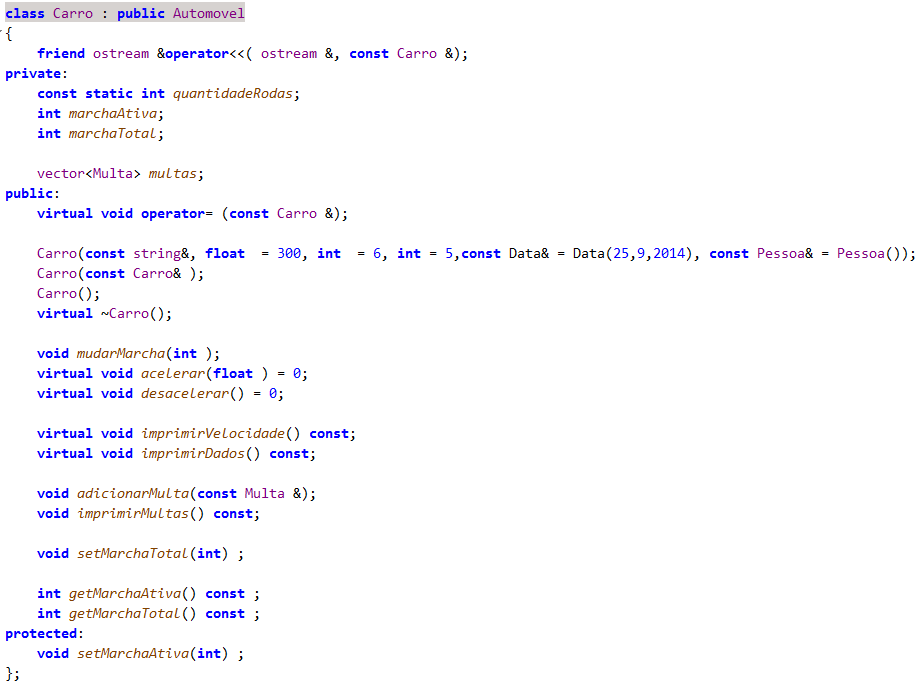
***Importante***: Para item abaixo deve ser copiado trechos do código que cumprem o requisito e explicado, se não for aparente, o porquê o requisito é cumprido. Sejam bem explícitos. Deve ser indicado também o .h ou .cpp no qual o trecho do código está. Eu avaliarei o código do Github a partir desse documento para confirmá-lo e também para detectar possíveis erros. **Quem não seguir o que está indicado aqui, não terá o projeto avaliado e perderá a atividade.**

**Requisitos funcionais**

***Polimorfismo***

1. ~~Duas classes abstratas, sendo que uma classe abstrata herda da outra classe abstrata na hierarquia de classes;~~

(Carro.h)~~~~

(Automovel.h)

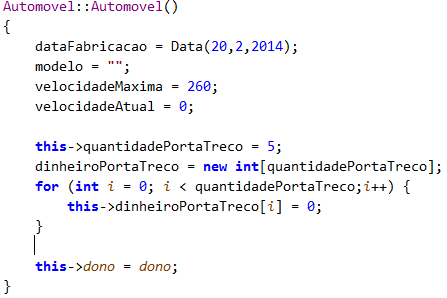


1. ~~Cada classe abstrata deve ter pelo menos um parâmetro, o construtor vazio e o construtor default. Deve ter também um método não virtual, que não pode ser set ou get;~~

Automóvel:

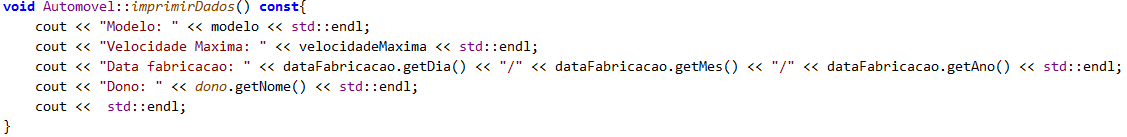
Parâmetros: Modelo, Velocidade Máxima, Velocidade Atual, Data de Fabricação, quantidade de porta trecos, dono, dinheiro em cada porta treco.

