MC322
Primeiro semestre de 2022

Laboratório 7

---

1 Objetivo

O objetivo desta atividade consiste em praticar os conceitos de polimorfismo, e enumeração.

Esse lab é mais longo e usaremos 2 aulas de lab. para fazer a sua implementação.

OBS: Nos diagramas UML, quadrados vermelhos indicam membros privados e círculos verdes indicam membros públicos.

2 Atividade

Aproveite os arquivos dos laboratórios anteriores e crie uma hierarquia de classes como apresentada na Figura 1.

Figura 1: Hierarquia de classes do jogo LaMa

Hoje vamos aproveitar tal hierarquia para aplicarmos os conceitos de polimorfismo vistos em classe implementando um processador simples de jogadas. Mas, antes, são necessárias algumas implementações que serão fundamentais para construir tal processador.

2.1 Organização em pacotes das classes

- Crie um novo projeto chamado Lab7RAxxxxxx, onde xxxxxx deve ser substituído pelo seu RA.
- Reaproveite as classes criadas nos labs anteriores seguindo a organização mostrada abaixo.

Crie pacotes **base**, **base.cartas**, **base.cartas.magias** e **util** deixando seu projeto com um estrutura similar a apresentada na Figura 2:

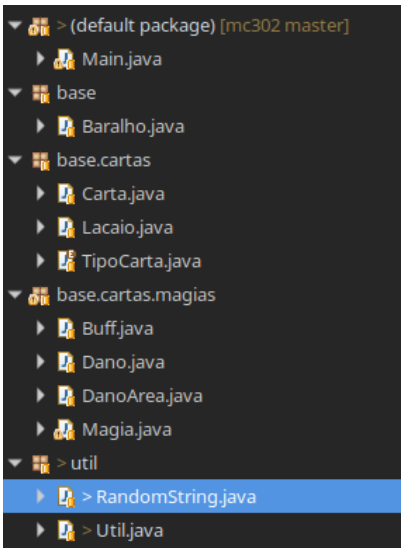

Figura 2: Estrutura do projeto

Lembre-se que cada classe pertencente a um determinado pacote, deve ter como primeira instrução no código fonte, a especificação de qual pacote a classe pertence. Por exemplo a classe **DanoArea** deve ter a instrução **package base.cartas.magias;**

Lembre-se também que para usar uma classe de um pacote em outro você deve importar tal pacote. Por exemplo a classe **Magia** é sub-classe de **Carta** e portanto, para que possamos extender esta classe, devemos em **Magia** fazer a importação **import base.cartas.Carta;**

2.2 Método usar(Carta)

Implemente este método nas classes **Lacaio** e **Dano** de tal forma que o método recebe uma **Carta** como parâmetro e aplica o dano sobre ela. O parâmetro para este lab. sempre será um **Lacaio**, então faça um downcast de **Carta** para o tipo **Lacaio**, e sete o novo valor de vida atual deste parâmetro decrementando sua vida atual pelo ataque do objeto **Lacaio** que teve o método invocado (ou pelo dano de **Dano** invocado).

Na classe **Buff** este método recebe uma **Carta** como parâmetro e deve aumentar o poder desta carta (que será sempre um **Lacaio**) com novos valores de ataque, vida atual e vida máxima pelos valores de **Buff** de aumento de ataque, e aumento de vida.

Na classe **DanoArea** tal método recebe uma lista de **Carta** como parâmetro e deve aplicar o seu valor de dano em todas as cartas da lista (de forma similar à carta **Dano**).

2.3 Enumeração

Como foi visto em classe, um tipo enumerado consiste em um conjunto de constantes representadas por identificadores únicos. Nessa atividade, iremos criar dois enumeradores com suas respectivas constantes:

- **TipoCarta:** LACAIO, BUFF, DANO, DANO_AREA
- **HabilidadesLacaio**: EXAUSTAO, PROVOCAR, INVESTIDA

Ambos devem ser criados dentro pacote `base.cartas`.

O enumerador **TipoCarta** será usado na geração de cartas aleatórios para especificarmos qual tipo de carta aleatória deve ser criada.

