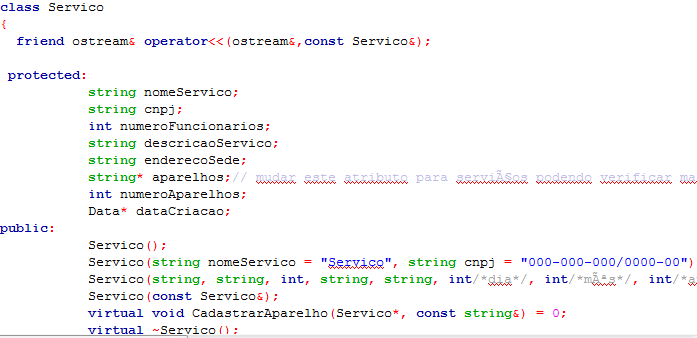
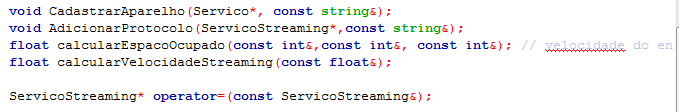
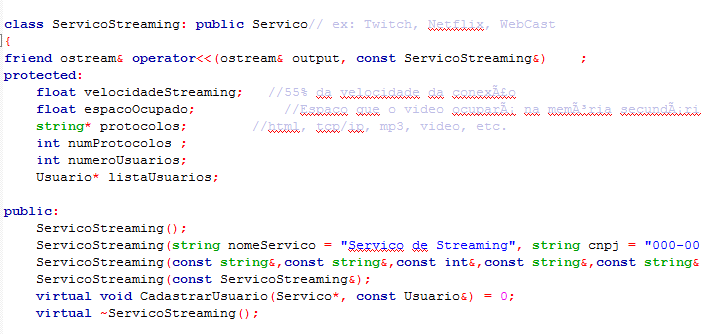
**Requisitos funcionais**

***Polimorfismo***

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

Clases: Servico.h abstrata03.PNG

Classe: ServicoStream.h



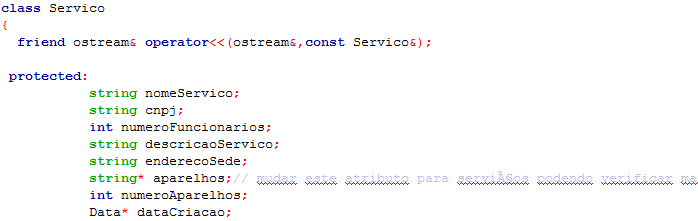
1. Pelo menos três classes concretas na hierarquia de classes;
2. Usar coerção de tipo C++ e não C, ou seja usar o **static\_cast** para fazer o máximo de reutilização de código.
3. Usar um vector de classes concretas, o **dynamic\_cast** e o **typeid** como indicado aqui: <https://basecamp.com/2595605/projects/7018448/messages/33080741>
4. 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***

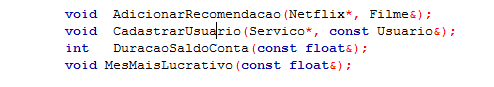
1. Pelo menos 4 atributos

Classes: Servico.h



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

Classe: Netflix.h



**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

Classe: Servico.h

03.PNG

1. Um atributo static. Correta modelagem dos statics?
2. Um atributo const static

Classe: Netflix.h

05.PNG

1. Dois métodos constantes (não pode ser get)

Classe: ServicoStreaming.h

04.PNG

Classe: Servico.h

06.PNG

1. Um array

Classe: ServicoStreaming.h

07.PNG

1. Uma função inline (não pode ser get ou set)

Classe: Servico.h

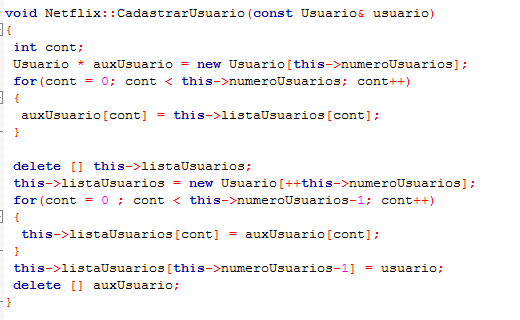
08.PNG

1. Método com passagem por referência usando ponteiro

Classe: Usuario.h

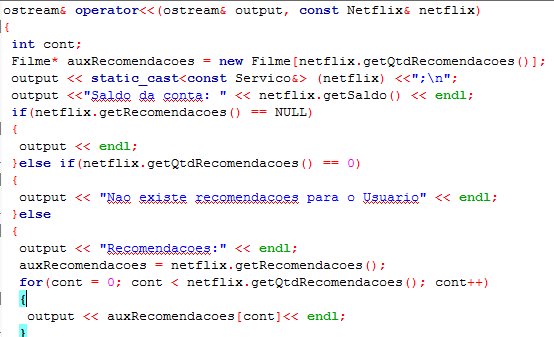
10.PNG

1. Método static – deve ser chamado no main
2. Composição com a classe Data. Fez uso do objeto criado?
3. O que é const deve ser const.
4. Alocação dinâmica de memória. A memória é desalocada?



