

Alunos: Caíque Bruno Fortunato
Pâmela Carvalho da Silva

Matrícula: 2013062731
2013073474

Trabalho Prático 2 - Cartas na mesa Implementação do jogo Burro





















































1. Introdução

a. Contextualização

O baralho é um conjunto de cartas numeradas que possuem numeração própria e um naipe (representação gráfica de um grupo), cuja utilização principal ocorre em diferentes jogos entre diversos jogadores.

Sendo utilizado em diversas partes do mundo, o baralho possui diferentes versões aplicadas a diferentes tipos de jogos. Uma das versões é a francesa, cujo baralho possui 52 cartas numeradas de 1 a 13, onde o número 1 é representado como A (as), o 11 como J (Valete), o 12 como Q (Damas) e, finalmente o 13 como K (Rei). Cada uma dessas 13 cartas possuem variações entre 4 naipes: copas, ouros, espadas e paus.

A imagem abaixo ilustra um baralho francês completo.

	1	2	3	4	5	6	7	8	9	10	Valete	Dama	Rei
Paus:													
Ouros:													
Copas:													
Espadas:													

Entre vários jogos composto por baralho francês existe um popularmente chamado de Burro no Brasil, que é conhecido por sua simplicidade, tempo curto de duração e diversão rápida entre jogadores.

Além disso, geralmente quem perde esse jogo geralmente paga uma prenda, que é seu diferencial em rodada entre amigos.

Este trabalho prático visa a implementação do Jogo Burro com o paradigma de Orientação a Objetos utilizando conceitos como: Polimorfismo, Herança, Encapsulamento e Classes, princípios SOLID, que visam garantir boa prática de programação e princípios de projeto em geral.

O resultado esperado é o funcionamento correto do Jogo Burro que resolva um dos problemas de jogos de baralho: o jogador jogar sozinho, ou melhor, contra o computador, na ausência de amigos, garantindo alguns momentos de distração e diversão.

b. Regras do Jogo burro



Como mencionado no tópico acima o Jogo Burro não possui muitas regras, sendo de fácil aprendizado. Entre as regras conhecidas estão:

1. Podem jogar de 2 a 4 jogadores.
2. O baralho deve ser embaralhado e cada jogador recebe 5 cartas, sendo que a distribuição deve ocorrer de modo que cada jogador obtenha apenas uma carta por vez.
3. As cartas restantes devem ser postas no meio da mesa e o jogo iniciado com o jogador imediatamente à esquerda de quem distribuiu as cartas. No entanto, na implementação o jogo começa com o primeiro usuário cadastrado.

4. O jogador inicial deve escolher uma carta qualquer e jogar em uma pilha de descarte.
5. Os demais jogadores devem jogar qualquer carta no mesmo naipe. Caso não tenha em mãos, os jogadores devem comprar cartas do baralho até sair outra do mesmo naipe.
6. Se não houver baralho e o jogador não tiver uma carta do mesmo naipe em mãos, o mesmo passa a vez para o próximo jogador.
7. Ganha direito de iniciar a próxima rodada quem jogar a carta mais alta, sendo que a ordem é posta de acordo com os valores das cartas em ordem decrescente.
8. O jogo termina quando um jogador fica sem cartas.

As regras apresentadas acima podem variar de acordo com a região ou cultura de onde o Burro é jogado. Para a definição de tais regras foram feitas entrevistas entre jogadores e pesquisa na internet.

2. Implementação

Para a implementação do jogo, foi utilizada a linguagem Oracle Java SDK8 construído na NetBeans IDE 8.1 em sua versão para Linux Ubuntu, sistema operacional onde o código foi compilado e testado durante sua construção.

Antes dos detalhes específicos da implementação serem apresentados, é importante visualizar dois diagramas criados: um estrutural e outro comportamental. Tais diagramas permitem compreender a estrutura e comportamento do sistema implementado.

