Curso Tecnologias Microsoft

Henrique Alves de Fernando - 09/05/2023

Aula 0 - Programação Orientada a Objetos

1) Paradigmas de programação

Orientação a procedimento:

- Foca em procedimentos e funções, depois quais são os dados
- Exemplo: C

Orientação à lógica:

- Foca em regras, padrões e inferências
- Especifica o que deve ser computado ao invés de como deve ser computado
- Exemplo: Prolog

Funcional:

- funções matemáticas (passagem de parâmetros, tipos de retorno, etc)
- Enfatiza avaliação de expressões
- Não utiliza comandos e algoritmos
- Não utiliza variáveis e atribuições
- Exemplos: Lisp, ML, Haskel

Orientação a objetos:

- Abstração, ligação dinâmica, herança
- Ênfase = estrutura de dados e funcionalidades associadas
- Inicia-se decidindo os objetos necessários
- Coleção de objetos que interagem entre si através de mensagens para resolver um problema maior
- Exemplos: C++, Java, C#

2) Objeto

- Não precisa ser um elemento físico. Pode ser uma ideia ou qualquer coisa abstrata.
- Entidade que une a representação da informação (Estrutura de dados) com sua manipulação (procedimentos)
- Representa entitade física, conceitual ou de software
- Possui capacidade de processamento e armazena um estado local
- **Propriedades:** estrutura de dados que representam o estado interno
- **Comportamento:** Métodos que agem sobre suas propriedades
- **Identidade:** Permite diferenciar um objeto do outro
- Benefícios:
    - **Abstração:** Ignora detalhes não relevantes e destaca os importantes. Ajuda a lidar com a complexidade
    - **Modularidade:** Programa feito a partir de módulos independentes, coesos e conectados, quebrando um sistema complexo em módulos individuais, mais facilmente gerenciáveis
    - **Encapsulamento:** Ocultar a informação. Interação de um objeto com o meio externo é realizado exclusivamente através dos métodos. Quanto menos acessível a informação é a outros objetos e pessoas, menor a chance de erros e mais facilmente gerenciáveis é o código
    - **Reusabilidade:** Objetos incorpooram as propriedades e comportamentos da classe. Reusabilidade de código entre os objetos da classe e do código de terceiros.

3) Classe

- Padrão para uma categoria de itens estruturalmente idênticos (objetos). É um modelo para os objetos.
- Declaração composta por:
    - **Modificador de acesso:** indica a visibilidade
    - **Identificação:** nome (primeira letra maiúscula e mesmo nome do arquivo)
    - **Henrança:** nome da superclace precedida de **: em C#**
    - **Interfaces:** lista de interfaces implementadas, separadas por vírgula, precedida da palavra-chave **implements**
    - **Corpo da classe:** implementação

4) Atributos

4.1) Atributos de instância:

- Copiados para todos os objetos criados
- Automaticamente inicializados pelo compilador se o programador não o fizer explicitamente
- `<modificador> tipo nomeAtributo;`
- Mudar um atributo de instância muda para apenas para o objeto em questão
- Para acessar, é necessária a referência de um objeto

4.2) Atributos de classe:

- Apenas uma cópia para todos os objetos da classe. Ela pertence à classe.
- `<modificador> static tipo nomeAtributo;`
- Modificar um atributo de classe em um objeto muda para todos
- É possível acessar usando apenas o nome da classe

5) Mensagens:

- Forma como os objetos se comunicam
- Componentes: objeto receptor, nome do método que se deseja executar, parâmetros necessários ao método

6) Métodos:

- Implementa algum aspescto do comportamento do objeto
- Definido na classe que pode acessar o estado interno de um objeto
- Sequência de ações que pode alterar os verificar seu estado
- Enquanto o valor dos atributos reside no objeto, o método reside na classe
- Assinatura:
    - **Modificadores:** tratam visibilidade e se ele é static
    - **Tipo de retorno**
    - **Identificador:** nome do método
    - **Lista de parâmetros**
- Corpo do método: Contém a implementação

