OBJECT CALISTHENICS

ELTON MINETTO

@EMINETTO

O QUE É?

Conjunto de boas práticas e regras de programação que podem ser aplicadas para melhorar a qualidade do código

Introduzido por Jeff Bay e publicado no livro ThoughtWorks Anthology

Inicialmente criadas tendo como base a linguagem Java e adaptações são necessárias para outros ambientes

ENTÃO...

~~OBJECT CALISTHENICS~~

CODE CALISTHENICS

- One level of indentation per method

- One level of indentation per method
- Don’t use the ELSE keyword

- One level of indentation per method
- Don’t use the ELSE keyword
- Wrap all primitives and Strings in classes

- One level of indentation per method
- Don’t use the ELSE keyword
- Wrap all primitives and Strings in classes
- First class collections

- One level of indentation per method
- Don’t use the ELSE keyword
- Wrap all primitives and Strings in classes
- First class collections
- One dot per line

- One level of indentation per method
- Don’t use the ELSE keyword
- Wrap all primitives and Strings in classes
- First class collections
- One dot per line
- ~~Don’t abbreviate~~

- One level of indentation per method
- Don’t use the ELSE keyword
- Wrap all primitives and Strings in classes
- First class collections
- One dot per line
- ~~Don’t abbreviate~~
- Keep all classes less than 50 lines

- One level of indentation per method
- Don’t use the ELSE keyword
- Wrap all primitives and Strings in classes
- First class collections
- One dot per line
- ~~Don’t abbreviate~~
- Keep all classes less than 50 lines
- ~~No classes with more than two instance variables~~

- One level of indentation per method
- Don’t use the ELSE keyword
- Wrap all primitives and Strings in classes
- First class collections
- One dot per line
- ~~Don’t abbreviate~~
- Keep all classes less than 50 lines
- ~~No classes with more than two instance variables~~
- ~~No getters or setters~~

