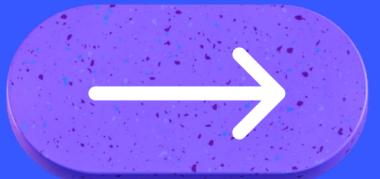


# Agenda de Barbearia



# Integrantes

Matheus Ferreira

Andre Luiz

Vinicius Paiva

Diogo Freitas

Pedro Rodrigues



# Motivação

Não deixar clientes  
esperando sentados

Garantia de  
horário

Facilitar o  
atendimento

Atendimento  
personalizado

**Visão Diária/Semanal:**  
Permita aos usuários ver a agenda diária ou semanal.

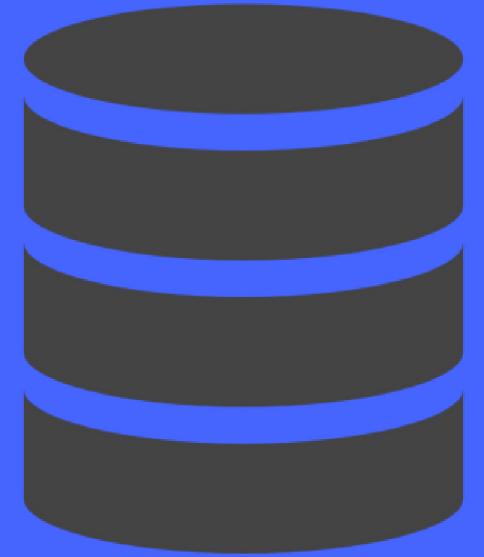
**Formulário de Agendamento:**  
Um formulário simples para agendar compromissos, incluindo campos para nome do cliente, serviço desejado, data e hora.

**Adicionar/Editar/Excluir Compromissos:**  
Funcionalidade para gerenciar compromissos facilmente





Aplicativo desenvolvido  
totalmente pela plataforma da  
NetBeans

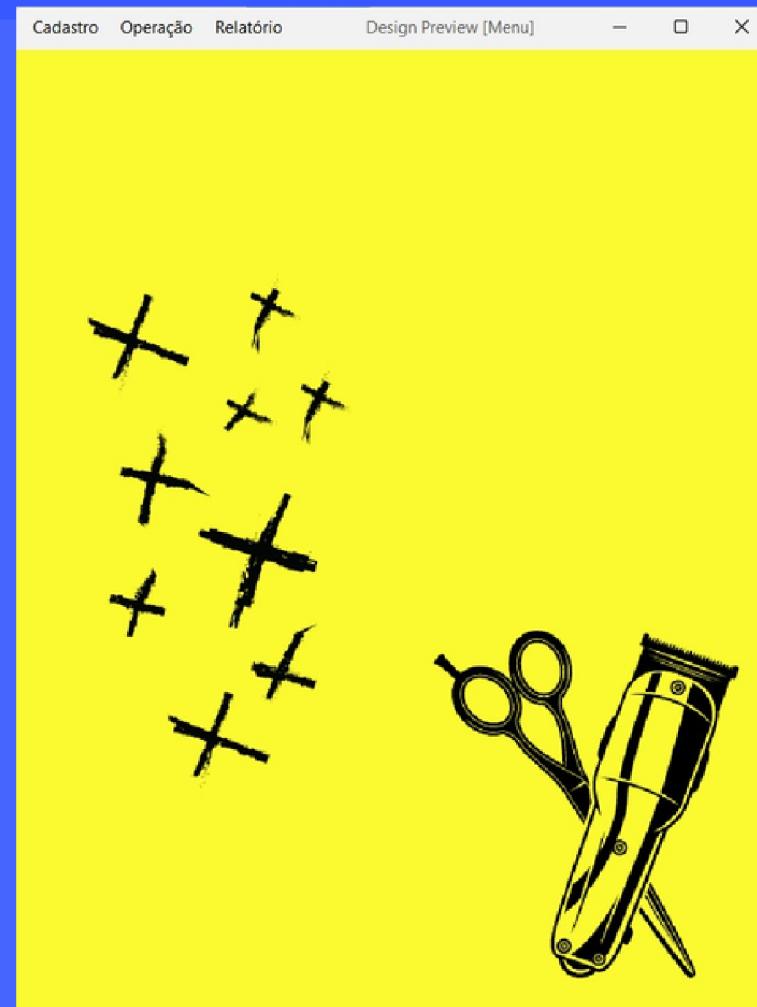


Usamos o padrão DAO como  
banco de dados

# Front



# Login



# Menu



# Agenda

# Desenvolvimento Back - End

Herança

Polimorfismo

Classes

MVC

# Herança

---

A herança é um conceito fundamental na programação orientada a objetos em Java, permitindo que uma classe (subclasse) herde características (atributos e métodos) de outra classe (superclasse).

