Backend
Python com Django

Prof. Bruno Iran Ferreira Maciel

Aug 2024

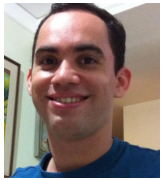
Estrutura de Tópicos das Aulas

Resumo do Conteúdo Programático

N	Aulas	Mês	Data	Conteúdo Previsto
1	1,5	Agosto	02/08/2024	Apresentação e boas-vindas
2	3	Agosto	02/08/2024	Lógica de Prog. e Padrões de Des. de Software
3	4,5	Agosto	03/08/2024	Lógica de Programação
4	6	Agosto	03/08/2024	Padrões de Desenvolvimento de Software
5	7,5	Agosto	09/08/2024	Lógica de Programação
6	9	Agosto	09/08/2024	Padrões de Desenvolvimento de Software
7	10,5	Agosto	10/08/2024	POO
8	12	Agosto	10/08/2024	Padrões de Desenvolvimento de Software
9	13,5	Agosto	16/08/2024	POO
10	15	Agosto	16/08/2024	Git
11	16,5	Agosto	17/08/2024	POO
12	18	Agosto	17/08/2024	Padrões de Desenvolvimento de Software
13	19,5	Agosto	23/08/2024	POO
14	21	Agosto	23/08/2024	Padrões de Desenvolvimento de Software
15	22,5	Agosto	24/08/2024	POO
16	24	Agosto	24/08/2024	Padrões de Desenvolvimento de Software
17	25,5	Agosto	30/08/2024	POO
18	27	Agosto	30/08/2024	Padrões de Desenvolvimento de Software
19	28,5	Agosto	31/08/2024	POO
20	30	Agosto	31/08/2024	Soft Skills
21	31,5	Setembro	06/09/2024	POO
22	33	Setembro	06/09/2024	Padrões de Desenvolvimento de Software
23	0	Setembro	07/09/2024	Feriado Nacional - Independência do Brasil
x		Setembro	13/09/2024	POO
x		Setembro	13/09/2024	Padrões de Desenvolvimento de Software
x		Setembro	14/09/2024	POO
x		Setembro	14/09/2024	Padrões de Desenvolvimento de Software
x		Setembro	20/09/2024	POO
x		Setembro	20/09/2024	Padrões de Desenvolvimento de Software
x		Setembro	21/09/2024	POO

N	Aulas	Mês	Data	Conteúdo Previsto
x		Outubro	04/10/2024	Django
x		Outubro	05/10/2024	Soft Skills
x		Outubro	05/10/2024	POO
x		Outubro	11/10/2024	Django
x		Outubro	11/10/2024	Web Services
x		Outubro	12/10/2024	Feriado Nacional - Nossa Senhora Aparecida
x		Outubro	18/10/2024	Django
x		Outubro	18/10/2024	Web Services
x		Outubro	19/10/2024	Django
x		Outubro	19/10/2024	Web Services
x		Outubro	25/10/2024	Django
x		Outubro	25/10/2024	Web Services
x		Outubro	26/10/2024	Django
x		Outubro	26/10/2024	Web Services
x		Novembro	01/11/2024	Django
x		Novembro	01/11/2024	Web Services
x		Novembro	02/11/2024	Feriado Nacional - Finados
x		Novembro	08/11/2024	Django
x		Novembro	08/11/2024	Web Services
x		Novembro	09/11/2024	Soft Skills
x		Novembro	09/11/2024	Django
x		Novembro	15/11/2024	Feriado Nacional - Proclamação da República
x		Novembro	16/11/2024	Imprensado - Não teremos aulas
x		Novembro	22/11/2024	Django
x		Novembro	22/11/2024	Web Services
x		Novembro	23/11/2024	Django
x		Novembro	23/11/2024	Web Services
x		Novembro	29/11/2024	Django
x		Novembro	29/11/2024	Web Services
x		Novembro	30/11/2024	Django
x		Novembro	30/11/2024	Web Services

Quem sou eu?



Prof. Dr. Bruno Iran Ferreira Maciel, 41

Atuo como professor; desenvolvedor; e consultor de TI.

- Doutor em Ciência da Computação, 2015-2020
- Mestre em Ciência da Computação, 2012-2014
- Especialista em Engenharia e Reúso de Software, 2011-2012
- Graduado em Sistemas de Informação, 2016-2016
- Graduado em Ciência da Computação, 2007-2011
- Técnico em Análise e Desenvolvimento de Software, 2007-2007
- CV completo http://bit.ly/brunomaciel-lattes

Apresentação pessoal, integração com a turma, introdução de conceitos básicos de desenvolvimento de softwae e despertar curiosidade dos alunos sobre o tema.

