

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

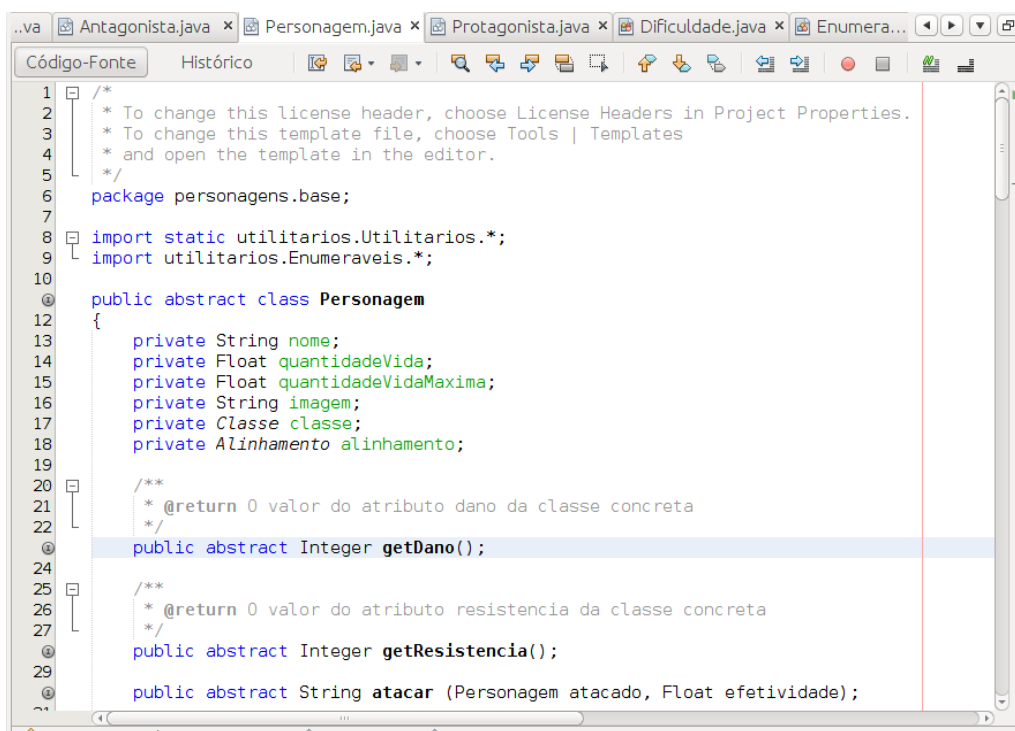
1) Passo a Passo para compilação e execução do programa

O trabalho em questão foi desenvolvido usando Java 1.8+ com o auxílio 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