ONE LEVEL OF INDENTATION PER METHOD

Aplicar esta regra permite que o nosso código seja mais legível

```go
type board struct {
    data [][]string
}

func NewBoard(data [][]string) *board {} //código omitido pra ficar legal no slide ;)

func (b *board) Board() string {
    var buffer = &bytes.Buffer{}

    // level 0
    for i := 0; i < 10; i++ {
        // level 1
        for j := 0; j < 10; j++ {
            // level 2
            buffer.WriteString(b.data[i][j])
        }
        buffer.WriteString("\n")
    }

    return buffer.String()
}
```

```go
type board struct {
    data [][]string
}

func NewBoard(data [][]string) *board {} //código omitido pra ficar legal no slide ;)

func (b *board) Board() string {
    var buffer = &bytes.Buffer{}

    // level 0
    for i := 0; i < 10; i++ {
        // level 1
        for j := 0; j < 10; j++ {
            // level 2
            buffer.WriteString(b.data[i][j])
        }
        buffer.WriteString("\n")
    }

    return buffer.String()
}
```

```go
type board struct {
    data [][]string
}

func NewBoard(data [][]string) *board {} //código omitido pra ficar legal no slide ;)

func (b *board) Board() string {
    var buffer = &bytes.Buffer{}

    // level 0
    for i := 0; i < 10; i++ {
        // level 1
        for j := 0; j < 10; j++ {
            // level 2
            buffer.WriteString(b.data[i][j])
        }
        buffer.WriteString("\n")
    }

    return buffer.String()
}
```

```go
type board struct {
    data [][]string
}

func NewBoard(data [][]string) *board {}

func (b *board) Board() string {
    var buffer = &bytes.Buffer{}

    b.collectRows(buffer)

    return buffer.String()
}

func (b *board) collectRows(buffer *bytes.Buffer) {
    for i := 0; i < 10; i++ {
        b.collectRow(buffer, i)
    }
}

func (b *board) collectRow(buffer *bytes.Buffer, row int) {
    for j := 0; j < 10; j++ {
        buffer.WriteString(b.data[row][j])
    }

    buffer.WriteString("\n")
}
```

```
type board struct {
    data [][]string
}

func NewBoard(data [][]string) *board {}

func (b *board) Board() string {
    var buffer = &bytes.Buffer{}

    b.collectRows(buffer)

    return buffer.String()
}

func (b *board) collectRows(buffer *bytes.Buffer) {
    for i := 0; i < 10; i++ {
        b.collectRow(buffer, i)
    }
}

func (b *board) collectRow(buffer *bytes.Buffer, row int) {
    for j := 0; j < 10; j++ {
        buffer.WriteString(b.data[row][j])
    }

    buffer.WriteString("\n")
}
```

```go
type board struct {
    data [][]string
}

func NewBoard(data [][]string) *board {}

func (b *board) Board() string {
    var buffer = &bytes.Buffer{}

    b.collectRows(buffer)

    return buffer.String()
}

func (b *board) collectRows(buffer *bytes.Buffer) {
    for i := 0; i < 10; i++ {
        b.collectRow(buffer, i)
    }
}

func (b *board) collectRow(buffer *bytes.Buffer, row int) {
    for j := 0; j < 10; j++ {
        buffer.WriteString(b.data[row][j])
    }

    buffer.WriteString("\n")
}
```

```
type board struct {
    data [][]string
}

func NewBoard(data [][]string) *board {}

func (b *board) Board() string {
    var buffer = &bytes.Buffer{}

    b.collectRows(buffer)

    return buffer.String()
}

func (b *board) collectRows(buffer *bytes.Buffer) {
    for i := 0; i < 10; i++ {
        b.collectRow(buffer, i)
    }
}

func (b *board) collectRow(buffer *bytes.Buffer, row int) {
    for j := 0; j < 10; j++ {
        buffer.WriteString(b.data[row][j])
    }

    buffer.WriteString("\n")
}
```

```go
type board struct {
    data [][]string
}

func NewBoard(data [][]string) *board {}

func (b *board) Board() string {
    var buffer = &bytes.Buffer{}

    b.collectRows(buffer)

    return buffer.String()
}

func (b *board) collectRows(buffer *bytes.Buffer) {
    for i := 0; i < 10; i++ {
        b.collectRow(buffer, i)
    }
}

func (b *board) collectRow(buffer *bytes.Buffer, row int) {
    for j := 0; j < 10; j++ {
        buffer.WriteString(b.data[row][j])
    }

    buffer.WriteString("\n")
}
```

```php
<?php
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email'] < 65)) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: '.$_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 64 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```

```php
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 64 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```

DON’T USE THE ELSE KEYWORD

A ideia deste item é evitarmos o uso da palavra chave else, gerando um código limpo e mais rápido, pois tem menos fluxos de execução

```go
type loginService struct {
    userRepository *repository.UserRepository
}

func NewLoginService() *loginService {
    return &loginService{
        userRepository: repository.NewUserRepository(),
    }
}

func (l *loginService) Login(userName, password string) {
    if l.userRepository.IsValid(userName, password) {
        redirect("homepage")
    } else {
        addFlash("error", "Bad credentials")
        redirect("login")
    }
}

func redirect(page string) {}

func addFlash(msgType, msg string) {}
```

```go
type loginService struct {
    userRepository *repository.UserRepository
}

func NewLoginService() *loginService {
    return &loginService{
        userRepository: repository.NewUserRepository(),
    }
}

func (l *loginService) Login(userName, password string) {
    if l.userRepository.IsValid(userName, password) {
        redirect("homepage")
    } else {
        addFlash("error", "Bad credentials")
        redirect("login")
    }
}

func redirect(page string) {}

func addFlash(msgType, msg string) {}
```

```go
type loginService struct {
    userRepository *repository.UserRepository
}

func NewLoginService() *loginService {
    return &loginService{
        userRepository: repository.NewUserRepository(),
    }
}

func (l *loginService) Login(userName, password string) {
    if l.userRepository.IsValid(userName, password) {
        redirect("homepage")
    } else {
        addFlash("error", "Bad credentials")
        redirect("login")
    }
}

func redirect(page string) {}

func addFlash(msgType, msg string) {}
```

```
func (l *loginService) Login(userName, password string) {
    if l.userRepository.IsValid(userName, password) {
        redirect("homepage")
        return
    }
    addFlash("error", "Bad credentials")
    redirect("login")
}
```