## Conceitos Básicos

- Superclasse (classe pai): Classe que é herdada.
- Subclasse (classe filha): Classe que herda de outra classe.
- Sintaxe: class Subclasse extends Superclasse

# Polimorfismo

---

O polimorfismo é um dos pilares da programação orientada a objetos (POO) e permite que objetos de diferentes classes sejam tratados de maneira uniforme. Em Java, existem dois tipos principais de polimorfismo: polimorfismo de tempo de compilação (sobrecarga) e polimorfismo de tempo de execução (sobrescrita).

## Tipos de Polimorfismo

### 1. Polimorfismo de Tempo de Compilação (Sobrecarga de Métodos)

- Ocorre quando dois ou mais métodos na mesma classe têm o mesmo nome, mas parâmetros diferentes (tipo, número ou ambos).
- Não depende da herança.

# Classes

---

Em Java, uma classe é uma estrutura fundamental da programação orientada a objetos, que define um tipo de objeto, especificando os dados (atributos) e os comportamentos (métodos) que os objetos desse tipo podem ter. Aqui está um resumo abrangente sobre classes em Java.

# MVC

- 
- MVC (Model-View-Controller) é um padrão de arquitetura que separa a aplicação em três componentes principais: Model, View e Controller. Essa separação facilita a organização do código, promovendo a reutilização, a escalabilidade e a manutenção.

# Classes

```
protected int id;
protected String nome;
protected char sexo;
protected Date dataDeNascimento;
protected String telefone;
protected String email;
protected String rg;

public Pessoa(int id, String nome) {
    this.id = id;
    this.nome = nome;
}

//Construtores
public Pessoa(int id, String nome, char sexo, String dataDeNascimento, String telefone, String email, String rg) {
    this.id = id;
    this.nome = nome;
    this.sexo = sexo;
    try{
        this.dataDeNascimento = new SimpleDateFormat(pattern: "dd/MM/yyyy").parse(source: dataDeNascimento);
    } catch (ParseException ex){
        Logger.getLogger(name: Agendamento.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
    }
    this.telefone = telefone;
    this.email = email;
    this.rg = rg;
}

//Gets e Sets
public int getId() {
    return id;
}
```

# Controllers

```
* @author aabrr
*/
public class LoginController {
    // Atributos que referenciam a view e um helper para manipulação de dados da view
    private final Login view;
    private LoginHelper helper;

    // Construtor da classe que inicializa a view e o helper
    public LoginController(Login view) {
        this.view = view;
        this.helper = new LoginHelper(view);
    }

    //Metodo para entrar no sistema
    public void entrarNoSistema(){
        //Obtem o modelo de usuario da view
        Usuario usuario = helper.obterModelo();
        //Cria um novo usuario no DAO, que é o banco de dados
        UsuarioDAO usuarioDAO = new UsuarioDAO();
        //Valida o usuario
        Usuario usuarioAutenticado = usuarioDAO.selectPorNomeESenha(usuario);
        //If e else para verificar se existe o usuario no banco de dados, caso tenha o menu entra e caso não tenha fala que o Usuario ou a senha são invalidos
        if(usuarioAutenticado != null){
            Menu menuPrincipal = new Menu();
            menuPrincipal.setVisible(b: true);
            this.view.dispose();
        }
        else{
            view.exibeMensagem(mensagem: "Usuario ou senha invalidos");
        }
    }

    public void fizTarefa(){
        System.out.println(x: "Busquei no banco de Dados");
        this.view.exibeMensagem(mensagem: "Executei o fiz tarefa");
    }
}
```

# Helpers

```
// Referência à view que será manipulada por este helper
private final Login view;

// Construtor que inicializa a view
public LoginHelper(Login view) {
    this.view = view;
}

// Método para obter um modelo de usuário a partir dos dados da view
@Override
public Usuario obterModelo() {
    String nome = view.getTextUsuario().getText(); // Obtém o nome de usuário da view
    String senha = view.getTextSenha().getText(); // Obtém a senha da view
    Usuario modelo = new Usuario(id: 0, nome, senha); // Cria um novo objeto Usuario com os dados obtidos
    return modelo;
}

// Método para definir os dados de um modelo de usuário na view
public void setarModelo(Usuario modelo) {
    String nome = modelo.getNome(); // Obtém o nome do modelo de usuário
    String senha = modelo.getSenha(); // Obtém a senha do modelo de usuário

    view.getTextUsuario().setText(t: nome); // Define o campo de texto do nome de usuário na view
    view.getTextSenha().setText(t: senha); // Define o campo de texto da senha na view
}

// Método para limpar os campos da view
@Override
public void limparTela() {
    view.getTextUsuario().setText(t: ""); // Limpa o campo de texto do nome de usuário
    view.getTextSenha().setText(t: ""); // Limpa o campo de texto da senha
}
```

# Considerações

Com este projeto, foi criado um sistema automatizado com a finalidade de executar agendamentos de horários em barbearias, facilitando para clientes e proprietários.

