Projeto POO - Biblioteca

- Aluna: Sofia Estrela Bernardes RGM: 38683555
- Aluna: Ana Iza Gomes Juvino RGM: 38589711

- Data Entrega: 05/06/2025
- Turma: ADS D2

Objetivo do projeto: Introduzir os conceitos de Programação Orientado a Objetos (POO) em Python e aplicar esses conhecimentos no planejamento e desenvolvimento de projetos.

O projeto tem como objetivo desenvolver um sistema de gerenciamento de biblioteca utilizando os princípios da Programação Orientada a Objetos (POO) em Python.

O sistema permite cadastrar e gerenciar os principais elementos de uma biblioteca, como:

1. Usuários que utilizam os serviços da biblioteca;
2. Funcionários responsáveis pelos atendimentos;
3. Livros disponíveis para empréstimo ou doação;
4. Empréstimos, registrando quando um livro é emprestado e por quem;
5. Doações de livros feitas por usuários ou de forma anônima.

Cada entidade (como Usuario, Livro, Emprestimo, Doação, Funcionário) é representada por uma classe Python, com seus próprios atributos e métodos. As informações são armazenadas em um banco de dados relacional, e o sistema oferece um menu de opções para facilitar a navegação e o uso.

- Função Main

```
from classes.usuario import Usuario           # Uso de classes para representar enti
from classes.funcionario import Funcionario
from classes.livro import Livro
from classes.emprestimo import Emprestimo
from classes.doacao import Doacao
from datetime import date

# Função que instancia objetos da classe Usuario e utiliza seu método salvar
def cadastrar_usuarios(): # Cria o método cadastrar usuários, enquanto a função esti
    while True:
        print("\nCadastro de Usuário")
        nome = input("Nome: ")
```

```
        cpf = input("CPF: ")
        email = input("Email: ")
        telefone = input("Telefone: ")
        endereco = input("Endereço: ")

        usuario = Usuario(nome=nome, cpf=cpf, email=email, telefone=telefone, endere
        usuario.salvar()  # Método da classe sendo utilizado

        continuar = input("Deseja cadastrar outro usuário? (s/n): ").lower()
        if continuar != 's':
            break

#Cadastro de funcionários – outra classe instanciada com seus atributos
def cadastrar_funcionarios():
    while True:
        print("\nCadastro de Funcionário")
        nome = input("Nome do funcionário: ")
        cpf = input("CPF do funcionário: ")
        cargo = input("Cargo do funcionário: ")

        funcionario = Funcionario(nome, cpf, cargo)  #Objeto da classe Funcionario
        funcionario.salvar()

        continuar = input("Deseja cadastrar outro funcionário? (s/n): ").lower()
        if continuar != 's':
            break

# Cadastro de livros – instanciando a classe Livro e usando o método salvar
def cadastrar_livros():
    while True:
        print("\nCadastro de Livro")
        titulo = input("Título: ")
        autor = input("Autor: ")
        ano_publicacao = input("Ano de publicação: ")
        editora = input("Editora: ")
        genero = input("Gênero: ")
        isbn = input("ISBN: ")

        livro = Livro(titulo=titulo, autor=autor, ano_publicacao=ano_publicacao,
                      editora=editora, genero=genero, isbn=isbn)
        livro.salvar()

        continuar = input("Deseja cadastrar outro livro? (s/n): ").lower()
        if continuar != 's':
            break

# Cadastro de empréstimos – classe com relacionamento entre Livro, Usuario e Funcion
def cadastrar_emprestimo():
    while True:
        try:
            print("\nCadastro de Empréstimo")
            cod_livro = int(input("Código do livro: "))
            cod_usuario = int(input("Código do usuário: "))
            cod_funcionario = int(input("Código do funcionário: "))
            data_emprestimo = date.today()  # Definindo atributo de data com a data
```

```
            # Objeto da classe Emprestimo sendo criado com seus atributos
            emprestimo = Emprestimo(cod_livro, cod_usuario, cod_funcionario, data_em
            emprestimo.salvar()
        except Exception as e:
            print("Erro ao salvar empréstimo:", e)

        continuar = input("Deseja cadastrar outro empréstimo? (s/n): ").lower()
        if continuar != 's':
            break

#  Cadastro de doações — classe com atributos e condição para anonimato
def cadastrar_doacao():
    while True:
        try:
            print("\nCadastro de Doação")
            cod_livro = int(input("Código do livro doado: "))
            anonimo = input("Deseja doar anonimamente? (s/n): ").lower()

            cod_usuario = None
            if anonimo != 's':
                cod_usuario = int(input("Código do usuário doador: "))

            data_doacao = date.today()

            # Objeto da classe Doacao sendo instanciado
            doacao = Doacao(cod_livro, data_doacao, cod_usuario)
            doacao.salvar()
        except Exception as e:
            print("Erro ao salvar doação:", e)

        continuar = input("Deseja cadastrar outra doação? (s/n): ").lower()
        if continuar != 's':
            break

# Menu principal para navegar entre funcionalidades — interface simples
def main():
    while True:
        print("\n=== MENU PRINCIPAL ===")
        print("1. Cadastrar Usuário")
        print("2. Cadastrar Funcionário")
        print("3. Cadastrar Livro")
        print("4. Cadastrar Empréstimo")
        print("5. Cadastrar Doação")
        print("0. Sair")

        opcao = input("Escolha uma opção: ")

        if opcao == '1':
            cadastrar_usuarios()
        elif opcao == '2':
            cadastrar_funcionarios()
        elif opcao == '3':
            cadastrar_livros()
        elif opcao == '4':
```

```
            cadastrar_emprestimo()
        elif opcao == '5':
            cadastrar_doacao()
        elif opcao == '0':
            print("Saindo do sistema...")
            break
        else:
            print("Opção inválida. Tente novamente.")

# Ponto de entrada do programa
if __name__ == "__main__":
    main()