Para isso, foi escolhido um diagrama estrutural e outro comportamental, sendo eles o diagrama de classes e o de atividades.

a. Diagrama de classes

De modo a compreender as classes criadas e o relacionamento entre as mesmas foi construído um Diagrama de Classes, que é um

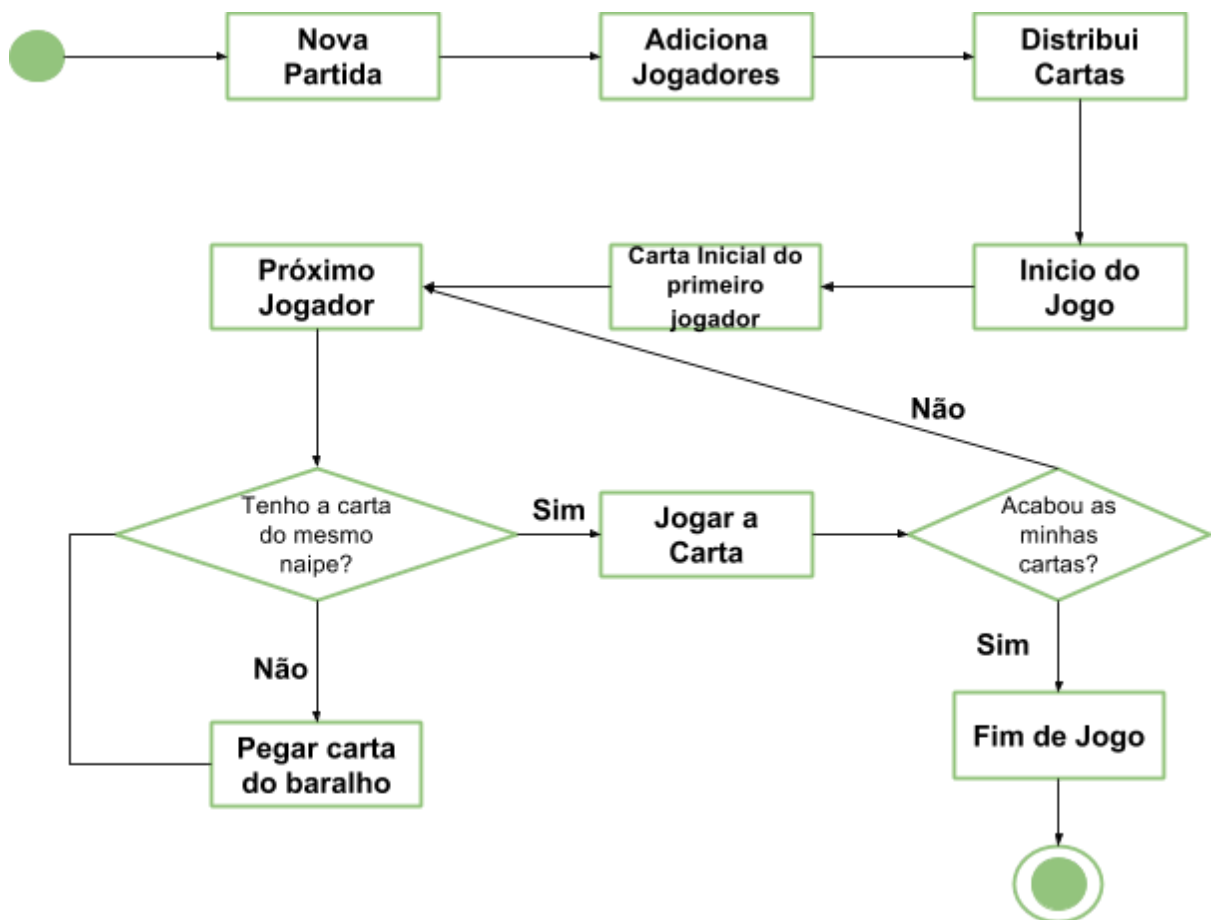
modelo UML com estrutura estática usado para modelar, principalmente, na modelagem de sistemas com o paradigma Orientado a Objetos.

Ver: Anexo 1.

b. Diagrama de atividades

O diagrama de atividades é um modelo UML comportamental que ilustra a dinâmica do sistema através do fluxo de controle de uma atividade a outra, modelando o fluxo de trabalho ou processos de negócio e funcionamento interno

Tal diagrama foi utilizado para compreender melhor a dinâmica do jogo Burro.



