Relatório de projeto POO

Equipe: Arthur David, Emerson Lucena, Gustavo Cesar e Yaslim Soares.

Descrição do jogo e informações gerais

O game LoLo, com características de “Jogo de Interpretação de Personagens” (RPG) e “Battle Royale”, onde resta apenas um vivo em confrontos, foi produzido e idealizado por Arthur David (1538730), Emerson Lucena (1538695), Gustavo Cesar (1538617) e Yaslim Soares (1538806).

O “game” possui um menu inicial, dividido em: Batalhar, em que são selecionados dois ou mais personagens **carregados** à batalha; Personagens, que quando selecionada direciona o usuário para um outro menu, contendo todas as funções de gerenciamento de personagens, permitindo criar um novo personagem, salvar todos criados, carregar personagem (é necessário ter personagens carregados para iniciar uma batalha), exibir o status dos guerreiros carregados, listar os personagens salvos, editar e deletar um personagem; e, por fim, a opção de encerrar o programa.

O jogo é dividido em rounds dinâmicos, ou seja, não tem limite, pois uma batalha só chega ao fim quando apenas um guerreiro permanece vivo. Cada classe de guerreiro possui pontos em comum, são eles: pontos de vida, poder de habilidade, poder de ataque físico, ataque especial e básico, armadura, resistência mágica e os pontos de habilidade que são incrementados para realizar o ataque especial.

Funcionalidades Básicas

Para estas funcionalidades fundamentais, foi criado um pacote chamado rpg.utility, no qual foi adicionado as classes que fariam suas respectivas funções (Salvar, Carregar, Apagar, Editar e UX). Abaixo segue, separados por tópicos, o detalhamento das funcionalidades básicas do jogo.

Salvar Personagem:

Na classe Salvar, do pacote rpg.utility, temos o método “save()” que recebe como parâmetro um objeto do tipo Personagem, e a partir disso, são criados dois arquivos para salvar este personagem, é criado um arquivo individual, com o nome do personagem e gravado em seu conteúdo todas as informações que ele possui, como o seu nome, o seu tipo (Mago, Tanque, etc), pontos de vida, e todos os outros atributos que os personagens possuem. E além de criar o arquivo individual, também é criado um arquivo chamado Lista de Personagens no qual é armazenado os nomes dos personagens que foram salvos. Esse arquivo será muito utilizado quando utilizarmos as funções de listar todos os personagens salvos, e nas funções

de apagar e editar. Também é criado um método chamado “saveEspecial()” que após ser passado um objeto do tipo Personagem, ele irá criar apenas o arquivo individual do personagem, método esse que irá ser útil quando quisermos editar um personagem.

Carregar Personagem:

Na classe Carregar, do pacote rpg.utility, temos o método “load()” que ao ser passado uma String como parâmetro com o nome do personagem em questão, iremos acessar o seu arquivo individual e capturar o seu nome, e o seu tipo, e após isso, o retorno do método é uma instância do personagem que foi consultado, com o nome que foi capturado, seja ele um Mago, Tanque ou qualquer outro tipo. Passando adiante, temos o método “listarPersonagens()” que acessa o arquivo “Lista de Personagens.txt” e exibe para o usuário todos os personagens que foram salvos. Além disso, também temos o método “selecaoPersonagem()”, que basicamente é um menu interativo para que o usuário possa escolher qual personagem ele deseja carregar, baseado no arquivo “Lista de Personagens.txt”, após ele escolher o personagem, é chamado a função “load”, para carregá-lo em uma variável auxiliar e assim para esta ser utilizada como retorno para o programa. Também temos métodos secundários, que checam a existência de um personagem com o mesmo nome em uma lista, ou no arquivo em que contém os personagens salvos, métodos esses que são utilizados na classe UX. Por fim, temos um método que acessa o arquivo com os personagens salvos, e armazena os seus nomes em uma lista de String e esta é usada como retorno do método. Utilizaremos esse método nas classes Editar e Apagar.