WRAP ALL PRIMITIVES AND STRINGS IN CLASSES

Tipos primitivos que possuem comportamento devem ser encapsulados, no nosso caso, em structs ou types e não em classes. Desta forma, a lógica do comportamento fica encapsulado e de fácil manutenção

```go
type order struct {
    pid int64
    cid int64
}

func CreateOrder(pid int64, cid int64) order {
    return order{
        pid: pid, cid: cid,
    }
}

func (o order) Submit() (int64, error) {
    // do some logic

    return int64(3252345234), nil
}
```

```go
type order struct {
    pid int64
    cid int64
}

func CreateOrder(pid int64, cid int64) order {
    return order{
        pid: pid, cid: cid,
    }
}

func (o order) Submit() (int64, error) {
    // do some logic

    return int64(3252345234), nil
}
```

```go
type order struct {
    pid int64
    cid int64
}

func CreateOrder(pid int64, cid int64) order {
    return order{
        pid: pid, cid: cid,
    }
}

func (o order) Submit() (int64, error) {
    // do some logic

    return int64(3252345234), nil
}
```

```go
type order struct {
    pid productID
    cid customerID
}

type productID int64
// some functions on productID type

type customerID int64
// some functions on customerID type

type orderID int64
func (oid orderID) String() string {
    return strconv.FormatInt(int64(oid), 10)
}
// some other functions on orderID type

func CreateOrder(pid int64, cid int64) order {
    return order{
        pid: productID(pid), cid: customerID(cid),
    }
}

func (o order) Submit() (orderID, error) {
    // do some logic
    return orderID(int64(3252345234)), nil
}
```

```go
type order struct {
    pid productID
    cid customerID
}

type productID int64
// some functions on productID type

type customerID int64
// some functions on customerID type

type orderID int64
func (oid orderID) String() string {
    return strconv.FormatInt(int64(oid), 10)
}
// some other functions on orderID type

func CreateOrder(pid int64, cid int64) order {
    return order{
        pid: productID(pid), cid: customerID(cid),
    }
}

func (o order) Submit() (orderID, error) {
    // do some logic
    return orderID(int64(3252345234)), nil
}
```

```go
type order struct {
    pid productID
    cid customerID
}

type productID int64
// some functions on productID type

type customerID int64
// some functions on customerID type

type orderID int64
func (oid orderID) String() string {
    return strconv.FormatInt(int64(oid), 10)
}
// some other functions on orderID type

func CreateOrder(pid int64, cid int64) order {
    return order{
        pid: productID(pid), cid: customerID(cid),
    }
}

func (o order) Submit() (orderID, error) {
    // do some logic
    return orderID(int64(3252345234)), nil
}
```

```go
type order struct {
    pid productID
    cid customerID
}

type productID int64
// some functions on productID type

type customerID int64
// some functions on customerID type

type orderID int64
func (oid orderID) String() string {
    return strconv.FormatInt(int64(oid), 10)
}
// some other functions on orderID type

func CreateOrder(pid int64, cid int64) order {
    return order{
        pid: productID(pid), cid: customerID(cid),
    }
}

func (o order) Submit() (orderID, error) {
    // do some logic
    return orderID(int64(3252345234)), nil
}
```

```go
type order struct {
    pid productID
    cid customerID
}

type productID int64
// some functions on productID type

type customerID int64
// some functions on customerID type

type orderID int64
func (oid orderID) String() string {
    return strconv.FormatInt(int64(oid), 10)
}
// some other functions on orderID type

func CreateOrder(pid int64, cid int64) order {
    return order{
        pid: productID(pid), cid: customerID(cid),
    }
}

func (o order) Submit() (orderID, error) {
    // do some logic
    return orderID(int64(3252345234)), nil
}
```

```go
type order struct {
    pid productID
    cid customerID
}

type productID int64
// some functions on productID type

type customerID int64
// some functions on customerID type

type orderID int64
func (oid orderID) String() string {
    return strconv.FormatInt(int64(oid), 10)
}
// some other functions on orderID type

func CreateOrder(pid int64, cid int64) order {
    return order{
        pid: productID(pid), cid: customerID(cid),
    }
}

func (o order) Submit() (orderID, error) {
    // do some logic
    return orderID(int64(3252345234)), nil
}
```

```go
type order struct {
    pid productID
    cid customerID
}

type productID int64
// some functions on productID type

type customerID int64
// some functions on customerID type

type orderID int64
func (oid orderID) String() string {
    return strconv.FormatInt(int64(oid), 10)
}
// some other functions on orderID type

func CreateOrder(pid int64, cid int64) order {
    return order{
        pid: productID(pid), cid: customerID(cid),
    }
}

func (o order) Submit() (orderID, error) {
    // do some logic
    return orderID(int64(3252345234)), nil
}
```