As habilidades de uma carta correspondentes ao enumerador **HabilidadesLacaio** constituem-se em:

- **EXAUSTAO**: O lacaio é baixado para a mesa, porém pode atacar somente no próximo turno.
- **PROVOCAR**: Quando um lacaio com essa habilidade está na mesa, todo ataque direcionado ao respectivo herói é direcionado unicamente para este lacaio.
- **INVESTIDA**: O lacaio pode atacar assim que é baixado para a mesa.

Adicione um campo privado à classe **Lacaio** do tipo **HabilidadesLacaio**, crie também os respectivos métodos `get` e `set` junto à uma sobrecarga do construtor que recebe a habilidade do lacaio como parâmetro.

2.4 Gerador de cartas aleatórias

De maneira a tornar a criação de cartas mais fácil para validação da implementação e também trabalhar com os conceitos de enumeração, vamos criar agora um método para geração de cartas aleatórias. Dentro da classe **Util** do pacote `util` crie o método com a seguinte assinatura:

```
1 public static Carta geraCartaAleatoria(Random gerador, int maxMana, int maxAtaque, int
      maxVida, TipoCarta tc)
```

Listing 1: Assinatura do método para gerar cartas aleatórias.

O primeiro argumento é um objeto da classe **Random** para geração de valores aleatórios. Os três próximos argumentos correspondem aos limitantes superiores para os valores de mana, ataque e vida para cada carta, respectivamente. O último argumento corresponde a qual tipo de carta deve ser gerado.

A implementação desse método deve seguir as seguintes regras:

- Os valores gerados para mana, ataque e vida devem estar no intervalo $[1, \max\{\text{Mana, Ataque, Vida}\}]$
- O último argumento pode receber o valor `null`, nesse caso, o tipo da carta também é sorteado aleatoriamente.
- A habilidade do lacaio deve ser sorteada aleatoriamente também.

Para gerar valores inteiros em um determinado intervalo, você pode usar o seguinte código:

```
1  public static int randInt(Random gerador, int min, int max) {
2          return gerador.nextInt((max - min) + 1) + min;
3  }
```

Listing 2: Geração de valor aleatório em intervalo fechado.

Perceba que também será necessário gerar nomes aleatórios para as cartas, o que pode ser realizado com a a classe **RandomString** do último lab:

```java
package util;
import java.util.Random;

// From: http://stackoverflow.com/a/41156
public class RandomString {

  private static final char[] symbols;

  static {
    StringBuilder tmp = new StringBuilder();
    for (char ch = '0'; ch <= '9'; ++ch)
      tmp.append(ch);
    for (char ch = 'a'; ch <= 'z'; ++ch)
      tmp.append(ch);
    symbols = tmp.toString().toCharArray();
  }

  private final Random random;

  private final char[] buf;

  public RandomString() {
    random = new Random();
  }

  public RandomString(Random gerador, int length) {
    if (length < 1)
      throw new IllegalArgumentException("length < 1: " + length);
    buf = new char[length];
    random = gerador;
  }

  public String nextString() {
    for (int idx = 0; idx < buf.length; ++idx)
      buf[idx] = symbols[random.nextInt(symbols.length)];
    return new String(buf);
  }
}
```

Listing 3: Classe RandomString.

Tal classe pode ser usada da seguinte maneira:

```java
RandomString stringGerador = new RandomString(gerador, MAX_NOME);
stringGerador.nextString();
```

Listing 4: Classe RandomString.

Onde `gerador` é um objeto da classe **Random** e `MAX_NOME` é a quantidade de caracteres desejada para a string.

Perceba que será necessário obter um valor do respectivo enumerador para gerar uma carta de qualquer tipo e também para atribuir alguma habilidade ao lacaio. A dica é que toda classe de enumeração possui o método estático `values()` que retorna um *array* com os valores das constantes do respectivo enumerador. *Arrays* em Java tem o atributo `length` que armazena o seu respectivo tamanho. A classe **Random** do Java possui o método `nextInt(int bound)`, que gera um número no intervalo $[0, \text{bound})$. Combinando esses dois recursos podemos obter um valor do enumerador aleatoriamente.