Construtor Vazio: (Automovel.cpp)



Construtor Default : (Automovel.h)C:\Users\Felipe\Desktop\Requesitos\AutomovelConstrutorDefault.PNG

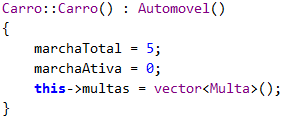
Método Não Virtual: (Automovel.cpp)



Carro:

Parâmetros: Quantidade Rodas, Marcha Total, Marcha ativa e vetor de multas.

Construtor Vazio: (Automovel.cpp)



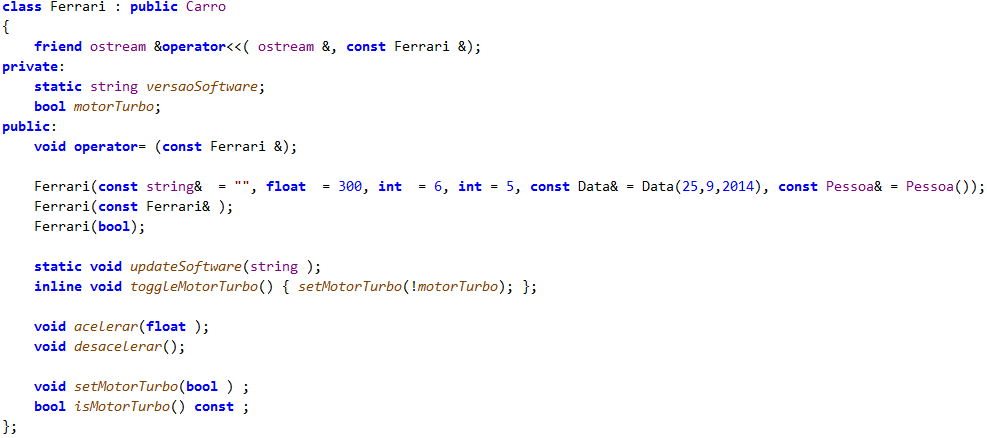
Construtor Default : (Automovel.h)C:\Users\Felipe\Desktop\Requesitos\CarroConstrutorDefault.PNG

Método Não Virtual: (Automovel.cpp)

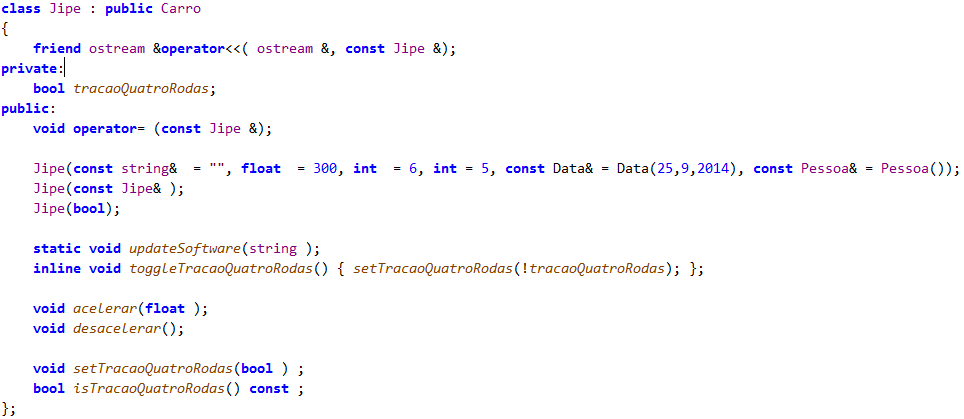
C:\Users\Felipe\Desktop\Requesitos\CarroAdicoinarMulta.PNG