Compromisso semanal

Encontros: sextas e sábados

- Período: 02/08/2024 à 30/11/2024

Sextas

- Início: 18h
- Térmico: 21h - Sem intervalo
- Térmico: 21h10 - Com intervalo de 10 minutos (a confirmar)

Sábados

- Início: 9h
- Térmico: 12h - Sem intervalo
- Térmico: 12h10 - Com intervalo de 10 minutos (a confirmar)

Reposições de aulas aos Sábados no período da tarde

- Início: 14h
- Térmico: 17h - Sem intervalo
- Térmico: 17h10 - Com intervalo de 10 minutos (a confirmar)

Metodologia das Aulas

Aulas:

- 18h-19h30
- 19h30-21h

- Resolução de dúvidas gerais e tolerância: 18h até 18h15 (15 minutos)
- Revisão da aula passada: 18h15 até 18h30 (15 minutos)
- Adição de novo conteúdo: 18h30 até 21h (2h30)

EMENTA

O curso tem como objetivo desenvolver as habilidades necessárias em programação, noções de padrões de desenvolvimento de software, arquitetura cliente-servidor, noções de banco de dados e frameworks back-end para que você seja capaz de projetar soluções web que sejam seguras, robustas e escaláveis, com base em tecnologias modernas e nas melhores práticas de desenvolvimento de software.

OBJETIVO

Compreender o funcionamento de características e arranjos básicos de desenvolvimento de software com foco em backend.
Permitir análise crítica das questões relativas aos conceitos estudados ao longo das aulas, bem como a identificação de áreas de pesquisa voltadas para o aperfeiçoamento das técnicas e desenvolvimento de novas aplicações.

Competências Específicas

- Lógica de Progamação.
- Padrões de projetos.
- Soft Skills.
- Padrões de projetos.
- POO.
- Git.
- Web Services.
- Django.

Github das aulas

https://github.com/brunom4ciel/material-python

Bibliografia básica

- **Ghetan, Giridhar. Aprendendo Padrões de Projeto em Python, 1ª Edição. Novatec. 2016.** Ghetan [2016]

Acrônimo 1

CVCS *Centralized Version Control Systems (em português, Sistema Centralizado de Controle de Versão)* 105
DVCS *Distributed Version Control Systems (em português, Sistema Distribuído de Controle de Versão)* 105
GUI *Graphical User Interface (em português, Interface Gráfica do Utilizador)* . . . 145
LVCS *Local Version Control Systems (em português, Sistema Local de Controle de Versão)* 105
MVC *Model-View-Controller (em português, Modelo-Visão-Controle ou também como Controlador de visualização de modelo)* 139, 143–145
POO *Programação Orientada a Objetos* . . . 81, 82, 86, 98, 154, 164, 178, 180, 186
VCS *Version Control Systems (em português, Sistema de Controle de Versão)* 104, 106, 107

Apresentação e boas-vindas

Padrões de desenvolvimento de software

...continuando Padrões de desenvolvimento de software.

- Padrões de Criação.
  - Singleton ✔
  - Abstract Factory ✔
  - Factory Method ✗
  - Builder ✗
  - Prototype ✗

Padrões de criação

Factory Method ✔

- Padrão que define um método para criar um objeto em uma classe-mãe, mas permite que subclasses modifiquem o tipo de objeto que será criado.
- Ele é ideal quando você tem uma superclasse que faz algo, mas quer que subclasses específicas decidam como criar esses objetos.

Padrões de criação

Exemplo

- Imagine que você tenha uma classe Documento que é responsável por abrir arquivos.
- Dependendo do tipo de documento (PDF, Word), você quer abrir o documento de maneiras diferentes.
- O **Factory Method** permite que cada tipo de documento implemente seu próprio método de abertura.

Padrão de projeto Factory Method

```python
import abc

class DocumentoAbstrata(metaclass=abc.ABCMeta):
    @abc.abstractmethod
    def abrir(self):
        pass
```

Código 1: Código da Classe Documento - codigo_016_documento_abstrata.py Abstrata

```python
from src import DocumentoAbstrata

class DocumentoPdfConcreta(DocumentoAbstrata):
    def abrir(self):
        return "Abrindo um documento PDF"
```

Código 2: Código da Classe Documento Pdf - codigo_016_documento_pdf_concreta.py Concreta

```python
from src import DocumentoAbstrata

class DocumentoMsWordConcreta(DocumentoAbstrata):
    def abrir(self):
        return "Abrindo um documento Microsoft Word"
```

Código 3: Código da Classe Documento Ms Word - codigo_016_documento_msword_concreta.py Concreta

Padrão de projeto Factory Method

```python
import sys
import os
# Adiciona o caminho do diretório 'atual' ao sys.path.
# Isso resolver problemas com imports de módulos que estão no mesmo diretório.
sys.path.append(os.path.join(os.path.dirname(__file__), '..', ''))
from src import DocumentoPdfConcreta
from src import DocumentoMsWordConcreta

# Cliente usando a Factory Method
def codigo_cliente_factory_method(tipo_documento: str):

    if(tipo_documento == "word" or tipo_documento == "pdf"):

        if tipo_documento == "word":
            documento = DocumentoMsWordConcreta()

        elif tipo_documento == "pdf":
            documento = DocumentoPdfConcreta()

        print(documento.abrir())
    else:

        print("Tipo de documento desconhecido")
```

Código 4: Código da função codigo_cliente_factory_method

*obs: Linhas de 1 a 6 são para ajustar a leitura dos arquivos para o Phyton, que estão no mesmo diretório.

