Alunos:

- **Marcos Vinicius Bueno Prestes - RA: 2465760**
- **Micael Ribeiro Rocha - RA: 2454424**

1) Padrões escolhidos:

- **Factory:** Escolhemos esse padrão criacional para definir uma interface ou método abstrato para a criação de objetos, mas delega a responsabilidade de instanciar esses objetos para as subclasses. A principal ideia por trás desse padrão é fornecer uma maneira flexível de instanciar classes, permitindo que a criação de objetos seja desacoplada do código que os utiliza.

- **Decorator:** Em relação ao padrão estrutural de projeto, o Decorator permite adicionar funcionalidades a um objeto dinamicamente, sem modificar sua estrutura básica ou classe original. Isso é feito ao envolver o objeto original com uma série de "decoradores", cada um adicionando ou modificando o comportamento do objeto decorado.

- **Observer:** Por fim, o padrão comportamental escolhido, o Observer, define uma relação de dependência "um-para-muitos" entre objetos, de forma que, quando um objeto (chamado de sujeito) muda de estado, todos os seus dependentes (observadores) são notificados e atualizados automaticamente.

2) Refatorações feitas:

- **Single Responsibility Principle:** O arquivo *index.js* é responsável por executar os componentes da aplicação de carros, portanto, para manter o princípio S do solid é necessário fazer com que esse componente seja responsável apenas pela execução. Assim, os métodos de input de dados foram migrados para o arquivo */src/input.js*, aumentando a flexibilidade do sistema no caso de outra classe precisar dos mesmos inputs e diminuindo as responsabilidades do *index.js*.

- **Open-Closed Principle:** No *index.js* é utilizado o padrão de projeto Decorator, o qual inicialmente foi implementado utilizando if. A fim de deixar o código fonte fechado para modificações, esse condicional foi retirado e a nova implementação modularizada no arquivo */src/decorators/Decorator.js* permite que, ao adicionar novos decorators no array arrDecorators, qualquer futura implementação não

acarrete em alterações no código já existente. Desse modo, o código fica mais estável, diminuindo a chance de bugs em funcionalidades que já estão implementadas.

- **Liskov Substitution Principle:** Segundo esse princípio, uma classe filha deve ser substituível por sua classe mãe. Assim, a fim de possibilitar esse comportamento, dentro de */src/cars/Carro.js* a classe Carro possui o maior nível de abstração possível para as classes CarroEletrico, CarroHibrido e CarroCombustao herdarem, sendo possível que a substituição seja possível sem maiores complicações.