Object Calisthenics

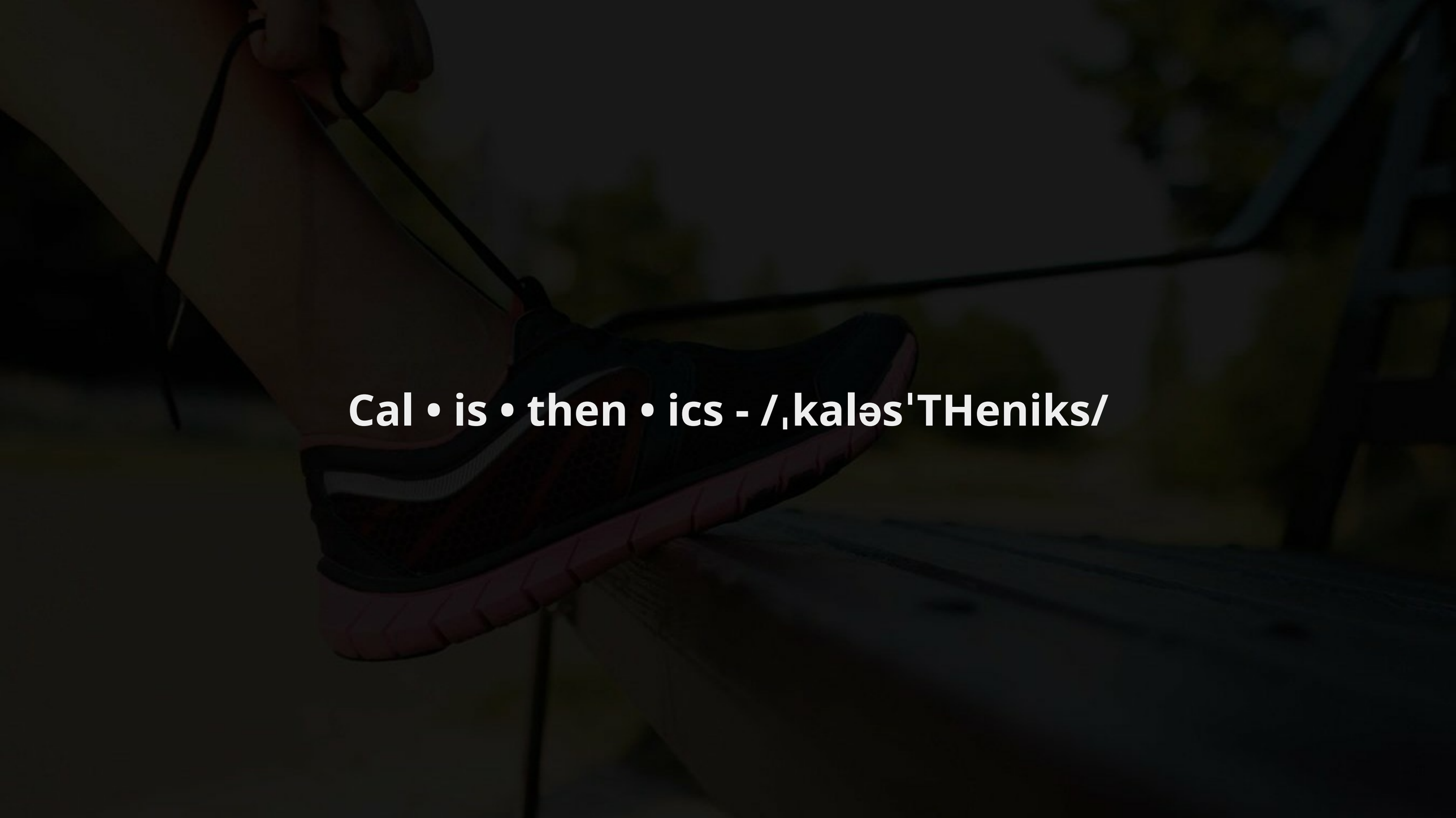
9 steps to better OO code

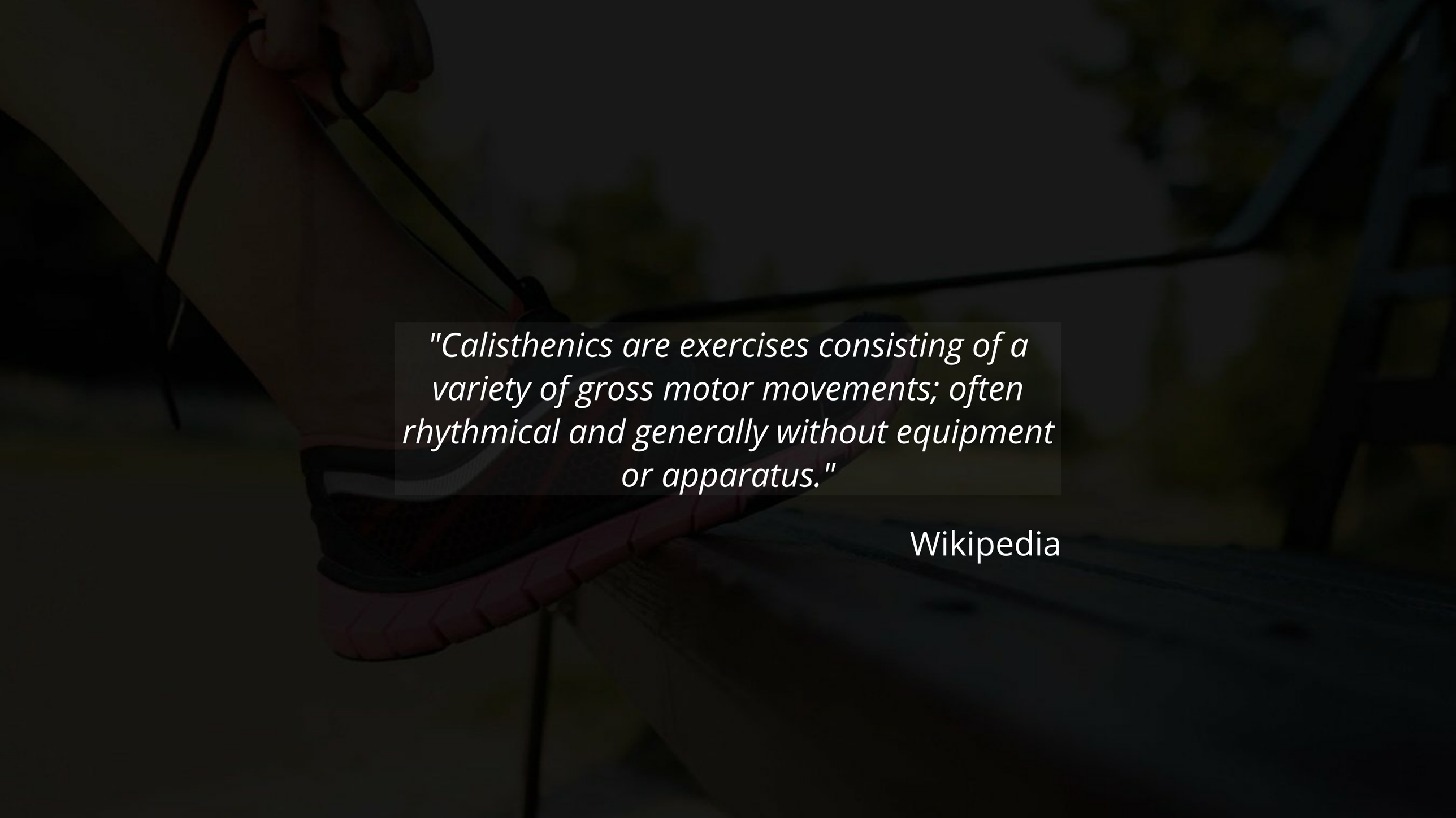
Agenda

Learn how to make our code more:

- readable
- reusable
- testable
- maintainable

Calisthenics

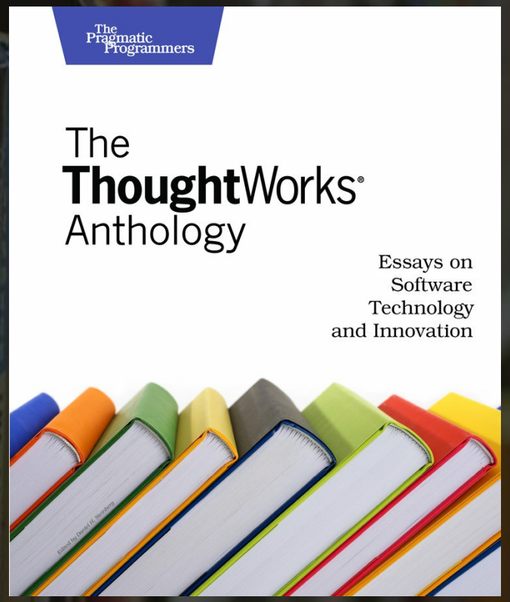

Cal • is • then • ics - /ˌkaləs'THeniks/



"Calisthenics are exercises consisting of a variety of gross motor movements; often rhythmical and generally without equipment or apparatus."

