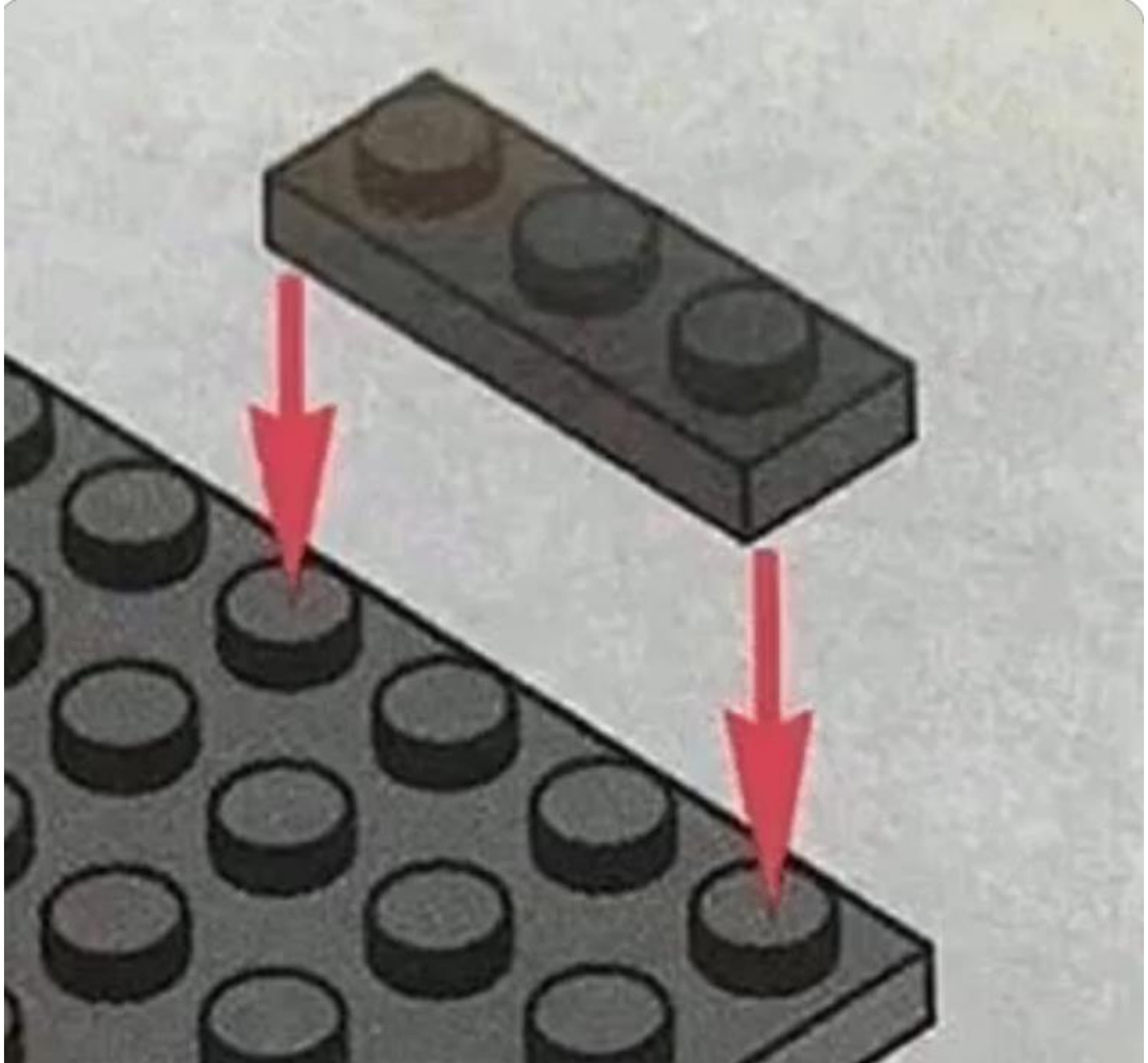
_GRASP

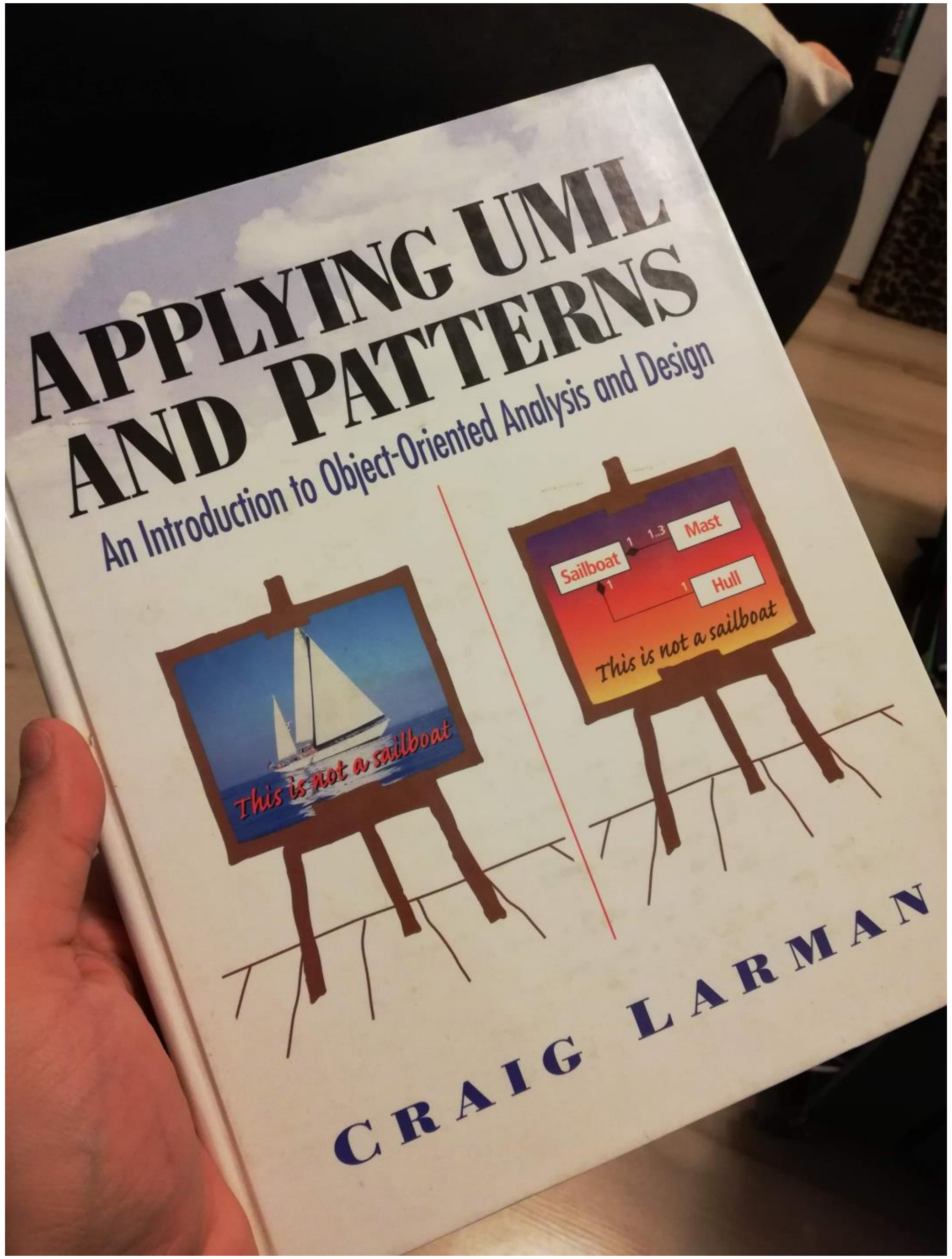
Karol Kreft

PHPers Summit 2019

06.09.2019, Poznań

Never settle
intive.com




APPLYING UML AND PATTERNS
An Introduction to Object-Oriented Analysis and Design
This is not a sailboat
Sailboat
Mast
Hull
1
1..3
1
1
This is not a sailboat
CRAIG LARMAN