Padrão de projeto Factory Method

```python
import pytest
from unittest.mock import patch
from src import codigo_cliente_factory_method

@pytest.mark.parametrize("valor_esperado", [(["Abrindo um documento PDF"])])
def teste_codigo_cliente_para_abrir_documento_pdf_com_sucesso(valor_esperado):
  with patch('builtins.print') as mock_print:
    codigo_cliente_factory_method("pdf")
    mock_print.assert_any_call(valor_esperado[0])
    assert mock_print.call_count == len(valor_esperado)

@pytest.mark.parametrize("valor_esperado", [(["Abrindo um documento Microsoft Word"])])
def teste_codigo_cliente_para_abrir_documento_msword_com_sucesso(valor_esperado):
  with patch('builtins.print') as mock_print:
    codigo_cliente_factory_method("word")
    mock_print.assert_any_call(valor_esperado[0])
    assert mock_print.call_count == len(valor_esperado)

@pytest.mark.parametrize("valor_esperado", [(["Tipo de documento desconhecido"])])
def teste_codigo_cliente_para_abrir_documento_outro_com_sucesso(valor_esperado):
  with patch('builtins.print') as mock_print:
    codigo_cliente_factory_method("outro")
    mock_print.assert_any_call(valor_esperado[0])
    assert mock_print.call_count == len(valor_esperado)
```

Código 5: Testes do Código do cliente para utilizar a Factory Method - test_codigo_016_cliente_factory_method.py

Padrão de projeto Factory Method

```python
from src import DocumentoMsWordConcreta, DocumentoPdfConcreta

class DocumentoFactoryMethodConcreta:

    def __init__(self, tipo_documento: str):
        if tipo_documento == "word":
            self.document = DocumentoMsWordConcreta()
        elif tipo_documento == "pdf":
            self.document = DocumentoPdfConcreta()

    def abrir(self):
        return self.document.abrir()
```

Código 6: Código da Classe Documento Factory Method Concreta - codigo_016_documento_factory_method_concreta.py

*obs: quaias as vantagens e desvantagens desta "segunda" implementação? que utilizou a classe **DocumentoFactoryMethodConcreta**.

Padrão de projeto Factory Method

```python
from src import DocumentoFactoryMethodConcreta

# Cliente usando a Factory Method
def codigo_cliente_factory_method2(tipo_documento: str):

    if(tipo_documento == "word" or tipo_documento == "pdf"):

        documento = DocumentoFactoryMethodConcreta(tipo_documento)

        print(documento.abrir())

    else:

        print("Tipo de documento desconhecido")
```

Código 7: Código do cliente para utilizar a Factory Method - codigo_016_codigo_cliente_factory_method2.py

Padrão de projeto Factory Method

```python
import pytest
from unittest.mock import patch
from src import codigo_cliente_factory_method2

@pytest.mark.parametrize("valor_esperado", [(["Abrindo um documento PDF"])])
def teste_codigo_cliente_para_abrir_documento_pdf_com_sucesso(valor_esperado):
  with patch('builtins.print') as mock_print:
    codigo_cliente_factory_method2("pdf")
    mock_print.assert_any_call(valor_esperado[0])
    assert mock_print.call_count == len(valor_esperado)

@pytest.mark.parametrize("valor_esperado", [(["Abrindo um documento Microsoft Word"])])
def teste_codigo_cliente_para_abrir_documento_msword_com_sucesso(valor_esperado):
  with patch('builtins.print') as mock_print:
    codigo_cliente_factory_method2("word")
    mock_print.assert_any_call(valor_esperado[0])
    assert mock_print.call_count == len(valor_esperado)

@pytest.mark.parametrize("valor_esperado", [(["Tipo de documento desconhecido"])])
def teste_codigo_cliente_para_abrir_documento_outro_com_sucesso(valor_esperado):
  with patch('builtins.print') as mock_print:
    codigo_cliente_factory_method2("outro")
    mock_print.assert_any_call(valor_esperado[0])
    assert mock_print.call_count == len(valor_esperado)
```

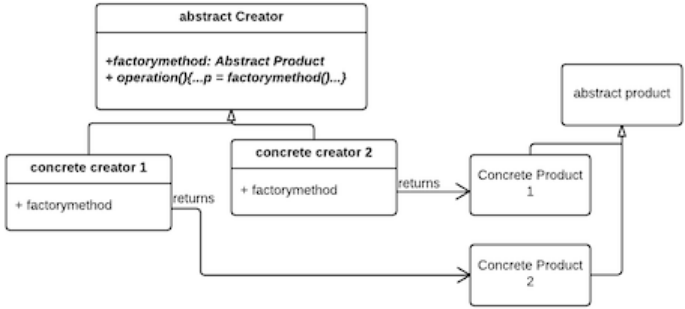
Código 8: Testes do Código do cliente para utilizar a Factory Method

Padrões de criação

Diferenças Fundamentais entre Abstract Factory e Factory Method?

- **Escopo de Aplicação**
  - Abstract Factory: Focado na criação de famílias de objetos relacionados.
  - Factory Method: Focado na criação de um único objeto, delegando a subclasses a decisão de qual tipo de objeto criar.
- **Complexidade**
  - Abstract Factory: Envolve várias classes e métodos para criar diferentes objetos relacionados.
  - Factory Method: Mais simples, envolvendo um único método de criação.
- **Flexibilidade**
  - Abstract Factory: Útil para criar produtos consistentes entre si.
  - Factory Method: Útil quando a criação de objetos pode variar em subclasses.

Abstract Factory



Lógica de Prog. e Padrões de Des. de Software

Exemplos de criação de classe

...continuando a partir da aula passada sobre Programação Orientada a Objetos (POO).