FIRST CLASS COLLECTIONS

Se você tiver um conjunto de elementos e quiser manipulá-los, crie uma estrutura dedicada para essa coleção. Assim comportamentos relacionados à coleção serão implementados por sua própria estrutura como filtros, ordenação, storage, etc.

```go
type person struct {
    name    string
    friends []string
}

func NewPerson(name string) *person {
    return &person{
        name:    name,
        friends: []string{},
    }
}

func (p *person) AddFriend(name string) {
    p.friends = append(p.friends, name)
}

func (p *person) RemoveFriend(name string) {
    new := []string{}
    for _, friend := range p.friends {
        if friend != name {
            new = append(new, friend)
        }
    }
    p.friends = new
}

func (p *person) GetFriends() []string {
    return p.friends
}

func (p *person) String() string {
    return fmt.Sprintf("%s %v", p.name, p.friends)
}
```

```go
type person struct {
    name    string
    friends []string
}

func NewPerson(name string) *person {
    return &person{
        name:    name,
        friends: []string{},
    }
}

func (p *person) AddFriend(name string) {
    p.friends = append(p.friends, name)
}

func (p *person) RemoveFriend(name string) {
    new := []string{}
    for _, friend := range p.friends {
        if friend != name {
            new = append(new, friend)
        }
    }
    p.friends = new
}

func (p *person) GetFriends() []string {
    return p.friends
}

func (p *person) String() string {
    return fmt.Sprintf("%s %v", p.name, p.friends)
}
```

```go
type person struct {
    name    string
    friends []string
}

func NewPerson(name string) *person {
    return &person{
        name:    name,
        friends: []string{},
    }
}

func (p *person) AddFriend(name string) {
    p.friends = append(p.friends, name)
}

func (p *person) RemoveFriend(name string) {
    new := []string{}
    for _, friend := range p.friends {
        if friend != name {
            new = append(new, friend)
        }
    }
    p.friends = new
}

func (p *person) GetFriends() []string {
    return p.friends
}

func (p *person) String() string {
    return fmt.Sprintf("%s %v", p.name, p.friends)
}
```

```go
type person struct {
    name    string
    friends []string
}

func NewPerson(name string) *person {
    return &person{
        name:    name,
        friends: []string{},
    }
}

func (p *person) AddFriend(name string) {
    p.friends = append(p.friends, name)
}

func (p *person) RemoveFriend(name string) {
    new := []string{}
    for _, friend := range p.friends {
        if friend != name {
            new = append(new, friend)
        }
    }
    p.friends = new
}

func (p *person) GetFriends() []string {
    return p.friends
}

func (p *person) String() string {
    return fmt.Sprintf("%s %v", p.name, p.friends)
}
```

```go
type person struct {
    name    string
    friends []string
}

func NewPerson(name string) *person {
    return &person{
        name:    name,
        friends: []string{},
    }
}

func (p *person) AddFriend(name string) {
    p.friends = append(p.friends, name)
}

func (p *person) RemoveFriend(name string) {
    new := []string{}
    for _, friend := range p.friends {
        if friend != name {
            new = append(new, friend)
        }
    }
    p.friends = new
}

func (p *person) GetFriends() []string {
    return p.friends
}

func (p *person) String() string {
    return fmt.Sprintf("%s %v", p.name, p.friends)
}
```

```go
type person struct {
    name    string
    friends []string
}

func NewPerson(name string) *person {
    return &person{
        name:    name,
        friends: []string{},
    }
}

func (p *person) AddFriend(name string) {
    p.friends = append(p.friends, name)
}

func (p *person) RemoveFriend(name string) {
    new := []string{}
    for _, friend := range p.friends {
        if friend != name {
            new = append(new, friend)
        }
    }
    p.friends = new
}

func (p *person) GetFriends() []string {
    return p.friends
}

func (p *person) String() string {
    return fmt.Sprintf("%s %v", p.name, p.friends)
}
```

```go
type friends struct {
    data []string
}

type person struct {
    name    string
    friends *friends
}

func NewFriends() *friends {} //código omitido pra ficar legal no slide ;)

func (f *friends) Add(name string) {} //código omitido pra ficar legal no slide ;)

func (f *friends) Remove(name string) {} //código omitido pra ficar legal no slide ;)

func (f *friends) String() string {} //código omitido pra ficar legal no slide ;)