Craig Larman

- www.craiglarman.com
- UML, OOP, Analysis & Design
- **GRASP**
- Agile and LeSS

The **GRASP** name was chosen to suggest the importance of **grasping** these principles to successfully design object-oriented software.

intive

General
Responsibility
Assignment
Software
Patterns

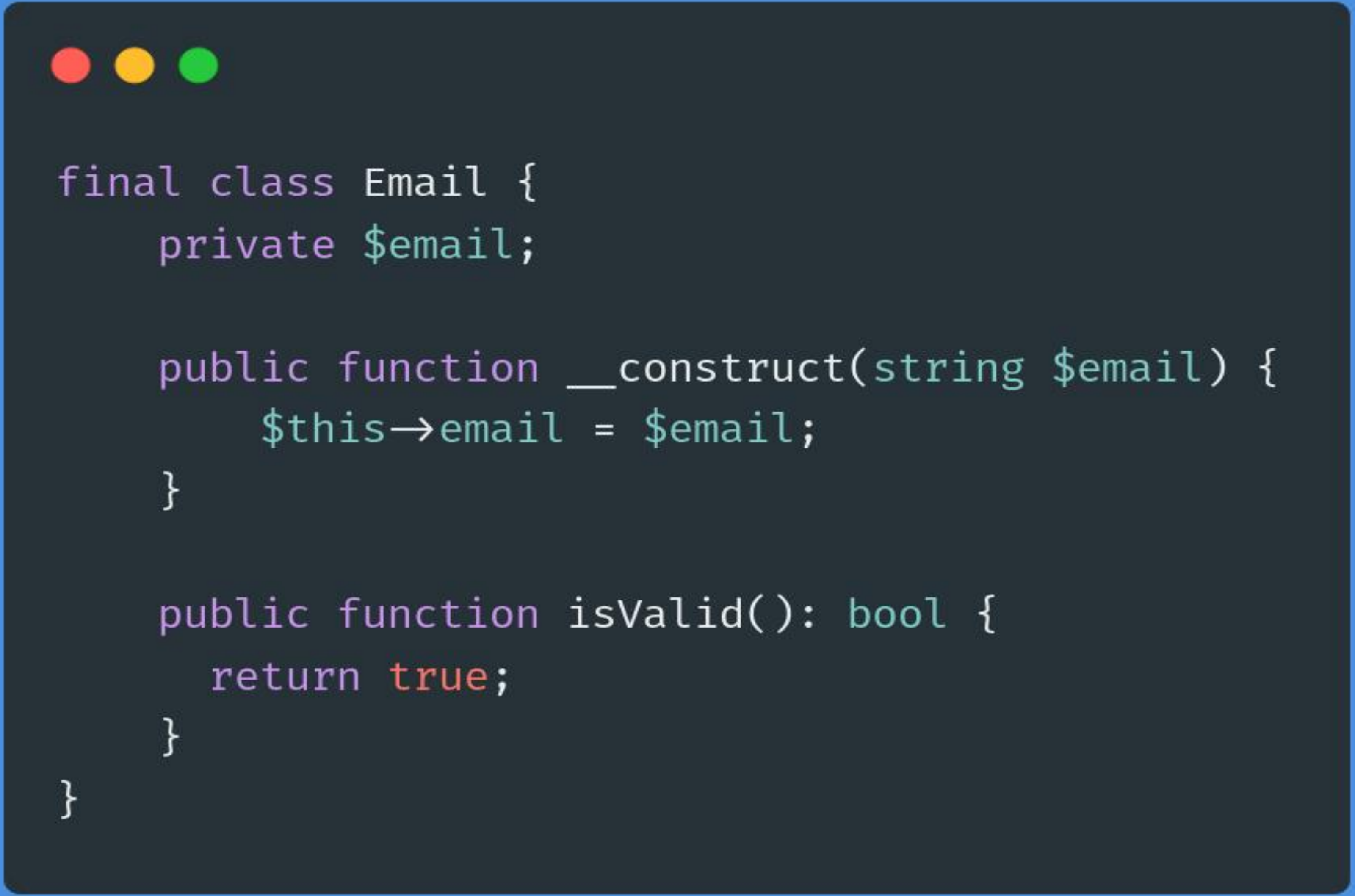
Responsibilities

intive

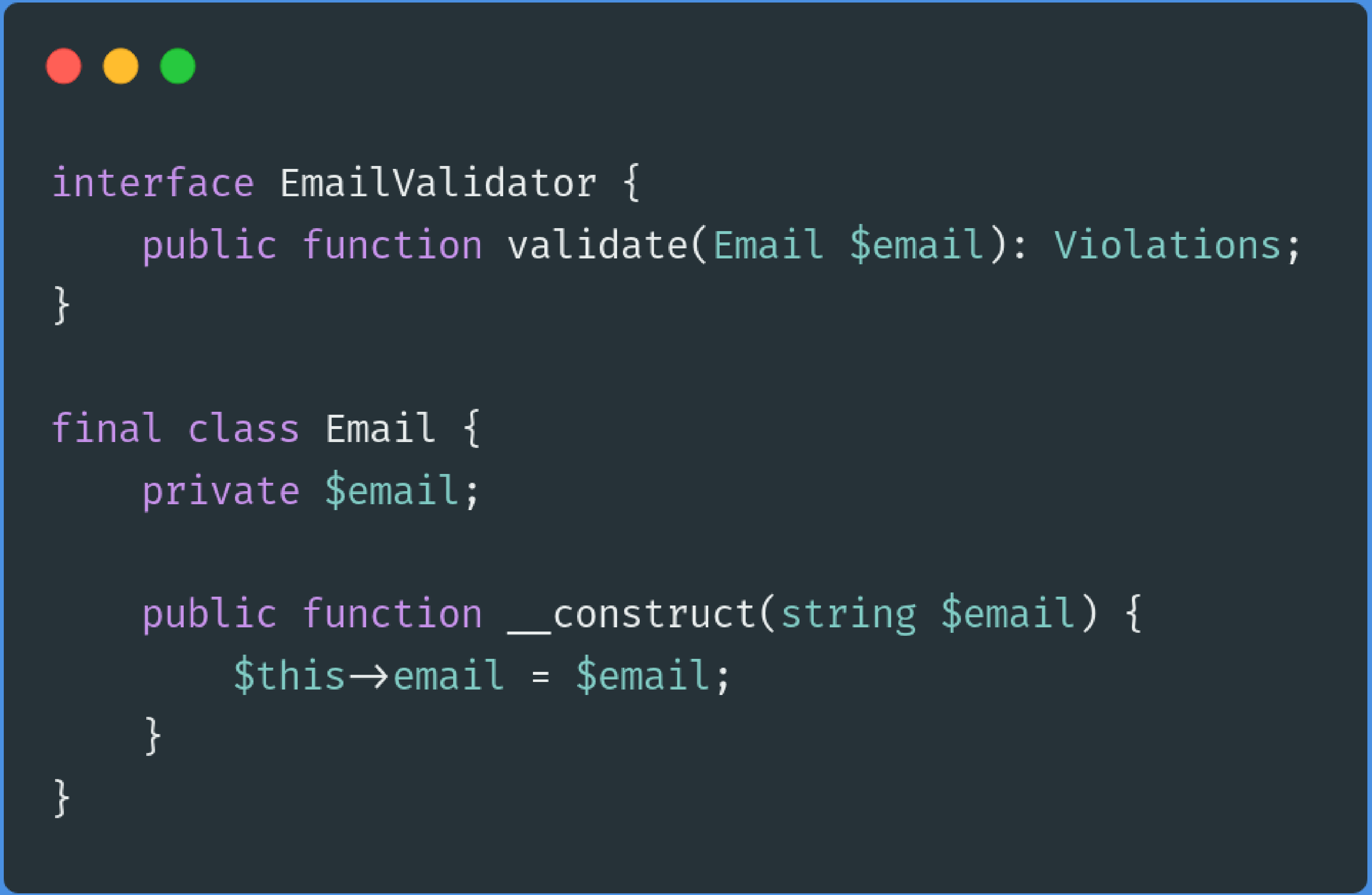
Responsibilities

{

- do something
- know something



```
final class Email {
    private $email;

    public function __construct(string $email) {
        $this→email = $email;
    }

    public function isValid(): bool {
      return true;
    }
}
```



```php
interface EmailValidator {
    public function validate(Email $email): Violations;
}

final class Email {
    private $email;

    public function __construct(string $email) {
        $this->email = $email;
    }
}
```

GRASP

1. **Information Expert**
2. **Creator**
3. **Controller**
4. **Low coupling**
5. **High cohesion**
6. **Indirection**
7. Polymorphism
8. Pure Fabrication
9. Protected Variations

Information Expert

_Assign a responsibility to the information expert – the class that has the information necessary to fulfill the responsibility.