Exercícios

Escreva o algoritmo e programa. Use Herança de POO para resolver.

- *Exercício 018*: Ler 3 números inteiros que correspondem a (1) quantidade atual em estoque, (2) quantidade máxima em estoque e (3) quantidade mínima em estoque de um produto. Calcular e escrever a quantidade média ((quantidade média = quantidade máxima + quantidade mínima)/2). Se a quantidade em estoque for maior ou igual a quantidade média escrever a mensagem 'Não efetuar compra', senão escrever a mensagem 'Efetuar compra'.

Interface de classe

Uma interface define um conjunto de métodos que uma classe deve implementar. Ela atua como um contrato.

- Em Python, não há uma palavra-chave específica para definir **interfaces**, mas podemos usar **classes abstratas** para criar **interfaces**.

Classes Abstratas

Uma classe abstrata é uma classe que não pode ser instanciada diretamente. Ela serve como um modelo para outras classes.

- Podemos criar uma classe abstrata com o **módulo abc** (Abstract Base Classes) em Python.

https://docs.python.org/pt-br/3/library/abc.html

Interface com Classes Abstratas - Caso com abc

```python
import abc
class ProdutoAbstrato(metaclass=abc.ABCMeta):
    @abc.abstractmethod
    def __init__(self, nome: str, codigo: int, preco: int, quantidade: int) -> None:
        pass
    @abc.abstractmethod
    def obtem_nome(self)-> str:
        pass
    @abc.abstractmethod
    def obtem_codigo(self)-> int:
        pass
    @abc.abstractmethod
    def obtem_preco(self)-> int:
        pass
    @abc.abstractmethod
    def obtem_serie(self)-> int:
        pass
    @abc.abstractmethod
    def obtem_lote(self)-> int:
        pass
    @abc.abstractmethod
    def altera_preco(self, novo_preco) -> bool:
        pass
    @abc.abstractmethod
    def altera_quantidade(self, novo_pedido: int) -> bool:
        pass
    @abc.abstractmethod
    def obtem_produto(self) -> str:
        pass
```

Código 9: Classe Produto Abstrata

*obs: **pass** é usada como um espaço reservado quando o código é necessário, mas nenhuma ação específica é exigida. Use **pass** como um espaço reservado para posteriormente implementação.

Interface com Classes Abstratas - Caso com abc

```python
from src import ProdutoAbstrato

class ProdutoConcreto(ProdutoAbstrato):
    serie: int # público
    __lote: int # privado
    def __init__(self, nome: str, codigo: int, preco: int, quantidade: int):
        self.nome = nome
        self.codigo = codigo
        self.preco = preco
        self.quantidade = quantidade
        self.serie = round(codigo * 1000)
        self.__lote = round(1000)
    def obtem_nome(self)-> str:
        return self.nome
    def obtem_codigo(self)-> int:
        return self.codigo
    def obtem_preco(self)-> int:
        return self.preco
    def obtem_serie(self)-> int:
        return self.serie
    def obtem_lote(self)-> int:
        return self.__lote
    def altera_preco(self, novo_preco) -> bool:
        preco_atual = self.preco
        self.preco = novo_preco
        if novo_preco > preco_atual: return True
        return False
    def altera_quantidade(self, novo_pedido: int) -> bool:
        if novo_pedido > self.quantidade: return False
        self.quantidade -= novo_pedido
        return True
    def obtem_produto(self) -> str:
        return '{}-{}-{}-{}-{}-{}'.format(self.__lote, self.serie, self.nome, self.codigo, self.preco, self.quantidade)
```

Código 10: Classe Produto Abstrato

Pytest

```python
from src import ProdutoConcreto
import pytest

@pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade'), [('Arroz', 1, 1, 1), ('Feijao', 1, 1, 1)])
def testa_classe_produto_concreto_geral_com_sucesso(nome, codigo, preco, quantidade):
    produto = ProdutoConcreto(nome, codigo, preco, quantidade)
    assert produto.obtem_nome() == nome and produto.obtem_codigo() == codigo and produto.obtem_preco() == preco and
        produto.obtem_lote() != 0 and produto.obtem_serie() != 0
    assert produto.obtem_produto() == '{}-{}-{}-{}-{}-{}'.format(produto.obtem_lote(), produto.obtem_serie(), nome, codigo,
        preco, quantidade)
    assert produto.altera_preco(2) == True
    assert produto.altera_preco(1) == False
    assert produto.altera_quantidade(2) == False
    assert produto.altera_quantidade(1) == True
```

Código 11: Cobertura de testes da classe Produto Concreto

Pytest

```
---------- coverage: platform linux, python 3.10.12-final-0 ----------
Name                                                 Stmts   Miss  Cover   Missing
----------------------------------------------------------------------------------
src/__init__.py                                         15      0   100%
src/codigo_001_somar.py                                  2      0   100%
src/codigo_002_lambda_somar.py                           1      0   100%
src/codigo_003_classe_produto.py                        31      0   100%
src/codigo_004_classe_produto_critico.py                19      0   100%
src/codigo_005_classe_produto_perecivel.py               7      0   100%
src/codigo_006_classe_veiculo_com_exception.py          18      0   100%
src/codigo_008_input.py                                 14      0   100%
src/codigo_009_print.py                                  9      0   100%
src/codigo_010_classe_singleton.py                      16      0   100%
src/codigo_011_classe_singleton_lazy.py                  7      0   100%
src/codigo_012_classe_produto_abstrato.py                2      0   100%
src/codigo_012_classe_produto_concreto.py               32      0   100%
src/codigo_013_classe_produto_abstrato_excecao.py       10      0   100%
src/codigo_013_classe_produto_concreto_excecao.py       32      0   100%
----------------------------------------------------------------------------------
TOTAL                                                  215      0   100%
Coverage HTML written to dir htmlcov

Required test coverage of 100% reached. Total coverage: 100.00%
```