Wikipedia

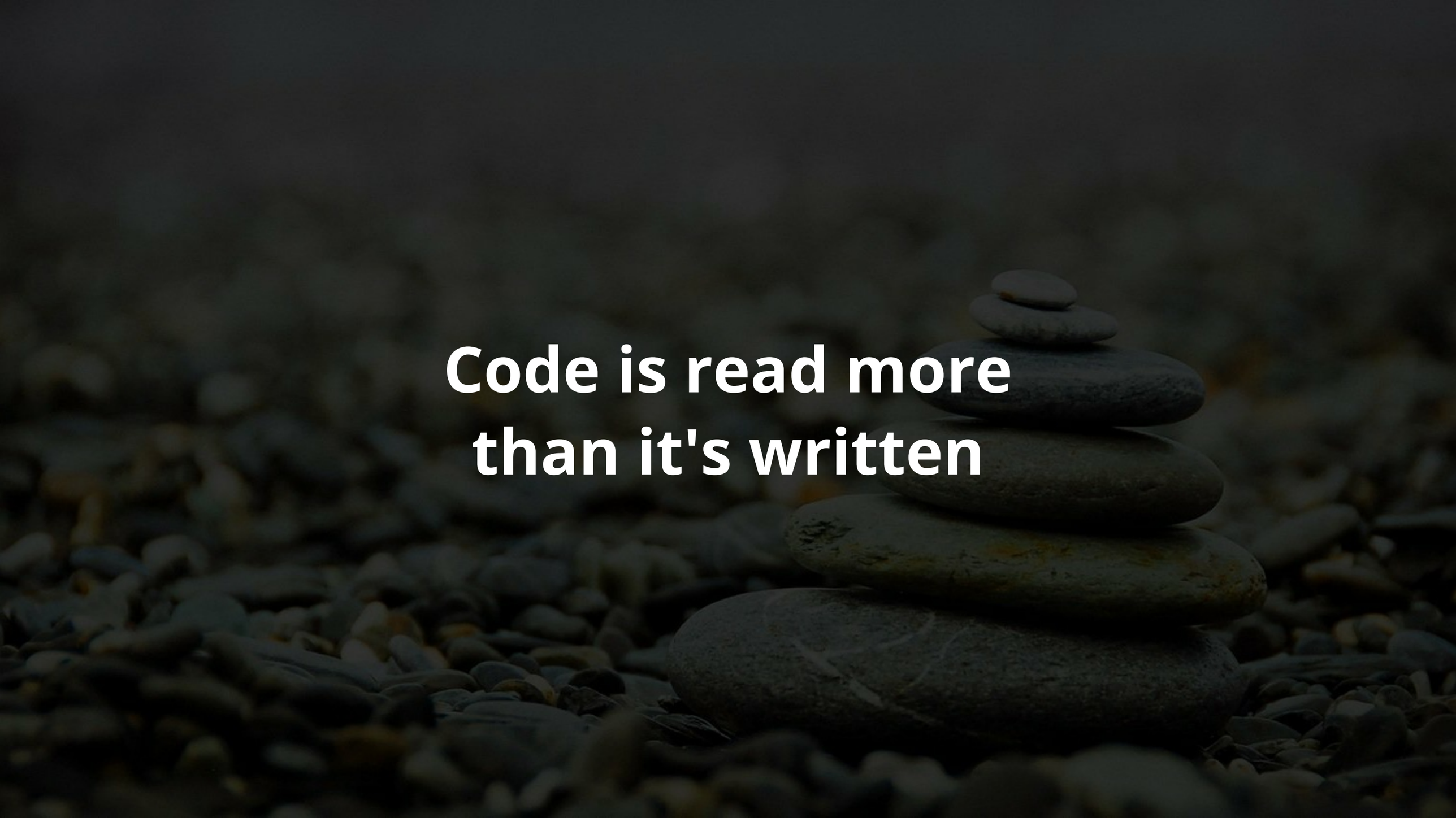
Object Calisthenics



Jeff Bay


Written for Java


Why bother?


Code is read more
than it's written

Rule #1

Only one level of indentation per method

```php
class Board {
    public function __construct(array $data) {
        $buf = '';

        // 0
        for ($i=0; $i<10; $i++) {
            // 1
            for ($j=0; $j<10; $j++) {
                // 2
                $buf .= $data[$i][$j]
            }
        }

        return $buf;
    }
}
```

```
class Board {
    public function __construct(array $data) {
        $buf = '';
        collectRows($buf);

        return $buf;
    }

    private function collectRows($buf) {
        for ($i=0; $i<10; $i++) {
            collectRow($buf, $i);
        }
    }

    private function collectRow($buf, $row) {
        for ($i=0; $i<10; $i++) {
            $buf .= $data[$row][$i];
        }
    }
}
```

Benefits

- Single responsibility
- Better naming
- Shorter methods
- Reusable methods

Rule #2

Do not use *else* keyword

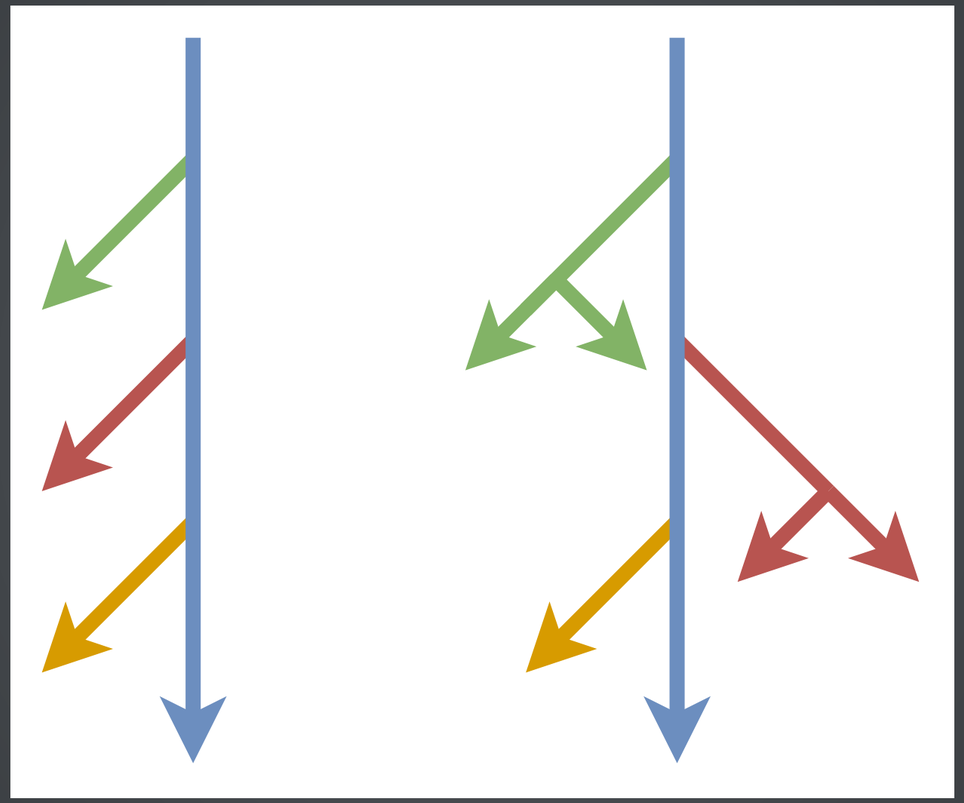
```
if (...) {
    ...
} elseif (...) {
    ...
} elseif (...) {
    ...
} elseif (...) {
    ...
} elseif (...) {
    ...
} elseif (...) {
    ...
} else {
    ...
}
```

```
public function login($username, $password) {
    if ($this->userRepository->isValid($username, $password)) {
        redirect("homepage");
    } else {
        addFlash("error", "Bad credentials");
        redirect("login");
    }
}
```

```
public function login($username, $password) {
    if ($this->userRepository->isValid($username, $password)) {
        return redirect("homepage");
    }

    addFlash("error", "Bad credentials");

    return redirect("login");
}
```



