Manual Detalhado do Sistema Fiesta!!!

Este manual explica o funcionamento do sistema **Fiesta!!!**, um software de gerenciamento de aluguéis para festas. O sistema é implementado em C# e utiliza uma interface de console para gerenciar clientes, temas, aluguéis, devoluções e relatórios. Abaixo, cada arquivo do código fonte é analisado detalhadamente, descrevendo sua estrutura, funcionalidades e como cada linha contribui para o sistema.

1. Arquivo: `Program.cs`

Este arquivo contém a classe principal do programa, responsável por inicializar o sistema e exibir o menu principal.

```
namespace Fiesta
{
    internal class Program
    {
```

- **Namespace Fiesta**: Agrupa todas as classes do sistema no namespace `Fiesta`.
- **Classe Program**: Classe principal com o ponto de entrada do programa, marcada como `internal` para visibilidade apenas dentro do namespace.

```
static void Main(string[] args)
{
```

- **Método Main**: Ponto de entrada do programa, onde a execução começa. Recebe argumentos da linha de comando, mas não os utiliza.

```
List<string> opcoes = new List<string>();
opcoes.Add("1 - Manter Clientes");
```

```
opcoes.Add("2 - Manter Temas");
opcoes.Add("3 - Registrar Aluguel");
opcoes.Add("4 - Registrar Devolução");
opcoes.Add("5 - Listar Aluguéis");
opcoes.Add("0 - Sair");
```

- **Lista de opções**: Cria uma lista de strings representando as opções do menu principal.
- Cada `Add` insere uma opção, como gerenciar clientes, temas, aluguéis, devoluções, relatórios ou sair do sistema.

```
Tela tela = new Tela(80, 25, ConsoleColor.Black, ConsoleColor.White);
```

- **Instanciação da classe Tela**: Cria um objeto `tela` com dimensões 80x25 caracteres, fundo preto e texto branco para gerenciar a interface do console.

```
ClienteCRUD clienteCRUD = new ClienteCRUD();
TemaCRUD temaCRUD = new TemaCRUD();
AluguelCRUD aluguelCRUD = new AluguelCRUD(clienteCRUD, temaCRUD);
```

- **Instanciação de CRUDs**: Cria objetos para gerenciar operações CRUD (Create, Read, Update, Delete) de clientes (`ClienteCRUD`), temas (`TemaCRUD`) e aluguéis (`AluguelCRUD`).
- `AluguelCRUD` recebe `clienteCRUD` e `temaCRUD` como dependências para validar clientes e temas durante o registro de aluguéis.

```
string opcao;
```

- **Variável opcao**: Armazena a escolha do usuário no menu.

```
while (true)
```

```
{
```

- **Loop principal**: Mantém o programa em execução até que o usuário escolha sair (opção "0").

```
tela.prepararTela("Fiesta!!!");
```

- **Preparar tela**: Limpa o console, desenha molduras e exibe o título "Fiesta!!!" centralizado.

```
opcao = tela.mostrarMenu(opcoes, 1, 2);
```

- **Mostrar menu**: Exibe as opções do menu na posição (coluna 1, linha 2) e captura a escolha do usuário, armazenada em `opcao`.

```
switch (opcao)
{
    case "1":
        clienteCRUD.ExecutarCRUD();
        break;
    case "2":
        temaCRUD.ExecutarCRUD();
        break;
    case "3":
        aluguelCRUD.RegistrarAluguel();
        break;
    case "4":
        aluguelCRUD.RegistrarDevolucao();
        break;
    case "5":
        aluguelCRUD.MostrarRelatorio();
        break;
    case "0":
        Console.WriteLine("Saindo...");
```

```
            return;
        default:
            Console.WriteLine("Opção inválida!");
            break;
    }
```

- **Switch para opções**: Avalia a escolha do usuário:
  - **"1"**: Executa o CRUD de clientes, permitindo cadastrar, alterar ou excluir clientes.
  - **"2"**: Executa o CRUD de temas, para gerenciar temas de festas.
  - **"3"**: Registra um novo aluguel.
  - **"4"**: Registra a devolução de um aluguel.
  - **"5"**: Exibe um relatório de aluguéis (ativos ou finalizados).
  - **"0"**: Exibe "Saindo..." e termina o programa com `return`.
  - **default**: Exibe "Opção inválida!" para entradas não reconhecidas.

2. Arquivo: `Tela.cs`

A classe `Tela` gerencia a interface de console, desenhando molduras, centralizando textos e exibindo menus.

```
public class Tela
{
```

- **Classe Tela**: Define métodos para manipular a interface gráfica do console.

```
private int largura;
private int altura;
private int linhaMensagem;
```

```
private int colunaIniMensagem;
private int colunaFinMensagem;
private ConsoleColor corFundo;
private ConsoleColor corTexto;
```

- **Atributos privados**:
  - `largura` , `altura` : Dimensões da tela (em caracteres).
  - `linhaMensagem` , `colunaIniMensagem` , `colunaFinMensagem` : Posições para mensagens na parte inferior da tela.
  - `corFundo` , `corTexto` : Cores de fundo e texto do console.

```
public Tela(int largura, int altura, ConsoleColor fundo, ConsoleColor texto)
{
    this.largura = largura;
    this.altura = altura;
    this.corFundo = fundo;
    this.corTexto = texto;
    this.definirMedidasMensagem();
}
```

- **Construtor parametrizado**: Inicializa a tela com largura, altura e cores especificadas.
- Chama `definirMedidasMensagem` para configurar as posições das mensagens.

```
public Tela()
{
    this.largura = 80;
    this.altura = 25;
    this.corFundo = ConsoleColor.Black;
    this.corTexto = ConsoleColor.White;
    this.definirMedidasMensagem();
}
```

- **Construtor padrão**: Define valores padrão (80x25, fundo preto, texto branco) e chama `definirMedidasMensagem`.

```
private void definirMedidasMensagem()
{
    this.linhaMensagem = this.altura - 1;
    this.colunaIniMensagem = 1;
    this.colunaFinMensagem = this.largura - 1;
}
```

