***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.**

**Usar um novo repositório!**

**Entrega final: 26 de novembro via GitHub. Prova oral sobre o código ainda a ser definida.**

**Entrega parcial: 19 de novembro via GitHub. Apenas Diagrama de Classes, interfaces e classes abstratas no branch general.**

**Requisitos Gerais**

***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
3. Diagrama UML completo (obrigatório salvar também o png do diagrama no gitHub)

**Requisitos de implementação**

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
7. Atributos static e const static
8. Método static
9. Construtores em todas as classes, e dois na hierarquia principal. Sempre validar os dados em todas as classes
10. Construtor cópia em uma das hierarquias
11. ArrayList
12. ENUM
13. 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;
14. Dividir o projeto em pacotes
15. Sobrescrever para todas as classes o método toString
16. 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.
17. 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