZGTE.new(fuel_pump: fuel_pump)
car = Car.new(engine: engine)
```

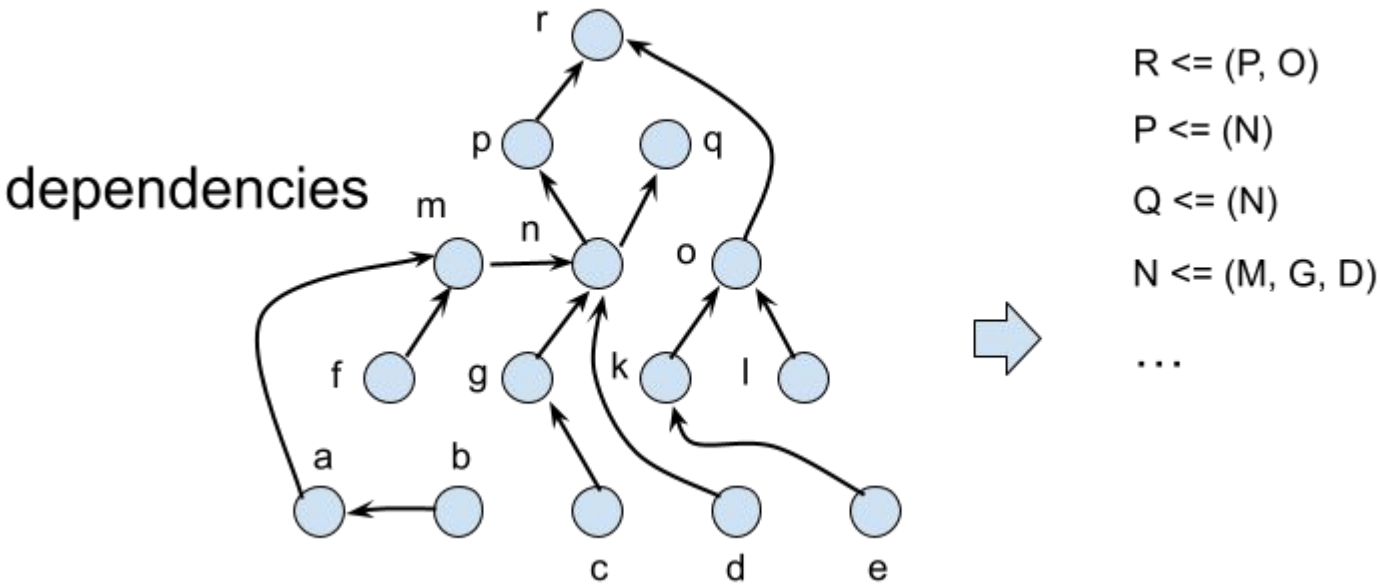
Dependency Inversion

Inversion of Control

Benefits:

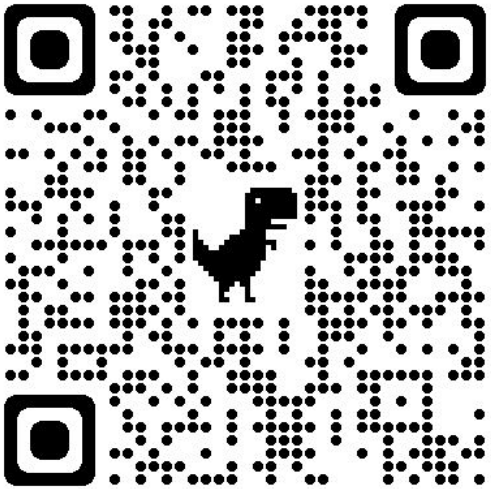
- Can clearly see all the dependencies
- Easier to change (each object and the entire app)
- Easies to test (can pass a fake object of mock as dependency, not need to patch)

Dependency Locator (Container)

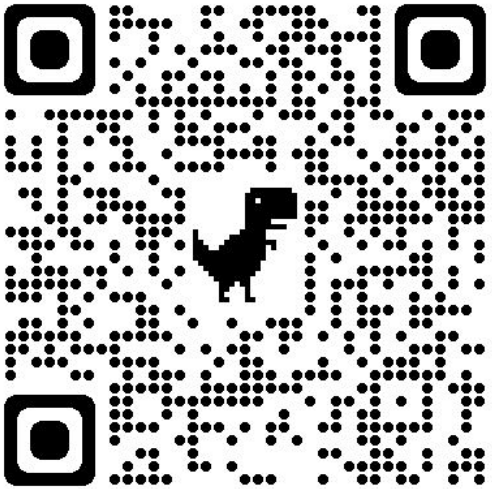


TODO List App example

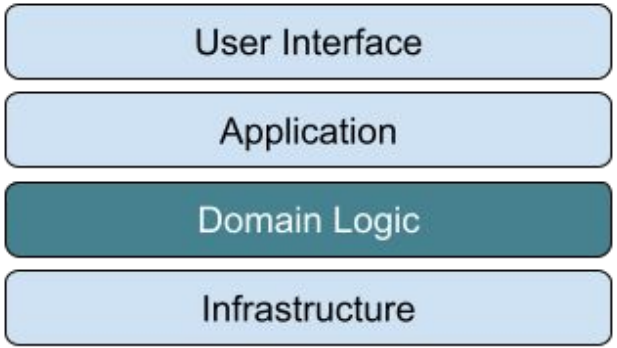
The App on GitHub



Dry::Container



Using Container inside a framework



Using Container with Django example

Providers