Universidade Estadual de Maringá

Centro de Tecnologia - Departamento de Informática

**Disciplina 5185**

**Paradigma de Programação Imperativa e Orientada a Objetos**

**Documentação**

Game DC Heroes

Rafael Soares Cé - RA56077

Renato Henrique Silva - RA57667

Maringá, PR

Novembro de 2014

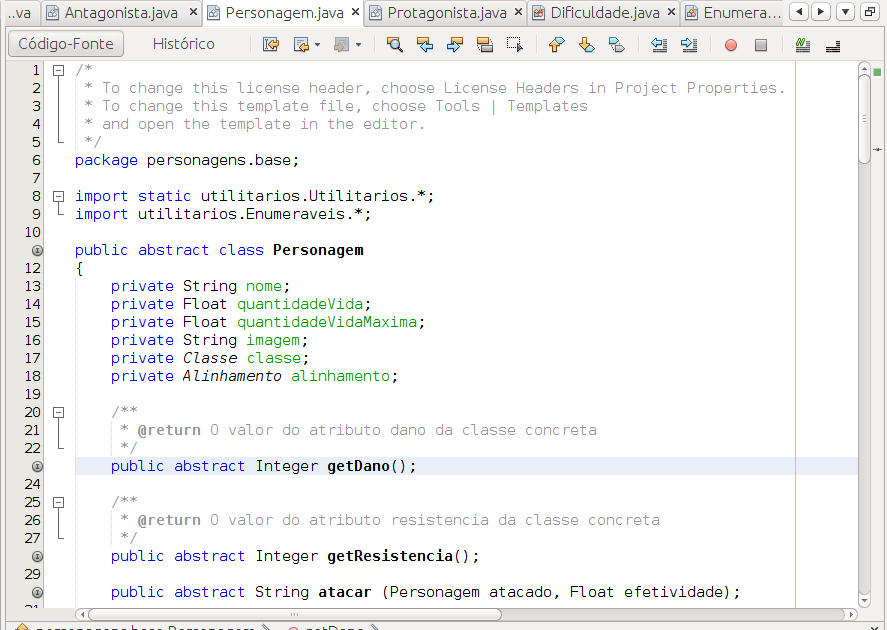
Documentação - Game DC Heroes

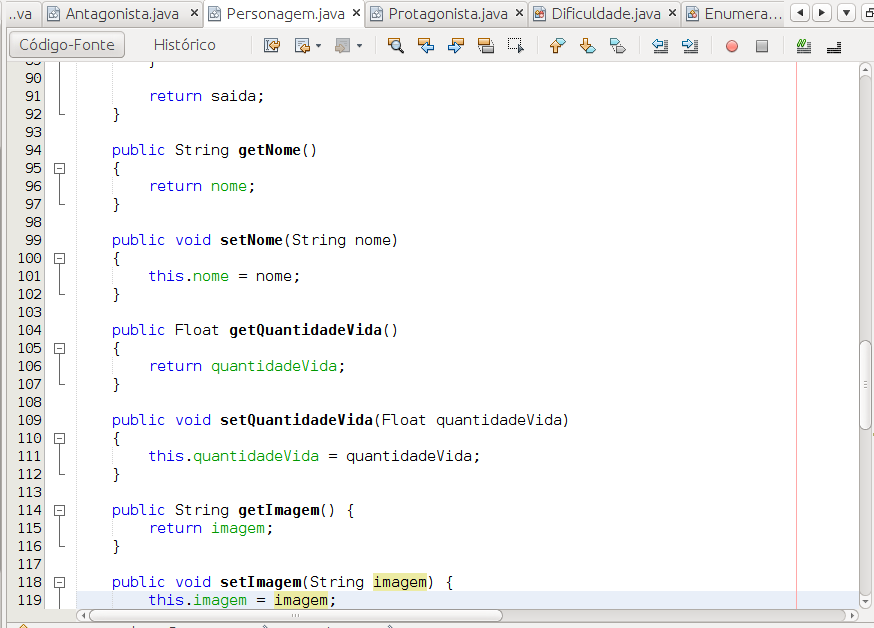
1. **Passo a Passo para compilação e execução do programa**

O trabalho em questão foi desenvolvido usando Java 1.8+ com o auxilio da IDE NetBeans 8.0.1. O projeto enviado junto a essa documentação pode ser aberto na referida versão do NetBeans com o Java Developer Kit (JDK) 1.8+ instalada no computador. Como não usa nenhuma biblioteca em especial, sua compilação e execução é simples a partir da tecla de atalho F6 dentro do NetBeans.