Apagar Personagem:

Na classe Apagar, temos dois métodos, um para selecionar o personagem que irá ser apagado, e o outro que irá apagar o seu arquivo individual e o seu nome na lista de personagens salvos. O método para selecionar o personagem, “pegarNome()”, é mostrado para o usuário todos os personagens que foram salvos, e após isso, é solicitado ao usuário que ele informe o personagem que será apagado, após isso, o nome desse personagem é armazenado em uma variável auxiliar, e esta configura o retorno da função. E por último, o método “apagarArquivo()”, que apaga o arquivo individual do personagem e o seu nome do arquivo “Lista de Personagens”. Essa função recebe, como parâmetro, o nome do personagem que será apagado. Primeiramente, é acessado o seu arquivo individual e o arquivo que contém todos os personagens salvos. Após isso, é criado uma lista de String com todos os personagens salvos e iremos retirar o personagem que será apagado dessa lista. Depois, o arquivo “Lista de Personagens.txt” é limpo, deixando o arquivo vazio. Após isso, ele é reescrito de acordo com a lista criada dentro do método, que não terá o personagem que foi apagado. E por fim é deletado o seu arquivo individual, com a função “.delete()”.

Editar Personagem:

Na classe Editar, temos três métodos, que se assemelham aos métodos da classe Apagar, dentre eles, o método de “pegarNome()”, que é implementado um menu para o usuário escolher um personagem entre os já salvos para editá-lo e, após isso, é utilizado o método “.load()” para buscar o seu arquivo individual e colocá-lo dentro de um objeto auxiliar do tipo Personagem, que é utilizado como retorno para esta função. O método "editarPersonagem()" é o responsável por manipular tanto o arquivo individual, quanto o arquivo que contém todos os personagens salvos. Esse método recebe como parâmetro um objeto do tipo Personagem. Semelhante ao método “apagarArquivo()”, é acessado o arquivo referente aos personagens salvos e o arquivo individual do personagem passado como parâmetro. Após isso é criado uma lista com todos os personagens salvos que será utilizada em outras operações na função. Em seguida, é solicitado ao usuário um novo nome ao personagem que será editado, após a interação do usuário, é capturado a posição que o nome antigo do personagem está na lista, após isso, ele é retirado da lista e é adicionado outro em sua posição. Com essa lista em mãos, limpamos o arquivo que contém os personagens salvos e reescrevemos ele de acordo com a lista que criamos, já com o personagem com seu novo nome. Após essa operação, iremos editar o seu arquivo individual, para isso, iremos excluir o arquivo que contém o personagem ainda não editado para criar outro de acordo com as especificações do usuário. Essa operação é feita através da interação do usuário, pedindo para que ele informe o novo tipo do personagem, e após isso é instanciado em um objeto do tipo Personagem, o personagem resultante de acordo com o tipo escolhido, e por último é chamado o método da classe Salvar, “saveEspecial()”, que cria o arquivo para esse nosso personagem.

Criar Personagem:

Para criar um personagem, temos o método “criarPersonagem()” na classe UX, nesse método é integrado um menu interativo para que o usuário possa informar as especificações do seu personagem, o nome e o seu tipo, após isso é criado um personagem de acordo com o seu tipo e adicionado a uma lista do tipo Personagem, onde estão armazenados todos os personagens que estão carregados no sistema.

Interação entre as classes com uma principal UX:

Para separarmos bem o que cada classe irá fazer, criamos a classe UX, que será a responsável por realizar a integração das demais classes de utilidade, nela foram passadas como atributos objetos do tipo Salvar, Carregar, Apagar, Editar e Batalha, que serão utilizados nos métodos dessa classe, tais como “editarPersonagem()”, “apagarPersonagem()”, etc, além disso temos uma lista do

tipo Personagem que será a responsável por armazenar todos os personagens que foram carregados no sistema. Resumindo, essa classe é a auxiliar responsável pelos métodos que serão chamadas na classe principal “Main”, nela todas as funcionalidades são realizadas, desde criar personagem até apagá-lo. No método “salvarPersonagens()”, é percorrido toda a lista de personagens e é salvo todos os personagens que estão nessa lista. Já para o método “carregarPersonagem()”, é chamado os métodos da classe Carregar e após a seleção do personagem, este é adicionado na lista de personagens. O mesmo acontece para os demais métodos.