```

- Classe Usuário

```
from db.conexao import conectar #Faz conexão com as classes declaradas no banco de da

class Usuario:  # Classes e Objetos — Definição de classe
    def __init__(self, nome, cpf, email, telefone, endereco):  #Atributos da classe d
        self.nome = nome
        self.cpf = cpf  #Encapsulamento sugerido (privado lógico)
        self.email = email
        self.telefone = telefone
        self.endereco = endereco

    def salvar(self):  #Método — comportamento da classe
        conexao = conectar()
        cursor = conexao.cursor()

        try:
            #Método que insere os dados no banco de dados (ação da classe)
            sql = "INSERT INTO tb_usuario (nome, cpf, email, telefone, endereco) VALU
            valores = (self.nome, self.cpf, self.email, self.telefone, self.endereco)
            cursor.execute(sql, valores)
            conexao.commit() # cursor.execute executa instruções do banco de dados en
            print("Usuário salvo com sucesso!")  #Método com saída de controle
        except Exception as e:
            print("Erro ao salvar usuário:", e)
        finally:
            cursor.close()
            conexao.close() #Caso haja erro ou falha nas execuções das instruções, as

```

- Classe Funcionário

```
from db.conexao import conectar #Faz conexão com as classes declaradas no banco de da

class Funcionario:  # Classes e Objetos – definição de uma entidade do sistema
    def __init__(self, nome, matricula, cargo):
        # Atributos – representam as características do objeto
        self.nome = nome
        self.matricula = matricula
        self.cargo = cargo

    def salvar(self):  # Método – comportamento responsável por persistência no banco
        conexao = conectar()
        cursor = conexao.cursor()

        sql = """
            INSERT INTO tb_funcionario (nome, matricula, cargo)
            VALUES (%s, %s, %s)
        """
        valores = (self.nome, self.matricula, self.cargo)

        # Lógica de persistência – ação do objeto
        cursor.execute(sql, valores)
        conexao.commit()

        cursor.close()
        conexao.close()

        print(f"Funcionário {self.nome} salvo com sucesso!")  #Feedback de ação execu