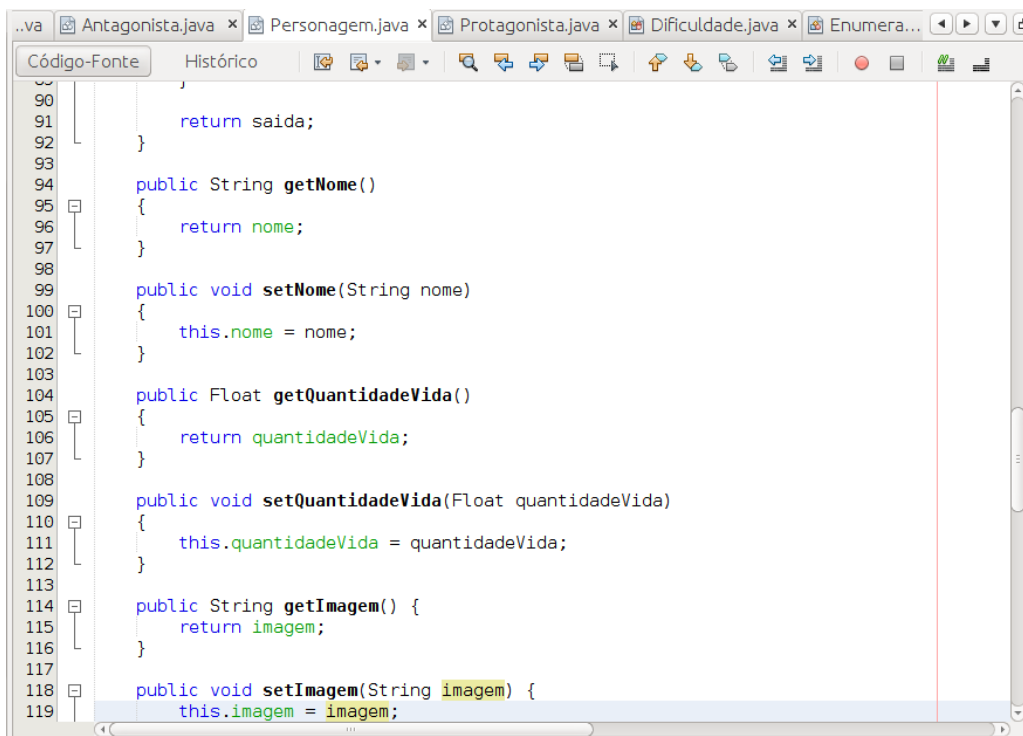
2.1) Encapsulamento

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.



```
1  /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package personagens.base;
7
8  import static utilitarios.Utilitarios.*;
9  import utilitarios.Enumeraveis.*;
10
11 public abstract class Personagem
12 {
13     private String nome;
14     private Float quantidadeVida;
15     private Float quantidadeVidaMaxima;
16     private String imagem;
17     private Classe classe;
18     private Alinhamento alinhamento;
19
20     /**
21      * @return 0 valor do atributo dano da classe concreta
22      */
23     public abstract Integer getDano();
24
25     /**
26      * @return 0 valor do atributo resistencia da classe concreta
27      */
28     public abstract Integer getResistencia();
29
30     public abstract String atacar (Personagem atacado, Float efetividade);
31 }
```

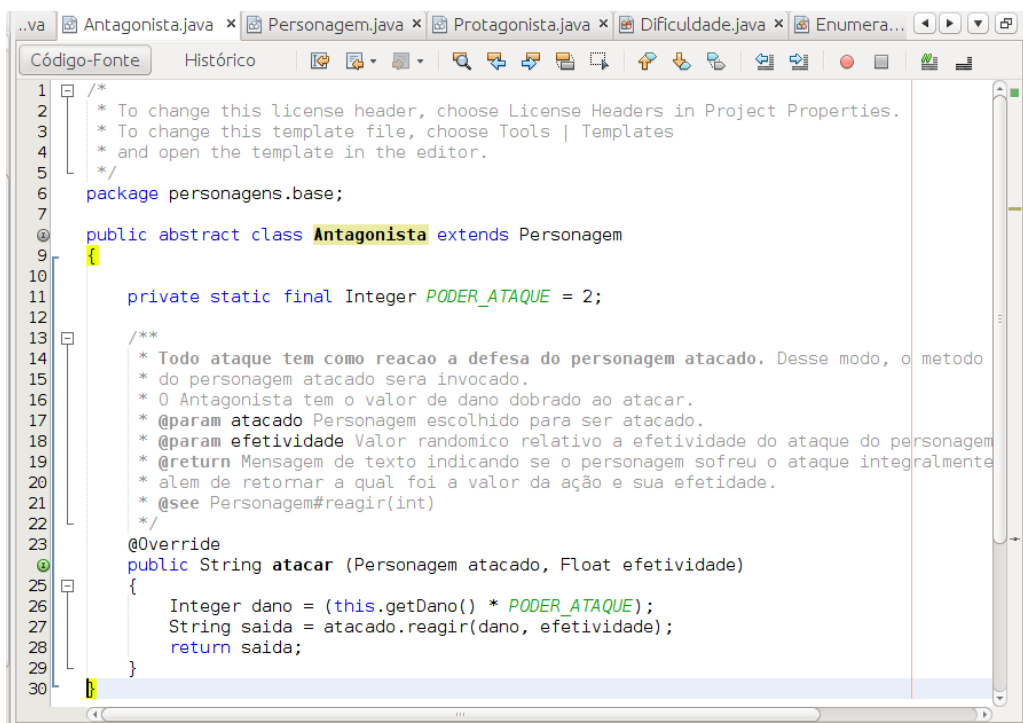
Os atributos não são diretamente acessíveis, mas possuem métodos getters e setters com modificador public dentro da classe Personagem.



```
90         return saida;
91     }
92 }
93
94 public String getNome()
95 {
96     return nome;
97 }
98
99 public void setNome(String nome)
100 {
101     this.nome = nome;
102 }
103
104 public Float getQuantidadeVida()
105 {
106     return quantidadeVida;
107 }
108
109 public void setQuantidadeVida(Float quantidadeVida)
110 {
111     this.quantidadeVida = quantidadeVida;
112 }
113
114 public String getImagem() {
115     return imagem;
116 }
117
118 public void setImagem(String imagem) {
119     this.imagem = imagem;
120 }
```

