Programação Orientada a Objetos

Aula 01 — Introdução ao Paradigma

Hugo Marcondes
Departamento Acadêmico de Eletrônica
DAELN

hugo.marcondes@ifsc.edu.br

INSTITUTO FEDERAL
Santa Catarina
Câmpus Florianópolis

Notes

Introdução

Notes

Visão tradicional da programação

- Programas vistos como uma sequência de operações executadas por um computador
- Foco na automação de processos bem definidos (ex.: cálculos numéricos)
- Eficaz para sistemas pequenos e simples, mas limitado com o crescimento da complexidade dos sistemas

Notes

Problemas com a visão tradicional

- **Decomposição Funcional Top-Down**
  - Primeiro passo: Definir entradas e saídas do processo
  - Decomposição em módulos funcionais
- **Desafios**
  - Difícil de gerenciar mudanças em aplicações complexas
  - Incapacidade de reutilizar código eficientemente

Notes

Complexidade

Notes

Sistemas Complexos

- O projeto de um sistema orientado a objetos visa principalmente tratar o projeto de sistemas complexos

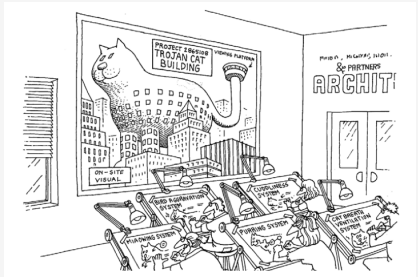
- As principais atributos de um sistema complexo são:
  - Estrutura Hierarquizada
  - Elementos primitivos relativos
  - Separação de responsabilidades (concerns)
  - Padrões comuns
  - Formas intermediárias estáveis

Notes

Estrutura Hierarquizada

"Frequently, complexity takes the **form of a hierarchy**, whereby a complex system is composed of **interrelated subsystems** that have in turn **their own subsystems**, and so on, until some lowest level of **elementary components** is reached."

Booch, 2007



The architecture of a complex system is a function of its components as well as the hierarchic relationships among these components.

Notes

Separação de Responsabilidades

"Intracomponent linkages are generally stronger than intercomponent linkages. This fact has the effect of **separating the high-frequency dynamics** of the components — involving the internal structure of the components — **from the low-frequency dynamics** — involving interaction among components."

- A diferença entre as **interações internas** e **externas** de componentes provêem uma **clara separação de responsabilidades** (separation of concerns) entre as várias partes do sistema.

Notes

Complexidade

- Elementos primitivos relativos
  - A escolha de quais os **componentes** em um sistema são **primitivos** é relativamente **arbitrário** e é em grande parte a critério do observador do sistema
- Padrões comuns
  - **Sistemas hierárquicos** são geralmente **compostas** por apenas **alguns tipos** diferentes de **sub-sistemas** em **várias combinações** e **arranjos**
- Formas intermediárias estáveis
  - Um **sistema complexo** que funciona é invariavelmente encontrado a partir da **evolução** de um **sistema simples funcional**. . . "Um sistema complexo projetado do zero nunca funciona e não pode ser remendado para que funcione. Você tem que começar de novo, começando com um sistema simples que funcione."

Notes

Engenharia de Software

- Surgimento do **Design de Software** e **Engenharia de Software**
  - Inspirado pela troca de ideias com outras áreas da engenharia
- Objetivo: Criar software reutilizável e adaptável usando componentes
  - Inspiração do design de sistemas eletromecânicos:
    - Soluções padronizadas, designs facilmente compreensíveis
    - Componentes intercambiáveis, reutilização de blocos de construção
  - Pergunta: O software pode ser construído de maneira semelhante, usando componentes prontos?

Notes

O papel da decomposição

> "The technique of mastering complexity has been known since ancient times: ***divide et impera*** (divide and rule)"

- No projeto de sistemas complexos é essencial realizar a decomposição do sistema em partes menores, cada qual pode ser refinada independentemente.
  - Decomposição algorítmica
  - Decomposição orientada a Objetos

Notes

Paradigma Orientado a Objetos

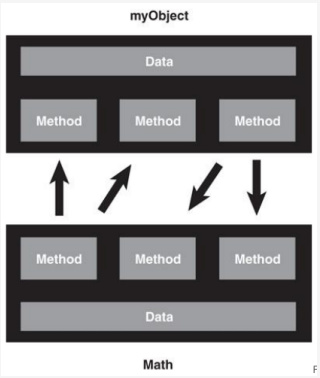
Notes

Paradigma Orientado a Objetos

"Object-oriented programming is a **method of implementation** in which programs are organized as **cooperative collections of objects**, each of which represents **an instance of some class**, and whose classes are all members of a **hierarchy of classes** united via **inheritance relationships**."

