Projeto de Software

GCT088 - Aula 1.2 - Conceitos de Orientação a Objetos

Bento Rafael Siqueira

Universidade Federal de Lavras (UFLA)

2 de setembro de 2025

Sumário

Conceitos de Orientação a Objetos:

- Abstração
- Classe e Relacionamentos
- Objetos e Construtores
- Encapsulamento
- Sobrecarga
- Herança
- Polimorfismo
- Atividade Integrada

O que é Abstração?

Definição:

- **Abstração** é o processo de simplificar sistemas complexos
- Foca nos aspectos essenciais, ignorando detalhes irrelevantes
- Permite representar conceitos do mundo real de forma simplificada
- Base para todos os outros conceitos de POO

Exemplo Prático:

- **Carro:** Motor, rodas, volante, freios (essencial)
- **Ignora:** Composição química do plástico, detalhes do motor
- **Resultado:** Modelo simplificado e funcional

Exemplo Prático: Sistema de Viagens

Abstração de um Destino:

- **Incluído:** Nome, país, cidade, clima, disponibilidade
- **Omitido:** Coordenadas geográficas, população, história detalhada
- **Foco:** Funcionalidades essenciais para o sistema

Arquivo de Exemplo:

- `Exemplos/Abstracao.cs`
- Classe `Destino` com propriedades essenciais
- Métodos para reserva e verificação de disponibilidade

O que é uma Classe?

Definição:

- **Classe** é um modelo/template para criar objetos
- Define atributos (propriedades) e comportamentos (métodos)
- Representa um conceito abstrato do mundo real
- Serve como "planta"para instanciar objetos

Componentes:

- **Atributos:** Características do objeto
- **Métodos:** Comportamentos do objeto
- **Construtores:** Inicialização do objeto
- **Relacionamentos:** Como se conecta com outras classes

Relacionamentos entre Classes

Três tipos principais:

1. **Associação:**
   - Relacionamento simples entre classes
   - Uma classe "usa"ou "conhece"outra
   - Independência entre os objetos
2. **Composição:**
   - Relacionamento "parte-de"forte
   - Objeto parte não existe sem o todo
   - Ciclo de vida dependente
3. **Agregação:**
   - Relacionamento "parte-de"fraco
   - Objeto parte pode existir independentemente
   - Ciclo de vida independente

Exemplo Prático: Sistema de Agência de Viagens

Arquivo de Exemplo:

- `Exemplos/Relacionamentos.cs`
- **Associação:** Viajante "usa"Destino
- **Composição:** Agência "contém"Departamentos
- **Agregação:** Departamento "tem"Funcionários

Características:

- Viajante pode existir sem destino específico
- Departamento não existe sem Agência
- Funcionário pode existir sem Departamento

O que é um Objeto?

Definição:

- **Objeto** é uma instância de uma classe
- Representa um exemplo específico do conceito
- Possui estado (valores dos atributos) e comportamento
- Existe em memória durante a execução do programa

Características:

- **Identidade:** Cada objeto é único
- **Estado:** Valores atuais dos atributos
- **Comportamento:** Métodos que pode executar
- **Ciclo de vida:** Criação, uso e destruição

Construtores

Definição:

- **Construtor** é um método especial para inicializar objetos
- Chamado automaticamente quando o objeto é criado
- Mesmo nome da classe
- Pode ter parâmetros para configurar o estado inicial

Tipos de Construtores:

- **Construtor Padrão:** Sem parâmetros
- **Construtor Parametrizado:** Com parâmetros
- **Construtor de Cópia:** Copia outro objeto

Exemplo Prático: Sistema de Pacotes de Viagem

Arquivo de Exemplo:

- `Exemplos/ObjetosConstrutores.cs`
- Classe `PacoteViagem` com múltiplos construtores
- Demonstração de sobrecarga de construtores

Construtores Implementados:

- **Padrão:** Valores padrão para todos os atributos
- **Parametrizado:** Código e nome do pacote
- **Completo:** Todos os atributos configuráveis

O que é Encapsulamento?