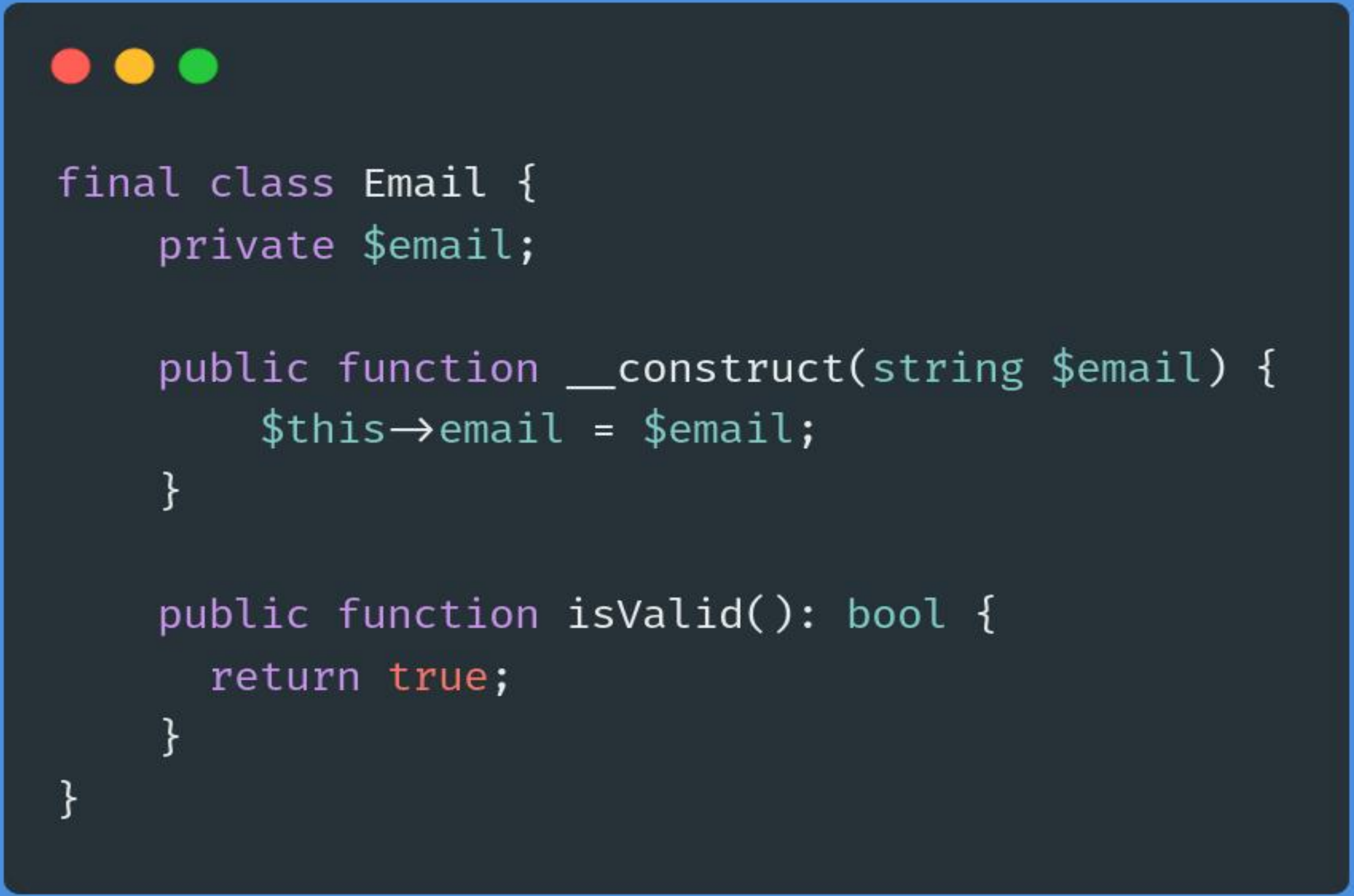
Information Expert

_Place responsibilities with data

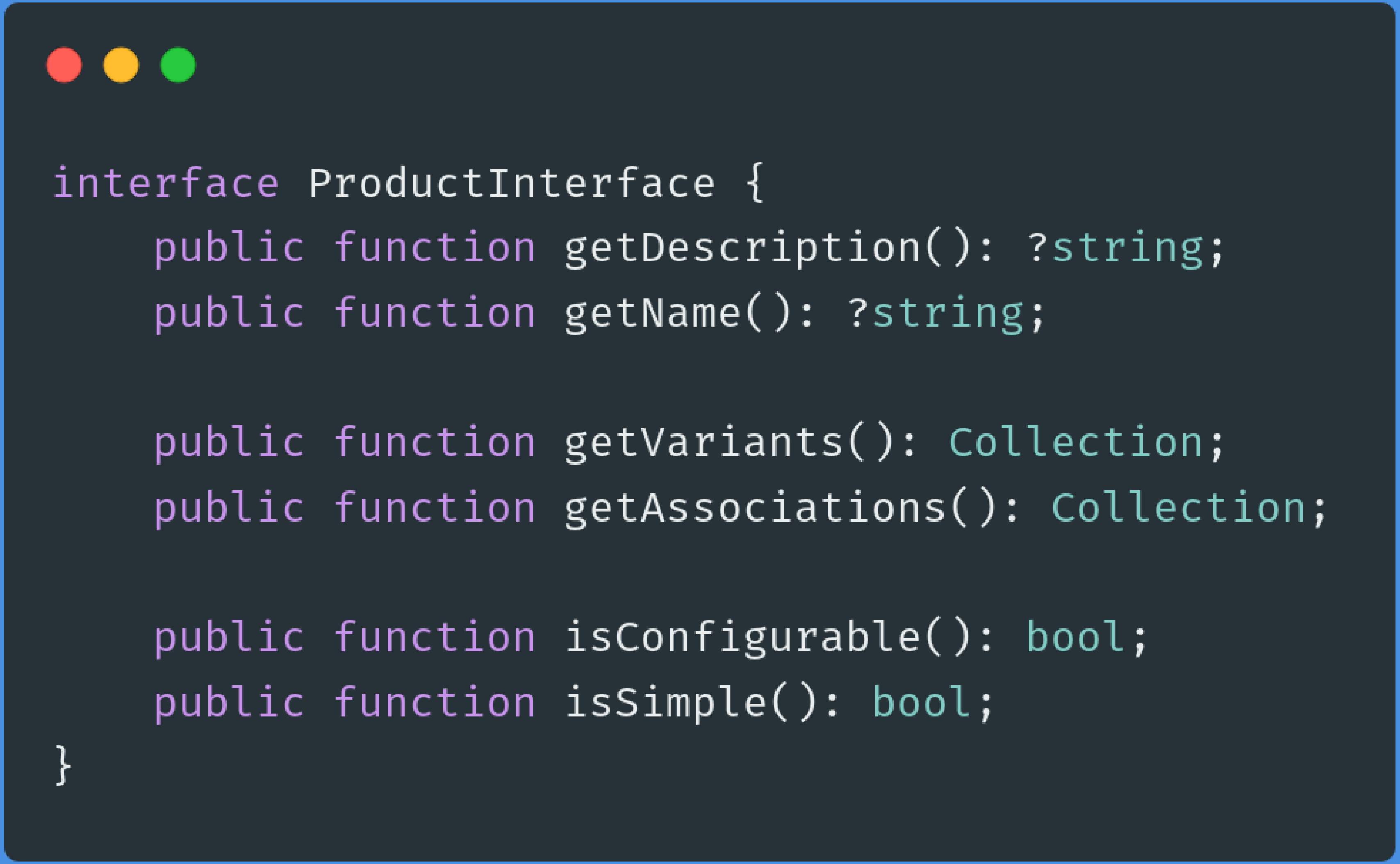
_That which knows, does

_Do it Myself

_Put Services with the Attributes They Work On



```
final class Email {
    private $email;

    public function __construct(string $email) {
        $this→email = $email;
    }

    public function isValid(): bool {
      return true;
    }
}
```



```