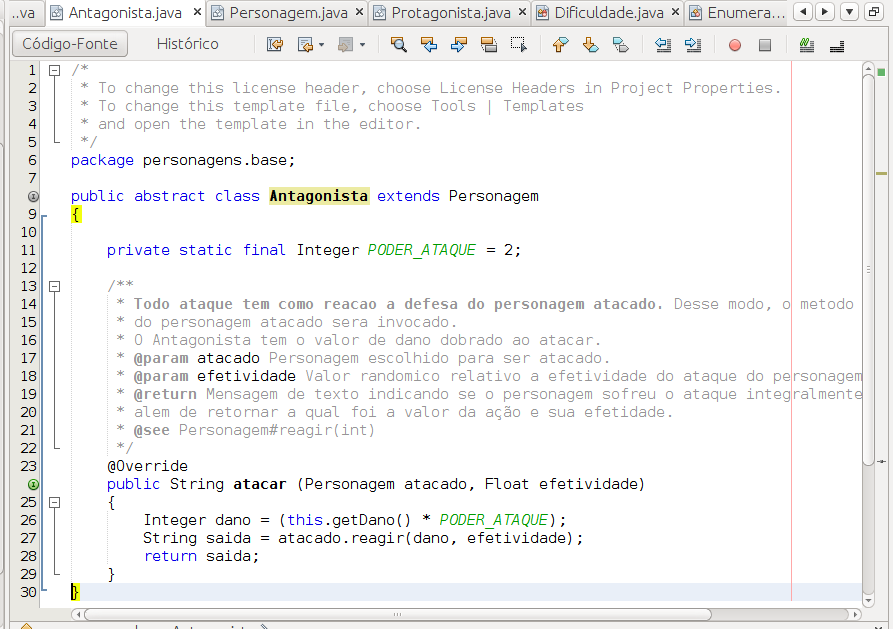
**2) Conceitos de Orientação a Objetos**

**2.1) Encapsulamento**

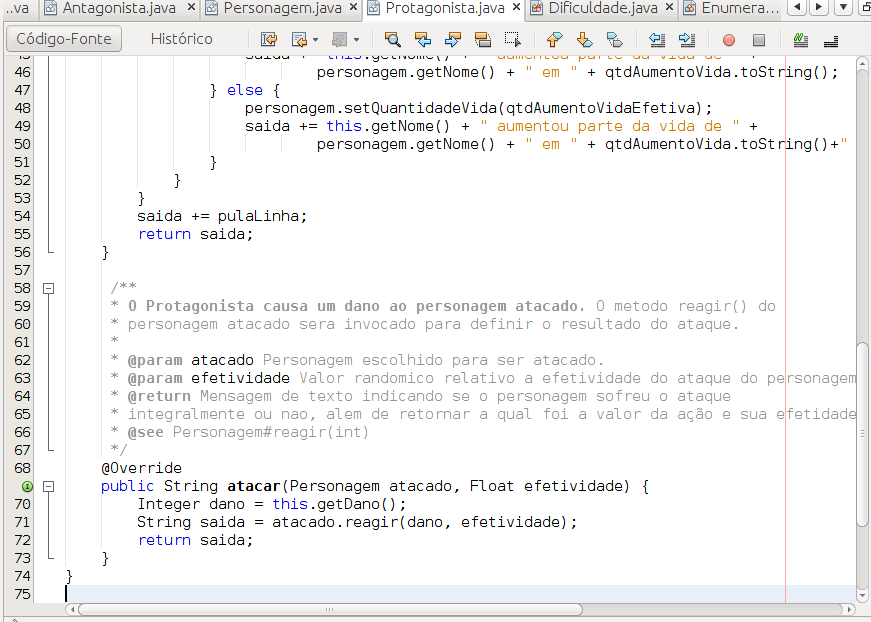
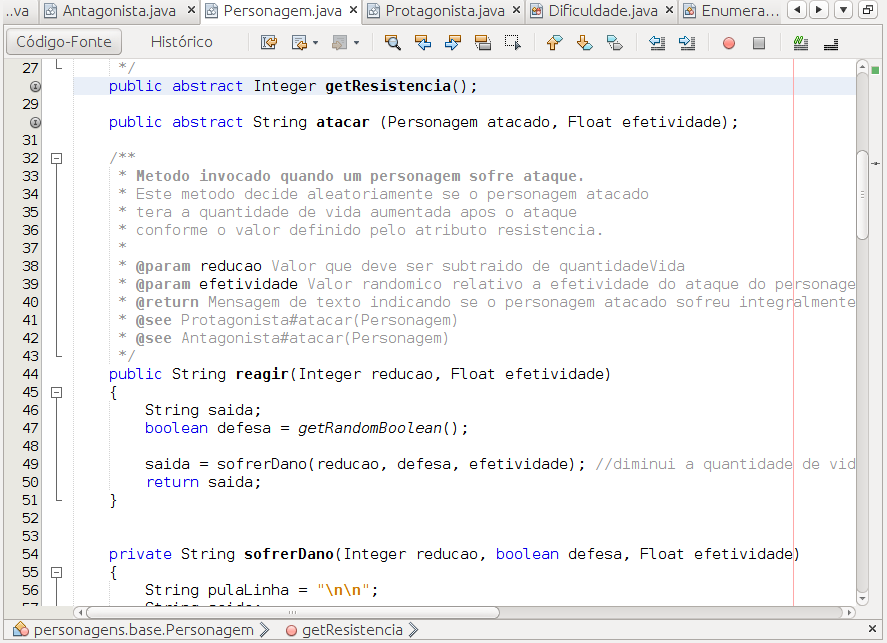
Encapsulamento em programação orientada a objetos é ocultar a representação real dos dados, deixando inacessível diretamente e então agrupando dados e suas operações em uma unidade independente. Exemplo: Atributos da classe Personagem com modificador private.