Definição:

- **Encapsulamento** é o princípio de ocultar detalhes internos
- Agrupa dados e métodos que operam sobre esses dados
- Controla o acesso aos dados através de interfaces
- Protege a integridade dos dados

Benefícios:

- **Segurança:** Dados protegidos contra acesso indevido
- **Flexibilidade:** Implementação pode mudar sem afetar usuários
- **Manutenibilidade:** Mudanças localizadas
- **Reutilização:** Código mais modular

Exemplo Prático: Controle de Reservas

Arquivo de Exemplo:

- `Exemplos/Encapsulamento.cs`
- Classe `ReservaViagem` com encapsulamento
- Atributos privados com controle de acesso

Características:

- **Atributos privados:** valorTotal, senhaAcesso, confirmada
- **Propriedades públicas:** Controle de leitura/escrita
- **Métodos públicos:** Operações seguras
- **Validação:** Verificação de senha para operações críticas

O que é Sobrecarga de Métodos?

Definição:

- **Sobrecarga** permite múltiplas versões do mesmo método
- Métodos com mesmo nome mas parâmetros diferentes
- Compilador escolhe a versão adequada baseado nos argumentos
- Aumenta a flexibilidade e legibilidade do código

Regras:

- **Nome igual** para todos os métodos
- **Parâmetros diferentes** (tipo, quantidade, ordem)
- **Retorno pode ser igual ou diferente**
- **Modificadores de acesso** podem variar

Exemplo Prático: Calculadora de Viagens

Arquivo de Exemplo:

- `Exemplos/Sobrecarga.cs`
- Classe `CalculadoraViagem` com sobrecarga
- Múltiplas versões de métodos de cálculo

Métodos Sobrecarregados:

- **CalcularPreco:** Diferentes combinações de parâmetros
- **AplicarDesconto:** Percentual ou tipo de cliente
- **ValidarDestino:** Validação simples ou com lista

O que é Herança?

Definição:

- **Herança** permite criar classes baseadas em outras
- Classe filha herda atributos e métodos da classe pai
- Promove reutilização de código
- Estabelece relacionamento "é-um"entre classes

Benefícios:

- **Reutilização:** Código da classe pai é reutilizado
- **Extensibilidade:** Novas funcionalidades podem ser adicionadas
- **Hierarquia:** Organização lógica das classes
- **Polimorfismo:** Base para polimorfismo

Exemplo Prático: Hierarquia de Transportes

Arquivo de Exemplo:

- `Exemplos/Heranca.cs`
- Classe base `Transporte`
- Classes filhas: `Aviao`, `Onibus`, `AviaoExecutivo`

Hierarquia Implementada:

- **Transporte:** Classe base com propriedades comuns
- **Aviao:** Herança simples com propriedades específicas
- **Onibus:** Herança simples com características próprias
- **AviaoExecutivo:** Herança múltipla de níveis

O que é Polimorfismo?

Definição:

- **Polimorfismo** significa "muitas formas"
- Permite que objetos de classes diferentes respondam ao mesmo método
- Método pode ter implementações diferentes em classes diferentes
- Aumenta flexibilidade e extensibilidade do código

Tipos:

- **Polimorfismo de Sobrescrita:** Método virtual/override
- **Polimorfismo de Sobrecarga:** Múltiplas versões do método
- **Polimorfismo de Interface:** Implementação de interfaces

Exemplo Prático: Sistema de Serviços de Viagem

Arquivo de Exemplo:

- `Exemplos/Polimorfismo.cs`
- Classe abstrata `ServicoViagem`
- Classes filhas: `Hospedagem`, `Passeio`, `Transfer`

Polimorfismo Demonstrado:

- **Métodos abstratos:** CalcularPrecoFinal(), ObterTipo()
- **Implementações específicas:** Cada classe filha implementa diferentemente
- **Uso polimórfico:** Lista de ServicoViagem com objetos diferentes

Sistema de Gestão de Viagens

Desafio: Implementar um sistema completo de agência de viagens usando todos os conceitos aprendidos.