interface ProductInterface {
    public function getDescription(): ?string;
    public function getName(): ?string;

    public function getVariants(): Collection;
    public function getAssociations(): Collection;

    public function isConfigurable(): bool;
    public function isSimple(): bool;
}
```

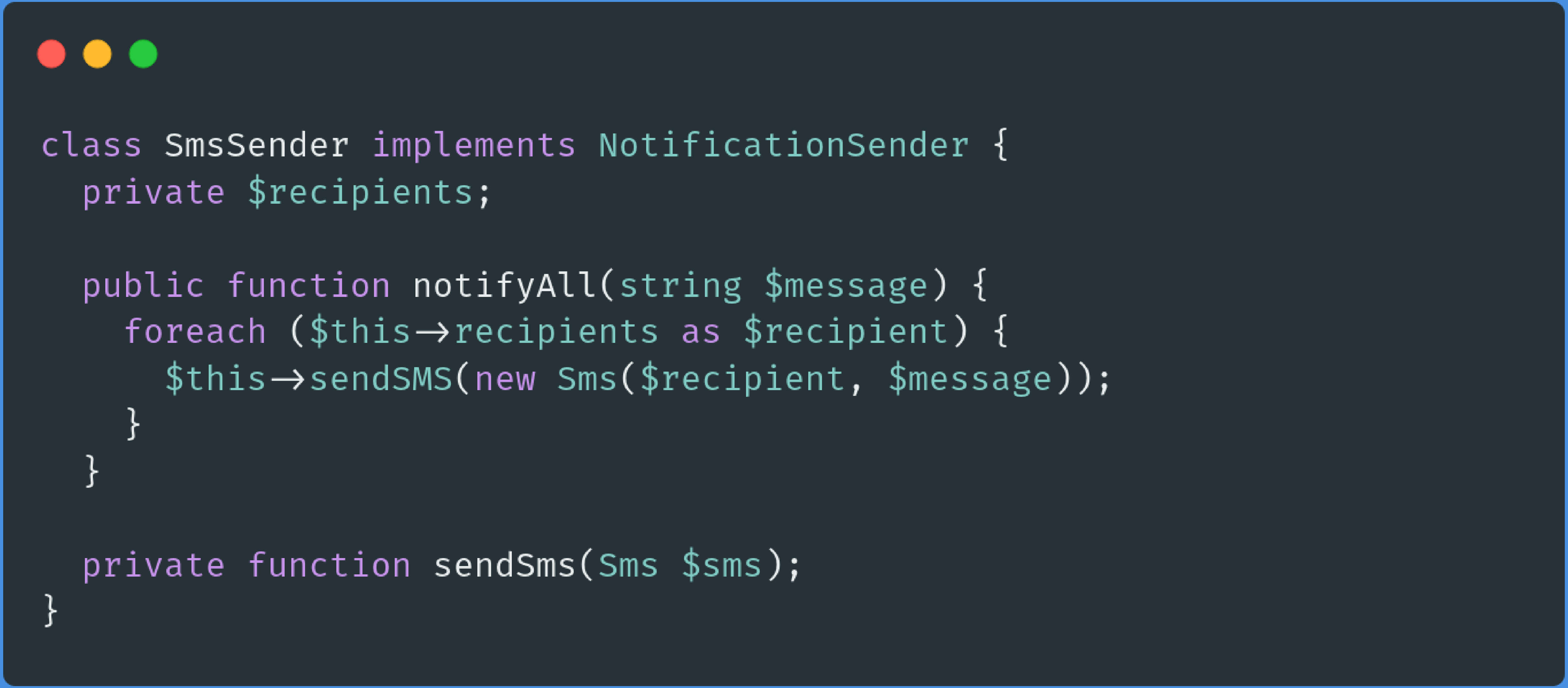
Never settle
intive.com

Creator

_Who should be responsible for creating a new instance of some class?