Booch, 2007

- Software definido como uma coleção de objetos que interagem entre si
- **Principais Características**:
  - Armazenam dados (atributos) e possuem comportamento (métodos)
  - **Comunicam-se** através de mensagens (métodos)
  - **Encapsulam** dados e métodos em uma única unidade
- **Contraste com o Design Tradicional**:
  - Ao invés de definir processos de ponta a ponta, o foco é identificar componentes que armazenam dados e executam procedimentos



Notes

Paradigma Orientado a Objetos

Notes

Projeto Orientada a Objetos

"Object-oriented design is a **method of design** encompassing the process of **object-oriented decomposition** and a notation for depicting both **logical** and **physical** as well as **static** and **dynamic models of the system** under design."

Booch, 2007

- Leva a decomposição orientada a objetos
- Por modelo lógico, entende-se a estrutura de classes e objetos
- Por modelo físico, entende-se a arquitetura modular e de processos do sistema

Notes

Análise Orientada a Objetos

"Object-oriented analysis is a **method of analysis** that examines **requirements** from the perspective of the **classes and objects** found in the **vocabulary** of the **problem domain**."

Booch, 2007

Notes

Os pilares da Orientação a Objetos

- A programação orientada por objetos é baseada em quatro pilares, conceitos que diferenciam ele de outros paradigmas de programação.
  - Abstração
  - Encapsulamento
  - Herança
  - Polimorfismo

Notes

Abstração

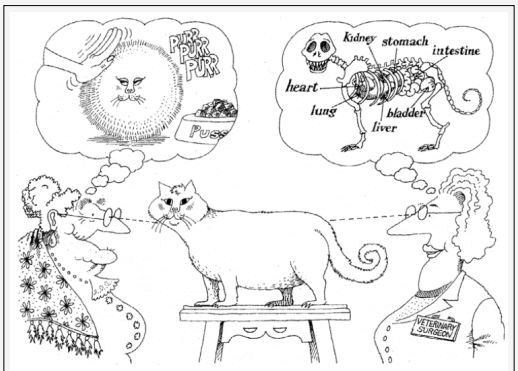
"An **abstraction** denotes the **essential characteristics** of an object that **distinguish** it from all **other** kinds of objects and thus **provide** crisply defined conceptual **boundaries**, relative to the **perspective of the viewer**."

Booch, 2007

- Uma **boa abstração** é aquela que **enfatiza** os detalhes que são **significativos** para o usuário e **suprime** os detalhes que, ao menos no determinado contexto, **não** são **relevantes**
- **Decidir** pelo conjunto **adequado** de abstrações para um domínio em específico é o **principal desafio** em um **projeto orientado a objetos**

Notes

Abstração



Abstraction focuses on the essential characteristics of some object, relative to the perspective of the viewer.

Notes

Abstração

- Um objeto é uma instância de uma determinada abstração no domínio do problema.
- Definem um contrato entre si
  - Operações que uma determinada abstração pode executar
  - Determina as assunções que podemos estabelecer em relação ao comportamento do objeto
  - Possuem propriedades estáticas e dinâmicas

Notes

Encapsulamento

Encapsulation is the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior; encapsulation serves to separate the contractual interface of an abstraction and its implementation

- Abstração e encapsulamento são conceitos complementares: Abstração foca no comportamento observável de um objetos, enquanto encapsulamento foca na implementação que emerge este comportamento.
- Encapsulamento provê barreiras explícitas entre diferentes abstrações e desta forma leva a uma clara separação de responsabilidades.
- "For abstraction to work, implementations must be encapsulated". Na prática, isto significa que cada classe deve ter duas partes: uma interface e uma implementação.

Notes

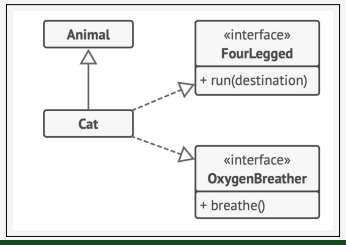
Encapsulamento

- O Encapsulamento é a habilidade de um objeto de esconder parte de seu estado e comportamentos de outros objetos, expondo apenas uma interface limitada para o resto do programa.
- Encapsular alguma coisa significa torná-la **privada**, e portanto acessível apenas por dentro dos métodos da sua própria classe. Há um modo um pouco menos restritivo chamado **protegido** que torna um membro da classe disponível para subclasses também.
- Interfaces e classes/métodos abstratos da maioria das linguagens de programação são baseados em conceitos de abstração e encapsulamento.

