***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 01 de dezembro.**

**Versão final do código dia 08 de dezembro.**

**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 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

Principais Classes abstratas do projeto: Carta e ConjuntoDeCartas.

1. Ao menos três interfaces. A terceira interface deve ser uma interface que liga duas hierarquias.

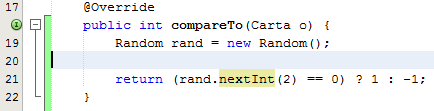
Interfaces: IAgua, IFogo, INeutro, ITerra, TipoM, que são utilizadas para identificar o tipo do monstro durante a batalha.

1. Usar a interface **Comparable** e sobrescrever o método **compareTo** em pelo menos uma hierarquia

Implementado na classe Carta.

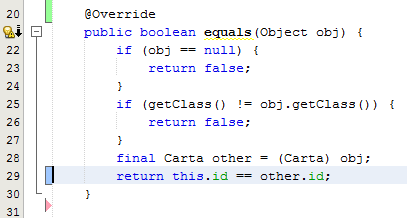


O método compareTo foi implementado da ordenar de maneira aleatória as cartas no inicio de cada batalha



1. Sobrescrever **equals** para de Object

Equals sobrescrito na classe Carta(Compara o id das cartas, pois nenhuma carta tem id repetido.



1. Todas as hierarquias devem ter classes Concretas, e em uma das hierarquias, três classes Concretas relacionadas: Exemplo Servico >> ServicoStream >> (Netflix, HBOStream, AmazonPrime, NowTv).

Na hierarquia Carta >> Monstro >> Monstro(TipoDoMonstro) têm 5 classes concretas na base.

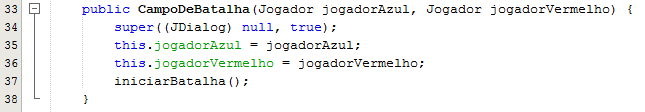
1. Sempre usar o super para o máximo de reaproveitamento de código em todas as classes

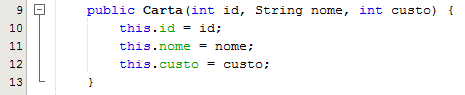
O super foi utilizado na maioria dos construtores.

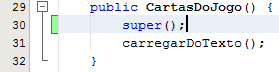
1. Atributos static e const static em todas as hierarquias de classe\
2. Método static em todas as hierarquias de classe
3. Três construtores para todas as classes, sendo um deles o construtor de cópia. Sempre validar os dados em todas as classes

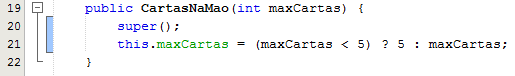
Nem todas as classes necessitavam de 3 construtores, ou um construtor de cópia, e escreve-los iria ser desnecessário, por esse motivo nem todas as classes tem três construtores.

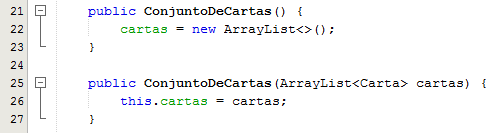
Construtor da classe CampoDeBatalha:

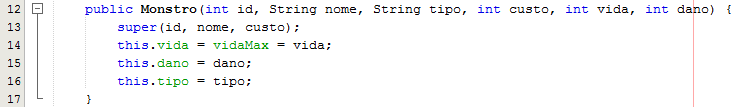


Construtor de Carta: 

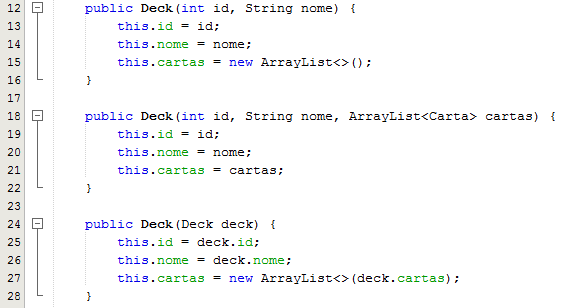
Construtor de CartasDoJogo(Classe Responsável por carregar as cartas do txt): 

Construtor de CartasNaMao(Classe que agrega as cartas na mao do jogador): 

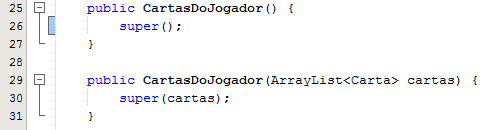
Construtores de ConjuntoDeCartas: 