Além dos diagramas, os métodos utilizados em cada classe serão melhor exemplificados a seguir:

Pacote: Baralho
<p>Carta.java = Cria uma carta que possui um valor e um naipe, possui métodos set e get um construtor para inserção de tais dados.</p> <p>Naipe.java = Classe enum que possui os quatro naipes do baralho.</p> <p>Valor.java = Classe enum que cria os valores das cartas, isto é, associa o nome com o valor. Além disso, possui um método que retorna o valor numérico da carta.</p> <p>JuizBaralho.java = Classe que possui vários métodos de modo a criar o baralho e verificar algumas propriedades do mesmo.</p> <p><i>JuizBaralho()</i>: Construtor público que cria 52 cartas do baralho, cada valor com a combinação de quatro cartas diferentes.</p> <p><i>VerificaBaralho()</i>: Método booleano que verifica se o baralho está vazio.</p> <p><i>RetornaCarta()</i>: Método que retorna a primeira carta do monte do baralho.</p> <p><i>RemoveCarta()</i>: Método que remove alguma carta do baralho.</p> <p><i>EmbaralhaCartas()</i>: Método que embaralha as cartas criadas.</p> <p><i>ImprimirMonteBaralho()</i>: Método que imprime o monte de baralho.</p>

Pacote: Usuários
<p>Competidor.java = Classe abstrata que possui as capacidades de um competidor, isto é, o que o competidor consegue fazer com as cartas na mão. Tal classe possui um id (do competidor) e uma carta de mão e métodos set e get para manipular tais variáveis.</p> <p>Outros métodos da classe são:</p> <p><i>AdicionaCarta(Carta carta)</i> = Insere uma nova carta na mão.</p> <p><i>VerificaMao()</i> = Verifica se a mão está vazia.</p>

MostraMao() = Mostra as cartas na mão.

RemoveCarta(Carta carta) = Remove alguma carta da mão.

EscolheCarta(int indice) = Retorna a carta escolhida.

TamanhoMao() = Retorna o tamanho da mão de cartas.

VerificaNaipe(Carta cartaPilha) = Verifica se o jogador possui alguma carta de determinado naipe.

MaiorCarta() = Retorna a maior carta da mão.

EscolherCartaNaipe(Carta carta) = Escolhe uma carta da mão.

Jogador.java = Classe que cria um jogador que herda de competidor. Possui um construtor público que insere o nome e um id assim como métodos set e get.

MaoDeCartas.java = Classe possui um array de cartas e representa a mão de cartas que um jogador possui. Ele implementa todas as operações mencionadas na classe Competidor.Java.

Pacote: Opções

IniciaJogo.java = Classe possui um construtor que inicia um novo Jogo Burro, criando uma mesa.

CadastroUsuarios.Java = Classe realiza o cadastro dos usuários no sistema, considerando se o jogador deseja jogar contra o sistema ou não assim como a quantidade de integrantes.

CadastraJogadores(LinkedList<Jogador> listaJogadores) = Classe que possui finalidade de efetuar novos cadastro de jogadores, pergunta ao integrante se deseja ou não jogar contra o sistema.

CadastraUsuario(String opcao, LinkedList<Jogador> listaJogadores) = Dada a opção escolhida no método apresentado acima, cadastra os usuários, inserindo-os no vetor de Jogadores.

Pacote: Util

Util.java = Classe tem como finalidade implementar opções úteis ao decorrer do programa.

public static void limpaTela() = Método estático limpa a tela.

Pacote: Burro

OrdenaResultados/OrdenaRodada.java= Classe que utiliza a interface Comparator e utiliza polimorfismo para comparar dois valores a fim de ordenar um vetor utilizando o sort do Collections.

Rodada.java = Classe que cria um tipo Jogador e Carta, possuindo métodos set e get para definição do conteúdo de tais tipos. A finalidade é ser utilizada durante o código para armazenar dados de jogador e determinada carta jogada pelo mesmo.

NotasPartida.java = Classe que cria uma variável para armazenar o nome de um jogador e a quantidade de cartas na mão, possuindo métodos set e get para definição do conteúdo de tais tipos. A finalidade é ser utilizada durante o código para apresentar resultados finais da partida.

JuizBurro.java = Classe possui finalidade de agir como um juiz do Jogo Burro, verificando integridade da jogada (se é válida ou não), determinando a nota final, entre outros.

public JuizBurro()

Construtor que cria uma lista auxiliar e de ganhadores, a ser utilizada nos métodos.

VerificaJogadaValida (Carta carta, Stack<Carta> pilhaDeDescarte)

Verifica se a rodada é válida, ou seja, se a carta que o jogador está jogando é do mesmo naipe da carta que está na pilha de descarte.