*obs: *pytest –maxfail=1 –cov-fail-under=100 –disable-warnings –cov=./src –cov-report=html –cov-report=term-missing*

Interface com Classes Abstratas - Caso com Exceção

```python
class ProdutoAbstratoExcecao:

    def __init__(self, nome: str, codigo: int, preco: int, quantidade: int) -> None:
        raise NotImplementedError()
    def obtem_nome(self)-> str:
        raise NotImplementedError()
    def obtem_codigo(self)-> int:
        raise NotImplementedError()
    def obtem_preco(self)-> int:
        raise NotImplementedError()
    def obtem_serie(self)-> int:
        raise NotImplementedError()
    def obtem_lote(self)-> int:
        raise NotImplementedError()
    def altera_preco(self, novo_preco) -> bool:
        raise NotImplementedError()
    def altera_quantidade(self, novo_pedido: int) -> bool:
        raise NotImplementedError()
    def obtem_produto(self) -> str:
        raise NotImplementedError()
```

Código 12: Classe Produto Abstrato Excecao

Interface com Classes Abstratas - Caso com Exceção

```python
from src import ProdutoAbstratoExcecao

class ProdutoConcretoExcecao(ProdutoAbstratoExcecao):
    serie: int # público
    __lote: int # privado
    def __init__(self, nome: str, codigo: int, preco: int, quantidade: int):
        self.nome = nome
        self.codigo = codigo
        self.preco = preco
        self.quantidade = quantidade
        self.serie = round(codigo * 1000)
        self.__lote = round(1000)
    def obtem_nome(self)-> str:
        return self.nome
    def obtem_codigo(self)-> int:
        return self.codigo
    def obtem_preco(self)-> int:
        return self.preco
    def obtem_serie(self)-> int:
        return self.serie
    def obtem_lote(self)-> int:
        return self.__lote
    def altera_preco(self, novo_preco) -> bool:
        preco_atual = self.preco
        self.preco = novo_preco
        if novo_preco > preco_atual: return True
        return False
    def altera_quantidade(self, novo_pedido: int) -> bool:
        if novo_pedido > self.quantidade: return False
        self.quantidade -= novo_pedido
        return True
    def obtem_produto(self) -> str:
        return '{}-{}-{}-{}-{}-{}'.format(self.__lote, self.serie, self.nome, self.codigo, self.preco, self.quantidade)
```

Código 13: Classe Produto Concreto Excecao

Pytest

```python
from src import ProdutoConcretoExcecao
import pytest

@pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade'), [('Arroz', 1, 1, 1), ('Feijao', 1, 1, 1)])
def testa_classe_produto_concreto_excecao_geral_com_sucesso(nome, codigo, preco, quantidade):
  produto = ProdutoConcretoExcecao(nome, codigo, preco, quantidade)
  assert produto.obtem_nome() == nome and produto.obtem_codigo() == codigo and produto.obtem_preco() == preco and
      produto.obtem_lote() != 0 and produto.obtem_serie() != 0
  assert produto.obtem_produto() == '{}-{}-{}-{}-{}-{}'.format(produto.obtem_lote(), produto.obtem_serie(), nome, codigo,
      preco, quantidade)
  assert produto.altera_preco(2) == True
  assert produto.altera_preco(1) == False
  assert produto.altera_quantidade(2) == False
  assert produto.altera_quantidade(1) == True
```

Código 14: Cobertura de testes da classe Produto Concreto Exceção

Lógica de Programação

Padrões de desenvolvimento de software

...continuando Padrões de desenvolvimento de software.

- Padrões de Criação.
  - Singleton ✔
  - Abstract Factory ✔
  - Factory Method ✔
  - Builder ✗
  - Prototype ✗

Padrões de criação - Builder

Builder (Construtor - 'de construir coisas') ✔

- Padrão de criação que se concentra em como construir objetos complexos de maneira controlada e eficiente.
- **Ele separa a construção de um objeto da sua representação final, permitindo a criação de diferentes representações ou configurações do mesmo objeto.**
- Esse padrão é especialmente útil quando um objeto precisa ser construído passo a passo, ou quando o processo de construção é muito complexo.

Padrões de criação - Builder

Exemplo

1. **Classe ProdutoBase** - Será o objeto complexo que queremos construir. Ele pode ter várias partes diferentes que são configuradas durante o processo de construção.
2. **Classe BuilderAbstrata** - É uma interface ou classe abstrata que define os métodos para criar as diferentes partes do Produto.
3. **Classe BuilderConcreta** - Implementa a interface do Builder e constrói as partes específicas do Produto.
4. **Classe Diretor** - define a ordem de construção das partes do Produto. Ele usa o Builder para construir o Produto passo a passo.
5. **Código do cliente para utilizar o Padrão Builder** - Quem usa o Diretor e o Builder para construir os objetos complexos.

