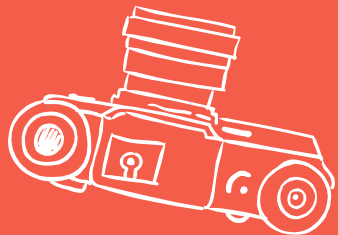


DESIGN PATTERNS

BY VLAD GREGURCO



 **Pentalog**
Software Factory



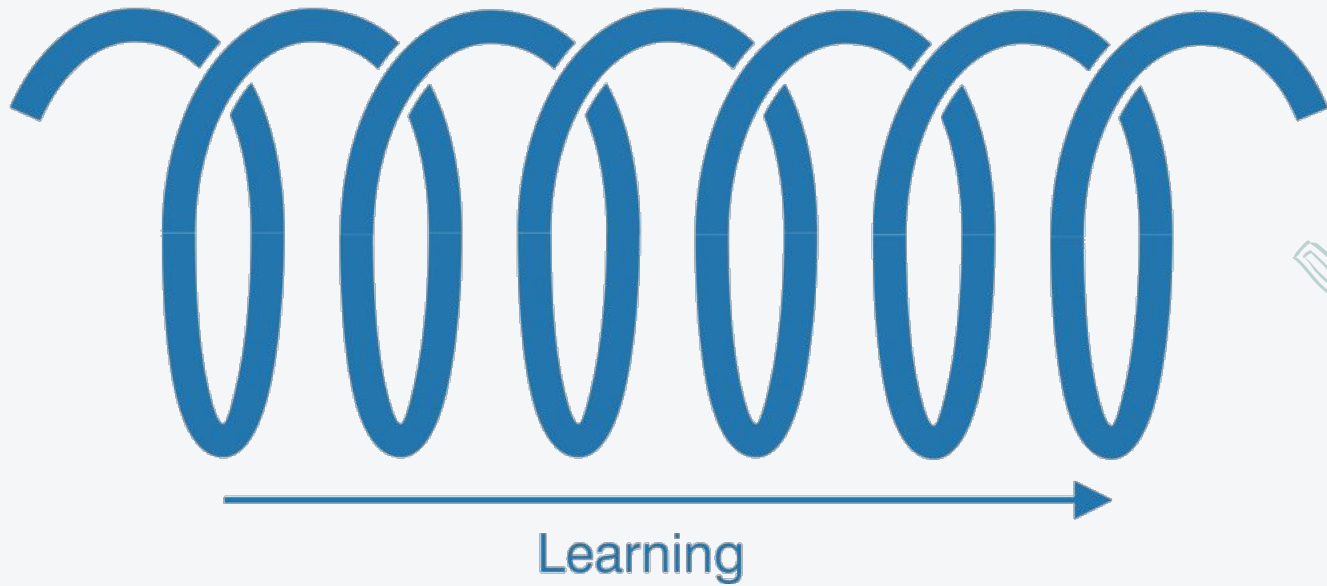


In software design, a
design pattern is an
abstract generic solution
To solve a particular
common problem.

