MC302
Primeiro semestre de 2017

Laboratório 9

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 (lucasolivdavid@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 deste laboratório é praticar conceitos de Exceções.

2 Atividade

Novamente, utilizaremos o código do laboratório anterior para exercitar conceitos de exceções e controle de erros.

1. Crie um novo projeto Java com o nome Lab9.
2. Reutilize o projeto do Lab8, para isso basta copiar e colocar as estrutura interna de um projeto no outro.

2.1 Exceções

Além do material da disciplina, é possível consultar o material disponível em: `www.caelum.com.br/apostila-java-orientacao-objetos/excecoes-e-controle-de-erros/`

2.2 Novas classes

Neste laboratório vamos separar ainda mais as responsabilidades no nosso código.

- Crie uma interface chamada *BaralhoService*, assim como ilustrado na Figura 2. O método `preencheAleatorio(...)`, que anteriormente estava na classe *Baralho*, agora será implementado em *BaralhoServiceImpl*. Mudaremos o retorno do método para `List<Carta>`. Atente para o código disponível da classe *Controle*. Perceba que eliminamos o método `preencheBaralho()` da classe *Controle*, o que antes ele executava agora será feito no método `executar()`.
- Crie uma interface chamada *MesaService*, assim como ilustrado na Figura 2. Essa interface conterá os dois métodos ilustrados na Figura 2. Novamente, a lógica, que antes pertencia a classe *Controle*, não será alterada, apenas mudaremos o local onde ela será realizada. A classe *MesaServiceImpl* conterá a lógica dos dois métodos `adicionaLacaios` e `addMaoInicial`, como podemos notar pela chamada de ambos na classe *Controle*.

- Vamos deletar a classe *ProcessadorControle*, que estava sendo usada apenas para chamar a Interface *ProcessadorService*. A única alteração que precisaremos fazer é declarar *ProcessadorService* na classe *Controle* e substituir os nomes nos locais em que antes era utilizado *ProcessadorControle*.
- Adicione à classe *Util* as variáveis inteiras que pertencem à classe *Controle*: maxLacaios, maxMana, maxAtaque e maxVida. Chamaremos elas de MAX_LACAIOS, MAX_MANA, MAX_ATAQUE e MAX_VIDA, respectivamente. Substitua o uso onde for necessário e apague as variáveis da classe *Controle*. Observe na seção 2.3. Não usaremos a variável *index* em *Controle*, então pode ser eliminada.
- Instanciações de classes estão sendo feitas no construtor *default*.
- Observe a Figura 1, espera-se que as dependências entre pacotes esteja dessa forma, após as alterações propostas. Note que não temos dependências circulares dessa forma. A ferramenta utilizada para obter essa informação se chama STAN, um *plug-in* do Eclipse. Recomendo que instalem, rodem nos laboratórios e analisem as diferenças.
- Crie um pacote chamado: *base.exception*. Crie duas exceções, *BaralhoVazioException* e *MesaNulaException*, fazendo com que herdem de *IllegalArgumentException* e *Exception*, respectivamente. Sobrescreva os construtores de ambas, passando como parâmetro uma *String*, que será a mensagem exibida quando forem invocadas.

