Herança em programação orientada a objetos

● Definição

O conceito de herança está fundamentado na definição de uma classe com base em outra. A classe usada como referencial pode ser chamada de superclasse, classe-mãe, classe-base ou generalização. A nova classe criada a partir de outra passa a ser uma subclasse, classe-filha, classe derivada ou especialização. Podemos redefinir métodos e criar novos atributos na subclasse. A implementação de herança em Java é realizada inserindo-se a palavra extends seguida do nome da superclasse após a definição da subclasse. (MENDES, 2009, p. 186).

O paradigma de orientação a objetos oferece uma estrutura hierárquica e modular para reutilização de código por meio de uma técnica conhecida como herança. Essa técnica permite projetar classes genéricas que podem ser especializadas em classes mais particulares, em que as classes especializadas reutilizam o código das mais genéricas. A classe genérica, também conhecida por classe base ou superclasse, define variáveis de instância “genéricas” e métodos que se aplicam em uma variada gama de situações. A classe que especializa, estende ou herda de uma superclasse não precisa fornecer uma nova implementação para os métodos genéricos, uma vez que os herda. Deve apenas definir aqueles métodos que são especializados para esta subclasse em particular. (GOODRICH; TAMASSIA, 2013, p.65).

● Utilidade

Quadro 1 - Quando usar herança

| **Situação** | **Explicação** |
| --- | --- |
| Semelhanças | Sempre que houver semelhanças entre duas ou mais classes, aplique herança. |
| Extensão | Se uma nova classe a ser definida possui certos recursos em adição aos recursos de uma classe existente, utilize herança e codifique apenas os recursos adicionais dessa nova classe. |
| Relacionamento | Se o relacionamento entre as classes é É-UMA ou É-UM-TIPO-DE ou É-TIPO ou É-TIPO-DE, selecione herança. |
| Generalização | O relacionamento hierárquico cria uma árvore de relacionamento com tipo especializado, ramificando-se de tipos mais generalizados. Subindo na hierarquia consegue uma maior generalização. Mas a classe existente não deve ser modificada quando a herança é optada. Herança é aconselhável quando a generalização é fixa e não requer nenhuma modificação ou mudança. |
| Especialização | O uso mais comum de herança é para especialização. A classe herdada é uma especializada tendo mais informações e funcionalidades. É um superconjunto da classe existente. Descendo na hierarquia leva à especialização. Mas muitos níveis resultam em complexidade. |
| Combinação | Se uma classe é herdada de mais de uma classe, ela resulta em múltiplas heranças. Múltiplas heranças combinam recursos de mais de uma classe existente. Algumas das linguagens orientadas a objetos não suportam múltiplas heranças diretamente. As limitações são mais comparadas com os benefícios de múltiplas heranças. Consequentemente, é muito importante julgar a necessidade de múltiplas heranças. |

Fonte: BUYYA; SELVI; CHU (2009)

Uma das características identificadoras das linguagens orientadas a objetos e sistemas é o suporte para herança. Com herança podemos definir uma nova classe permitindo que ela assuma algumas das características de uma classe previamente definida, normalmente reduzindo a quantidade de trabalho necessária para definir a nova classe. (HOLMES; JOYCE, 2001, p. 290).

● Vantagens

Segundo Buyya, Selvi e Chu (2009), usar herança ajuda a: reutilizar o código existente e ampliar a funcionalidade; adicionar novos membros à classe derivada para especializar a classe; substituir a implementação de métodos existentes substituindo um método que já existe na classe base. Os métodos na classe derivada ajudam a exibir comportamento polimórfico; organiza componentes do programa em categorias e subcategorias resultando em classificações de programas. Classificação é o uso de herança mais amplamente aceito embora outros mecanismos também possam ser utilizados para classificação; gerar sistemas de programas mais rapidamente e facilmente usando componentes reutilizáveis.

Segundo Holmes e Joyce (2001), os benefícios do uso de herança são: aumento na capacidade de reutilizar classes; aumenta o nível de abstração em um programa; melhora a clareza no design de classes permitindo que a implementação de métodos seja adiada em superclasses e apareça em subclasses.