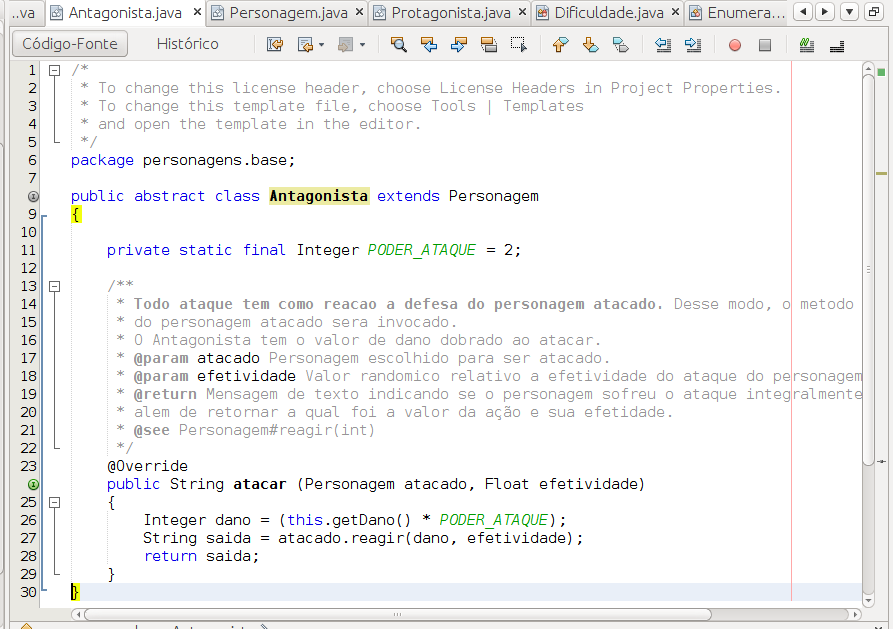
Os atributos não são diretamente acessíveis, mas possuem métodos getters e setters com modificador public dentro da classe Personagem. ****

