

MC302
Primeiro semestre de 2017

Laboratório 7

Professores: Esther Colombini (esther@ic.unicamp.br) e Fábio Luiz Usberti (fusberti@ic.unicamp.br)

PEDs: (Turmas ABCD) Elisangela Santos (ra149781@students.ic.unicamp.br), Lucas Faloni (lucasfaloni@gmail.com), Lucas David (lucasolividavid@gmail.com), Wellington Moura (wellington.tylon@hotmail.com)

PEDs (Turmas EF) Natanael Ramos (naelr8@gmail.com), Rafael Arakaki (rafaelkendyarakaki@gmail.com)

PAD: (Turmas ABCD) Igor Torrente (igortorrente@hotmail.com)

PAD: (Turmas EF) Bleno Claus (blenoclaus@gmail.com)

1 Objetivo

O objetivo desta atividade consiste em praticar os conceitos de polimorfismo, enumeração e tipos de relacionamentos (associação, agregação e composição).

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

2 Atividade

Nos laboratórios anteriores criamos a hierarquia de classes apresentada na Figura 1. 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.

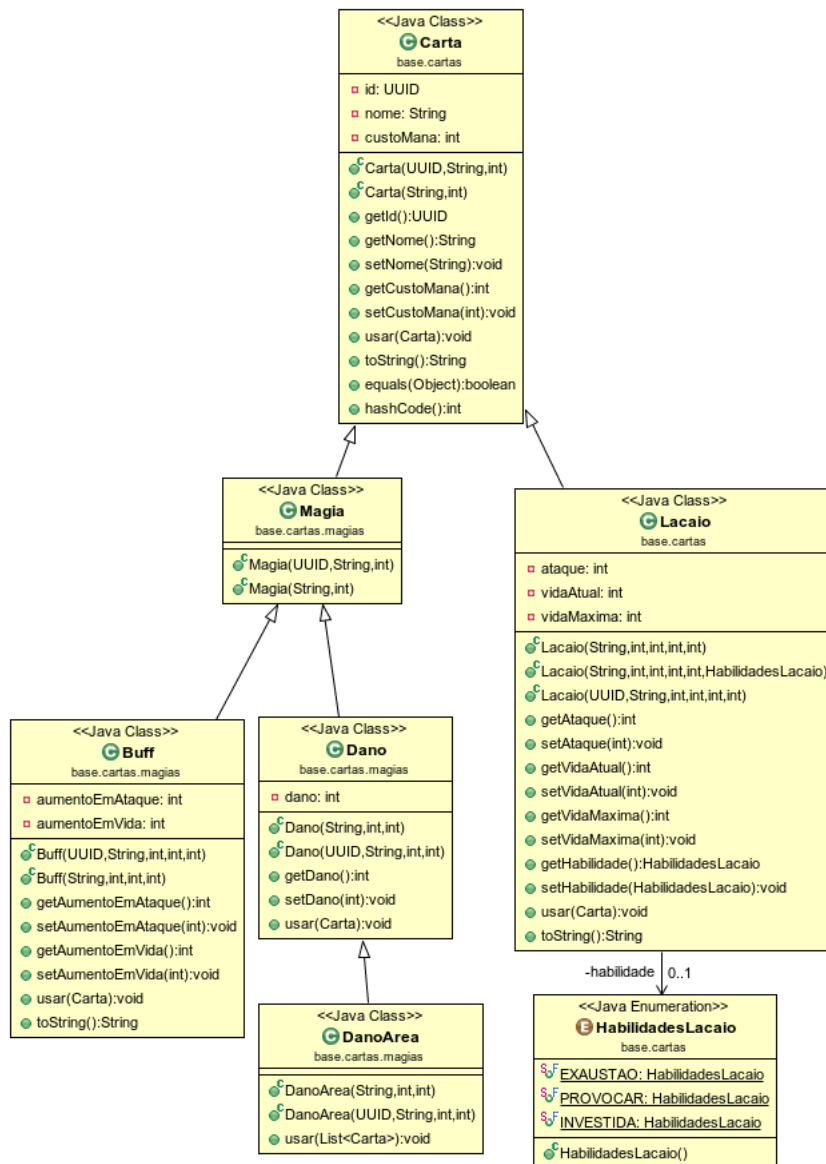


Figura 1: Hierarquia de classes do jogo LaMa

É importante que você tenha a estrutura de pacotes/classes dos laboratórios anteriores.