Menu de atividades:

O menu interativo foi dividido em duas partes, o menu para a batalha, e o menu de personagens. O menu para a batalha consiste na seleção de personagens já carregados, para a realização da batalha, tendo suas exceções, tais como, a impossibilidade de realização de uma batalha, caso não tenha personagens suficientes carregados. Já para o menu de personagens, tem-se diversas funcionalidades, sendo as principais, a função de criar, editar, excluir e listar os personagens.

Funcionalidades personagens:

O projeto foi implementado seguindo a seguinte ideia: os campos de batalhas são voltados para guerreiros subdivididos em tipos de lutadores, são eles: Assassino, Lutador, Mago e Tanque, em que cada um tem especificações e habilidades próprias. As classes de cada guerreiro estão localizadas na pasta *“X://rpg.poo/src/rpg/personagens”.*

Assassino:

Especialista em ataque físico. Guerreiro com mais dano de ataque físico.

pontos De Vida = 160;
poder Ataque Físico = 45;
armadura = 30;
resistência Mágica = 30;
ataque Especial = necessário dois pontos de habilidade!

Lutador:

Especialista em aguentar danos, com armadura elevada, e alto ataque físico.

pontos De Vida = 250;
poder Ataque Físico = 30;
armadura = 45;
resistência Mágica = 45;
ataque Especial = necessário dois pontos de habilidade!

Mago:

Especialista em ataques mágicos. O único guerreiro com ataque mágico.

pontos De Vida = 175;
poder De Habilidade = 40;
poder Ataque Físico = 0;

armadura = 35;
resistência Mágica = 35;
ataque Especial = necessário um ponto de habilidade!

Tanque:

Especialista em resistência. Guerreiro mais duro de matar, porém, com baixo dano.

pontos De Vida = 325;
poder Ataque Físico = 25;
armadura = 60;
resistência Mágica = 60;
ataque Especial = necessário um ponto de habilidade!

Funcionalidades Batalha

A classe “Batalha”, localizada *“X://rpg.poo/src/rpg/batalha/Batalha.java”* , implementa a função de batalha e possui restrição de execução para, no mínimo, dois guerreiros carregados para o programa.

Além disso, a classe Batalha possui um método chamado “batalhar()”, que tem como parâmetro uma lista de objetos com uma réplica dos guerreiros, instanciados na classe *“X://rpg.poo/src/rpg/utility/UX”* nas **linhas 345-359**. O método “batalhar()” é chamado na **linha 357** deste mesmo diretório. Após o método ser chamado, é instanciado uma lista de objetos chamada lutadores, com todos os guerreiros passados como parâmetro. Antes de iniciar a batalha, no diretório acima, é listado todos os guerreiros disponíveis para serem selecionados para a batalha. O usuário tem a opção de selecionar quais personagens irão lutar. Após isso, é pedido a permissão do usuário para iniciar a batalha. Após ser aceito, é verificado a quantidade de guerreiros mais uma vez e os dados dos personagens são listados com a função mostrarDados(), presente na classe Personagem, e sobrescrita, utilizando polimorfismo, em cada classe específica dos guerreiros. A cada round a ordem de jogar dos guerreiros é sorteada, manipulando as posições da lista de lutadores com a função "Collections.shuffle(lutadores)", **na linha 64 da classe Batalha.** A partir de então, após o sorteio de ordem de jogada, o guerreiro a jogar é removido do vetor para evitar que ataque a si, e o restante dos guerreiros é listado como opção de ataque. Mas além disso, o usuário tem a opção de aprimorar defesa, armadura ou resistência mágica, ou realizar “PowerUp”, que incrementa mais um ponto no atributo pontos de habilidade. Ao realizar o ataque especial, é verificado a pontuação desse atributo para liberar o ataque, caso não, é ofertada a opção de ataque padrão novamente. Ao selecionar qualquer tipo de ataque, especial ou padrão, é sorteada uma intensidade de ataque que é exposta ao usuário, variando de básico, moderado, intenso e crítico, com o incremento no dano de 0%, 0.1%, 0.2% e 0.4%, respectivamente. Observação: o ataque padrão verifica qual o atributo de maior dano (ataque físico ou ataque mágico) e o faz. Importante ressaltar que, apesar de o método “ataqueEspecial()” receber como parâmetro o guerreiro sobataque, este não foi utilizado, pois