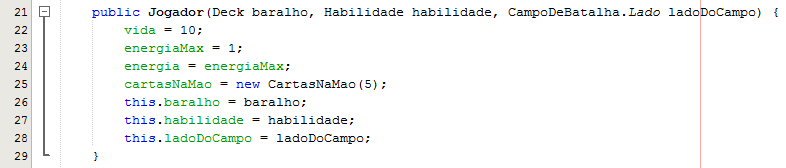
Construtor de Monstro: 

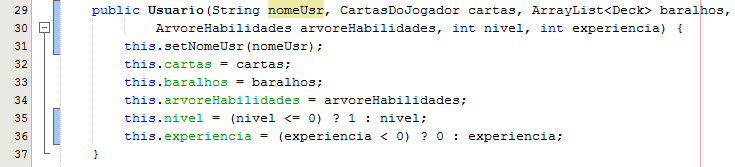
Construtor de Magia: 

Construtores de Deck: 

Construtor de MonstroAgua: 

Construtores de CartasDoJogador: 

Construtor de Jogador: 

Construtor de Usuario: 

1. ArrayList na hierarquia principal

ArrayList em ConjuntoDeCartas:

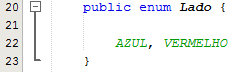
Usado em todos os conjutos de cartas: Cartas do Jogo, Cartas do Jogador, Deck, Cartas na Mão.



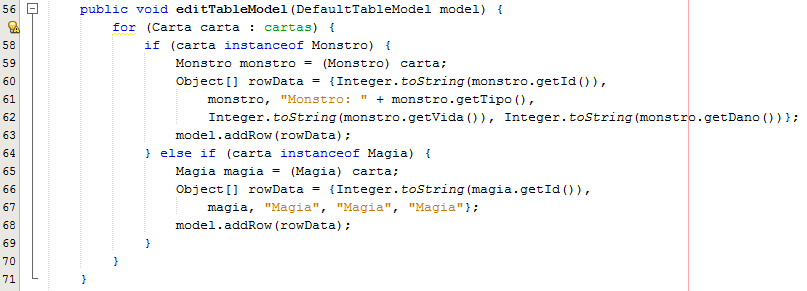
1. ENUM na hierarquia principal

Enum na classe CampoDeBatalha

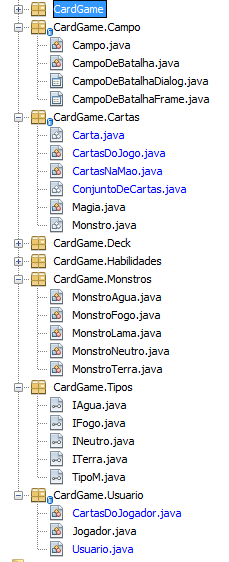
Enumera os lados do campo(Azul e vermelho):



1. 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;

Instanceof não foi utilizado no Main, entretanto foi utilizado na classe Conjunto de cartas, no método que preenche as tabelas com as cartas, ele checa para saber se a carta é um monstro ou uma magia.

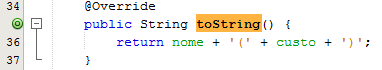
1. Dividir o projeto em pacotes



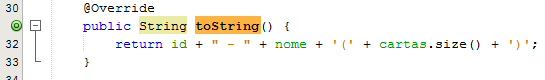
1. Sobrescrever para todas as classes o método toString

Não foi necessário sobrescrever o método toString em todas as classes, pois ele seria inútil na maioria delas, logo, os que foram sobrescritos serão listados abaixo:

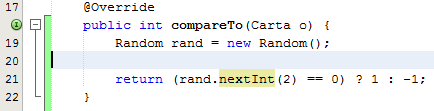
Método toString da Classe Carta:



Método toString da Classe Deck:



1. 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.

Usado no Método compareTo de Carta para ordenar as cartas de maneira aleatória, usado no inicio de cada batalha, a reordenação aleatória de cada baralho: 

1. 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.

No projeto foram usados JFrames e JDialogs, pois achei uma escolha mais sensata em detrimento do JOptionPane.

Abaixo são apresentadas as imagens dos JFrames/JDialogs que foram utilizados.