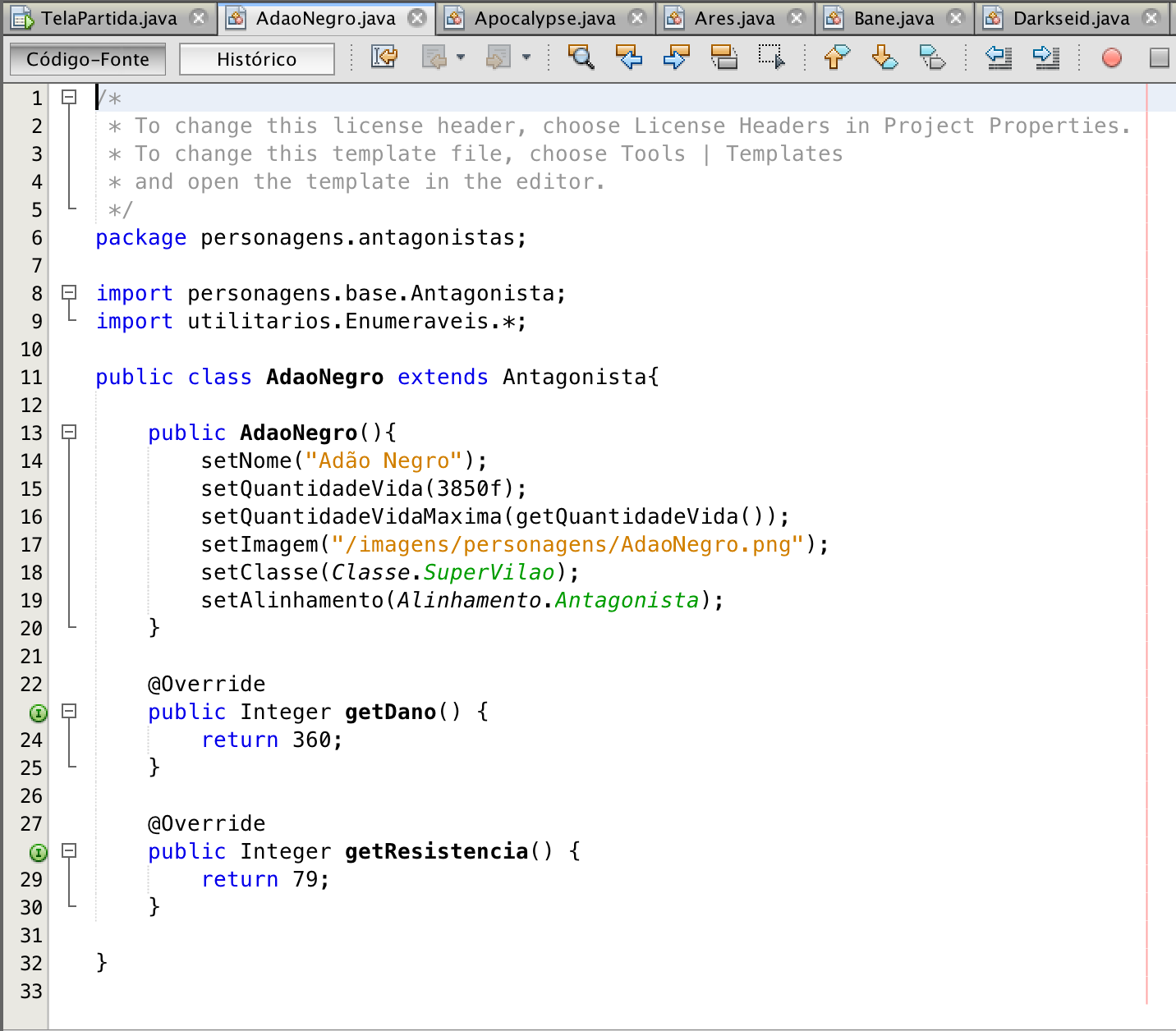
**2.2) Herança**

Herança é um princípio da orientação a objetos onde novas classes (chamadas de subclasses ou ainda classes filhas) herdem tanto atributos quanto métodos de outra classe (chamada de super classe ou então classe pai). Exemplo Antagonista é um Personagem, herdando todos os seus atributos e métodos.

