**1 HERANÇA EM PROGRAMAÇÃO ORIENTADA A OBJETOS**

1.1 Definição

“Herança é um termo que se refere a uma classe nova, que pode ser criada a partir de outra já existente. Ou seja, ela pode "herdar" atributos e comportamentos da classe a ser estendida. Por exemplo, uma camisa herda o material que ela é confeccionada de vestuário, como TECIDO (a herança pode ser de características e de comportamentos). Isso significa que, a partir de uma classe existente, pode-se acrescentar funcionalidades criando uma nova classe baseada nela. A camisa pode ter uma característica que não é comum em todas as camisas, apenas nos polos. A classe na qual é gerada a nova classe é chamada de classe básica (ancestral, superclasse ou classe genérica) e a nova é chamada de classe derivada (descendente, subclasse, ou ainda, classe especializada). Assim, a classe derivada herdará os atributos e comportamentos da básica.” (ARAUJO, 2017).

“Ao criar uma classe, em vez de declarar membros completamente novos, você pode designar que a nova classe deve herdar membros de uma classe existente. A classe existente é chamada de superclasse e a nova classe de subclasse. Cada subclasse pode tornar-se uma superclasse para futuras subclasses. Uma subclasse pode adicionar seus próprios campos e métodos. Portanto, uma subclasse é mais específica que sua superclasse e representa um grupo mais especializado de objetos. É por isso que herança é às vezes conhecida como especialização.” (DEITEL, 2010, p. 279).

1. 2 Utilidade

É uma forma de reutilização de software em que uma nova classe é criada absorvendo membros de uma classe existente e aprimorada com capacidades novas ou modificadas. Com a herança, você pode economizar tempo durante o desenvolvimento de um programa baseado novas classes no software existente testado, depurado e de alta qualidade. Isso também aumenta a probabilidade de que um sistema seja implementado e mantido efetivamente. (DEITEL, 2010, p. 279)

1. 3 Vantagens e desvantagens

“Um problema com a herança é que uma subclasse pode herdar métodos de que não necessita ou que não deveria ter. Mesmo quando um método de superclasse é adequado a uma subclasse, essa subclasse precisa frequentemente de uma versão personalizada do método. (DEITEL, 2010, p. 281).

1. 4 O que são classes abstratas

“Quando pensamos num tipo de classe, supomos que os programas criam objetos desse tipo. Às vezes é útil declarar classes - chamadas classe abstratas - para as quais você nunca pretende criar objetos. Como elas só são usadas como superclasses em hierarquias de herança, abstratas são incompletas. As subclasses devem declarar as “partes ausentes” para tornarem-se classes “concretas”, a partir das quais você pode instanciar objetos. Caso contrário, essas subclasses, também, serão abstratas.

O propósito de uma classe é fornecer uma superclasse apropriada a partir da qual outras classes podem herdar e assim podem compartilhar um design comum.” (DEITEL, 2010, P. 309).

**2 POLIMORFISMO**

2.1 Definição

“Polimorfismo, que significa "várias formas", é um termo designado a objetos de classes distintas, que podem reagir de uma maneira diferente ao fazerem o uso de um mesmo método. É um dos responsáveis pela especialização de classes em uma aplicação OO, pois comportamentos definidos, mas não implementados nas classes mais genéricas, serão implementados nas novas subclasses. Quando implementamos uma classe, podemos ter nela a definição de propriedades e métodos. Alguns métodos podem ser acessados pelo objeto cliente desta classe, mas outros podem ter um uso específico, o que não liberaria esse acesso sem uma certa segurança. Podemos, por exemplo, ter um método que realiza o saque de dinheiro de uma conta bancária, no qual o saldo precisa ser atualizado, mas, para isso, é preciso fazer o uso de um serviço específico para atualização do saldo que não pode ser exposto ao método chamador.” (ARAUJO, 2017).

“No paradigma orientado a objetos existe um recurso muito útil chamado polimorfismo, que permite muitas vantagens como a extensibilidade. De acordo com Deitel e Deitel (2010), com o polimorfismo podemos projetar e implementar sistemas que são facilmente extensíveis, isso significa que novas classes podem ser adicionadas com pouca ou nenhuma modificação às partes gerais do programa. A palavra polimorfismo vem do grego e quer dizer "muitas formas". Daí a nossa analogia com o camaleão, que também assume várias formas de cores para se camuflar e se proteger de predadores. **O polimorfismo permite ao desenvolvedor usar o mesmo elemento de várias formas diferentes, ou seja, um objeto pode comportar de maneiras diferentes ao receber uma mensagem. Basicamente existe quatro tipos de polimorfismo:**

1. Estático ou sobrecarga

2. Dinâmico ou sobrescrita

3. De inclusão

4. Paramétrico

O polimorfismos estático sobrecarga acontece quando temos o mesmo método implementado várias vezes na mesma classe. Segundo Horstmann e Cornell (2010), essa capacidade é chamada de sobrecarga e clonagem. O sobrecarregamento ocorre se vários métodos tiverem o mesmo nome, mas parâmetros diferentes. Isso é o que chamamos de assinatura do método que, de acordo com Barnes e Koliing (2010), é o cabeçalho do método.” (MASIERO, OBERLEITER, 2011).

**3 ENCAPSULAMENTO**

3. 1 Definição

“Encapsulamento é um mecanismo que trata de dar segurança aos objetos, mantendo-os controlados em relação ao seu nível de acesso. Desta maneira, o encapsulamento contribui fundamentalmente para diminuir os malefícios causados pela interferência externa sobre os dados, pois isola partes do código. Isso reduz o acoplamento, que mede o quanto um elemento depende e conhece o outro.” (ARAUJO, 2017).

“Um dos recursos mais interessantes na orientação a objetos é o encapsulamento, um mecanismo que provê proteção de acesso aos membros internos de um objeto. Lembre-se de que uma classe tem responsabilidade sobre os atributos que ela contém. Desta forma, há certas propriedades de uma classe que devem ser tratadas exclusivamente por métodos de mesa, que são implementações projetadas para manipular essas propriedades de forma correta. As propriedades não devem ser acessadas diretamente de fora do escopo de uma classe, pois dessa forma a classe não fornece mais garantias sobre os atributos que ela contém, perdendo, assim, a responsabilidade sobre eles.” (DALL'OGLIO, 2018)

Referências:

ARAUJO, Everton. **Orientação a Objetos em C#: Conceitos e implementações em .NET.** Editora Casa do Código, São Paulo, 2017.

DEITEL, Paul; DEITEL, Harvey. Java: como programar. 8 ed. São Paulo: Pearson Prentice Hall, 2010.

OBERLEITNER, Allen; MASIERO, Andrey. **PROGRAMAÇÃO ORIENTADA A OBJETOS**. Editora Senac:São Paulo, 2011.

DALL'OGLIO, Pablo. **Programando com Orientação a Objetos**. Editora Novatec, 4ª Edição, São Paulo, 2018.