INSTITUTO FEDERAL
Minas Gerais
Campus São João Evangelista

Valdir de Souza Carvalho Neto **SI 241 - Linguagem de Programação I**

Relatório de Seminário

Tema: Herança e Polimorfismo

1. INTRODUÇÃO

O presente relatório tem como objetivo descrever a atividade prática realizada no dia 03 de dezembro de 2025, referente ao seminário sobre "Herança e Polimorfismo". O foco da aula foi abordar estratégias para a construção de código reutilizável, escalável e de fácil manutenção utilizando a linguagem Java.

Durante a atividade, enfrentamos o problema comum no desenvolvimento de sistemas: a duplicidade de código ao lidar com diferentes tipos de entidades que compartilham características. Para solucionar isso, aplicamos os conceitos fundamentais da Orientação a Objetos, criando um sistema de gerenciamento de funcionários onde pudemos ver na prática como a Herança organiza o código e como o Polimorfismo permite flexibilidade nas regras de negócio.

2. DESENVOLVIMENTO

A atividade consistiu na implementação de uma hierarquia de classes representando o cenário de uma empresa. Inicialmente, definimos uma classe base (Superclasse) chamada *Funcionario*. Nela, centralizamos os atributos comuns a qualquer pessoa colaboradora da empresa: *nome*, *cpf* e *salario*. Utilizamos o modificador de acesso *protected* nestes atributos, uma prática que permite que as classes filhas visualizem esses dados diretamente, mantendo o encapsulamento em relação a classes externas.

Para representar um cargo mais específico, criamos a classe *Gerente*, que estende *Funcionario* através da palavra-chave *extends*. Isso validou a regra "É Um" (Is-A), pois logicamente todo Gerente é um Funcionário. Na prática, isso significou que a classe *Gerente* herdou automaticamente todos os atributos e métodos da classe pai, sem a necessidade de reescrever código. Adicionamos apenas o que era específico para esse cargo: o atributo

setorResponsavel. No construtor do *Gerente*, utilizamos a palavra-chave *super(...)* para chamar o construtor da classe *Funcionario*, garantindo que a inicialização dos dados herdados fosse feita corretamente pela classe base.

O ponto alto da prática foi a aplicação do Polimorfismo através da sobrescrita de métodos (Override). Observamos que, embora ambos sejam funcionários, o comportamento de certas ações deve ser diferente:

- **Bonificação:** Na classe *Funcionario*, implementamos o método *getBonificacao()*, que retorna 10% do salário. Porém, para o *Gerente*, a regra de negócio exigia um cálculo diferenciado. Utilizamos a anotação *@Override* para redefinir este método na classe *Gerente*, alterando o cálculo para 15% do salário mais um acréscimo de R$ 1.000,00.
- **Impressão de Dados:** Também sobrescrevemos o método *Imprimir()*. No *Gerente*, usamos *super.Imprimir()* para aproveitar a lógica de impressão padrão (Nome, CPF, Salário) e adicionamos apenas a linha para exibir o "Setor Responsável", evitando duplicação de lógica.

Na classe *Main*, instanciamos objetos de ambos os tipos. Foi possível verificar que, ao chamar o método *alterarInformacao* ou calcular a bonificação, o sistema sabia exatamente qual comportamento executar baseando-se no tipo do objeto (seja ele um funcionário comum ou um gerente), caracterizando o polimorfismo em tempo de execução.

3. CONCLUSÃO

A atividade prática demonstrou claramente as vantagens do uso de Herança e Polimorfismo. Ao centralizar o código comum na classe *Funcionario*, conseguimos um sistema mais limpo e organizado, eliminando redundâncias. A principal lição aprendida foi que a Herança não serve apenas para economizar linhas de código, mas para criar uma estrutura lógica onde comportamentos genéricos podem ser especializados nas subclasses.

Conclui-se que o uso dessas técnicas torna o sistema extensível. Se no futuro precisarmos adicionar novos cargos (como Diretor ou Estagiário), bastará estender a classe base sem prejudicar o funcionamento das classes já existentes, garantindo um software de fácil manutenção e evolução.