A opção de aprimorar defesa fornece a opção de melhorar a armadura ou a resistência mágica. Depois de todos jogarem, é verificado o tamanho da lista de lutadores, caso restem mais que um, um novo round é iniciado. Quando resta apenas um guerreiro vivo, o método “batalhar()” é finalizado e o programa retorna o vencedor da partida. Em seguida os dados alterados de cada personagem são perdidos e prevalecem os originais e o usuário é redirecionado para o menu principal.

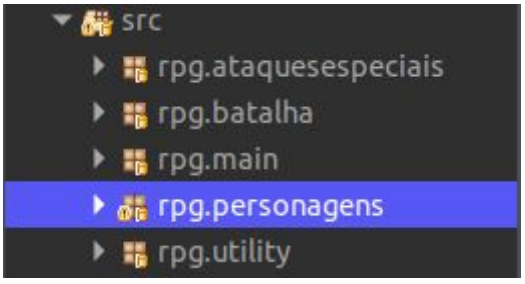
Critérios de avaliação:

Abaixo segue, separado por tópicos, o detalhamento de onde e como foram utilizadas as principais funcionalidades da Programação Orientada a Objetos.

Onde e como foi usado a herança?

A herança acontece quando uma classe(subclasse) herda atributos e métodos de outra classe(superclasse), podendo ter seus próprios atributos e métodos.

Em nosso projeto é possível localizar a implementação desse conceito em *“X://rpg.poo/src/rpg/personagens/”*.



Localização em uma IDE

Podemos encontrar a superclasse no arquivo *“Pesonagem.java”* no qual define atributos que serão usadas nas seguintes subclasses: *“Assassino.java”*, *“Lutador.java”*, *“Mago.java”*, *“Tanque.java”*.

Cada subclasse possui suas próprias funcionalidades que as diferenciam das demais.

Onde e como foi usado o encapsulamento?

A ideia de encapsulamento é de controlar o acesso aos dados, para que não fiquem suscetíveis à ações indevidas.

Essa ideia foi colocada em prática em nosso projeto e é possível encontrá-la em diversos locais na aplicação. Lista de locais onde foi usado encapsulamento(considerando que quando usamos ***public,*** o conceito de encapsulamento não acontece):

- *"X://rpg.po/src/rpg/ataquesespeciais/Ataques.java"*
- *“X://rpg.poo/src/rpg/batalha/Batalha.java”*

- *“X://rpg.poo/src/rpg/personagens/…*
  .../Personagem.java”
  .../Assassino.java”
  .../Lutador.java”
  .../Mago,java”
  .../Tanque.java”
- *“X://rpg.poo/src/rpg/utility/…*
  .../Apagar.java”
  .../Editar.java”
  .../UX.java”

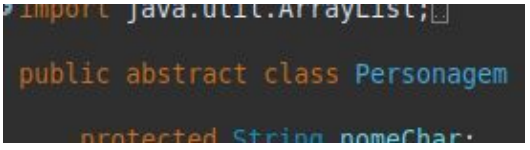
Utilização do encapsulamento

Assim como no conceito, utilizamos ***private*** para deixar com que certos dados fossem possível ser alterados diretamente apenas pela própria classe, e utilizamos ***protected*** em outros dados para possibilitar alteração direta somente na própria classe e nas classes herdeiras a ela.

Onde e como foi usada a classe abstrata?

Segundo o conceito, uma classe abstrata é uma superclasse genérica e não pode ser instanciada, servindo de modelo para as outras classes que são subclasses mais específicas.

Encontramos a classe abstrata em prática no seguinte local do projeto: *“X://rpg.poo/src/rpg/personagens/Personagens.java”*.