1. friend Operator<<

Classe: Netflix.cpp



1. Operator=

Classe: Netflix.cpp

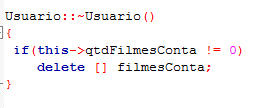


1. vector push\_back

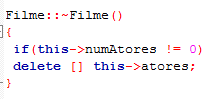
Requisitos para as classes adicionais (pelo menos duas)

* Operator =

Classe: Usuario.cpp

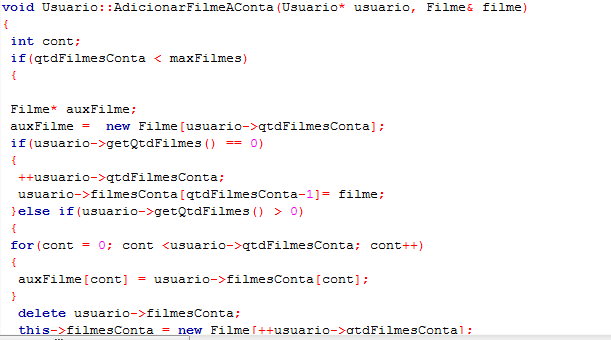


Classe: Filme.cpp

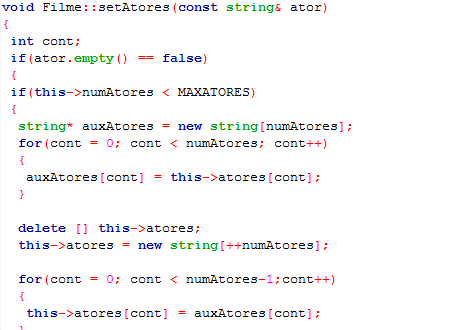


* Alocação dinâmica - se houver vazamento de memória a classe toda é desconsiderada