BENEFITS

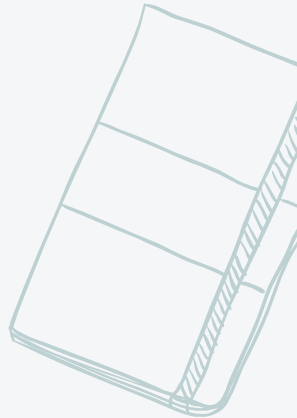


TRIED AND TESTED SOLUTIONS





DOCUMENTED



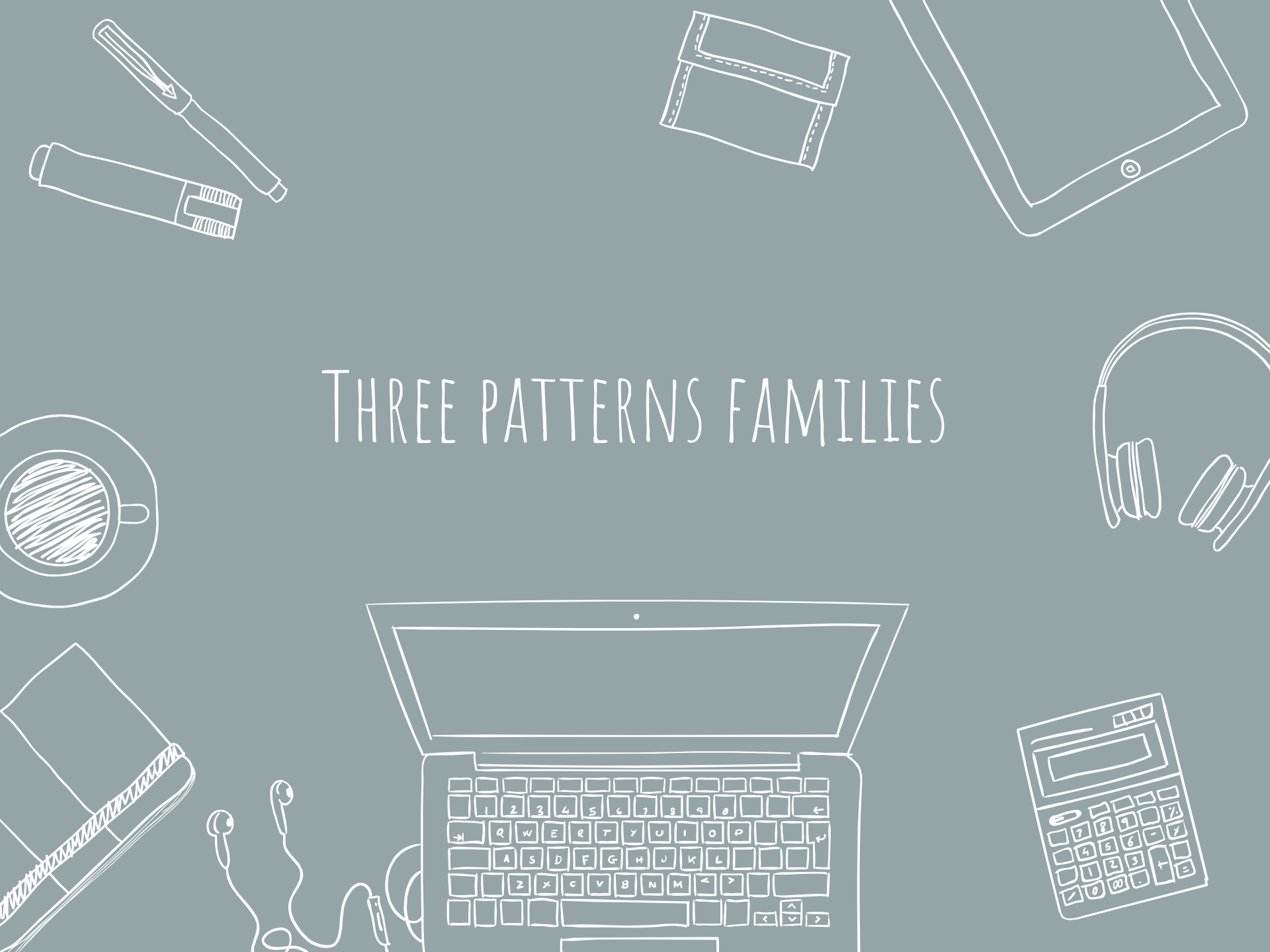
FLEXIBLE



DISCUSSION



THREE PATTERNS FAMILIES





CREATIONAL PATTERNS

- Abstract Factory
- Builder
- Factory Method
- Lazy Initialization
- Prototype
- Singleton