Padrões de criação - Builder

Resumo

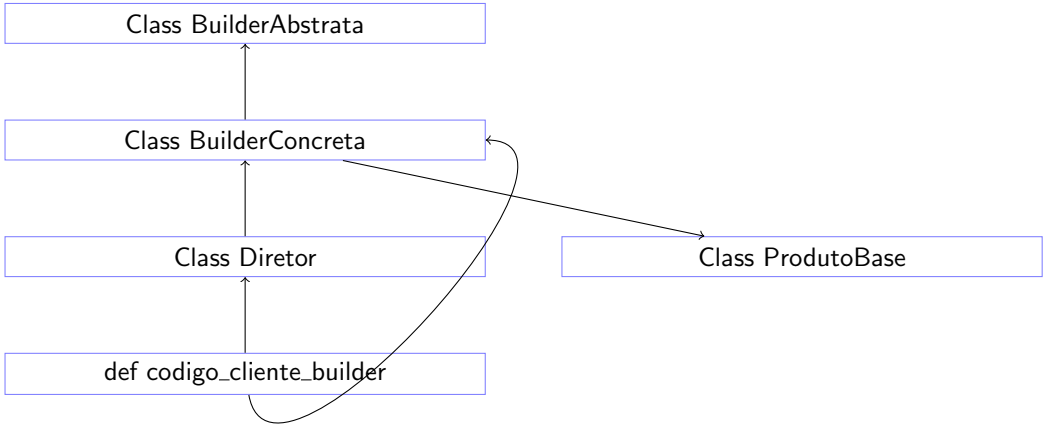
- Produto: O objeto que está sendo construído.
- Builder: Interface que define os métodos de construção.
- BuilderConcreta: Implementa os métodos do Builder e constrói as partes do Produto.
- Diretor: Controla o processo de construção, definindo a ordem em que as partes são construídas.

Padrões de criação - Builder

Aplicação

- O padrão Builder é útil quando queremos construir objetos que exigem uma criação passo a passo, ou quando diferentes representações do mesmo objeto podem ser necessárias.

Padrões de criação - Builder



Padrões de criação - Builder

Class ProdutoBase	Possui →	Lógica do produto.
Class BuilderAbstrata	Possui →	Interface para definir a construção do produto.
Class BuilderConcreta	Possui →	Implementa a lógica para definir a construção do produto.
Class DiretorConcreta	Possui →	Implementa a lógica para definir a ordem de construção do produto.
def codigo_cliente_builder	Possui →	Implementa o código do cliente para obter o produto final.

Padrão de projeto Bulder

```python
class ProdutoBase():
    def __init__(self):
        self.parte1 = None
        self.parte2 = None
        self.parte3 = None

    def __str__(self):
        return f"Produto: Parte1={self.parte1}, Parte2={self.parte2}, Parte3={self.parte3}"
```

Código 15: Código da Classe Produto Base - codigo_017_produto_base.py

```python
import abc

class BuilderAbstrato(metaclass=abc.ABCMeta):
    @abc.abstractmethod
    def build_parte1(self):
        pass

    @abc.abstractmethod
    def build_parte2(self):
        pass

    @abc.abstractmethod
    def build_parte3(self):
        pass
```

Código 16: Código da Classe Builder Abtratato - codigo_017_builder_abstrato.py

Padrão de projeto Bulder

```python
from .codigo_017_builder_abstrato import BuilderAbstrato
from .codigo_017_produto_base import ProdutoBase

class BuilderConcreta(BuilderAbstrato):

    def __init__(self):
        self.produtoBase = ProdutoBase()

    def build_parte1(self)-> str:
        self.produtoBase.parte1 = "Parte 1"

    def build_parte2(self)-> str:
        self.produtoBase.parte2 = "Parte 2"

    def build_parte3(self)-> str:
        self.produtoBase.parte3 = "Parte 3"

    def obter_resultado(self)-> ProdutoBase:
        return self.produtoBase
```

Código 17: Código da Classe Builder Concreta - codigo_017_builder_concreta.py

Padrão de projeto Bulder

```python
from .codigo_017_builder_abstrato import BuilderAbstrato

class Diretor:
    def __init__(self, builder: BuilderAbstrato):
        self.builder = builder

    def construir(self):
        self.builder.build_parte1()
        self.builder.build_parte2()
        self.builder.build_parte3()
```

Código 18: Código da Classe Diretor Concreta - codigo_017_diretor_base.py

Padrão de projeto Bulder

```python
if __name__ == "__main__":
    import sys
    import os
    sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))
    from padroesdeprojetos.criacao.codigo_017_builder_concreta import BuilderConcreta
    from padroesdeprojetos.criacao.codigo_017_diretor_base import Diretor
else:
    from .codigo_017_builder_concreta import BuilderConcreta
    from .codigo_017_diretor_base import Diretor

def codigo_cliente_builder():
    # Criando um ConcreteBuilder
    builder = BuilderConcreta()

    # Passando o builder para o Director
    diretor = Diretor(builder)

    # O diretor constrói o produto
    diretor.construir()

    # Obtendo o produto final
    produto = builder.obter_resultado()

    print(f"{produto}")

if __name__ == "__main__":
    codigo_cliente_builder()
```

Código 19: Código da Função Código do cliente - codigo_017_codigo_cliente_builder.py

Padrões de Desenvolvimento de Software

Exemplos de criação de classe

...continuando a partir da aula passada sobre POO.