_Choose class B when

- _B aggregates A objects
- _B contains A objects
- _B closely uses A objects
- _B as the initializing data that are required to creating A



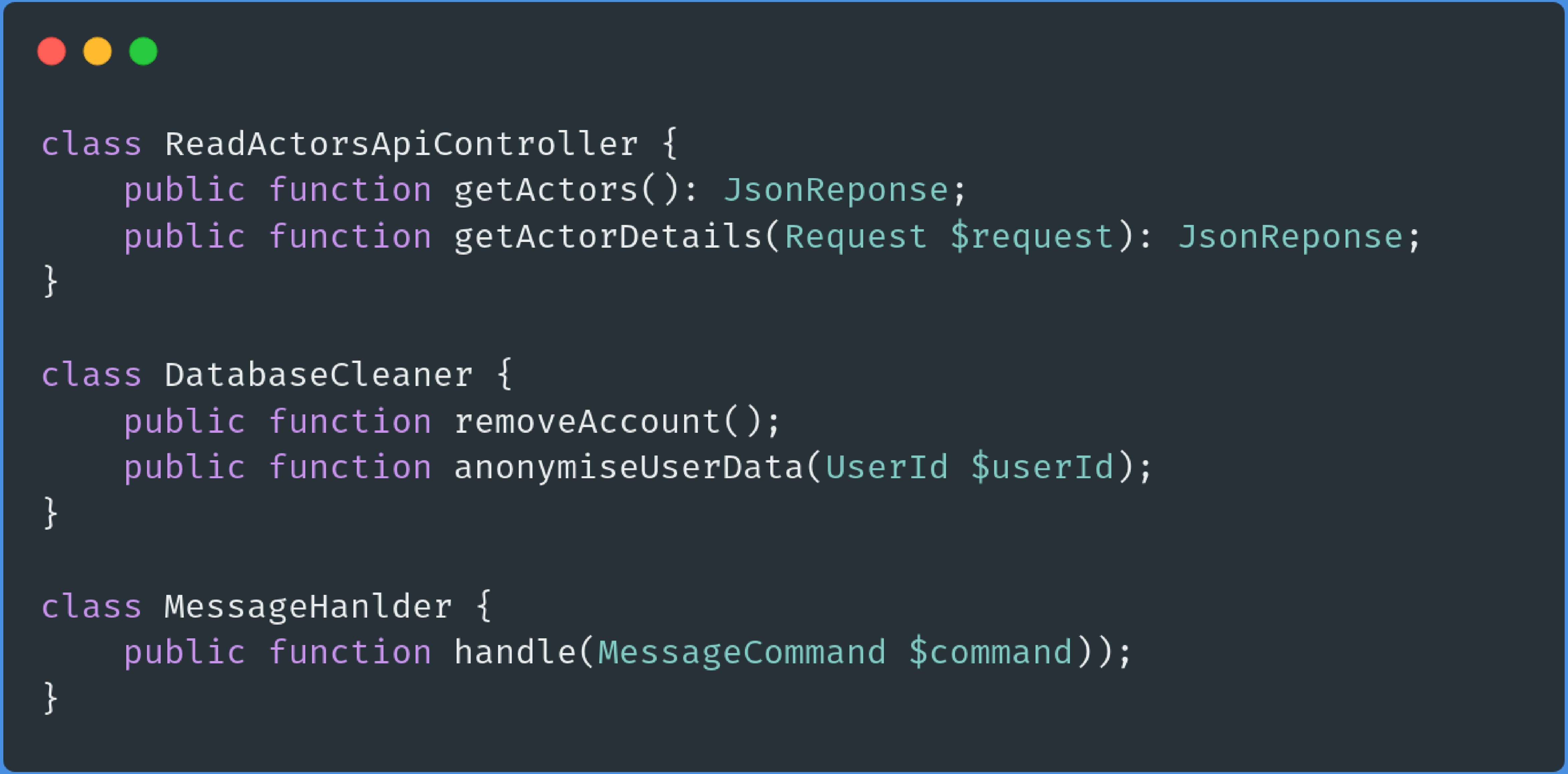
```php
class SmsSender implements NotificationSender {
  private $recipients;

  public function notifyAll(string $message) {
    foreach ($this→recipients as $recipient) {
      $this→sendSMS(new Sms($recipient, $message));
    }
  }

  private function sendSms(Sms $sms);
}
```

intive

Controller

_Who should be responsible for handling system event?

Never settle
intive.com



```
class ReadActorsApiController {
    public function getActors(): JsonReponse;
    public function getActorDetails(Request $request): JsonReponse;
}

class DatabaseCleaner {
    public function removeAccount();
    public function anonymiseUserData(UserId $userId);
}

class MessageHanlder {
    public function handle(MessageCommand $command));
}
```

Bloated Controller

_Controllers which handle too many system events leading to **low cohesion**. This can be avoided by addition of a few more controllers.

_Always remember about delegating responsibilities to other objects.

_Use **Command Pattern** in a message-handling systems.

Low coupling

High cohesion

Low Coupling

_How to support low dependency and increased reuse?

High Cohesion

_How to keep complexity manageable?

intive

Low Coupling

_Assign responsibilities so that coupling remains **low**.

High Cohesion

_Assign responsibilities so that cohesion remains **high**.

High Coupling

_Changes in related classes force local changes

_Harder to understand in isolation

_Harder to reuse because its use requires the additional presence of the classes it dependent upon

_Coupling may not be important if reuse is not a goal.

