

# Cassino Solitário – Milestone II

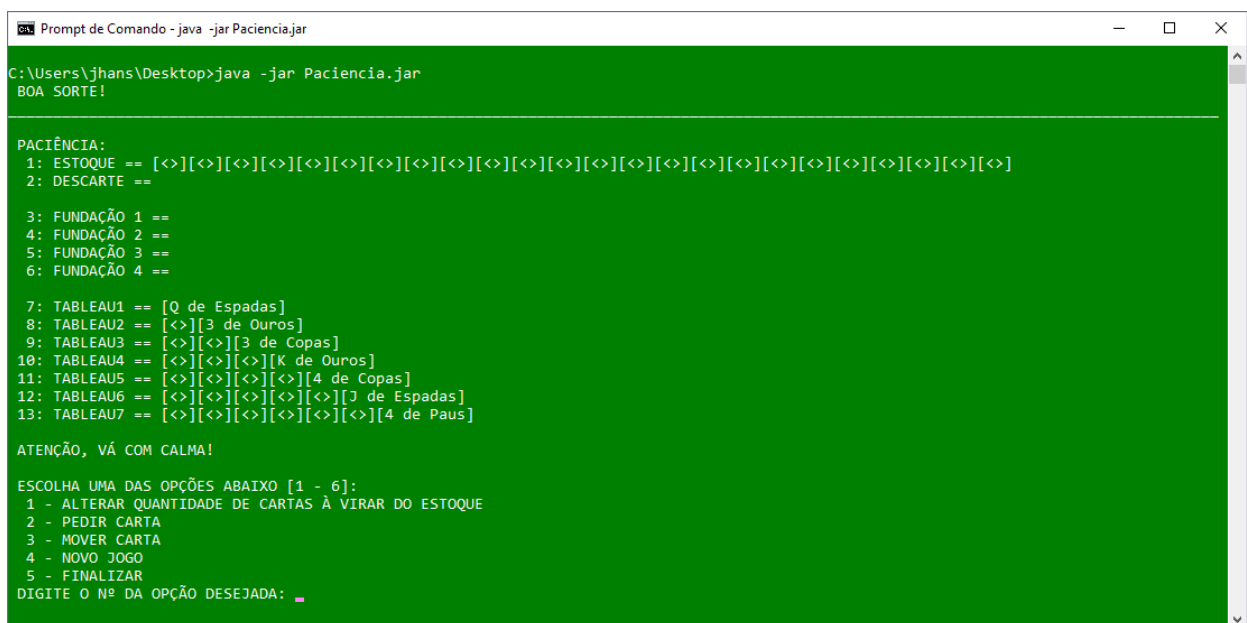
Universidade Estadual de Feira de Santana - UEFS  
EXA 836 – Padrões e Frameworks, Engenharia de Computação

Jhansen Barreto

jhansen.ecomp@gmail.com

## 1. Descrição: Jogando o *Paciência*

O projeto *Cassino Solitário* teve sua primeira parte findada com a entrega do Milestone II. Nesta etapa, o projeto apresentado no Milestone anterior foi implementado, dando vida ao primeiro jogo do projeto, o *Paciência*.



```
Prompt de Comando - java -jar Paciencia.jar

C:\Users\jhans\Desktop>java -jar Paciencia.jar
BOA SORTE!

PACIÊNCIA:
1: ESTOQUE == [<>][<>][<>][<>][<>][<>][<>][<>][<>][<>][<>][<>][<>][<>][<>][<>][<>][<>]
2: DESCARTE ==

3: FUNDAÇÃO 1 ==
4: FUNDAÇÃO 2 ==
5: FUNDAÇÃO 3 ==
6: FUNDAÇÃO 4 ==

7: TABLEAU1 == [Q de Espadas]
8: TABLEAU2 == [<>][3 de Ouros]
9: TABLEAU3 == [<>][<>][3 de Copas]
10: TABLEAU4 == [<>][<>][<>][K de Ouros]
11: TABLEAU5 == [<>][<>][<>][<>][4 de Copas]
12: TABLEAU6 == [<>][<>][<>][<>][<>][J de Espadas]
13: TABLEAU7 == [<>][<>][<>][<>][<>][<>][4 de Paus]

ATENÇÃO, VÁ COM CALMA!

ESCOLHA UMA DAS OPÇÕES ABAIXO [1 - 6]:
1 - ALTERAR QUANTIDADE DE CARTAS À VIRAR DO ESTOQUE
2 - PEDIR CARTA
3 - MOVER CARTA
4 - NOVO JOGO
5 - FINALIZAR
DIGITE O Nº DA OPÇÃO DESEJADA: 
```

Figura 1: Jogo *Paciência*

Como pode ser notado na figura acima, a interface do jogo é totalmente caracter, algo que era requisito do projeto para essa primeira parte. Dessa forma, o usuário interage com o jogo apenas digitando os números correspondentes às ações desejadas.