2.2) Herança

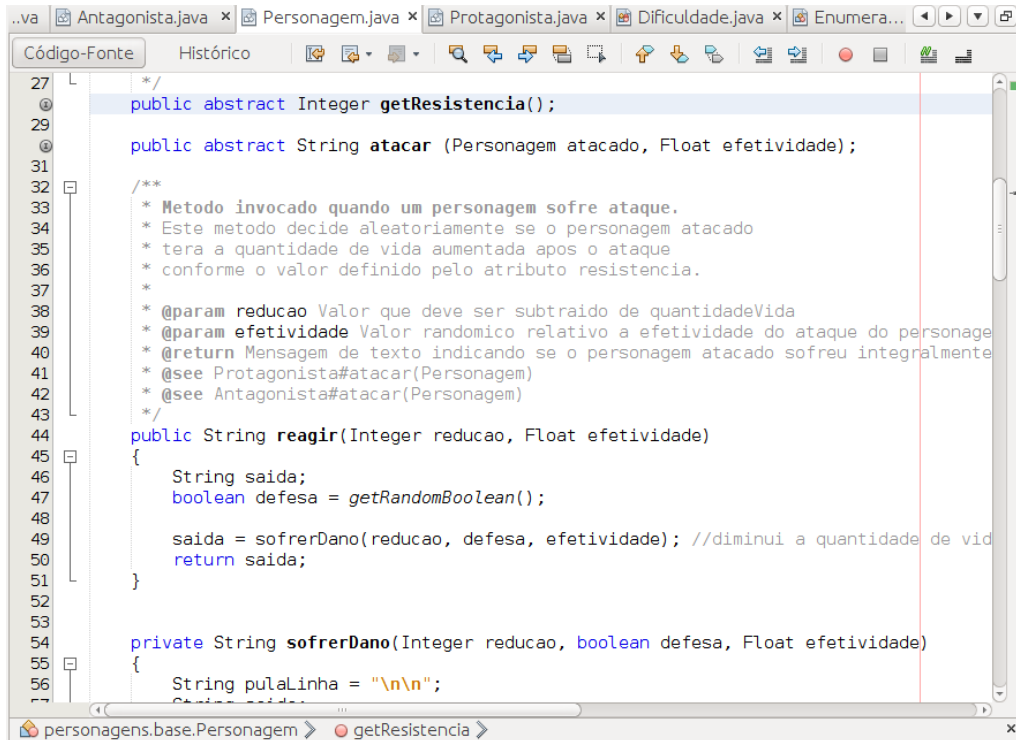
Herança é um princípio da orientação a objetos onde novas classes (chamadas de subclasses ou ainda classes filhas) herdem tanto atributos quanto métodos de outra classe (chamada de super classe ou então classe pai). Exemplo Antagonista é um Personagem, herdando todos os seus atributos e métodos.