DistribuiCartas(LinkedList<Jogador> listaJogadores, JuizBaralho baralho)

Distribui cinco cartas para cada jogador cadastrado. Ou seja, distribui uma carta para cada jogador até completar cinco cartas para cada.

EmbaralhaCartas(JuizBaralho baralho)

Método que embaralha a pilha de cartas.

ResultadoFinal(LinkedList<Jogador> listaJogadores)

Método que calcula o resultado final da partida, ou seja, quantas cartas que cada jogador ficou na mão.

ImprimeResultado(ArrayList<NotasPartida> listaGanhadores)

Método imprime o resultado gerado pelo método anterior, informando ao usuário a tabela de resultados.

VerificaValorIndice(int indice, Jogador jogador)

Verifica se o valor do índice das cartas é válido, ou seja, se o jogador digitou o índice certo ao escolher o índice da carta.

DefineOrdemJogadores(ArrayList<Rodada> listaRodada, LinkedList<Jogador> listaJogadores)

Método que define a ordem dos jogadores, ou seja, reorganiza os jogadores de acordo com o valor da carta jogada na partida.

MesaBurro.java = Casse que possui como objetivo simular uma mesa do Jogo Burro, com o baralho, carta de descarte e jogada dos jogadores, possuindo métodos que permitem tais operações.

MesaBurro(LinkedList<Jogador> listaJogadores)

Construtor público que cria uma nova mesa, possuindo baralho, juiz, uma pilha de descarte, rodada e, por fim, chama o método que inicia o jogo.

CartaPilhaDescarte()

Retorna a primeira carta da pilha de descarte, ou seja, a última carta jogada na pilha, correspondente a jogada anterior.

CompraCartas(Jogador jogador)

Permite que qualquer jogador compre uma nova carta no monte de baralho, realizando procedimentos necessários: remove carta do baralho, insere na mão do jogador, etc...

JogadaSystem(Jogador system, LinkedList<Jogador> listaJogadores)

Representa a jogada do sistema por completo.

JogaCartaOperacoes(Carta cartaEsc, Jogador jogador)

Método que realiza operações quando o jogador joga alguma carta. Similar ao CompraCartas(), o método adiciona a carta na pilha de descarte e remove da mão do jogador.

NovaJogada(Jogador jogador)

Representa a jogada do jogador humano por completo, perguntando o usuário qual procedimento ele deseja seguir, ou seja, se quer comprar carta (chama outro método), se quer jogar carta, chama JogadaReal, entre outros.

JogadaReal(Jogador jogador)

Método responsável por representar a jogada de um jogador humano quando ele deseja jogar uma carta na pilha de descarte. O mesmo é responsável por verificar a integridade da jogada (chama o JuizBurro para verificar) e por atualizar a mão de carta do jogador, por exemplo.

IniciaJogo(LinkedList<Jogador> listaJogadores)

Método que realiza o jogo, de acordo com o diagrama de atividades, chamando os respectivos métodos: *NovaJogada* para representar a jogada do jogador humano ou *JogadaSystem* para representar a jogada do sistema. As jogadas ocorrem até que um jogador fique com a mão vazia.