Utilização da classe abstrata

Com o uso da classe abstrata, não é possível criar um objeto nessa classe, qualquer objeto criado será especificamente de uma subclasse concreta. Temos a lista de subclasses concretas pertencentes à classe **Personagem**:

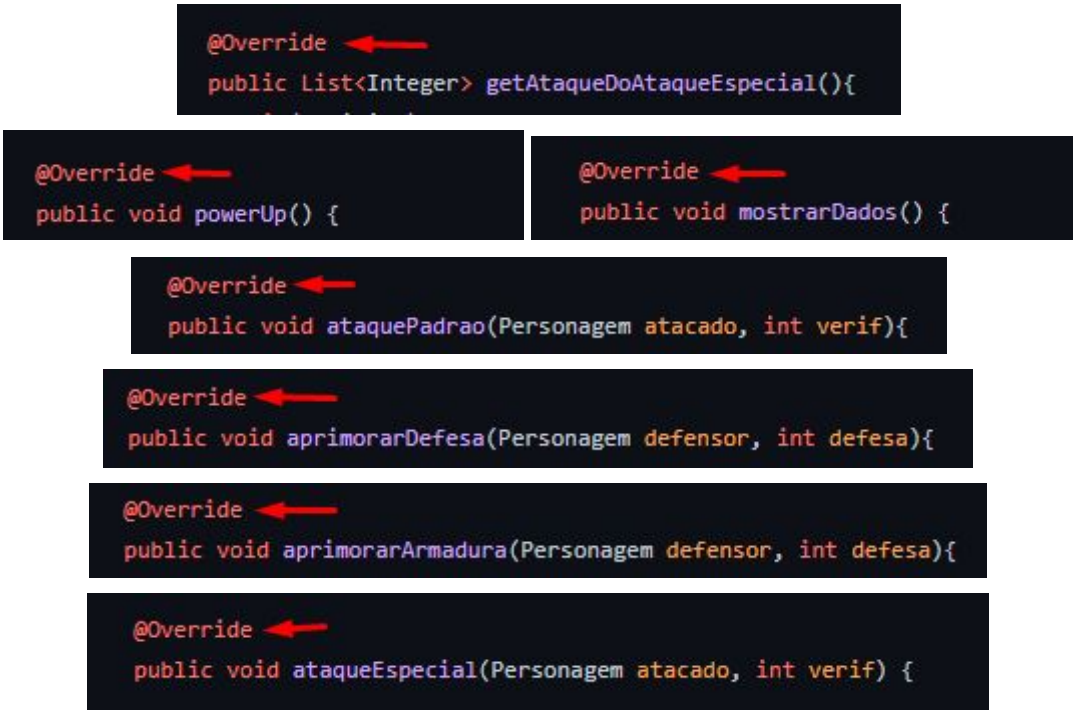
- “*X://rpg.poo/src/rpg/personagens/Assassino.java”*

- “*X://rpg.poo/src/rpg/personagens/Lutador.java”*
- *“X://rpg.poo/src/rpg/personagens/Mago.java”*
- *“X://rpg.poo/src/rpg/personagens/Tanque.java”*

Onde e como foi usado o polimorfismo?

O principal conceito de polimorfismo e o que foi utilizado, na elaboração deste projeto, é o dinamismo, por parte do programa, contido em sua execução. Percebe-se isso na sobrescrita(overriding) de métodos de uma classe progenitora, ou superclasse, para suas filhas, em que ambas têm o mesmo método mas com implementações diferentes.

O conceito de polimorfismo pode ser encontrado, principalmente, nas classes filhas da classe “Personagem”, que representam os tipos de lutadores. Pode-se localizar a aplicação do conceito em *“X://rpg.poo/src/rpg/personagens”,* nas classes: “Assassino.java”, “Lutador.java”, “Mago.java” e “Tanque.java”, nos métodos “ataqueEspecial”, “getAtaqueEspecial”, “powerUp”, “ataquePadrao”, “aprimorarDefesa” e “aprimorarArmadura”. Esses métodos estão contidos na Interface e na classe abstrata “Personagem.java”, que herda seus métodos para as classes filhas, no caso, as classes de guerreiros, que aplicam implementações diferentes.



Trechos da aplicação do Polimorfismo (Override).

Onde e como foi usado as interfaces?

Quando uma classe implementa uma interface, se compromete a fornecer o comportamento publicado por esta interface, ou seja, essa classe obrigatoriamente utilizará os métodos que foram declarados na interface implementada.

