Projeto de Programação Orientada a Objetos

Carlos Estellita Neto, Crislane Maria Da Silva Costa, Douglas Araújo da Silva.

Bacharelado em Ciência da Computação - Universidade Estadual do Ceará (UECE) – Fortaleza – CE – Brasil

Resumo:

Neste presente trabalho, implementamos um jogo de turno estilo Battle Royale que visa a utilização de técnicas de Programação Estruturada e Orientada a Objetos.

Para o desenvolvimento deste relatório, a equipe decidiu explicar a execução de OO mais detalhadamente na seção "4".

1. Funcionalidades Básicas:

1.1. Criar, Deletar, Editar, Mostras as Informações de um personagem:

Em Batalha.java, tem um Método criarPersonagem(), onde o jogador deve escolher seu Nome, Tribo e Classe.

Na Batalha.java, existe um Método excluirPersonagem(), além de mostrar todos nomes de personagens existentes, ele também recebe a resposta do usuário para excluir um arquivo.

Na Classe Batalha.java, há o Método editarPersonagem(), este mostra todos os nomes de personagens existentes e recebe a resposta do usuário para poder editar nome, tribo e classe.

No início da partida, na Batalha.java, é ativado o Método mostrarPersonagens(), cuja finalidade é imprimir na tela os jogadores na partida.

1.2. Salvar e Carregar os personagens criados:

Após criar personagem, é ativado o Método salvarPersonagem(). Apesar de estar na Classe Batalha.java, ele é implementado na Classe ManipuladoArquivos.java.

Na Batalha.java, existe um Método carregarPersonagem(), este pega um personagem de um arquivo e põe dentro de um Vetor que armazena os jogadores na partida.

1.3.Criar um menu inicial para realizar as atividades:

O menu inicial na Classe Main.java, é composto por apenas duas opções:

"1" para jogar, "2" para sair. Em seguida, os usuários devem escolher um número inteiro "x" de jogadores para poderem criar, carregar, editar e deletar personagens no menu secundário. Utilizamos loopings, condicionais e tratamento de erro.

2. Funcionalidades de Batalha:

2.1. Batalha feita em rodadas, onde cada personagem realiza uma única ação e a ordem das ações é sorteada:

No Método "luta()", da Batalha.java, fizemos um vetor "vetorAleatorio" que recebe o retorno do Método "rodadaAleatoria" na Classe Sistema.java.

2.2. Ações: Atacar, Defender, Power-Up:

Na Classe Batalha.java, fizemos o Método "atacar()", que se refere ao ataque básico de cada classe; nele, há alguns incrementos (como o Power-Up e demais que serão explicados mais tarde). Outrossim, também na Classe Batalha.java, criamos o Método "defesa()", que ao ser ativado, retorna o dano recebido menos o nível de defesa.

2.3 Intensidade das ações: define-se um valor base e acrescenta-se uma fator de incremento aleatório dentro de um intervalo pré-definido.

No Método atacar(), da Batalha.java, há uma aplicação de Ataque Crítico. Ao atacar, existe a probabilidade de 40% de um jogador causar +5 de dano.

3. Funcionalidades dos Personagens:

3.1. Devem existir pelo menos três tipos:

No Personagem.java, existe um Método Construtor que inicializa o atributo "tribo", que pode ser: Fogo, Água e Planta. Ademais, também se inicializa o atributo "classe", que funciona como "subclasse" da tribo, que pode ser: Guerreiro, Arqueiro e Mago.

3.2. A equipe define as vantagens e desvantagens de cada tipo em relação aos demais.

Os elementos possuem vantagens e desvantagens entre si: Fogo possui vantagem contra Planta, Planta vantagem contra Água, Água vantagem contra Fogo. Se Fogo atacar Planta, por exemplo, ele causará +5 de dano. Caso Fogo ataque Água, será -5 de dano. Caso Fogo ataque Fogo, não haverá alteração no dano

causado. As vantagens e desvantagens são um dos incrementos que se encontram no Método atacar() da Batalha.java.