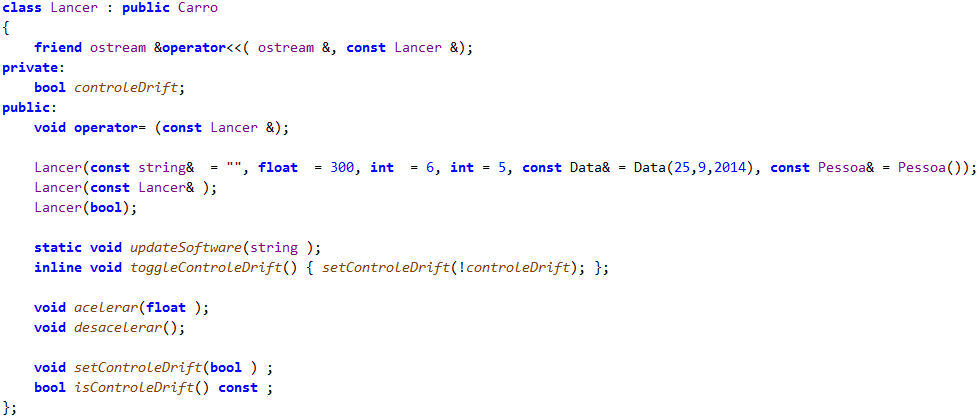
1. ~~Pelo menos três classes concretas na hierarquia de classes;~~

Ferrai:

~~~~

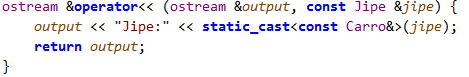
Jipe:

~~~~

Lancer:~~~~

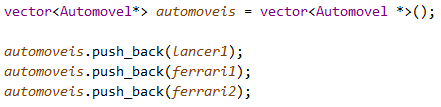
1. ~~Usar coerção de tipo C++ e não C, ou seja usar o~~ **~~static\_cast~~** ~~para fazer o máximo de reutilização de código.~~

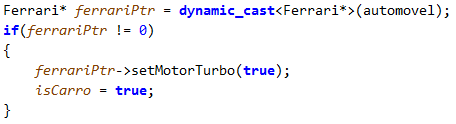
Exemplo de uso de static\_cast: (Jipe.cpp)



1. ~~Usar um vector de classes concretas, o~~ **~~dynamic\_cast~~** ~~e o~~ **~~typeid~~** ~~como indicado aqui:~~ [~~https://basecamp.com/2595605/projects/7018448/messages/33080741~~](https://basecamp.com/2595605/projects/7018448/messages/33080741)

Vetor (main.cpp)

~~~~

Dynamic\_cast(Main.cpp)~~~~

Typeid(Main.cpp)

C:\Users\Felipe\Desktop\Requesitos\typeId.PNG

1. ~~Criar uma função no arquivo do~~ **~~main~~**~~, que aceita um ponteiro da classe genérica e mostrar o seu uso para as classes concretas~~; ******

***Importante***: **Todos os Requisitos abaixo ainda devem ser feitos e serão avaliados**

***Todos os atributos e funções membros devem estar relacionados a classe***

1. Pelo menos 4 atributos
2. Pelo menos 4 funções membros sem incluir get e set

**Requisitos de implementação**

1. Todos os atributos devem ser inicializados. Fez validação de dados
2. Três construtores, incluindo um construtor de cópia e construtor com parâmetros defaults. Verifica alocação dentro do construtor de cópia.
3. Deve ter um atributo string
4. Um atributo static. Correta modelagem dos statics?
5. Um atributo const static
6. Dois métodos constantes (não pode ser get)
7. Um array
8. Uma função inline (não pode ser get ou set)
9. Método com passagem por referência usando ponteiro
10. Método static – deve ser chamado no main
11. Composição com a classe Data. Fez uso do objeto criado?
12. O que é const deve ser const.
13. Alocação dinâmica de memória. A memória é desalocada?
14. friend Operator<<
15. Operator=
16. vector push\_back

Requisitos para as classes adicionais (pelo menos duas)

* Operator =
* Alocação dinâmica - se houver vazamento de memória a classe toda é desconsiderada
* Usar o destrutor
* Construtor de cópia
* Operator << friend
* Um const static

**Requisitos herança**

* Diagrama de classes (obrigatório salvar também o png do diagrama no gitHub)
  + Herança pública
  + Construtor de cópia, e sobrecargas dos operadores de atribuição (=) e << (cout << base) para a classe base e derivada
  + Usar Protected acessando diretamente os atributos na classe derivada
  + Alocação dinâmica de memória na classe base e derivada
  + Sobrescrita de método: chamar dentro do método da classe derivada o método correspondente da classe base usando ::
  + No main: criar um ponteiro da classe base para alocar memória para a classe derivada e chamar os vários métodos implementados