Low Cohesion

_Hard to comprehend

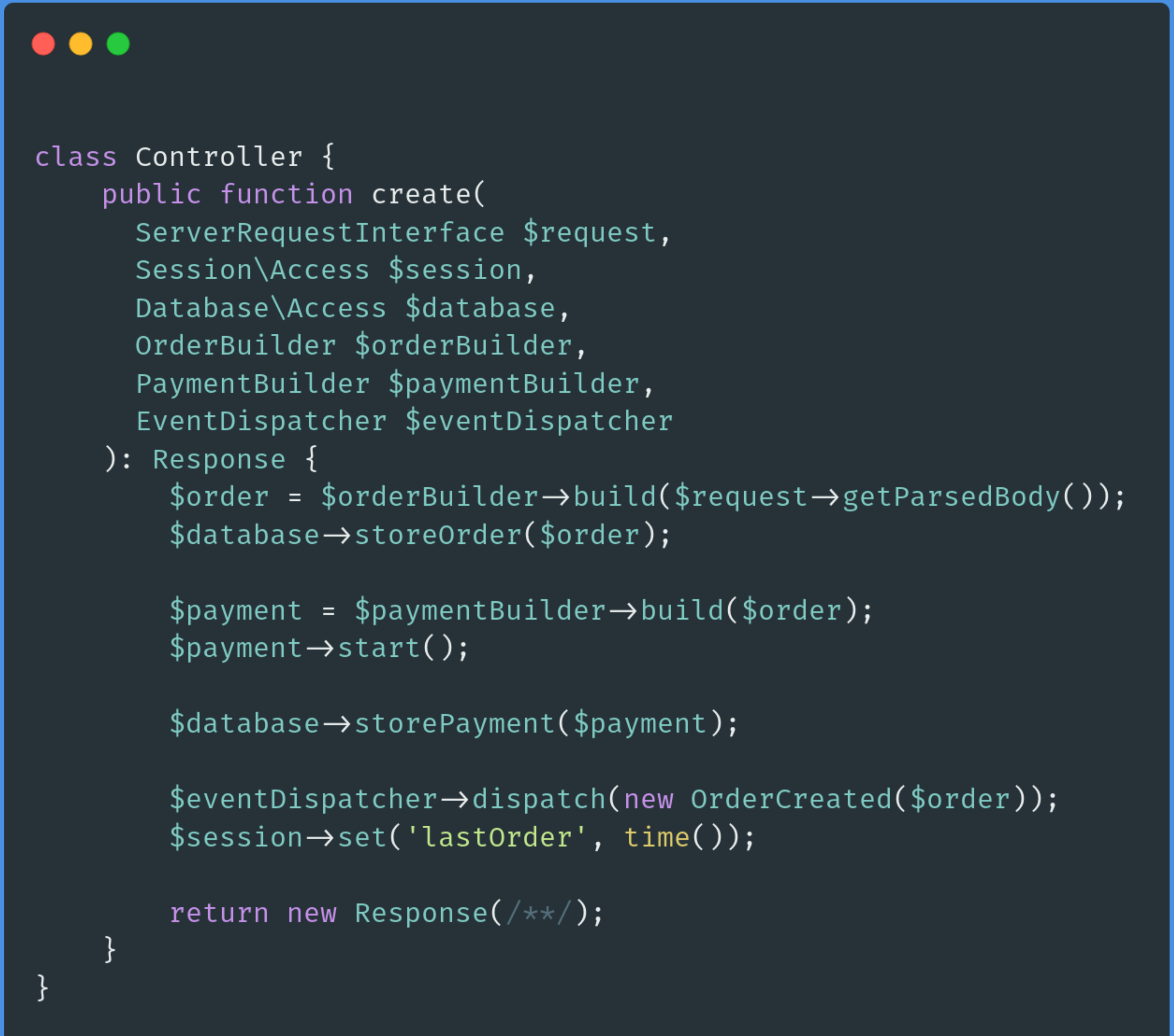
_Hard to reuse

_Hard to maintain

_Delicate; constantly effected by change

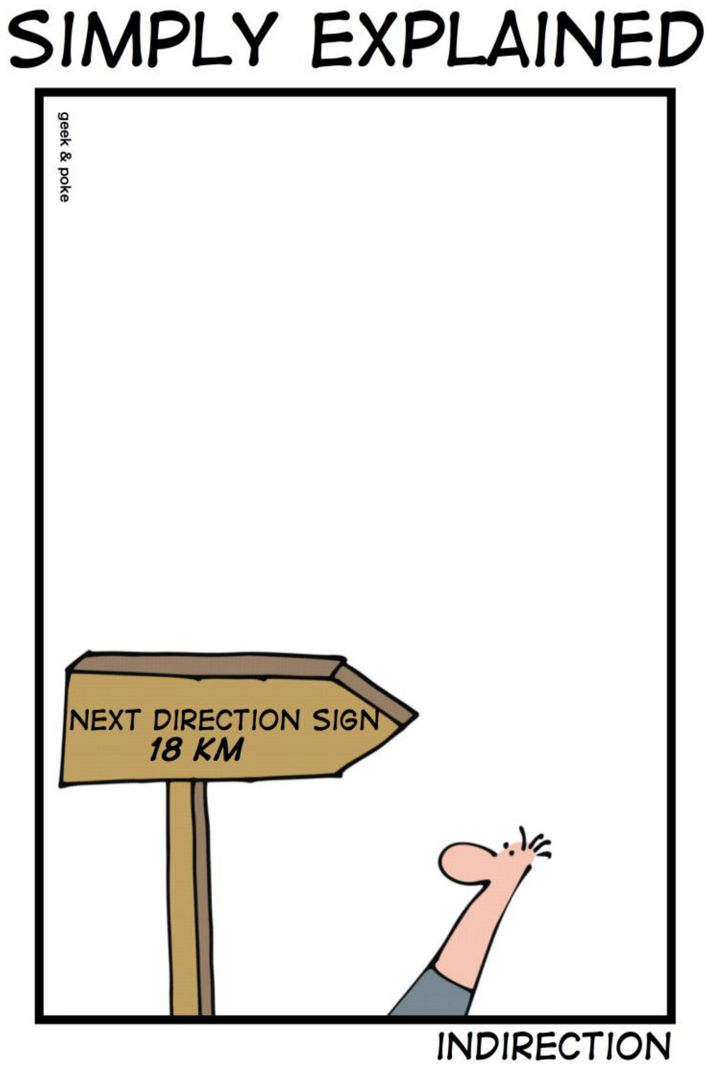
_Low Coupling and High Cohesion are principles to keep in mind during all design decisions.

_They are evaluative patterns which a designer applies while evaluating all design decisions.