3.3 Características: pontos de vida, nível de ataque, nível de defesa e percentual de ganho base do Power-Up:

Na Classe Personagem, há o atributo "vida" que se refere aos pontos de vida do personagem. Além disso, também há o Método Construtor que inicializa o ataque e defesa com base na classe escolhida pelo usuário.

O Power-Up é um incremento do ataque(), que, ao ser utilizado, o nível de ataque será o dobro mais 10, ou seja, o percentual é: 100% + 10.

3.4 .Cada tipo precisa de pelo menos uma ação especial:

No Método atacar(), da Classe Batalha.java, um dos outros incrementos é um contador que recebe a quantidade de dano causado. Caso o contador chegue a 100, o personagem poderá ativar o Poder Especial.

Guerreiro	Arqueiro	Mago
Escolhe um adversário para causar 35 de dano e receber 15 de dano.	Causa 21 de dano que pode ser dividido em até três adversários.	Restaura 35 de vida.

4. Utilização de OO para a implementação das Funcionalidades:

4.1 Herança:

Para a utilização de Herança, nós criamos uma Classe Abstrata chamada "Personagem.java", cuja finalidade é possuir todos os aspectos que um personagem deve ter, como: tribo, classe, vida etc. Em seguida, criamos mais três classes, Guerreiro.java, Arqueiro.java e Mago.java, que herdam os atributos da Personagem.java.

Personagem.java

```java
import java.util.*;

// Funcionalidade da Classe:
//  Servir como modelo para as Classes Arqueiro, Guerreiro, Mago.

public abstract class Personagem{
  private String nome;
  private int tribo; // Fogo [1] - Água [2] - Planta [3]
  private int classe; // Guerreiro [1] - Arqueiro [2] - Mago [3]

  private int vida = 200;
  private int ataque;
  private int defesa;
  private int defesaAtivada = 0; // [1] Ativada - [0] Desativada
  private int vidaRetirada = 0;
  private int percentualPowerUp = 0; // 1 Rodadas Ausentes

```

Guerreiro.java

```java
import java.util.*;

// Ataque = 20
// Defesa = 7
// Poder Especial - Dano de 35 Perder 15

class Guerreiro extends Personagem{
```

Arqueiro.java

```java
import java.util.*;

// Ataque = 15
// Defesa = 5
// Poder Especial - Causar 21 de dano divido em até 3 personagens (3 ataques de 7 de dano )

class Arqueiro extends Personagem{
```

Mago.java

```java
import java.util.*;

// Ataque = 10
// Defesa = 20
// Poder Especial - Regenera 35 de vida

class Mago extends Personagem{
```

4.2 Atributos e Métodos Encapsulados:

Na classe Personagem.java, colocamos atributos encapsulados com "private", a partir disso, geramos os Métodos "get" e "set".

Personagem.java

```java
// Get e Set Nome
public String getNome(){
  return this.nome;
}
public void setNome(String nome){
  this.nome = nome;
}

// Get e Set Tribo
public int getTribo(){
  return this.tribo;
}
public void setTribo(int tribo){
  this.tribo = tribo;
}

// Get e Set Classe
public int getClasse(){
  return this.classe;
}
public void setClasse(int classe){
  this.classe = classe;
}

// Get e Set Vida
public int getVida(){
  return this.vida;
}
public void setVida(int vida){
  this.vida = vida;
}

// Get e Set Ataque
public int getAtaque(){
  return this.ataque;
}
```

Personagem.java

```java
// Get e Set Ataque
public int getAtaque(){
  return this.ataque;
}
public void setAtaque(int ataque){
  this.ataque = ataque;
}

// Get e Set Defesa
public int getDefesa(){
  return this.defesa;
}
public void setDefesa(int defesa){
  this.defesa = defesa;
}

// Get e Set Defesa
public int getDefesaAtivada(){
  return this.defesaAtivada;
}
public void setDefesaAtivada(int defesaAtivada){
  this.defesaAtivada = defesaAtivada;
}

// Get e Set Percentual Power Up
public int getPercentualPowerUp(){
  return this.percentualPowerUp;
}
public void setPercentualPowerUp(int percentualPowerUp){
  this.percentualPowerUp = percentualPowerUp;
}

// Get e Set vidaRetirada
public int getVidaRetirada(){
  return this.vidaRetirada;
}
public void setVidaRetirada(int vidaRetirada){
  this.vidaRetirada += vidaRetirada;
}
```

4.3. Aplicação de Classes Abstratas, Polimorfismo e In-

terfaces:

4.3.1 Aplicação de Classe Abstrata e Polimorfismo:

Como citado anteriormente, criamos a Classe Abstrata Personagem.java para fazer a implementação das classes herdadas. Nela, contém o Método Abstrato que usamos para aplicar o Polimorfismo nas classes herdadas.

