

A series of overlapping geometric shapes, primarily triangles and quadrilaterals, in shades of teal and purple, located in the top-left corner of the slide.

AceLeraDev Java

Módulo 8 - Object Calisthenics

A series of overlapping geometric shapes, primarily triangles and quadrilaterals, in shades of teal and purple, located in the bottom-right corner of the slide.

The slide features decorative geometric lines in purple and teal. In the top-left corner, a purple line forms a large 'L' shape, and a teal line forms a smaller 'L' shape. In the bottom-right corner, a teal line forms a large 'L' shape, and a purple line forms a smaller 'L' shape. These lines create a modern, abstract background.

Tópicos da Aula

- SOLID;
- CLEAN CODE;
- OBJECT CALISTHENICS;
- TDD.

Object Calisthenics

- 9 regras no livro "The ThoughtWorks Anthology" escrito por Jeff Bay.
- Object vem da Orientação a Objeto.
- Calisthenics vem de exercícios.
- Traduzinho: São boas práticas na orientação a objetos.
- As regras possuem o foco em manutenabilidade, legibilidade, testabilidade e compreensividade.

Regra 01: Apenas um nível de indentação por método

- Quanto mais níveis de indentação mais difícil fica a leitura.
- Na maioria das vezes é muito difícil você ler o código sem precisar compilá-lo na sua mente.
- Quantos códigos hadouken vocês já fizeram?

```

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();
}

```



```
class Board {  
    public String board() {  
        StringBuilder buf = new StringBuilder();  
  
        // 0  
        for (int i = 0; i < 10; i++) {  
            // 1  
            for (int j = 0; j < 10; j++) {  
                // 2  
                buf.append(data[i][j]);  
            }  
            buf.append("\n");  
        }  
  
        return buf.toString();  
    }  
}
```

Extract Method Pattern

- "Refactoring", Martin Fowler.
- Não diminui as linhas, mas aumenta a legibilidade.

```
class Board {  
    public String board() {  
        StringBuilder buf = new StringBuilder();  
  
        collectRows(buf);  
  
        return buf.toString();  
    }  
  
    private void collectRows(StringBuilder buf) {  
        for (int i = 0; i < 10; i++) {  
            collectRow(buf, i);  
        }  
    }  
  
    private void collectRow(StringBuilder buf, int row) {  
        for (int i = 0; i < 10; i++) {  
            buf.append(data[row][i]);  
        }  
  
        buf.append("\n");  
    }  
}
```

Regra 02: Nao utilize ELSE

- Ou utiliza a Programacao Defensiva

```
public void login(String username, String password) {  
    if (userRepository.isValid(username, password)) {  
        redirect("homepage");  
    } else {  
        addFlash("error", "Bad credentials");  
  
        redirect("login");  
    }  
}
```


De preferencia para retornos anticipados

```
public void login(String username, String password) {  
    if (userRepository.isValid(username, password)) {  
        return redirect("homepage");  
    }  
  
    addFlash("error", "Bad credentials");  
  
    return redirect("login");  
}
```

Ou utilize a Programação Defensiva

```
public void login(String username, String password) {  
    String redirectRoute = "homepage";  
  
    if (!userRepository.isValid(username, password)) {  
        addFlash("error", "Bad credentials");  
        redirectRoute = "login";  
    }  
  
    redirect(redirectRoute);  
}
```

Regra 03: Encapsule seus primitivos e Strings

- Se existem regras relacionadas a esses atributos você deve criar uma classe apenas para eles, encapsulando-os.

Regra 04: Classes específicas para Collections

- Qualquer classe que contenha uma Collection não deve conter outros atributos.
- Ela deve ser focada estritamente para a Collection e lidar com as regras de negócio dela.
- Com isso sua collections é encapsulada e existe uma classe para lidar apenas com ela.

Regra 05: Um ponto por linha

- Essa regra não se aplica a Builders.

```
class Location {  
    public Piece current;  
}
```

```
class Piece {  
    public String representation;  
}
```

```
class Board {  
    public String boardRepresentation() {  
        StringBuilder buf = new StringBuilder();  
  
        for (Location loc : squares()) {  
            buf.append(loc.current.representation.substring(0, 1));  
        }  
  
        return buf.toString();  
    }  
}
```

Lei de Demeter: Se comunique com seus relacionados

```
class Location {  
    private Piece current;  
  
    public void addTo(StringBuilder buf) {  
        current.addTo(buf);  
    }  
}
```

```
class Piece {  
    private String representation;  
  
    public String character() {  
        return representation.substring(0, 1);  
    }  
  
    public void addTo(StringBuilder buf) {  
        buf.append(character());  
    }  
}
```

```
class Board {  
    public String boardRepresentation() {  
        StringBuilder buf = new StringBuilder();  
  
        for (Location location : squares()) {  
            location.addTo(buf);  
        }  
  
        return buf.toString();  
    }  
}
```


Regra 06: Não abrevie

- A pergunta certa é: Porque você quer abreviar?
- O método está sendo utilizado em muitos lugares? Talvez tenha código duplicado
- Se seu método tem um nome muito grande, talvez ele esteja violando o Princípio de Responsabilidade Única.
- Não existe motivos para se abreviar.

Regra 07: Mantenha todas as classes pequenas

- Classes com muitas linhas significa que ela possui muitas responsabilidades.
- Quanto maior for o arquivo, mais difícil será a leitura.

Regra 08: Sem classes com mais de dois atributos de instância

- Alta Coesao;
- Baixo acoplamento;
- Melhor encapsulamento.

Regra 09: Sem Getter/Setter ou Properties

- Siga a regra: "Diga, não peça."

```
// Game
private int score;


public void setScore(int score) {
    this.score = score;
}

public int getScore() {
    return score;
}

// Usage
game.setScore(game.getScore() + ENEMY_DESTROYED_SCORE);
```

```
// Game  
public void addScore(int delta) {  
    score += delta;  
}
```

```
// Usage  
game.addScore(ENEMY_DESTROYED_SCORE);
```

The image features a light blue background with decorative geometric lines in teal and purple. In the top-left corner, there are overlapping lines forming a partial hexagonal shape. In the bottom-right corner, there are similar overlapping lines, also forming a partial hexagonal shape. The text is centered in the middle of the image.

**Experimente as regras e depois
diga sua opinião!**

Feedback da aula

