Universidade Estadual do Norte Fluminense Darcy Ribeiro

Curso: Ciência de Computação **Data:** 14/09/2023

Disciplina: POO

Professora: Annabell Tamariz

Nome do Aluno: Rômulo Souza Fernandes **Matrícula:** 00119110559

RELATÓRIO SOBRE O USO DO PARADIGMA O.O COM A LINGUAGEM RUBY

Neste relatório, será explicado a aplicação dos princípios do Paradigma de Orientação a Objetos (O.O.) para solucionar os Desafios 1 e 2 das Aulas 6 e 7 do Canal One Bit Code, bem como o desafio da Aula 8 relacionado ao sistema de futebol, utilizando a linguagem Ruby.

DESAFIO 1 - AULA 6 ONE BIT CODE

1. Definição das Classes:

Primeiramente, foram definidas três classes: Esportista, Jogadordefutebol, e Maratonista. Cada uma dessas classes representa um elemento dentro do contexto esportivo e utiliza o conceito de herança em Orientação a Objetos.

A classe Esportista é a classe mãe. Ela contém um método chamado competir, que é compartilhado por todas as subclasses. Isso permite a organização hierárquica das classes, onde as subclasses podem herdar características e comportamentos da classe mãe.

As classes Jogadordefutebol e Maratonista são subclasses da classe Esportista. Isso significa que elas herdam todos os métodos e atributos da classe Esportista, incluindo o método competir. No entanto, cada uma dessas subclasses adiciona seu próprio método correr com implementações específicas.

2. Método competir na Classe Esportista:

Na classe Esportista, foi definido o método competir, que imprime a mensagem "Participando de uma competição". Isso representa a encapsulação de um comportamento comum a todos os esportistas. Aqui, estamos seguindo o princípio da encapsulação, que é um dos princípios-chave do O.O. O encapsulamento permite que os detalhes internos de uma classe sejam ocultos e acessíveis apenas por meio de métodos definidos.

3. Métodos correr nas Classes Jogadordefutebol e Maratonista:

As classes Jogadordefutebol e Maratonista possuem métodos correr, mas cada um deles tem uma implementação diferente. Isso demonstra o conceito de polimorfismo, onde as subclasses podem fornecer implementações específicas para métodos herdados da classe mãe. No nosso caso, ambas as subclasses estão substituindo o método correr da classe Esportista com suas próprias versões.

4. Criação de Instâncias:

No código fornecido, criamos duas instâncias das classes Jogadordefutebol e Maratonista, representando um jogador de futebol e um maratonista.

5. Chamada de Métodos:

Finalmente, chamamos os métodos correr e competir nas instâncias jogadordefutebol e maratonista, respectivamente. Essa é a aplicação do conceito de polimorfismo, onde, embora as duas instâncias sejam do tipo Esportista, seus métodos correr se comportam de maneira diferente com base na implementação em suas classes específicas.

Resultados da Execução:

jogadordefutebol.correr resultará na impressão de "Correndo atrás da bola".
maratonista.competir resultará na impressão de "Participando de uma competição".

Em resumo, o uso do Paradigma de Orientação a Objetos neste código permitiu a criação de classes, herança, encapsulamento e polimorfismo para modelar e resolver o Desafio 1 de forma organizada e orientada a objetos.

DESAFIO 2 - AULA 7 ONE BIT CODE

1. Criação das Classes:

No arquivo produto.rb, foi criada a classe Produto. Utilizando encapsulamento, foram definidas propriedades: nome e preco, com métodos de leitura e escrita. No arquivo mercado.rb, foi criada a classe Mercado, onde uma instância de Mercado é inicializada com um objeto Produto. Isso representa uma associação entre as classes Mercado e Produto, com a classe Mercado utilizando composição.

2. Criação da Classe Mercado:

No arquivo app.rb, as classes Mercado e Produto foram importadas. Foram criadas instâncias da classe Produto (produto1 e produto2), definindo seus nomes e preços. Uma instância da classe Mercado (merc) foi associada a diferentes produtos em momentos distintos, ilustrando a composição de objetos, onde a mesma instância de mercado pode ser associada a diferentes produtos em momentos diferentes. O método comprar foi chamado nas instâncias de Mercado (merc), exibindo informações do produto associado a cada instância de mercado.