Utilizamos o conceito de interface em nosso projeto e podemos localizá-la em *"X://rpg.poo/src/rpg/personagens/Interface.java"*, onde podemos ver vários métodos que terão que ser implementados pelas subclasses concretas à superclasse abstrata **Personagem** que implementa essa interface(detalhes em "Onde e como foi usada a classe abstrata?").

```
public interface Interface {
    public TiposDePersonagem getTipo();
    public void setTipo(TiposDePersonagem tipo);
    public int getPontosDeVida();
    public int getPoderDeHabilidade();
    public int getPoderAtaqueFisico();
    public String getNomeChar();
    public void setNomeChar(String nomeChar);
    public Ataques getAtaqueEspecial();
    public Ataques getAtaqueBasico();
    public int getArmadura();
    public int getResistenciaMagica();
    public void mostrarDados();
    public void ataqueEspecial(Personagem atacado, int verif);
```

Trecho da interface utilizada em nosso projeto

```
public abstract class Personagem implements Interface{

    protected String nomeChar;
    protected int pontosDeVida;
```

Trecho classe que implementa a interface

Onde e como foi realizado o tratamento de exceções?

Antes de explicarmos o que é tratamento de exceções, vamos explicar o que são exceções.

Uma exceção é algo problemático que ocorre durante a execução, ou compilação, de um software. Sendo assim, o tratamento de exceções é o processo de resolver exceções, por meio de funções apropriadas, para que o programa continue ou termine normalmente.

Usamos isso em nosso projeto nos seguintes casos:

1. *"X://rpg.poo/src/rpg/batalha/Batalha.java":* Utilizamos o tratamento com diferentes finalidades. São elas: processar qualquer erro de execução na batalha; percorrer vetores de forma dinâmica, utilizados para facilitar o processo de apenas um guerreiro vivo; listar vetores de forma segura; e a mais importante, acontece **na linha 253,** em que o ataque especial só acontece com a execução do método "ataqueEspecial()" **na linha 249,** pois caso este método não seja executado corretamente, graças a exceção presente neste método dos pontos de habilidade, o vetor retornado da função é vazio, logo, apresentará erro caso o programa acesse suas posições vazias. No catch desse mesmo try, é executado outras linhas de código para a batalha prosseguir normalmente.

2. *“X://rpg.poo/src/rpg/utility/…*

2.1 .../Apagar.java”: Utilizamos o tratamento com diferentes casos, como, erro ao deletar o personagem e erro ao carregar o personagem.

2.2 .../Carregar.java”: Utilizamos o tratamento com diferentes casos, como, erro ao carregar personagem e erro ao ler a lista de personagem.

2.3 .../Editar.java”: Utilizamos o tratamento com diferentes casos, como, erro ao editar personagem, caractere inválido e erro inesperado.

2.4 .../Salvar.java”: Utilizamos o tratamento com diferentes casos, como, erro ao salvar personagem.

2.5 .../UX.java”: Utilizamos o tratamento com diferentes casos, como, erro ao criar personagem, erro inesperado e erro ao iniciar a batalha.

```
try{
    // iremos capturar o seu nome e o seu tipo para podermos coloca-lo no programa em execucao
    File arq = new File(arquivo);
    Scanner s = new Scanner(arq);
    String tipo = s.nextLine();
    String nome = s.nextLine();

    if(TiposDePersonagem.valueOf(tipo).equals(TiposDePersonagem.MAGO)) {···
    } else if(TiposDePersonagem.valueOf(tipo).equals(TiposDePersonagem.ASSASSINO)) {···
    } else if(TiposDePersonagem.valueOf(tipo).equals(TiposDePersonagem.LUTADOR)) {···
    } else if(TiposDePersonagem.valueOf(tipo).equals(TiposDePersonagem.TANQUE)) {···
    } else {···

}
catch(Exception e){
    System.out.println("Erro ao carregar personagem!\n");
    System.out.println("Pressione ENTER para continuar...");
    new java.util.Scanner(System.in).nextLine();
}
```

Exemplo de utilização do Tratamento de exceções