Uma vez criado o gerador de cartas aleatórias, crie um método com a seguinte assinatura na classe **Baralho**:

```
public void preencheAleatorio(Random gerador, int tamanho, int maxMana, int maxAtaque,
    int maxVida)
```

Listing 5: Preenche um baralho aleatoriamente.

Tal método deverá preencher o baralho com **min**{`Util.MAX_CARDS`, `tamanho`} geradas aleatoriamente.

2.5 Processador de jogadas

Todas as classes a seguir deverão ser criadas no pacote `base`.

Implementaremos agora classes auxiliares para o processador de jogadas. Tais classes são: **Jogada** e **Mesa**. A classe **Jogada** corresponde à Figura 3:

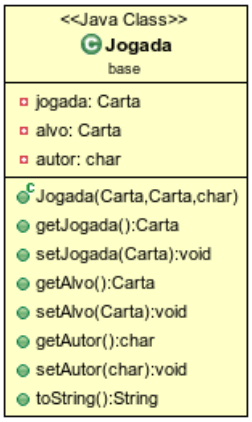

Figura 3: Classe **Jogada**

Os atributos `jogada`, `alvo` e `autor` correspondem à qual carta foi jogada, qual é o alvo e quem foi o autor da jogada, respectivamente. Quando o alvo é o herói adversário, o parâmetro `alvo` recebe o valor `null`. O autor pode assumir o valor do tipo `char` 'P' (**P**rimeiro jogador) ou 'S' (**S**egundo jogador). O método `toString()` deve retornar uma *string* com o autor, carta jogada e alvo.

A classe **Mesa** corresponde à Figura 4:

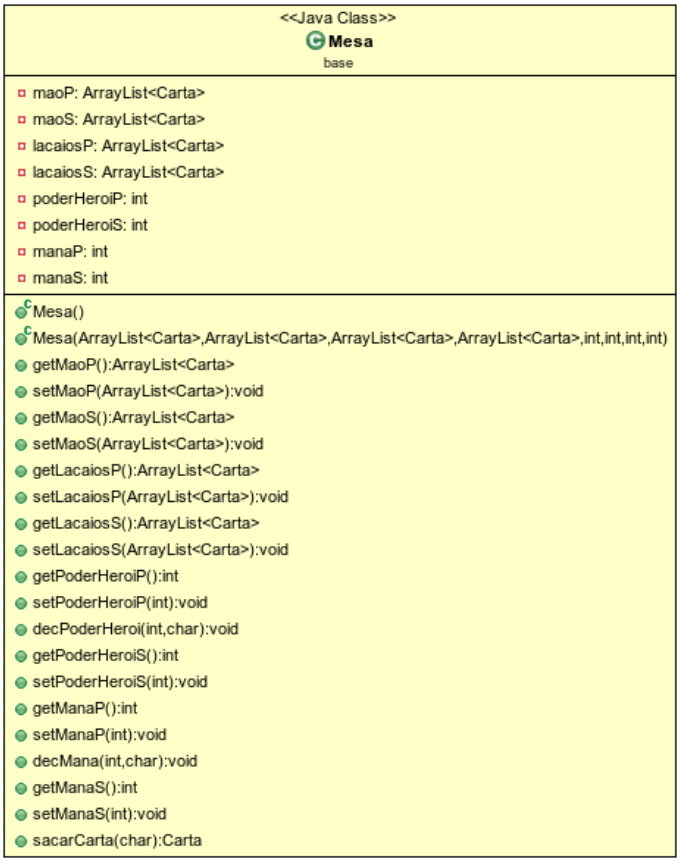

Figura 4: Classe **Mesa**

Os atributos `mao`, `lacaios`, `poderHeroi` e `mana` correspondem às cartas na mão, lacaios baixados, vida atual do herói e mana restante para cada um dos dois jogadores, respectivamente. Além dos métodos *getters* e *setters* dos atributos da classe, adicionam-se os seguintes métodos:

- `decPoderHeroi(int, char)`: Decrementa na quantidade do primeiro argumento o poder heróico do jogador corresponde ao segundo argumento.
- `decMana(int, char)`: Decrementa na quantidade do primeiro argumento a mana do jogador corresponde ao segundo argumento.
- `sacarCarta(char)`: **Retira** e retorna uma carta da mão do jogador especificado pelo argumento. A posição da qual a carta é retirada é arbitrária. Note que a carta deve ser removida da lista de cartas do jogador.