- **Método definirMedidasMensagem**: Configura a linha de mensagens como a última linha da tela e as colunas inicial e final para mensagens.

```
public void prepararTela(string titulo)
{
    Console.ForegroundColor = corTexto;
    Console.BackgroundColor = corFundo;
    Console.Clear();
    this.desenharMoldura(0, 0, this.largura, this.altura);  // tela
    this.desenharMoldura(0, 0, this.largura, 2);            // titulo
    this.desenharMoldura(0, this.altura-2, this.largura, this.altura); // rodape
    this.centralizar(titulo, 1, 0, this.largura);
}
```

- **Método prepararTela**: Prepara a interface inicial:
  - Define as cores de texto e fundo.
  - Limpa o console com `Console.Clear`.
  - Desenha três molduras: uma para a tela inteira, uma para o título (linhas 0 a 2) e uma para o rodapé (últimas duas linhas).
  - Centraliza o título na linha 1.

```
public void limparArea(int ci, int li, int cf, int lf)
{
    for (int x = ci; x <= cf; x++)
    {
        for (int y = li; y <= lf; y++)
        {
            Console.SetCursorPosition(x,y);
            Console.Write(" ");
        }
    }
}
```

- **Método limparArea**: Limpa uma área retangular do console (de `ci,li` a `cf,lf`) preenchendo com espaços em branco.

```
public void desenharMoldura(int colIni, int linIni, int colFin, int linFin)
{
    // limpar a area da moldura
    this.limparArea(colIni, linIni, colFin, linFin);
```

- **Método desenharMoldura**: Desenha uma moldura retangular com bordas.
- Primeiro, limpa a área da moldura.

```
    // linhas horizontais
    for (int x = colIni; x <= colFin; x++)
    {
        Console.SetCursorPosition(x, linIni);
        Console.Write("-");
        Console.SetCursorPosition(x, linFin);
        Console.Write("-");
    }
```

- Desenha linhas horizontais superiores e inferiores com o caractere `-`.

```
    // linhas verticais
    for (int y = linIni; y <= linFin; y++)
    {
        Console.SetCursorPosition(colIni, y);
        Console.Write("|");
        Console.SetCursorPosition(colFin, y);
        Console.Write("|");
    }
```

- Desenha linhas verticais esquerda e direita com o caractere `|`.

```
    // cantos
    Console.SetCursorPosition(colIni, linIni); Console.Write("+");
    Console.SetCursorPosition(colIni, linFin); Console.Write("+");
    Console.SetCursorPosition(colFin, linIni); Console.Write("+");
    Console.SetCursorPosition(colFin, linFin); Console.Write("+");
}
```

- Desenha os cantos da moldura com o caractere `+`.

```
public void centralizar(string texto, int lin = 0, int colIni = 0, int colFin = 0)
{
    if (lin == 0) lin = this.linhaMensagem;
    if (colIni == 0) colIni = this.colunaIniMensagem;
    if (colFin == 0) colFin = this.colunaFinMensagem;
```

- **Método centralizar**: Centraliza um texto em uma linha específica.
- Define valores padrão para linha e colunas se não forem fornecidos.

```
    this.limparArea(colIni+1, lin, colFin-1, lin);
    int colTexto = ((colFin - colIni - texto.Length) / 2) + colIni;
    Console.SetCursorPosition(colTexto, lin);
    Console.Write(texto);
}
```

- Limpa a área onde o texto será exibido.
- Calcula a posição inicial para centralizar o texto.
- Posiciona o cursor e escreve o texto.

```
public string mostrarMenu(List<string> opcoes, int colIni, int linIni)
{
    string opcaoEscolhida = "";
```

- **Método mostrarMenu**: Exibe um menu com opções e captura a escolha do usuário.

```
    // procura pela maior largura entre as opções do menu
    int largura = 0;
    foreach (string opcao in opcoes)
    {
        if (opcao.Length > largura)
        {
            largura = opcao.Length;
        }
    }
```

- Determina a largura necessária para a moldura do menu com base na opção mais longa.

```
// define a largura e altura para a moldura do menu
int colFin = colIni + largura + 1;
int linFin = linIni + opcoes.Count + 3;
this.desenharMoldura(colIni, linIni, colFin, linFin);
```

- Calcula as dimensões finais da moldura do menu.
- Desenha a moldura.

```
linIni++;
this.centralizar("Menu",linIni,colIni,colFin);
```

- Centraliza o título "Menu" na linha seguinte.

```
// mostra as opções do menu
colIni++;
linIni++;
for (int i = 0; i < opcoes.Count; i++)
{
    Console.SetCursorPosition(colIni, linIni);
    Console.WriteLine(opcoes[i]);
    linIni++;
}
```

- Exibe cada opção do menu em uma linha, incrementando a posição vertical.

```
// pergunta a opção escolhida
Console.SetCursorPosition(colIni, linIni);
Console.Write("Escolha uma opção: ");
opcaoEscolhida = Console.ReadLine();
```

- Solicita a escolha do usuário e armazena em `opcaoEscolhida`.

```
    // retorna a opção escolhida
    return opcaoEscolhida;
}
}
```

- Retorna a opção escolhida e fecha o método e a classe.

3. Arquivo: `BaseCRUD.cs`

A classe abstrata `BaseCRUD<T>` define a lógica genérica para operações CRUD, servindo como base para `ClienteCRUD`, `TemaCRUD` e outras classes.

```
using System;
using System.Collections.Generic;
```

- **Diretivas using**: Importam namespaces para console e coleções.

```
public abstract class BaseCRUD<T> where T : class, new()
{
```

- **Classe abstrata BaseCRUD**: Genérica, com restrição `T` para classes que podem ser instanciadas (`new()`).

```
protected List<T> lista; // "Banco de dados"
protected T registro; // Registro "da vez"
protected Tela tela;
```

```
protected int linha, coluna, linhaEntrada, colunaEntrada, larguraTotal, larguraDados, posicao;
protected List<string> campos;
```