● Desvantagens

Entre os problemas da Herança está o fraco encapsulamento entre classes e subclasses e o forte acoplamento entre elas onde ao mudar uma superclasse pode afetar todas as subclasses além de violar o princípio básico de projeto OO em que devemos ter sempre um baixo acoplamento entre as classes. (DEVMEDIA, 2022).

A principal desvantagem de usar herança é que as duas classes (classe parente e criança) ficam fortemente acopladas. Isso significa que se alterarmos o código da classe parente, isso afetará todas as classes crianças que estão herdando/derivando a classe parente e, portanto, não podem ser independentes umas das outras. (STUDYTONIGHT, 2022, tradução nossa).

● O que são classes abstratas

Uma classe abstrata é desenvolvida para representar classes e conceitos abstratos. A classe abstrata é sempre uma superclasse que não permite que nenhum objeto seja criado a partir dela, ou seja, não se pode executar a operação de new usando uma classe abstrata. Uma classe abstrata é um meio termo entre uma classe concreta que contém uma implementação para todos os métodos declarados e uma interface na qual nenhum dos métodos declarados será implementado, e sim pelas classes que a implementam. (MENDES, 2009, p. 196).

“Uma classe abstrata é aquela que contém uma declaração de método vazia (isto é, uma declaração de método sem implementação) e definições concretas de métodos e variáveis de instância.” (GOODRICH; TAMASSIA, 2013).

Polimorfismo

● Definição

Literalmente, “polimorfismo” significa “muitas formas”. No contexto de projeto orientado a objetos, entretanto, refere-se à habilidade de uma variável de objeto de assumir formas diferentes. Linguagens orientadas a objetos, como Java, referenciam objetos usando variáveis de referência. (GOODRICH; TAMASSIA, 2013, p.67).

Polimorfismo significa que uma operação (método) pode ser implementada em diferentes meios em diferentes classes. Uma classe que providencia uma assinatura de um método (com ou sem implementação) para uma variedade das suas classes derivadas para providenciar diferentes implementações é conhecida como uma classe polimórfica. (BUYYA; SELVI; CHU, 2009, p.227, tradução nossa).

Encapsulamento

● Definição

“O encapsulamento é o processo de proteger os membros de uma classe (atributos e métodos), permitindo que somente os membros necessários (públicos) sejam acessados pelos usuários da classe.” (MENDES, 2009).

Outro princípio importante em projeto orientado a objetos é o conceito de encapsulamento, que estabelece que os diferentes componentes de um sistema de software não devem revelar detalhes de suas respectivas implementações. Uma das maiores vantagens do encapsulamento é que ele oferece ao programador liberdade na implementação dos detalhes do sistema. A única restrição ao programador é manter a interface abstrata percebida pelos de fora. (GOODRICH; TAMASSIA, 2013, p. 62).

Referências

BUYYA, Rajkumar; SELVI, S Thamarai; CHU, Xingchen. **Object-Oriented Programming with Java**: essentials and applications. New Delhi: Tata McGraw Hill, 2009.

DEVMEDIA. **Herança versus Composição: qual utilizar?** Disponível em: https://www.devmedia.com.br/heranca-versus-composicao-qual-utilizar/26145. Acesso em: 31 maio 2022.

GOODRICH, Michael T.; TAMASSIA, Roberto. **Estruturas de Dados & Algoritmos**. 5. ed. Porto Alegre: Bookman, 2013.

HOLMES, Barry; JOYCE, Daniel. **Object-oriented programming with Java**. Sudbury: Jones & Bartlett, 2001.

MENDES, Douglas Rocha. **Programação Java com ênfase em orientação a objetos**. São Paulo: Novatec, 2009.

STUDYTONIGHT. **Inheritance (IS-A relationship) in Java**. Disponível em: https://www.studytonight.com/java/inheritance-in-java.php. Acesso em: 31 maio 2022.