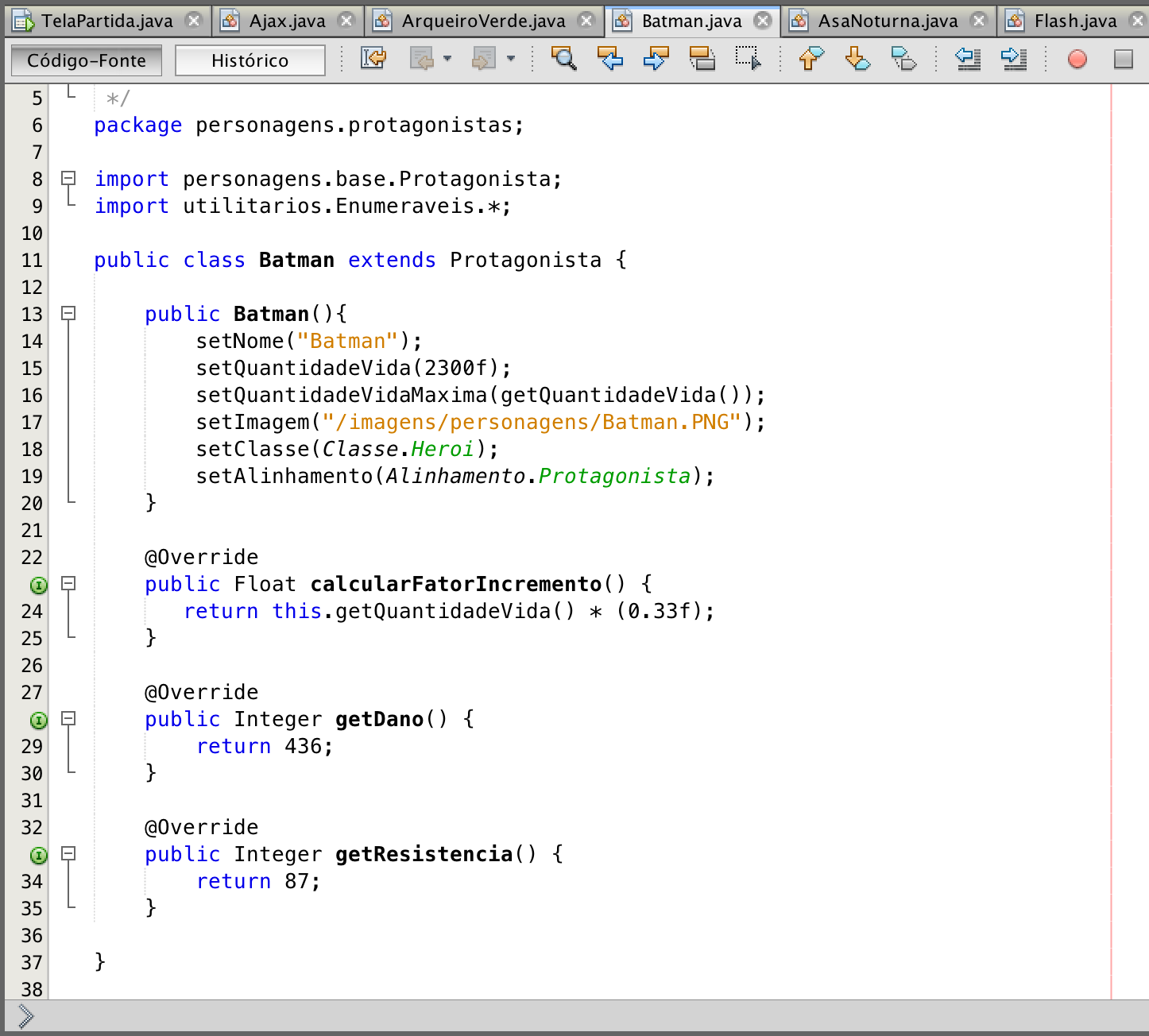
**2.3) Polimorfismo**

Polimorfismo é diretamente relacionado a herança. Uma subclasse contem implicitamente todos os atributos e métodos de sua superclasse, que é uma classe mais abstrata, as subclasses concretas com implementações diferentes dos mesmos métodos representem diferentes comportamentos da classe abstrata que referencia. Exemplo: Um Personagem tem um ataque, contudo o ataque no Antagonista é dobrado, diferenciando do Protagonista, porém tanto Protagonista como Antagonista são Personagens, e no projeto aqui apresentado existem 15 classes concretas de Antagonista e mais 15 classes concretas de Protagonista.



Exemplo de uma classe concreta de Antagonista.

Exemplo de uma classe concreta de Protagonista:



**3) Decisões de Projeto**

**3.1) Funcionamento geral do jogo**

O jogo quando é iniciado, apresenta uma splash screen com a logo *DC Heroes,*  representando a temática do jogo, onde os personagens envolvidos são os famosos personagens das historias em quadrinho da editora norte americana *DC Comics*. A primeira tela apresentada solicita ao usuário que digite seu nome, a fim de diferenciar as equipes (Jogador x Computador).

Logo em seguida é apresentada a tela onde o Jogador seleciona os personagens que irá fazer parte da sua equipe no jogo.

Ao clicar em adicionar personagem é apresentado uma interface onde é possível selecionar entre os personagens disponíveis, sendo 30 diferentes personagens, onde 15 são protagonistas e 15 antagonistas. Personagens protagonistas, além do ataque, tem poder de cura, ou seja, incrementam a vida de qualquer outro personagem da mesma equipe, por sua vez os antagonistas tem poder de ataque dobrado. Tanto protagonistas como antagonistas sabem se defender de ataques sofridos, isso é uma decisão randômica e feita em tempo de execução. A tela de seleção de personagem apresenta todos os dados sobre os personagens, diferenciando também visualmente na apresentação os protagonistas em azul e os antagonistas em vermelho.

Selecionado o(s) personagem(s), é solicitado o nível de dificuldade da partida, sendo eles: Fácil, Normal, Difícil e Insano. Eles se diferenciam na quantidade de vida aplicada aos personagens da equipe do Computador. Selecionando fácil, a vida de todos os personagens da equipe do Computador será reduzida há 80% do total, normal não há alteração na quantidade de vida, difícil é feito um acréscimo de 50% e por fim no nível insano a vida de todos os personagens do Computador é dobrada.

Então é iniciado o jogo com o primeiro turno para a equipe do Jogador, nesse momento já foi gerado uma equipe para o Computador com a mesma quantidade de personagens da equipe do Jogador e com as mesmas proporções de protagonistas e antagonistas. Na tela é possível selecionar qual personagem irá executar a ação e qual a ação será aplicada (atacar ou curar), obedecendo as restrições anteriormente citadas. Dependendo da ação é possível escolher um personagem a ser atacado(equipe adversária) ou curado(própria equipe). Quando é executada uma jogada, existe uma efetividade da jogada em questão, que é definida de forma randômica e aplicada sobre o ação. Essa efetividade varia de 0 a 1, exemplo: Adão Negro possui dano de 360 e está atacando o personagem adversário Zod que tem quantidade de vida de 4900. Se o ataque for com efetividade de 0.8, o dano será reduzido a 288 portanto, deixando Zod com vida de 4612 (nesse caso não houve defesa do personagem), em caso de efetividade igual a 1, é feito um dano dobrado, chamado de Dano Crítico.