STRUCTURAL PATTERNS

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy






BEHAVIORAL PATTERNS

- Chain of Responsibility
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

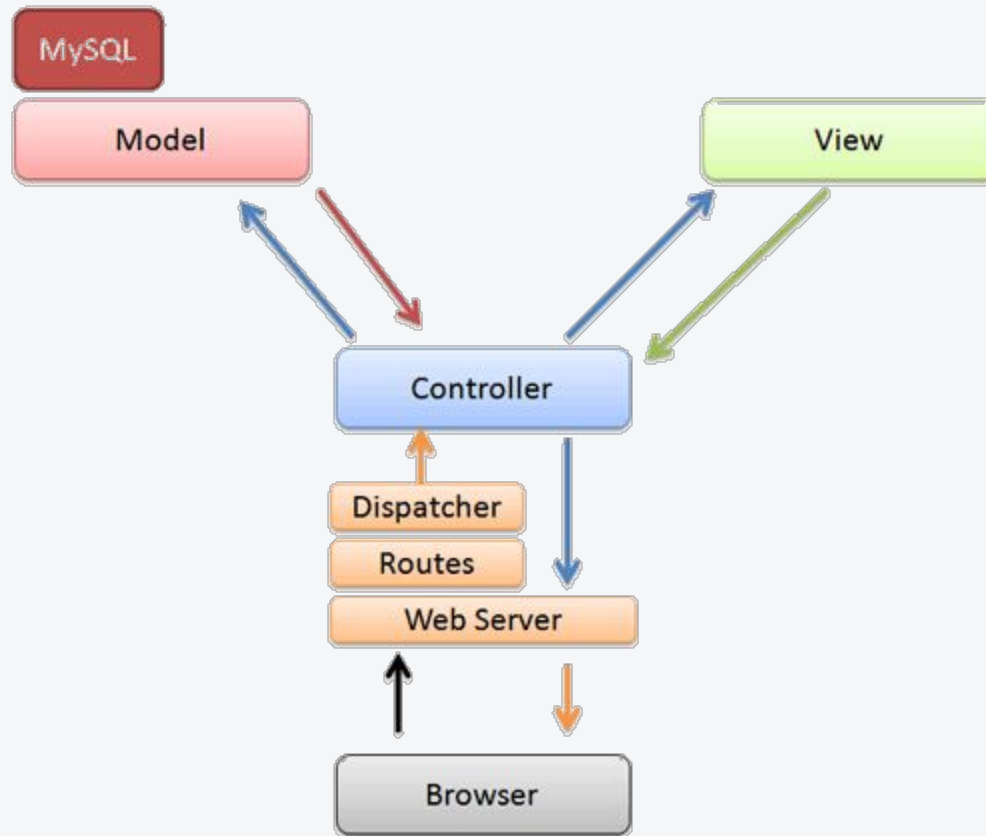


MVC

*Divide an application into three
interconnected parts*



MVC





ITERATOR

*The iterator pattern
allows to traverse a
container and access
its elements.*

ITERATOR

```
interface Iterator
{
    public function current();
    public function key();
    public function next();
    public function rewind();
    public function valid();
}
```

ITERATOR

C BookListIterator	
f	bookList
f	currentBook
m	__construct(bookList)
m	current()
m	next()
m	key()
m	valid()
m	rewind()

C Book	
f	author
f	title
m	__construct(title, author)
m	getAuthor()
m	getTitle()
m	getAuthorAndTitle()

C BookListReverseIterator	
f	bookList
f	currentBook
m	__construct(bookList)
m	next()
m	valid()

C BookList	
f	books
m	getBook(bookNumberToGet)
m	addBook(book)
m	removeBook(bookToRemove)
m	count()

Powered by yFiles.



DEPENDENCY INJECTION

To implement a loosely coupled architecture in order to get better testable, maintainable and extendable code.

DEPENDENCY INJECTION

DatabaseConfiguration

f 🔒 password

f 🔒 port

f 🔒 host

f 🔒 username

m 🔓 getHost()

m 🔓 getPort()

m 🔓 getUsername()

m 🔓 getPassword()

DatabaseConnection

f 🔒 configuration

m 🔓 getDsn()