  O construtor padrão deve criar uma mesa sem cartas nas mãos dos jogadores, sem lacaios na mesa, com poder heróico de ambos jogadores igual a uma constante de valor arbitrário criada na classe **Util** (de nome `PODER_HEROI`) e com o valor de mana inicial em 1.

  Agora criaremos a classe **ProcessadorJogada**, correspondente ao diagrama da Figura 5:

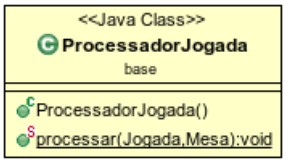

Figura 5: Classe **ProcessadorJogada**

Essa classe possui o método estático `processa(Jogada, Mesa)` que recebe como parâmetros a jogada a ser processada e um objeto mesa, respectivamente. Dados os parâmetros do método, sua implementação deve seguir a lógica descrita a seguir:

1. Decremente a mana do autor da jogada pelo custo de mana da carta jogada.
2. Verifique qual o tipo da carta jogada usada com o operador `instanceof`, se é:
   - **DanoArea**: Ataque todos os lacaios baixados pelo adversário e também decremente o poder heróico do adversário.
   - **Dano**: É necessário verificar se existe um lacaio com a habilidade `PROVOCAR` dentre os lacaios do adversário Caso exista, o respectivo lacaio deve ser atacado. Caso contrário, ataca-se a carta correspondente ao `alvo` do objeto da classe **Jogada**.
   - **Buff**: Só pode ser usada se o `alvo` é do tipo **Lacaio**.
   - **Lacaio**: É necessário verificar se a habilidade do lacaio é a de `EXAUSTAO`. Caso seja, o mesmo deve ser adicionado aos lacaios do respectivo jogador, agora com a habilidade de `INVESTIDA`. Caso seja `INVESTIDA`, ataca-se a carta correspondente ao `alvo` do objeto da classe **Jogada**. Caso seja `PROVOCAR`, aplica-se o mesmo comportamento da habilidade `EXAUSTAO`.
3. Atualize os lacaios de ambos jogadores, recuperando apenas aqueles que tem vida atual > 0.

Quando for usar uma carta em outra, lembre-se do método `usar` implementado anteriormente. A utilidade dele se mostra nesse método `processar`.

Por enquanto **não vamos nos preocupar se o jogador tem mana suficiente** para usar a carta jogada.

Para fins de manter um *log* da jogada, imprima antes de executar a jogada os seguintes dados:

- Autor da jogada;
- Carta da jogada;
- Carta alvo;
- Quantidade de e os lacaios do adversário;
- Poder heróico do adversário.

Após processar a jogada, imprima os dois últimos itens **atualizados**, de maneira a verificar se a jogada foi aplicada corretamente.

Em todas as implementações não se esqueça de manter os valores em estado válido em métodos nos quais são alterados.

Exemplo: Ao atacar um lacaio, não deixar a vida com valor negativo, o que é um estado inválido.

2.6 Validação

Para validar a implementação, vamos criar uma classe **Main** com um possível cenário do jogo LaMa.