func NewPerson(name string) *person {} //código omitido pra ficar legal no slide ;)

func (p *person) AddFriend(name string) {
    p.friends.Add(name)
}

func (p *person) RemoveFriend(name string) {
    p.friends.Remove(name)
}

func (p *person) GetFriends() *friends {
    return p.friends
}

func (p *person) String() string {
    return fmt.Sprintf("%s [%v]", p.name, p.friends)
}
```

```go
type friends struct {
    data []string
}

type person struct {
    name    string
    friends *friends
}

func NewFriends() *friends {} //código omitido pra ficar legal no slide ;)

func (f *friends) Add(name string) {} //código omitido pra ficar legal no slide ;)

func (f *friends) Remove(name string) {} //código omitido pra ficar legal no slide ;)

func (f *friends) String() string {} //código omitido pra ficar legal no slide ;)

func NewPerson(name string) *person {} //código omitido pra ficar legal no slide ;)

func (p *person) AddFriend(name string) {
    p.friends.Add(name)
}

func (p *person) RemoveFriend(name string) {
    p.friends.Remove(name)
}

func (p *person) GetFriends() *friends {
    return p.friends
}

func (p *person) String() string {
    return fmt.Sprintf("%s [%v]", p.name, p.friends)
}
```

```go
type friends struct {
    data []string
}

type person struct {
    name    string
    friends *friends
}

func NewFriends() *friends {} //código omitido pra ficar legal no slide ;)

func (f *friends) Add(name string) {} //código omitido pra ficar legal no slide ;)

func (f *friends) Remove(name string) {} //código omitido pra ficar legal no slide ;)

func (f *friends) String() string {} //código omitido pra ficar legal no slide ;)

func NewPerson(name string) *person {} //código omitido pra ficar legal no slide ;)

func (p *person) AddFriend(name string) {
    p.friends.Add(name)
}

func (p *person) RemoveFriend(name string) {
    p.friends.Remove(name)
}

func (p *person) GetFriends() *friends {
    return p.friends
}

func (p *person) String() string {
    return fmt.Sprintf("%s [%v]", p.name, p.friends)
}
```

```go
type friends struct {
    data []string
}

type person struct {
    name    string
    friends *friends
}

func NewFriends() *friends {} //código omitido pra ficar legal no slide ;)

func (f *friends) Add(name string) {} //código omitido pra ficar legal no slide ;)

func (f *friends) Remove(name string) {} //código omitido pra ficar legal no slide ;)

func (f *friends) String() string {} //código omitido pra ficar legal no slide ;)

func NewPerson(name string) *person {} //código omitido pra ficar legal no slide ;)

func (p *person) AddFriend(name string) {
    p.friends.Add(name)
}

func (p *person) RemoveFriend(name string) {
    p.friends.Remove(name)
}

func (p *person) GetFriends() *friends {
    return p.friends
}

func (p *person) String() string {
    return fmt.Sprintf("%s [%v]", p.name, p.friends)
}
```

```go
type friends struct {
    data []string
}

type person struct {
    name    string
    friends *friends
}

func NewFriends() *friends {} //código omitido pra ficar legal no slide ;)

func (f *friends) Add(name string) {} //código omitido pra ficar legal no slide ;)

func (f *friends) Remove(name string) {} //código omitido pra ficar legal no slide ;)

func (f *friends) String() string {} //código omitido pra ficar legal no slide ;)

func NewPerson(name string) *person {} //código omitido pra ficar legal no slide ;)

func (p *person) AddFriend(name string) {
    p.friends.Add(name)
}

func (p *person) RemoveFriend(name string) {
    p.friends.Remove(name)
}

func (p *person) GetFriends() *friends {
    return p.friends
}

func (p *person) String() string {
    return fmt.Sprintf("%s [%v]", p.name, p.friends)
}
```

```go
type friends struct {
    data []string
}

type person struct {
    name    string
    friends *friends
}

func NewFriends() *friends {} //código omitido pra ficar legal no slide ;)

func (f *friends) Add(name string) {} //código omitido pra ficar legal no slide ;)

func (f *friends) Remove(name string) {} //código omitido pra ficar legal no slide ;)

func (f *friends) String() string {} //código omitido pra ficar legal no slide ;)

func NewPerson(name string) *person {} //código omitido pra ficar legal no slide ;)

func (p *person) AddFriend(name string) {
    p.friends.Add(name)
}

func (p *person) RemoveFriend(name string) {
    p.friends.Remove(name)
}

func (p *person) GetFriends() *friends {
    return p.friends
}

func (p *person) String() string {
    return fmt.Sprintf("%s [%v]", p.name, p.friends)
}
```