Extract code

Default value

Polymorphism

Strategy pattern

State pattern

Benefits

- Avoids code duplication
- Lower complexity
- Readability

Rule #3

Wrap primitive types if it has behaviour

Value Object in DDD

```php
public function checkDate(int $year, int $month, int $day)
{
    ...
}

// 10th of December or 12th of October?
$validator->checkDate(2016, 10, 12);
```

```
public function checkDate(Year $year,
                          Month $month,
                          Day $day)
{
    ...
}

$validator->checkDate(new Year(2016),
                      new Month(10),
                      new Day(12)
);
```

Benefits

- Encapsulation
- Type hinting
- Attracts similar behaviour

Rule #4

Only one -> per line

OK: Fluent interface

```
$validator->addFilter(new EmailFilter())
          ->addFilter(new NotEmptyFilter());
```

Not OK: getter chain

```
$token = $this->getService(Service::AUTH)
            ->authUser($user, $password)
            ->getResult()
            ->getToken();

// 1. What if non object is returned?
// 2. How about exceptions handling?
```

```
class Location {
    /** @var Piece */
    public current;
}

class Piece {
    /** @var string */
    public representation;
}

class Board {
    public function boardRepresentation(array $board) {
        $buf = '';

        foreach ($board as $field) {
            $buf .= substring($field->current->representation, 0, 1);
        }

        return $buf;
    }
}
```

```
class Location {
    /** @var Piece */
    private $current;

    public function addTo($buf) {
        return $this->current->addTo($buf);
    }
}
class Piece {
    /** @var string */
    private $representation;

    public function character() {
        return substring(representation, 0, 1);
    }

    public function addTo($buf) {
        return $buf . $this->character();
    }
}
class Board {
    public function boardRepresentation(array $board) {
        $buf = '';
        /** @var Location $field */
        foreach ($board as $field) {
            $field->addTo($buf);
        }

        return $buf;
```

Benefits

- Encapsulation
- Demeter's law
- Open/Closed Principle

Rule #5

Do not abbreviate

Why abbreviate?

Name too long?

Too many responsibilities

Split & extract

Duplicated code?

Refactor!

Benefits

- Clear intentions
- Indicate underlying problems

Rule #6

Keep your classes small

What is small class?

- 15-20 lines per method
- 50 lines per class
- 10 classes per module

200 lines per class

10 methods per class

15 classes per namespace

Benefits

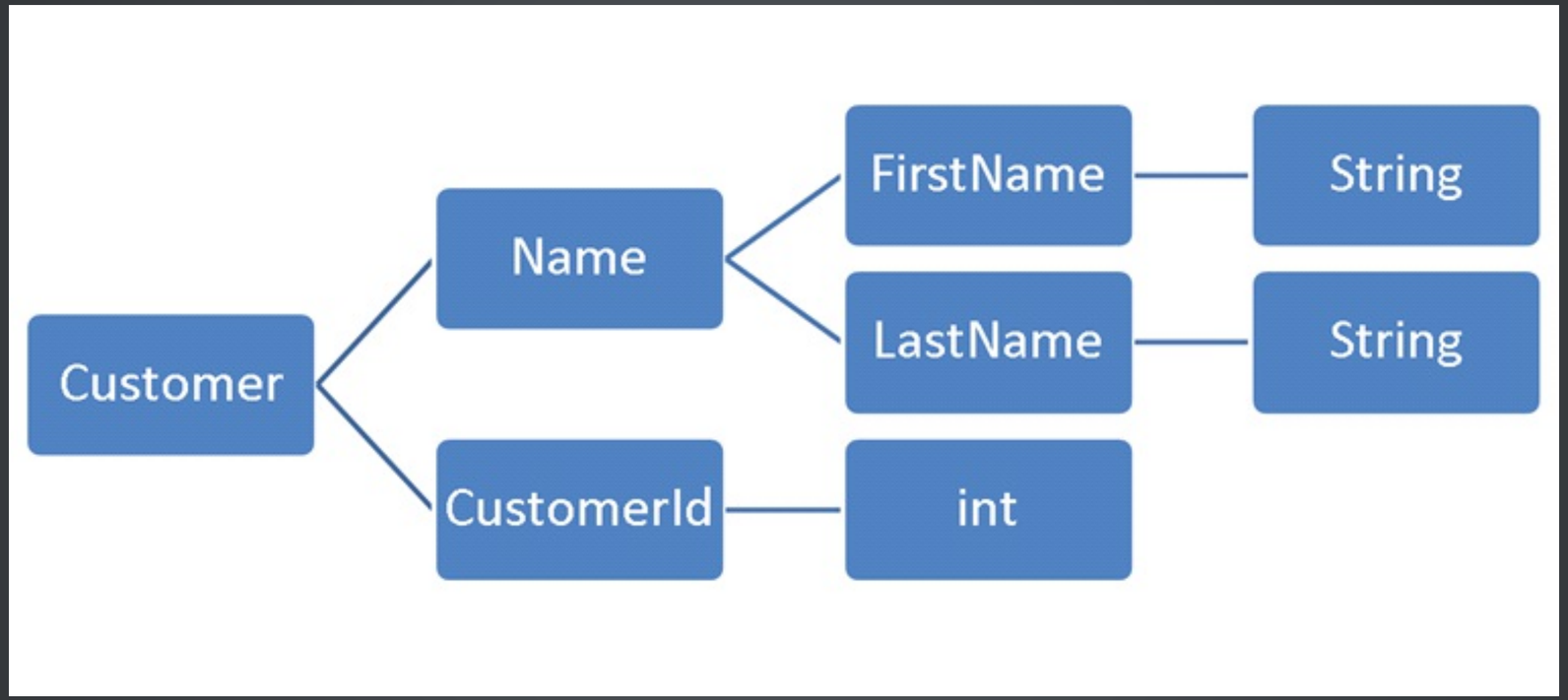
- Single Responsibility
- Smaller namespaces

Rule #7

No more than ~~2~~ 5 instance variable per class

Class should handle single variable state

In some cases it might be two variables


Customer
Name
CustomerId
FirstName
LastName
String
String
int

```
class CartService {
    private $userService;
    private $logger;
    private $cart;
    private $translationService;
    private $entityManager;
    private $authService;

    // ...
}
```

Benefits

- High cohesion
- Encapsulation
- Fewer dependencies

Rule #8

First class collections

Doctrine's ArrayCollection

Benefits

- Single Responsibility

Rule #9

Do not use setters/getters

Accessors are fine

Don't make decisions outside of class

Let class do it's job

Tell, don't ask

```
class Game {
    /** @var int */
    private score;

    public function setScore(score) {
        $this->score = score;
    }

    public function getScore() {
        return $this->score;
    }
}

// Usage
$game->setScore($game->getScore() + ENEMY_DESTROYED_SCORE);
```

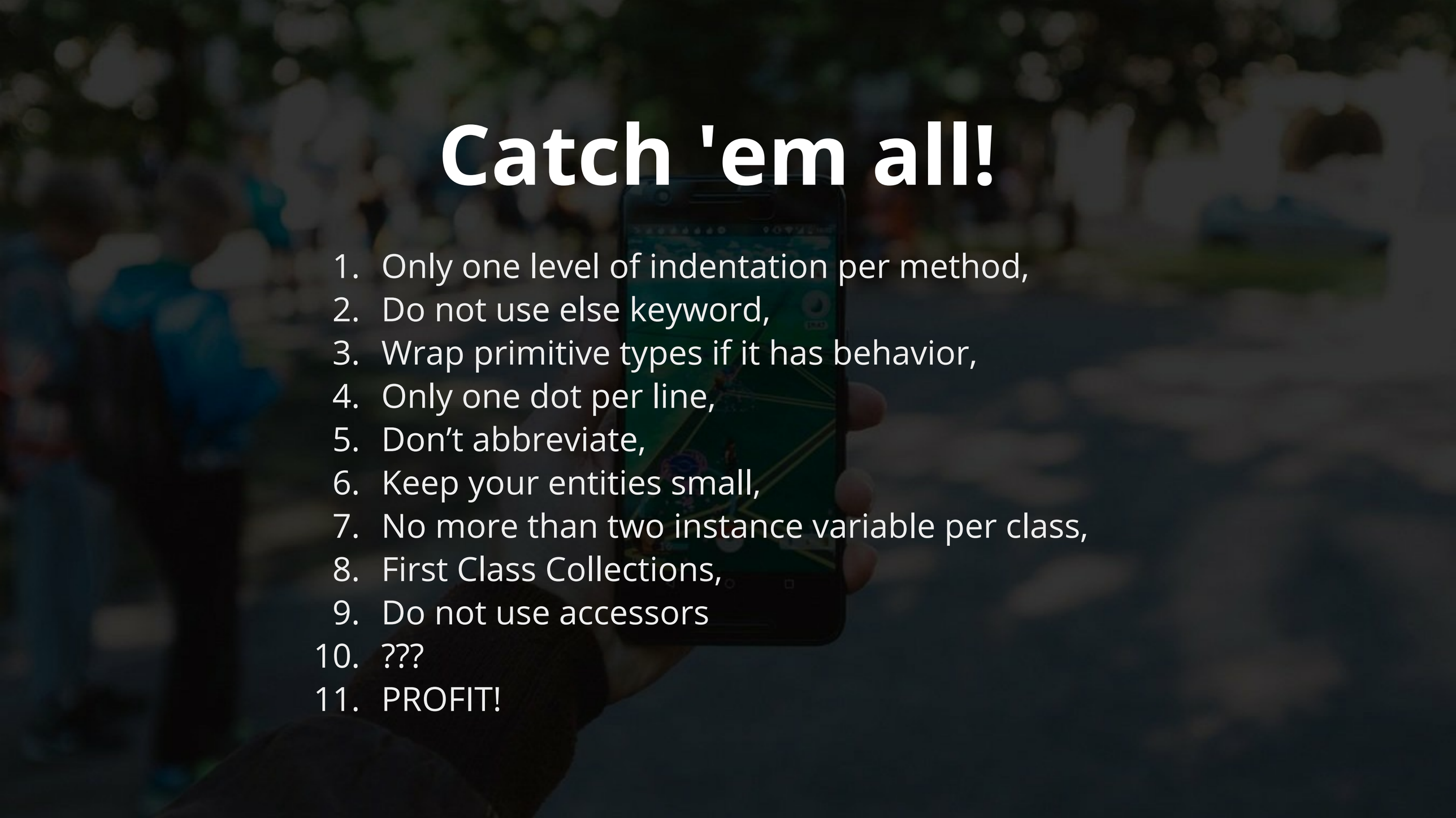
```
class Game {
    /** @var int */
    private score;

    public function addScore($delta) {
        $this->score += $delta;
    }
}

// Usage
$game->addScore(ENEMY_DESTROYED_SCORE);
```