```

- Classe Livro

```
from db.conexao import conectar #Faz conexão com as classes declaradas no banco de da


class Livro:  #Classes e Objetos – definição da estrutura da entidade Livro
    def __init__(self, titulo, autor, ano_publicacao, editora, genero, isbn):
        # Atributos – definem as características do objeto Livro
        self.titulo = titulo
        self.autor = autor
        self.ano_publicacao = ano_publicacao
        self.editora = editora
        self.genero = genero
        self.isbn = isbn

    def salvar(self):  #Método – comportamento da classe Livro
        conexao = conectar()
        cursor = conexao.cursor()

        # Método com lógica de persistência no banco de dados
        sql = """
            INSERT INTO tb_livro (titulo, autor, editora, ano_publicacao, genero, isb
```

```
            VALUES (%s, %s, %s, %s, %s, %s)
        """
        valores = (
            self.titulo, self.autor, self.editora,
            self.ano_publicacao, self.genero, self.isbn
        )

        cursor.execute(sql, valores)
        conexao.commit()

        cursor.close()
        conexao.close()

        print(f"Livro '{self.titulo}' salvo com sucesso!")  #Confirmação de execução

```

- Classe Empréstimo

```
from db.conexao import conectar # conexão com banco de dados
from datetime import date # módulo que lida com datas e horários

class Emprestimo:  #Classes e Objetos — Definição da classe
    def __init__(self, data_emprestimo, data_prevista, cod_usuario, cod_funcionario,
        # ✔ 1. Atributos — características de um empréstimo
        self.data_emprestimo = data_emprestimo
        self.data_prevista = data_prevista
        self.data_real = data_real
        self.cod_usuario = cod_usuario
        self.cod_funcionario = cod_funcionario

    def salvar(self):  #Método — comportamento: salvar no banco
        conexao = conectar()
        if conexao is None:
            print("Não foi possível salvar o empréstimo.")
            return
        cursor = conexao.cursor()

        sql = """
            INSERT INTO tb_emprestimo (data_emprestimo, data_prevista, data_real, cod
            VALUES (%s, %s, %s, %s, %s)
        """
        valores = (self.data_emprestimo, self.data_prevista, self.data_real, self.cod

        try:
            cursor.execute(sql, valores)
            conexao.commit()
            print("Empréstimo registrado com sucesso!")  #Polimorfismo possível em `s
        except Exception as e:
            print(f"Erro ao salvar empréstimo: {e}")
        finally:
            cursor.close()
```

```
            conexao.close()

```

- Classe Doação

```
from db.conexao import conectar

class Doacao:  # Classe — representa a entidade "Doação"
    def __init__(self, livro, data_doacao, cod_usuario):
        # Atributos — definem características de cada doação
        self.livro = livro                # código ou nome do livro doado
        self.data_doacao = data_doacao    # data do registro da doação
        self.cod_usuario = cod_usuario    # pode ser um usuário identificado ou anôni

    def salvar(self):  # Método — comportamento que salva a doação no banco
        conexao = conectar()
        cursor = conexao.cursor()

        # Lógica de persistência — insere os dados na tabela de doações
        sql = "INSERT INTO tb_doacao (cod_livro, data_doacao, cod_usuario) VALUES (%s
        valores = (self.livro, self.data_doacao, self.cod_usuario)

        try:
            cursor.execute(sql, valores)
            conexao.commit()
            print("Doação salva com sucesso!")  #Confirmação do comportamento executa
        except Exception as e:
            print(f"Erro ao salvar doação: {e}")
        finally:
            cursor.close()
            conexao.close()

def main():
    print("Cadastro de Doação")
    livro = input("Nome do livro: ")
    data_doacao = input("Data da doação (AAAA-MM-DD): ")

    # Planejamento: cod_usuario pode ser 0 (anônimo) ou um ID real
    cod_usuario_anonimo = 0

    # Objeto sendo instanciado com os dados informados
    doacao = Doacao(livro, data_doacao, cod_usuario_anonimo)
    doacao.salvar()  # Método sendo chamado

if __name__ == "__main__":
    main()


```