Tratamento de exceções

O tratamento de exceção, na ciência da computação, é o mecanismo responsável pelo tratamento da ocorrência de condições que alteram o fluxo normal da execução de programas de computadores.

Para condições consideradas parte do fluxo normal de execução, ver os conceitos de sinal e evento.

Saiba mais em

https://docs.python.org/pt-br/3/whatsnew/2.6.html#pep-3110-exception-handling-changes

https://docs.python.org/pt-br/3/reference/compound_stmts.html#except-clause

https://docs.pytest.org/en/stable/how-to/assert.html

Tratamento de exceções

A sintaxe básica é:

```
try:
   Instruções # o código da funcionalidade.
...
except <ExceptType>:
   Instruções # o código para tratamento da exceção.
...
finally: # Caso o fluxo não seja interrompido, sempre é executado o finally.
   Instruções
```

https://docs.python.org/pt-br/3/tutorial/errors.html

Tratamento de exceções

```python
class Veiculo:
    chassi: int # público
    __motor: int # privado
    def __init__(self, chassi: int, motor: int):
        self.chassi = chassi
        self.__motor = motor
    def obtem_chassi(self)-> int:
        return self.chassi
    def obtem_motor(self)-> int:
        return self.__motor
    def altera_chassi(self, novo_chassi: int) -> bool:
        try:
            if novo_chassi != self.chassi:
                self.chassi = novo_chassi
                return True
            else:
                raise ValueError("Chassi igual ao atual")
        except ValueError as error:
            raise ValueError(error)
```

Código 20: Tratamento de exceções

Pytest

```python
from src import Veiculo
import pytest

@pytest.mark.parametrize(('chassi', 'motor'), [('KHG969878976G7DF9G7', 'SGD97S9')])
def teste_obtem_chassi_e_motor_com_sucesso(chassi, motor):
  veiculo = Veiculo(chassi, motor)
  assert  veiculo.obtem_chassi() == chassi and veiculo.obtem_motor() == motor

@pytest.mark.parametrize(('chassi', 'motor', 'novo_chassi'), [('KHG969878976G7DF9G7', 'SGD97S9', 'KHG969878976G7DF9G7')])
def teste_alterar_chassis_iguais(chassi, motor, novo_chassi):
  veiculo = Veiculo(chassi, motor)
  with pytest.raises(ValueError) as excinfo:
    veiculo.altera_chassi(novo_chassi)
  assert str(excinfo.value) == "Chassi igual ao atual" and excinfo.type is ValueError

@pytest.mark.parametrize(('chassi', 'motor', 'novo_chassi'), [('KHG969878976G7DF9G7', 'SGD97S9', 'KHG969878976G7DF9G6')])
def teste_alterar_chassis_diferentes(chassi, motor, novo_chassi):
  veiculo = Veiculo(chassi, motor)
  assert veiculo.altera_chassi(novo_chassi) == True
```

Código 21: Cobertura de testes da classe Veiculo

Polimorfismo em classe

Polimorfismo permite que objetos de diferentes classes sejam tratados como objetos de uma classe comum.

- Polimorfismo é o princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma identificação, assinatura, mas comportamentos distintos, especializados para cada classe derivada, usando para tanto uma referência a um objeto do tipo da superclasse.

Polimorfismo em classe

```python
from src.codigo_003_classe_produto import Produto

class ProdutoPerecivel(Produto):
    def __init__(self, nome: str, codigo: int, preco: int, quantidade: int, validade: int):
        super().__init__(nome, codigo, preco, quantidade)
        self.validade = validade

    def obtem_produto(self) -> str:
        return '{}-{}-{}-{}-{}'.format(self.nome, self.codigo, self.preco, self.quantidade, self.validade)
```

Código 22: Polimorfismo de classe

*obs: **class ProdutoPerecivel(Produto)** é uma classe derivada de Produto: indica que **ProdutoPerecivel** é uma subclasse da classe **Produto**.

Pytest

```python
from src import ProdutoPerecivel
import pytest

@pytest.mark.parametrize(('nome', 'codigo', 'preco', 'quantidade', 'validade'), [('Arroz', 1, 10, 100, 20)])
def teste_obtem_produto_com_sucesso(nome, codigo, preco, quantidade, validade):
  produto_perecivel = ProdutoPerecivel(nome, codigo, preco, quantidade, validade)
  assert  produto_perecivel.obtem_produto() == '{}-{}-{}-{}-{}'.format(nome, codigo, preco, quantidade, validade)
```

Código 23: Cobertura de testes da classe Produto Perecivel

Pytest

```
python3 -m pytest --cov -q test_codigo_005_classe_produto_repecivel.py
.

---------- coverage: platform linux, python 3.10.12-final-0 ----------
Name                                                 Stmts   Miss  Cover
------------------------------------------------------------------------
codigo_003_classe_produto.py                            23     11    52%
codigo_005_classe_produto_perecivel.py                   7      0   100%
test_codigo_005_classe_produto_repecivel.py              6      0   100%
------------------------------------------------------------------------
TOTAL                                                   36     11    69%

1 passed in 0.02s
```