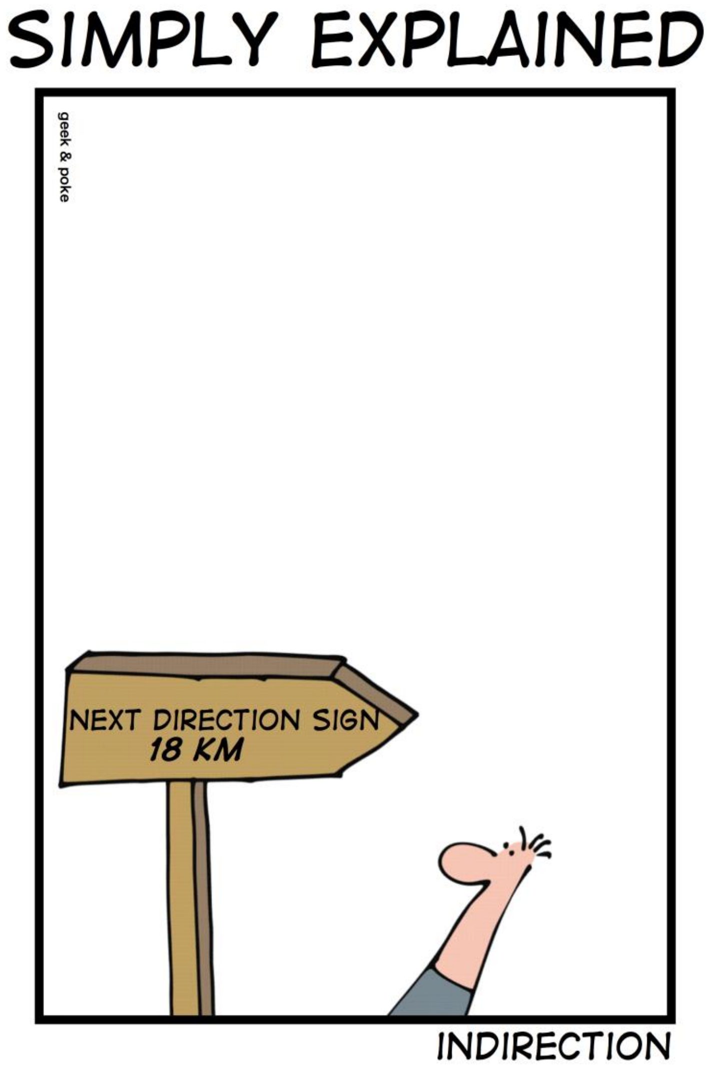
```php
class Controller {
    public function create(
      ServerRequestInterface $request,
      Session\Access $session,
      Database\Access $database,
      OrderBuilder $orderBuilder,
      PaymentBuilder $paymentBuilder,
      EventDispatcher $eventDispatcher
    ): Response {
        $order = $orderBuilder->build($request->getParsedBody());
        $database->storeOrder($order);

        $payment = $paymentBuilder->build($order);
        $payment->start();

        $database->storePayment($payment);

        $eventDispatcher->dispatch(new OrderCreated($order));
        $session->set('lastOrder', time());

        return new Response(/**/);
    }
}
```

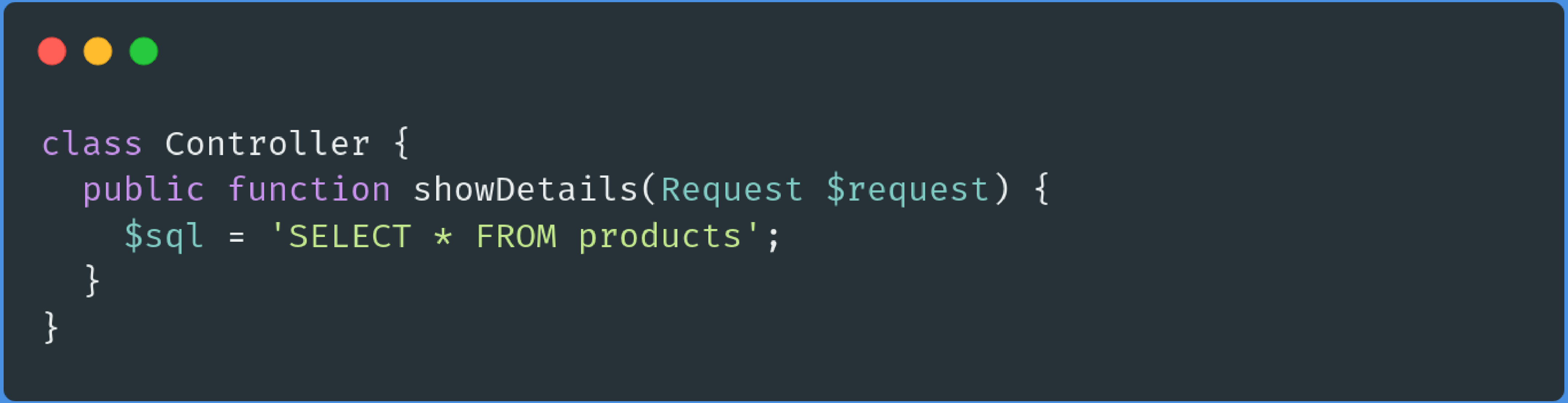
_Where to assign a responsibility to avoid direct coupling between two or more things?



_Where to assign a responsibility to avoid direct coupling between two or more things?

_Assign the responsibility to an intermediate object to mediate between other components or services to avoid direct coupling.





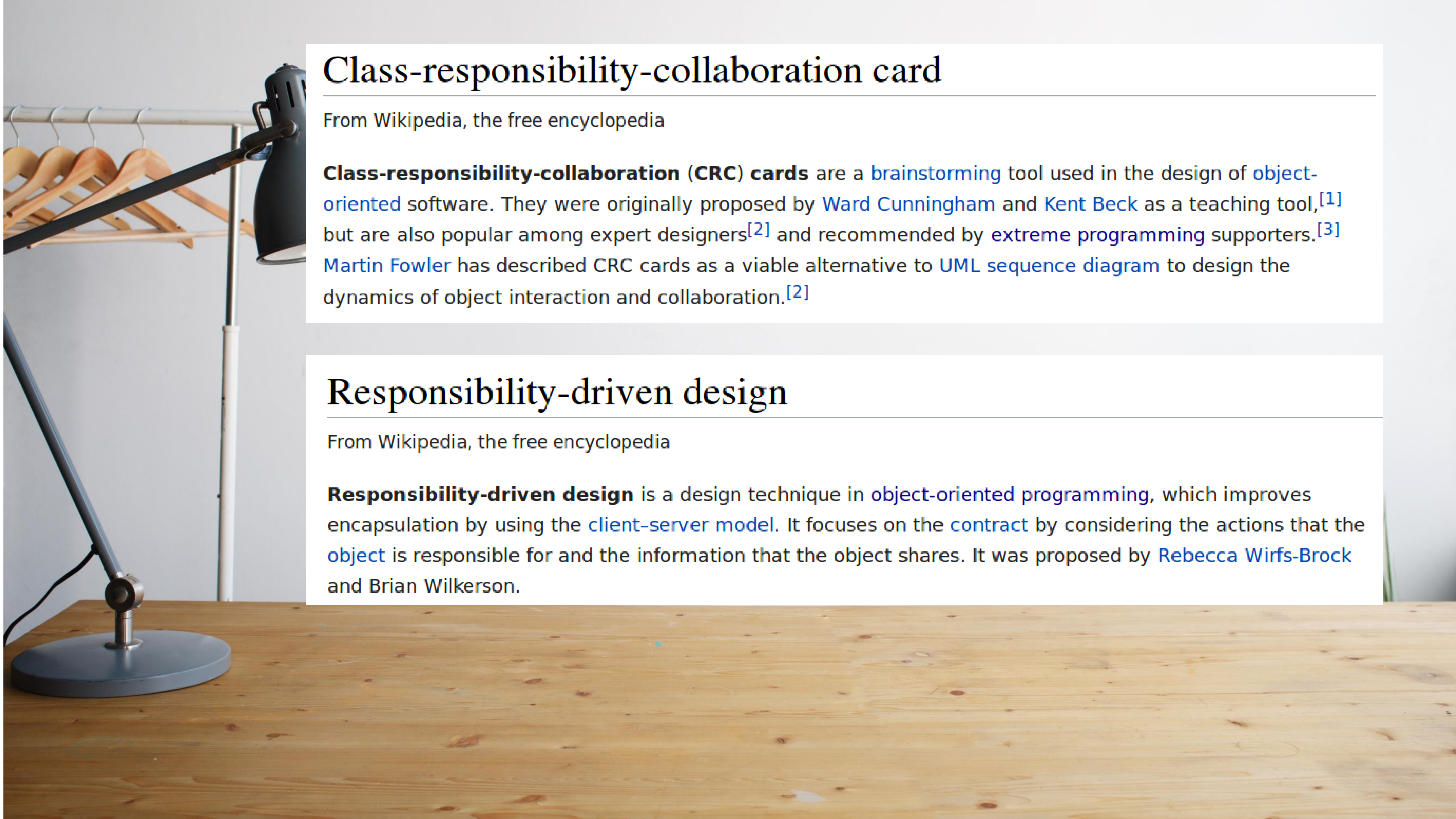
```
class Controller {
  public function showDetails(Request $request) {
    $sql = 'SELECT * FROM products';
  }
}
```