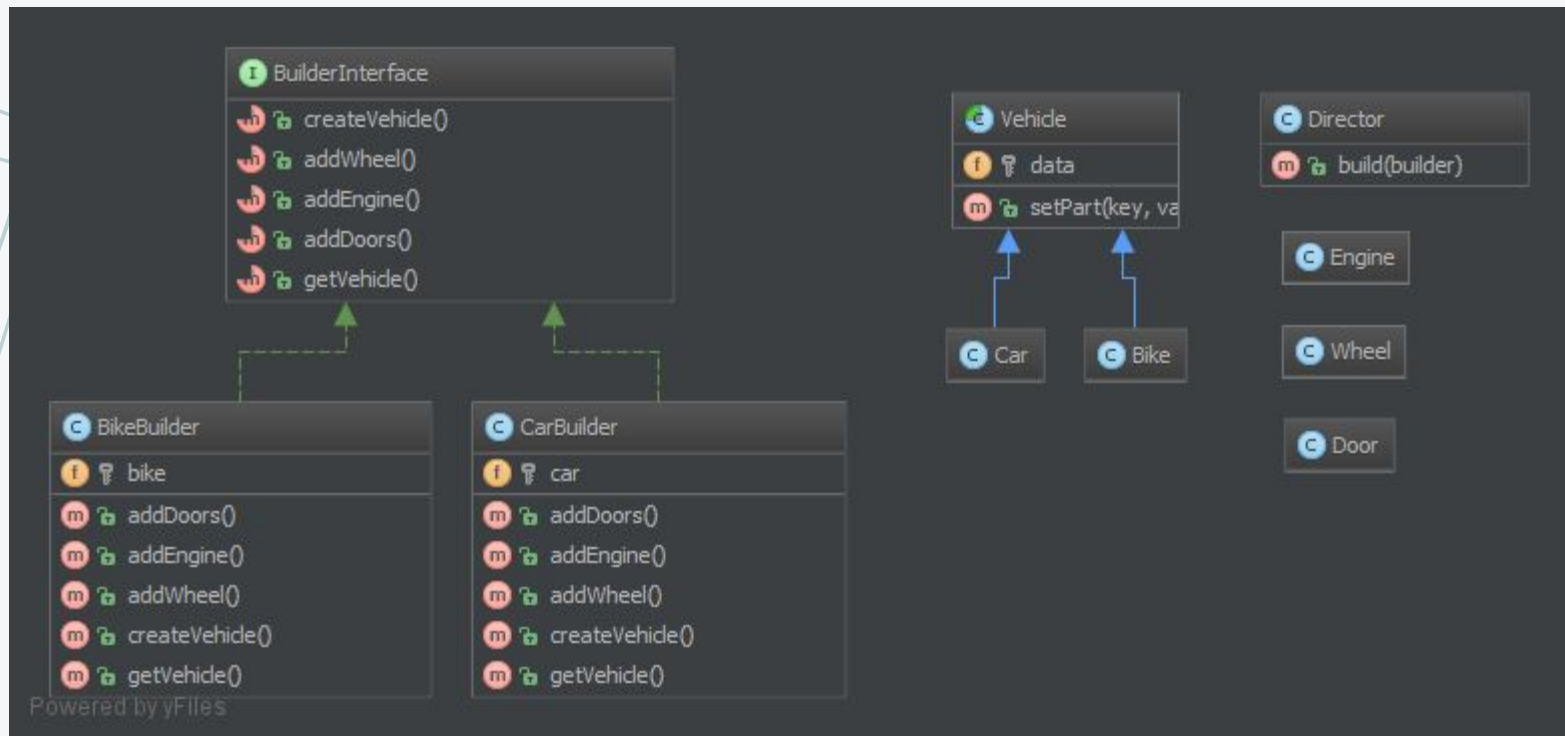
Powered by yFiles



BUILDER

Builder is an interface that build parts of a complex object.

BUILDER








SIMPLE FACTORY


Simple Factory is a simple factory pattern.



SIMPLE FACTORY

 Bicycle

  driveTo(destination)

 SimpleFactory

  createBicycle()