- **Atributos protegidos**:
  - `lista` : Simula um banco de dados armazenando registros.
  - `registro` : Registro atual sendo manipulado.
  - `tela` : Objeto para gerenciar a interface.
  - `linha` , `coluna` , etc.: Controlam posições e dimensões na tela.
  - `campos` : Rótulos dos campos do registro.

```
public BaseCRUD()
{
    this.lista = new List<T>();
    this.tela = new Tela();
    this.larguraTotal = 40; // Valor padrão
    this.larguraDados = 0;
    this.coluna = 20;
    this.linha = 10;
    this.campos = new List<string>();
    this.InicializarCampos();
}
```

- **Construtor**: Inicializa a lista, a tela, dimensões padrão e chama `InicializarCampos` .

```
protected abstract void InicializarCampos(); // Define os rótulos dos campos
protected abstract void MontarTela(int coluna, int linha); // Monta a tela do CRUD
protected abstract void EntrarDados(int qual); // Entrada de dados (código ou demais campos)
protected abstract void MostrarDados(); // Exibe os dados do registro encontrado
```

```
protected abstract void AlterarRegistro(); // Altera os dados do registro
protected abstract int? GetCodigo(T registro); // Obtém o código do registro
```

- **Métodos abstratos**: Devem ser implementados pelas classes derivadas para personalizar o comportamento do CRUD.

```
public void ExecutarCRUD()
{
    while (true)
    {
```

- **Método ExecutarCRUD**: Controla o fluxo principal do CRUD em um loop contínuo.

```
        // 1 - Montar a tela do CRUD
        this.MontarTela(this.coluna, this.linha);
```

- Chama o método abstrato para desenhar a tela do CRUD.

```
        // 2 - Preparar um registro em branco
        this.registro = new T();
```

- Cria um novo registro vazio.

```
        // 3 - Perguntar ao usuário o código
        this.tela.centralizar("Deixe o campo vazio para sair.");
        this.EntrarDados(1);
        if (this.registro == null || this.GetCodigo(this.registro) == 0) break;
```

- Exibe instrução para sair.

- Solicita o código do registro (qual=1).
- Sai do loop se o registro for nulo ou o código for 0.

```
// 4 - Procurar pela chave no "banco de dados"
bool achou = this.BuscarCodigo(this.GetCodigo(this.registro).Value);
```

- Busca o código na lista de registros.

```
// 5 - Se não achou a chave
if (!achou)
{
    // 5.1 - Informar que não achou
    this.tela.centralizar("Registro não encontrado. Deseja cadastrar (S/N): ");
```

- Se o código não existe, exibe mensagem perguntando se deseja cadastrar.

```
    // 5.2 - Perguntar se deseja cadastrar
    string resp = Console.ReadLine();
```

- Captura a resposta do usuário.

```
    // 5.3 - Se o usuário deseja cadastrar
    if (resp.ToLower() == "s")
    {
        // 5.3.1 - Perguntar os dados restantes
        this.EntrarDados(2);
```

- Se a resposta for "s", solicita os demais dados (qual=2).

```
            // 5.3.2 - Confirmar cadastro
            this.tela.centralizar("Confirma cadastro (S/N) : ");
            resp = Console.ReadLine();
```

- Pede confirmação para o cadastro.

```
            // 5.3.3 - Se confirmado, incluir o registro
            if (resp.ToLower() == "s")
            {
                this.IncluirRegistro();
            }
        }
    }
```

- Se confirmado, inclui o registro na lista.

```
    // 6 - Se achou a chave
    else
    {
        // 6.1 - Mostrar os dados
        this.MostrarDados();
```

- Se o código existe, exibe os dados do registro.

```
        // 6.2 - Perguntar se deseja voltar, alterar ou excluir
        this.tela.centralizar("Deseja Voltar/Alterar/Excluir (V/A/E) : ");
        string resp = Console.ReadLine();
```

- Pergunta ao usuário a ação desejada.

```
        // 6.3 - Se deseja alterar
        if (resp.ToLower() == "a")
        {
            // 6.3.1 - Perguntar novos dados
            this.tela.centralizar("Digite apenas o dado que deseja alterar");
            this.EntrarDados(2);
```

- Se escolher alterar, solicita novos dados.

```
            // 6.3.2 - Confirmar alteração
            this.tela.centralizar("Confirma alteração (S/N) : ");
            resp = Console.ReadLine();
```

- Pede confirmação para a alteração.

```
            // 6.3.3 - Se confirmado, alterar o registro
            if (resp.ToLower() == "s")
            {
                this.AlterarRegistro();
            }
        }
```

- Se confirmado, altera o registro.

```
        // 6.4 - Se deseja excluir
        if (resp.ToLower() == "e")
        {
            // 6.4.1 - Confirmar exclusão
```

```
                this.tela.centralizar("Confirma exclusão (S/N) : ");
                resp = Console.ReadLine();
```

- Se escolher excluir, pede confirmação.

```
                // 6.4.2 - Se confirmado, excluir o registro
                if (resp.ToLower() == "s")
                {
                    this.ExcluirRegistro();
                }
            }
        }
    }
}
```

- Se confirmado, exclui o registro e continua o loop.

```
protected bool BuscarCodigo(int codigo)
{
    for (int i = 0; i < this.lista.Count; i++)
    {
        if (this.GetCodigo(this.lista[i]) == codigo)
        {
            this.posicao = i;
            return true;
        }
    }
    return false;
}
```

- **Método BuscarCodigo**: Procura um registro pelo código, atualiza `posicao` e retorna `true` se encontrado.

```
protected void IncluirRegistro()
{
    this.lista.Add(this.registro);
}
```

- **Método IncluirRegistro**: Adiciona o registro atual à lista.

```
protected void ExcluirRegistro()
{
    this.lista.RemoveAt(this.posicao);
}
```

- **Método ExcluirRegistro**: Remove o registro na posição especificada.

```
public List<T> ObterLista()
{
    return this.lista;
}
}
```

- **Método ObterLista**: Retorna a lista de registros.
- Fecha a classe.