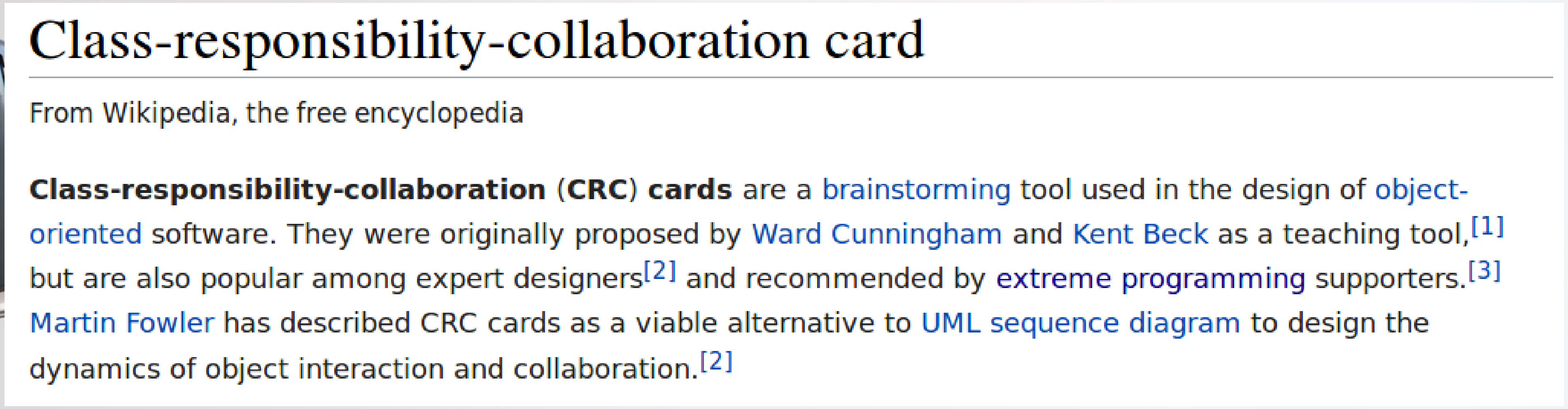
intive

GRASP

1. **Information Expert**
2. **Creator**
3. **Controller**
4. **Low coupling**
5. **High cohesion**
6. **Indirection**
7. Polymorphism
8. Pure Fabrication
9. Protected Variations

Do we need object oriented design?

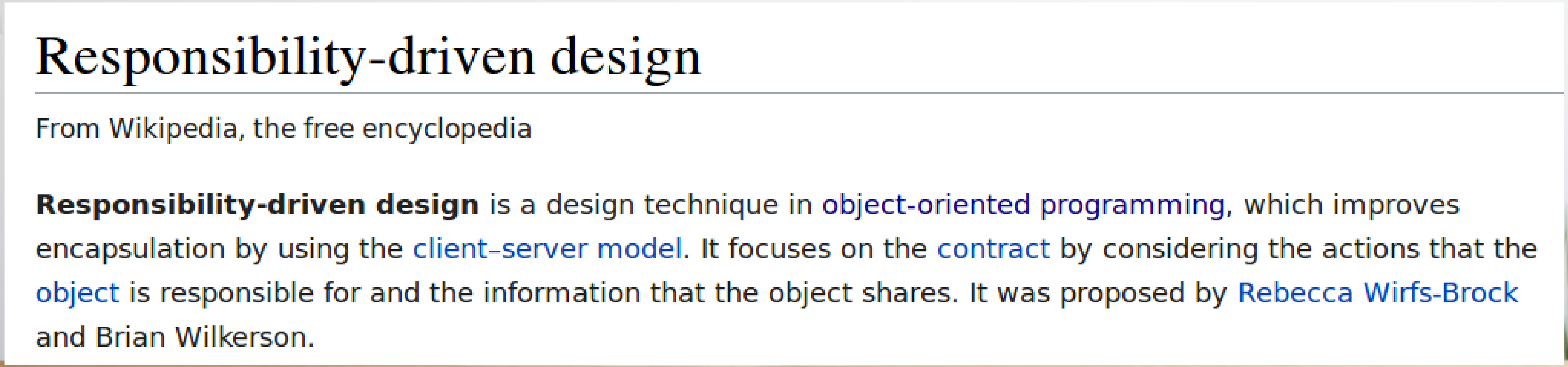




Class-responsibility-collaboration card

From Wikipedia, the free encyclopedia

Class-responsibility-collaboration (**CRC**) **cards** are a brainstorming tool used in the design of object-oriented software. They were originally proposed by Ward Cunningham and Kent Beck as a teaching tool,[1] but are also popular among expert designers[2] and recommended by extreme programming supporters.[3] Martin Fowler has described CRC cards as a viable alternative to UML sequence diagram to design the dynamics of object interaction and collaboration.[2]



Responsibility-driven design

From Wikipedia, the free encyclopedia

Responsibility-driven design is a design technique in object-oriented programming, which improves encapsulation by using the client–server model. It focuses on the contract by considering the actions that the object is responsible for and the information that the object shares. It was proposed by Rebecca Wirfs-Brock and Brian Wilkerson.

intive

phpers

Feedback Time!

Podziel się ze mną opinią na temat talka o GRASP. Nie powinno Ci to zająć więcej niż minutę (chyba że chcesz).

http://bit.ly/grasp-talk

@karol_kreft

Photos

- Photo by La-Rel Easter on Unsplash
- Photo by Feliphe Schiarolli on Unsplash
- Photo by Andrej Lišakov on Unsplash
- Photo by Rohit Choudhari on Unsplash
- Photo by Javier Allegue Barros on Unsplash