Usando Expressões Lambda e Classes Anônimas para Implementar Interfaces Funcionais em Java

No desenvolvimento em Java, há diferentes maneiras de implementar interfaces funcionais. Duas dessas abordagens são o uso de expressões lambda e a utilização de classes anônimas. Vamos explorar como essas duas técnicas podem ser aplicadas, utilizando como exemplo uma interface chamada `Veiculo`.

O Cenário: Interface `Veiculo` e a Classe `MinhaClasseUsandoLambda`

Primeiro, consideremos a seguinte interface:

```
public interface Veiculo {
    public void dirigir(String comando);
}
```

A interface `Veiculo` define um único método abstrato, `dirigir(String comando)`, o que a torna uma **interface funcional**. Uma interface funcional é uma interface que possui exatamente um método abstrato, permitindo que seja implementada usando expressões lambda.

Implementando `Veiculo` com Expressão Lambda

Na classe `MinhaClasseUsandoLambda`, utilizamos uma expressão lambda para implementar o método `dirigir` da interface `Veiculo`:

```
public class MinhaClasseUsandoLambda {

    public static void acelerar(String comando, Veiculo v) {
        v.dirigir(comando);
    }

    public static void main(String[] args) {
        // Implementacao usando uma expressao lambda
        Veiculo meuCaminhao = c -> System.out.println(c);

        acelerar("Acelerar!", meuCaminhao);
    }
}
```

Aqui, **a expressão lambda `c -> System.out.println(c)` é utilizada para fornecer a implementação do método `dirigir`**. O código é simples e conciso, fazendo uso das funcionalidades modernas de Java para escrever menos código e, ainda assim, manter a legibilidade.

Implementando `Veiculo` com Classe Anônima

Agora, vejamos a implementação da interface `Veiculo` utilizando uma classe anônima na classe `MinhaClasseUsandoAnonima`:

```
public class MinhaClasseUsandoAnonima {

    public static void acelerar(String comando, Veiculo v) {
        v.dirigir(comando);
    }

    public static void main(String[] args) {
        // Implementacao explicita usando uma classe anonima
        Veiculo meuCaminhao = new Veiculo() {
            @Override
            public void dirigir(String comando) {
                System.out.println(comando);
            }
        };

        acelerar("Acelerar!", meuCaminhao);
    }
}
```

Nesta abordagem, **criamos uma instância de uma classe anônima que implementa a interface `Veiculo`**. A classe anônima permite que a implementação do método `dirigir` seja definida diretamente no local onde a interface é usada, sem a necessidade de criar uma classe separada.

Comparando as Abordagens

Vantagens das Expressões Lambda

1. **Concisão**: Expressões lambda permitem escrever menos código. Como visto na classe `MinhaClasseUsandoLambda`, a implementação do método `dirigir` é reduzida a uma única linha.
2. **Legibilidade**: Para desenvolvedores familiarizados com lambdas, essa abordagem pode tornar o código mais fácil de ler e entender, especialmente em casos onde o comportamento é simples.
3. **Moderno e Funcional**: As expressões lambda representam um estilo de programação mais funcional, que é mais comum em linguagens modernas e permite escrever código de forma mais declarativa.

Desvantagens das Expressões Lambda

1. **Complexidade para Novatos**: Para quem é novo em Java ou em programação funcional, as expressões lambda podem parecer menos intuitivas e difíceis de entender inicialmente.
2. **Menos Flexibilidade**: Expressões lambda são mais adequadas para implementações simples. Para lógica mais complexa, o uso de lambdas pode prejudicar a clareza do código.

Vantagens das Classes Anônimas

1. **Clareza e Explícito**: Classes anônimas deixam explícito o que está sendo implementado, o que pode ser mais fácil de entender, especialmente para iniciantes.

2. **Capacidade de Manipular Estado**: Ao contrário das expressões lambda, as classes anônimas podem ter estado interno e métodos adicionais, oferecendo mais flexibilidade.
3. **Suporte Completo para Métodos Herdados**: Se você precisa sobrescrever vários métodos de uma classe ou interface, as classes anônimas são a melhor opção.

Desvantagens das Classes Anônimas

1. **Verbosidade**: O código com classes anônimas tende a ser mais longo e, em casos simples, isso pode prejudicar a legibilidade.
2. **Antiquado**: Com a introdução de expressões lambda em Java 8, o uso de classes anônimas para implementar interfaces funcionais pode parecer antiquado e menos eficiente.

Ambas as abordagens—expressões lambda e classes anônimas—têm seus méritos e desvantagens. A escolha entre elas depende do contexto e da complexidade da tarefa que você está tentando resolver. Para implementações simples e concisas, as expressões lambda são uma excelente escolha. Por outro lado, se você precisar de mais flexibilidade ou se estiver trabalhando com uma lógica mais complexa, as classes anônimas podem ser mais apropriadas.