

Padrões de Projeto

Introdução à disciplina

Sobre a disciplina

Conteúdo:

- Estudo de **Padrões de projetos** de softwares.
- Identificação dos tipos de padrões e cenários de uso.
- Entender as vantagens de cada abordagem.

Avaliação:

- Entrega de atividades e apresentação.
- Desenvolvimento de um software usando padrões de projetos (definir).

Tópicos da aula

1. O que são padrões de projetos.
2. Tipos de padrões de projetos.
3. Importancia dos padrões de projeto no desenvolvimento de software.
4. Conceitos da orientação a objetos.

Inspiração

Cada padrão descreve um problema que ocorre repetidamente em certos contextos fornecendo uma solução de forma que seja possível utilizar **repetidas** vezes esta solução.

Desafios do desenvolvimento de software

- Código evolui: **mudanças** ocorrerão mais cedo ou mais tarde.
- Exemplos:
 - Formas de pagamento
 - Tipo de frete para envio de mercadorias
- Manutenção de código pode se tornar um pesadelo
- Dependendo do modo como o software foi projetado, pode sair mais "**barato**" desenvolver um novo sistema do alterá-lo. *Entretanto, nenhum dos casos é desejável.*

Estrutura da solução

- Padrões surgiram da observação de que certos **problemas** se repetem durante o desenvolvimento de software (mesmo em sistemas e tecnologias diferentes).
- Após uma análise desses problemas, foi escolhida a **melhor** solução.
- Cada solução é identificadas por um **nome**.
- Discutimos o projeto de sistema do ponto de vista dos padrões de projetos (Fábricas, Fachadas, Estratégias, ...) ao invés de estruturas de dados (pilha, fila, árvore, ...)

Literatura

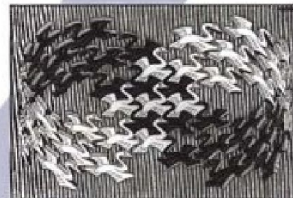
Elements of Reusable Object-Oriented Software (1995) é a referência clássica, onde é definido o conceito de padrões de projetos juntamente com um catálogo de 23 padrões.

Apelidados de **Gang of Four (GoF)**

Design Patterns

Elements of Reusable
Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



Foreword by Grady Booch



ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Formato de um padrão de projeto

Os padrão de descritos no catálogo possuem:

- Nome
- Problema (explicação e contexto)
- Solução
- Consequências (resultados esperados)

Objetivos

Um padrão de projeto busca **identificar**:

- As **classes** e **objetos** participantes
- Os papéis e colaboração entre os participantes
- As reponsabilidades de cada participante
- O problema a ser solucionado.
- Quais cenários pode ser aplicável
- Consequências e *trade-offs* do seu uso

Tipos de Padrões de Projetos

- **Criacional:** relacionados à criação de objetos
- **Estrutural:** trata da associação entre classes e objetos.
- **Comportamental:** sobre as interações e divisões de responsabilidades entre as classes ou objetos.

Padrão Criacional

Oferecem mecanismos para criação de objetos, permitindo maior independência entre as classes.

- Encapsulam a lógica de criação de objetos.
- Facilitam o controle sobre como e quando os objetos são criados
 - Evitar a criação excessiva de instâncias
 - Promover organização do código.
 - Diminuir o **acoplamento** entre classe cliente e classe instanciada.

Padrão Estrutural

Foco na composição e relação entre classes e objetos, para melhorar a organização do código.

- Composição de objetos para formar uma nova funcionalidade.
- Redução da complexidade do sistema ao **encapsular** vários componentes em uma interface mais simples.
- Promover a reutilização de componentes existentes

Padrão **Comportamental**

Definem regras de interações entre objetos e suas responsabilidades na colaboração para realizar uma tarefa.

- Focam em como os objetos se relacionam
- Distribuição da responsabilidades entre objetos.
- Permitem que os sistemas sejam adaptáveis a modificação de comportamentos em tempo de execução.
- Encapsulam algoritmos e comportamentos em objetos que possam ser passados, armazenados ou manipulados.

Catálogo de padrões (GoF)

CRIACIONAL

- **Factory**
- Abstract Factory
- **Builder**
- Prototype
- **Singleton**

ESTRUTURAL

- Adapter
- Bridge
- Composite
- **Decorator**
- **Facade**
- Flyweight
- **Proxy**

COMPORTAMENTAL

- Interpreter
- **Template Method**
- Chain of Responsibility
- Iterator
- Mediator
- Memento
- **Observer**
- State
- **Strategy**
- Visitor

Princípios do projeto de software

- Identifique os aspectos da aplicação que variam e separe-os daqueles que se mantêm constante.
- O código que varia deve ser encapsulado para não afetar outras partes do código em caso de mudanças.
- Ao encapsular o código aberto a mudança é possível realizar alterações futuras sem quebrar a parte do código dependente que não varia (fechada a mudança).

Princípios do projeto de software

- Padrões facilitam o design das partes do sistema que evoluem ao separá-las, tornando-as independentes das outras partes.
- Programe para uma interface e não para uma implementação (concreta).
- Isso significa programar para um supertipo.
- Explorar o polimorfismo através de um supertipo.
 - Exemplo: programar para o supertipo “Pagamento” e não à classe “PagamentoPix”.
 - Evita-se assim criar dependência com partes do código abertas a mudança.

Paradigma OO

Padrões de projetos foram definidos para o paradigma **orientado a objetos**.

Ao estudar as soluções que cada padrão propõe é importante ter claro os conceitos da POO:

1. Classe abstrata / interface
2. Herança
3. Polimorfismo

Programação OO no Python

Classe abstrata

```
from abc import ABC, abstractmethod

class Pagamento(ABC):
    def __init__(self, valor: float): # construtor
        # declarado e inicializado no construtor
        self.valor = valor # atributo

    def status(self): # método concreto
        print(f"valor do pagamento: {self.valor:.2f}")

    @abstractmethod
    def realizar(self): # método abstrato
        pass
```

Vocabulário:

- classe **abstrata** (ABC)
- **construtor** (__init__)
- **self**
- **atributo**
- **métodos**
- @abstractmethod

Classe concreta

Vocabulário:

- objeto (p)
- classe concreta
- herança

```
class PagamentoPix(Pagamento):  
    def realizar(self):  
        """ Implementa método abstrato de Pagamento """  
        self.valor = self.valor + 10  
        print("pagamento efetivado")
```

```
# cria uma instancia de pagamento:  
p = PagamentoPix(100)  
p.realizar()  
p.status()
```

Tipo Pessoa

Definir as classes **Pessoa**, **PessoaFisica** e **PessoaJuridica** no Python, identificando quais são abstratas e concretas. **Atributos**: nome, renda anual, cpf e cnpj.

- Gerar um instância para cada tipo (PF e PJ)
- Implementar o método **calcularIR()** de acordo com o tipo de Pessoa:
 - PessoaFisica - paga 25% de IR sobre renda anual.
 - PessoaJuridica - paga 18%
- Calcular o valor de IR à pagar com base na renda informada.

Atividade 1

Implementar os tipos **PagamentoBoleto** e **PagamentoCartao** e o método **realizar()**. Considerar os seguintes valores para cada forma de pgto:

- boleto: valor + 12%
- cartão: valor + 15%

Criar 3 instâncias de pagamento com mesmo valor e exibir o valor final para as diferentes formas de pagamento escolhidas.

A escolha da forma de pagamento é decidida em tempo de execução