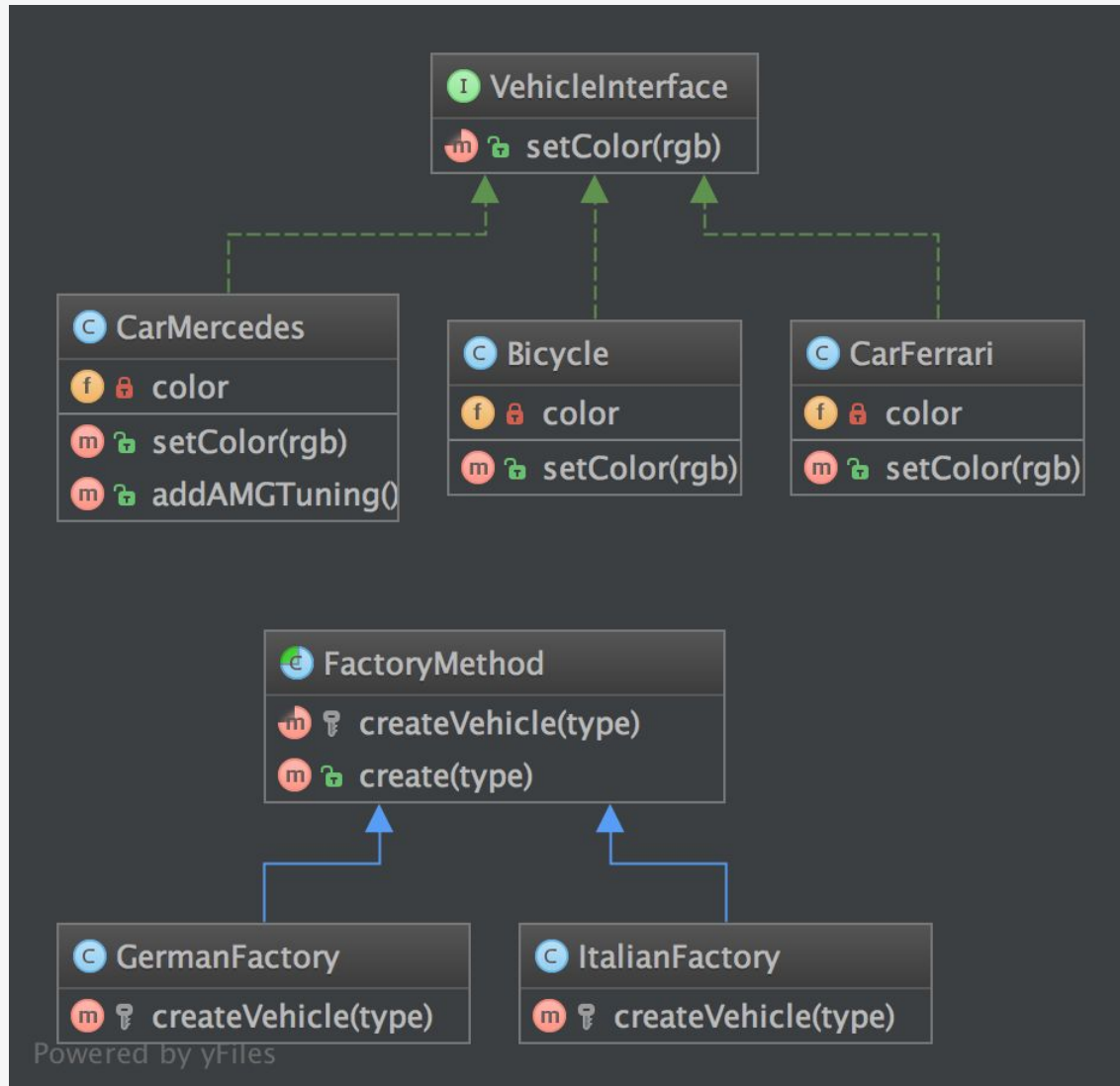
Powered by yFiles



FACTORY METHOD

Define an interface for creating an object, but let subclasses decide which class to instantiate

FACTORY METHOD

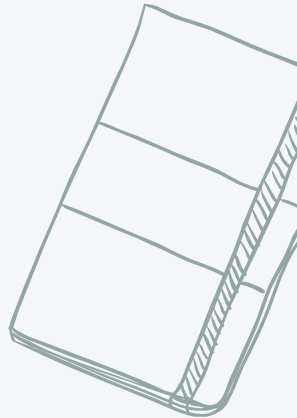




LAZY INITIALIZATION


The lazy initialization pattern is the tactic of delaying the creation of an object, the calculation of a value, or some other expensive process until the first time it is really needed.

LAZY INITIALIZATION
+ FACTORY METHOD
= SERVICE CONTAINER








THE SERVICE CONTAINER



The service container allows you to standardize and centralize the way objects are constructed in your application.



SERVICE CONTAINER

```
class Container implements ResettableContainerInterface
{
    protected $services = [];
    protected $methodMap = [];

    public function get($id)
    {
        if (isset($this->services[$id])) {
            return $this->services[$id];
        }



        $method = $this->methodMap[$id];

        return call_user_func([$this, $method]);
    }
}
```



MEDIATOR

The mediator pattern defines an object that encapsulates how a set of objects interact.







MEDIATOR





FACADE

The primary goal of a Facade Pattern is not to avoid you having to read the manual of a complex API. It's only a side-effect.





FACADE

C Facade


f  os

f  bios


m  turnOn()


m  turnOff()

I BiosInterface


m  execute()


m  waitForKeyPress()

m  launch(os)

m  powerDown()

I OsInterface

m  halt()

m  getName()

Powered by yFiles



QUESTIONS?





thank you!