Personagem.java

```
109     public abstract boolean poderEspecial(int jogador, Vector<Personagem>
        personagem_partida);
110
111 }
```

Na imagem acima, criamos o Método Abstrato "poderEspecial". Como este é um Método Abstrato de uma Classe Abstrata, não há implementação dele na Classe Personagem.java, logo, a execução do Poder Especial é feita pelas classes herdeiras Guerreiro.java, Arqueiro.java e Mago.java, o que seria o Polimorfismo, pois o método poderEspecial terá diferentes execuções dependendo de qual classe o usuário escolher.

Guerreiro.java

```
12    public boolean poderEspecial(int jogador, Vector<Personagem>
      personagem_partida){
13      Scanner teclado = new Scanner(System.in);
14
15      int openente1;
16
17      // Tratamento de Exceções da variável oponente1 (Alvo Inválido/Inexistente)
18      try{
19        System.out.printf("\n Digite seu oponente: ");
20        openente1 = teclado.nextInt()-1;
21        personagem_partida.get(openente1);
22      } catch (InputMismatchException e) {
23        System.out.printf("\n\n  OPONENTE INVALIDO");
24        return false;
25      } catch (ArrayIndexOutOfBoundsException e){
26        System.out.printf("\n\n  OPONENTE INEXISTENTE");
27        return false;
28      }
29
30      // Execução do Poder Especial
31      personagem_partida.get(openente1).setVida(personagem_partida.get(openente1)
        .getVida()-35);
32      personagem_partida.get(jogador).setVida(personagem_partida.get(jogador)
        .getVida()-15);
33
34      return true;
35    }
36  }
```

Poder Especial do Guerreiro utilizando Polimorfismo:

Arqueiro.java

```java
public boolean poderEspecial(int jogador, Vector<Personagem> personagem_partida){

  Scanner teclado = new Scanner(System.in);
  int openente1;
  int openente2;
  int openente3;

  // Tratamento de Exceções das variáveis Oponentes (Alvos Inválidos/Inexistentes)
  try{
    System.out.printf("\n  Oponente1: ");
    openente1 = teclado.nextInt()-1;
    personagem_partida.get(openente1);

    System.out.printf("\n  Oponente2: ");
    openente2 = teclado.nextInt()-1;
    personagem_partida.get(openente2);

    System.out.printf("\n  Oponente3: ");
    openente3 = teclado.nextInt()-1;
    personagem_partida.get(openente3);

  } catch (InputMismatchException e) {
    System.out.printf("\n\n  VALOR INVALIDO");
    return false;
  } catch (ArrayIndexOutOfBoundsException e){
    System.out.printf("\n\n  OPONENTE INEXISTENTE");
    return false;
  }

  // Execução do Poder Especial
  personagem_partida.get(openente1).setVida(personagem_partida.get(openente1).getVida()-7);
  personagem_partida.get(openente2).setVida(personagem_partida.get(openente2).getVida()-7);
  personagem_partida.get(openente3).setVida(personagem_partida.get(openente3).getVida()-7);

  return true;
 }
}
```

Poder Especial do Arqueiro utilizando Polimorfismo:

Mago.java

```java
public boolean poderEspecial(int jogador, Vector<Personagem>
personagem_partida){

  // Execução do Poder Especial
  personagem_partida.get(jogador).setVida(personagem_partida.get(jogador)
  .getVida()+35);

  return true;
 }

}
```

Poder Especial do Mago utilizando Polimorfismo:

4.3.2 Aplicação de Interfaces:

Criamos a Interface "FuncionalidadesBatalha.java", para armazenar as assinaturas dos Métodos para assim puder implementar na classe Batalha.java. Além disso, criamos a Interface "FuncionalidadesArquivos" para implementar na Classe "ManipuladorArquivos.java".

