

Object Calisthenics

9 steps to better OO code

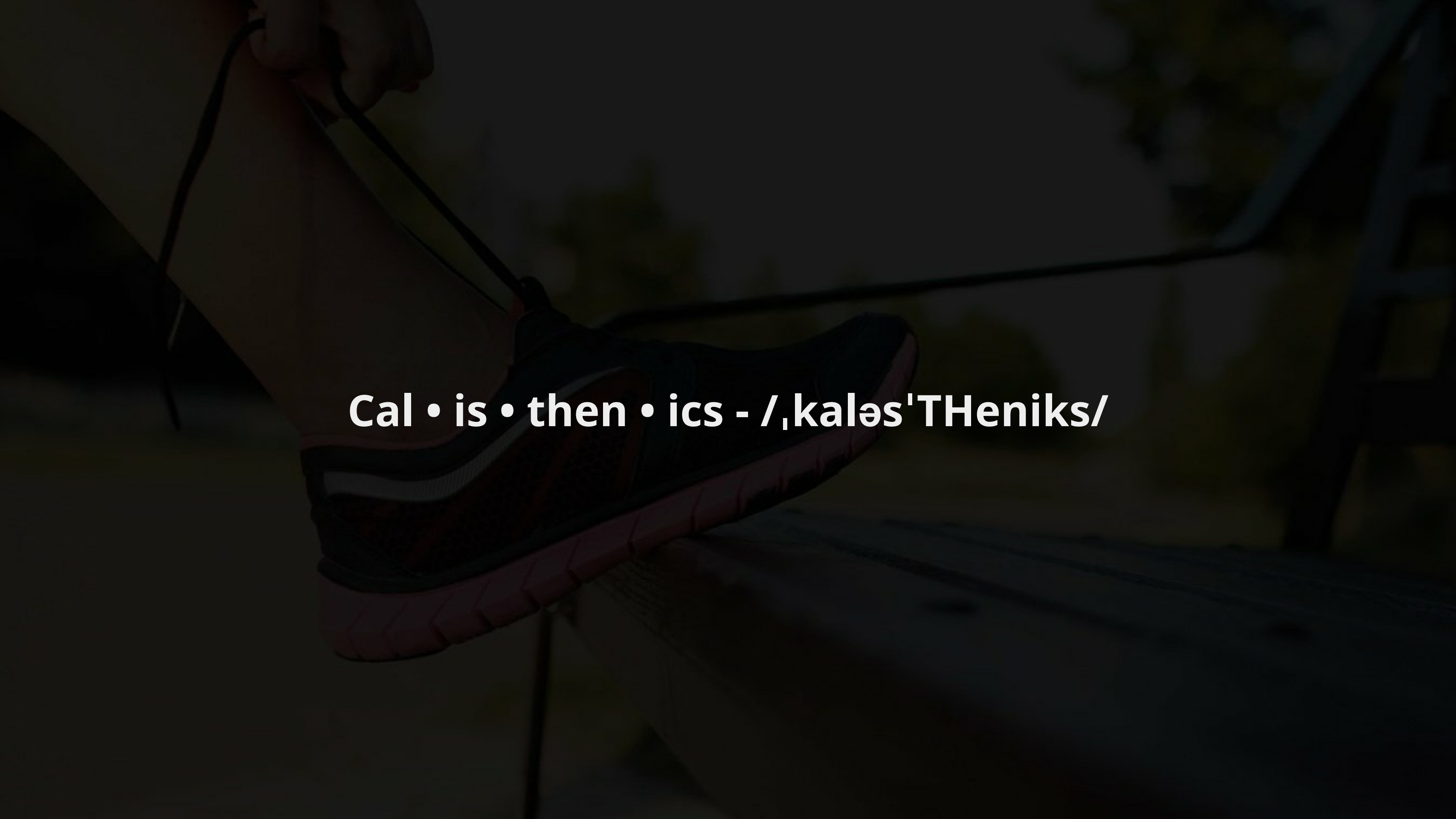
Agenda

Learn how to make our code more:

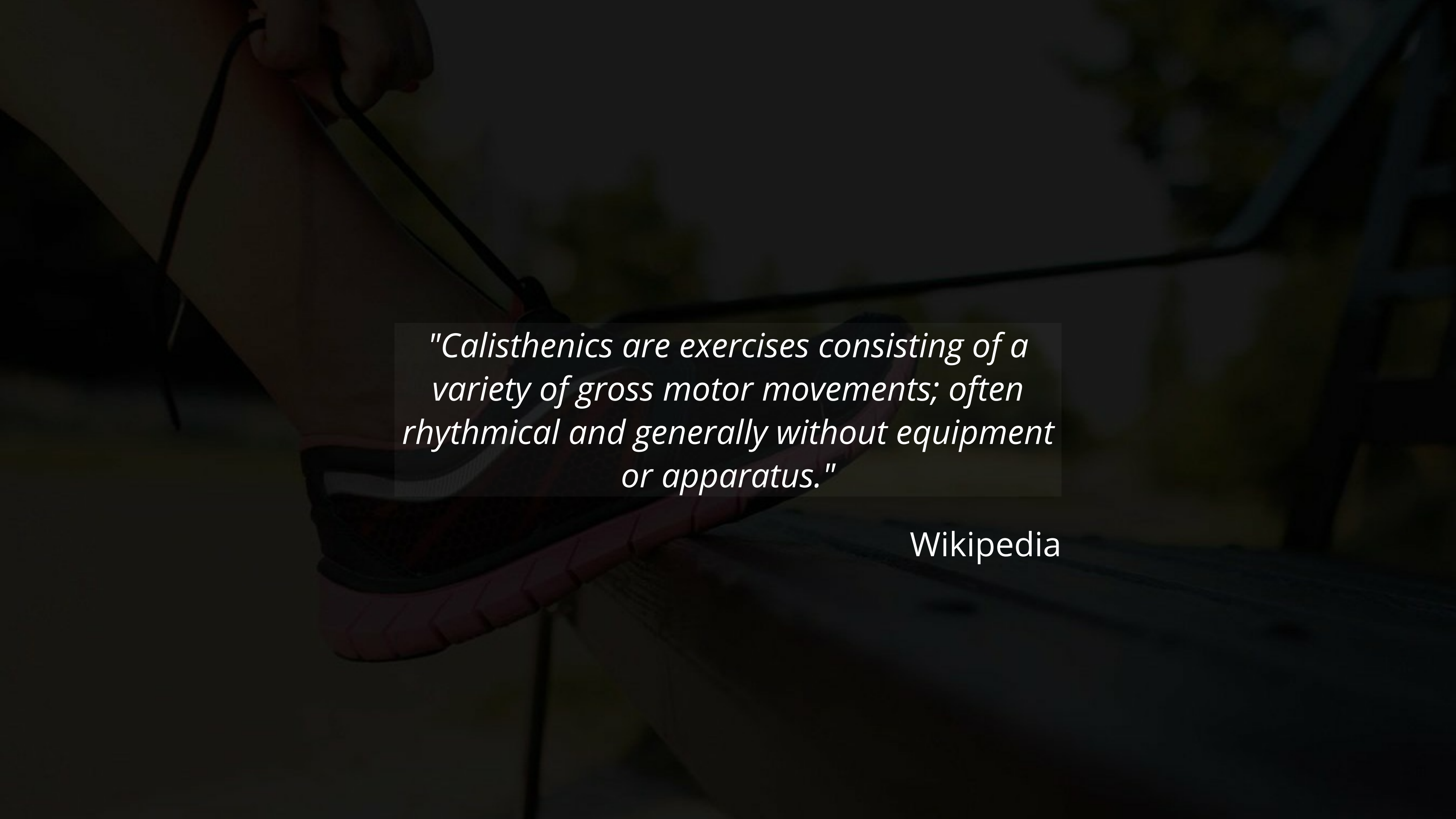
- readable
- reusable
- testable
- maintainable

A close-up, low-angle shot of a person's hand holding a jump rope. The hand is positioned at the top left, with the rope extending diagonally across the frame. The background is dark and out of focus, showing what appears to be a concrete floor and a metal railing. The overall lighting is dim, creating a moody atmosphere.

Calisthenics

A close-up photograph of a person's hand pulling the laces of a black and white sneaker. The shoe is positioned diagonally across the frame. The background is a blurred outdoor setting with a brick wall and some greenery. The text is overlaid in the center of the image.

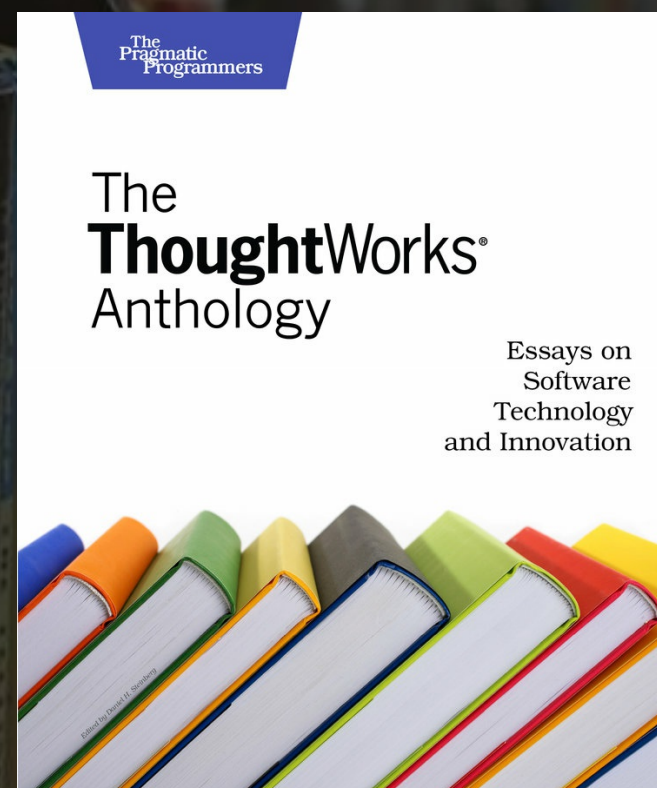
Cal • is • then • ics - /,kaləs'THeniks/

A person is shown from the side, performing a pull-up on a metal bar. The person's arms are extended upwards, gripping the bar. The background is dark and out of focus, suggesting an indoor gym setting. The lighting is dramatic, highlighting the person's arms and the bar.

"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

A stack of smooth, dark grey stones is balanced on a beach. The stones are stacked in a slightly irregular but stable manner, with the top stone being the smallest and the bottom stone being the largest. The background is a soft-focus view of a beach with many other smooth stones scattered around. The overall lighting is dim, creating a moody and contemplative atmosphere. The text "Why bother?" is overlaid in a clean, white, sans-serif font, centered horizontally and slightly above the middle of the image.

Why bother?

A stack of smooth, dark grey stones is balanced on a beach. The stones are stacked in a slightly irregular but stable manner, with the top stone being the smallest and the bottom stone being the largest. The background is a soft-focus view of a beach with many other smooth stones scattered around. The overall lighting is dim, creating a calm and serene atmosphere. The text 'Code is read more than it's written' is overlaid in a clean, white, sans-serif font, centered horizontally and positioned in the upper-middle part of the image.

**Code is read more
than it's written**

Rule #1

Only one level of
indentation per method


```
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

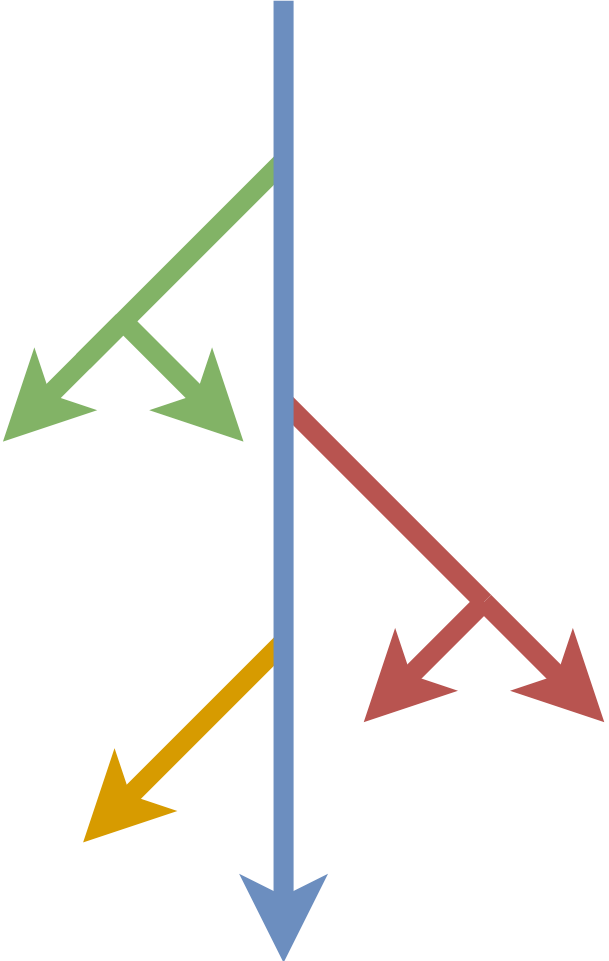
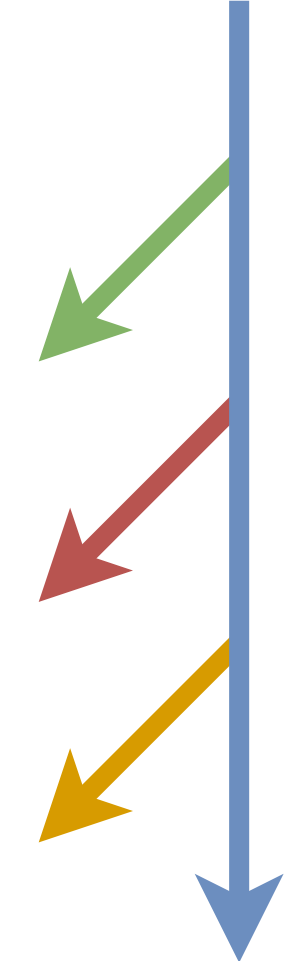
Rule #2

Do not use *else* keyword


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

```
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 \rightarrow 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) {
            $buf .= substr($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 substr($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

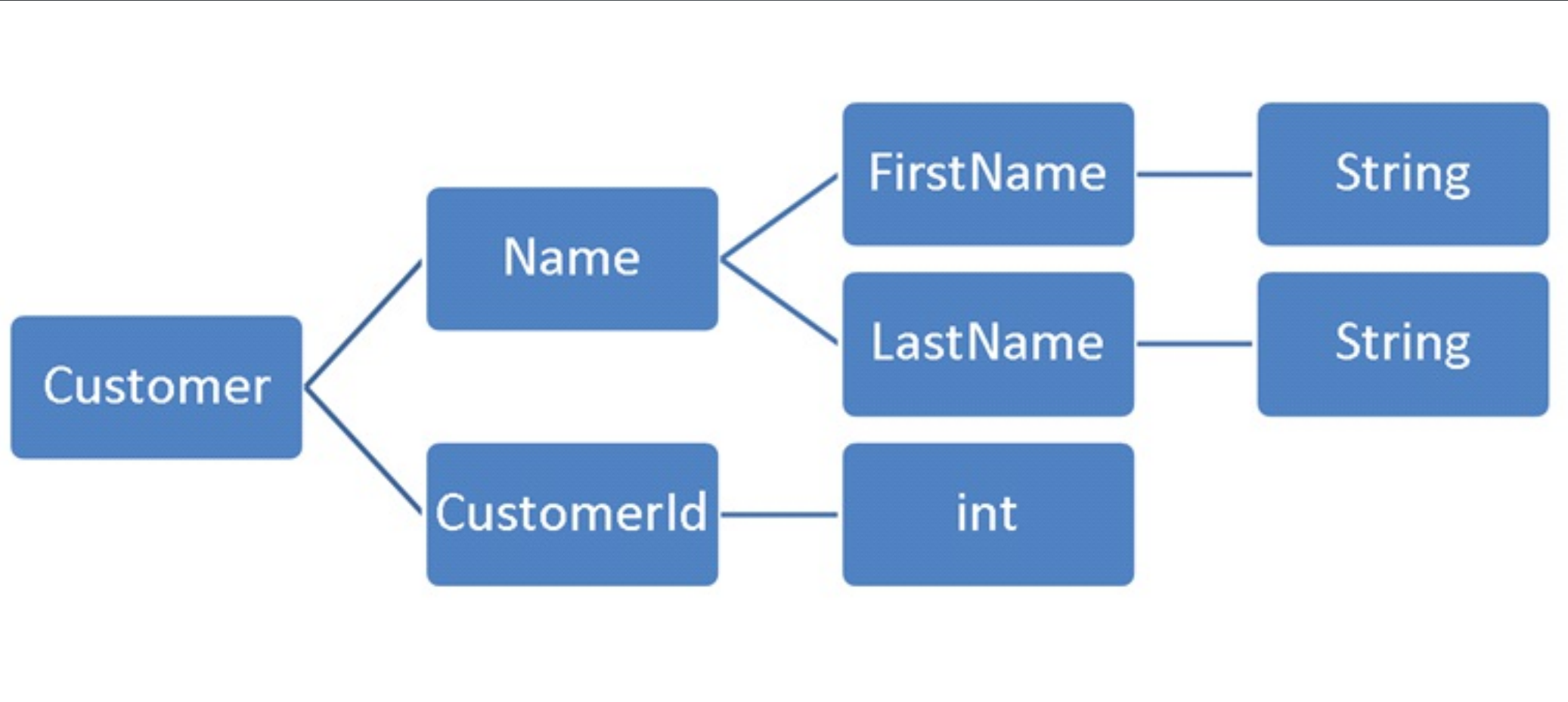
- Single Responsibility
- Smaller namespaces

Rule #7

**No more than 25 instance
variable per class**

**Class should handle single
variable state**

**In some cases it might be
two variables**



```
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

$\log 15 = \log 3 + \log 5$
 $\log 15^2 = 2 \log 15 = 2(\log 3 + \log 5) = 2 \log 3 + 2 \log 5$
 $\log \frac{15^2}{5^2 \cdot 3^2} = \log 15^2 - \log 5^2 - \log 3^2 = 2 \log 3 + 2 \log 5 - 2 \log 5 - 2 \log 3 = 0$
423

**Create new project up to
1000 lines long**

**Apply presented rules as
strictly as possible**

Draw conclusions

Customize these rules

A person is silhouetted against a vast, dark landscape, sitting on a rock and looking out over a sea of clouds or a field of low-lying vegetation. The scene is dimly lit, suggesting dusk or dawn, with a faint glow on the horizon. The overall mood is contemplative and serene.

Final thoughts

A person is silhouetted against a vast, dark landscape, sitting on a rock and looking out over a sea of clouds or a field of low-lying vegetation. The scene is dimly lit, suggesting dusk or dawn, with a faint glow on the horizon. The overall mood is contemplative and serene.

These are not best practices

A person is silhouetted against a vast, dark landscape, sitting on a rock and looking out over a sea of clouds or a field of low-lying vegetation. The scene is dimly lit, suggesting dusk or dawn, with a faint orange glow on the horizon. The overall mood is contemplative and serene.

These are just guidelines



Use with caution!



Thank you!