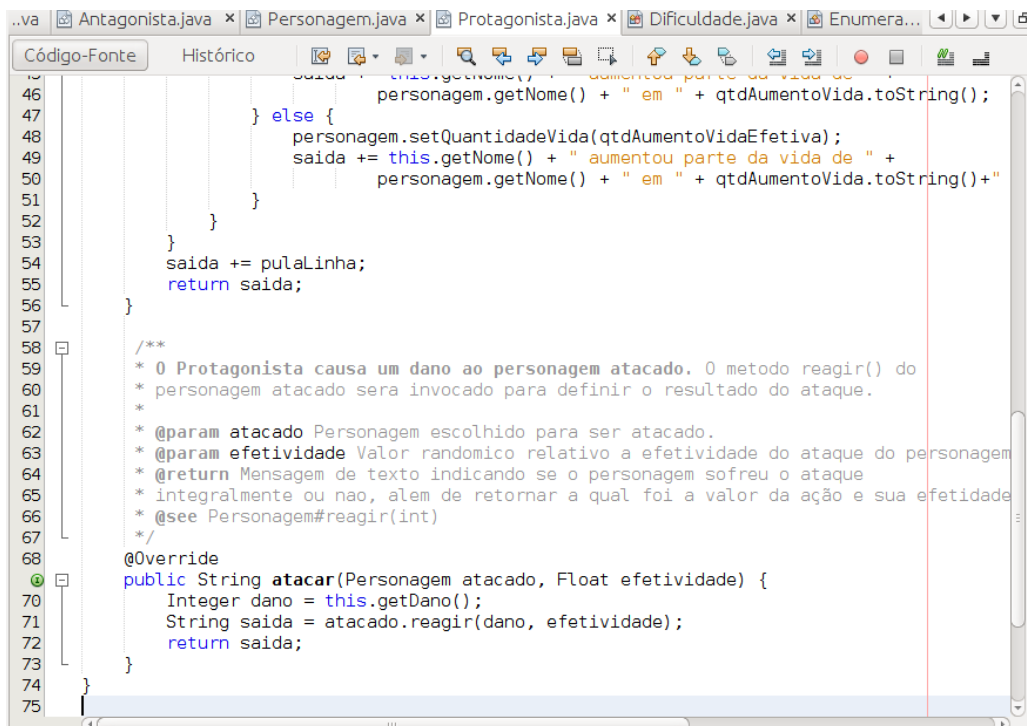
```
1  /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package personagens.base;
7
8  public abstract class Antagonista extends Personagem
9  {
10
11     private static final Integer PODER_ATAQUE = 2;
12
13     /**
14      * Todo ataque tem como reacao a defesa do personagem atacado. Desse modo, o metodo
15      * do personagem atacado sera invocado.
16      * O Antagonista tem o valor de dano dobrado ao atacar.
17      * @param atacado Personagem escolhido para ser atacado.
18      * @param efetividade Valor randomico relativo a efetividade do ataque do personagem
19      * @return Mensagem de texto indicando se o personagem sofreu o ataque integralmente
20      * alem de retornar a qual foi a valor da ação e sua efetividade.
21      * @see Personagem#reagir(int)
22      */
23     @Override
24     public String atacar (Personagem atacado, Float efetividade)
25     {
26         Integer dano = (this.getDano() * PODER_ATAQUE);
27         String saida = atacado.reagir(dano, efetividade);
28         return saida;
29     }
30 }
```