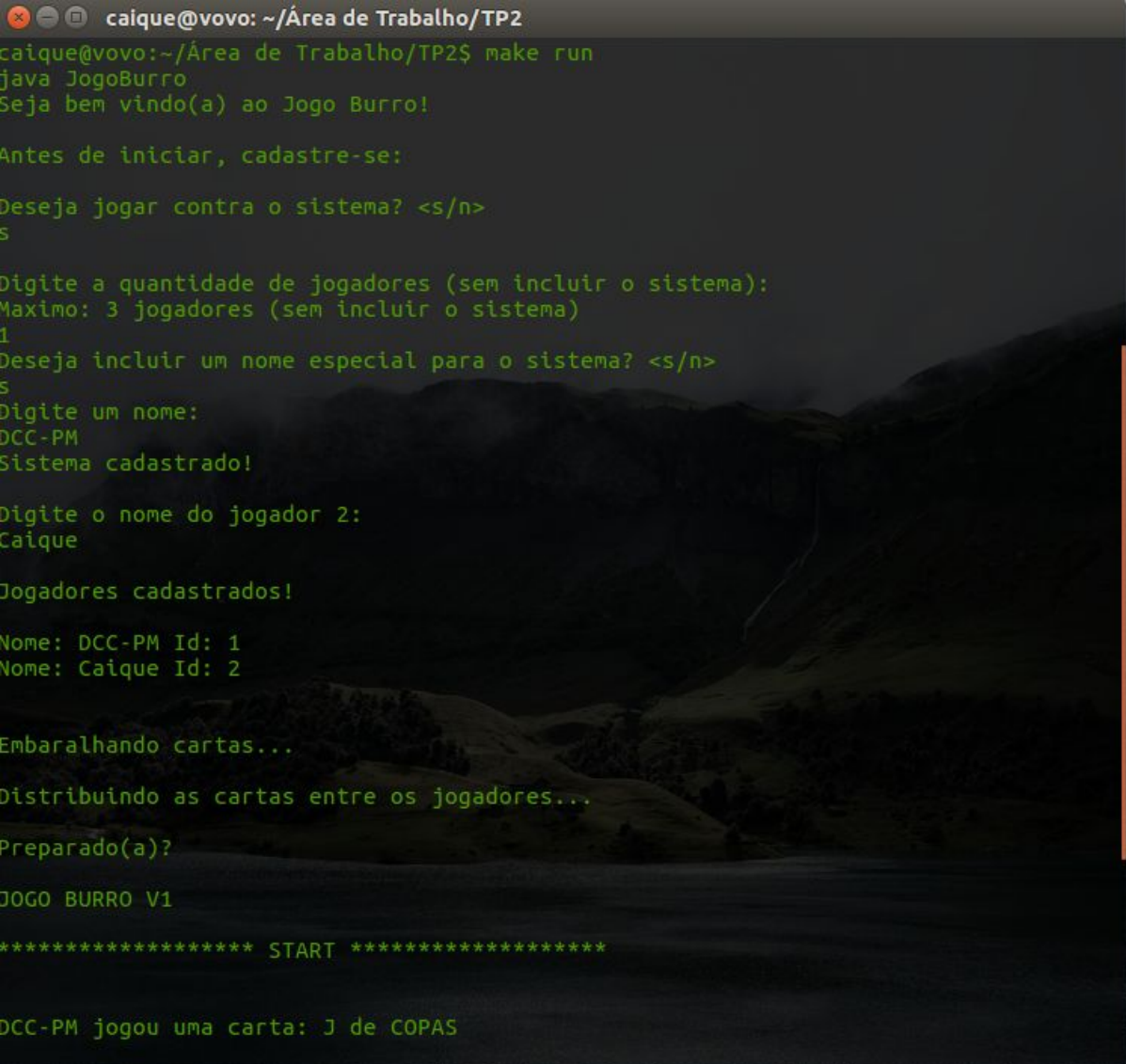
3. Testes

Foram realizados alguns testes, incluindo a participação de usuários para verificar o funcionamento do sistema. Durante tais testes não foram encontrados erros.

A aplicação foi executada em um computador Intel Core 2 Duo com 3GB de memória RAM, arquitetura de 32 bits e 150GB de memória interna, sem apresentar lentidão e travamento, mesmo porque se trata de uma aplicação simples.

Para executar o código basta digitar make e make run no diretório do trabalho em um computador Linux ou MacOSX.

Segue abaixo alguns prints de tela da aplicação em funcionamento.