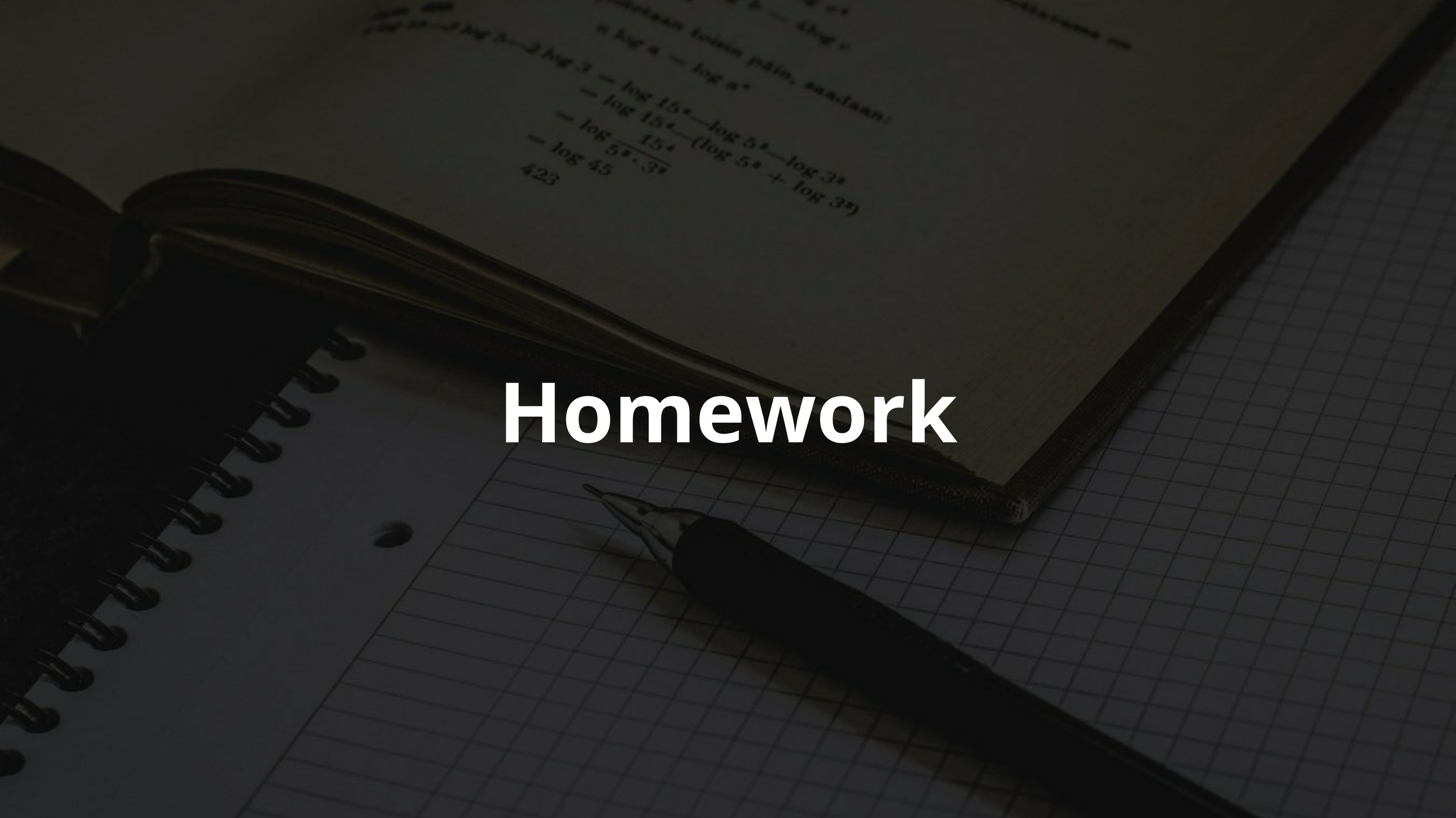
Benefits

- Open/Closed Principle

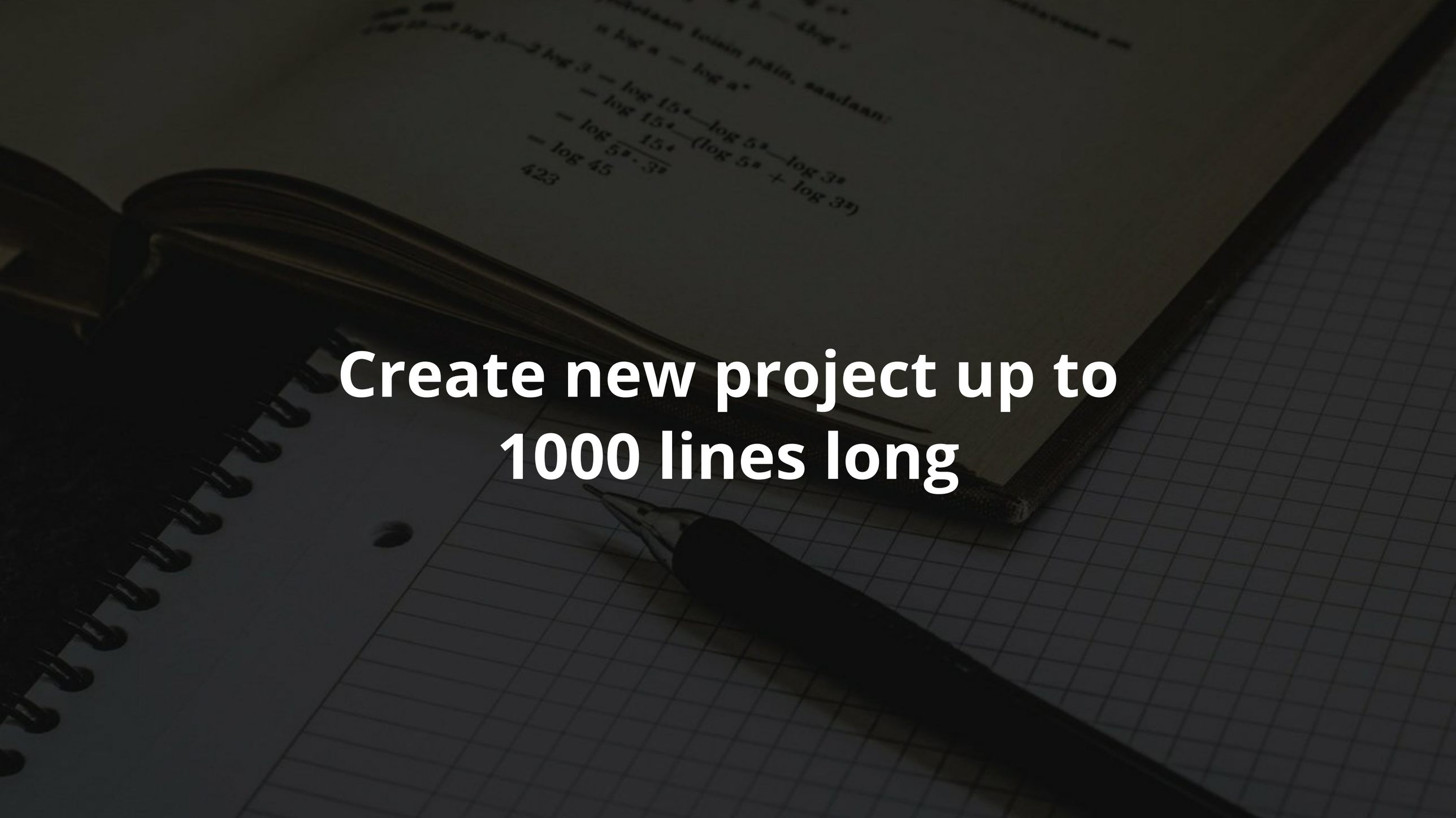


Catch 'em all!

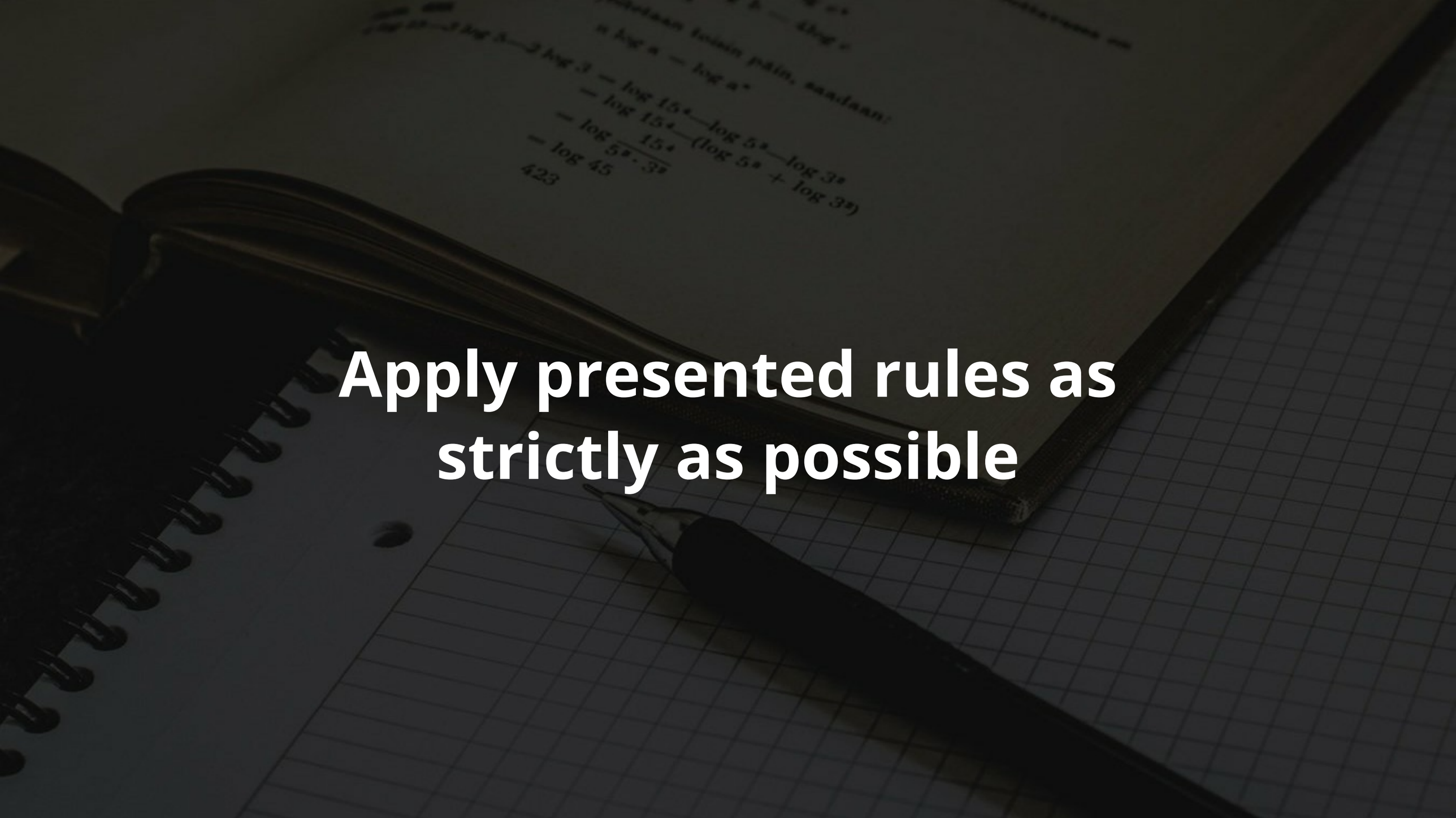
1. Only one level of indentation per method,
2. Do not use else keyword,
3. Wrap primitive types if it has behavior,
4. Only one dot per line,
5. Don’t abbreviate,
6. Keep your entities small,
7. No more than two instance variable per class,
8. First Class Collections,
9. Do not use accessors
10. ???
11. PROFIT!