Notes

Herança

- A Herança é a habilidade de construir novas classes em cima de classes já existentes.
- Reutilização de código
- Estende a classe existente e coloca a funcionalidade adicional dentro de uma subclasse
- Hierarquia de classe — tipo "é uma"



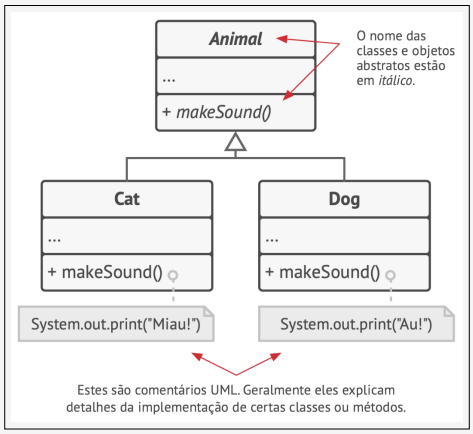
Notes

Polimorfismo

- O Polimorfismo é a habilidade de um programa detectar a classe real de um objeto e chamar sua implementação mesmo quando seu tipo real é desconhecido no contexto atual.
- Você também pode pensar no polimorfismo como a habilidade de um objeto "fingir" que é outra coisa, geralmente uma classe que ele estende ou uma interface que ele implementa.

Notes

Polimorfismo



Notes

Polimorfismo

```
1  bolsa = [new Gato(), new Cão()];
2
3  foreach (Animal a : bolsa)
4    a.produzirSom()
5
6  // Miau!
7  // Au!
```


Notes

Classes e Objetos

Notes

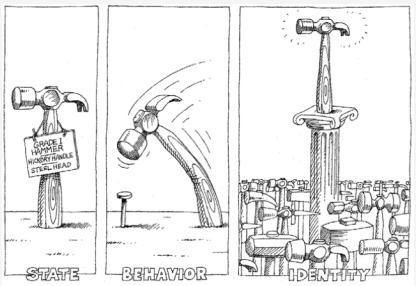
Objetos

- Da perspectiva da cognição humana, um objeto pode ser:
  - Uma coisa tangível/visível
  - Algo que pode ser compreendido intelectualmente
  - Algo em que uma ação pode ser direcionado
- Um objeto representa um indivíduo, uma unidade, item ou entidade identificável, seja real ou abstrata, com um papel bem definido no domínio do problema

Notes

Objetos

"An **object** is an entity that has **state**, **behavior**, and **identity**. The **structure** and **behavior** of **similar objects** are defined in their **common class**. The terms instance and object are interchangeable."



An object has state, exhibits some well-defined behavior, and has a unique identity.

Notes

Comportamento de objetos

- Nenhum objeto existe isoladamente.
- Comportamento é como um objeto age e reage, em termos de sua mudança de estado e passagem de mensagens
  - O comportamento de um objeto representa a sua atividade externa visível
- Uma operação é alguma ação que um objeto executa em outro de forma a provocar uma reação
  - Java — métodos
  - C++ — função membro
  - Smalltalk — mensagem

Notes

Comportamento de objetos

- Uma mensagem é simplesmente uma operação que um objeto pode realizar em outro
- Operação e mensagem são termos intercambiáveis
- A passagem de mensagens é uma parte da equação que define o comportamento de um objeto.
- O estado do objeto também afeta o seu comportamento
- O estado do objeto representa o resultado cumulativo de seu comportamento

Notes

Operações em objetos

- Uma operação denota um serviço que é provido para outros objetos
  - **Modificadores**: uma operação que altera o estado de um objeto.
  - **Selecionadores**: uma operação que acessa o estado de um objeto (sem alterá-lo)
  - **Interadores**: uma operação que permite o acesso a "partes" de um objeto em uma ordem bem definida
  - **Construtores**: uma operação para criar e inicializar o estado de um objeto
  - **Destrutores**: uma operação para livrar o estado de um objeto e "destruí-lo".

Notes

Operações em objetos

- Todos os métodos associados a um objeto particular compreendem o seu protocolo. O protocolo de um objeto define assim o envelope do comportamento permitido de um objeto e assim compreende toda a visão estática e dinâmica do objeto.
- Um papel é uma máscara que um objeto usa e assim define um contrato entre uma abstração e seus clientes
- A existência de estado dentro de um objeto significa que a ordem em que as operações são invocadas é importante
  - Cada objeto é como uma minúscula máquina independente

Notes

Identidade

- Identidade é uma propriedade que distingue um objeto de todos os demais
- A identidade única de cada objeto é preservada ao longo do tempo de vida do objeto, mesmo que seu estado seja modificado.

Notes

Relações entre objetos

- Dependência
- Associação
- Agregação
- Composição

Notes

Dependência

- A dependência é o mais básico e o mais fraco tipo de relações entre classes.
- Existe uma dependência entre duas classes se algumas mudanças na definição de uma das classes pode re- sultar em modificações em outra classe.