A screenshot of a terminal window with a dark background and a landscape image. The window title is 'caique@vovo: ~/Área de Trabalho/TP2'. The terminal shows the execution of 'make run' followed by 'java JogoBurro'. The program prompts the user for registration, asking if they want to play against the system, the number of players (max 3), and if they want a special name for the system. The user enters 'DCC-PM' as a name. The program then asks for the name of player 2, which is 'Caique'. It lists the registered players: 'DCC-PM Id: 1' and 'Caique Id: 2'. The program then shuffles cards and distributes them. It asks 'Preparado(a)?' and then displays 'JOGO BURRO V1'. A separator line of asterisks with 'START' in the middle is shown. Finally, it shows 'DCC-PM jogou uma carta: J de COPAS'.

```
caique@vovo: ~/Área de Trabalho/TP2
caique@vovo:~/Área de Trabalho/TP2$ make run
java JogoBurro
Seja bem vindo(a) ao Jogo Burro!

Antes de iniciar, cadastre-se:

Deseja jogar contra o sistema? <s/n>
s

Digite a quantidade de jogadores (sem incluir o sistema):
Maximo: 3 jogadores (sem incluir o sistema)
1
Deseja incluir um nome especial para o sistema? <s/n>
s
Digite um nome:
DCC-PM
Sistema cadastrado!

Digite o nome do jogador 2:
Caique

Jogadores cadastrados!

Nome: DCC-PM Id: 1
Nome: Caique Id: 2

Embaralhando cartas...
Distribuindo as cartas entre os jogadores...

Preparado(a)?

JOGO BURRO V1

***** START *****

DCC-PM jogou uma carta: J de COPAS
```

```
caique@vovo: ~/Área de Trabalho/TP2

Caique, voce possui 5 cartas:

Carta 1 ==> DEZ de PAUS
Carta 2 ==> A de COPAS
Carta 3 ==> K de OURO
Carta 4 ==> DOIS de ESPADAS
Carta 5 ==> OITO de ESPADAS

O que voce deseja fazer?
Comprar uma carta 'c' ou jogar uma da mao 'j'?
j

Digite o indice da carta que deseja jogar:
1

Jogada invalida! O naipe nao corresponde ao naipe da ultima carta jogada. Tente novamente
2

*****

DCC-PM ganhou a rodada.

DCC-PM jogou uma carta: J de ESPADAS

Carta da pilha de descarte:
J de ESPADAS

Vez de: Caique

Caique, voce possui 4 cartas:

Carta 1 ==> DEZ de PAUS
Carta 2 ==> K de OURO
Carta 3 ==> DOIS de ESPADAS
Carta 4 ==> OITO de ESPADAS

O que voce deseja fazer?
```

4. Conclusão

Com a finalização do trabalho foi possível alcançar diferentes objetivos propostos: aplicação de conceitos do paradigma de Orientação a Objetos como herança, encapsulamento, classes além da prática de princípios SOLID. Ou seja, preocupamos, na medida do possível, cumprir boas práticas de programação de acordo com o que foi aprendido em sala. A aplicação de princípios de projeto também foi fundamental para que fosse possível separar em pacotes as classes que possuem alta coesão e relacionamento entre si, o que melhora a manutenção e entendimento do código, por exemplo.

Assim, preocupamos em separar pacotes de modo que se trocássemos o pacote Burro por um Truco, por exemplo, não seja necessário tantas alterações nos outros pacotes.

Diferentemente do trabalho anterior, dessa vez foi necessário construir um diagrama de classes e atividades para melhor compreender o que seria feito com o código. Ou seja, antes de implementar, tais diagramas foram analisados, modificados até chegar em um ponto onde a implementação fosse mais clara e que eliminasse possíveis erros, exceções. Logo, vimos a importância de analisar o problema, modelar e finalmente implementar.

No entanto, infelizmente vimos no final da disciplina padrões de projeto, que melhoram ainda mais o código. Devido ao tempo, não foi possível melhorar ainda mais o código como desejávamos, aplicando ferramentas que poderiam ser ainda mais úteis.

A principal dificuldade encontrada foi a modelagem, ou seja, a melhor maneira de apresentar os dados. No entanto, o funcionamento da aplicação foi de 100% nos casos de testes realizamos, assumimos que o mesmo funciona, cumprindo o que foi solicitado.

Outra dificuldade encontrada foi no tratamento de exceções do sistema, tratando todos os erros e possíveis caminhos feito pelo usuário.

No entanto, a principal contribuição do trabalho foi a modelagem e aplicação da orientação a objetos e reconhecermos que podemos melhorar nos outros trabalhos, preocupando cada vez mais com os princípios de qualidade do software.

5. Bibliografia

- API: <http://docs.oracle.com/javase/8/docs/api/>
- Regras do burro:
https://web.fe.up.pt/~arocha/AED1/0405/trabalhos/trabalho2_4.html
- Manipulação de pilhas:
<https://www.caelum.com.br/apostila-java-estrutura-dados/pilhas/#6-7-a-pi-do-java>
- Classe enum: <http://www.devmedia.com.br/tipos-enum-no-java/25729>

6. Anexos

a. Anexo 1