- **Construtores:** usados para criar e inicializar objetos novos. Aloca espaço de memória para um objeto. Quando se faz `NomeClasse objeto` , só é definido um endereço de memória que objeto vai ficar. Funciona similarmente a um ponteiro. Mensmo nome da classe, devem ser públicos,
- **Destrutores:** usados para destruir métodos; C++ e C#. ~NomeDaClasse
- **Métodos de acesso:** Criados para permitir que atributos protegidos possam ter seus valores lidos e modificados por elementos de outras classes (**Gets e Sets**)
- Se só declarar uma variável e tentar usar um get, dá erro de compilação (ela está apontando para o nulo)
- Métodos de instância podem acessar variáveis e métodos de instância e de classe diretamente
- Métodos de classe podem acessar apenas métodos e variáveis de classe diretamente. Para acessar de instância, é necessaria a referência de um objeto

7) Sobrecarga:

Sobrecarga de métodos:

- Métodos com o mesmo nome e com assinaturas diferentes
- Tipo de retorno e visibilidade não entram como critério de diferenciação
- Mesma lista de parâmetros, mesmo que com tipos de retornos distintos resulta em erro de compilação

8) Modularização:

- **Pacotes:** Conjunto de classes na mesma estrutura hierárquica de diretórios
- **Classes:** projetos de objetos, onde há características e comportamentos deles
- **Métodos:** Serviços implementados na forma de um conjunto de instruções da linguagem que realizam alguma tarefa específica

9) Escopo:

- Porção de código onde a entidade declarada pode ser chamada pelo seu nome
    - **Parâmetro:** corpo do método onde foi declarado

    - **Variável local:** do ponto de declaração até final do bloco que a contém
    - **Método ou atributo:** Corpo da classe onde está declarado
- Se tem o mesmo nome de um atributo, o atributo é ocultado em um processo de sombreamento

10) Herança:

- Relação do tipo **generalização**/ especialização
- Objetos de um filho podem ser substituídos por objetos do pai
- Elementos específicos incorporam a estrutura e comportamento de elementos mais gerais
- É herdado: Estado (atributos e relacionamentos) e comportamento (operações)
- Classe System é a primordial em C#. Todas as outras herdam dela
- Herança múltipla não existe em C#. Não pode ter pai e mãe
- `super()` = construtor da superclasse. A primeira tarefa de um construtor de subclasse deve ser chamar o da superclasse. Se não fizer, o compilador chama o construtor default da superclasse
- Métodos públicos da classe ancestral pode ser chamado nas descendentes
- `super` representa a classe ancestral
- Não pode ser circular

Sobrescrita (override):

- Permite especializar os métodos herdados, alterando seu comportamento
- Cria-se um método na subclasse com mesma assinatura e tipo de retorno (pode ser subtipo) que da superclasse
- Impossível reduzir a visibilidade de um método em sobrescrita; promoção de visibilidade pode ocorrer (private -> public é ok)
- **Sobrescrita**
    - Mesmo nome, assinatura, requer herança
- **Sobrecarga:**
    - Mesmo nome, assinaturas diferentes, mesma classe ou na hierarquia

A palavra final:

- **Variável:** não pode assumir outro valor

- **Método:** não pode ser sobrescrito
- **Classe:** não admite herança

11) Relacionamentos:

- Mecanismo de interação entre as classes
- Têm multiplicidade
- Unidirecional: Apenas uma das classes contém a referência para a outra
- Bi-direcional
- Cada entidade tem um campo de relacionamento que se refere à outra entidade; Deve ser evitado

Associação:

- Classes interagem, mas não há relação todo-parte e uma não depende da outra para existir
- Unária, binária, -ou

Agregação:

- Relação todo-parte. Indica que um objeto parte é atributo de um objeto todo

Composição:

- Relação parte-todo. Se o objeto maior for removido, suas partes filhas também são