2.3) Polimorfismo

Polimorfismo é diretamente relacionado a herança. Uma subclasse contém 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.



```
27  /*
28  29  public abstract Integer getResistencia();
30  31
32  33  public abstract String atacar (Personagem atacado, Float efetividade);
34  35
36  37  /**
38  39  * Metodo invocado quando um personagem sofre ataque.
40  41  * Este metodo decide aleatoriamente se o personagem atacado
42  43  * tera a quantidade de vida aumentada apos o ataque
44  45  * conforme o valor definido pelo atributo resistencia.
46  47  *
48  49  * @param reducao Valor que deve ser subtraido de quantidadeVida
49  50  * @param efetividade Valor randomico relativo a efetividade do ataque do personagem
50  51  * @return Mensagem de texto indicando se o personagem atacado sofreu integralmente
51  52  * @see Protagonista#atacar(Personagem)
52  53  * @see Antagonista#atacar(Personagem)
53  54  */
54  55  public String reagir(Integer reducao, Float efetividade)
55  56  {
56  57  String saida;
57  58  boolean defesa = getRandomBoolean();
58  59
59  60  saida = sofrerDano(reducao, defesa, efetividade); //diminui a quantidade de vida
60  61  return saida;
61  62  }
62  63
63  64  private String sofrerDano(Integer reducao, boolean defesa, Float efetividade)
64  65  {
65  66  String pulaLinha = "\n\n";
66  67  String saida;
```



```
46  saida += this.getNome() + " aumentou parte da vida de " +
47  personagem.getNome() + " em " + qtdAumentoVida.toString();
48  } else {
49  personagem.setQuantidadeVida(qtdAumentoVidaEfetiva);
50  saida += this.getNome() + " aumentou parte da vida de " +
51  personagem.getNome() + " em " + qtdAumentoVida.toString();
52  }
53  }
54  saida += pulaLinha;
55  return saida;
56  }
57
58  /**
59  * O Protagonista causa um dano ao personagem atacado. O metodo reagir() do
60  * personagem atacado sera invocado para definir o resultado do ataque.
61  *
62  * @param atacado Personagem escolhido para ser atacado.
63  * @param efetividade Valor randomico relativo a efetividade do ataque do personagem
64  * @return Mensagem de texto indicando se o personagem sofreu o ataque
65  * integralmente ou nao, alem de retornar a qual foi a valor da ação e sua efetividade
66  * @see Personagem#reagir(int)
67  */
68  @Override
69  public String atacar(Personagem atacado, Float efetividade) {
70  Integer dano = this.getDano();
71  String saida = atacado.reagir(dano, efetividade);
72  return saida;
73  }
74  }
75  }
```

```

1  /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package personagens.base;
7
8  public abstract class Antagonista extends Personagem
9  {
10
11     private static final Integer PODER_ATAQUE = 2;
12
13     /**
14      * Todo ataque tem como reacao a defesa do personagem atacado. Desse modo, o metodo
15      * do personagem atacado sera invocado.
16      * 0 Antagonista tem o valor de dano dobrado ao atacar.
17      * @param atacado Personagem escolhido para ser atacado.
18      * @param efetividade Valor randomico relativo a efetividade do ataque do personagem
19      * @return Mensagem de texto indicando se o personagem sofreu o ataque integralmente
20      * alem de retornar a qual foi a valor da açao e sua efetidade.
21      * @see Personagem#reagir(int)
22      */
23     @Override
24     public String atacar (Personagem atacado, Float efetividade)
25     {
26         Integer dano = (this.getDano() * PODER_ATAQUE);
27         String saida = atacado.reagir(dano, efetividade);
28         return saida;
29     }
30 }