- Crie um novo projeto chamado Lab7.
- Copie a mesma estrutura de diretórios e classes utilizada no lab6.

A estrutura do seu projeto deverá ficar similar a apresentada na Figura 2:

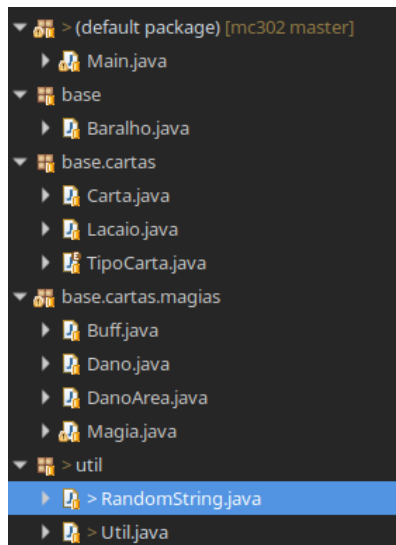


Figura 2: Estrutura do projeto

2.1 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`.

Já foram vistos os tipos de cartas descritos no enumerador **TipoCarta** no laboratório 4. 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.

2.2 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 $[1, \max\{\text{Mana}, \text{Ataque}, \text{Vida}\}]$
- O último argumento pode receber o valor `null`, nesse caso, o tipo da carta também é sorteado aleatoriamente.
- A habilidade do laçao deve ser sorteada aleatoriamente também.

Para gerar valores 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 seguinte classe **RandomString**:

```
1 package util;
2 import java.util.Random;
3
4 // From: http://stackoverflow.com/a/41156
5 public class RandomString {
6
7     private static final char[] symbols;
8
9     static {
10         StringBuilder tmp = new StringBuilder();
11         for (char ch = '0'; ch <= '9'; ++ch)
12             tmp.append(ch);
13         for (char ch = 'a'; ch <= 'z'; ++ch)
14             tmp.append(ch);
15         symbols = tmp.toString().toCharArray();
16     }
17
18     private final Random random;
19
20     private final char[] buf;
21
22     public RandomString(Random gerador, int length) {
23         if (length < 1)
24             throw new IllegalArgumentException("length < 1: " + length);
25         buf = new char[length];
26         random = gerador;
27     }
28
29     public String nextString() {
30         for (int idx = 0; idx < buf.length; ++idx)
31             buf[idx] = symbols[random.nextInt(symbols.length)];
32         return new String(buf);
33     }
34 }
```

Listing 3: Classe RandomString.

Tal classe pode ser usada da seguinte maneira:

```
1 RandomString stringGerador = new RandomString(gerador, MAX_NOME);
2 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, bound)`. Combinando esses dois recursos, como 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**:

```
1 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 $\min\{\text{Util.MAX_CARDS}, \text{tamanho}\}$ geradas aleatoriamente.

2.3 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:

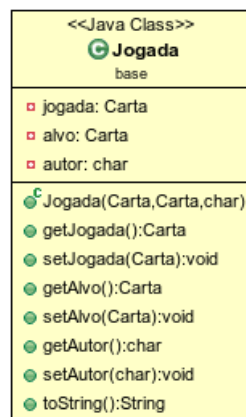


Figura 3: Classe **Jogada**

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**rimero 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:

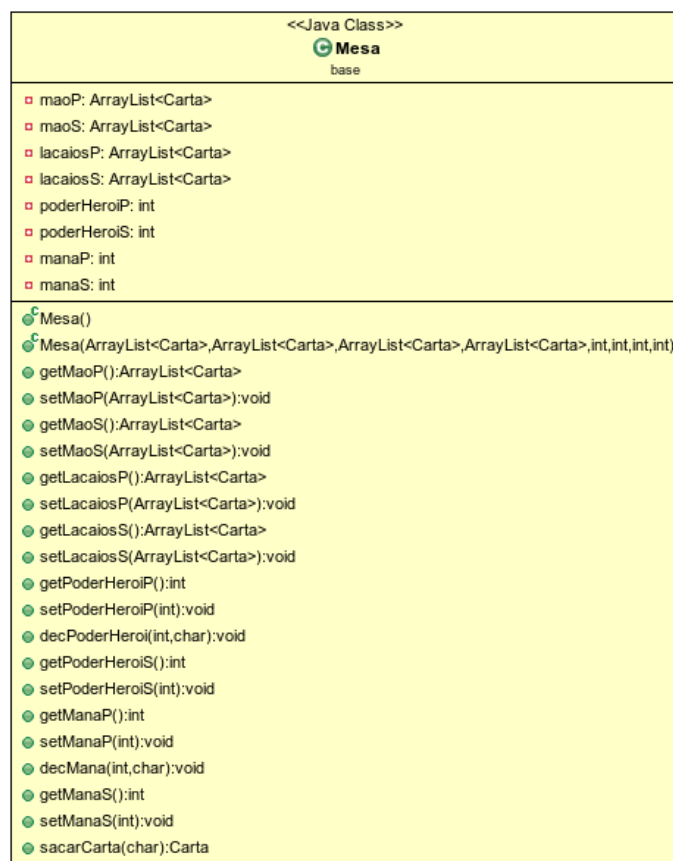


Figura 4: Classe **Mesa**

Os atributos `mao`, `lacaioS`, `vidaHerói` 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:

- `decPoderHerói(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.

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:

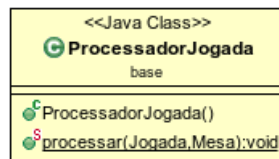


Figura 5: Classe **ProcessadorJogada**

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 o operador `instanceof`, se é:
 - **DanoArea**: Ataque todos os lacaio baixados pelo adversário e também decumente o poder heróico do adversário.
 - **Dano**: É necessário verificar se existe um lacaio com a habilidade `PROVOCAR` dentre os lacaio do adversário (**Dica**: Podemos usar `streams` como aprendemos no lab anterior). 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 lacaio do respectivo jogador, agora com a habilidade de `INVESTIDA`. Caso contrário, ataca-se a carta correspondente ao `alvo` do objeto da classe **Jogada**.
3. Atualize os lacaio de ambos jogadores, recuperando apenas aqueles que tem vida atual > 0 (**Dica**: Novamente, `streams` podem ser usadas.)

Quando for usar uma carta em outra, lembre-se do método `usar` implementado nos laboratórios anteriores, 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 lacaio 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.

2.4 Validação

Para validar a implementação, vamos criar 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 $\max\{\text{Mana, Vida, Ataque}\}$ arbitrários.
- Crie um objeto da classe **Mesa** com o construtor padrão.
- Agora gere uma quantidade arbitrária de lacaio para cada um dos jogadores, como se fossem lacaio 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 lacaio na mesa.
- Para as jogadas criadas, invoque o método `processar` da classe **ProcessadorJogada**.

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?
- Qual a relação (associação, agregação, composição) entre as classes **Mesa** e **Carta**? Por que?
- Qual a relação (associação, agregação, composição) entre as classes **ProcessadorJogada** e **Mesa**? Por que?
- 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

Para submeter a atividade, utilize o Moodle (<https://www.ggte.unicamp.br/ea>). Crie um arquivo texto com as respostas para cada item da seção tarefas e as saídas geradas pelo código. Compacte o código-fonte contido no diretório `src` juntamente com arquivo de respostas no formato `.zip` ou similar e nomeie-o **Lab7-000000.zip**, trocando '000000' pelo seu número de RA. Submeta o arquivo na seção correspondente para esse laboratório no moodle da disciplina MC302.

Certifique-se de entregar um código compilável, incluindo todas as subpastas dos pacotes em prol de facilitar a correção.

Datas de entrega

- Dia **09/05** Turma **ABCD** até às 23:55h
- Dia **13/05** Turma **EF** até às 23:55h