Ao escolher a primeira opção do menu de jogo, o usuário está optando em virar 1 (uma) ou 3 (três) cartas da pilha de estoque. Escolhendo apenas 1 (uma) carta, a pilha de descarte recebe essa carta de face para cima e sempre mostra apenas a última carta recebida, que é a carta do topo. Escolhendo 3 (três) cartas, a pilha de descarte recebe e passa a mostrar sempre as 3 (três) últimas cartas recebidas, seguindo a mesma lógica, sendo que a primeira carta mostrada na fila é a carta do topo e é a única que pode ser movida para uma outra pilha.

A segunda opção é a responsável por “pedir carta”. Ela é quem move as cartas da pilha de estoque para descarte, de acordo com a quantidade definida pelo usuário, citada no parágrafo anterior. Caso o usuário não configure essa quantidade, por padrão o jogo inicia virando apenas uma carta.

Na terceira opção, o jogador pode mover as cartas, seguindo as regras universais do jogo, também descritas nos requisitos do projeto.

Escolhendo a quarta opção é possível iniciar um novo jogo, onde o programa reinicia o jogo com um novo baralho, embaralhado de maneira diferente da anterior. Por fim, a quinta e última opção deve ser escolhida sempre que o usuário desejar finalizar a partida, porém, caso o jogador consiga mover todas as cartas para as pilhas de fundações, satisfazendo assim a condição de vitória, o jogo o parabeniza e se encerra por conta própria, sem a necessidade do jogador finalizar a partida.

## 2. Modelo Conceitual e Especificações

A figura abaixo traz o modelo conceitual da arquitetura do projeto, seguida de uma explicação do comportamento e interação entre as entidades.

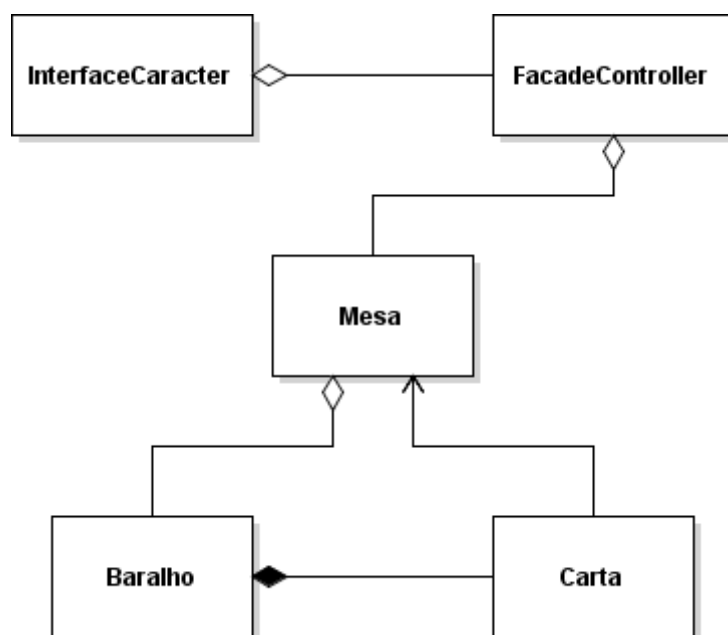


Figura 2: Modelo Conceitual

Como mostra a Figura 2, a *InterfaceCaracter*, que é a classe responsável apenas por exibir o estado do jogo e coletar as ações do usuário, não conhece nenhuma outra entidade do sistema além do *FacadeController*, classe que é responsável por repassar as ações do usuário para a *Mesa*, que é a classe que vai realizar as ações.

Dessa forma, ao iniciar uma partida, a *Mesa* recebe uma coleção de objetos *Carta*, criada pela classe *Baralho*, faz a distribuição inicial do jogo colocando as cartas em suas respectivas pilhas e fazendo todos os movimentos solicitados pelo jogador. Como já parece claro, *Mesa* é a entidade que simula a mesa de jogo real.

### 2.1 Implementação

Cada entidade foi implementada com suas responsabilidades bem definidas, de acordo com as necessidades e limitações atreladas ao projeto direta ou indiretamente.

#### 2.1.1 InterfaceCaracter

Essa classe implementa apenas 5 (cinco) métodos muito simples. Um método para imprimir o estado de cada pilha, um para imprimir o menu do jogo, um para capturar a escolha do jogador e

direcionar para o façade, um para interagir com o jogador na escolha entre virar 1 (uma) ou 3 (três) cartas do estoque e outro para interagir na escolha de movimento de cartas, perguntado pilha de origem, pilha de destino e quantidade de cartas à serem movidas.

### 2.1.2 FacadeController

Classe que só repassa as chamadas da interface. É a classe que conhece a *Mesa*, faz todos os pedidos do jogador e retorna para a interface tudo que foi pedido, após a *Mesa* realizar a ação.