```

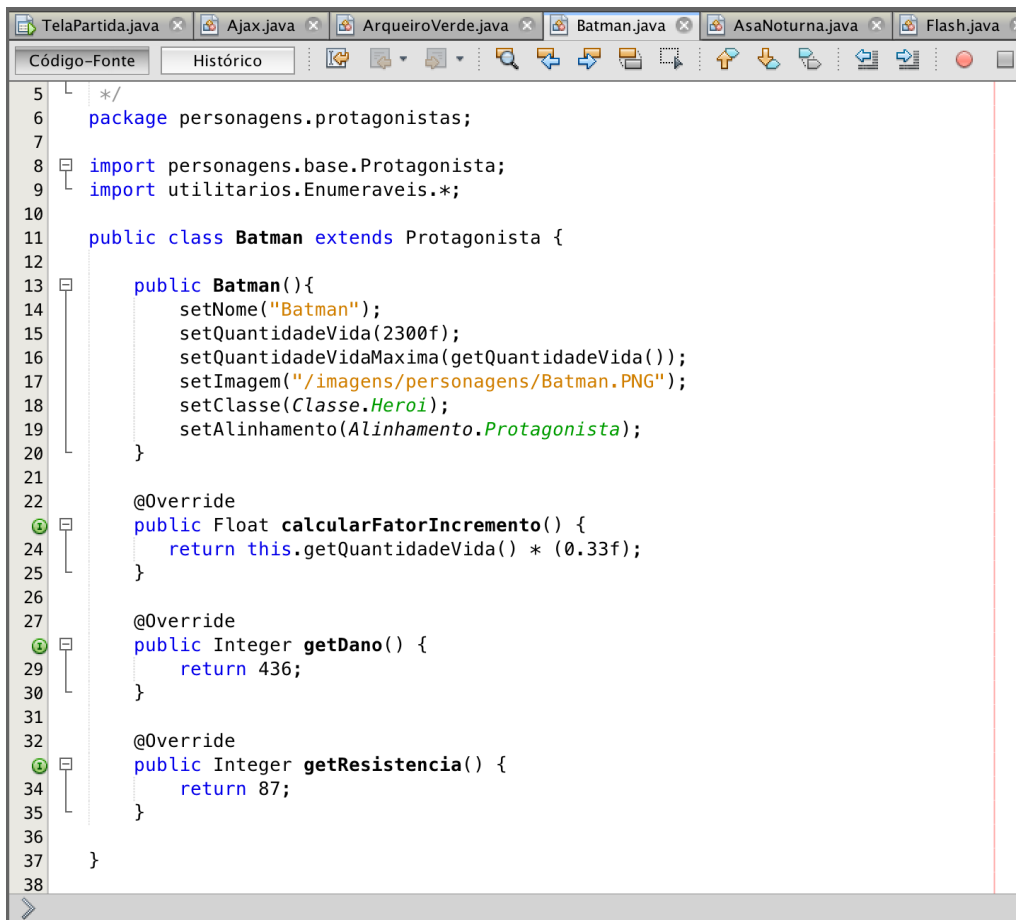
Exemplo de uma classe concreta de Antagonista.

```

1  /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package personagens.antagonistas;
7
8  import personagens.base.Antagonista;
9  import utilitarios.Enumeraveis.*;
10
11 public class AdaoNegro extends Antagonista{
12
13     public AdaoNegro(){
14         setNome("Adão Negro");
15         setQuantidadeVida(3850f);
16         setQuantidadeVidaMaxima(getQuantidadeVida());
17         setImagem("/imagens/personagens/AdaoNegro.png");
18         setClasse(Classe.SuperVilao);
19         setAlinhamento(Alinhamento.Antagonista);
20     }
21
22     @Override
23     public Integer getDano() {
24         return 360;
25     }
26
27     @Override
28     public Integer getResistencia() {
29         return 79;
30     }
31
32 }
33

```

Exemplo de uma classe concreta de Protagonista:



```
5  */
6  package personagens.protagonistas;
7
8  import personagens.base.Protagonista;
9  import utilitarios.Enumeraveis.*;
10
11 public class Batman extends Protagonista {
12
13     public Batman(){
14         setNome("Batman");
15         setQuantidadeVida(2300f);
16         setQuantidadeVidaMaxima(getQuantidadeVida());
17         setImagem("/imagens/personagens/Batman.PNG");
18         setClasse(Classe.Heroi);
19         setAlinhamento(Alinhamento.Protagonista);
20     }
21
22     @Override
23     public Float calcularFatorIncremento() {
24         return this.getQuantidadeVida() * (0.33f);
25     }
26
27     @Override
28     public Integer getDano() {
29         return 436;
30     }
31
32     @Override
33     public Integer getResistencia() {
34         return 87;
35     }
36
37 }
38
```

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 histórias 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.

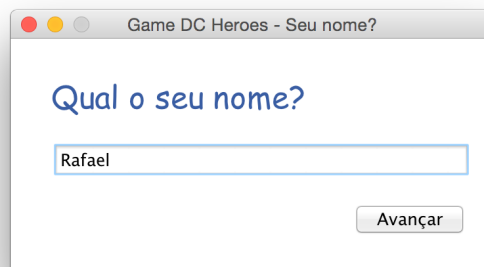
3.2) Classes controladoras implementadas

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 equipe 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.