ONE DOT PER LINE

Essa regra cita que você não deve encadear funções e sim usar as que fazem parte do mesmo contexto.

```go
type board struct {
    Squares []*square
}

type square struct {
    current *piece
}

type piece struct {
    name string
}

func NewSquare(piece *piece) *square {}//código omitido pra ficar legal no slide

func NewPiece(name string) *piece {}//código omitido pra ficar legal no slide

func NewBoard() *board {}//código omitido pra ficar legal no slide

func (b *board) BoardRepresentation() string {
    var buffer = &bytes.Buffer{}
    for _, s := range b.Squares {
        buffer.WriteString(s.current.name)
    }
    return buffer.String()
}
```

https://gist.github.com/eminetto/af2ac89e79f945ff2ace8c71b3299a52

```go
type board struct {
    Squares []*square
}

type square struct {
    current *piece
}

type piece struct {
    name string
}

func NewSquare(piece *piece) *square {}//código omitido pra ficar legal no slide

func NewPiece(name string) *piece {}//código omitido pra ficar legal no slide

func NewBoard() *board {}//código omitido pra ficar legal no slide

func (b *board) BoardRepresentation() string {
    var buffer = &bytes.Buffer{}
    for _, s := range b.Squares {
        buffer.WriteString(s.current.name)
    }
    return buffer.String()
}
```