3. Uso das Classes e Composição:

No arquivo app.rb, as classes Mercado e Produto foram incluídas utilizando a diretiva require_relative. Após isso, foram instanciados dois objetos da classe Produto, denominados produto1 e produto2, e foram definidos valores para seus atributos nome e preco.

Em seguida, criou-se uma única instância da classe Mercado, chamada "merc", e essa instância foi associada a diferentes produtos em momentos distintos. Esse procedimento ilustra o conceito de composição de objetos, em que a mesma instância de mercado pode ser relacionada a diferentes produtos em momentos específicos.

Por fim, o método comprar foi invocado em cada instância de Mercado (merc), resultando na exibição das informações referentes ao produto associado a essa instância de mercado em cada chamada.

Resultados da Execução:

Ao executar o código em app.rb, serão geradas as mensagens de compra para os produtos "arroz" e "feijão", incluindo seus respectivos preços.

Resumidamente, a aplicação do Paradigma de Orientação a Objetos neste código possibilitou a estruturação de classes, o encapsulamento de propriedades, a composição de objetos e a interação entre eles, resultando na resolução do Desafio 2 de forma plenamente orientada a objetos.

SISTEMA DE FUTEBOL - AULA 8

1. Definição das Classes:

No contexto deste sistema de futebol, foram definidas duas classes principais: JogadorDeFutebol e Time. Cada uma dessas classes representa uma entidade dentro do universo esportivo e aplica os princípios do Paradigma de Orientação a Objetos.

A classe JogadorDeFutebol atua como a classe base. Ela inclui atributos como primeiro_nome, ultimo_nome, numero_camisa, e o método set_numero_camisa. Esse método garante a validação do número da camisa e ilustra a importância do encapsulamento, ocultando os detalhes internos da validação.

A classe Time é responsável por representar os times de futebol. Ela possui atributos como nome e uma lista de jogadores. A associação entre Time e JogadorDeFutebol exemplifica a composição, onde uma classe (Time) contém objetos de outra classe (JogadorDeFutebol).

2. Método nome na Classe JogadorDeFutebol:

Dentro da classe JogadorDeFutebol, o método nome foi definido. Esse método retorna o nome completo do jogador, aplicando o conceito de encapsulamento para acessar os atributos primeiro_nome e ultimo_nome e formatá-los de acordo.

3. Método posicao na Classe JogadorDeFutebol:

A classe JogadorDeFutebol inclui o método posicao, que determina a posição do jogador com base no número da camisa. Isso demonstra a utilização do polimorfismo, uma vez que diferentes faixas de números de camisa correspondem a diferentes posições no campo.

4. Criação de Instâncias:

No exemplo fornecido, criamos duas instâncias da classe Time (representando "Time A" e "Time B") e várias instâncias da classe JogadorDeFutebol. Isso ilustra a criação de objetos com base nas classes definidas.

5. Chamada de Métodos:

Para visualizar as informações dos jogadores em cada time, utilizamos o método mostrar_jogadores nas instâncias de Time. Esse processo de chamada de métodos demonstra como a composição de objetos e a interação entre classes podem ser aplicadas de forma eficaz.

Resultados da Execução:

Ao executar o código fornecido para o sistema de futebol, os seguintes resultados são observados:

Jogadores do Time Time A:
Jogador: Ronaldo Cristiano
Camisa: 7
Posição: Atacante
Jogador: Messi Lionel
Camisa: 30
Posição: Atacante

Jogador: Ramos Sergio
Camisa: 4
Posição: Zagueiro

Jogadores do Time Time B:
Jogador: Jr Neymar
Camisa: 10
Posição: Atacante

Jogador: van Dijk Virgil
Camisa: 4
Posição: Zagueiro

Jogador: Iniesta Andres
Camisa: 8
Posição: Meio-Campista

Neste contexto, a aplicação do Paradigma de Orientação a Objetos possibilitou a criação das classes JogadorDeFutebol e Time, a definição de atributos e métodos encapsulados para representar jogadores e times de futebol, a composição de objetos ao adicionar jogadores aos times e, finalmente, a interação entre esses objetos ao exibir as informações dos jogadores de cada time. Isso resultou em uma abordagem estruturada e orientada a objetos para representar e resolver os desafios relacionados ao sistema de futebol.