### 2.1.3 Mesa

Essa é a classe com maior peso no sistema. É responsável por toda a funcionalidade, fazendo o intermédio entre as pilhas. Implementa diversos métodos simples que apenas retornam o estado das pilhas, mostrando a organização das cartas, além dos métodos que movem as cartas entre as pilhas desejadas e faz o saque das cartas no estoque, movendo-as para a pilha de descarte.

Para mover as cartas, *Mesa* implementa 5 (cinco) métodos que são na verdade a implementação das regras do jogo no que se refere ao movimento de cartas entre pilhas. É possível mover apenas 1 (uma) carta por vez da pilha de descarte para uma pilha de fundação, apenas 1 (uma) carta por vez da pilha de descarte para uma pilha de fileira, apenas 1 (uma) carta por vez de uma pilha de fundação para uma pilha de fileira, apenas 1 (uma) carta por vez de uma pilha de fileira para uma pilha de fundação e várias cartas entre pilhas de fileiras.

Além desses métodos, a classe implementa um outro método que verifica se a condição de vitória foi aceita.

### 2.1.4 Baralho

Essa classe é muito simples e tem como responsabilidade apenas instanciar as 52 (cinquenta e duas) cartas necessárias para uma partida de *Paciência*.

### 2.1.5 Carta

Essa classe define o objeto carta e implementa os métodos necessários para dar suporte ao uso do objeto. É possível “esconder carta”, “mostrar carta” e coletar todas as informações necessárias de uma carta, como: cor, naipe, número/ícone.

## 3. Padrões de Projeto

Como já foi mostrado na Figura 2 e explicado na seção 2, a classe *FacadeController* é a responsável por separar a interface de interação com o usuário do resto do sistema, o que configura a utilização do padrão de projeto *Façade*.

A utilização do padrão traz alguns benefícios para o projeto, dentre eles: reduzir a complexidade liberando acesso a métodos de alto nível e encapsulando os demais, produz uma interface comum e simplificada, reduz o acoplamento entre as camadas do projeto, etc.

Esse padrão foi adotado também pensando nas futuras modificações. É uma interface comum que pode intermediar outras classes, quando a segunda parte do Projeto Cassino Solitário for implementada.

### 3.1 Padrões de Projeto que podem ser Utilizados

Um padrão que não foi implementado no projeto mas pode ajudar em futuras alterações é o padrão *Strategy*. Esse padrão tem como objetivo criar “estratégias” para cada variante e fazer com que o método delegue o algoritmo para uma instância. Por exemplo: no *Paciência* é necessário ter métodos para mover carta. Com o uso do *Strategy*, o algoritmo para mover cartas pode ter diversas variantes, tendo assim o “mover carta” para tipos diferentes de jogos de cartas.

#### 4. Pontos fortes e Pontos fracos

A baixa utilização de padrões de projeto dificulta um pouco a manutenibilidade do código, pois, nem tudo pode ser reaproveitado, forçando assim a implementação de outra estratégia específica para alguma ação, já que pode não haver uma estratégia genérica. A classe *Mesa* por exemplo, poderia implementar o movimento das cartas de maneira diferente, para ser reaproveitado em outros jogos, contudo, a carga de trabalho para um único desenvolvedor entregar o projeto no prazo estabelecido foi determinante para que algumas decisões precisassem ser tomadas, e a maior decisão foi priorizar o total funcionamento do jogo solicitado.

A primeira parte do projeto foi totalmente finalizada com sucesso. O jogo funciona completamente, seguindo todas as especificações requisitadas.

- Iniciar jogo: O estado inicial do jogo acontece conforme as regras, com a distribuição necessária. O jogador pode iniciar um novo jogo a qualquer momento da partida, trazendo um baralho misturado de forma diferente e redistribuindo as cartas na mesa.
- Virar carta do estoque: A carta do estoque será virada para a pilha de descarte. É possível configurar o programa para virar 1 (uma) ou 3 (três) cartas, a qualquer momento do jogo.
- Virar carta da pilha de fileira: Essa é uma ação automática do jogo. Quando todas as cartas com face virada para cima são movidas para outra pilha e o topo da fileira fica com uma carta virada para baixo, essa carta é automaticamente revelada, sem a necessidade do pedido do jogador.
- Pedir dados da carta: Sempre que uma carta está com a face virada para cima, ela é apresentada com seu respectivo valor e naipe, se estiver virada para baixo é apresentada a seguinte sequência “[< >]”.
- Mover cartas: Todas as regras foram implementadas corretamente e todos os movimentos possíveis podem ser realizados.
- Parabenizar: O jogo verifica a condição de vitória, parabeniza o jogador e finaliza a partida.
- Encerrar programa: O jogador pode finalizar a partida a qualquer momento, caso não deseje continuar jogando.

Além de todos os requisitos alcançados, a vista (interface com o usuário) implementada totalmente caracter é muito simples de interagir, bem intuitiva e objetiva.