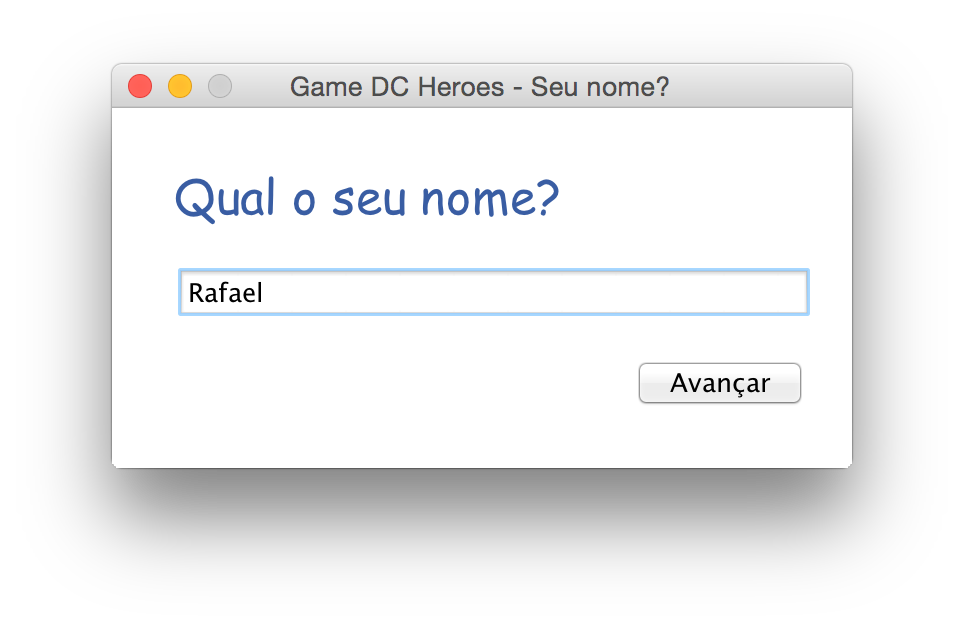
Após o turno do Jogador vem o turno do Computador, onde tudo é selecionado de forma randômica e apresentado ao usuário, bastando então clicar em jogar para ver os resultados. O jogo acaba quando uma equipe não tiver mais personagens.

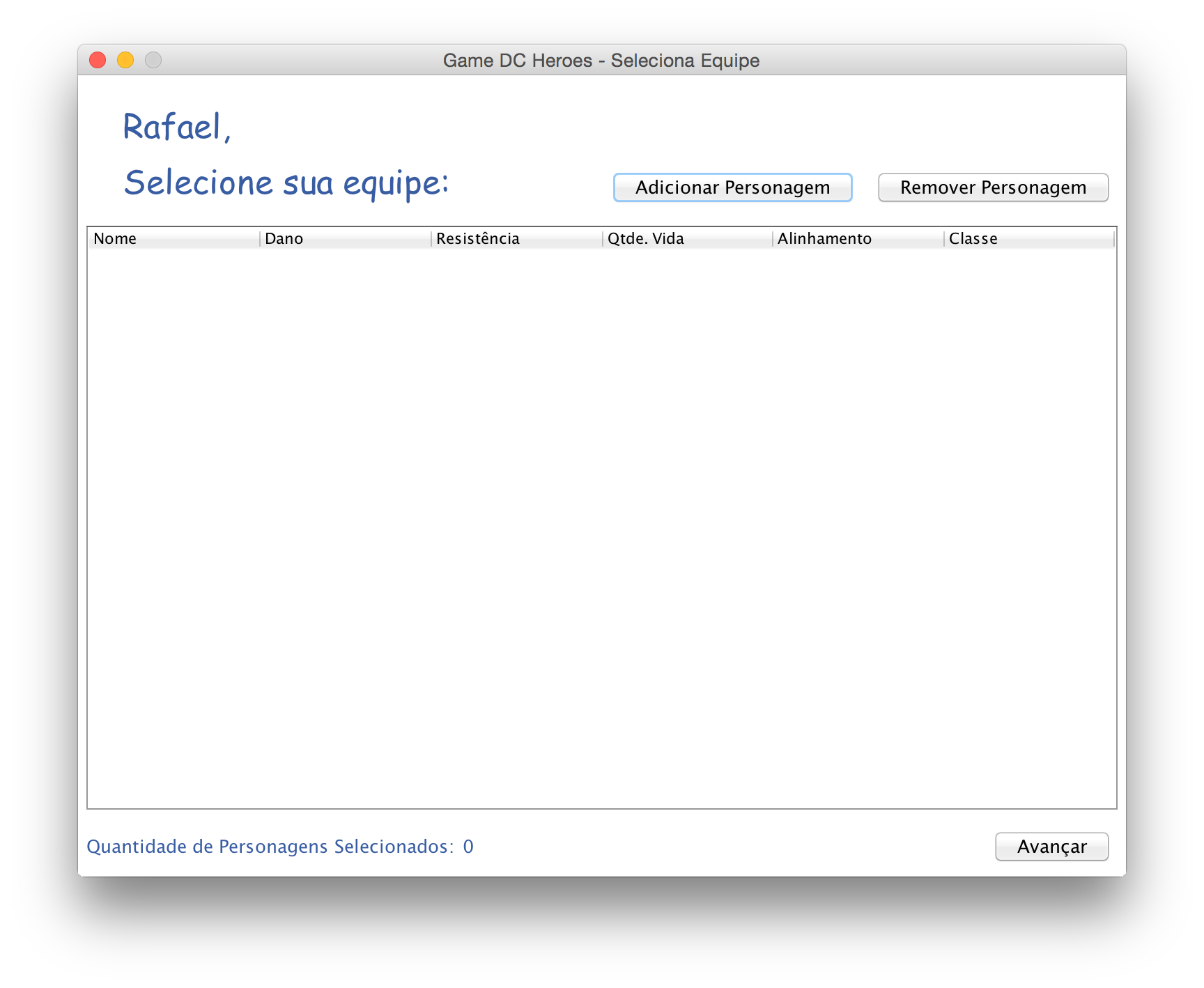
**3.2) Classes controladoras implementadas**

**PersonagemControle:** com diversos métodos implementados que fazem referencia ao personagens. Por exemplo, getPersonagensPorAlinhamento(Alinhamento) retorna um list de com todos os personagens do alinhamento recebido por parâmetro.