Requisitos:

- **Abstração:** Modelar destinos, clientes, reservas
- **Classes e Relacionamentos:** Associação, composição, agregação
- **Objetos e Construtores:** Criar instâncias com diferentes construtores
- **Encapsulamento:** Proteger dados sensíveis
- **Sobrecarga:** Múltiplas formas de criar objetos
- **Herança:** Hierarquia de serviços e clientes
- **Polimorfismo:** Diferentes tipos de serviços

Arquivo de Exemplo Integrado

Arquivo Principal:

- `Exemplos/SistemaViagem.cs`
- Sistema completo demonstrando todos os conceitos
- Classe `SistemaViagem` com método `DemonstrarSistema()`

Conceitos Aplicados:

- **Abstração:** ServicoBase como conceito abstrato
- **Herança:** Hospedagem e Transporte herdam de ServicoBase
- **Polimorfismo:** ExibirDetalhes() e ObterTipo() implementados diferentemente
- **Relacionamentos:** Associação, composição e agregação
- **Encapsulamento:** Atributos protegidos e controle de acesso

Conceitos Aplicados na Atividade

Verificação dos Conceitos:

✓ **Abstração:**
- ServicoBase como conceito abstrato
- Foco nas propriedades essenciais

✓ **Classe e Relacionamentos:**
- Associação: Cliente usa Agência
- Composição: Agência contém Departamentos
- Agregação: Agência tem Serviços

✓ **Objetos e Construtores:**
- Múltiplos construtores para Hospedagem
- Inicialização flexível de objetos

✓ **Encapsulamento:**
- Atributos protegidos em ServicoBase
- Controle de acesso aos dados

Conceitos Aplicados na Atividade (Continuação)

✓ **Sobrecarga de Métodos:**
- Múltiplos construtores em Hospedagem
- Diferentes formas de criar objetos

✓ **Herança:**
- Hospedagem e Transporte herdam de ServicoBase
- Reutilização de código comum

✓ **Polimorfismo:**
- ExibirDetalhes() com implementações diferentes
- ObterTipo() retorna tipos específicos
- Lista de ServicoBase contém objetos diferentes

Resultado: Sistema completo e funcional demonstrando todos os conceitos!

Resumo dos Conceitos

Conceitos Fundamentais de POO:
1. Abstração: Simplificar sistemas complexos **2. Classe:** Modelo para criar objetos **3. Objetos:** Instâncias com estado e comportamento **4. Encapsulamento:** Proteção e controle de acesso **5. Sobrecarga:** Múltiplas versões do mesmo método **6. Herança:** Reutilização e extensão de código **7. Polimorfismo:** Flexibilidade e extensibilidade
Benefícios:

- Código mais organizado e reutilizável
- Manutenção mais fácil
- Extensibilidade aumentada
- Modelagem mais próxima do mundo real

Próximos Passos

Análise e Projeto Orientado a Objetos:

- **Modelagem de Domínio:** Identificar entidades e relacionamentos
- **Diagramas UML:** Casos de uso, classes, sequência
- **Padrões de Análise:** GRASP e outros padrões
- **Refatoração:** Melhorar design de código existente
- **Padrões de Design:** GoF e outros padrões
- **Princípios SOLID:** Boas práticas de design
- **Arquitetura de Software:** Organização de sistemas

Prática Recomendada:

- Modelar o sistema de viagens com UML
- Aplicar padrões de design
- Refatorar código existente
- Implementar arquiteturas escaláveis

Obrigado!

Dúvidas? **Próxima aula:** Análise e Projeto Orientado a Objetos

Contato:

- **Professor:** Bento Rafael Siqueira
- **Disciplina:** GCT088 - Projeto de Software
- **Universidade:** UFLA