4. Arquivo: `ClienteDTO.cs`

Define a estrutura de dados para um cliente.

```
public class ClienteDTO
{
```

- **Classe ClienteDTO**: Representa um cliente no sistema.

```
    // atributos
    private int? codigo;
    private string nome;
    private string email;
    private string telefone;
```

- **Atributos privados**: Armazenam o código (opcional), nome, email e telefone do cliente.

```
    // propriedades
    public int? Codigo { get => codigo; set => codigo = value; }
    public string Nome { get => nome; set => nome = value; }
    public string Email { get => email; set => email = value; }
    public string Telefone { get => telefone; set => telefone = value; }
```

- **Propriedades públicas**: Permitem acesso controlado aos atributos.

```
    public ClienteDTO(int codigo, string nome, string email, string telefone)
    {
        Codigo = codigo;
        Nome = nome;
        Email = email;
        Telefone = telefone;
    }
```

- **Construtor parametrizado**: Inicializa um cliente com valores fornecidos.

```
public ClienteDTO()
{
    Codigo = 0;
    Nome = "";
    Email = "";
    Telefone = "";
}
```

- **Construtor padrão**: Cria um cliente com valores padrão (vazios ou zero).

5. Arquivo: `ClienteCRUD.cs`

Implementa o CRUD para clientes, herdando de `BaseCRUD<ClienteDTO>`.

```
public class ClienteCRUD : BaseCRUD<ClienteDTO>
{
```

- **Classe ClienteCRUD**: Especializa `BaseCRUD` para gerenciar clientes.

```
protected override void InicializarCampos()
{
    this.campos.AddRange(new[] { "Código   :", "Nome     :", "Email    :", "Telefone :" });
    this.larguraDados = this.larguraTotal - this.campos[0].Length;
}
```

- **Método InicializarCampos**: Define os rótulos dos campos e calcula a largura disponível para dados.

```
protected override void MontarTela(int coluna, int linha)
{
    // Definir dimensões da moldura
    int coluna2 = coluna + this.larguraTotal;
    int linha2 = linha + this.campos.Count + 2; // +2 para título e borda
```

- **Método MontarTela**: Configura as dimensões da moldura do CRUD.

```
    // Desenhar moldura
    this.tela.desenharMoldura(coluna, linha, coluna2, linha2);
```

- Desenha a moldura.

```
    // Exibir título
    linha++;
    this.tela.centralizar("Cadastro de Cliente", linha, coluna, coluna2);
```

- Centraliza o título "Cadastro de Cliente".

```
    // Exibir rótulos dos campos
    coluna++;
    linha++;
    this.colunaEntrada = coluna + 1 + this.campos[0].Length; // Define posição para entrada de dados
    this.linhaEntrada = linha; // Define linha inicial para entrada de dados
    for (int i = 0; i < this.campos.Count; i++)
    {
        Console.SetCursorPosition(coluna, linha + i);
        Console.Write(this.campos[i]);
```

```
    }
}
```

- Exibe os rótulos dos campos e define posições para entrada de dados.

```
protected override void EntrarDados(int qual)
{
    string entrada;
    if (qual == 1)
    {
        Console.SetCursorPosition(this.colunaEntrada, this.linhaEntrada);
        entrada = Console.ReadLine();
        if (entrada.Length > 0)
        {
            ((ClienteDTO)this.registro).Codigo = int.Parse(entrada);
        }
        else
        {
            this.registro = new ClienteDTO(); // Registro vazio para sair
        }
    }
```

- **Método EntrarDados (qual=1)**: Solicita o código do cliente.
- Se vazio, cria um registro vazio para sair.

```
    else // qual == 2
    {
        this.tela.limparArea(this.colunaEntrada, this.linhaEntrada + 1, this.colunaEntrada + 25, this.linhaEntrada + 3);
        for (int i = 1; i < this.campos.Count; i++)
        {
            Console.SetCursorPosition(this.colunaEntrada, this.linhaEntrada + i);
            entrada = Console.ReadLine();
```

```
            switch (i)
            {
                case 1: ((ClienteDTO)this.registro).Nome = entrada; break;
                case 2: ((ClienteDTO)this.registro).Email = entrada; break;
                case 3: ((ClienteDTO)this.registro).Telefone = entrada; break;
            }
        }
    }
}
```

- **Método EntrarDados (qual=2)**: Solicita nome, email e telefone, limpando a área de entrada.

```
protected override void MostrarDados()
{
    var cliente = (ClienteDTO)this.lista[this.posicao];
    Console.SetCursorPosition(this.colunaEntrada, this.linhaEntrada + 1);
    Console.Write(cliente.Nome);
    Console.SetCursorPosition(this.colunaEntrada, this.linhaEntrada + 2);
    Console.Write(cliente.Email);
    Console.SetCursorPosition(this.colunaEntrada, this.linhaEntrada + 3);
    Console.Write(cliente.Telefone);
}
```

- **Método MostrarDados**: Exibe os dados do cliente encontrado.

```
protected override void AlterarRegistro()
{
    var cliente = (ClienteDTO)this.registro;
    var clienteExistente = (ClienteDTO)this.lista[this.posicao];
    if (cliente.Nome != "") clienteExistente.Nome = cliente.Nome;
    if (cliente.Email != "") clienteExistente.Email = cliente.Email;
```

```
    if (cliente.Telefone != "") clienteExistente.Telefone = cliente.Telefone;
}
```

- **Método AlterarRegistro**: Atualiza apenas os campos não vazios do cliente existente.

```
protected override int? GetCodigo(ClienteDTO registro)
{
    return registro.Codigo;
}
```

- **Método GetCodigo**: Retorna o código do cliente.

```
public List<ClienteDTO> ObterListaClientes()
{
    return this.lista;
}
}
```

- **Método ObterListaClientes**: Retorna a lista de clientes para uso em `AluguelCRUD`.
- Fecha a classe.

6. Arquivo: `TemaDTO.cs`

Define a estrutura de dados para um tema de festa.

```
public class TemaDTO
{
```

- **Classe TemaDTO**: Representa um tema de festa.

```
// atributos
private int? codigo;
private string nome;
private string categoria;
private double valor;
```

- **Atributos privados**: Armazenam código, nome, email, disponibilidade e valor do tema.

```
// propriedades
public int? Codigo { get => codigo; set => codigo = value; }
public string Nome { get => nome; set => nome = value;
public string Categoria { get => categoria; set => categoria = value; }
public string Disponivel { get => disponivel; set => disponivel = value; }
public double Valor { get => valor; set; => valor = value; }
```

- **Propriedades públicas**: Permitem acesso aos atributos.

```
public TemaDTO(int codigo, string nome, string categoria, string disponivel, double valor)
{
    Codigo = codigo;
    Nome = nome;
    Categoria = categoria;
    Disponivel = disponivel;
    Valor = valor;
}
```

- **Construtor parametrizado**: Inicializa um tema com valores fornecidos.

```
    public TemaDTO()
    {
        Codigo = 0;
        Nome = "";
        Categoria = "";
        Disponivel = "";
        Valor = 0.0;
    }
}
```

- **Construtor padrão**: Cria um tema com valores padrão.

7. Arquivo: `TemaCRUD.cs`

Implementa o CRUD para temas, herdando de `BaseCRUD<TemaDTO>`.

```
using System;
using System.Collections.Generic;
```

- **Diretivas using**: Importam namespaces necessários.

```
public class TemaCRUD : BaseCRUD<TemaDTO>
{
```

- **Classe TemaCRUD**: Especializa `BaseCRUD` para gerenciar temas.

```
protected override void InicializarCampos()
{
    this.campos.AddRange(new[] { "Código    :", "Nome      :", "Categoria :", "Disponível:", "Valor     :" });
    this.larguraDados = this.larguraTotal - this.campos[0].Length;
}
```

- **Método InicializarCampos**: Define rótulos e calcula largura dos dados.

```
protected override void MontarTela(int coluna, int linha)
{
    // Definir dimensões da moldura
    int coluna2 = coluna + this.larguraTotal;
    int linha2 = linha + this.campos.Count + 2; // +2 para título e borda
```

- **Método MontarTela**: Configura a moldura do CRUD.

```
    // Desenhar moldura
    this.tela.desenharMoldura(coluna, linha, coluna2, linha2);
```

- Desenha a moldura.

```
    // Exibir título
    linha++;
    this.tela.centralizar("Cadastro de Tema", linha, coluna, coluna2);
```

- Centraliza o título "Cadastro de Tema".

```
    // Exibir rótulos dos campos
    coluna++;
    linha++;
    this.colunaEntrada = coluna + 1 + this.campos[0].Length; // Define posição para entrada de dados
    this.linhaEntrada = linha; // Define linha inicial para entrada de dados
    for (int i = 0; i < this.campos.Count; i++)
    {
        Console.SetCursorPosition(coluna, linha + i);
        Console.Write(this.campos[i]);
    }
}
```

- Exibe os rótulos e define posições de entrada.

```
protected override void EntrarDados(int qual)
{
    string entrada;
    if (qual == 1)
    {
        Console.SetCursorPosition(this.colunaEntrada, this.linhaEntrada);
        entrada = Console.ReadLine();
        if (entrada.Length > 0)
        {
            ((TemaDTO)this.registro).Codigo = int.Parse(entrada);
        }
        else
        {
            this.registro = new TemaDTO(); // Registro vazio para sair
        }
    }
```

- **Método EntrarDados (qual=1)**: Solicita o código do tema.

```
    else // qual == 2
    {
        this.tela.limparArea(this.colunaEntrada, this.linhaEntrada + 1, this.colunaEntrada + 25, this.linhaEntrada + 4);
        for (int i = 1; i < this.campos.Count; i++)
        {
            Console.SetCursorPosition(this.colunaEntrada, this.linhaEntrada + i);
            entrada = Console.ReadLine();
            switch (i)
            {
                case 1: ((TemaDTO)this.registro).Nome = entrada; break;
                case 2: ((TemaDTO)this.registro).Categoria = entrada; break;
                case 3: ((TemaDTO)this.registro).Disponivel = entrada; break;
                case 4: ((TemaDTO)this.registro).Valor = entrada.Length > 0 ? double.Parse(entrada) : 0.0; break;
            }
        }
    }
}
```

- **Método EntrarDados (qual=2)**: Solicita nome, categoria, disponibilidade e valor.

```
protected override void MostrarDados()
{
    var tema = (TemaDTO)this.lista[this.posicao];
    Console.SetCursorPosition(this.colunaEntrada, this.linhaEntrada + 1);
    Console.Write(tema.Nome);
    Console.SetCursorPosition(this.colunaEntrada, this.linhaEntrada + 2);
    Console.Write(tema.Categoria);
    Console.SetCursorPosition(this.colunaEntrada, this.linhaEntrada + 3);
    Console.Write(tema.Disponivel);
    Console.SetCursorPosition(this.colunaEntrada, this.linhaEntrada + 4);
    Console.Write(tema.Valor.ToString("F2"));
}
```

- **Método MostrarDados**: Exibe os dados do tema encontrado.

```
protected override void AlterarRegistro()
{
    var tema = (TemaDTO)this.registro;
    var temaExistente = (TemaDTO)this.lista[this.posicao];
    if (temaTema.Nome != "") temaExistente.Nome = tema.Nome;
    if (tema.Categoria != "") temaExistente.Categoria = tema.Categoria;
    if (tema.Disponivel != "") temaExistente.Disponivel = tema.Disponivel;
    if (tema.Valor != 0.0) temaExistente.Valor = tema.Valor;
}
```

- **Método AlterarRegistro**: Atualiza apenas os campos não vazios.

```
protected override int? GetCodigo(TemaDTO registro)
{
    return registro.Codigo;
}
```

- **Método GetCodigo**: Retorna o código do tema.

```
public List<TemaDTO> ObterListaTemas()
{
    return this.lista;
}
}
```

- **Método ObterListaTemas**: Retorna a lista de temas.
- Fecha a classe.

8. Arquivo: `AluguelDTO.cs`

Define a estrutura de dados para um aluguel.

```
using System;
```

- **Diretiva using**: Importa `System` para `DateTime`.

```
public class AluguelDTO
{
```

- **Classe AluguelDTO**: Representa um aluguel.

```
    public int? Codigo { get; set; }
    public ClienteDTO Cliente { get; set; }
    public TemaDTO Tema { get; set; }
    public DateTime DataFesta { get; set; }
    public string LocalFesta { get; set; }
    public string HoraInicio { get; set; }
    public string HoraEncerramento { get; set; }
    public DateTime DataEmprestimo { get; set; }
    public DateTime DataDevolucao { get; set; }
    public double ValorAluguel { get; private set; }
```

- **Propriedades públicas**: Armazenam informações do aluguel, com `ValorAluguel` tendo setter privado.

```
    public AluguelDTO(
        int codigo,
        ClienteDTO cliente,
```

```
        TemaDTO tema,
        DateTime dataFesta,
        string localFesta,
        string horaInicio,
        string horaEncerramento,
        DateTime dataEmprestimo,
        DateTime dataDevolucao)
    {
        Codigo = codigo;
        Cliente = cliente;
        Tema = tema;
        DataFesta = dataFesta;
        LocalFesta = localFesta;
        HoraInicio = horaInicio;
        HoraEncerramento = horaEncerramento;
        DataEmprestimo = dataEmprestimo;
        DataDevolucao = dataDevolucao;
        ValorAluguel = CalcularValorAluguel();
    }
```

- **Construtor parametrizado**: Inicializa um aluguel e calcula o valor com `CalcularValorAluguel`.

```
    private double CalcularValorAluguel()
    {
        TimeSpan diferenca = DataDevolucao - DataEmprestimo;
        int dias = diferenca.Days;
        if (dias < 1)
            dias = 1;
        return dias * Tema.Valor;
    }
}
```

- **Método CalcularValorAluguel**: Calcula o valor do aluguel com base na diferença de dias entre empréstimo e devolução, multiplicada pelo valor do tema.
- Garante pelo menos 1 dia de cobrança.
- Fecha a classe.

9. Arquivo: `AluguelCRUD.cs`

Gerencia operações de aluguéis, incluindo registro, devolução e relatórios.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

- **Diretivas using**: Importam namespaces necessários.

```
public class AluguelCRUD
{
```

- **Classe AluguelCRUD**: Gerencia aluguéis, mas não herda de `BaseCRUD` devido a requisitos específicos.

```
    // atributos
    private List<AluguelDTO> listaAlugueis; // "Banco de dados"
    private AluguelDTO aluguel;             // Aluguel "da vez"
    private ClienteCRUD clienteCRUD;        // Para validar clientes
    private TemaCRUD temaCRUD;             // Para validar temas e atualizar disponibilidade
    private Tela tela;
```

```
private int linha, coluna;              // Linha e coluna da tela para desenhar a moldura
private int linhaEntrada, colunaEntrada; // Linha e coluna inicial para entrada de dados
private int larguraTotal, larguraDados; // Largura total da tela e largura dos dados
private List<string> campos;            // Lista de rótulos dos campos
```

- **Atributos privados**: Armazenam a lista de aluguéis, aluguel atual, dependências, configurações de tela e rótulos.

```
public AluguelCRUD(ClienteCRUD clienteCRUD, TemaCRUD temaCRUD)
{
    this.listaAlugueis = new List<AluguelDTO>();
    this.aluguel = null;
    this.clienteCRUD = clienteCRUD;
    this.temaCRUD = temaCRUD;
    this.tela = new Tela();
    this.campos = new List<string> {
        "Código       :",
        "Cod. Cliente :",
        "Cod. Tema    :",
        "Data Festa   :",
        "Local Festa  :",
        "Hora Início  :",
        "Hora Fim     :",
        "Data Emprést.:",
        "Data Devolução:"
    };
    this.larguraTotal = 60; // Aumentado para acomodar nomes
    this.larguraDados = this.larguraTotal - this.campos[0].Length;
    this.coluna = 20;
    this.linha = 10;
}
```

- **Construtor**: Inicializa os atributos, incluindo a lista de campos e dimensões da tela.

```
private void entrarDados(int qual)
{
    if (qual == 1)
    {
        // Entrada de código
        Console.SetCursorPosition(colunaEntrada, linhaEntrada);
        string entrada = Console.ReadLine();
        if (entrada.Length == 0)
        {
            this.aluguel = null;
            return;
        }
        this.aluguel = new AluguelDTO(
            int.Parse(entrada),
            new ClienteDTO(), // Placeholder
            new TemaDTO(),    // Placeholder
            DateTime.MinValue,
            "",
            "",
            "",
            DateTime.MinValue,
            DateTime.MinValue
        );
    }
```

- **Método entrarDados (qual=1)**: Solicita o código do aluguel.
- Se vazio, define `aluguel` como nulo.
- Caso contrário, cria um `AluguelDTO` com placeholders.

```
    else // qual == 2
    {
```

```
// Limpar área dos campos (exceto código)
this.tela.limparArea(colunaEntrada, linhaEntrada + 1, colunaEntrada + 45, linhaEntrada + campos.Count - 1);
```

- **Método entrarDados (qual=2)**: Limpa a área para entrada dos demais campos.

```
// Entrada para os demais campos
for (int i = 1; i < campos.Count; i++)
{
    bool entradaValida = false;
    while (!entradaValida)
    {
        Console.SetCursorPosition(colunaEntrada, linhaEntrada + i);
        string entrada = Console.ReadLine();
        try
        {
            switch (i)
            {
                case 1: // Cod. Cliente
                    int codCliente = entrada.Length > 0 ? int.Parse(entrada) : 0;
                    this.aluguel.Cliente = this.buscarCliente(codCliente);
                    if (this.aluguel.Cliente != null)
                    {
                        // Exibir nome do cliente ao lado
                        Console.SetCursorPosition(colunaEntrada + 10, linhaEntrada + i);
                        Console.Write(this.aluguel.Cliente.Nome);
                        entradaValida = true;
                        this.tela.centralizar(""); // Limpar rodapé
                    }
                    else
                    {
                        this.tela.centralizar("Código inválido!");
                    }
                    break;
```

- Solicita o código do cliente, busca o cliente e exibe seu nome se encontrado.
- Exibe erro se inválido.

```
case 2: // Cod. Tema
    int codTema = entrada.Length > 0 ? int.Parse(entrada) : 0;
    this.aluguel.Tema = this.buscarTema(codTema);
    if (this.aluguel.Tema != null)
    {
        // Exibir nome do tema ao lado
        Console.SetCursorPosition(colunaEntrada + 10, linhaEntrada + i);
        Console.Write(this.aluguel.Tema.Nome);
        entradaValida = true;
        this.tela.centralizar(""); // Limpa rodapé
    }
    else
    {
        this.tela.centralizar("Código inválido!");
    }
    break;
```

- Solicita o código do tema, busca o tema e exibe seu nome se encontrado.

```
case 3: // DataFesta
    this.aluguel.DataFesta = entrada.Length > 0 ? DateTime.Parse(entrada) : DateTime.MinValue;
    entradaValida = true;
    break;
case 4: // LocalFesta
    this.aluguel.LocalFesta = entrada;
    entradaValida = true;
    break;
case 5: // HoraInicio
    this.aluguel.HoraInicio = entrada.Length > 0 ? entrada : "";
```

```
                            entradaValida = true;
                            break;
                        case 6: // HoraEncerramento
                            this.aluguel.HoraEncerramento = entrada.Length > 0 ? entrada : "";
                            entradaValida = true;
                            break;
                        case 7: // DataEmprestimo
                            this.aluguel.DataEmprestimo = entrada.Length > 0 ? DateTime.Parse(entrada) : DateTime.MinValue;
                            entradaValida = true;
                            break;
                        case 8: // DataDevolucao
                            this.aluguel.DataDevolucao = entrada.Length > 0 ? DateTime.Parse(entrada) : DateTime.MinValue;
                            entradaValida = true;
                            break;
                    }
                }
                catch (FormatException)
                {
                    if (i <= 2) // Cliente ou Tema
                        this.tela.centralizar("Código inválido!");
                    else
                        entradaValida = true; // Erros tratados em RegistrarAluguel
                }
            }
        }
    }
}
```

- Solicita os demais campos, tratando erros de formato.

```
private void montarTelaAluguel(int coluna, int linha)
{
    int coluna2 = coluna + this.larguraTotal;
    int linha2 = linha + campos.Count;
```

```
    this.tela.desenharMoldura(coluna, linha, coluna2, linha2);
    linha++;
    this.tela.centralizar("Registrar Aluguéis", linha, coluna, coluna2);
    coluna++;
    linha++;
    this.colunaEntrada = coluna + 1 + campos[0].Length;
    this.linhaEntrada = linha;
    for (int i = 0; i < campos.Count; i++)
    {
        Console.SetCursorPosition(coluna, linha + i);
        Console.Write(campos[i]);
    }
}
```

- **Método montarTelaAluguel**: Desenha a tela para registro de aluguéis.

```
private void montarTelaDevolucao(int coluna, int linha)
{
    int coluna2 = coluna + this.larguraTotal;
    int linha2 = linha + campos.Count;
    this.tela.desenharMoldura(coluna, linha, coluna2, linha2);
    linha++;
    this.tela.centralizar("Registrar Devolução", linha, coluna, coluna2);
    coluna++;
    linha++;
    this.colunaEntrada = coluna + 1 + campos[0].Length;
    this.linhaEntrada = linha;
    Console.SetCursorPosition(coluna, linha);
    Console.Write(campos[0]); // Código
}
```

- **Método montarTelaDevolucao**: Desenha a tela para registro de devoluções.

```
public void RegistrarAluguel()
{
    this.montarTelaAluguel(this.coluna, this.linha);
    this.aluguel = null;
    this.tela.centralizar("Deixe o campo vazio para sair.");
    this.entrarDados(1);
    if (this.aluguel == null || !this.aluguel.Codigo.HasValue) return;
```

- **Método RegistrarAluguel**: Inicia o registro de um aluguel.
- Monta a tela, solicita o código e sai se nulo.

```
    if (this.buscarCodigo(this.aluguel.Codigo.Value))
    {
        this.tela.centralizar("Código de aluguel já existe!");
        Console.ReadLine();
        return;
    }
```

- Verifica se o código já existe.

```
    try
    {
        this.entrarDados(2);
        if (this.aluguel.Cliente == null)
        {
            this.tela.centralizar("Cliente não encontrado!");
            Console.ReadLine();
            return;
        }
        if (this.aluguel.Tema == null)
        {
```

```
            this.tela.centralizar("Tema não encontrado!");
            Console.ReadLine();
            return;
        }
        if (this.aluguel.Tema.Disponivel.ToLower() != "s")
        {
            this.tela.centralizar("Tema não está disponível!");
            Console.ReadLine();
            return;
        }
        if (this.aluguel.DataEmprestimo > this.aluguel.DataDevolucao)
        {
            this.tela.centralizar("Data de empréstimo não pode ser posterior à devolução!");
            Console.ReadLine();
            return;
        }
```

- Solicita os demais dados e valida cliente, tema, disponibilidade e datas.

```
        this.tela.centralizar("Confirma cadastro (S/N) : ");
        string resp = Console.ReadLine();
        if (resp.ToLower() == "s")
        {
            this.listaAlugueis.Add(this.aluguel);
            this.aluguel.Tema.Disponivel = "N";
            this.tela.centralizar("Aluguel registrado com sucesso!");
            Console.ReadLine();
        }
    }
    catch (FormatException)
    {
        this.tela.centralizar("Entrada inválida! Verifique os formatos.");
        Console.ReadLine();
```

```
    }
}
```

- Confirma o cadastro, adiciona o aluguel, marca o tema como indisponível e exibe mensagem de sucesso.

```
public void RegistrarDevolucao()
{
    this.montarTelaDevolucao(this.coluna, this.linha);
    this.aluguel = null;
    this.tela.centralizar("Deixe o campo vazio para sair.");
    this.entrarDados(1);
    if (this.aluguel == null || !this.aluguel.Codigo.HasValue) return;
```

- **Método RegistrarDevolucao**: Inicia o registro de uma devolução.

```
    try
    {
        int posicao = -1;
        AluguelDTO aluguelEncontrado = null;
        for (int i = 0; i < this.listaAlugueis.Count; i++)
        {
            if (this.listaAlugueis[i].Codigo == this.aluguel.Codigo.Value)
            {
                posicao = i;
                aluguelEncontrado = this.listaAlugueis[i];
                break;
            }
        }
        if (posicao == -1)
        {
            this.tela.centralizar("Aluguel não encontrado!");
            Console.ReadLine();
```

```
            return;
        }
```

- Busca o aluguel pelo código.

```
        this.mostrarDadosAluguel(aluguelEncontrado);
        this.tela.centralizar("Confirma devolução (S/N) : ");
        string resp = Console.ReadLine();
        if (resp.ToLower() == "s")
        {
            aluguelEncontrado.DataDevolucao = DateTime.Now.Date;
            aluguelEncontrado.Tema.Disponivel = "S";
            this.tela.centralizar("Devolução registrada com sucesso!");
            Console.ReadLine();
        }
    }
    catch (FormatException)
    {
        this.tela.centralizar("Código inválido!");
        Console.ReadLine();
    }
}
```

- Exibe os dados, confirma a devolução, atualiza a data e marca o tema como disponível.

```
private bool buscarCodigo(int codigo)
{
    return this.listaAlugueis.Exists(a => a.Codigo == codigo);
}
```

- **Método buscarCodigo**: Verifica se um código de aluguel existe.

```
private ClienteDTO buscarCliente(int codigo)
{
    foreach (var cliente in this.clienteCRUD.ObterListaClientes())
    {
        if (cliente.Codigo == codigo)
            return cliente;
    }
    return null;
}
```

- **Método buscarCliente**: Busca um cliente pelo código.

```
private TemaDTO buscarTema(int codigo)
{
    foreach (var tema in this.temaCRUD.ObterListaTemas())
    {
        if (tema.Codigo == codigo)
            return tema;
    }
    return null;
}
```

- **Método buscarTema**: Busca um tema pelo código.

```
private void mostrarDadosAluguel(AluguelDTO aluguel)
{
    this.tela.limparArea(colunaEntrada, linhaEntrada + 1, colunaEntrada + 45, linhaEntrada + campos.Count - 1);
    Console.SetCursorPosition(colunaEntrada, linhaEntrada + 1);
    Console.Write(aluguel.Cliente.Codigo);
    Console.SetCursorPosition(colunaEntrada + 10, linhaEntrada + 1);
    Console.Write(aluguel.Cliente.Nome);
```

```
    // ... (demais campos)
}
```

- **Método mostrarDadosAluguel**: Exibe os dados de um aluguel.

```
public List<AluguelDTO> GetListaAlugueis()
{
    return this.listaAlugueis;
}
```

- **Método GetListaAlugueis**: Retorna a lista de aluguéis.

```
public void MostrarRelatorio()
{
    this.montarTelaRelatorio(1, 5);
    this.tela.centralizar("Mostrar aluguéis Ativos/Finalizados (A/F) : ");
    string tipo = Console.ReadLine().ToLower();
    if (tipo != "a" && tipo != "f")
    {
        this.tela.centralizar("Opção inválida!");
        Console.ReadLine();
        return;
    }
```

- **Método MostrarRelatorio**: Monta a tela e solicita o tipo de relatório (ativos ou finalizados).

```
    var hoje = DateTime.Now.Date;
    var alugueisFiltrados = tipo == "a"
        ? this.listaAlugueis.FindAll(a => a.DataDevolucao >= hoje)
        : this.listaAlugueis.FindAll(a => a.DataDevolucao < hoje);
```

- Filtra aluguéis com base na data de devolução.

```
    this.tela.limparArea(colunaEntrada, linhaEntrada, colunaEntrada + 50, linhaEntrada + 10);
    int linhaAtual = linhaEntrada;
    Console.SetCursorPosition(colunaEntrada, linhaAtual);
    Console.WriteLine($"{"Código",-8} {"Cliente",-20} {"Tema",-20} {"Data Festa",-12} {"Valor",10}");
    linhaAtual++;
    foreach (var aluguel in alugueisFiltrados)
    {
        Console.SetCursorPosition(colunaEntrada, linhaAtual);
        Console.WriteLine(
            $"{aluguel.Codigo,-8} " +
            $"{aluguel.Cliente.Nome,-20} " +
            $"{aluguel.Tema.Nome,-20} " +
            $"{aluguel.DataFesta:dd/MM/yyyy,-12} " +
            $"{aluguel.ValorAluguel:F2,10}"
        );
        linhaAtual++;
    }
```

- Exibe o relatório em formato tabular.

```
    if (alugueisFiltrados.Count == 0)
    {
        this.tela.centralizar("Nenhum aluguel encontrado!");
    }
    this.tela.centralizar("Pressione qualquer tecla para continuar...");
    Console.ReadLine();
}
```

- Exibe mensagem se não houver aluguéis e aguarda interação.

```
private void montarTelaRelatorio(int coluna, int linha)
{
    int coluna2 = coluna + 70; // Maior para suportar a tabela
    int linha2 = linha + 12;   // Espaço para até 10 aluguéis
    this.tela.desenharMoldura(coluna, linha, coluna2, linha2);
    linha++;
    this.tela.centralizar("Relatório de Aluguéis", linha, coluna, coluna2);
    coluna++;
    linha++;
    this.colunaEntrada = coluna;
    this.linhaEntrada = linha;
}
}
```

- **Método montarTelaRelatorio**: Desenha a tela para o relatório.
- Fecha a classe.

Funcionalidades Disponibilizadas

1. **Gerenciamento de Clientes**:
    - Cadastro, alteração e exclusão de clientes via `ClienteCRUD`.
    - Interface com campos para código, nome, email e telefone.
2. **Gerenciamento de Temas**:
    - Cadastro, alteração e exclusão de temas via `TemaCRUD`.
    - Campos incluem código, nome, categoria, disponibilidade e valor.

3. **Registro de Aluguéis**:
    - Criação de aluguéis via `AluguelCRUD`.
    - Valida cliente, tema e datas, marcando o tema como indisponível.
4. **Registro de Devoluções**:
    - Atualiza a data de devolução e marca o tema como disponível.
5. **Relatórios**:
    - Exibe aluguéis ativos (data de devolução futura) ou finalizados (data passada).
    - Formato tabular com código, cliente, tema, data da festa e valor.

Conceitos de Orientação a Objetos aplicados