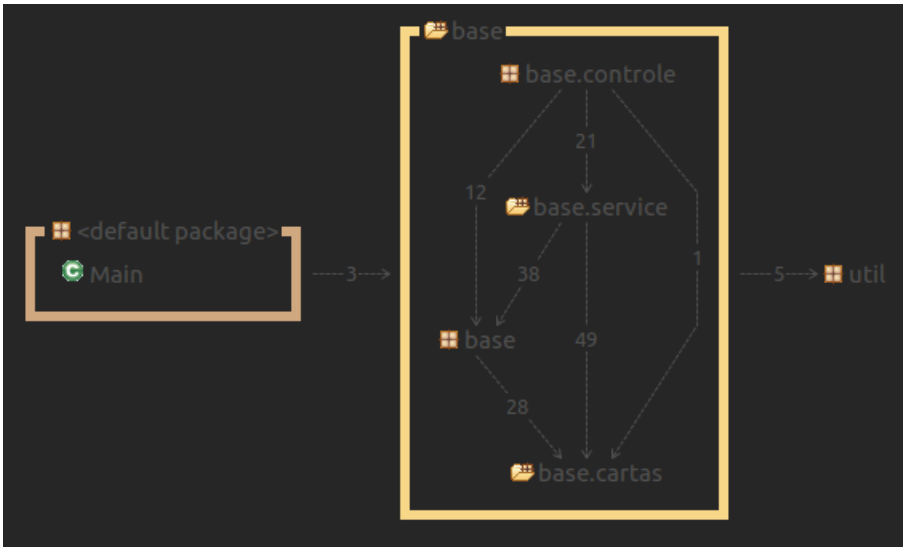

Figura 1: Organização de pacotes

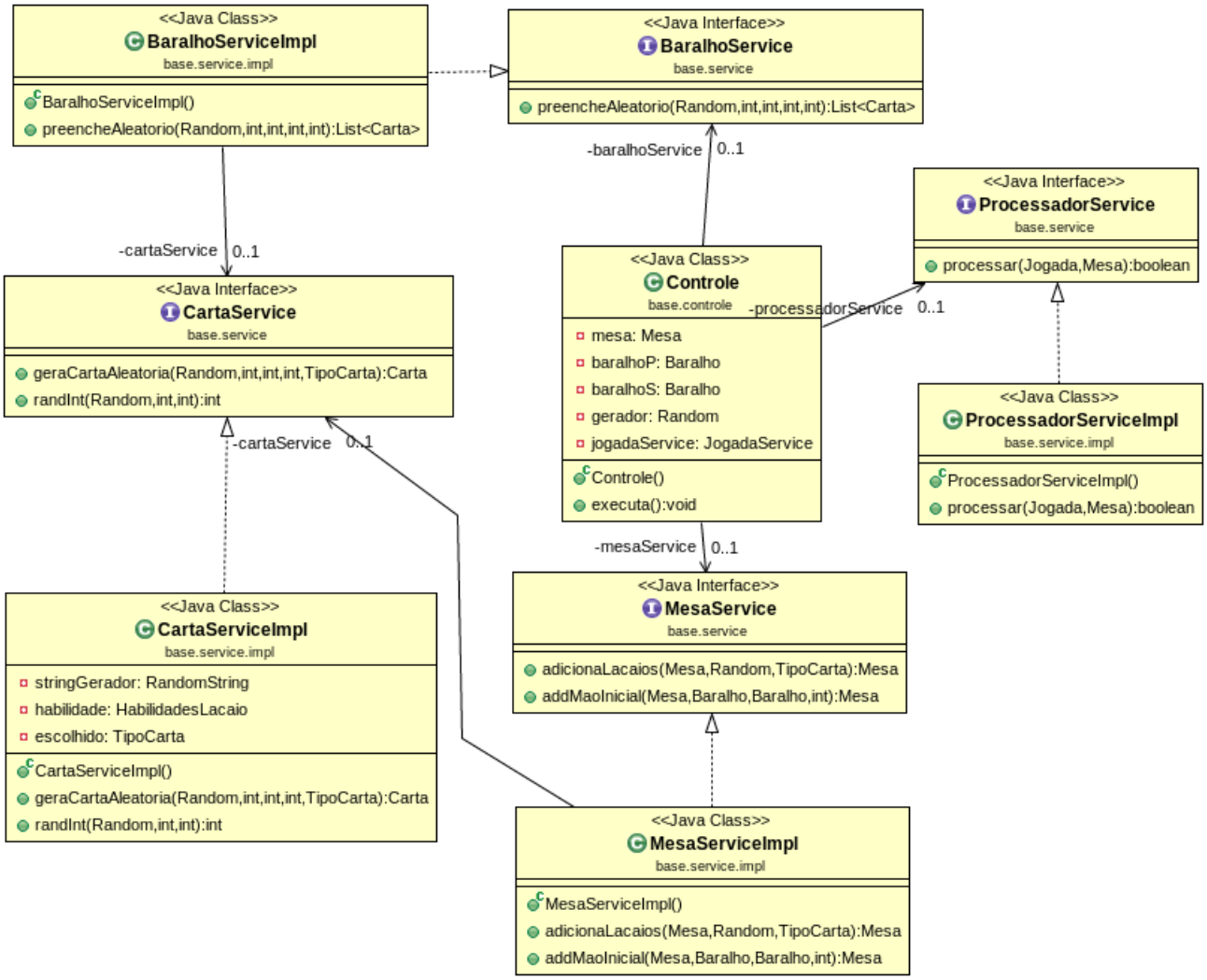

Figura 2: Organização de classes

2.3 Classe Util

```java
package util;

public class Util {
    public final static int MAX_CARDS = 30;
    public final static int MAX_NOME = 5;
    public final static int MAO_INI = 3;
    public final static int MAX_TURNOS = 10;
    public final static int PODER_HEROI = 10;
    public final static int MANA_INI = 1;
    public final static int MAX_LACAIOS = 10;
    public final static int MAX_MANA = 2;
    public final static int MAX_ATAQUE = 6;
    public final static int MAX_VIDA = 6;
}
```