- Crie dois baralhos, um para o primeiro jogador e outro para o segundo, preencha-os aleatoriamente com valores de max{Mana,Vida,Ataque} arbitrários.
- Crie um objeto da classe **Mesa** com o construtor padrão.
- Agora gere uma quantidade arbitrária de lacaios para cada um dos jogadores, como se fossem lacaios que estivessem baixados na mesa no momento, adicionando-os ao objeto mesa.
- Crie uma constante `MAO_INI` com o valor 3 na classe **Util** que armazena a quantidade de cartas que cada jogador recebe no início do jogo, em seguida, compre `MAO_INI` cartas para cada um dos jogadores (Adicionando ao objeto mesa). Lembre-se que o segundo jogador pode comprar uma carta a mais.
- Saque uma carta da mão do primeiro jogador e crie uma jogada com a mesma, tendo como alvo o herói adversário. Faça o mesmo para o segundo jogador.
- Saque uma carta da mão do primeiro jogador e crie uma jogada com a mesma, tendo como alvo um lacaio baixado pelo adversário selecionado aleatoriamente. Faça o mesmo para o segundo jogador. Observe que essas jogadas só são possíveis se o respectivo adversário possui lacaios na mesa.
- Para as jogadas criadas, invoque o método `processar` da classe **ProcessadorJogada**.

```
import base.Baralho;
import base.Jogada;
import base.Mesa;
import base.ProcessadorJogada;
import base.cartas.Carta;
import base.cartas.Lacaio;
import base.cartas.TipoCarta;
import base.cartas.magias.Dano;
import base.cartas.magias.DanoArea;
import util.Util;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Comparator;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.List;
import java.util.Random;
import java.util.TreeSet;

public class Main {

    public static void main(String[] args) {

        Mesa mesa = new Mesa();
        Baralho bP = new Baralho();
        Baralho bS = new Baralho();
        int maxLacaios = 10;
        int maxMana = 2;
        int maxAtaque = 6;
        int maxVida = 6;
        int index;
        Random gerador = new Random();
```

```
34          bP.preencheAleatorio(gerador, util.Util.MAX_CARDS, maxMana, maxAtaque, maxVida)
                ;
35          bS.preencheAleatorio(gerador, util.Util.MAX_CARDS, maxMana, maxAtaque, maxVida)
                ;
36
37          for (int i = 0; i < maxLacaios; i++) {
38              mesa.getLacaiosP().add(util.Util.geraCartaAleatoria(gerador, maxMana,
                    maxAtaque, maxVida, TipoCarta.LACAIO));
39              mesa.getLacaiosS().add(util.Util.geraCartaAleatoria(gerador, maxMana,
                    maxAtaque, maxVida, TipoCarta.LACAIO));
40          }
41
42          for (int i = 0; i < util.Util.MAO_INI; i++) {
43              mesa.getMaoP().add(bP.comprar());
44              mesa.getMaoS().add(bS.comprar());
45          }
46          mesa.getMaoS().add(bS.comprar());
47
48          Jogada jP = new Jogada(mesa.sacarCarta('P'), null, 'P');
49          ProcessadorJogada.processar(jP, mesa);
50          Jogada jS = new Jogada(mesa.sacarCarta('S'), null, 'S');
51          ProcessadorJogada.processar(jS, mesa);
52
53          System.out.printf("\n Jogadas com lacaios:\n");
54
55          index = gerador.nextInt(mesa.getLacaiosS().size());
56          jP = new Jogada(mesa.sacarCarta('P'), mesa.getLacaiosS().get(index), 'P');
57          ProcessadorJogada.processar(jP, mesa);
58          if (!mesa.getLacaiosP().isEmpty()) {
59              index = gerador.nextInt(mesa.getLacaiosP().size());
60              jS = new Jogada(mesa.sacarCarta('S'), mesa.getLacaiosP().get(index), 'S');
61              ProcessadorJogada.processar(jS, mesa);
62          }
63
64      }
65
66  }
```

Listing 6: Geração de valor aleatório em intervalo fechado.

3 Questões

Responda as seguintes questões em um **arquivo texto** e submeta junto ao código no Moodle:

- Por que é possível criar um baralho de cartas de diferentes classes (**Lacaio**, **Buff**, etc.)? Quando usamos o método `usar` dos respectivos objetos destas classes, qual implementação é utilizada?
- As constantes da classe **Util** nunca são alteradas durante a execução do programa. Qual palavra-chave da linguagem Java podemos usar para indicar que elas nunca serão alteradas?

4 Submissão

Submeta um arquivo lab7RAxxxxxx.zip com o seu lab compactado no formato zip no google classroom.