https://gist.github.com/eminetto/af2ac89e79f945ff2ace8c71b3299a52

```go
type board struct {
    Squares []*square
}

type square struct {
    current *piece
}

type piece struct {
    name string
}

func NewSquare(piece *piece) *square {}//código omitido pra ficar legal no slide

func NewPiece(name string) *piece {}//código omitido pra ficar legal no slide

func NewBoard() *board {}//código omitido pra ficar legal no slide

func (b *board) BoardRepresentation() string {
    var buffer = &bytes.Buffer{}
    for _, s := range b.Squares {
        buffer.WriteString(s.current.name)
    }
    return buffer.String()
}
```

https://gist.github.com/eminetto/af2ac89e79f945ff2ace8c71b3299a52

LEI DE DEMETER

LEI DE DEMETER

- Cada unidade deve ter conhecimento limitado sobre outras unidades: apenas unidades próximas se relacionam.

LEI DE DEMETER

- Cada unidade deve ter conhecimento limitado sobre outras unidades: apenas unidades próximas se relacionam.
- Cada unidade deve apenas conversar com seus amigos; Não fale com estranhos.

LEI DE DEMETER

- Cada unidade deve ter conhecimento limitado sobre outras unidades: apenas unidades próximas se relacionam.
- Cada unidade deve apenas conversar com seus amigos; Não fale com estranhos.
- Apenas fale com seus amigos imediatos.

```go
type board struct {
    Squares []*square
}

type square struct {
    current *piece
}
func (s *square) addTo(buffer *bytes.Buffer) {
    s.current.addTo(buffer)
}

type piece struct {
    name string
}
func (p *piece) addTo(buffer *bytes.Buffer) {
    buffer.WriteString(p.name)
}

func NewSquare(piece *piece) *square {}//código omitido pra ficar legal no slide

func NewPiece(name string) *piece {}//código omitido pra ficar legal no slide

func NewBoard() *board {}//código omitido pra ficar legal no slide

func (b *board) BoardRepresentation() string {
    var buffer = &bytes.Buffer{}
    for _, s := range b.Squares {
        s.addTo(buffer)
    }
    return buffer.String()
}
```

https://gist.github.com/eminetto/6a0ad46da38e2e7d068aec7fdc492b49

```go
type board struct {
    Squares []*square
}

type square struct {
    current *piece
}
func (s *square) addTo(buffer *bytes.Buffer) {
    s.current.addTo(buffer)
}

type piece struct {
    name string
}
func (p *piece) addTo(buffer *bytes.Buffer) {
    buffer.WriteString(p.name)
}

func NewSquare(piece *piece) *square {}//código omitido pra ficar legal no slide

func NewPiece(name string) *piece {}//código omitido pra ficar legal no slide

func NewBoard() *board {}//código omitido pra ficar legal no slide

func (b *board) BoardRepresentation() string {
    var buffer = &bytes.Buffer{}
    for _, s := range b.Squares {
        s.addTo(buffer)
    }
    return buffer.String()
}
```

https://gist.github.com/eminetto/6a0ad46da38e2e7d068aec7fdc492b49

```go
type board struct {
    Squares []*square
}

type square struct {
    current *piece
}
func (s *square) addTo(buffer *bytes.Buffer) {
    s.current.addTo(buffer)
}

type piece struct {
    name string
}
func (p *piece) addTo(buffer *bytes.Buffer) {
    buffer.WriteString(p.name)
}

func NewSquare(piece *piece) *square {}//código omitido pra ficar legal no slide

func NewPiece(name string) *piece {}//código omitido pra ficar legal no slide

func NewBoard() *board {}//código omitido pra ficar legal no slide

func (b *board) BoardRepresentation() string {
    var buffer = &bytes.Buffer{}
    for _, s := range b.Squares {
        s.addTo(buffer)
    }
    return buffer.String()
}
```