Notes

Associação

- A associação é um relacionamento no qual um objeto usa ou interage com outro
- Em geral, você usa uma associação para representar algo como um atributo em uma classe
  - Ligação sempre presente
- Pode ser unidirecional ou bidirecional

Notes

Dependência vs Associação

```
1  class Professor is
2    field Student student
3    // ...
4    method teach(Course c) is
5      // ...
6      this.student.remember(c.getKnowledge())
```


Notes

Agregação

- A agregação é um tipo especializado de associação que representa relações individuais (one-to-many), múltiplas (many-to-many), e totais (whole-part) entre múltiplos objetos
- um objeto “tem” um conjunto de outros objetos e serve como um contêiner ou coleção
- O componente pode existir sem o contêiner e pode ser ligado através de vários contêineres ao mesmo tempo
- Sempre é direcional
- A parte pode existir sem o todo

Notes

Composição

- A composição é um tipo específico de agregação, onde um objeto é composto de um ou mais instâncias de outro
- A distinção entre esta relação e as outras é que o componente só pode existir como parte de um contêiner
- Não confundir com o conceito de composição de objetos no princípio de “composition over inheritance”

Notes

Panorama geral das relações existentes

- **Dependência**: Classe A pode ser afetada por mudanças na classe B
- **Associação**: Objeto A sabe sobre objeto B. Classe A depende de B
- **Agregação**: Objeto A sabe sobre objeto B, e consiste de B. Classe A depende de B
- **Composição**: Objeto A sabe sobre objeto B, consiste de B, e gerencia o ciclo de vida de B. Classe A depende de B

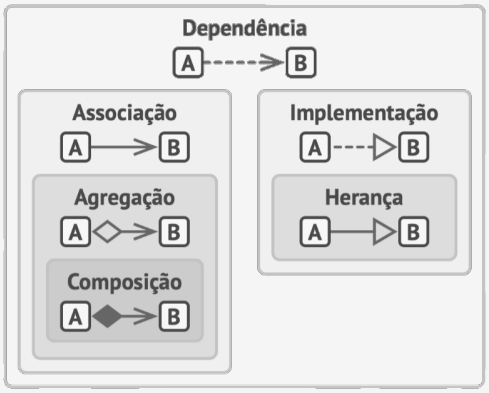
Notes

Relações de Classes

- **Implementação**: Classe A define métodos declarados na interface B. objetos de A podem ser tratados como B. Classe A depende de B
- **Herança**: Classe A herda a interface e implementação da classe B mas pode estendê-la. Objetos de A podem ser tratados como B. Classe A depende de B

Notes

Relações entre objetos e classes



Notes

Análise e Projeto

- Durante a análise e dos estágio iniciais do projeto, o desenvolvedor deve:
  - Identificar as classes que formam o vocabulário do domínio do problema.
  - Definir a estrutura pela qual os objetos trabalham em conjunto para prover o comportamento que satisfaça os requisitos do problema

Notes

A qualidade de uma Abstração

- Como podemos avaliar se uma classe ou objeto é bem projetado ?
  - **Acoplamento**: Dependência entre módulos
  - **Coesão**: Grau de conectividade entre elementos de um mesmo módulo
  - **Suficiência**: O módulo captura suficientemente as características das abstrações?
  - **Completude**: A interface captura todas as características de forma significativa ?
  - **Operações primitivas**: Suas operações são primitivas ? Podem ser implementadas eficientemente com a representação da abstração ?

Notes

Resumindo

Notes

Resumindo

- Um objeto tem estado, comportamento e identidade
- A estrutura e o comportamento de objetos semelhantes são definidos através de uma classe
- O estado de um objeto engloba todas as propriedades (normalmente estáticas) do objeto mais os valores correntes (usualmente dinâmicas) de cada uma destas propriedades.
- Comportamento é como um objeto age e reage em termos de suas alterações de estado e passagem de mensagens.
- Identidade é a propriedade de um objeto que o distingue de todos os outros objetos.

Notes

Resumindo

- A classe descreve um conjunto de objetos que compartilham uma estrutura comum e um comportamento comum
- Os tipos de relacionamentos incluem dependência, associação, agregação, composição e herança.
- Abstrações são as classes (objetos) que formam o vocabulário do domínio do problema
- Um conjunto de objetos trabalham em conjunto para fornecerem um comportamento que satisfaça algum requisito do problema
- A qualidade de uma abstração pode ser medida pelo seu acoplamento, coesão, suficiência, completude e primitivismo

Notes

That's all folks!

INSTITUTO FEDERAL
Santa Catarina
Câmpus Florianópolis

Notes

Notes

Notes

Notes