Classe:Usuario.cpp

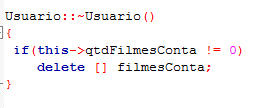


Classe: Filme.cpp

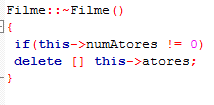


* Usar o destrutor

Classe: Usuario.cpp

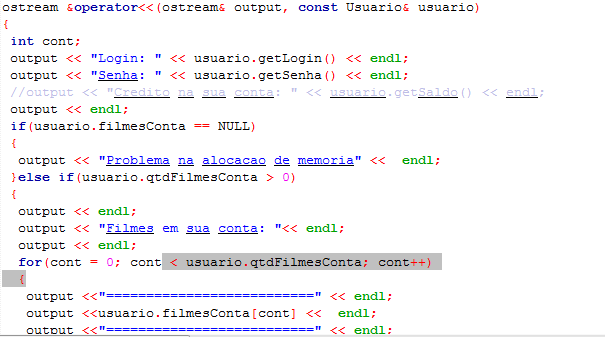


Classe: Filme.cpp

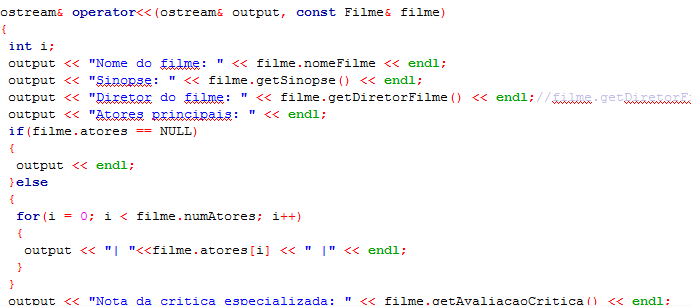


* Construtor de cópia
* Operator << friend

Classe: Usuario.cpp



Classe: Filme.cpp



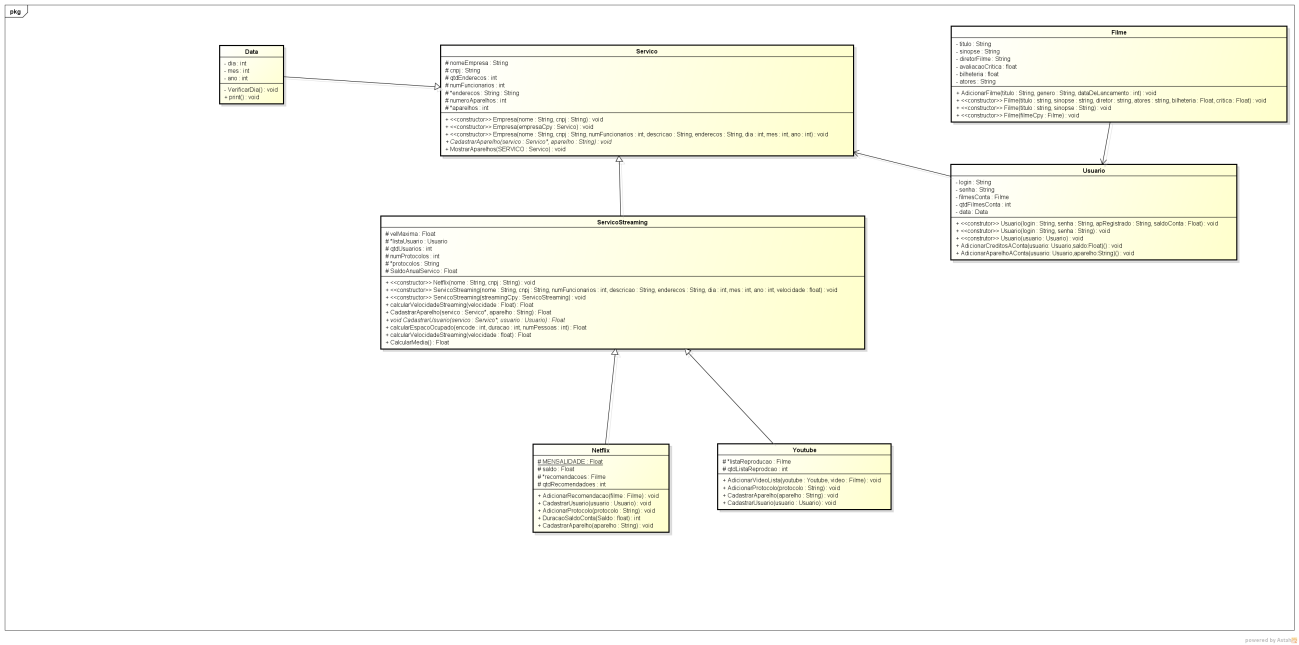
* Um const static

Classe: Usuario.h

17.PNG

**Requisitos herança**

* Diagrama de classes (obrigatório salvar também o png do diagrama no gitHub)

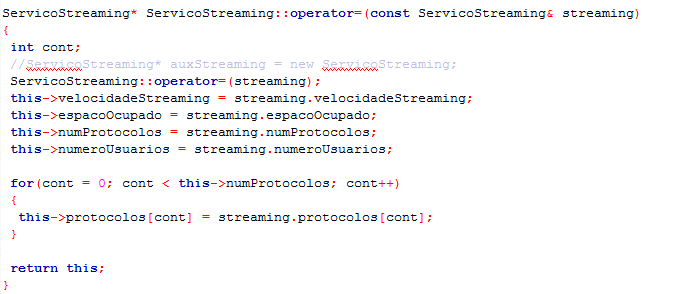
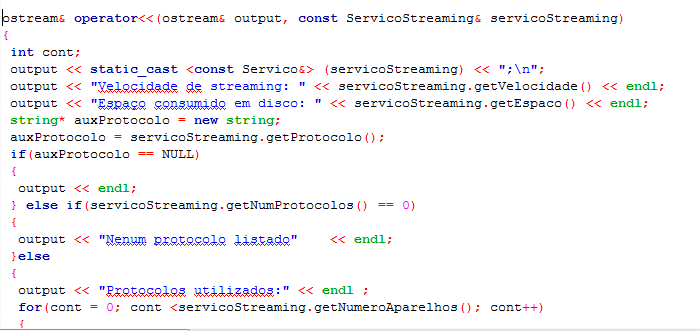


* + Herança pública

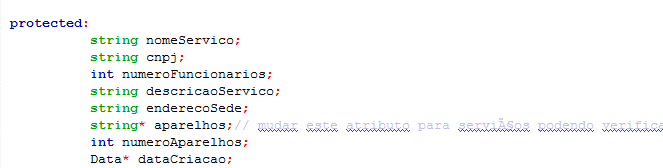
Classe: ServicoStreaming.h

* 22.PNG
  + Construtor de cópia, e sobrecargas dos operadores de atribuição (=) e << (cout << base) para a classe base e derivada

Classe: ServicoStreaming.cpp

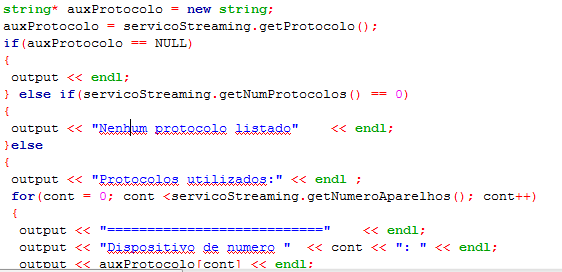
* 
  + Usar Protected acessando diretamente os atributos na classe derivada

Classe : Servico.h



* + Alocação dinâmica de memória na classe base e derivada

Classe:ServicoStreaming.cpp

* 
  + 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