Pytest

```
python3 -m pytest --cov
============================================================================= test session starts
=====================================================
platform linux -- Python 3.10.12, pytest-8.3.2, pluggy-1.5.0
rootdir: /library/personal/repository/latex/softex-2024-material-backend-python/outros/codigos/py
plugins: metadata-3.1.1, cov-5.0.0
collected 22 items


test_codigo_001_somar.py ..
                                                    [  9%]
test_codigo_002_lambda_somar.py ..
                                                    [ 18%]
test_codigo_003_classe_produto.py ...........
                                                    [ 72%]
test_codigo_004_classe_produto_critico.py .....
                                                    [ 95%]
test_codigo_005_classe_produto_repecivel.py .
                                                    [100%]

---------- coverage: platform linux, python 3.10.12-final-0 ----------
Name                                               Stmts   Miss  Cover
----------------------------------------------------------------------
codigo_001_somar.py                                    2      0   100%
codigo_002_lambda_somar.py                             1      0   100%
codigo_003_classe_produto.py                          23      0   100%
codigo_004_classe_produto_critico.py                  17      0   100%
codigo_005_classe_produto_perecivel.py                 7      0   100%
test_codigo_001_somar.py                               5      0   100%
test_codigo_002_lambda_somar.py                        5      0   100%
test_codigo_003_classe_produto.py                     26      0   100%
test_codigo_004_classe_produto_critico.py             22      0   100%
test_codigo_005_classe_produto_repecivel.py            6      0   100%
----------------------------------------------------------------------
TOTAL                                                114      0   100%


============================================================================= 22 passed in 0.10s =
```

Polimorfismo em classe

Sobreposição.

- Sobreposição (Override): é sobrescrever, ou seja, definir um novo comportamento para um método que já existe. Isso acontece quando a classe em questão herda (estende) outra classe e se cria um método com a mesma assinatura da classe "pai" na classe filha.

Encapsulamento em classe

Encapsulamento é a proteção dos atributos ou métodos de uma classe, em Python existem somente o public e o private e eles são definidos no próprio nome do atributo ou método.

```
class Veiculo:
    chassi = 1 # atributo publico
    __motor = 2 # atributo privado a classe Veiculo. O símbolo __* define como privado.

class Carro(Veiculo):
    __placa = 3 # atributo privado a classe Carro

    def __init__(self):
        print(self.chassi)
        print(self.__placa)

veiculo = Veiculo()
print(veiculo.chassi) # imprime 1

carro = Carro() # Erro
# print(carro.__motor) # Erro, pois __motor é privado a classe Veiculo.
# print(carro.__placa) # Erro, __placa é um atributo privado, somente chamado pela classe Carro.
```

Encapsulamento em classe

Exemplo anteior

```
class Veiculo:
    chassi = 1 # atributo publico
    __motor = 2 # atributo privado a classe Veiculo. O símbolo __* define como privado.

class Carro(Veiculo):
    __placa = 3 # atributo privado a classe Carro

    def __init__(self):
        print(self.chassi)
        print(self.__placa)

veiculo = Veiculo()
print(veiculo.chassi) # imprime 1

carro = Carro() # Erro
# print(carro.__motor) # Erro, pois __motor é privado a classe Veiculo.
# print(carro.__placa) # Erro, __placa é um atributo privado, somente chamado pela classe Carro.
```

Lógica de Programação

Backend
Python com Django

Prof. Bruno Iran Ferreira Maciel

BRUNO

Aug 2024

Exercícios 1

Exercício 001 *Ler 4 valores (considere que não serão informados valores iguais). Escreva a soma dos dois últimos números.* 19

Exercício 002 *Ler 2 valores e se o segundo valor informado for ZERO, deve ser lido um novo valor, ou seja, para o segundo valor não pode ser aceito o valor zero e imprimir o resultado da divisão do primeiro valor lido pelo segundo valor lido. (utilizar a estrutura REPETIR)* 19

Exercício 003 *Ler as idades de 2 homens e de 2 mulheres (considere que as idades dos homens serão sempre diferentes entre si, bem como as das mulheres). Calcule e escreva a soma das idades do homem mais velho com a mulher mais nova, e o produto das idades do homem mais novo com a mulher mais velha.* 19

Exercício 004 *Ler o salário fixo e o valor total das vendas efetuadas pelo vendedor de uma empresa. Sabendo-se que ele recebe uma comissão de 3% sobre o total das vendas até R$ 1.500,00 mais 5% sobre o que ultrapassar este valor, calcular e escrever o seu salário total.* 19

Exercício 005 *Ler 11 valores numéricos, somar os 10 primeiros e guardar em uma variável A e o décimo primeiro valor, guardar em uma variável B. Escreva os valores de A e B. A seguir (utilizando apenas atribuições entre variáveis) troque os seus conteúdos fazendo com que o valor que está em A passe para B e vice-versa. Ao final, escreva os valores que ficaram armazenados nas variáveis.* 31

Exercícios 2

Exercícios 3

Exercícios 4

Referências

Referências 1

G. Ghetan. *Aprendendo Padrões de Projeto em Python*. Novatec, 1 edition, 2016.