2.4 Classe Controle

```java
package base.controle;

import static util.Util.MAO_INI;
import static util.Util.MAX_ATAQUE;
import static util.Util.MAX_CARDS;
import static util.Util.MAX_MANA;
import static util.Util.MAX_VIDA;
//Alguns imports omitidos
public class Controle {

    private Mesa mesa;
    private Baralho baralhoP;
    private Baralho baralhoS;
    private Random gerador;
    private JogadaService jogadaService;
    private BaralhoService baralhoService;
    private MesaService mesaService;
    private ProcessadorService processadorService;

    public Controle() {
        this.baralhoP = new Baralho();
        this.baralhoS = new Baralho();
        this.mesa = new Mesa();
        gerador = new Random();
        jogadaService = new JogadaServiceAgressivaImpl();
        baralhoService = new BaralhoServiceImpl();
        mesaService = new MesaServiceImpl();
        processadorService = new ProcessadorServiceImpl();
    }

    public void executa() {

        baralhoP.addCartas(baralhoService.preencheAleatorio(gerador, MAX_CARDS,
            MAX_MANA, MAX_ATAQUE, MAX_VIDA));
        baralhoS.addCartas(baralhoService.preencheAleatorio(gerador, MAX_CARDS,
            MAX_MANA, MAX_ATAQUE, MAX_VIDA));

        mesa = mesaService.adicionaLacaios(mesa, gerador, TipoCarta.LACAIO);
        mesa = mesaService.addMaoInicial(mesa, baralhoP, baralhoS, MAO_INI);


        List<Jogada> jogadas = jogadaService.criaJogada(mesa, 'P');
        for (Jogada jogada : jogadas) {
            if (processadorService.processar(jogada, mesa)) {
                System.out.println("###### " + jogada.getAutor() + " venceu!");
                break;
            }
        }

    }
}
```

3 Questões

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

1. Na classe *Controle*, na chamada do método preencheAleatorio, de *BaralhoService*, substitua o parâmetro *gerador* por *null*. Qual exceção é lançada? Qual sua estratégia para analisar a pilha de exceções que é lançada e localizar o erro?

2. Utilizaremos *BaralhoVazioException* para verificar se o método *preencheAleatorio*, em *BaralhoServiceImpl*, realmente retornará uma lista não vazia. Para isso, basta que seja lançada uma exceção caso a lista esteja vazia, ao final do processo em *preencheAleatorio*, impedindo que seja retornada uma lista vazia.

3. Utilizaremos *MesaNulaException* para verificar se o atributo *mesa* não é nulo. Em *adicionarLacaios*, na classe *MesaServiceImpl*, faça uma verificação se o atributo *mesa* é nulo, caso afirmativo lance essa exceção. Ao chamar `throw new MesaNulaException(''Sua mensagem'');` o que houve de diferente em relação ao item anterior, ou seja, quando chamamos *BaralhoVazioException*? Delegue a responsabilidade de tratar a exceção para *Controle*.

4. Por que é preferível que se use exceções do próprio *Java*, como *IllegalArgumentException*, para resolver problemas como o do item anterior (mesa nula)?

5. Crie uma nova exceção chamada, *ValorInvalidoException*. Essa será usada para verificar se valores são inválidos, ex.: quantidade de cartas na mão inicial dos jogadores. Em *MesaServiceImpl* verifique se o valor de *maoIni* é menor que 3, em caso afirmativo *ValorInvalidoException* deve ser lançado e exibir uma mensagem com o valor de *maoIni*.

6. O que podemos fazer para garantir que uma mensagem de "Partida encerrada" seja sempre exibida pelo nosso código, mesmo que ocorra exceções durante o processo? Faça alterações no método *executar* de *Controle* para conseguir tal efeito.

7. Por que o trecho de código a seguir é uma má prática?

```
1 } catch(Exception e) {
2     System.out.println(e);
3 }
```

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 **Lab9-000000.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 **30/05** Turma **ABCD** até às 23:55h
- Dia **26/05** Turma **EF** até às 23:55h