Game DC Heroes - Seleciona Equipe

Rafael,
Selecione sua equipe:

Nome	Dano	Resistência	Qtde. Vida	Alinhamento	Classe
------	------	-------------	------------	-------------	--------

Quantidade de Personagens Selecionados: 0



Adão Negro

Alinhamento: Antagonista
Classe: SuperVilao
Dano: 360
Resistência: 79
Qtde. Vida: 3850.0

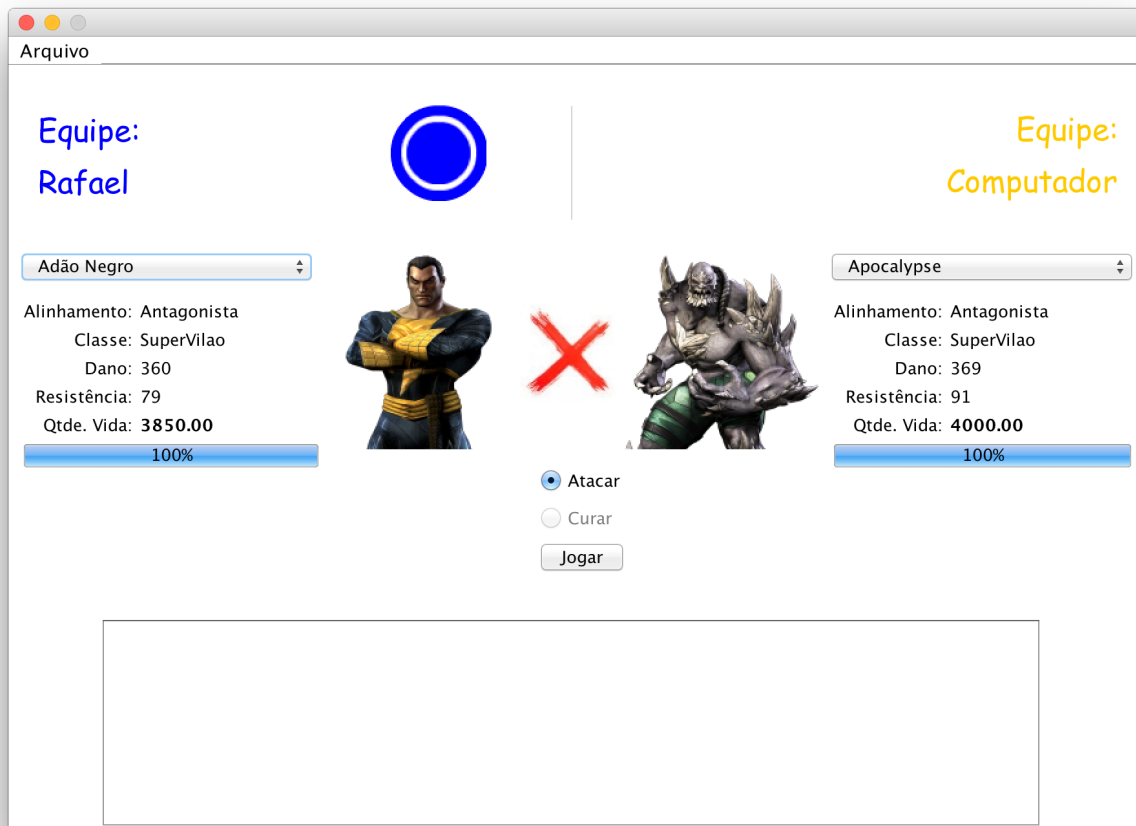
Qual a dificuldade da Partida?

☐ Fácil

☒ Normal

☐ Difícil

☐ Insano



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, Apocalypse, 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: *Herói, Mago, SuperHerói, Vilão, Bruxo, SuperVilão*

TipoJogador: *Humano, Computador*

Dificuldade: *FÁCIL, NORMAL, DIFÍCIL, 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 contém 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.

