***Importante***: Para cada 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 arquivo da classe em que está o trecho do código. 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.**

**Criar repositórios desde o início do desenvolvimento. Primeiro commit deve estar com todas as classes declaradas sem implementação.**

**Primeira versão do código completa dia 13 de abril – a ser avaliado em sala de aula.**

**A partir das correções, entregar versão final do código dia 20 de abril.**

**No dia da apresentação deve ser mostrado o código rodando para vários casos de teste.**

**Será avaliado o quanto o projeto está elaborado. Trabalhos muito simples com métodos apenas mostrando mensagens, serão penalizados.**

**Requisitos de implementação para C++**

1. Todas as classes concretas devem vir de classes abstratas. Pelo menos três hierarquias de classes. Uma das hierarquias deve ter três níveis. Exemplo: Personagem (abstract) >> Ciborgue (Abstract) >> Robocop; Class Arma (Abstract) >> Beretta93R
2. Em todas as classes: **construtor de cópia**, **operatores<< e +=,** e **construtor default**. Fazer o máximo de reaproveitamento de código usando **static\_cast**
3. Todas as hierarquias devem ter classes Concretas, e em uma das hierarquias, três classes Concretas relacionadas: Exemplo Servico >> ServicoStream >> (Netflix, HBOStream, AmazonPrime, NowTv). Em uma pesquisa de 10 segundos: http://www.tomsguide.com/us/pictures-story/620-top-online-streaming-video.html
4. Atributos static e const static em todas as hierarquias de classe
5. Método static em todas as hierarquias de classe
6. Construtores em todas as classes, e três para todas as classes da hierarquia principal. Sempre validar os dados em todas as classes
7. Vector em todas em todas as hierarquias de classe
8. ENUM na hierarquia principal
9. Usar o **dynamic cast** e **typeid** no main junto com as classes concretas. Para uma da classe concreta identificada, chamar um método dessa classe e fazer uma ação;
10. Usar o rand. Nota: deve ser usado conforme o contexto do projeto. Se for usado em um método genérico sem relação com a classe e apenas para cumpri-lo, esse requisito será desconsiderado. http://en.cppreference.com/w/cpp/numeric/random/rand
11. No main o usuário deve fazer entrada via teclado e interagir com a aplicação

**Requisitos de implementação para Java**

1. Todas as classes concretas devem vir de interfaces ou classes abstratas. Pelo menos três hierarquias de classes. Uma das hierarquias deve ter três níveis. Exemplo: Personagem >> Ciborgue >> Robocop; Class Arma (interface) >> Beretta93R
2. Ao menos três interfaces. A terceira interface deve ser uma interface que liga duas hierarquias como no exemplo da interface **corredor** (Figura 1).
3. Usar a interface **Comparable** e sobrescrever o método **compareTo** em pelo menos uma hierarquia
4. Sobrescrever **equals** para de Object
5. Todas as hierarquias devem ter classes Concretas, e em uma das hierarquias, três classes Concretas relacionadas: Exemplo Servico >> ServicoStream >> (Netflix, HBOStream, AmazonPrime, NowTv). Em uma pesquisa de 10 segundos: http://www.tomsguide.com/us/pictures-story/620-top-online-streaming-video.html
6. Sempre usar o super para o máximo de reaproveitamento de código em todas as classes
7. Atributos static e const static em todas as hierarquias de classe
8. Método static em todas as hierarquias de classe
9. Três construtores para todas as classes, sendo um deles o construtor de cópia. Sempre validar os dados em todas as classes
10. ArrayList na hierarquia principal
11. ENUM na hierarquia principal
12. Usar o **instanceof** no main junto com as classes concretas. Para uma da classe concreta identificada, chamar um método dessa classe e fazer uma ação;
13. Dividir o projeto em pacotes
14. Sobrescrever para todas as classes o método toString
15. Usar a classe Random do pacote java.util (java.util.Random). Nota: deve ser usado conforme o contexto do projeto. Se for usado em um método genérico sem relação com a classe e apenas para cumpri-lo, esse requisito será desconsiderado.
16. No main o usuário deve fazer entrada via teclado e interagir com a aplicação. Opcional de bônus: pode ser usada a classe JOptionPane do pacote javax.swing. Vejam: showInputDialog e showMessageDialog.



Figura 1 – Interface Corredor conectando duas hierarquias de classe