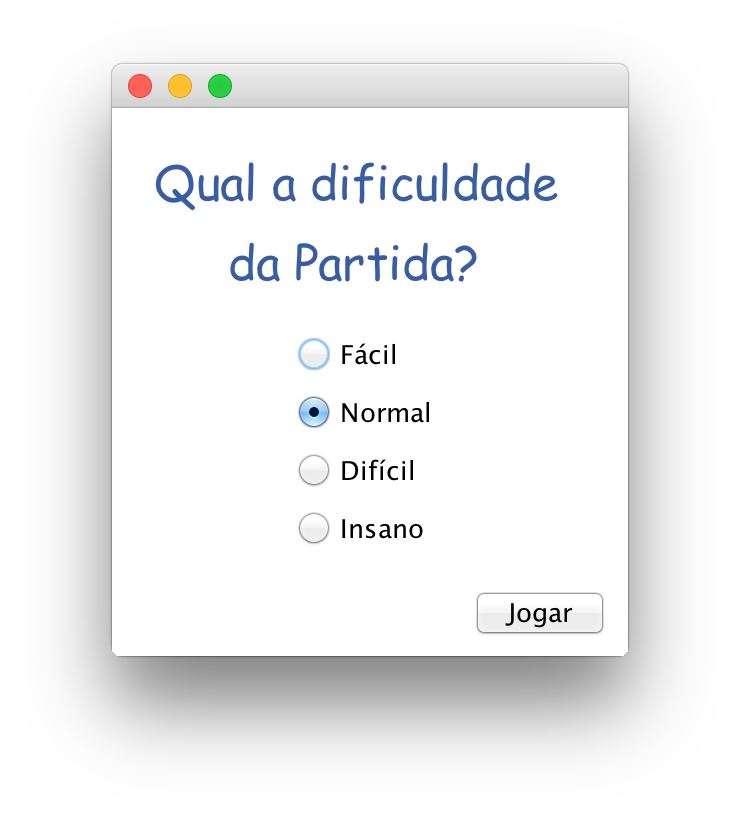
**EquipeControle:** possui um metodo de geração da equipe do Computador com base na esquipe selecionada pelo Jogador.

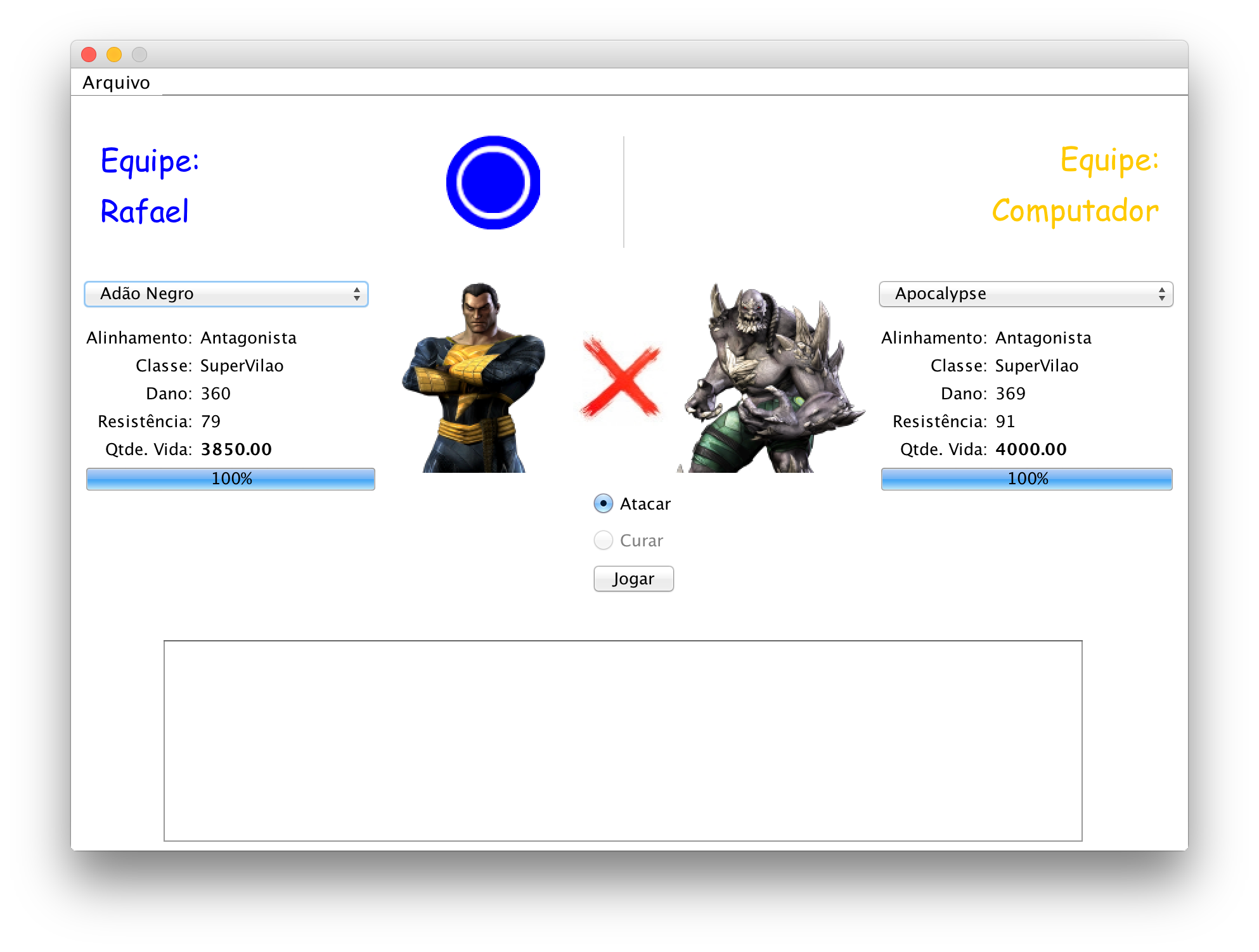
**3.3) Classes de interação com o usuário**

Para a melhor interação com o usuário no jogo, foi utilizado a toolkit Swing para Java, possibilitando a criação de interfaces gráficas.

****

****

****

****

**3.4) Classes concretas**

**GameDCHeroes:** contem o método main() do jogo e mostra a splash screen inicial.

**Jogador:** classe com métodos e atributos dos jogadores da partida.

**Equipe**: atributos e metodos para se formar uma equipe, no caso um Jogador e os personagens escolhidos.

**Partida:** formação de uma partida que inclui duas equipes, Jogador e Computador.

**Pacote personagens.antagonistas:** contem 15 classes concretas de personagens antagonistas com seus diferentes valores de atributos e métodos, sendo as classes: AdaoNegro, Apocalipse, Ares, Arlequina, Bane, Coringa, Darkseid, Exterminador, LexLuthor, Lobo, MulherGato, Nevasca, Sinestro, SolomonGrundy e Zod.

**Pacote personagens.protagonistas:** contem 15 classes concretas de personagem protagonistas com seus respectivos valores de atributos e métodos, sendo as classes: Ajax, Aquaman, ArqueiroVerde, AsaNoturna, Batman, Cyborg, Flash, LanternaVerde, MulherGaviao, MulherMaravilha, Ravena, Shazam, SuperHomem e Zatanna.

**Utilitarios:** possui métodos auxiliares a todo o decorrer do jogo.

**3.5) Enumeráveis**

**Alinhamento:** *Protagonista, Antagonista*

**Classe:** *Heroi, Mago, SuperHeroi, Vilao, Bruxo, SuperVilao*

**TipoJogador:** *Humano, Computador*

**Dificuldade:** *FACIL, NORMAL, DIFICIL, INSANO*

**3.6) Organização das classes em pacotes**

No decorrer do projeto foram criados 10 pacotes para as classes:

**gamedcheroes**: classes que envolvem o Game DC Heroes, tendo uma classe com o mesmo nome do pacote que contêm o método main() do projeto. Possui também classes de Jogador, Equipe e Partida.

**gamedcheroes.controle**: controles necessários as classes do pacote gamedcheroes.

**imagens**: pacote exclusivo para imagens apresentadas no jogo.

**imagens.personagens**: pacote com imagens dos personagens do jogo.

**personagens.base**: contem classes bases abstratas de personagem.

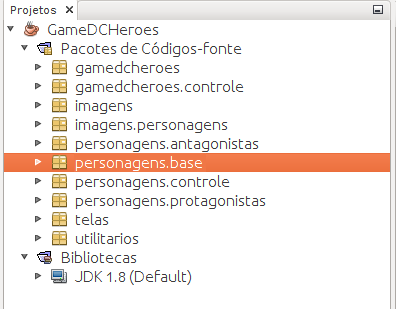
**personagens.antagonistas**: pacote com todas as classes concretas de antagonista.

**personagens.protagonistas**: pacote com todas as classes concretas de protagonista.

**personagens.controle**: pacote criado para a classe de controle aos personagens.

**telas**: classes de interfaces gráficas para interação com o usuário.

**utilitarios**: classes com métodos utilitários e os enumeráveis usados no decorrer do jogo.

****