https://gist.github.com/eminetto/6a0ad46da38e2e7d068aec7fdc492b49

```go
type board struct {
    Squares []*square
}

type square struct {
    current *piece
}
func (s *square) addTo(buffer *bytes.Buffer) {
    s.current.addTo(buffer)
}

type piece struct {
    name string
}
func (p *piece) addTo(buffer *bytes.Buffer) {
    buffer.WriteString(p.name)
}

func NewSquare(piece *piece) *square {}//código omitido pra ficar legal no slide

func NewPiece(name string) *piece {}//código omitido pra ficar legal no slide

func NewBoard() *board {}//código omitido pra ficar legal no slide

func (b *board) BoardRepresentation() string {
    var buffer = &bytes.Buffer{}
    for _, s := range b.Squares {
        s.addTo(buffer)
    }
    return buffer.String()
}
```

https://gist.github.com/eminetto/6a0ad46da38e2e7d068aec7fdc492b49

```go
type board struct {
    Squares []*square
}

type square struct {
    current *piece
}
func (s *square) addTo(buffer *bytes.Buffer) {
    s.current.addTo(buffer)
}

type piece struct {
    name string
}
func (p *piece) addTo(buffer *bytes.Buffer) {
    buffer.WriteString(p.name)
}

func NewSquare(piece *piece) *square {}//código omitido pra ficar legal no slide

func NewPiece(name string) *piece {}//código omitido pra ficar legal no slide

func NewBoard() *board {}//código omitido pra ficar legal no slide

func (b *board) BoardRepresentation() string {
    var buffer = &bytes.Buffer{}
    for _, s := range b.Squares {
        s.addTo(buffer)
    }
    return buffer.String()
}
```

https://gist.github.com/eminetto/6a0ad46da38e2e7d068aec7fdc492b49

~~Don’t abbreviate~~

Esta regra é uma das que não se aplica diretamente a Go. A comunidade tem suas próprias regras para a criação de nomes de variáveis, inclusive razões por usarmos nomes menores. Recomendo a leitura deste capítulo do ótimo Practical Go: Real world advice for writing maintainable Go programs

KEEP ALL CLASSES LESS THAN 50 LINES

Apesar de não existir o conceito de classes em Go, a ideia desta regra é que as entidades sejam pequenas. Podemos adaptar a ideia para criarmos structs e interfaces pequenas e que podem ser usadas, via composição, para formar componentes maiores

```go
type Repository interface {
    Find(id entity.ID) (*entity.User, error)
    FindByEmail(email string) (*entity.User, error)
    FindByChangePasswordHash(hash string) (*entity.User, error)
    FindByValidationHash(hash string) (*entity.User, error)
    FindByChallengeSubmissionHash(hash string) (*entity.User, error)
    FindByNickname(nickname string) (*entity.User, error)
    FindAll() ([]*entity.User, error)
    Update(user *entity.User) error
    Store(user *entity.User) (entity.ID, error)
    Remove(id entity.ID) error
}
```

```go
type Reader interface {
    Find(id entity.ID) (*entity.User, error)
    FindByEmail(email string) (*entity.User, error)
    FindByChangePasswordHash(hash string) (*entity.User, error)
    FindByValidationHash(hash string) (*entity.User, error)
    FindByChallengeSubmissionHash(hash string) (*entity.User, error)
    FindByNickname(nickname string) (*entity.User, error)
    FindAll() ([]*entity.User, error)
}

type Writer interface {
    Update(user *entity.User) error
    Store(user *entity.User) (entity.ID, error)
    Remove(id entity.ID) error
}

type Repository interface {
    Reader
    Writer
}
```

~~No classes with more than two instance variables~~

Esta regra não parece fazer sentido em Go

~~No Getters/Setters~~

Não é um costume da comunidade usar este recurso, como pode ser visto neste tópico do Effective Go

REFERÊNCIAS

Os códigos completos e demais referências podem ser encontrados neste post:

Object Calisthenics em Golang

https://eltonminetto.dev

https://codenation.dev

@eminetto