FuncionalidadesBatalha.java

```java
public interface FuncionalidadesBatalha
{
  public void atacar(int atacante, int oponente);
  public void defender(int jogador);
  public void powerUp(int jogador);
  public void mostrarPersonagens();
  public int personagemVivos();
  public void luta();
  public boolean criarPersonagem();
  public void getNomesArquivos();
  public void mostrarArquivos();
  public boolean carregarPersonagem();
}
```

Interface FuncionalidadesBatalha.java:

FuncionalidadesArquivos.java

```
1  public interface FuncionalidadesArquivos
2  {
3    public void salvarPersonagem(String nome, int tribo,
     int classe);
4    public void salvarNomesArquivos(String nomesArquivos);
5    public void lerArquivo(String caminho);
6  }
```

Interface FuncionalidasArquivos.java:

4.4 Tratamento de Exceções:

Os tipos de erros tratados foram:

4.4.1 Erro de InputMismatchException:

Ocorre esta exceção quando o usuário digita um caractere ao invés de inteiro.

```
// Menu Principal
System.out.printf("\n *---------------------------------------* ");
System.out.printf("\n |               TRIFORCE                | ");
System.out.printf("\n *---------------------------------------* ");
System.out.printf("\n |       [1]. JOGAR                      | ");
System.out.printf("\n |       [2]. SAIR                       | ");
System.out.printf("\n *---------------------------------------* ");

Scanner teclado = new Scanner(System.in);

System.out.printf("\n\n  Digite sua escolha: ");

int respostaMenuPrincipal;

// Tratamento de Exceções da variável respostaMenuPrincipal
try{
  respostaMenuPrincipal = teclado.nextInt();
} catch (InputMismatchException e) {
  System.out.printf("\n\n   VALOR INVALIDO");
  Sistema.esperar();
  continue;
}
```

Acima, um exemplo de como foi tratado o erro no Menu Principal:

4.4.2 Erro de ArrayIndexOutOfBoundsException:

Este erro é disparado quando o usuário digita um Array maior que seu tamanho.

```java
// Tratamento de Exceções das variáveis Oponentes (Alvos Inválidos/Inexistentes)
try{
  System.out.printf("\n  Oponente1: ");
  openente1 = teclado.nextInt()-1;
  personagem_partida.get(openente1);

  System.out.printf("\n  Oponente2: ");
  openente2 = teclado.nextInt()-1;
  personagem_partida.get(openente2);

  System.out.printf("\n  Oponente3: ");
  openente3 = teclado.nextInt()-1;
  personagem_partida.get(openente3);

} catch (InputMismatchException e) {
  System.out.printf("\n\n  VALOR INVALIDO");
  return false;
} catch (ArrayIndexOutOfBoundsException e){
  System.out.printf("\n\n  OPONENTE INEXISTENTE");
  return false;
}
```

Acima, um exemplo de erro tratado no Poder Especial da Classe Arqueiro.java:

4.4.3 Erro de FileNotFoundException:

Este erro ocorre quando o programa tenta encontrar um arquivo inexistente.

4.4.4 Erro de IOException:

Esta exceção ocorre quando há erro na leitura de um Arquivo.

```java
// Salvar nomes no arquivo nomes Personagem
@Override
public void salvarNomesArquivos(String nomesArquivos){

  try {
    FileWriter file = new FileWriter
    ("ArquivosTexto/nomesPersonagem.txt", true);

    BufferedWriter buffWrite = new BufferedWriter(file);

    buffWrite.append(nomesArquivos + "\n");

    buffWrite.close();

    file.close();
  }
  catch (FileNotFoundException e){
    System.out.println("Arquivo não encontrado");
  }
  catch (IOException e) {
    System.out.println("Erro na leitura do arquivo");
  }

}
```

Acima, há o tratamento de erro de IOException e FileNotFoundException na Classe ManipuladorArquivos.java: