DESIGN PATTERNS

por Luiz Fernando

O QUE É UM DESIGN PATTERN?

Um Design Pattern, ou padrão de projeto, é uma solução reutilizável para um problema comum que ocorre durante o processo de design e desenvolvimento de software. Ele descreve uma abordagem testada e comprovada para resolver um problema específico, fornecendo diretrizes e estruturas que podem ser aplicadas em diferentes contextos.

Os Design Patterns são criados com base em boas práticas e experiências anteriores, e têm como objetivo ajudar os desenvolvedores a resolver problemas de design de forma eficiente, promovendo a reutilização de soluções já estabelecidas. Eles fornecem uma linguagem comum para comunicar e compartilhar conhecimento sobre design de software entre os membros de uma equipe de desenvolvimento.

QUAIS SÃO OS TIPO DE DESIGN PATTERN?

Um Design Pattern, ou padrão de projeto, é uma solução reutilizável para um problema comum que ocorre durante o processo de design e desenvolvimento de software. Ele descreve uma abordagem testada e comprovada para resolver um problema específico, fornecendo diretrizes e estruturas que podem ser aplicadas em diferentes contextos.

QUAIS SÃO OS TIPO DE DESIGN PATTERN?

1. Padrões de Criação (Creational Patterns):
2. Padrões Estruturais (Structural Patterns):
3. Padrões Comportamentais (Behavioral Patterns):
4. Padrões de Arquitetura (Architectural Patterns):
5. Padrões de Concorrência (Concurrency Patterns):
6. Padrões de Segurança (Security Patterns):
7. Padrões de Integração (Integration Patterns):
7. Padrões de Segurança (Security Patterns):

ARCHITECTURAL PATTERNS

Padrões de Arquitetura (Architectural Patterns) são padrões que fornecem diretrizes de alto nível para a estruturação e organização de sistemas de software. Eles abordam a arquitetura geral do sistema, definindo a forma como os componentes se relacionam e interagem entre si para atender aos requisitos funcionais e não funcionais do sistema. Esses padrões ajudam a criar sistemas escaláveis, flexíveis e de fácil manutenção.

MVC

```
// Model
class Task {
  constructor(description, completed = false) {
    this.description = description;
    this.completed = completed;
  }
}

// View
class TaskView {
  render(task) {
    console.log(`[${task.completed ? 'x' : ' '}] ${task.description}`);
  }
}

// Controller
class TaskController {
  constructor() {
    this.taskView = new TaskView();
  }

  createTask(description) {
    const task = new Task(description);
    this.taskView.render(task);
  }
}

// Utilização do MVC
const taskController = new TaskController();
taskController.createTask("Fazer compras");
```

MVP

```
// Model
class Task {
  constructor(description, completed = false) {
    this.description = description;
    this.completed = completed;
  }
}

// View
class TaskView {
  render(task) {
    console.log(`[${task.completed ? 'x' : ' '}] ${task.description}`);
  }
}

// Presenter
class TaskPresenter {
  constructor() {
    this.taskView = new TaskView();
  }

  createTask(description) {
    const task = new Task(description);
    this.taskView.render(task);
  }
}

// Utilização do MVP
const taskPresenter = new TaskPresenter();
taskPresenter.createTask("Fazer compras");
```

MVVM

```
// Model
class Task {
  constructor(description, completed = false) {
    this.description = description;
    this.completed = completed;
  }
}

// ViewModel
class TaskViewModel {
  constructor() {
    this.tasks = [];
  }

  createTask(description) {
    const task = new Task(description);
    this.tasks.push(task);
  }
}

// Utilização do MVVM
const taskViewModel = new TaskViewModel();
taskViewModel.createTask("Fazer compras");
console.log(taskViewModel.tasks);
```