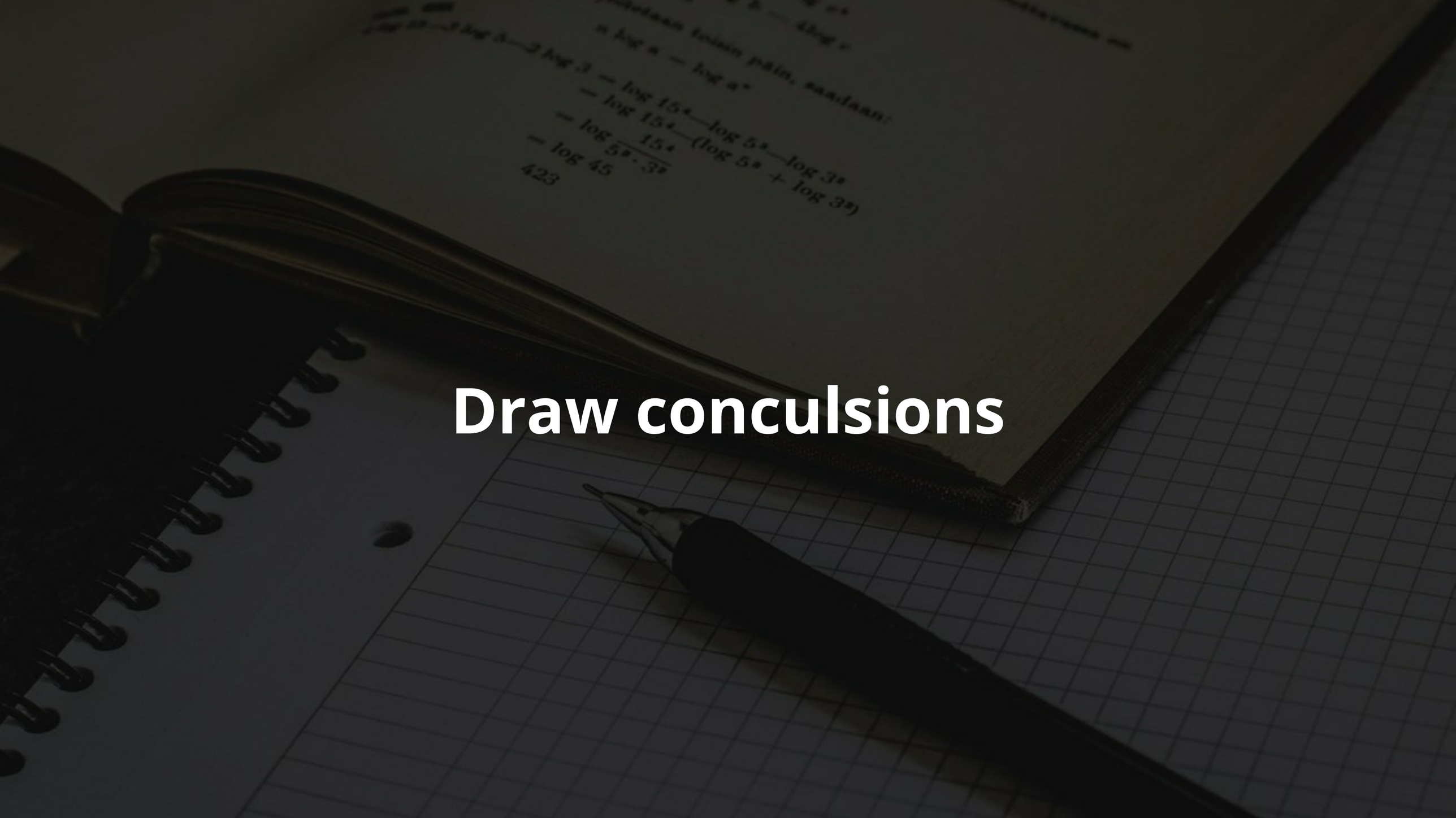
Homework



Create new project up to 1000 lines long



Apply presented rules as strictly as possible



Draw conculsions

Customize these rules


Final thoughts


These are not best practices


These are just guidelines


Use with caution!


Thank you!